Learning Deep Representations : Toward a better new understanding of the deep learning paradigm

Ludovic Arnold

école doctorale d'Informatique de l'Université Paris Sud (ED427)
Laboratoire des Sciences et Mécaniques de l'Ingénieur (LIMSI)
Équipe INRIA TAO

THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS-SUD

SPÉCIALITÉ : INFORMATIQUE

soutenue le 25/06/2013 par

LUDOVIC ARNOLD

LEARNING DEEP REPRESENTATIONS

Toward a better understanding of the deep learning paradigm

Co-directeurs de thèse:

Mme. Hélène PAUGAM-MOISY Professeur (Université Lyon 2), LRI-TAO
M. Philippe TARROUX Professeur (ENS), LIMSI

Composition du jury:

M.	Yoshua BENGIO Full Professor (Université de Montréal), LISA	Rapporteur
M.	Stéphane CANU Professeur (INSA Rouen), LITIS	Rapporteur
M.	Thierry ARTIÈRES Professeur (Université Paris 6), LIP6	Examinateur
Mme.	Michèle SEBAG Directeur de Recherche CNRS, LRI-TAO	Examinateur

LEARNING DEEP REPRESENTATIONS

LUDOVIC ARNOLD

Toward a better understanding of the deep learning paradigm

June 2013

Once the machine thinking method has started, it would not take long to outstrip our feeble powers. ... At some stage therefore we should have to expect the machines to take control.

Alan Turing, 1951

The point about this approach is that it scales beautifully. Basically you just need to keep making it bigger and faster, and it will get better. There's no looking back now.

Geoffrey Hinton, 2012

ABSTRACT

Since 2006, deep learning algorithms which rely on deep architectures with several layers of increasingly complex representations have been able to outperform state-of-the-art methods in several settings. Deep architectures can be very efficient in terms of the number of parameters required to represent complex operations which makes them very appealing to achieve good generalization with small amounts of data. Although training deep architectures has traditionally been considered a difficult problem, a successful approach has been to employ an unsupervised layer-wise pre-training step to initialize deep supervised models. First, unsupervised learning has many benefits w.r.t. generalization because it only relies on unlabeled data which is easily found. Second, the possibility to learn representations layer by layer instead of all layers at once further improves generalization and reduces computational time. However, deep learning is a very recent approach and still poses a lot of theoretical and practical questions concerning the consistency of layer-wise learning with many layers and difficulties such as evaluating performance, performing model selection and optimizing layers.

In this thesis we first discuss the limitations of the current variational justification for layer-wise learning which does not generalize well to many layers. We ask if a layer-wise method can ever be truly consistent, i.e. capable of finding an optimal deep model by training one layer at a time without knowledge of the upper layers. We find that layer-wise learning can in fact be consistent and can lead to optimal deep generative models. To that end, we introduce the Best Latent Marginal (BLM) upper bound, a new criterion which represents the maximum log-likelihood of a deep generative model where the upper layers are unspecified. We prove that maximizing this criterion for each layer leads to an optimal deep architecture, provided the rest of the training goes well. Although this criterion cannot be computed exactly, we show that it can be maximized effectively by auto-encoders when the encoder part of the model is allowed to be as rich as possible. This gives a new justification for stacking models trained to reproduce their input and yields better results than the state-of-the-art variational approach. Additionally, we give a tractable approximation of the BLM upper-bound and show that it can accurately estimate the final log-likelihood of models. Taking advantage of these theoretical advances, we propose a new method for performing layer-wise model selection in deep architectures, and a new criterion to assess whether adding more layers is justified. As for the difficulty of training layers, we also study the impact of metrics and parametrization on the commonly used gradient descent procedure for log-likelihood maximization. We show that gradient descent is implicitly linked to the metric of the underlying space and that the Euclidean metric may often be an unsuitable choice as it introduces a dependence on parametrization and can lead to a breach of symmetry. To alleviate this issue, we study the benefits of the natural gradient and show that it can restore symmetry, regrettably at a high computational cost. We thus propose that a centered parametrization may alleviate the problem with almost no computational overhead.

RÉSUMÉ

Depuis 2006, les algorithmes d'apprentissage profond qui s'appuient sur des modèles comprenant plusieurs couches de représentations croissantes en complexité ont pu surpasser l'état de l'art dans plusieurs domaines. Les modèles profonds peuvent être très efficaces en termes du nombre de paramètres nécessaires pour représenter des opérations complexes, ce qui les rend très intéressants pour obtenir une bonne généralisation avec de faibles quantités de données. Bien que l'entraînement des modèles profonds ait été traditionnellement considéré comme un problème difficile, une approche réussie a été d'utiliser une étape de pré-entraînement couche par couche, non supervisée, pour initialiser des modèles profonds supervisés. Tout d'abord, l'apprentissage non-supervisé présente de nombreux avantages par rapport à la généralisation car il repose uniquement sur des données non étiquetées qu'il est facile de trouver. Deuxièmement, la possibilité d'apprendre des représentations couche par couche, au lieu de toutes les couches à la fois, améliore encore la généralisation et réduit les temps de calcul. Cependant, l'apprentissage profond est une approche très récente et pose encore beaucoup de questions théoriques et pratiques relatives à la consistance de l'apprentissage couche par couche, avec de nombreuses couches, et à la difficulté d'évaluer la performance, de sélectionner les modèles et d'optimiser la performance des couches.

Dans cette thèse, nous examinons d'abord les limites de la justification variationnelle actuelle pour l'apprentissage couche par couche qui ne se généralise pas bien à de nombreuses couches et demandons si une méthode couche par couche peut jamais être vraiment consistante, c'est à dire capable de trouver un modèle profond optimal en entraînant un modèle de bas en haut, sans connaissance des couches supérieures. Nous constatons que l'apprentissage couche par couche peut en effet être consistant et peut conduire à des modèles génératifs profonds optimaux. Pour ce faire, nous introduisons la borne supérieure de la meilleure probabilité marginale latente (*BLM upper bound*), un nouveau critère qui représente la log-vraisemblance maximale d'un modèle génératif profond quand les couches supérieures ne sont pas connues. Nous prouvons que la maximisation de ce critère pour chaque couche conduit à une architecture profonde optimale, à condition que le reste de l'entraînement se passe bien. Bien que ce critère ne puisse pas être calculé de manière exacte, nous montrons qu'il peut être maximisé efficacement par des auto-encodeurs quand l'encodeur du modèle est autorisé à être aussi riche que possible. Cela donne une nouvelle justification pour empiler les modèles entraînés pour reproduire leur entrée et donne de meilleurs résultats que l'approche variationnelle, qui est la meilleure méthode actuellement connue. En outre, nous donnons une approximation calculable de la *BLM upper* *bound* et montrons qu'elle peut être utilisée pour estimer avec précision la log-vraisemblance finale des modèles. Tirant parti de ces avancées théoriques, nous proposons une nouvelle méthode pour la sélection de modèles couche par couche pour les modèles profonds, et un nouveau critère pour déterminer si l'ajout de couches est justifié. Quant à la difficulté d'entraîner chaque couche, nous étudions aussi l'impact des métriques et de la paramétrisation sur la procédure de descente de gradient couramment utilisée pour la maximisation de la vraisemblance. Nous montrons que la descente de gradient est implicitement liée à la métrique de l'espace sous-jacent et que la métrique Euclidienne peut souvent être un choix inadapté car elle introduit une dépendance sur la paramétrisation et peut entraîner une violation de la symétrie. Pour pallier ce problème, nous étudions les avantages du gradient naturel et montrons qu'il peut être utilisé pour restaurer la symétrie, mais avec un coût de calcul élevé. Nous proposons donc qu'une paramétrisation centrée peut également rétablir la symétrie, mais avec une très faible surcharge computationnelle.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Hélène Paugam-Moisy for her unwaivering support in difficult times. She was always there when I needed her, and was truly dedicated to supporting me. I couldn't have found a better thesis advisor. I thank Yann Ollivier for his much needed supervision in mathematics and his friendly ear in general. I also thank Philippe Tarroux whose vision of AI pretty much guided my decision to start this Ph.D, Michèle Sebag co-director of the TAO Team for her insights into the field of Machine Learning, and Christian Gagné for the opportunity he gave me to visit the LSVN in Québec.

I would also like to thank all the Ph.D. students with whom I was able to have enlightening discussions such as Pierre Delarboulas, Jean Marc Montanier, Pierre Allegraud, Sylvain Chevallier and Sébastien Rebecchi.

CONTENTS

ACRONYMS

AI	Artificial Intelligence
BLM	Best Latent Marginal
CD	Contrastive-Divergence
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
EDA	Estimation of Distribution Algorithm
EM	Expectation Maximization
iid	independent and identically distributed
KL	Kullblack-Leibler
ML	Machine Learning
RBM	Restricted Boltzmann Machine
SVM	Support Vector Machine
MCMC	Monte Carlo Markov Chain
MSE	Mean Square Error
PBIL	Population Based Incremental Learning
RBF	Radial Basis Function

NOTATIONS

ALGEBRA

$\mathbf{x}$	a real vector.
x_i	i^{th} component of the vector $\mathbf{x}$.
$\mathbf{x}_i$	the i^{th} vector in a set of vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$.
$\mathbf{A}$	a matrix.
$\mathbf{A}^T$	the transpose of $\mathbf{A}$.
a_{ij}	the entry of the matrix $\mathbf{A}$ at row i, column j.
$\\|\mathbf{x}\\|_p$	L^p norm $\\|\mathbf{x}\\|_p = \left(\sum_i \|x_i\|^p\right)^{\frac{1}{p}}$.
$\\|\mathbf{x}\\|$	L^2 norm (a.k.a. Euclidean norm).

OPTIMIZATION

$\arg\min_{x \in \operatorname{dom} f} f(x)$	x^* such that $f(x^*) = \min_{x \in \operatorname{dom} f} f(x)$.
$\arg\max_{x \in \operatorname{dom} f} f(x)$	x^* such that $f(x^*) = \max_{x \in \operatorname{dom} f} f(x)$.

DIFFERENTIATION

$\frac{\partial f(x,y,z)}{\partial x}$	partial derivative of f with respect to x.
$\frac{\partial^2 f(x,y,z)}{\partial x \partial y}$	second derivative with respect to x and y.
$\nabla f(\mathbf{x})$	gradient of f at $\mathbf{x}$ (first derivative). $\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_D}\right)$.
$\nabla^2 f(\mathbf{x})$	Hessian of f at $\mathbf{x}$ (second derivative) given by the symmetric matrix $\nabla^2 f(\mathbf{x})_{i,j} = \frac{\partial f(\mathbf{x})}{\partial x_i \partial x_j}$.

PROBABILITIES

$\mathbf{x} \sim P$	$\mathbf{x}$ is distributed according to the distribution P. Equivalently: $\mathbf{x}$ is a sample from P.
$\mathcal{U}(\mathcal{S})$	uniform law over the set $\mathcal{S}$.
$\mathcal{B}(p)$	Bernoulli probability of parameter p.
$\mathcal{N}(\mu, \sigma^2)$	normal law of mean μ and variance σ^2.
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	multidimensional normal law of mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.
$\mathcal{B}eta(a, b)$	Beta distribution of parameters a, b.

LIST OF FIGURES

INTRODUCTION

The goal of Artificial Intelligence (AI) is to have a system perform a task that requires intelligence.

Such tasks can be found in a variety of domains:

- *Language:* Translation, Summarizing, Topic extraction
- *Vision:* Classification, Segmentation, Image retrieval
- *Games:* Chess, Go, Strategy games
- *...and others:* Regression, Decision, Risk analysis

For some tasks such as Chess, the performance of the computer now surpasses that of the human being, while for other tasks, it has yet to even approach it. In particular, many problems solved effortlessly by human beings in the fields of vision and language turn out to be very difficult to solve using algorithms. While the term AI defines the problem, it does not refer to any specific method for solving it. A wide variety of approaches have emerged over the years, but all have failed to create a general purpose AI: an AI with the same capacity for reasoning as the human mind.

Consider now an AI system as described above. Such a system will have to perform tasks, take decisions and make choices. All those actions add up to constitute what we could call the *behavior* of the system. Machine Learning (ML) is based on the important realization that intelligent behavior is too complex to be simply "programmed". Instead, the system will *learn its behavior from data.* A system endowed with this capacity to learn some kind of intelligent behavior is simply called a *model*, and the process by which we *train* a model given data is called a *machine learning algorithm.*

In practice, a model is defined by some equation or algorithm which, before learning, has a set of undetermined variables called *parameters.* During the learning procedure, the data is used to choose values for the parameters which will maximize the capacity of the model to perform the objective task. This "capacity to perform" is measured by what is called a *fitness function* or *objective function.* To do this, we turn to *optimization* which is a branch of mathematics consisting in the study of how to choose parameters to maximize an objective function. An ML problem can then be understood as an optimization one, where the objective is to maximize some performance measure w.r.t a final task. Clearly, optimization is useful to machine learning, but as we will see, learning can also be useful to optimization. In essence, learning algorithms can be used to learn the landscape of an objective function, facilitating the search for suitable parameters.

As for the data from which the system will learn, it should be informative of the objective task. For example, in the field of *supervised learning*, the goal is to learn a model given examples of what the model should do in several situations. The data then consists in a set of examples (stimulus $\mathbf{x} \rightarrow$ response $\mathbf{y}$) which describe how the system should ideally respond to several input stimuli. Supervised learning has a large number of applications such as:

- Prediction: given an observation $\mathbf{x}$ at time t, what is the probable observation $\mathbf{y}$ at time $t+1$.
- Games: given the state of a game board $\mathbf{x}$, what is the next best move $\mathbf{y}$.
- Search engines: given a search query $\mathbf{x}$, what is the most relevant result $\mathbf{y}$.
- Pattern recognition: e.g. in the case of Handwritten characters, given the pixels of a scanned zip-code $\mathbf{x}$, what is the zip-code $\mathbf{y}$.

A problem quickly arises when the link between $\mathbf{x}$ and $\mathbf{y}$ is not straightforward. This leads practitioners to use a *preprocessing* step in which an expert has to find a set of *features* $\mathbf{f}(\mathbf{x})$ such that learning the relation between $\mathbf{f}(\mathbf{x})$ and $\mathbf{y}$ becomes simpler. However, hand-crafted features are costly because they require expert knowledge, often acquired after years of experimentation.

In *representation learning*, a learning algorithm is used to find interesting features from the data. Learning features, instead of using a preprocessing step has many benefits as it makes the whole approach less dependent on human input, and thus more general. Even if this can seem challenging, learning useful representations can be accomplished in practice with *unsupervised learning* where learning is done on a dataset of training examples $\mathbf{x}$ without the corresponding answers $\mathbf{y}$. Unsupervised learning includes tasks such as:

- *clustering*, where the objective is to group similar observations.
- *compression* or *dimensionality reduction*, where the objective is to learn a representation smaller than the input.
- *density estimation*, where the objective is to find a probability distribution which is likely to have generated the dataset.

One aspect of particular relevance to this thesis and to unsupervised learning is the possibility to consider several *layers* of processing, i.e. a *deep architecture*. Although learning deep architectures raises serious computational issues, the principle has been applied successfully in recent years using a layer wise approach, essentially trying to learn features one layer at a time instead of trying to learn them all at once. The approach can be summarized as follows: first train a set of features $\mathbf{f}_1(\mathbf{x})$ to better represent the input $\mathbf{x}$, then consider learning higher level features $\mathbf{f}_2(\mathbf{f}_1(\mathbf{x}))$ using $\mathbf{f}_1(\mathbf{x})$ as a new representation of the dataset. In this setting, the features $\mathbf{f}_1$ are trained to explain the data $\mathbf{x}$, and $\mathbf{f}_{k+1}$ is trained to explain the data as represented by the features $\mathbf{f}_k$.

With this in mind, LEARNING DEEP REPRESENTATIONS refers to the problem of learning multiple layers of interesting features for a given dataset.

As we have already seen, optimization is a very important topic for Machine Learning as it serves to choose parameter values which maximize an objective function. Several approaches to optimization are given in Figure 0.1. As for machine learning, the organization of several possible tasks is represented in Figure 0.2.

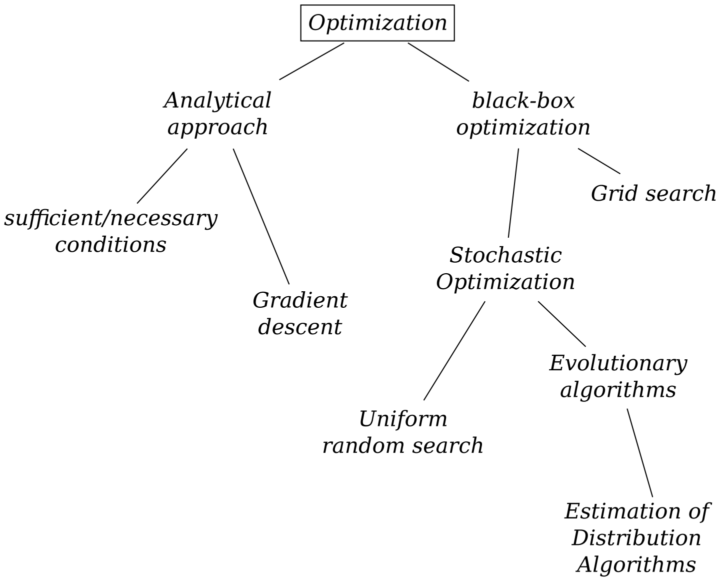


Figure 0.1: Approaches to optimization (non-exhaustive).

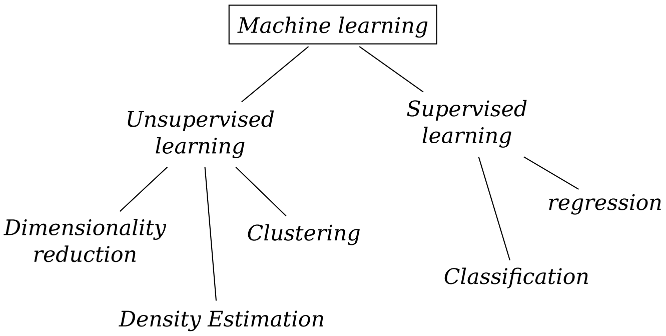


Figure 0.2: Machine Learning tasks (non-exhaustive).

This thesis is composed of three parts:

PART I: OPTIMIZATION AND MACHINE LEARNING starts with a definition of what constitutes an optimization problem and describes several approaches to find a solution. It then describes how ML problems can be posed as optimization problems. Finally, we introduce the probabilistic perspective on ML which has gained a lot of attention in recent years.

PART II: DEEP LEARNING begins with a presentation of neural networks which are the most successful approach to deep learning yet. It then describes how neural networks can be used in the context of learning deep representations. Finally, the part ends with a review of recent advances in deep learning, introducing the questions which motivate the author's contributions

PART III: CONTRIBUTIONS contains the three main publications of the author, replaced in their context and commented to make clear their contribution in the light of the questions which arose in Part II.

Finally, the thesis concludes with a summary of the author's contributions, a discussion of their impact on the current understanding of the deep learning paradigm, and with new perspectives of research which arise from the accomplished work.

Part I

OPTIMIZATION AND MACHINE LEARNING

1

OPTIMIZATION

A solution to a learning problem is usually found by an optimization procedure, i.e. maximizing some performance measure (e.g. classification accuracy) or minimizing a *loss function* (e.g. minimize the number of misclassified examples) over a dataset. However, optimization is a much larger problem, namely that of finding the parameters of a function which maximize the associated value. Optimization applies to a large variety of problems such as designing efficient engines or minimizing costs, provided there exists a function which can measure the fitness of candidate solutions.

In this chapter, we start by a definition of what constitutes an optimization problem and discuss the important issue of local minima. We discuss the special case of convex functions for which every local minimizer is a global one, and that of continuously differentiable functions for which optimum values are among those where the derivative of the objective function is zero. Then, we present the gradient descent algorithm which is a widely used method in ML. Finally, we introduce the black-box optimization setting where inner workings of the objective function are assumed unknown and present Estimation of Distribution Algorithms (EDAs) which are especially suited to this context.

1.1 Problem statement

When faced with an optimization problem, the goal can be:

- to find optimal parameters $\mathbf{x}^*$ for which $f(\mathbf{x}^*)$ has the least possible value –in which case we refer to it as a *minimization problem*–, or

- to find optimal parameters $\mathbf{x}^*$ for which $f(\mathbf{x}^*)$ has the greatest possible value –in which case we refer to it as a *maximization problem.*

The space of values of $\mathbf{x}$ considered as possible solutions is called the domain of f and is noted $\operatorname{dom} f$. We now give a formal definition.

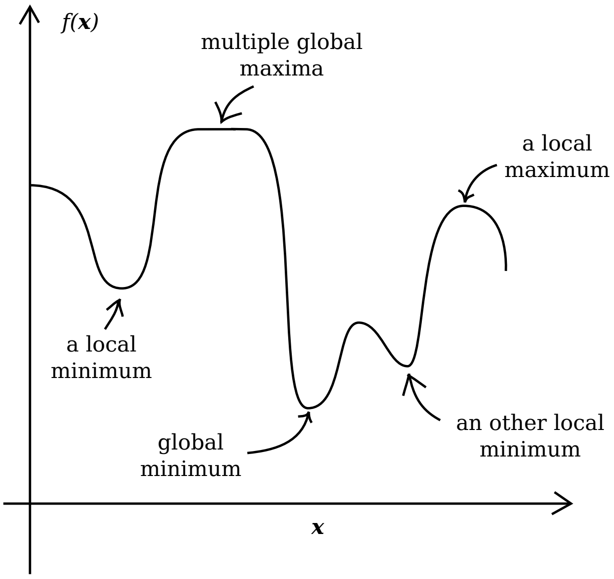


Figure 1.1: Global and local optima of a function f.

Definition 1.1. *(global optimization problem).* Let us consider the problem of minimizing a function f over it's domain $\operatorname{dom} f$ which we note

$$\min_{\mathbf{x} \in \operatorname{dom} f} f(\mathbf{x}).$$

Let $\mathbf{x}^*$ be a solution to the above minimization problem which we note

$$\mathbf{x}^* = \operatorname*{arg\,min}_{\mathbf{x} \in \operatorname{dom} f} f(\mathbf{x}),$$

then $\mathbf{x}^*$ is called a global minimum and satisfies

$$\forall \mathbf{x} \in \operatorname{dom} f, f(\mathbf{x}^*) \leq f(\mathbf{x}).$$

A global maximization problem and a global maximum are defined analogously using the notations max and arg max. The terms optimum and global optimum can be used indiscriminately to refer to maxima or minima.

It is often very hard to find a global optimum because it is defined as being better than all possible values of $\mathbf{x}$ in the available domain. Therefore it is sometimes necessary to consider only the simpler problem of *local optimization.*

Definition 1.2. *(local optimum).* $\mathbf{x}^*$ is a local minimum of f iff

$$\exists \boldsymbol{\epsilon} > 0, \forall \mathbf{x} \in \operatorname{dom} f, \|\mathbf{x} - \mathbf{x}^*\| < \boldsymbol{\epsilon} \Rightarrow f(\mathbf{x}^*) \leq f(\mathbf{x}).$$

The definition of a local maximum is analogous with $f(\mathbf{x}^*) \geq f(\mathbf{x})$.

In other words, $\mathbf{x}^*$ is a local minimum if it is possible to find a small neighborhood of $\mathbf{x}^*$ such that $\mathbf{x}^*$ is a global minimum of the restriction of f to this neighborhood. See Figure 1.1 for a visual representation of global and local optima of a simple function.

In particular, any global optimum is also a local optimum for which any choice of neighborhood is acceptable.

1.2 The curse of dimensionality

Optimization in spaces of high dimensionality can be somewhat counter intuitive. Consider for instance the unit cube in dimension n. By comparison, the volume of the cube of side 0.99 which contains 99% of the volume of the unit cube in dimension 1 only contains about 36% of it in 100 dimensions and $4.3 \times 10^{-5}\%$ in dimension 1000. Thus, almost all the volume of a 1000-dimensional hypercube is concentrated in an infinitesimal region near its boundary with almost no volume in the center.

This can be interpreted as a fundamental difference in the behavior of distances in spaces of high dimensionality. The above example can be understood as a manifestation of the fact that almost no points are close together in high dimension because they have so many ways of being dissimilar.

In the context of optimization, the unusual behavior of high dimensional spaces can become very problematic. For instance, it could seem reasonable to use a *grid search approach* (see Algorithm 1.1, i.e. testing the fitness of 100 values of x at intervals of length 0.01. However, in a 1000-dimension space, the number of function evaluations needed to form such a grid would be 100^{1000}: intractable even for trivial problems.

These issues are by no means insurmountable but can turn up in many situations. We will try to address them when they do.

1.3 Convex functions

The particular case of convex functions plays an important role w.r.t. to the problem of local minima.

Algorithm 1.1 The grid-search algorithm.

```
INPUT:      $f$, a function with $\mathrm{dom}\, f = [0,1]^D$.
            $d$, distance between observations.
OUTPUT:     $\hat{\mathbf{x}}$, approximation of a global minimum.
VARIABLES:  $\mathbf{x}_t$, candidate solution of the algorithm at time $t$.
BEGIN
$\hat{\mathbf{x}} = (0, 0, \dots, 0)$
for $u_1$ from 0 to 1 by step $d$:
        for $u_2$ from 0 to 1 by step $d$:
                $\vdots$
                for $u_D$ from 0 to 1 by step $d$:
                        $\mathbf{x}_t := (u_1, u_2, \dots, u_D)$
                        if $f(\mathbf{x}_t) < f(\hat{\mathbf{x}})$ then $\hat{\mathbf{x}} := \mathbf{x}_t$
return $\hat{\mathbf{x}}$.
END
```

Definition 1.3. (*convex function*). Let f be a function defined over some domain $\mathrm{dom}\, f$. f is said to be convex, iff

$$\forall \mathbf{a}, \mathbf{b} \in \mathrm{dom}\, f, \forall k \in [0;1], f(k\mathbf{a} + (1-k)\mathbf{b}) \leq kf(\mathbf{a}) + (1-k)f(\mathbf{b})$$

This means that any point between **a** and **b** has an image by f which is below the line segment joining $(\mathbf{a}, f(\mathbf{a}))$ and $(\mathbf{b}, f(\mathbf{b}))$ as depicted in Figure 1.2. Convex functions have the very interesting property that any local minimum is in fact a global minimum, thus simplifying the problem for practitioners.

1.4 Continuous differentiable functions

In the case of continuous differentiable functions, every local optimum satisfies the so-called 1st order necessary condition, i.e. the gradient at that point has to be 0. Note that not all points which satisfy this condition are local optima, as in the function x^3 at 0.

Definition 1.4. *(1^{st} order necessary optimality condition).* Let $\mathbf{x}^*$ be a local optimum of f, then $\nabla f(\mathbf{x}^*) = 0$.

This means that at every local optimum, the graph of f has a horizontal tangent[1]. To get a visual intuition of this fact, the reader is again referred to Figure 1.1.

Additionally, if f is continuous and twice differentiable, the second order derivative, i.e. the Hessian can be used to define both a necessary condition and a sufficient condition for local optimality.

[1] this horizontal tangent is in fact a horizontal tangent hyperplane if the dimension of $\mathrm{dom}\, f$ is larger than 1.

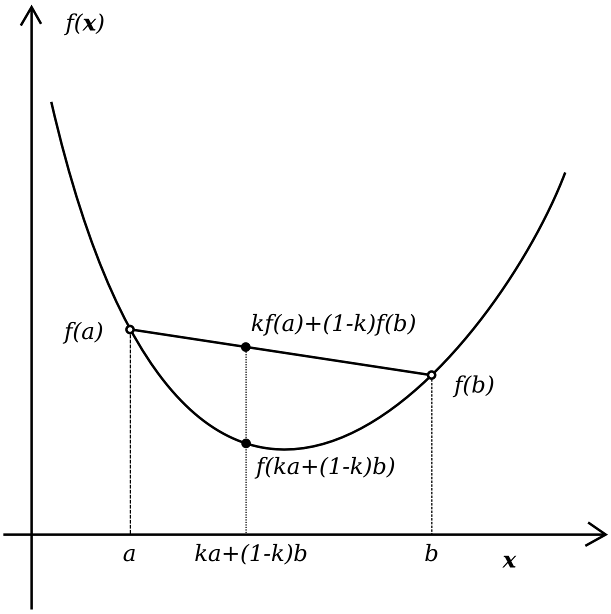


Figure 1.2: The graph of a convex function f. The values of f between two points a and b are always below the chord, i.e. the line segment between $(\mathbf{a}, f(\mathbf{a}))$ and $(\mathbf{b}, f(\mathbf{b}))$.

Definition 1.5. *(2^{nd} order necessary optimality condition).* Let $\mathbf{x}^*$ be a local minimum of f, then $\nabla f(\mathbf{x}^*) = 0$ and $\nabla^2 f(\mathbf{x}^*)$ is positive semidefinite.

Definition 1.6. *(2^{nd} order sufficient optimality condition).* Suppose that f is continuous and twice differentiable, and $\nabla f(\mathbf{x}^*) = 0$ and $\nabla^2 f(\mathbf{x}^*)$ is positive definite. Then $\mathbf{x}^*$ is a strict local minimum.

The above conditions could be used to prove for instance that a function $f : x \to x^2 + x - 1$ has a global minimum at $x^* = 1$. The principle can then be generalized to arbitrarily complicated functions in high dimensional spaces, provided the gradient and the Hessian can be found analytically.

1.5 Gradient descent

A common local optimization method is the gradient descent algorithm. The gradient $\nabla f(\mathbf{x})$ has the direction of greatest increase of the function f at $\mathbf{x}$.

$$\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|} = \lim_{\epsilon \to 0} \operatorname*{arg\,max}_{\mathbf{z}, \|\mathbf{z}\| \leq 1} f(\mathbf{x} + \epsilon \mathbf{z})$$

Algorithm 1.2 The gradient descent algorithm.

```
INPUT:       $f$, a function.
             $\delta t$, the step size.
             $K$, the number of steps.
OUTPUT:      $\hat{\mathbf{x}}$, approximation of a global minimum.
VARIABLES:   $\mathbf{x}_t$, candidate solution of the algorithm at time $t$.
BEGIN
repeat $K$ times:
        $\mathbf{x}_{t+1} := \mathbf{x}_t - \delta t \nabla f(\mathbf{x}_t)$
return last position $\hat{\mathbf{x}} := \mathbf{x}_{tmax}$.
END
```

The gradient can be computed using the partial derivatives w.r.t. each component of the input vector $\mathbf{x}$:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_D} \right)$$

In gradient descent[2] (see Algorithm 1.2), we start at some initial guess $\mathbf{x}_0$ and iteratively take small steps of size δt in the direction of $-\nabla f(\mathbf{x}_k)$. In practice it is common to stop the algorithm after a predefined number of steps or when a the objective function has not decreased for some time. In the limit of infinitesimal step size, there is a guarantee that the algorithm decreases the value of f at each step, and a guarantee that the algorithm converges to a local minimum if it doesn't encounter a saddle point at which $\nabla f(\mathbf{x}_k) = 0$. However, a bigger step size allows the algorithm to move faster in the domain of f, possibly leading to faster convergence when it does not lead to oscillations. Figure 1.3 gives an example of gradient descent trajectory towards a local minimum.

Gradient descent often behaves poorly when the objective function has narrow valleys which cause oscillations. When confronted with such functions, a possible approach is to use 2nd order information from the Hessian, e.g. using Newton's method (Nocedal and Wright, 2006) or Hessian-Free optimization (Martens, 2010; Martens and Sutskever, 2011; Sutskever et al., 2011).

Surprisingly, gradient descent does not suffer from the curse of dimensionality and could in fact be considered to benefit from many dimensions. Common issues with gradient descent have to do with the gradient getting stuck in local minima and on plateaus where the derivative is zero. However, in spaces of high dimension, these issues are much less common because every dimension increases the possibility of finding a way out.

Nonetheless, the gradient descent algorithm depends on the possibility to compute the partial derivatives at each step. This is only possible when an explicit formula is available for the objective function, which is not always the case.

[2] Gradient ascent is defined identically except for a change of sign in the update.

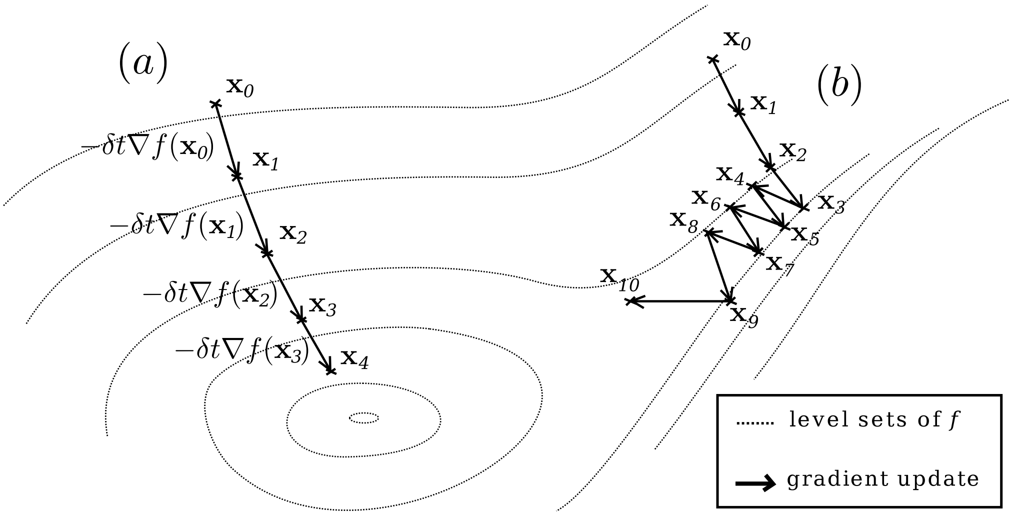


Figure 1.3: Two possible gradient descent trajectories. In (a) the objective function is well behaved which allows the gradient to move smoothly towards the optimum. In (b) the gradient starts to oscillate as it falls into a narrow valley, thus converging more slowly.

1.6 Black-box optimization and Stochastic optimization

In many cases, the objective function is not given by a specific formula but is only given in the form of an evaluation process. For example, if the objective is to find the best number of pistons k in an engine to maximize the mileage per gallon $f(k)$, f does not have an analytical formulation. In this example, the objective function presents the additional difficulty of having a discrete domain, namely the set of positive natural numbers $k \in \mathbb{N}^*$. In discrete optimization, necessary conditions, sufficient conditions and gradient based algorithms are inapplicable.

Nevertheless, f can still be evaluated for any candidate solution that might be considered. In the previous example, $f(k)$ can be evaluated by actually building an engine with k pistons and measuring the mileage per gallon empirically, clearly a costly process.

A problem such as the one we just described is known as a black-box optimization problem.

Definition 1.7. *(black-box optimization problem).* Let us consider the minimization problem

$$\min_{\mathbf{x} \in \operatorname{dom} f} f(\mathbf{x}),$$

where the only available operation on f is evaluation, namely obtaining $f(\mathbf{x})$ from any given value $\mathbf{x}$.
Then the above problem is referred to as a black-box optimization problem.

Algorithm 1.3 The uniform random search algorithm.

```
INPUT:      $f$, a function.
OUTPUT:     $\hat{\mathbf{x}}$, approximation of a global minimum.
VARIABLES:  $\mathbf{x}_t$, candidate solution of the algorithm at time $t$.
BEGIN
$\mathbf{x}_0 \sim \mathcal{U}(\operatorname{dom} f)$
$\hat{\mathbf{x}} := \mathbf{x}_0$
until satisfied:
        $\mathbf{x}_{t+1} \sim \mathcal{U}(\operatorname{dom} f)$
        if $f(\mathbf{x}_{t+1}) < f(\hat{\mathbf{x}})$ then $\hat{\mathbf{x}} := \mathbf{x}_{t+1}$
return best guess so far $\hat{\mathbf{x}}$.
END
```

The efficiency of black box optimization strategies is usually measured in terms of the number of evaluations to reach a target value or reciprocally in terms of the target value attainable for a given number of evaluations.

In this context, stochastic optimization algorithms propose to iteratively try new candidate solutions based on a probability distribution on the domain of f. Consider for instance the example of uniform random search which consists in repeatedly sampling candidate solutions in a uniform distribution over $\operatorname{dom} f$, i.e. $\mathbf{x}_t \sim \mathcal{U}(\operatorname{dom} f)$, and returning the best candidate found so far (see Algorithm 1.3).

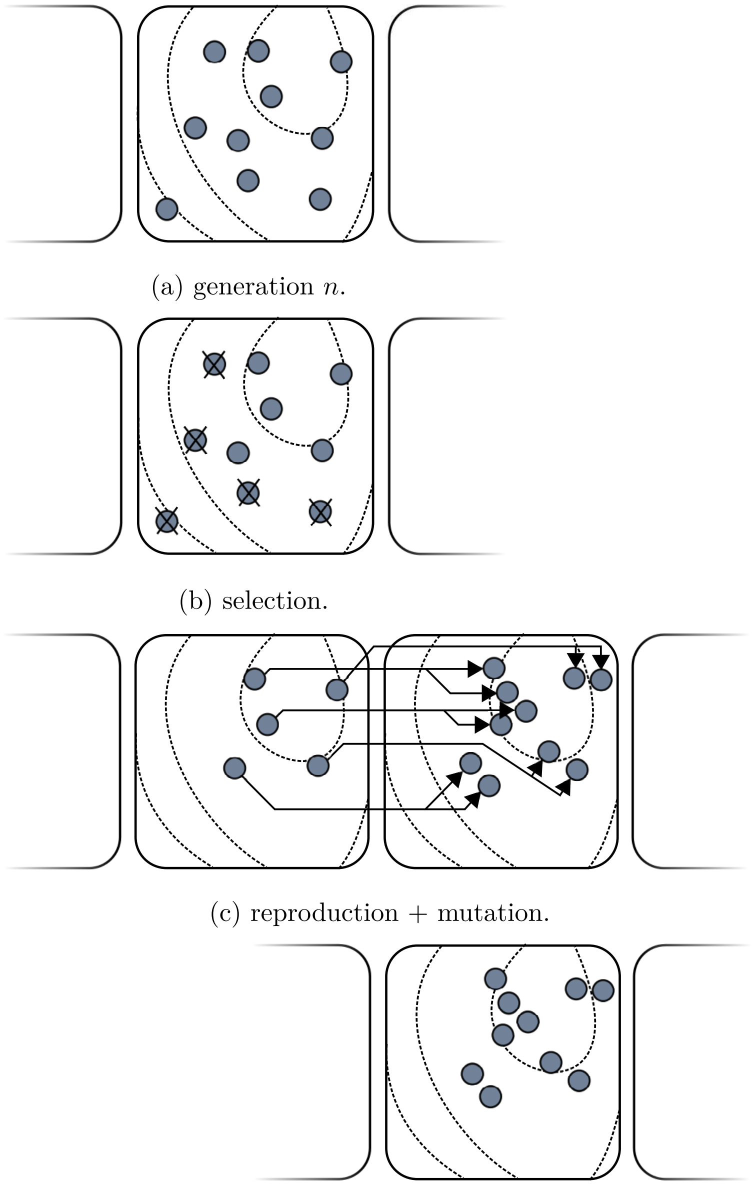
Although this algorithm does not seem very efficient[3], it has a number of interesting properties. First, it is a *global* optimization method and never gets stuck into local minima; second, there is a guarantee that each step can only improve performance. Finally, this algorithm is capable of ignoring dimensions which are not relevant to the problem and can therefore be much more efficient than a grid search approach (Bergstra and Bengio, 2012).

An interesting improvement of this method concerns the possibility of adapting the probability distribution based on past observations. In essence, the goal is then to *learn* from previous attempts, where better values might be located in the search space.

1.7 Evolutionary algorithms

Evolutionary algorithms propose to optimize an objective function by using an artificial evolution process to select relevant parameters. In this setting, parameter values are seen as organisms evolving in an environment which only allows survival and reproduction of the fittest –the fittest being the best parameter values according to the objective function. The algorithm starts with a set of proposal solutions which is seen as a *population* of *individuals*. Based on the eval-

[3] Statements about the superiority of a particular optimization method should always be considered in the light of the no free lunch theorem (Wolpert and Macready, 1997), which states that no optimization algorithm is strictly better than another on all problems.



(a) generation n.

(b) selection.

(c) reproduction + mutation.

(d) generation $n + 1$.

Figure 1.4: Visualization of an evolutionary algorithm on a 2D toy problem. The dotted lines represent level sets of the fitness function. At generation n (a), the best individuals according to the fitness function are selected (b). These individuals are then the basis of a reproduction process (c) leading to a new generation (d). The process can then continue with step (b) for the new generation. Notice that generation $n + 1$ has progressed towards better values of the fitness function.

uation of these individuals according to the function f, the best individuals are then selected for *reproduction*. Reproduction serves to generate new individuals using a *mutation operator*, and sometimes a *cross-over operator*. Finally, the new individuals replace the previous population thus creating a new *generation*. This process is summarized in Figure 1.4.

The mutation operator is meant to propose *variations* of current individuals to allow *exploration* of the search space over time. However, the mutation operator takes the best individuals of the previous population as input, thus it should only propose *small* variations, *exploiting* the fact that previously selected points have –relatively speaking– a higher fitness. Preferring larger variations (to move faster in parameter space) or smaller variations (to better exploit the fitness of the best current individuals) is a recurring problem commonly referred to as the *exploration–exploitation* dilemma.

The mutation operator can be interpreted as defining a neighborhood on individuals, where the function f is assumed to have small variations. Possible mutations then correspond to nearby individuals and are thus guaranteed to have similar f-values (if the assumption holds). This assumption that the f-values do not differ to much after mutation is what makes the so-called exploitation possible.

However, w.r.t. the algorithm, the important notion of neighborhood is on populations, not on individuals. If the reproduction step creates a few unfit individuals, they will not be selected for reproduction and therefore cannot impact later generations. What matters is that the reproduction process generates a population close to the previous one as a whole.

The mutation operator implicitly defines a neighborhood on populations by allowing a small variation of each individual. However, there are some settings where nearby populations can also be obtained by allowing the combination of several individuals into a new one. when such an operator exists, it is called a cross-over operator.

This implicit definition of neighborhood in parameter space makes evolutionary algorithms particularly efficient when the parameter space is discrete such as, sequences, trees or graphs, where the usual notion of distance is rarely helpful.

However, despite their effectiveness, evolutionary algorithms have long been criticized for their lack of theoretical justification. A point which is addressed in the next section.

1.8 EDAs

Estimation of Distribution Algorithms (EDAs) are mathematically principled evolutionary algorithm which do away with the biological analogy. As the name suggests EDAs are based on an unsupervised learning topic: density estimation, which will be reviewed thoroughly in chapter 3. EDAs achieve state of the art performance in the black-box optimization setting where the goal is to optimize a function f without any knowledge of how f computes its values.

EDAs propose to represent individuals as samples from a probability distribution: the so called *proposal distribution.* A population is then a set of independent and identically distributed (iid) samples from this distribution. In the preceding section, we saw that the mutation and cross-over operators served to define small possible movements around a current population. The EDA approach has the advantage of transforming the problem of moving towards better populations in the input space –which may not be well behaved– to a proxy problem which is usually well behaved: moving towards better proposal distributions.

Formally the algorithm generates a new population by taking μ samples from a proposal distribution $P_{\theta^t}(\mathbf{x})$. These μ individuals are then ranked according to their f-values and the σ best individuals are used to update the proposal distribution with a log-likelihood gradient ascent step, i.e.

$$P_{\theta^{t+1}}(\mathbf{x}) = P_{\theta^t}(\mathbf{x}) + \delta t \nabla \log P_{\theta^t}(\mathbf{x})$$

where δt is the step size of the gradient ascent update. The algorithm can then run for a number of steps, until a satisfactory solution is found. Figure 1.5 gives an example of EDA update for a Gaussian proposal distribution.

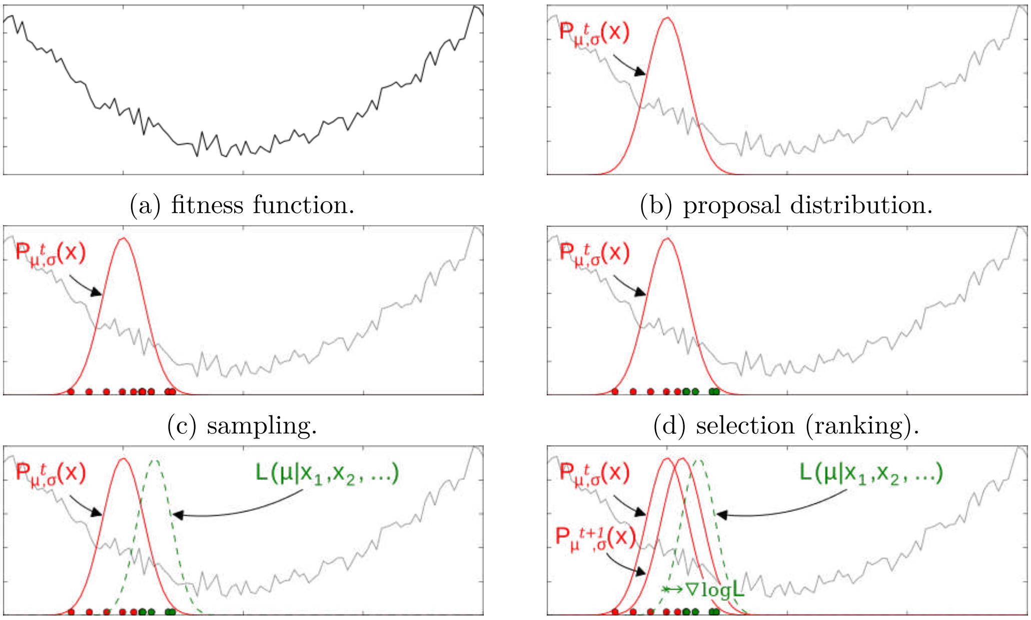
Although the purpose of the algorithm is to maximize $\mathbb{E}\left[f(\mathbf{x})\right]$, it consists in the maximization of a surrogate objective: $\mathbb{E}\left[w(\mathbf{x})\right]$, where the weight function $w(\mathbf{x})$ is equal to 1 for the σ best individuals, and 0 otherwise. This has the advantage of making the approach invariant w.r.t. monotone transformations of the objective function f.

The maximization of the surrogate objective $\mathbb{E}\left[w(\mathbf{x})\right]$ is done with gradient ascent:

$$\begin{aligned}
\nabla \mathbb{E}\left[w(\mathbf{x})\right] &= \nabla \int_{\operatorname{dom} f} w(\mathbf{x}) P_{\theta^t}(\mathbf{x}) \\
&= \int_{\operatorname{dom} f} w(\mathbf{x}) \nabla P_{\theta^t}(\mathbf{x}) \\
&= \int_{\operatorname{dom} f} \nabla \log P_{\theta^t}(\mathbf{x}) w(\mathbf{x}) P_{\theta^t}(\mathbf{x})
\end{aligned}$$

where taking σ samples from $w(\mathbf{x})P_{\theta^t}(\mathbf{x})$ can be done by taking samples from $P_{\theta^t}(\mathbf{x})$ and keeping only the σ best according to f.

This general framework can be adapted to optimize functions in $\mathbb{R}^D$ e.g. with Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen, 2008), or in discrete spaces such as $\{0,1\}^D$ with Population Based Incremental Learning (PBIL) (Baluja, 1994). It has the advantage of allowing a move towards several proposal solutions at once, contrary to methods such as hill climbing. The Population Based Incremental Learning (PBIL) algorithm for optimization over $\{0,1\}^D$ is given in Algorithm 1.4.



(a) fitness function.

(b) proposal distribution.

(c) sampling.

(d) selection (ranking).

(e) likelihood of selected points w.r.t. μ.

(f) gradient update and new distribution.

Figure 1.5: The steps of an EDA given a fitness function (a) and a Gaussian proposal distribution at time t (b). First, samples are drawn from the proposal distribution (c). The σ best samples are then selected according to their f-values (d). Finally, the likelihood of the selected samples w.r.t. the parameters (e) can be increased with a step of log-likelihood gradient ascent leading to a new proposal distribution at time $t+1$ (f).

Algorithm 1.4 The Population Based Incremental Learning (PBIL) algorithm.

```
INPUT:      $f$, a function with $\mathrm{dom}\, f = \{0,1\}^D$.
            $N$, number of proposal samples.
            $N_s$, number of samples to select at each step.
            $\delta t$, the step-size.
            $m$, the probability of a mutation.
            $\alpha$, the mutation shift.
OUTPUT:     $\hat{\mathbf{x}}$, approximation of a global minimum.
VARIABLES:  $(p_{t,1}, p_{t,1}, \dots, p_{t,D})$, the parameters of an independent
            Bernoulli proposal distribution at time $t$:
            $P_{\theta^t}(\mathbf{x}) = p_{t,1}^{x_1}(1-p_{t,1})^{(1-x_1)} \times \cdots \times p_{t,D}^{x_D}(1-p_{t,D})^{(1-x_D)}$
            for $p_{t,i}$ the probability that $x_i = 1$ at time $t$.
            $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}$ the proposal samples at time $t$, not to be
            confused with $x_1, \dots, x_D$ the components of a vector $\mathbf{x}$.
BEGIN
$p_{0,1}, \dots, p_{0,D} := \frac{1}{2}, \dots, \frac{1}{2}$
until satisfied:
        $\mathbf{x}^{(1)} \sim P_{\theta^t}(\mathbf{x}), \dots, \mathbf{x}^{(N)} \sim P_{\theta^t}(\mathbf{x})$
        rank samples ensuring $\mathbf{x}^{(1)} \leq \cdots \leq \mathbf{x}^{(N)}$
        # update probability vector
        for $i$ from 1 to $N_s$:
                for $j$ from 1 to $D$:
                        $p_{t+1,i} := p_{t,i} \times (1 - \delta t) + x_j^{(i)} \times \delta t$
        # mutate probability vector
        for $j$ from 1 to $D$:
                if $\mathcal{U}([0;1]) < m$:
                then $p_{t+1,i} := p_{t+1,i} \times (1-\alpha) + \mathcal{U}(\{0,1\}) \times \alpha$
return best solution $\hat{\mathbf{x}} := \mathbf{x}^{(1)}$.
END
```

Summary

- An optimization problem consists in finding parameters which minimize or maximize an objective function.
- In the general case, a function can have several local optima or plateaus.
- High dimensional spaces behave in non intuitive ways which can dramatically affect optimization performance.
- Any local optimum of a convex function is in fact a global optimum.
- Local optima can sometimes be found analytically using properties of the gradient and the Hessian.
- Gradient descent is an iterative optimization method which follows the direction of steepest descent at each step and can benefit from high dimensional spaces.
- In black-box optimization the landscape of the objective function cannot be studied analytically and can only be discovered through evaluation.
- Evolutionary algorithms are optimization methods particularly adapted to the black box scenario, which involve moving a population of candidate solutions towards better fitness.
- Evolutionary algorithms allow the practitioner to choose a useful implicit metric with the mutation and cross-over operators.
- EDAs are a mathematically principled form of evolutionary algorithm which move towards better solutions in parameter space by updating a proposal distribution.

We now turn to ML and show how learning problems can be formally defined with the help optimization.

2

FROM OPTIMIZATION TO MACHINE LEARNING

Optimization methods give practical means to minimize or maximize objective functions. An ML problem can then be posed from an optimization perspective by choosing a suitable objective function. The optimization procedure is then responsible for maximizing the fitness of a model for a specific task.

This approach is particularly suited to supervised learning problems where the objective is to learn a function $f^* : \mathbf{x} \rightarrow \mathbf{y}$ from a set of training examples $\{(\mathbf{x}_1, \mathbf{y}_1), \dots (\mathbf{x}_N, \mathbf{y}_N)\}$. In this case, the objective function can simply be defined as the average error of the model over this set of examples. An optimization method can then be used to find the model which minimizes the error.

We now give a presentation of supervised and unsupervised learning problems and discuss the question of generalization. Then, we study several examples: linear classification, the K-means algorithm and polynomial regression. This leads us to pose the questions of hyper-parameter selection and of feature extraction. Finally, we present the semi-supervised learning problem and show how it can be used to achieve better performance in supervised settings with the help of unsupervised data.

2.1 Supervised and unsupervised learning

In supervised learning, the objective is to approximate an unknown function f^* from a number of observations $(\mathbf{x}, \mathbf{y})$ with the assumption that $\mathbf{y} = f^*(\mathbf{x})$. These observations are called *learning examples* and are usually compiled into a set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots (\mathbf{x}_N, \mathbf{y}_N)\}$ called a *dataset*.

Definition 2.1. (*supervised learning problem*) Consider a set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots (\mathbf{x}_N, \mathbf{y}_N)\}$ and a class of functions $\mathcal{H}$. The problem of finding a function $\hat{f} \in \mathcal{H}$ matching *inputs* $\mathbf{x}_i$ to their *expected output* $\mathbf{y}_i$ as in

$$\hat{f}(\mathbf{x}_i) \approx \mathbf{y}_i$$

is called a *supervised learning problem.* The set $\mathcal{D}$ is referred to as the *training dataset* or simply training set. The class of function $\mathcal{H}$ is called the *hypothesis space.* The expected output $\mathbf{y}$ is often referred to as the *label* or the *target.*

Supervised learning problems arise in many settings but can often be reduced to either a classification problem when the label $\mathbf{y}$ is a natural number $\mathbf{y} \in \mathbb{N}$ or, a regression problem when $\mathbf{y}$ is a real number $\mathbf{y} \in \mathbb{R}^D$. We now define these two problems formally as optimization problems.

Definition 2.2. (*classification problem*) Classification concerns the case when the label $\mathbf{y}$ can be interpreted as a class variable and $\mathbf{y} \in \mathbb{N}$. The loss to minimize is usually the *misclassification rate*, i.e. the 0-1 loss:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \mathbb{1}\{\mathbf{y} \neq f(\mathbf{x})\}$$

where $\mathbb{1}$ is the indicator function equal to 1 when the argument evaluates to true and 0 otherwise.

Definition 2.3. (*regression problem*) Regression concerns the case when the target variable $\mathbf{y}$ is in $\mathbb{R}^K$ and the loss to minimize is usually the average distance between $\mathbf{y}$ and $f(\mathbf{x})$, i.e. the Mean Square Error (MSE):

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} [\mathbf{y} - f(\mathbf{x})]^2$$

In unsupervised learning we consider a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ where there is no label. The goal can be to better understand the structure of the dataset $\mathcal{D}$ or, to learn new representations and improve performance in a supervised setting.

Clustering, dimensionality reduction, and density estimation are common unsupervised learning problems and are described below. Note that we do not give a complete and formal definition of these problems, but rather try to give examples of the corresponding optimization problems.

Clustering The objective of a clustering algorithm is to group similar observations into clusters, such that points inside a cluster are similar to each other. Formally, a clustering algorithm returns a partition of observations into disjoint sets $\mathcal{C}_1, \ldots, \mathcal{C}_K$ called *clusters.* The model is often given by a set of points

$\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\}$ called *exemplars* or *centroids* such that each $\mathbf{c}_i$ is representative of the elements in the cluster $\mathcal{C}_i$. The objective is then e.g. to minimize the average distance from points in a cluster to their representing centroid, i.e. :

$$\{\hat{\mathbf{c}}_1, \hat{\mathbf{c}}_2, \ldots, \hat{\mathbf{c}}_K\} = \underset{\{\mathbf{c}_1,\mathbf{c}_2,\ldots,\mathbf{c}_K\}}{\arg\min} \sum_{i=1}^{K} \sum_{\mathbf{x}\in\mathcal{C}_i} \|\mathbf{x} - \mathbf{c}_i\| \,.$$

Dimensionality reduction The purpose of dimensionality reduction is to find a representation of lower dimensionality for the dataset $\mathcal{D}$. This can be done in several ways, for instance by assuming that the training samples are all on a sub-manifold of the input space or by trying to find a variable $\mathbf{y}$ with $\dim \mathbf{y} < \dim \mathbf{x}$ such that $\mathbf{x}$ can be reconstructed from $\mathbf{y}$, thus trying to solve the following optimization problem:

$$\{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_N\}, \hat{f} = \underset{\{\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_N\}, f\in\mathcal{H}}{\arg\min} \sum_{i=1}^{N} \|\mathbf{x}_i - f(\mathbf{y}_i)\|$$

Note that the optimization problem is then on the function f and the values $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N$ which are not provided.

Density estimation Given the training dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, the goal of density estimation is to find a probability density which could have generated the dataset, often with the assumption that the training samples are iid. This is usually done with the maximization of the log-likelihood of $\mathcal{D}$ under a parametrized family of distributions $p_\theta(\mathbf{x})$ i.e. choosing $p_{\hat{\theta}}(\mathbf{x})$ such that

$$\hat{\theta} = \underset{\theta}{\arg\max} \sum_{\mathbf{x}\in\mathcal{D}} \log p_\theta(\mathbf{x}).$$

Density estimation will be reviewed in more detail in Chapter 3.

2.2 Generalization

In the above problems, the hypothesis space $\mathcal{H}$ plays an important role. Consider, in a supervised setting, the function f such that $f(\mathbf{x}) = \mathbf{y}$ for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ and $g(\mathbf{x}) = 0$ otherwise. Clearly if f is in $\mathcal{H}$, it is optimal for any classification or regression problem according to Definitions 2.2 and 2.3. However, this function is not a desirable solution. Indeed, if a model is trained using examples $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, the objective is not to have a model that performs well only on these given examples. Instead, we want a model to *generalize* from the examples in the dataset, and give accurate values $\mathbf{y}$ for unseen points $\mathbf{x}$ which were not used during training.

Consequently, the hypothesis space should not contain functions such as the one presented above where the points in the dataset can be learned exactly at

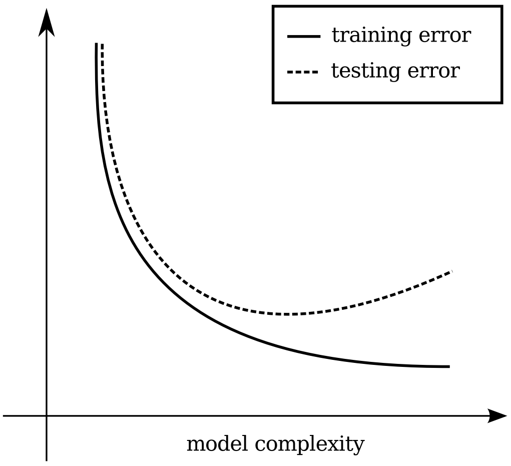


Figure 2.1: Expected variation of the training and testing error with increasing model complexity.

the expense of generalization, but should represent a hypothesis on the form of possible solutions by containing only functions which have a credible behavior. A common hypothesis for instance is that of smoothness, i.e. f is assumed to have similar values for inputs which are close to each other.

When a model matches the training examples too closely at the expense of generalization, it is said to be *over-fitting*. Over-fitting results from having a hypothesis space too big for the dataset and containing spurious functions such as the one given above. Conversely, when the hypothesis space is too small and a model is unable to capture important variations of the dataset, we say that the model is *under-fitting*.

It seems important to point out that learning without generalization is just a method for storing information and therefore, is not learning at all: the purpose of learning *is* generalization. Accordingly, we cannot assess performance from the loss on the training dataset which can be arbitrarily low when the hypothesis space is too big: we must use an other disjoint dataset. Thus, it is usual to have two datasets: a *training set* on which to minimize the loss during training, and a disjoint *testing set* to evaluate the final performance of a trained model. Figure 2.1 gives an account of how training error and testing error are expected to vary when model complexity increases.

2.3 Supervised Example: Linear classification

Let us now consider a simple classification problem on the dataset in Figure 2.2. The dataset has two classes and the input $\mathbf{x}$ is in two dimensions. We use $\times$ to represent points belonging to the first class and $\circ$ to represent points in the second class.

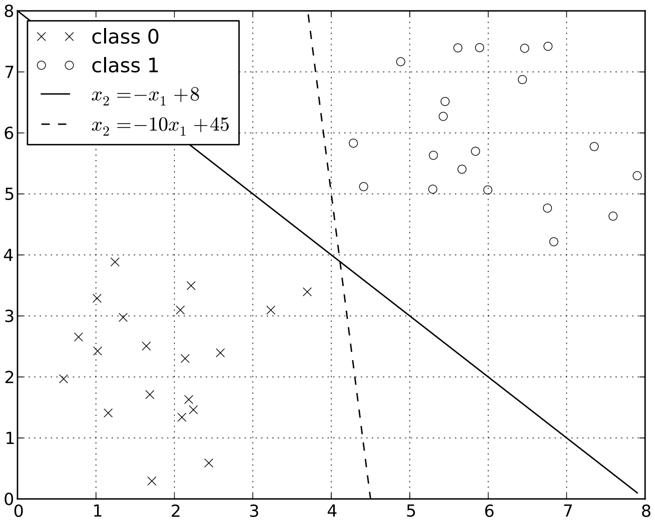


Figure 2.2: A classification dataset with two classes. The graph shows two possible separating hyperplanes.

The objective in a classification problem is to find a *separating surface* which places points from one class on one side of this surface, and points from the other class on the other side. Note that the dataset of Figure 2.2 is *linearly separable* meaning that it is possible to find a hyperplane (in this case a line) which separates the target classes. Not all datasets are linearly separable and Figure 2.3 gives examples of separating surfaces for such datasets.

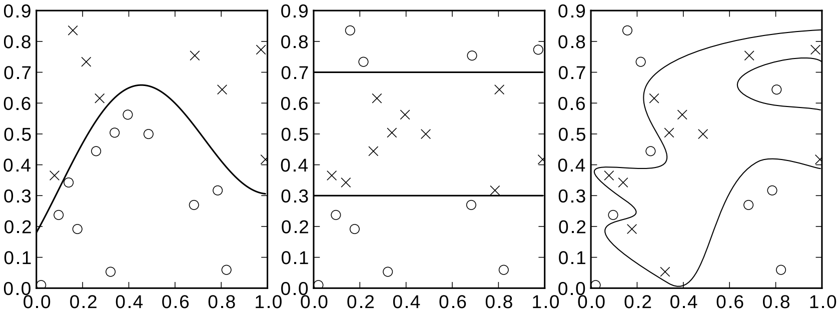


Figure 2.3: Three linearly non-separable classification datasets and a possible separating surface.

Because our dataset is linearly separable, we can then resort to a linear model to perform classification[1],e.g. :

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathbf{a}$$

where the parameters of the model are $\mathbf{a}$ and $\mathbf{w}$. Note that even though we are in a binary classification problem, $f(\mathbf{x})$ is in $\mathbb{R}$ and not in $\{0, 1\}$. In practice, the classification decision is made by using $\text{sign}(f(\mathbf{x}))$ instead of $f(\mathbf{x})$ itself. The real value can then be seen as a measure of confidence in the result. With the above model, it is common to optimize a proxy of the problem given in Definition 2.2, i.e using the MSE which is continuously differentiable instead of the misclassification rate:

$$\hat{\mathbf{a}}, \hat{\mathbf{w}} = \underset{\mathbf{a},\mathbf{w}}{\arg\min} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \left[\mathbf{y} - (\mathbf{w}^T\mathbf{x} + \mathbf{a})\right]^2$$

The problem can then be solved with gradient descent (see Chapter 1).

Figure 2.2 gives linear separation surfaces which are examples of solutions to the above optimization problem.

2.4 Unsupervised Example: Clustering and K-means

The K-means algorithm is a clustering algorithm in which centroids $\mathbf{c}_i$ are simply the arithmetic mean of points in the cluster $\mathcal{C}_i$. Accordingly, the loss to minimize is the average distance of a point $\mathbf{x}$ to the nearest centroid:

$$L(\mathcal{D}) = \sum_{i=1}^{K} \sum_{\mathbf{x}\in\mathcal{C}_i} \|\mathbf{x} - \mathbf{c}_i\| .$$

The K-means algorithm only requires a number of clusters K and random initial centroids as input. It then alternates between two steps:

1. Place in cluster $\mathcal{C}_i$, the points nearest to the centroid $\mathbf{c}_i$.

2. Set each centroid $\mathbf{c}_i$ to the arithmetic mean of the points in cluster $\mathcal{C}_i$.

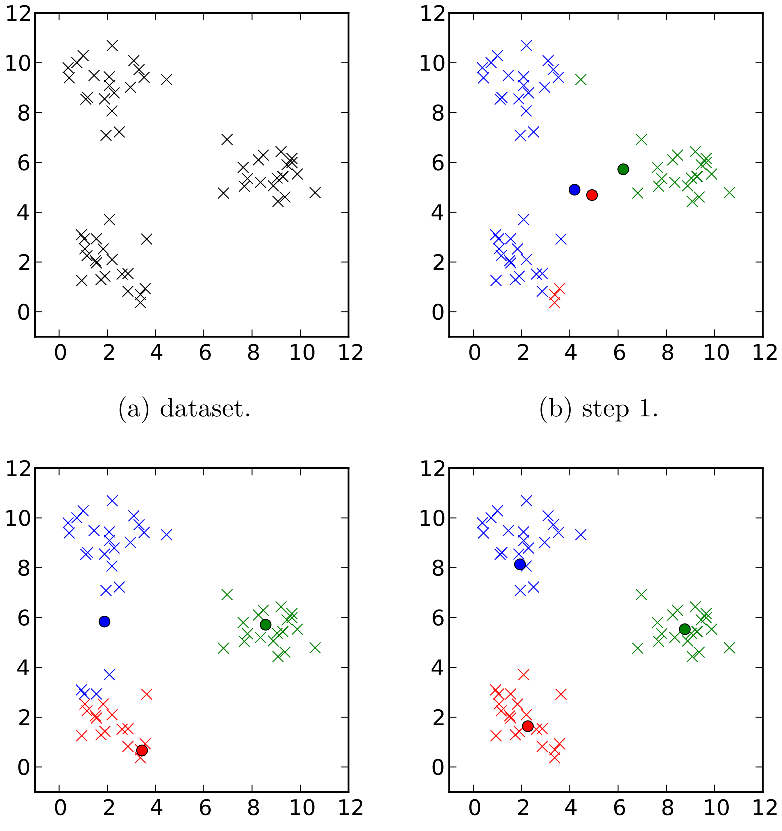
Algorithm 2.1 gives the complete algorithm and Figure 2.4 shows an example of convergence on a toy dataset.

Although we show an example of convergence, the K-means algorithm does not always converge to an appropriate solution as it can converge to a local minimum.

[1] When confronted with a new dataset, it is often a good idea to check linear separability with a linear model before trying to use more complex models.

Algorithm 2.1 The K-means clustering algorithm.

```
INPUT:      $K$, the number of clusters.
            $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\}$, initial set of centroids at step $0$.
OUTPUT:     $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\}$, final set of centroids.
VARIABLES:  $C = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K\}$, the set of sets $\mathcal{C}_i$ containing the
            points closest to $\mathbf{c}_i$.
BEGIN
until satisfied:
        for $i$ from 1 to $K$:
                $\mathcal{C}_i := \{\mathbf{x} \in \mathcal{D} | \mathbf{c_i}$ is the centroid closest to $\mathbf{x}\}$
        for $i$ from 1 to $K$:
                $\mathbf{c}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{x} \in \mathcal{C}_i} \mathbf{x}$
return $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\}$.
END
```



(a) dataset.

(b) step 1.

(c) step 2.

(d) step 3.

Figure 2.4: Convergence of the K-means algorithm on a toy 2-dimensional clustering dataset with $K = 3$. The dataset set is given in (a). The centroids ●,● and ● and the points closest to them (×,× and ×) are represented with the same color in steps 1 to 3.

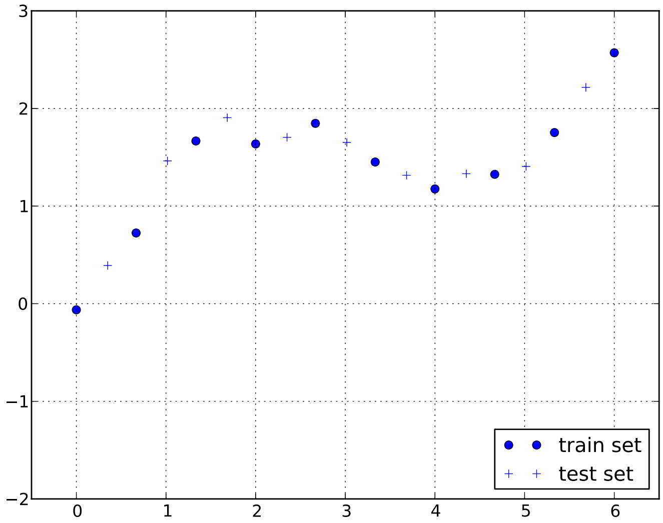


Figure 2.5: Example of a regression dataset: a noisy version of the function $\sin(x) + \frac{1}{2}x$.

The K-means algorithm can be suited to extract features for classification, however probabilistic models can be much more powerful to capture complex structure in data.

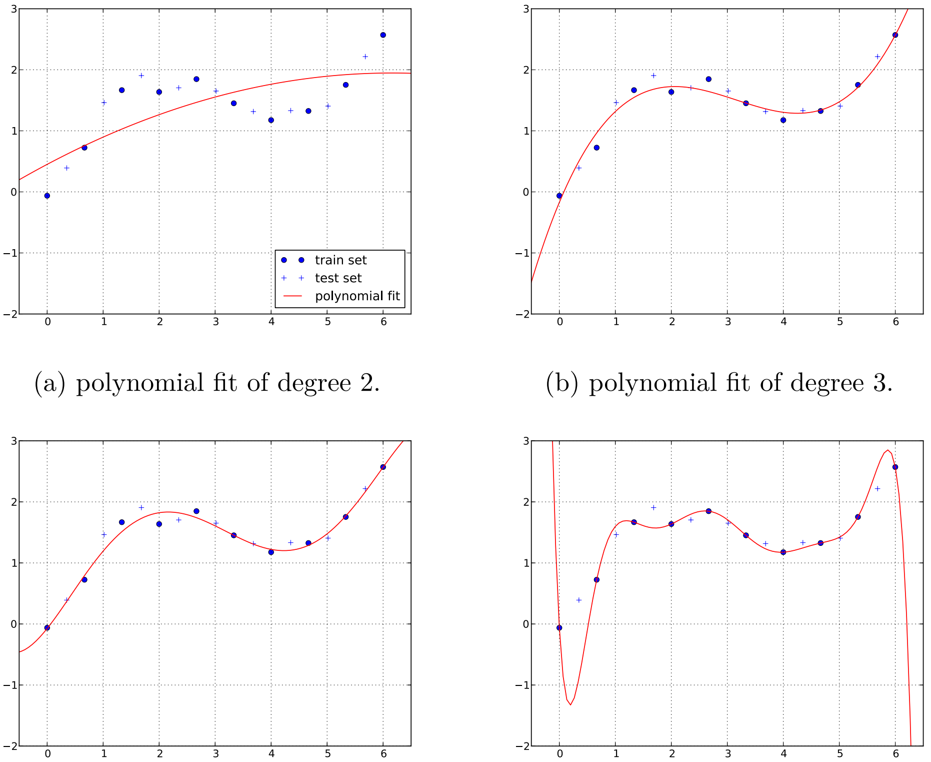
2.5 Supervised Example: Polynomial regression

Let us now consider a simple regression problem on the dataset of Figure 2.5. The problem consists in finding an approximation of the function $f(x) = \sin(x) + \frac{1}{2}x$ given observations of $f(x)$ perturbed by a small Gaussian random noise. Notice that the observations are split in two datasets: the training set for learning parameters, and the test set to evaluate the quality of the resulting model.

In order to solve this problem, we propose to use polynomial regression, that is to try and fit a polynomial of fixed degree K to the training points. The model can be defined by:

$$\hat{f}(x) = \sum_{k=0}^{K} a_k x^k$$

where $(a_0, \ldots, a_K)$ are the parameters of the model.



(a) polynomial fit of degree 2.

(b) polynomial fit of degree 3.

(c) polynomial fit of degree 5.

(d) polynomial fit of degree 9.

Figure 2.6: Best polynomial fits for several degrees on the example regression dataset. (a) gives a case of under-fitting. (b) and (c) are examples of what would be considered good fits. In (d) we see an example of over-fitting.

Learning then consists in an optimization problem where the goal is to find the parameters a_k that minimize the MSE[2], namely find $\hat{f}$ such that

$$\hat{f} = \underset{(a_0,\ldots,a_K)}{\arg\min} \sum_{(x,y)\in\mathcal{D}} \left[y - \sum_{k=0}^{K} a_k x^k \right]^2$$

Depending on the degree we chose for the polynomial, Figure 2.6 shows that the results can be very different. When the degree is too small, we can see an example of under-fitting: the model is too simple to represent the target function accurately. On the other hand, if the degree of the polynomial is too large, we risk the problem of over-fitting: the model can fit the training points more closely but points in the testing set are not well approximated anymore.

[2] There are many readily available algorithms capable of solving this kind of problem. Here we use the polyfit function of the numpy python package.

Trying to choose the best degree K for the polynomial, consists in a secondary optimization problem. In this setting, K is a special kind of parameter, i.e. a *hyper-parameter*.

2.6 Model selection

A learning algorithm consists in an optimization algorithm applied to the parameters of a model in order to minimize a specific loss. Nonetheless, it is often the case that the learning algorithm itself depends on some parameters being set, as e.g. the complexity of the model (the degree K in the previous example) or the learning rate of a gradient descent procedure.

Such parameters which are outside of the main optimization procedure are called *hyper-parameters*. Accordingly, the problem of choosing suitable values for the hyper-parameters is called *hyper-parameter selection* or *model selection* and consists in an optimization problem in which learning models is considered as a sub-problem.

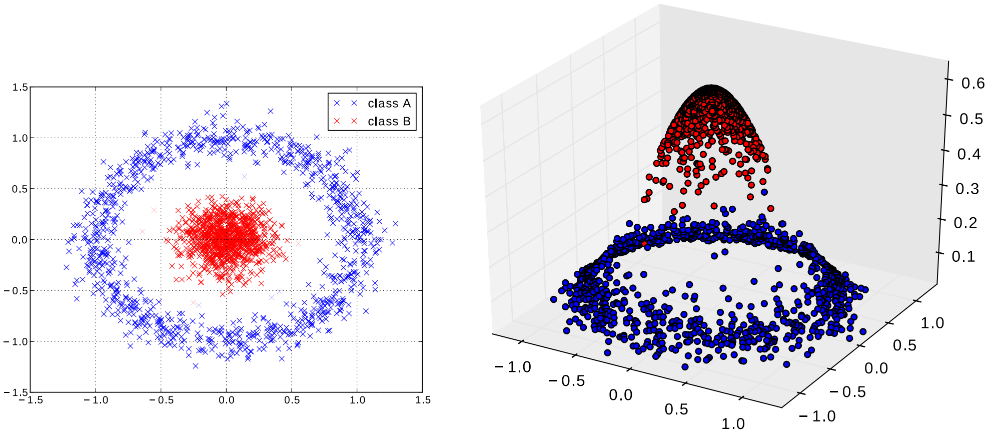
Although we need to optimize the hyper-parameters w.r.t the performance on some dataset, we cannot choose model complexity according to the training set because it would lead to poor generalization. In our example, choosing the best degree K according to the training set would inevitably lead to choosing higher degrees for the polynomial, even though they do not make for a better fit on the testing set.

However, the testing set is meant to be used for evaluating the performance of a model on unseen data and cannot therefore be used during hyper-parameter selection. If it were, the optimization process would choose values particularly suited to maximize performance on the test set and thus artificially increase the test set performance.

To solve this problem, the solution usually retained is to use a third dataset called a *validation dataset* to optimize the hyper-parameters. The testing error can then be used safely to evaluate performance.

2.7 Changing representations

In the preceding problems, we have seen how to learn a function $f^* : \mathbf{x} \to \mathbf{y}$ from examples. However, when the relation between the input $\mathbf{x}$ and the label $\mathbf{y}$ is too complex to be captured by simple models, we are faced with both a generalization problem and a computational problem. Generalization becomes difficult because large models are more prone to over-fitting and optimization becomes more expensive because it takes place in a high dimensional space.



(a) A non linearly separable dataset.

(b) Possible feature space representation.

Figure 2.7: Illustration of the simplification power of a feature space. Given the non linearly separable classification problem (a), the projection $\mathbf{\Phi}(x, y) = (x, y, z)$ where z is a feature such that $z = \exp\left[-(x^2 + y^2)/2\right]$ makes the problem linearly separable in (b).

2.7.1 *Preprocessing and feature space*

A solution, instead of trying to increase model complexity, possibly at the expense of generalization and computational cost, is to create a vector of features $\mathbf{\Phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_K(\mathbf{x}))$ as a *pre-processing* step, which are then meant to be used as input of the learning procedure. The objective of the new supervised learning problem is then to approximate the function $f^* : \mathbf{\Phi}(\mathbf{x}) \rightarrow \mathbf{y}$. If the features $\phi_i(\mathbf{x})$ extract relevant information from the raw data $\mathbf{x}$, it can result in a simplified problem which can hopefully be solved using a simple model. The feature extraction function $\mathbf{\Phi}$ can be seen as projecting data into a *feature space*, i.e. representing data so as to make the euclidean distance between training examples $d_E(\mathbf{\Phi}(\mathbf{x}), \mathbf{\Phi}(\mathbf{x}'))$ more meaningful than in the input space. Figure 2.7 shows how such a projection can make a non linearly separable problem separable.

2.7.2 *The kernel trick*

In the illustration above, the projection $\mathbf{\Phi}$ is easily computable which allows us to show the data in the feature space. However, most linear methods do not require the exact coordinates $\mathbf{\Phi}(\mathbf{x})$ of points in the new feature space but only rely on the inner products $\langle \mathbf{w}, \mathbf{x} \rangle$ between features $\mathbf{w}$ and input vectors $\mathbf{x}$. Whereas the usual inner product is taken in the input space where $\langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}^T\mathbf{x}$, the so called *kernel trick* proposes to directly define an inner product $K(\mathbf{w}, \mathbf{x})$ without

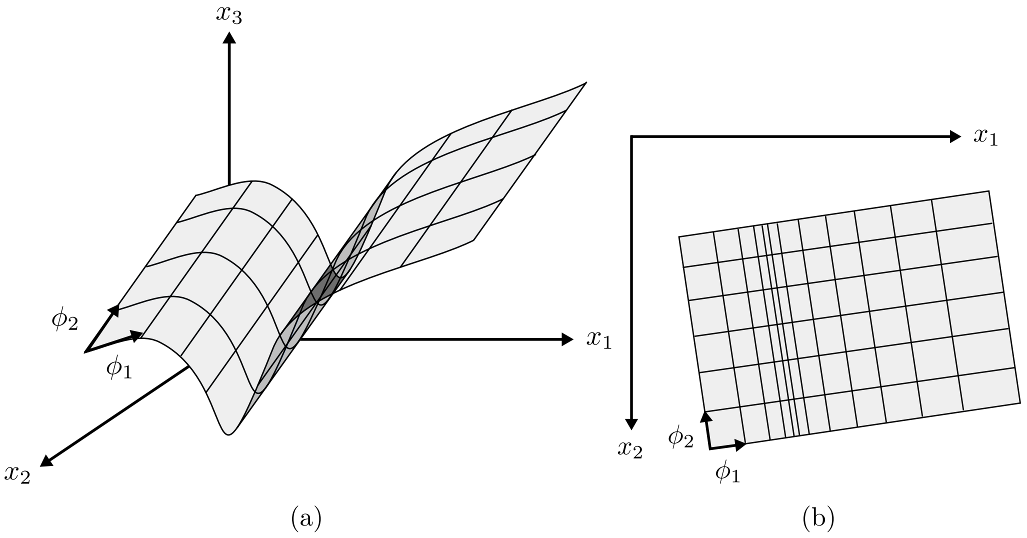


Figure 2.8: Illustration of a two dimensional manifold immersed in 3D space (a). The same two dimensional manifold immersed in 2D space (b).

explicitly formulating the feature space to which it corresponds. Common choices of kernels include the polynomial kernel:

$$K(\mathbf{w}, \mathbf{x}) = (\mathbf{w}^T\mathbf{x} + c)^d \text{ for } d \geq 0$$

and Radial Basis Function (RBF) kernels, such as the Gaussian kernel:

$$K(\mathbf{w}, \mathbf{x}) = \exp\left[-\frac{\|\mathbf{w} - \mathbf{x}\|^2}{2\sigma^2}\right]$$

Figure 2.7 can be interpreted as showing the value of the RBF kernel $K(\mathbf{x}, \mathbf{0})$ corresponding to the projection of data points $\mathbf{x}$ on the zero vector in the RBF feature space.

Note that the RBF kernel corresponds to a projection into a feature space of infinite dimension which may seem counter-productive. However, high dimensional spaces make it much easier to find a linear separating surface which simplifies the problem considerably.

2.7.3 *The manifold perspective*

Under the manifold hypothesis, the data is assumed to lie on a manifold (usually low dimensional) immersed in the input space. Figure 2.8(a) gives a representation of a two dimensional manifold in 3D space. Instead of considering the coordinates of points $\mathbf{x}$ in the original input space, an interesting approach is then to try and find a feature space which recovers the structure of this manifold, i.e. to find features $\mathbf{\Phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_K(\mathbf{x}))$ which correspond to the coordinates of $\mathbf{x}$ inside the manifold. This search for a good coordinate system is implicitly linked to the notion of metric. Namely, if the features $\mathbf{\Phi}$ represent the coordinates of points inside the manifold, the Euclidean distance $\|\mathbf{\Phi}(\mathbf{x}_1) - \mathbf{\Phi}(\mathbf{x}_2)\|$ of two points in this new coordinate system should be more informative than the Euclidean distance $\|\mathbf{x}_1 - \mathbf{x}_2\|$ in the input space.

Although the manifold perspective usually considers low dimensional manifolds immersed in a high dimensional space, the idea can also be applied to consider manifolds of same (or even greater[3]) dimension than the input. Figure 2.8(b) shows how the manifold perspective can be useful to understand the impact of metrics in this setting: even though it shows a two dimensional manifold in 2D space, the Euclidean metric in the input space does not reflect the structure of the manifold, especially in the region where it folds. By using features, it may be possible to recover a more suitable coordinate system, and thus a better metric.

2.7.4 *Unsupervised representation learning*

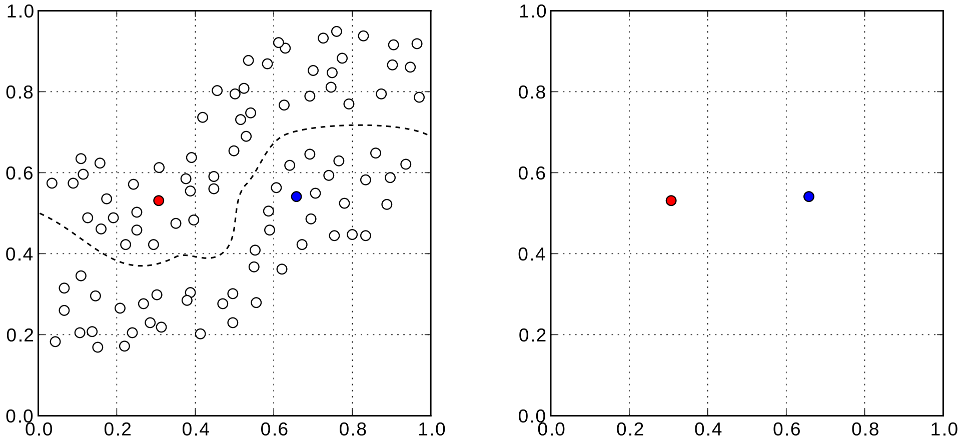
Although the projection into a feature space can be a very powerful tool, interesting features are often found with hard work, sometimes after years of research. Nevertheless, it is sometimes possible to *learn* interesting features with an unsupervised algorithm, i.e. with *unsupervised representation learning,* a central point of this thesis. Assuming a suitable set of features can be learned, the final supervised problem becomes simpler.

Putting aside the usually simple final problem, one could argue that learning representations only transforms a supervised learning problem into an equally difficult unsupervised learning problem. However, this transformation has several benefits.

BETTER GENERALIZATION In any practical application, the input variable $\mathbf{x}$ is expected to carry a lot of information about itself and only little information about the target variable $\mathbf{y}$. Accordingly, an unsupervised learning problem on the variable $\mathbf{x}$ has access to a lot of information during learning and is thus less prone to over-fitting than the supervised learning problem on $\mathbf{x}$ and $\mathbf{y}$. Additionally, once a representation has been obtained with unsupervised learning, the final supervised learning problem can be solved with a small number of parameters which means that over-fitting is, again, less likely.

ACCESS TO MORE DATA WITH SEMI-SUPERVISED LEARNING Learning representations with an unsupervised learning algorithm has the advantage that it only depends on unlabeled data which is easily available in most settings. Because more data is available, more complex models can be learned without an adverse effect on generalization. In essence, the complete learning procedure

[3] Unfortunately, the author was unable to make a nice figure illustrating this fact.



(a) A semi-supervised dataset.

(b) Corresponding supervised dataset.

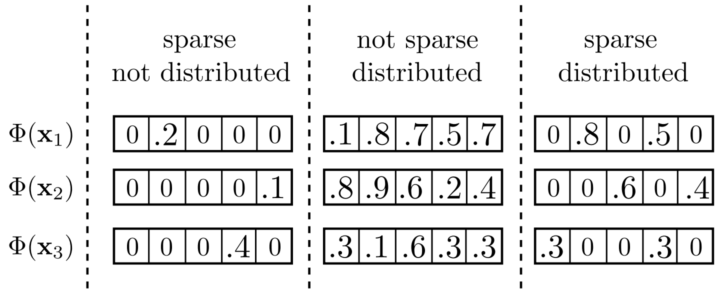
Figure 2.9: Example of the influence of unlabeled data in the semi-supervised setting. The unlabeled samples are represented with ○, and the labeled samples with ● and ● depending on their class. The unlabeled data in (a) gives a good picture of the data distribution and may allow more complex models to be learned than in (b) where there are too few samples to find a suitable separating surface.

can then leverage the information contained in an unlabeled dataset to perform better on a supervised task. This approach is known as *semi-supervised learning*. Formally, a semi-supervised learning problem given two datasets

$$\begin{aligned} \mathcal{D}_L &= \{(\mathbf{x}_1, \mathbf{y}_1), \dots (\mathbf{x}_L, \mathbf{y}_L)\} \\ \text{and } \mathcal{D}_U &= \{\mathbf{x}_{L+1}, \dots, \mathbf{x}_N\} \end{aligned}$$

consists in finding a function $\hat{f}$ such that $\hat{f}(\mathbf{x}_i) \approx \mathbf{y}_i$ when a label is available. Figure 2.9 gives an illustration of the usefulness of unlabeled data to solve a supervised task.

Sparse and distributed representations Although it is hard to define what constitutes a good representation in general, sparse and distributed representations seem to have interesting properties and have been the subject of a lot of attention in recent years. Formally, a sparse representation is such that the feature vector $\mathbf{\Phi}(\mathbf{x})$ contains many zeros. The sparsity is often measured with the L_0 norm which counts the number of non-zero elements. When trying to learn a representation with an unsupervised learning algorithm, sparsity can be seen as a way to impose a constraint on the hypothesis space $\mathcal{H}$, reducing the size of the search space and therefore improving generalization. Distributed representations concern the case where each input is represented by several features. Although this does not result in a reduced search space, the representation of input patterns according to several distinct attributes allows for *non-local generalization*.



	sparse not distributed	not sparse distributed	sparse distributed
$\Phi(\mathbf{x}_1)$	0 .2 0 0 0	.1 .8 .7 .5 .7	0 .8 0 .5 0
$\Phi(\mathbf{x}_2)$	0 0 0 0 .1	.8 .9 .6 .2 .4	0 0 .6 0 .4
$\Phi(\mathbf{x}_3)$	0 0 0 .4 0	.3 .1 .6 .3 .3	.3 0 0 .3 0

Figure 2.10: Examples of sparse and distributed representations. A sparse non-distributed representation can only have one non-zero element for any input vector (left). A non-sparse distributed representation uses all variables to represent an input (center). A sparse distributed representation can use several non zero elements but must have many zeros (right).

Namely, let us consider that a set of features $\mathbf{\Phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_K(\mathbf{x}))$ has been obtained with an unsupervised algorithm. Even if an input $\mathbf{x}$ is not close to any training example, it may still be interpreted as an unseen combination of existing features. An illustration of sparse and distributed representations is given in Figure 2.10.

Interestingly, training sparse distributed representations on natural images[4] seems to result in features which have some resemblance with the neuron receptive fields of a primate's visual cortex (Olshausen and Field, 1996, 1997). Therefore, sparse distributed representations may have (besides their theoretical benefits) the advantage of representing information in a manner similar to ours.

[4] The so-called natural image distribution usually corresponds to images found in the outside world, i.e. images of trees, plains and lakes, but also animals, buildings, planes, boats etc.

Summary

- ML problems can be posed as optimization problems where the objective function represents an error or performance measure on a dataset of training examples.
- The objective of classification is to minimize the misclassification rate. The goal of regression is to minimize the MSE.
- The fundamental problem of ML is generalization to unseen examples.
- ML relies on models to represent assumptions about the regularities of a dataset.
- Generalization is achieved by controlling model complexity to avoid under-fitting or over-fitting.
- K-means is a clustering algorithm which alternates between an update of clusters given centroids and an update of the centroids given clusters.
- Model selection can be seen as a secondary learning problem where the objective is to find hyper-parameters which help maximizing generalization.
- Supervised learning can benefit from a new representation which corresponds to a mapping of the input to a new feature space.
- A new feature space can be obtained without human intervention by learning representations with an unsupervised algorithm.
- Learning representations with an unsupervised algorithm has several benefits w.r.t. generalization.
- Learning sparse representations may improve generalization if we can assume that inputs can be represented by a limited number of features.
- Learning distributed representations may allow for non-local generalization if each example can be interpreted as combining several features.
- Combining sparse and distributed representations leads to features which are similar to those found in primate brains.

We now consider the probabilistic approach and show how it applies in the ML setting.

3

LEARNING WITH PROBABILITIES

We have seen how a learning algorithm can be posed as an optimization one. However, learning from data benefits greatly from a probabilistic perspective. Namely, Bayesian probability theory gives a sound mathematical framework for updating models based on data observations.

In this chapter, we start by giving a quick review of basic notions in probability theory and then present how to estimate distributions in the Bayesian framework. This leads us to review the concepts of maximum likelihood and maximum a posteriori. We give an example in the case of the Gaussian family and give the example of polynomial regression a probabilistic perspective. This theoretical framework then allows us to introduce the possibility of learning representations with probabilistic models. We describe the Expectation Maximization (EM) algorithm and how it can be applied to Gaussian mixtures. Finally, we revisit optimization by considering the specific problem of maximizing the likelihood of a probabilistic model and show how a suitable metric (the Fisher metric) can lead to the natural gradient which improves the ordinary gradient descent procedure by making it invariant w.r.t. parametrization.

3.1 Notions in probability theory

A probability can be seen as a number between 0 and 1 indicating a degree of belief. This is referred to as the Bayesian view, in contrast with the frequentist view in which a probability represents the average number of times an event occurs, in the limit of infinitely many experiments. In support of the Bayesian view, Cox (1946) argues that any system of beliefs consistent with common sense must satisfy the rules of probability which we state informally in table 3.1.

Note that the rules of frequentist and Bayesian probabilities are the same, but the Bayesian interpretation has the advantage of being much more widely applicable, and without the need for complicated arguments to justify why the question concerns a repeatable experiment.

Consider for instance the event A="The world will end in 2012". It is quite natural to consider the belief we have in the realization of this event as a probability. We can then reflect on $P(A)$ using the rules of probability to connect

BASIC NOTIONS	
$P(A)$	represents the belief that some event A will happen.
$P(\bar{A})$	represents the belief that A will not happen.
$P(A) = 0$	represents impossibility of A.
$P(A) = 1$	represents certainty of A.
$P(A \cap B)$	represents the belief that A and B will both happen.
$P(A \cup B)$	represents the belief that either A or B will happen (possibly both).
$P(A\|B)$	represents the belief that A will happen given that B has happened.
AND NOW A FEW RULES	
$P(\bar{A}) = 1 - P(A)$	Complementary event.
$P(A \cap B) = P(A)P(B)$ if and only if A and B are independent .	Independent events
$P(A \cup B) = P(A) + P(B) - P(A \cap B)$.	Union of events.
$P(A \cap B) = P(A\|B)P(B)$	Conditional probability.
RANDOM VARIABLES	
$P(X = x)$	represents the probability that a random variable X will take the value x.
$P(x)$	is a shorthand for $P(X = x)$ when there is no ambiguity on the random variable.
$P(X = x, Y = y)$	is called the joint probability of X and Y. It can be thought of as $P(X = x \cap Y = y)$.
$P(x, y)$	is a shorthand for $P(x = X, y = Y)$ when there is no ambiguity on the random variables.
RULES ON RANDOM VARIABLES	
$P(x, y) = P(x)P(y)$ if and only if X and Y are independent.	Independent random variables.
$P(x, y) = P(x\|y)P(y) = P(y\|x)P(x)$	Product rule.
$P(x) = \sum_y P(x, y)$	Sum rule.
$P(y\|x) = \dfrac{P(x\|y)P(y)}{P(x)}$	Bayes' rule.

Table 3.1: Basic notions and rules of probability theory.

it to other events, maybe considering conditional probabilities such as $P(A|B)$ with "North Korea will test a nuclear delivery system in 201". In the frequentist interpretation, it is difficult to see A or B as repeatable events and therefore to study them with a probabilistic perspective without an abstruse argument considering Quantum theory and many universes.

Importantly, the world did not end in 2012 and when the frequentist can only say that we observed one of two possible outcomes, the Bayesian can now assert with confidence $P(A) = 0$. In the Bayesian framework, it is natural to update our beliefs when confronted with experimental evidence, i.e. to *learn from data*. We now present how to leverage this possibility in the context of ML.

3.1.1 *Sampling from complex distributions*

Learning with probabilities often involves complex distributions in high dimensional spaces which makes them difficult to approach analytically. Nevertheless, by taking samples, it becomes possible to estimate almost any quantity of interest empirically (Robert and Casella, 2005; Neal, 1993). Most notably, these quantities of interest are often expressed as expectations:

$$\mathbb{E}_{x\sim p(x)}\left[h(x)\right] = \int_{\mathcal{X}} h(x)p(x)dx$$

where $x \in \mathcal{X}$, and $x \sim p(x)$ represents the fact that x is a sample from the distribution p. Using iid samples $x_1, x_2, \ldots, x_N$, the above expectation can be estimated with the average of $h(x)$:

$$\hat{\mathbb{E}}_{x\sim p(x)}\left[h(x)\right] = \frac{1}{N}\sum_{i=1}^{N} h(x_i)$$

However, taking the required samples in the distribution p can be difficult. Even when the density $p(x)$ is tractable, there is often no easy way to take samples from p[1]. It is then interesting to temporarily circumvent the problem. A first approach is to take samples from a different distribution q using a selection or weighting scheme to ensure that the resulting samples are distributed according to the target distribution p, the principle behind rejection sampling and importance sampling. A second possibility is to use an Monte Carlo Markov Chain (MCMC) method such as Metropolis-Hastings or Gibbs sampling which relies on a Markov chain to produce samples.

Rejection sampling, importance sampling, the Metropolis-Hastings algorithm and Gibbs sampling are presented below.

[1] In many cases, the practitioner must deal with the additional difficulty of only knowing $p(x)$ up to a constant.

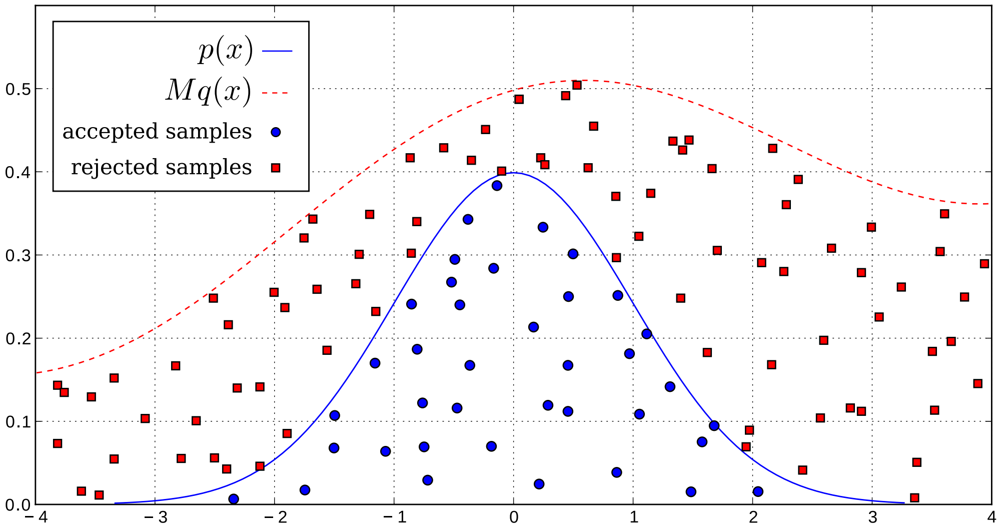


Figure 3.1: In rejection sampling, taking a sample x from q and a sample u from $\mathcal{U}[0; Mq(x)]$, results in a uniform distribution of points (x, u) below the graph of $Mq(x)$. Samples from p can then be obtained by accepting only the samples such that $u < p(x)$.

Rejection sampling Let us suppose that the distribution p is bounded by Mq for M a constant and q some distribution from which samples can be taken easily. We can then generate samples (x, u) from the joint distribution $\mathcal{U}(\mathcal{X} \times [0; Mq(x)])$: taking a sample x from q, and then taking a sample u in $[0; Mq(x)]$.

These samples correspond to uniformly distributed points below the graph of $Mq(x)$. The distribution p can then be recovered by accepting only samples such that $u < p(x)$. This process is explained in Figure 3.1 and given in Algorithm 3.1.

If the envelope $Mq(x)$ is too far from $p(x)$, a large number of samples from $q(x)$ will be needed before one is accepted. Thus the efficiency of rejection

Algorithm 3.1 The rejection sampling algorithm.

```
INPUT:      $p(x)$, the target distribution.
            $q(x)$, an instrumental distribution easy to sample from.
            $M$, a constant such that $\forall x \in \mathcal{X}, p(x) \leq Mq(x)$
OUTPUT:     $x$, a sample from the distribution $p$.
VARIABLES:  $u$, an auxiliary random variable.
BEGIN
until a sample is accepted:
        $x \sim q(x)$
        $u \sim \mathcal{U}[0; Mq(x)]$
        if $u < p(x)$ accept the sample $x$
        otherwise reject the sample
END
```

sampling depends on whether the acceptance ratio $\frac{\text{number of accepted samples}}{\text{total}}$ can be kept sufficiently high.

Furthermore, rejection sampling does not behave well in high dimensional spaces. Remember that in high dimension, a hypercube of side 0.99 only contains a small fraction of the volume of a larger cube of side 1. Similarly, in high dimensional spaces, the density $p(x)$ tends to represent only a small fraction of the envelope $Mq(x)$ leading to very low acceptance rates.

Importance sampling It is possible to sample from any distribution without rejecting samples, using a sampling distribution q and using the importance sampling identity, i.e.

$$\int_{\mathcal{X}} h(x)p(x)dx = \int_{\mathcal{X}} h(x)\frac{p(x)}{q(x)}q(x)dx$$

Accordingly, an estimate for the above expectation can be obtained by taking iid samples $x_1, x_2, \ldots, x_N$ from q and computing:

$$\sum_{i=1}^{N} h(x)\frac{p(x)}{q(x)}$$

where $\frac{p(x)}{q(x)}$ are the importance sampling weights.

Importance sampling has the advantage of not rejecting any sample, therefore all samples are used as part of the estimation. However, if the sampling distribution q does not fit p closely enough, important regions of p can be completely ignored and lead to an infinite variance of the estimator.

Although importance sampling can be very efficient, finding a suitable distribution q which is both easy to sample from and close enough to the target distribution f is sometimes impossible which is why more complex sampling schemes are sometimes needed.

Metropolis-Hastings algorithm Let us consider Markov chains which admit a unique stationary distribution π. By designing a Markov chain such that $\pi(x) = p(x)$, we can start at a random position $\mathbf{x}^{(0)}$ and run the Markov chain until it converges to its stationary distribution to produce a sample from $p(x)$. f

To ensure that $\pi(x) = p(x)$, it is sufficient that the transition probabilities $T(x \to x')$ from one state to the next satisfy the condition of detailed balance, i.e. :

$$\forall x, \forall x',\, p(x)T(x \to x') = p(x')T(x' \to x)$$

where $T(x \to x')$ is the transition probability of the Markov chain from state x to state x'. A Markov chain which satisfies the above property and is ergodic can be proved to converge to the distribution $p(x)$ as time goes to infinity.

Algorithm 3.2 The Metropolis-Hastings algorithm.

```
INPUT:      $p(\mathbf{x})$, the target distribution.
            $T(\mathbf{x} \to \mathbf{x}')$, the proposal probability of jumping from
            state $\mathbf{x}$ to state $\mathbf{x}'$.
            $\mathbf{x}^{(0)}$, the initial state of the Markov chain.
            $B$, a burn-in period.
            $K$, a number of iterations between collected samples.
OUTPUT:     $\mathcal{S} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, a set of samples from the distribution
            $p$.
VARIABLES:  $\mathbf{x}^{(t)}$, the state of the Markov chain at time $t$.
BEGIN
for $t = 0$ to $B + N \times K - 1$:
        while $\mathbf{x}'$ is not accepted:
                $\mathbf{x}' \sim T(\mathbf{x}^{(t)} \to \mathbf{x}')$
                $a \sim \mathcal{U}[0;1]$
                if $a < \frac{p(\mathbf{x}')T(\mathbf{x}' \to \mathbf{x})}{p(\mathbf{x})T(\mathbf{x} \to \mathbf{x}')}$:  accept $\mathbf{x}'$
                else:  reject $\mathbf{x}'$
        $\mathbf{x}^{(t+1)} := \mathbf{x}'$
        if $t \geq B$ and $((t - B) \mod K) = 0$:   $\mathcal{S} := \mathcal{S} \cup \{\mathbf{x}^{(t+1)}\}$
return $S$
END
```

The Metropolis-Hastings algorithm which is based on this principle is given in Algorithm 3.2.

In practice, the convergence to p as time goes to infinity means that a Markov chain must run for many iterations before a sample can reasonably be assumed to be distributed according to the target distribution. Note that, even if $x^{(t+1)}, x^{(t+2)}, \ldots, x^{(t+N)}$ are theoretically better samples in terms of convergence, nearby samples are not independent. This means that it is necessary to wait a number of steps between each sample to ensure that the Markov chain has enough time to produce independent samples.

Despite the fact that only a fraction of the algorithm's iterations actually result in a valid sample, the somewhat high computational cost is counterbalanced by the fact that the Metropolis-Hastings algorithm does not suffer from the curse of dimensionality. This makes the Metropolis-Hastings algorithm and its variants especially useful when confronted with complex distributions in high dimension.

Gibbs sampling When trying to sample from a joint distribution $p(\mathbf{x}) = p(x_1, x_2, \ldots, x_D)$ it is sometimes easy to sample from the conditional distributions $p(x_i | x_1, x_2, \ldots x_{i-1}, x_{i+1}, \ldots, x_D)$. The Gibbs sampling algorithm is a variant of the Metropolis-Hastings algorithm which proposes to repeatedly sample each variable given the others. This results in a Markov chain which converges to the joint distribution $p(x_1, x_2, \ldots, x_D)$.

Algorithm 3.3 The Gibbs sampling algorithm.

```
INPUT:      $p(x_i|x_1, x_2, \dots x_{i-1}, x_{i+1}, \dots, x_D)$, the conditional
            probabilities of the target distribution $p(\mathbf{x})$.
            $\mathbf{x}^{(0)} = x_1^{(0)}, x_2^{(0)}, \dots, x_D^{(0)}$, the initial state of the Markov
            chain.
            $K$, a number of iterations before taking a sample.
OUTPUT:     $\mathbf{x}$, a sample from the distribution $p$.
VARIABLES:  $\mathbf{x}^{(t)} = x_1^{(t)}, x_2^{(t)}, \dots, x_D^{(t)}$, the state of the Markov chain at
            time $t$.
BEGIN
for $t = 1$ to $K$:
        for $i = 1$ to $D$:
                $x_i^{(t)} = p(x_i|x_1^{(i)}, x_2^{(i)}, \dots x_{i-1}^{(i)}, x_{i+1}^{(i-1)}, \dots, x_D^{(i-1)})$
$\mathbf{x} := x_1^{(K)}, x_2^{(K)}, \dots, x_D^{(K)}$
return $\mathbf{x}$
END
```

A visualization of the Gibbs sampling algorithm is shown in Figure 3.2, the full algorithm is given in Algorithm 3.3.

Gibbs sampling is often a very efficient sampling algorithm when the conditional distributions are available.

3.2 Density estimation

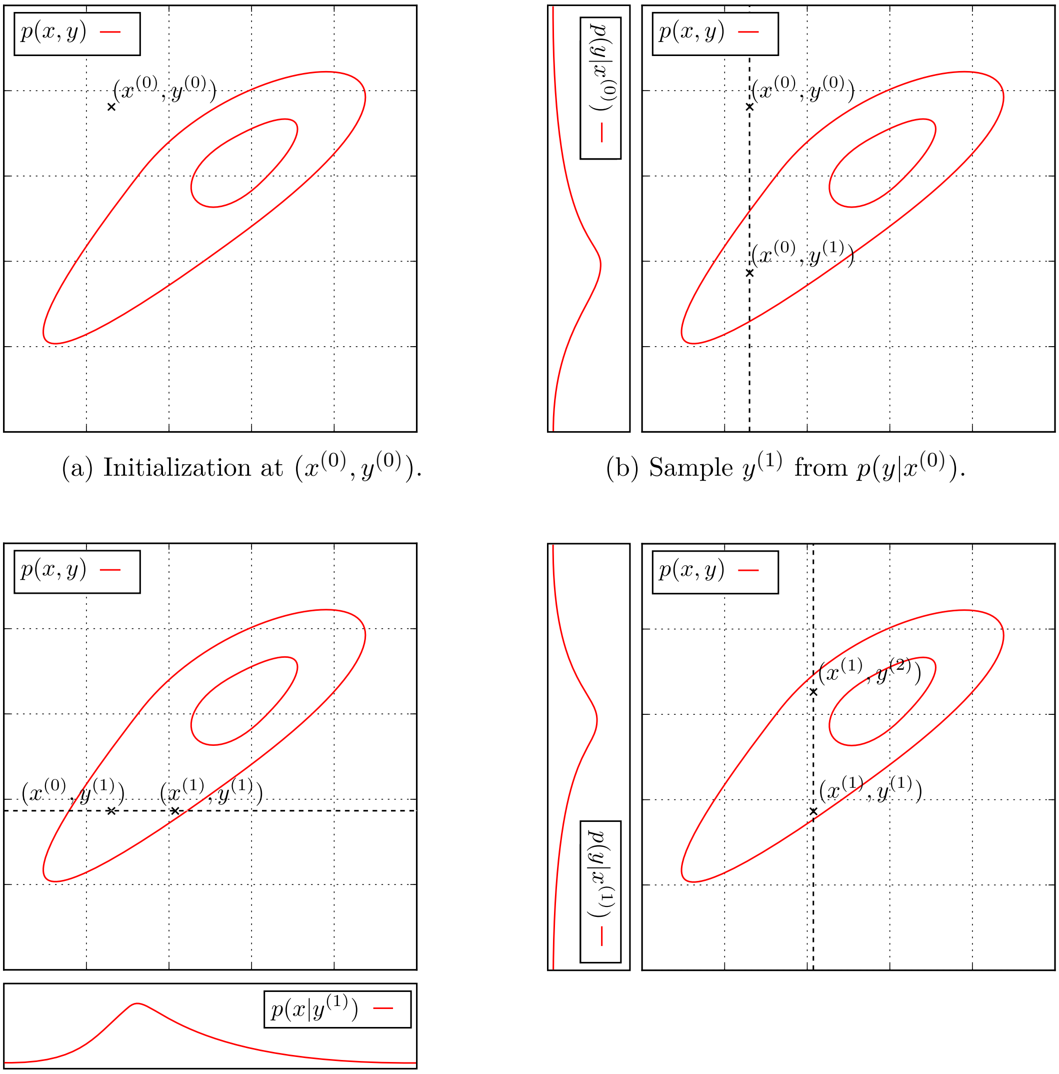
Learning with a probabilistic approach often reduces to the problem of density estimation: trying to find a probability distribution $p(\mathbf{x})$ which is likely to have generated the dataset $\mathcal{D}$.

The search for a suitable $p(\mathbf{x})$ is usually limited to a specific family of probability distributions such as the Gaussian family $\mathcal{N}(\mu, \sigma^2)$ of mean μ and variance σ^2. In practice, the parameters can be regrouped in a set of parameters θ; for the Gaussian distribution, we have for instance $\theta = \{\mu, \sigma^2\}$. The problem is then to find the likely parameters θ given that $p_\theta(\mathbf{x})$ should have generated the dataset $\mathcal{D}$.

One can easily understand that not all parameter values are equally likely. For instance, if the dataset consists of points x between 100 and 101, the standard normal distribution $\mathcal{N}(0, 1)$ centered on 0 is a very unlikely candidate.

Although the above notations may seem to be specific to unsupervised learning, density estimation also applies to supervised learning. The goal for a dataset $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$ is then to find a conditional distribution $p(\mathbf{y}|\mathbf{x})$ which is likely to have generated each $\mathbf{y}_i$ given $\mathbf{x}_i$.

We now look into several approaches which can be used to estimate distributions.



(a) Initialization at $(x^{(0)}, y^{(0)})$.

(b) Sample $y^{(1)}$ from $p(y|x^{(0)})$.

(c) Sample $x^{(1)}$ from $p(x|y^{(1)})$.

(d) Sample $y^{(2)}$ from $p(y|x^{(1)})$.

Figure 3.2: Visualization of the Gibbs sampling algorithm for a joint distribution $p(x, y)$. The algorithm starts at a random position $(x^{(0)}, y^{(0)})$ and then alternatively samples according to $p(y|x)$ and $p(x|y)$.

3.2.1 *KL-divergence and likelihood*

In the previous chapter, we considered several loss functions, each adapted to a particular problem. In the context of density estimation, we can use the Kullblack-Leibler (KL)-divergence, which is given by:

$$d_{\mathrm{KL}}(p, q) = -\sum_{\mathbf{x}} \log\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) p(\mathbf{x})$$

where the sum runs over all possible values of $\mathbf{x}$.

The KL-divergence can be used as a measure of difference between distributions but it is not symmetric (i.e. in the general case $d_{\mathrm{KL}}(p, q) \neq d_{\mathrm{KL}}(q, p)$) and does not respect the triangular inequality, therefore *it is not a distance*.

If we consider the *empirical data distribution* $p_{\mathcal{D}}$ defined by the training dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, where each sample has a probability $p_{\mathcal{D}}(\mathbf{x}_i) = \frac{1}{N}$, we can then try to fit a model p_θ to this data distribution by minimizing the KL-divergence, i.e. solving

$$\theta^* = \arg\min_\theta d_{\mathrm{KL}}(p_{\mathcal{D}}, p_\theta)$$

Note that the KL-divergence can be rewritten as

$$d_{\mathrm{KL}}(p_{\mathcal{D}}, p_\theta) = \sum_{\mathbf{x}} \log p_{\mathcal{D}}(\mathbf{x}) p_{\mathcal{D}}(\mathbf{x}) - \underbrace{\sum_{\mathbf{x}} \log p_\theta(\mathbf{x}) p_{\mathcal{D}}(\mathbf{x})}_{\text{log-likelihood}}$$

where the first term does not depend on θ and the second term is referred to as the *log-likelihood*, a concept which will be reviewed thoroughly in the following sections. From the above equation, it follows that minimizing the KL-divergence is equivalent to maximizing the log-likelihood, i.e. :

$$\theta^* = \arg\max_\theta \sum_{\mathbf{x}} \log p_\theta(\mathbf{x}) p_{\mathcal{D}}(\mathbf{x})$$

or equivalently, using the definition of $p_{\mathcal{D}}$:

$$\theta^* = \arg\max_\theta \sum_{\mathbf{x} \in \mathcal{D}} \log p_\theta(\mathbf{x})$$

The problem of density estimation can therefore be solved by minimization of the KL-divergence of equivalently, with the maximization of the log-likelihood.

3.2.2 *Bayes' rule*

In the previous section, we tried to find the best parameter θ to minimize the KL-divergence. The parametrization of the distribution by θ is noted $p_\theta(\mathbf{x})$ in the optimization perspective, however with a Bayesian perspective, θ is seen as a random variable and the model then corresponds to the probability of $\mathbf{x}$ *given*

θ, i.e. $p(\mathbf{x}|\theta)$. Bayes' rule can then be used to find the probability of parameter values θ given a dataset $\mathcal{D}$:

$$\underbrace{p(\theta|\mathcal{D})}_{\text{posterior}} = \frac{\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}\ \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{\sum_{\theta} p(\mathcal{D}|\theta)p(\theta)}_{\text{evidence}}}.$$

Likelihood $p(\mathcal{D}|\theta)$ is the likelihood of the dataset $\mathcal{D}$ under the model. It consists in the probability of the dataset $\mathcal{D}$ under a specific model parametrized by θ, i.e. the belief that the model parametrized by θ could have generated $\mathcal{D}$. If we assume that points in the dataset are iid, $p(\mathcal{D}|\theta)$ is equal to the product of the point-wise probabilities, i.e. $p(\mathcal{D}|\theta) = \prod_{\mathbf{x}\in\mathcal{D}} p(\mathbf{x}|\theta)$. With the notation of θ as a random variable, the likelihood of a single sample $p(\mathbf{x}|\theta)$ is in fact the model's probability distribution $p_\theta(\mathbf{x})$, e.g. for a D-dimensional Gaussian family, we would have

$$p(\mathbf{x}|\theta) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}\,|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

where $\boldsymbol{\mu}$ is the means parameter, and $\boldsymbol{\Sigma}$ is the $D \times D$ covariance matrix, and $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$.

Posterior $p(\theta|\mathcal{D})$ corresponds to the belief that θ is a likely parameter value of the distribution $p(x)$, given the dataset $\mathcal{D}$. When we are only interested in the best possible parameter value, maximizing the posterior leads to the most probable value of the parameter θ given the dataset $\mathcal{D}$. However, the posterior is a probability distribution and therefore gives a probability to all possible values of θ. This is especially useful to assess the variance of an estimation.

Evidence $\sum_\theta p(\mathcal{D}|\theta)p(\theta)$ is of little practical importance and can simply be seen as a normalization constant to ensure that the probabilities sum up to 1.

Prior $p(\theta)$ corresponds to the a-priori probability of θ, that is, the belief we have that θ is a reasonable parameter, *before having seen the dataset.* This can seem a bit paradoxical which is why we will return to this question shortly.

3.3 Maximum a-posteriori and maximum likelihood

When trying to learn a distribution, the goal is often to find the best possible parameter value. The problem is then to maximize the posterior distribution with respect to the parameter θ, i.e. , we are looking for θ^* such that

$$\theta^* = \arg\max_\theta p(\theta|\mathcal{D}).$$

θ^* is then called the *maximum a-posteriori* estimate because it maximizes the posterior distribution.

Note that the evidence $\sum_\theta p(\mathcal{D}|\theta)p(\theta)$ can in fact be written $p(\mathcal{D})$ (marginalization rule) and does not depend on the parameter θ. Applying Bayes' rule, the maximization problem is therefore equivalent to

$$\theta^* = \arg\max_\theta p(\mathcal{D}|\theta)p(\theta)$$

where we take the likelihood and the prior into account as expected.

In cases where there is no useful prior, the prior can be chosen to be uniform and therefore does not depend on θ, i.e. $p(\theta) = cst$. We can

further simplify the optimization problem into:

$$\theta^* = \arg\max_\theta p(\mathcal{D}|\theta).$$

θ^* is then called the *maximum likelihood estimate* and is the value of θ which maximizes the likelihood of the data under the model.

In practice it is often useful to consider maximizing the *log*-likelihood $\log p(\mathcal{D}|\theta)$ instead of the the likelihood itself[2]. The two optimization problems are equivalent because the logarithm is a monotonously increasing function[3]. Additionally, when the dataset $\mathcal{D}$ is composed of iid samples, the likelihood decomposes as a product of point-wise probabilities, as in $p(\mathcal{D}|\theta) = \prod_{\mathbf{x}\in\mathcal{D}} p(\mathbf{x}|\theta)$. The logarithm then serves to obtain a sum over the dataset, i.e the log-likelihood of the dataset is equal to the average log-likelihood on the dataset: $\log p(\mathcal{D}|\theta) = \sum_{\mathbf{x}\in\mathcal{D}} \log p(\mathbf{x}|\theta)$.

3.4 Choosing a prior

In the limit of infinitely many observations and for a prior which is non-zero everywhere, the posterior distribution tends to the likelihood itself. Conversely, when the dataset is empty, the posterior distribution is equal to the prior. In other words, maximum a-posteriori depends strongly on the prior when there are few observations and is close to maximum likelihood if there are many.

[2] Maximizing the log-likelihood was already proposed in the previous section as a way to minimize the KL-divergence with the empirical data distribution $p_\mathcal{D}$.

[3] This equivalence relies on the assumption that samples from the dataset are iid.

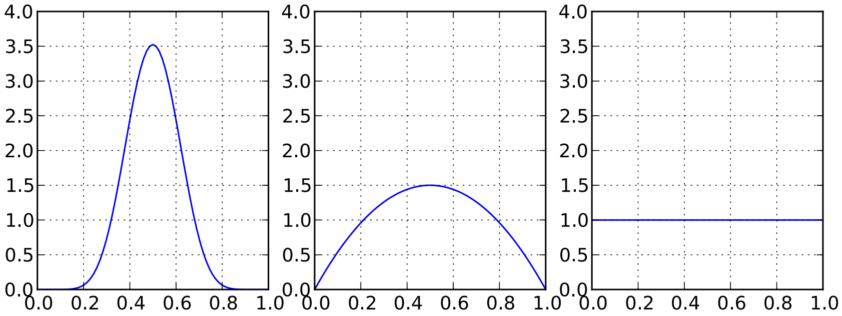


Figure 3.3: Three possible choices of prior for a Bernoulli distribution. $\mathcal{B}eta(10, 10)$ (left), $\mathcal{B}eta(2, 2)$ (middle), $\mathcal{B}eta(1, 1)$ or equivalently uniform distribution (right).

Let us consider the problem of estimating the probability of a coin toss resulting in "heads". The problem can be seen as a problem of estimating the parameter θ of a Bernoulli distribution:

$$\begin{aligned} P(\mathbf{x} = 1) &= \theta \\ P(\mathbf{x} = 0) &= 1 - \theta, \end{aligned}$$

where $\mathbf{x}$ is a random Bernoulli variable such that 1 correspond to heads and 0 to tails. In this example, a reasonable prior could be a distribution peaked around $\theta = 0.5$ decreasing towards 1 and towards 0, making explicit our belief that most coins have almost equal probability of coming heads or tails, and that it would probably be very difficult to find a coin which always falls on the same side[4]. Figure 3.3 gives three examples of priors which may be suited to this problem.

The Beta distribution in Figure 3.3 has a special relation to the Bernoulli distribution, namely it is a *conjugate prior* of the Bernoulli distribution. Conjugate priors have the interesting property of ensuring that the posterior is in the same family of distributions as the prior. If the prior is given by a Beta distribution and the likelihood is a Bernoulli distribution (as in our example) then the posterior is also a Beta distribution.

It is important to realize that the choice of a prior is a subjective one by definition. If the practitioner does not want to make this choice, or if all values of θ are in-differentiable, it is common practice to choose the uniform distribution which does not depend on the parameter θ and assigns equal probability to all possible values.

However, the uniform distribution is not a *non-informative* prior because it carries information about the structure of the parameter space. Namely, if we have $\theta \in \mathbb{R}$, a uniform prior represents the belief that there is as much probability density in the interval $]0, 1[$ than in any other interval $]z, z + 1[$, when any interval contains in fact as many real numbers as $\mathbb{R}$ itself.

[4] We assume of course that the coin in question does not have a face on both sides, as is often the case when the problem occurs in practice.

3.5 Example: Maximum likelihood for the Gaussian

Let us now, as an example, derive the maximum likelihood estimates for a D-dimensional Gaussian probability distribution. The goal is then to maximize the log-likelihood, i.e. to find

$$\boldsymbol{\mu}^* = \arg\max_{\mu} \log p(\mathcal{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

and replacing the model by its definition:

$$\boldsymbol{\mu}^* = \arg\max_{\mu} \log \prod_{\mathbf{x}\in\mathcal{D}} \frac{1}{(2\pi)^{D/2}\,|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

The problem simplifies into

$$\boldsymbol{\mu}^* = \arg\min_{\mu} \sum_{x\in D} (\mathbf{x}-\boldsymbol{\mu})^2$$

As seen in chapter 1, if the problem has a solution, it must satisfy the first order necessary condition, i.e. the first derivative of $\sum_{\mathbf{x}\in D}(\mathbf{x}-\boldsymbol{\mu})^2$ with respect to $\boldsymbol{\mu}$ must be 0 at the optimum $\boldsymbol{\mu}^*$:

$$\frac{\partial \log p(\mathcal{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \boldsymbol{\mu}} = \sum_{\mathbf{x}\in D} \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$$

and therefore

$$\sum_{\mathbf{x}\in D} \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}^*) = 0$$

which is equivalent to

$$\boldsymbol{\mu}^* = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \mathbf{x}$$

Therefore the maximum likelihood estimate for the mean parameter of a D-dimensional Gaussian is the arithmetic mean of the training samples.

A similar computation yields the maximum likelihood estimate for the covariance matrix:

$$\boldsymbol{\Sigma}^* = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}}(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu}^T)$$

3.6 Example: Probabilistic polynomial regression

In the case of polynomial regression, an interesting possibility is to consider $y = f(x)$ as a conditional probability $p(y|x)$. This density is then given by the polynomial $\sum_{i=0}^{k} a_i x^i$ to which we add a normally distributed error term $\mathcal{N}(0, \sigma^2)$:

$$\begin{aligned} p(y|x) &= \sum_{i=0}^{k} a_i x^i + \mathcal{N}(0, \sigma^2) \\ &= \mathcal{N}(\sum_{i=0}^{k} a_i x^i, \sigma^2). \end{aligned}$$

The model can be seen as the probability of measuring some value y given x. For each x, the measurement y is distributed according to a normal law of mean $\sum_{i=0}^{k} a_i x^i$ and of variance σ^2.

In this example, the quantity to maximize can be computed analytically by replacing $p(y|x)$ by the definition of a normal law.

$$\begin{aligned} \log p(\mathcal{D}) &= \sum_{i=1}^{N} \log p(y_i|x_i) \\ &= \sum_{i=1}^{N} \log \left[\frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(y_i - \sum_{i=0}^{k} a_i x^i)^2}{2\sigma^2}) \right] \\ &= N \log \left(\frac{1}{\sigma\sqrt{2\pi}} \right) - \sum_{i=1}^{N} \frac{(y_i - \sum_{i=0}^{k} a_i x^i)^2}{2\sigma^2}. \end{aligned}$$

Adding and multiplying by the appropriate constant terms (here we consider σ^2 to be a constant, the problem being on μ) which do not change the maximization problem, and taking into account the minus sign which transforms the maximization problem into a minimization problem, the quantity to minimize is

$$\sum_{i=1}^{N} \left[y_i - \sum_{i=0}^{k} a_i x^i \right]^2 = \sum_{i=1}^{N} \left[y_i - \hat{f}(x_i) \right]^2$$

which is exactly the MSE seen in Chapter 2. Therefore, minimizing the mean squared error when fitting polynomials consists in learning the above probabilistic model.

3.7 Latent variables and Expectation Maximization

The probabilistic models seen so far can give an interesting interpretation of a dataset, but we have not yet discussed how to leverage these algorithms to find new representations. This can be done in the context of estimating distributions with the help of *latent variables* (Ghahramani, 2004). Latent variables are sometimes called hidden variables or unobserved variables by opposition to the observed variables $\mathbf{x} = x_1, \ldots, x_D$.

A simple way to introduce latent variables in a probability distribution is to use a joint distribution over both observed and hidden variables which can then be marginalized over the hidden variables, as in

$$p(\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h}),$$

where the sum over $\mathbf{h}$ is to be understood as a sum over all possible values of $\mathbf{h}$. As expected, the resulting distribution is on the observed variable $\mathbf{x}$ alone.

When the model is trained, the latent variables can be seen as representing explanatory variables which best help to model the distribution on the observed variables.

A problem arises when trying to learn the joint distribution $p(\mathbf{x}, \mathbf{h})$ because by definition, the dataset $\mathcal{D}$ only contains samples from the observed variable $\mathbf{x}$, and not samples $\mathbf{x}, \mathbf{h}$ as would be required to learn a joint distribution $p(\mathbf{x}|\mathbf{h})$. As a result, it is often impossible to find a closed-form solution to the maximum likelihood of models with latent variables.

Nonetheless, latent variable models can usually be trained with a variant of the Expectation Maximization (EM) algorithm (Dempster et al., 1977; Borman, 2004) which alternates between two steps[5]:

- (*expectation*) Compute the inference distribution $p(\mathbf{h}|\mathbf{x})$; construct samples $\mathbf{x}, \mathbf{h}$ with $\mathbf{h}$ the most probable value of $\mathbf{h}$ according to $p(\mathbf{h}|\mathbf{x})$.
- (*maximization*) maximize the likelihood of $p(\mathbf{x}, \mathbf{h})$ given the samples $\mathbf{x}, \mathbf{h}$ obtained previously.

Although it is not easy to derive this algorithm, it can be shown to converge to a local minimum (Dempster et al., 1977; Wu, 1983).

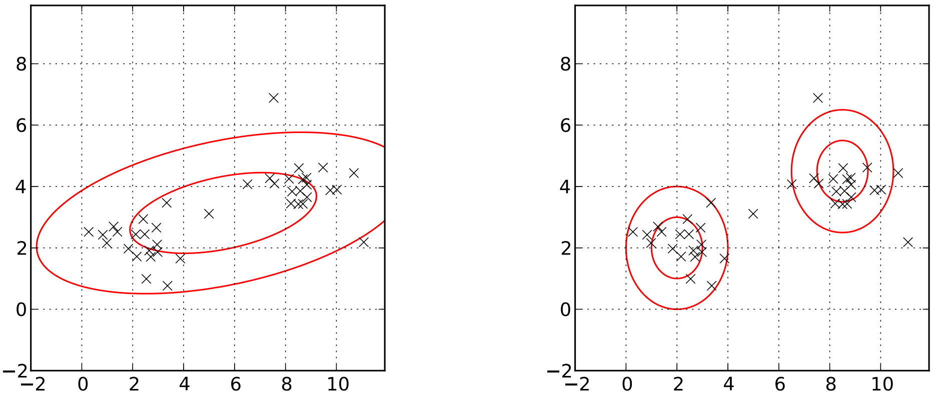
Note than EM is not the only way to train models with latent variables and it is sometimes preferable to use an other optimization algorithm such as gradient descent.

3.8 Example: Gaussian mixtures and EM

Sometimes the Gaussian distribution is not complex enough to accurately represent the input distribution. Figure 3.4(a) gives an example where the maximum likelihood estimate for a single Gaussian in 2 dimensions does not accurately capture the structure of the dataset.

However, if we combine K Gaussians such that each point from the dataset is sampled from one such Gaussian, the model becomes much more expressive as in Figure 3.4(b). In fact, given enough Gaussians, it becomes possible to approximate any distribution with arbitrary accuracy. This can be done with

[5] In fact, we present here a point-estimate variant of EM called classification EM. See (Gupta and Chen, 2011) for a complete description of EM.



(a) Best fit with a bivariate Gaussian.

(b) Best fit with a Gaussian mixture.

Figure 3.4: The Gaussian distribution being unimodal, a single Gaussian is unable to capture the structure of this dataset. Using a mixture of Gaussians allows for a better fit. The red lines give the points at 1 and 2 standard deviations from the mean of each Gaussian.

the introduction of a hidden variable $\mathbf{h}$ to represent the choice of a Gaussian. The model is then a latent variable model, i.e.

$$p(\mathbf{x}|\theta) = \sum_{\mathbf{h}} p(\mathbf{x}|\mathbf{h}, \theta)p(\mathbf{h}|\theta)$$

where $\mathbf{h}$ is the random variable associated with the random choice of a Gaussian among the K possible choices. In this example, we choose the *one-hot representation* for the variable $\mathbf{h}$, i.e. all possible values of the vector $\mathbf{h}$ are such that exactly one component is 1 and all the others are 0. We note $\mathbf{h}_i$ the value of $\mathbf{h}$ corresponding to the choice of the i^{th} Gaussian. For instance, for $K = 3$, the possible values are $\mathbf{h}_1 = [1, 0, 0]$, $\mathbf{h}_2 = [0, 1, 0]$, $\mathbf{h}_3 = [0, 0, 1]$.

The probability of choosing the i^{th} Gaussian is given by $p(\mathbf{h} = \mathbf{h}_i|\theta) = \pi_i$, where the vector $\boldsymbol{\pi}$ is called the *mixing parameter*. The parameters π_i are such that their sum is 1 to ensure that the probability distribution stays normalized.

Once the Gaussian i is chosen, the probability of sampling a vector $\mathbf{x}$ is simply given by

$$p(\mathbf{x}|\mathbf{h} = \mathbf{h}_i, \theta) = \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

where $\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i$ are the mean and covariance matrix of the i^{th} Gaussian distribution.

Recomposing the full probability distribution, we have

$$p(\mathbf{x}|\theta) = \sum_{i=1}^{K} \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\pi_i$$

where the sum over the index i covers all possible Gaussian affectations. The latent variable $\mathbf{h}$ is no longer visible in the above equation but is implicitly marginalized over with the index i. The parameter θ regroups all the parameters of the model, i.e. $\theta = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K, \pi_1, \ldots, \pi_K\}$.

In the case of Gaussian mixtures, we cannot find a closed-form solution to the maximum likelihood problem. However, we can use the EM algorithm introduced above to find a local maximum of the likelihood. For the first step of EM, we must find $p(\mathbf{h}|\mathbf{x})$ to infer the hidden variables from a data instance $\mathbf{x}$. This conditional probability is given by Bayes' rule, namely

$$\begin{aligned} p(\mathbf{h} = \mathbf{h_i}|\mathbf{x}, \theta) &= \frac{p(\mathbf{x}|\mathbf{h} = \mathbf{h}_i, \theta) p(\mathbf{h} = \mathbf{h_i}|\theta)}{p(\mathbf{x}|\theta)} \\ &= \frac{\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\pi_i}{\sum_{j=1}^{K} \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\pi_j} \end{aligned}$$

This implies a sum over all Gaussians in the denominator and as such can be costly when a large number of Gaussians are involved. Although the EM algorithm can be run with arbitrary mixing parameters π_i and covariance matrices $\boldsymbol{\Sigma}_i$, the case where all π_i are equal and where the covariance matrices are a multiple of the identity matrix, i.e. are of the form $\boldsymbol{\Sigma}_i = \sigma I$ with σ a constant common to all Gaussians, has an interesting interpretation. In this case, the most probable value of $\mathbf{h}$ for an observation $\mathbf{x}$ is $\mathbf{h}_i$ if $\boldsymbol{\mu}_i$ is the mean closest to $\mathbf{x}$.

For the second step of EM, i.e. the maximization of the likelihood of $p(\mathbf{x}, \mathbf{h} = \mathbf{h}_i)$ w.r.t. the mean parameters $\boldsymbol{\mu}_i$, it simply consists in maximizing the likelihood of $\mathbf{x}$ under the i^{th} Gaussian. The maximum likelihood for the Gaussian is derived in Section 3.5 above and is given by

$$\boldsymbol{\mu}_i^* = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x},\mathbf{h}} \mathbf{x}.$$

In the special case of equal mixing parameters and with isotropic Gaussians, the two steps above correspond exactly to the K-means algorithm where a centroid $\mathbf{c}_i$ corresponds to the mean of the i^{th} Gaussian $\boldsymbol{\mu}_i$. The K-means algorithm has therefore a probabilistic interpretation as the application of the EM algorithm to a mixture of equiprobable Gaussians of equal isotropic variance.

Although EM can recover the K-means algorithm under specific conditions, EM is much more general and can be used with arbitrary mixing parameters and covariance matrices.

3.9 Optimization revisited in the context of maximum likelihood

We have now reviewed how a learning problem results from solving an optimization one, sometimes with a probabilistic interpretation. We now come back to optimization with a special focus on the importance of choosing a metric.

An efficient optimization method must concentrate the search in regions of the search space which have a higher chance of containing an optimum. This is often done with an iterative approach in which the goal is to make at each step a small movement from the current position toward better values.

This kind of method depends on a metric to define neighborhoods in the search space in which the objective function assumed to have small variations. Unfortunately, practitioners often choose the canonical metric i.e. the Euclidean distance in $\mathbb{R}^d$ or the Hamming distance in $\{0,1\}^d$ regardless of its suitability to the problem under consideration.

We start by presenting how the use of a canonical metric in the context of log-likelihood maximization with gradient descent leads to an undesirable dependence on parametrization and then, we present the natural gradient which is simply the ordinary gradient in the Fisher metric and is specially adapted for moving in the space of probability distributions.

3.9.1 *Gradient dependence on metrics and parametrization*

In the previous chapters, we have often proposed the gradient descent algorithm to maximize the log-likelihood w.r.t the parameters of a model. However, gradient descent can yield different trajectories depending on the parametrization of a model. More precisely, the gradient descent procedure is often considered in parameter space, usually with the Euclidean metric.

Consider the Gaussian family in one dimension with the usual parametrization, i.e.

$$p_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

where the parameters are the mean μ and the variance σ^2. The matter is especially confusing with the Gaussian family because the variance is noted as the standard deviation squared. Should we therefore take the derivative w.r.t. σ^2 or w.r.t. σ, and more importantly are the two alternatives equivalent ? They are not:

$$\begin{aligned}
\frac{\partial \log p_{\mu,\sigma}(x)}{\partial(\sigma^2)} &= \frac{\mu^2 - 2\mu x - \sigma^2 + x^2}{2\sigma^4} \\
\frac{\partial \log p_{\mu,\sigma}(x)}{\sigma} &= \frac{\mu^2 - 2\mu x - \sigma^2 + x^2}{\sigma^3}
\end{aligned}$$

with the partial derivative w.r.t. the parameter μ remaining unchanged.

This can be attributed to the Euclidean metric which is implicitly used in these computations. The gradient gives the direction of greatest increase for an infinitesimal movement $\delta\theta$ in parameter space such that $\|\delta\theta\| < \epsilon$. When we consider the parameters μ, σ^2, the distance $\|\delta\theta\| = \sqrt{(\delta\mu)^2 + (\delta\sigma^2)^2}$, is different than the one obtained by considering the parameters μ, σ, i.e. $\|\delta\theta\| =$

$\sqrt{(\delta\mu)^2 + (\delta\sigma)^2}$. Consequently, the two gradients give the direction of greatest increase in log-likelihood but allow for movements of different amplitude in parameter space.

Although this problem of choosing a parametrization is especially clear with the choice of σ or σ^2 as variance parameter, it is important to realize that the Euclidean metric introduces a spurious connection between the amplitude of gradient steps along different parameters. When we generalize to multi-dimensional Gaussians, these spurious connections typically favor Gaussians which are close to isotropic because the Euclidean metric gives an equal importance to every parameter of the covariance matrix.

In fact, every metric defined by a constant matrix in parameter space will favor one kind of Gaussian over another.

3.9.2 *The natural gradient*

A consequence of the previous section is that there are infinitely many possible gradients of the log-likelihood, each one corresponding to a different choice of metric. The natural gradient (Amari, 1998; Amari et al., 2000) is defined as the gradient in the Fisher information metric which is an approximation of the KL-divergence. As it relies on the KL-divergence, which is a common measure of difference between distributions, the natural gradient can be seen as moving on a manifold of probability distributions which is independent of any parametrization. Although the KL-divergence is not a metric, for infinitesimal movements $\delta\theta$ around θ, the KL-divergence $d_{KL}(P_\theta, P_{\theta+\delta\theta})$ can be approximated by its Hessian: the Fisher information metric. Because $\delta\theta = 0$ corresponds to the global minimum of the KL-divergence, the Hessian corresponds to a second order approximation. The Fisher metric is defined by the so-called Fisher matrix:

$$F_{ij} \quad = \quad \mathbb{E}\left[\frac{\partial \log p_\theta(\mathbf{x})}{\partial \theta_i}\frac{\partial \log p_\theta(\mathbf{x})}{\partial \theta_j}|\theta\right]$$

Formally, the expression of a gradient ∇_A in some arbitrary metric A, for A a symmetric positive definite matrix, is given by

$$\nabla_A = A^{-1}\nabla$$

The natural gradient $\tilde{\nabla}$ is then simply the gradient in the Fisher metric F, i.e.:

$$\tilde{\nabla} \log p_\theta(\mathbf{x}) = F^{-1}\nabla \log p_\theta(\mathbf{x})$$

From this definition, the natural gradient corresponds to an infinitesimal movement in the space of distributions as opposed to a movement in parameter space. This makes the natural gradient *invariant to parametrization* because the metric measures a distance between the distributions themselves, independently of parameters. As a side effect, the invariance to parametrization often results in further invariances, depending on the probability distribution family, such as invariances w.r.t. rescaling, translation and rotation of $\mathbf{x}$ in the case of the multi-dimensional Gaussian.

Interestingly, the step size or learning rate of a natural gradient update is a quantity in bits or nats and therefore measures a quantity of information. The learning rate can then be seen as measuring how much information each step should provide.

Although the natural gradient has many theoretical advantages, it has a computational disadvantage, at least with a naive implementation, because it requires computing and inverting a matrix of size $(\dim \theta)^2$ at each step.

The natural gradient gives a good example of the influence of choosing a suitable metric in the context of maximizing the likelihood of a model. It is also a reminder of the intricate link between learning and optimization, here in the case of an optimization procedure which is specially designed for learning distributions.

Summary

- From a Bayesian probabilistic perspective, it is natural to update our beliefs with data.
- It is sometimes necessary to use methods such as rejection sampling, importance sampling, the Metropolis-Hastings algorithm or Gibbs sampling to sample from complex distributions.
- Probabilistic models can be trained by minimizing the KL-divergence between the empirical data distribution and the model distribution.
- Equivalently, a probabilistic model can be trained by maximizing the log-likelihood of a dataset under the model.
- Bayes' formula gives a method for choosing the best parameters given data: maximum-a-posteriori.
- The prior distribution gives probabilities to model parameters before having seen a dataset.
- When the prior is considered uniform, maximum-a-posteriori is equivalent to maximum-likelihood.
- Probabilistic models can have latent variables which can be understood as unobserved explanatory factors.
- Models with latent variables can be trained with the EM algorithm which alternates between computing the expected latent variables given the current maximum likelihood estimate, and maximizing the log-likelihood given affectations of the latent variables.
- Training Gaussian mixtures with EM can be seen as a probabilistic generalization of the K-means clustering algorithm.
- The log-likelihood gradient in the Euclidean metric is affected by parametrization
- The natural gradient based on the Fisher metric is invariant by re-parametrization and can introduce further invariances during optimization.

This chapter concludes our presentation of ML. We now turn to deep learning, the main topic of this thesis.

Part II

DEEP LEARNING

4

ARTIFICIAL NEURAL NETWORKS

Although deep learning could in theory apply to any kind of model, almost all approaches to deep learning so far are based on artificial neural networks.

Artificial neural networks (Bishop, 1995) regroup a large variety of models which use neurons as their elementary computation unit. Although this class of models was historically inspired from biological processes, it is now a integral part of the mathematical ML framework. Neural networks can be used in supervised settings for classification of regression, in unsupervised settings for dimensionality reduction and learning representations, and can be interpreted in a probabilistic perspective.

We start by presenting the artificial neuron which is the basis of all artificial neural architectures. From there, we discuss how neurons can be structured in networks to perform complex computations especially in the contexts of classification, regression and dimensionality reduction. Finally, we present probabilistic interpretations of neural networks and describe how they can be used to estimate distributions.

4.1 The artificial neuron

4.1.1 *Biological inspiration*

Biological systems are capable of performing very complex computations to survive in their environment, find food or escape predators. These complex behaviors are controlled by a nervous system composed of nerve cells or *neurons*. One of the most remarkable properties of such systems is their scalability from just a few hundred neurons (302 neurons for the roundworm Caenorhabditis elegans) to billions of neurons (around 85 billion in the human brain). To be more precise, it seems that neurons can be combined so that an increase in the number of neurons leads to an increase in cognitive abilities (Herculano-Houzel, 2009).

Although the ways in which complex behavior can emerge from large numbers of neurons is still poorly understood, the equations governing the excitability of a single neuron are very well understood, as for instance with the Hodgkin-Huxley model (Hodgkin and Huxley, 1952).

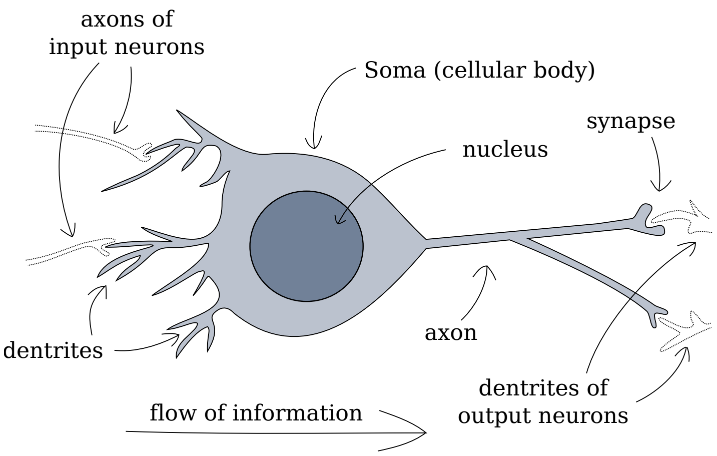


Figure 4.1: The structure of a biological neuron. Information comes from input neurons in the form of action potentials. If the neuron receives enough action potentials from its pre-synaptic neurons, it *fires a spike*, sending an action potential through its axon to the post-synaptic neurons.

Figure 4.1 describes the general architecture of a biological neuron. A neuron reacts to inputs as follows: when the *dendrites* of a neuron receive some excitatory (resp. inhibitory) input, the membrane potential of the neuron increases (resp. decreases) gradually. If the membrane voltage reaches a specific threshold, an *action potential* is initiated and propagated along its axon to post-synaptic neurons. In the rest of this paper, we will forget this biological inspiration and the term neuron will be used to refer to artificial neurons which we describe now.

4.1.2 *The artificial neuron model*

In order to transmit information, artificial neurons have an *activation* value[1].

For the purpose of computation, an artificial neuron (see Figure 4.2) has weighted connections to a set of *input neurons*. The input neurons can be seen as a vector $\mathbf{x} = x_1, x_2, \dots, x_D$, where $\mathbf{x}$ is a vector of dimension D, and x_i corresponds to the activation of the i^{th} input neuron. The activation value y of a neuron can be computed given the input activations x_i and the connection weights w_i according to

$$y = \phi(b + \sum_i w_i x_i)$$

where ϕ is called the *activation function* and b is called the *bias* of the neuron y. The term $b + \sum_i w_i x_i$ taken as input of ϕ is called the *pre-activation*. The bias can be considered as a regular weight w_0 connected to an input neuron x_0 such that $x_0 = 1$. Consequently, a neuron performs a scalar product between the extended

[1] This activation value can be seen as playing a role similar to the firing frequency of a biological neuron.

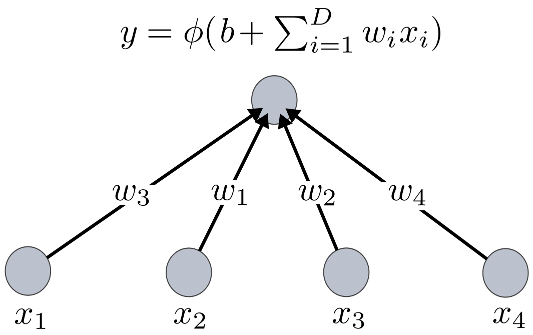


Figure 4.2: Computational properties of an artificial neuron. The activation of a neuron is computed as a weighted sum of the activations of the input neurons, transformed by an activation function ϕ. The weights of the connections determine how much influence an input neuron has on the output neuron.

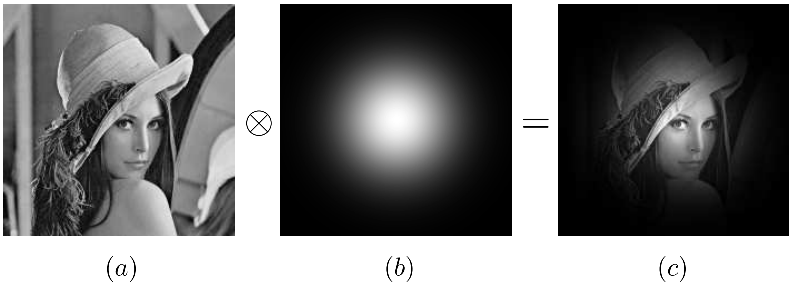


Figure 4.3: The result of filtering an input image with a weight vector. The image (c) is the element-wise product of (a) and (b). If the pixel intensities of (a) and (b) are elements of the input vector $\mathbf{x}$ and the weight vector $\mathbf{w}$ respectively, then the average intensity of (c) is the pre-activation $\mathbf{w}^\top \mathbf{x}$. When a weight w_i is near 0 as in the black region of (b), the corresponding input x_i is filtered out and does not influence the final result.

input vector $\mathbf{x}$ containing $x_0 = 1$ and the weight vector $\mathbf{w} = w_0, w_1, w_2, \ldots, w_D$, and transforms the result according to the activation function ϕ, i.e.

$$y = \phi(\mathbf{w}^\top \mathbf{x})$$

4.1.3 *A visual representation for images*

The scalar product of the weights and the input can be interpreted as a projection of the input vector on the weight vector. In particular, if $\mathbf{x}$ represents the pixels of an image, it is common to represent the weight vector as a *visual filter*. Figure 4.3 gives an example of filtering operation which can be realized by a single neuron.

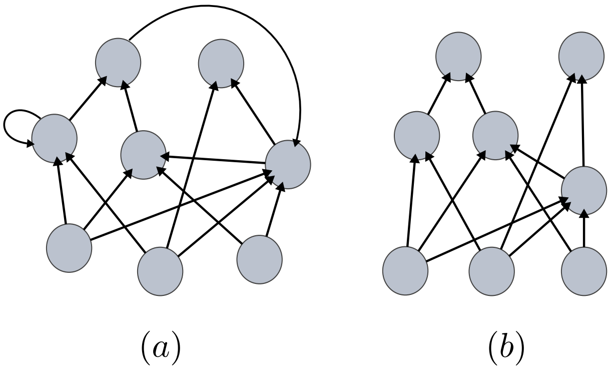


Figure 4.4: Different topologies of neural networks: a recurrent neural network (a) and a feed-forward neural network (b).

4.2 Feed-forward neural networks

Artificial neurons can only make a simple computation by themselves. However, they can be arranged in *neural networks* to perform more complex operations. These networks can then be used to compute the activations of a set of output neurons $\mathbf{y} = (y_1, y_2, \dots, y_{D'})$ given the activations of input neurons $\mathbf{x} = (x_1, x_2, \dots, x_D)$. The computation usually involves a set of *hidden neurons* $\mathbf{h}$ which perform intermediary computations.

Although neurons can in theory be arranged quite arbitrarily, in practice they are often arranged in an acyclic graph which means that the input of a neuron does not depend on its output, even indirectly. Neural networks organized with such a topology are referred to as *feed-forward neural networks* because the activations can be propagated forward in the network. By contrast, *recurrent neural networks* can contain cyclic connections. Recurrent neural networks are potentially better at modeling dynamical systems, but the presence of cycles makes training much more difficult. Figure 4.4 gives examples for a general recurrent neural network and a feed-forward network.

Let us now consider feed-forward networks where the neurons are organized in *layers*. In this terminology, the input vector $\mathbf{x}$ is the *input layer* ℓ_0. Subsequent layers ℓ_k then regroup neurons which only receive input from neurons in the previous layer ℓ_{k-1}. This makes possible the computation of neural activations in a feed-forward, layer-wise manner. The weights used to compute the activation of a layer ℓ_k from the activations of a layer ℓ_{k-1} then form a matrix $\mathbf{W}$ where w_{ij} gives the connection weight from the i^{th} neuron of layer ℓ_{k-1} to the j^{th} neuron of layer ℓ_k. For two subsequent layers $\mathbf{x}$ and $\mathbf{y}$ the computation rule becomes

$$\forall j, y_j = \phi(b_j + \sum_i w_{ij} x_i),$$

or using matrix notation

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

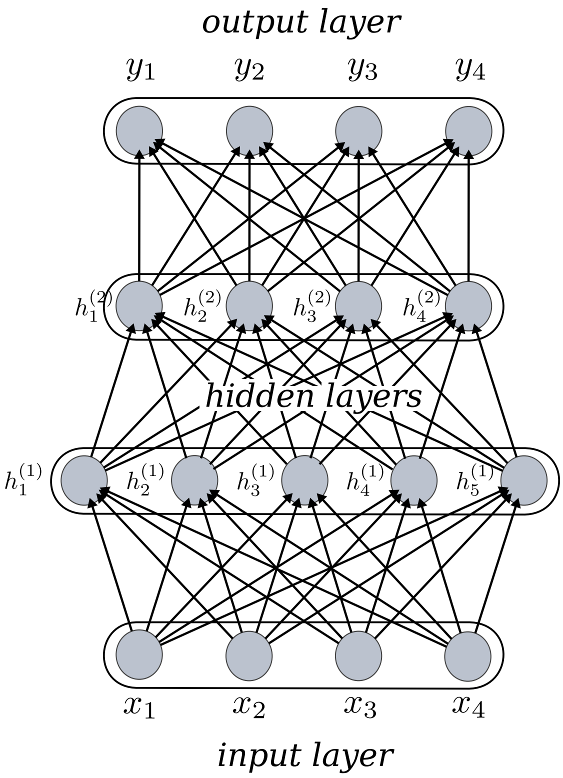


Figure 4.5: A multi-layer neural network. Activations can be propagated layer by layer from the input layer $\mathbf{x}$ to the output layer $\mathbf{y}$.

where the function ϕ is simply the element wise application of the activation function.

When a neural network involves more layers than the input and output layers, the network is called a *multi-layer neural network*. The hidden neurons are then arranged in several hidden layers as shown in Figure 4.5.

The usual notations employed for neural networks can sometimes be confusing. For instance, the activations of the output neurons are usually noted $\mathbf{y}$, whereas in the context of supervised learning these activations would be noted $f(\mathbf{x})$, $\mathbf{y}$ referring to the label. For instance a neural network with two layers would compute $\mathbf{y} = f(\mathbf{x}) = \phi(\mathbf{W}_2\phi(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$. Additionally, the pre-activation of neurons is usually denoted by a variable a, and is not to be confused with the activation itself $\phi(a)$. Finally, the activation functions are usually given as functions of a variable x which is not to be understood as the input but as a general variable in $\mathbb{R}$.

4.3 Activation functions

Until now we have considered an undefined activation ϕ. However, there are several common choices of activation function ϕ, each suited to different applications.

LINEAR ACTIVATION A first possibility is to use the identity function $\phi(x) = x$ as activation function. A layer then performs a linear operation $\mathbf{W}\mathbf{x} + \mathbf{b}$ on the input vector $\mathbf{x}$. When several such layers are combined, the combination of linear operations being linear, the model still reduces to a single linear operation. In the case of two layers with parameters $\mathbf{W}_1, \mathbf{b}_1$ and $\mathbf{W}_2, \mathbf{b}_2$, the output of the networks can be computed as

$$\mathbf{y} = \mathbf{W}_2(\underbrace{\mathbf{W}_1\mathbf{x} + \mathbf{b}_1}_{1^{\text{st}}\text{layer output}}) + \mathbf{b}_2$$

which is equivalent to a single layer of parameters $\mathbf{W}_r, \mathbf{b}_r$:

$$\mathbf{y} = \underbrace{\mathbf{W}_2\mathbf{W}_1}_{\mathbf{W}_r}\mathbf{x} + \underbrace{\mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2}_{\mathbf{b}_r}$$

Adding layers then has the disadvantage of adding parameters, thus making optimization more difficult, while not increasing the expressivity of the model. However, a linear activation function can be useful for the last layer of a multi-layer neural network when the output is a real variable, as e.g. in regression problems. To ensure that the multi-layer network does not reduce to a single layer, a non-linear activation function (a.k.a. a *non-linearity*) is then used in the other layers of the network.

HEAVISIDE STEP FUNCTION The Heaviside step function is defined as follows

$$h(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

In a neural network, the result is then 1 or 0 depending on the sign of the pre-activation $\mathbf{W}\mathbf{x} + \mathbf{b}$. This corresponds to a partition of the input space $\mathbb{R}^D$ according to a separating hyperplane: a neuron y has value 0 if $\mathbf{x}$ is on one side of the hyperplane, 1 if it is on the other side. This makes the step function adapted to linearly separable classification problems (Minsky and Papert, 1969). However it is now rarely used because it is not differentiable which makes optimization difficult. A differentiable sigmoid function is often preferred to circumvent this issue.

SIGMOID ACTIVATION The class of sigmoid functions is the most common choice of activation function. Formally a sigmoid function is an “s”-shaped function, such as the hyperbolic tangent or the logistic function[2] $\mathrm{sigm}(x) = \frac{1}{1+\exp(-x)}$. Logistic functions can be seen as continuously differentiable approximations of the Heaviside step function. As such they also create a linear separation surface in the input space. The expressivity of the model increases when several layers are considered because sigmoid functions are non-linear. Neural networks with only one sigmoidal hidden layer are in fact universal approximators (Cybenko, 1989; Hornik et al., 1989), i.e. they can represent any function given a sufficient number of neurons. The logistic function takes values in the interval $[0, 1]$ and is therefore often interpreted as the activation probability of a binary neuron. Formally the model for a single layer is then given by

$$P(\mathbf{y}|\mathbf{x}) = \prod_j P(y_j|\mathbf{x}) = \prod_j \mathrm{sigm}(b_j + \sum_i w_{ij}x_i)$$

SOFTMAX ACTIVATION In the context of classification, the softmax activation function is often preferred to sigmoid functions for the output layer. It can be interpreted as a smooth maximum of several activations. Contrary to the other activation functions presented above, the softmax activation function is not applied to each neuron of a layer individually, but rather is applied to the whole layer at once:

$$\mathrm{softmax}(\mathbf{x}) = \left(\frac{\exp(x_1)}{\sum_{i=1}^{D_z} \exp(x_i)}, \ldots, \frac{\exp(x_D)}{\sum_{i=1}^{D_z} \exp(x_i)} \right)$$

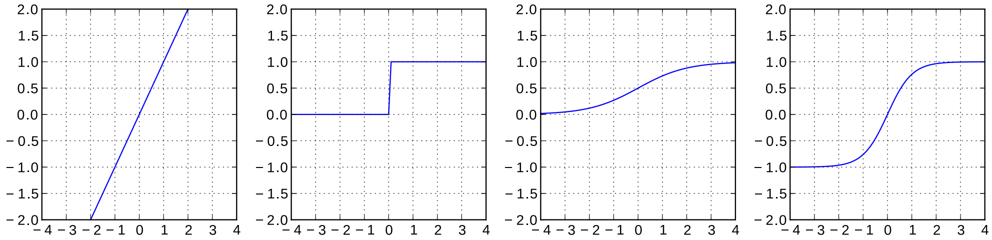
The softmax function can also be interpreted as giving the probabilities of activation of each neuron according to a categorical distribution, i.e. $P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(\mathbf{Wx} + \mathbf{b})$. Each component then gives the probability of a class being selected. The denominator of the above equation then ensures that the probabilities sum up to 1.

The step function, hyperbolic tangent and logistic functions are represented in Figure 4.6.

4.4 TRAINING WITH BACK-PROPAGATION

In a supervised setting, the training of feed-forward neural networks is usually done with a form of gradient descent. This requires the computation of partial derivatives of the error function w.r.t. the weights. In a feed-forward network, these partial derivatives can be computed with the so-called back-propagation algorithm (Rumelhart et al., 1986; LeCun et al., 1998b). In this section, we

[2] A common abuse of language consists in referring to the logistic function as “the sigmoid function”. This explains the mathematical notation $\mathrm{sigm}(z)$ commonly used for the logistic function.



(a) (b) (c) (d)

Figure 4.6: Three common activation functions: linear activation (a), Heaviside step function (b), logistic function (c) and hyperbolic tangent (d).

consider a dataset $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_N, \mathbf{t}_N)\}$ so that outputs of the neural network $\mathbf{y}$ are not confused with the target variable.

Back-propagation can be applied when both the error functions and the activation functions are differentiable. It is then possible to use the chain rule to obtain:

$$\frac{\partial E(\mathbf{x})}{\partial w_{ij}} = \frac{\partial E(\mathbf{x})}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}$$

where $a_j = b_j + \sum_i w_{ij} z_i$ is the pre-activation of an output neuron as a function of activations z_i of neurons in the previous layer. The error function E is a per-sample error which takes the model into account. For instance if we use the MSE, $E(\mathbf{x}) = (\mathbf{t} - \mathbf{y})^2$ with $\mathbf{t}$ the target and $\mathbf{y} = f(\mathbf{x})$ is the output of the neural network.

It is common to note $\delta_j = \frac{\partial E(\mathbf{x})}{\partial a_j}$. Additionally, $\frac{\partial a_j}{\partial w_{ij}} = z_i$ and therefore

$$\frac{\partial E(\mathbf{x})}{\partial w_{ij}} = \delta_j z_i$$

Thus the derivative w.r.t. w_{ij} is the product of the activation of the neuron z_i with δ_j. For the output layer the terms δ_j^{output} can be computed by another application of the chain rule, i.e.

$$\delta_j^{\text{output}} = \frac{\partial E(\mathbf{x})}{\partial y_j} \frac{\partial y_j}{\partial a_j} = \frac{\partial E(\mathbf{x})}{\partial y_j} \phi'(a_j)$$

In the case of the MSE, we have $\frac{\partial E(\mathbf{x})}{\partial y_j} = y_j - t_j$ and thus $\delta_j^{\text{output}} = (y_j - t_j)\phi'(a_j)$.

For the hidden layers, the errors δ_j can be propagated backwards recursively through the network, i.e. for two layers ℓ_m and ℓ_{m+1}:

$$\delta_j^m = \frac{\partial E(\mathbf{x})}{\partial a_j} = \sum_k \frac{\partial E(\mathbf{x})}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

where we sum over all units k which have unit j as input. This leads to the back-propagation rule

$$\delta_j^m = \phi'(a_j) \sum_k w_{kj} \delta_k^{m+1}$$

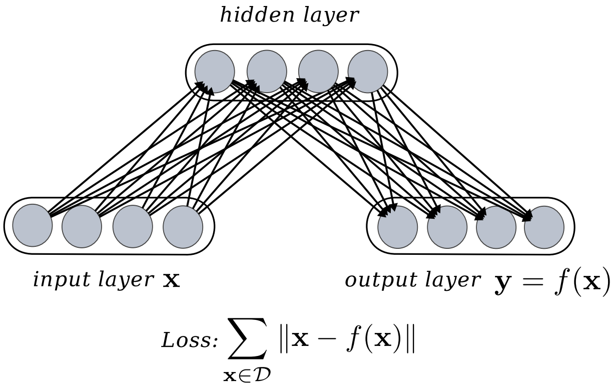

hidden layer
input layer $\mathbf{x}$
output layer $\mathbf{y} = f(\mathbf{x})$
Loss: $\sum_{\mathbf{x} \in \mathcal{D}} \|\mathbf{x} - f(\mathbf{x})\|$

Figure 4.7: The structure of an auto-encoder. The target output $\mathbf{y}$ is the input itself

This rule can then be applied iteratively to lower hidden layers. The resulting partial derivatives can then be used as part of a gradient procedure to minimize the error over a dataset.

The back-propagation algorithm has a complexity in $\mathcal{O}(\dim \theta)$ where θ regroups all parameters in the model and is therefore very efficient.

4.5 Auto-encoders

The neural networks seen so far apply mostly to supervised learning. However, if we are to study deep learning, we need a way to learn representations in an unsupervised way. A particularly interesting application of neural networks for the unsupervised setting is the possibility to perform dimensionality reduction, compression or learn new representations with an auto-encoder (Bourlard and Kamp, 1988; Hinton, 1989).

Formally, an auto-encoder is simply a feed-forward neural network trained to reproduce its input with back-propagation. The unsupervised dataset $\mathcal{D} = \{\mathbf{x}_1, \dots \mathbf{x}_N\}$ is then seen as a supervised dataset $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{x}_1), \dots (\mathbf{x}_N, \mathbf{x}_N)\}$ where the target is the input itself. The structure of an auto-encoder is represented in Figure 4.7.

Although training a feed-forward network to reproduce its input may seem pointless, the hidden layer $\mathbf{h}$ can be seen as a new representation of the data. The first layer of the auto-encoder then consists in an *encoder* which transforms $\mathbf{x}$ in the corresponding representation $\mathbf{h}$, and the second layer is effectively a *decoder* which is trained to recover an approximation of the original value $\mathbf{x}$ from the hidden representation $\mathbf{h}$.

In the general case, if the hidden layer is at least as large as the input and if there is no additional restriction, an auto-encoder will tend to learn the identity for both layers which does not lead to an interesting hidden representation. However, if the hidden layer is smaller than the input, an auto-encoder will try to learn a compressed representation in the hidden layer $\mathbf{h}$. The first layer is

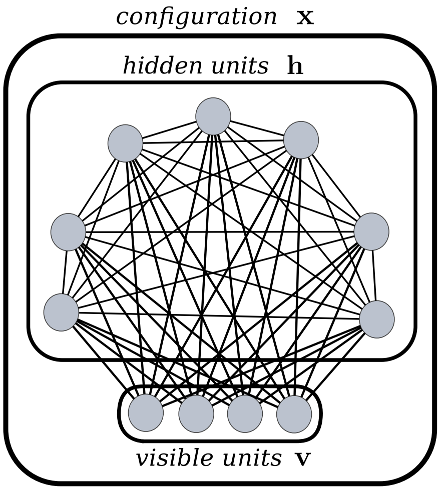

configuration $\mathbf{x}$
hidden units $\mathbf{h}$
visible units $\mathbf{v}$

Figure 4.8: The Boltzmann machine architecture with visible units ($\mathbf{v}$), hidden units ($\mathbf{h}$) and the joint configuration $\mathbf{x} = \mathbf{v}, \mathbf{h}$.

then responsible for compression of the data, and the second layer is responsible for reconstructing the input given the compressed data. To make sure that the auto-encoder does not learn the identity, it is also possible to constrain the hidden representation to be *sparse* (Ng, 2011), to use a *denoising criterion* (Vincent et al., 2008) or to use a *contractive criterion* (Rifai et al., 2011). Training auto-encoders with these restrictions been shown to be very effective as a preprocessing step to learn features in the context of representation learning.

4.6 Boltzmann Machines

Another way to learn representations with neural networks is to use probabilistic neural networks such as Boltzmann machines (Ackley et al., 1985). Boltzmann machines are undirected neural networks where each neuron is referred to as a unit x_i which can be active, i.e. equal to 1 or inactive, i.e. equal to 0. Note that the network is not feed-forward but is completely connected in the general case as in Figure 4.8.

The Boltzmann machine defines a probability density over all configurations $\mathbf{x} = x_1, \ldots, x_D$ as:

$$P_\theta(\mathbf{x}) \quad = \quad \frac{e^{-E_\theta(\mathbf{x})}}{\sum_{\tilde{\mathbf{x}} \in \{0,1\}^D} e^{-E_\theta(\tilde{\mathbf{x}})}}$$

where the function $E_\theta(\mathbf{x})$ is known as the *energy function*, given by

$$E_\theta(\mathbf{x}) = -\sum_{i=1}^{D} a_i x_i - \sum_{i=1}^{D} \sum_{j=1}^{D} w_{ij} x_i x_j$$

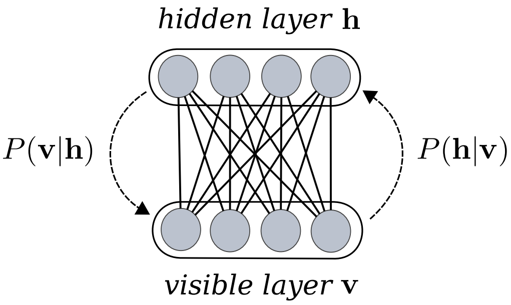

hidden layer $\mathbf{h}$
$P(\mathbf{v}|\mathbf{h})$
$P(\mathbf{h}|\mathbf{v})$
visible layer $\mathbf{v}$

Figure 4.9: The RBM architecture with the visible ($\mathbf{v}$) and hidden ($\mathbf{h}$) layers.

and the parameter $\theta = \{\mathbf{a}, \mathbf{W}\}$ corresponds to the biases a_i of each unit x_i and the symmetric weights w_{ij} connecting units x_i and x_j. The normalizing constant of the distribution $\sum_{\tilde{\mathbf{x}}} e^{-E_\theta(\tilde{\mathbf{x}})}$ is known as the partition function and is intractable in general as it requires to sum an exponential number of terms.

Boltzmann machines can be used as latent variable models in which case the global configuration $\mathbf{x}$ is divided into a set of latent variables $\mathbf{h}$, which are known as *hidden units* and a set of observed variables $\mathbf{v}$, which are known as *visible units* (see Figure 4.8). Accordingly, the probability over the visible units $\mathbf{v}$ is obtained by marginalization:

$$\begin{aligned} P_\theta(\mathbf{v}) &= \sum_{\tilde{\mathbf{h}}} P_\theta(\mathbf{v}, \tilde{\mathbf{h}}) \\ &= \sum_{\tilde{\mathbf{h}}} \frac{e^{-E_\theta(\mathbf{v},\tilde{\mathbf{h}})}}{\sum_{\tilde{\mathbf{v}},\tilde{\mathbf{h}}} e^{-E_\theta(\tilde{\mathbf{v}},\tilde{\mathbf{h}})}} \end{aligned}$$

where the energy function is the same as above with $\mathbf{x} = \mathbf{v}, \mathbf{h}$. Training Boltzmann machines with gradient descent is intractable in general as it requires taking samples from the distribution P_θ. The original paper introducting Boltzmann machines relies on simulated annealing (Kirkpatrick et al., 1983; Ackley et al., 1985).

4.7 Restricted Boltzmann machines

RBMs (Smolensky, 1986) are Boltzmann machines where visible units (resp. hidden units) have no connections between themselves. The structure of an RBM forms an undirected bipartite graph and is given in Figure 4.9. The probability distribution associated with an RBM is:

$$P_\theta(\mathbf{v}) = \sum_{\tilde{\mathbf{h}}} \frac{e^{-E_\theta(\mathbf{v},\tilde{\mathbf{h}})}}{\sum_{\tilde{\mathbf{v}},\tilde{\mathbf{h}}} e^{-E_\theta(\tilde{\mathbf{v}},\tilde{\mathbf{h}})}}.$$

where the energy function $E_\theta(\mathbf{v}, \mathbf{h})$ is given by:

$$E_\theta(\mathbf{v}, \mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} w_{ij} v_i h_j,$$

The parameter $\theta = \{\mathbf{a}, \mathbf{b}, \mathbf{W}\}$ regroups the biases a_i on visible units v_i, the biases on hidden units h_j and the connection weights w_{ij} between units v_i and h_j. Although RBMs seem less general than Boltzmann machines, they are in fact universal approximators (Le Roux and Bengio, 2008), i.e. they can approximate any distribution to arbitrary accuracy, provided there are enough hidden units.

The bipartite structure of the RBM has the advantage of making the conditional probabilities $P(\mathbf{h}|\mathbf{v})$ and $P(\mathbf{v}|\mathbf{h})$ tractable. The computation is done using a probabilistic variant of the usual neural network propagation rule:

$$P(\mathbf{v}|\mathbf{h}) = \prod_i P(v_i|\mathbf{h}) \quad \text{and} \quad P(v_i = 1|\mathbf{h}) = \operatorname{sigm}\left(a_j + \sum_j h_j w_{ij}\right),$$
$$P(\mathbf{h}|\mathbf{v}) = \prod_j P(h_j|\mathbf{v}) \quad \text{and} \quad P(h_j = 1|\mathbf{v}) = \operatorname{sigm}\left(b_j + \sum_i v_i w_{ij}\right),$$

where $\operatorname{sigm}(x) = 1/(1 + \exp(-x))$ is the logistic activation function.

The above energy function leads to a distribution over binary units and therefore is not suitable for continuous valued inputs. Nevertheless, the energy function can be changed by including a quadratic term on the visible units to define the Gaussian-Bernoulli RBM (Krizhevsky and Hinton, 2009):

$$E(\mathbf{v}, \mathbf{h}) = \sum_i \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_{i,j} w_{ij} \frac{v_i}{\sigma_i} h_j,$$

where σ_i represents the variance of the input variable v_i. Using this energy function, the conditional probability $P(\mathbf{h}|\mathbf{v})$ is mostly unchanged, but $P(\mathbf{v}|\mathbf{h})$ is given by a multivariate Gaussian of mean $a_i + \sigma_i \sum_j w_{ij} h_j$ and diagonal covariance matrix:

$$P(v_i = z|\mathbf{h}) = \frac{1}{\sigma_i\sqrt{2\pi}} \cdot e^{-\frac{\left(z - a_i - \sigma_i \sum_j w_{ij} h_j\right)^2}{2\sigma_i^2}},$$
$$P(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_i \frac{v_i}{\sigma_i} w_{ij}\right).$$

By construction, RBMs learn distributed representation and although sparsity is not explicitly enforced, in practice, RBMs also tend to learn sparse representation (Krizhevsky and Hinton, 2009). To better understand the nature of the distributed representation, Figure 4.10 shows the relations between the modes of a Gaussian Bernoulli RBM in the case where there are only two input dimensions.

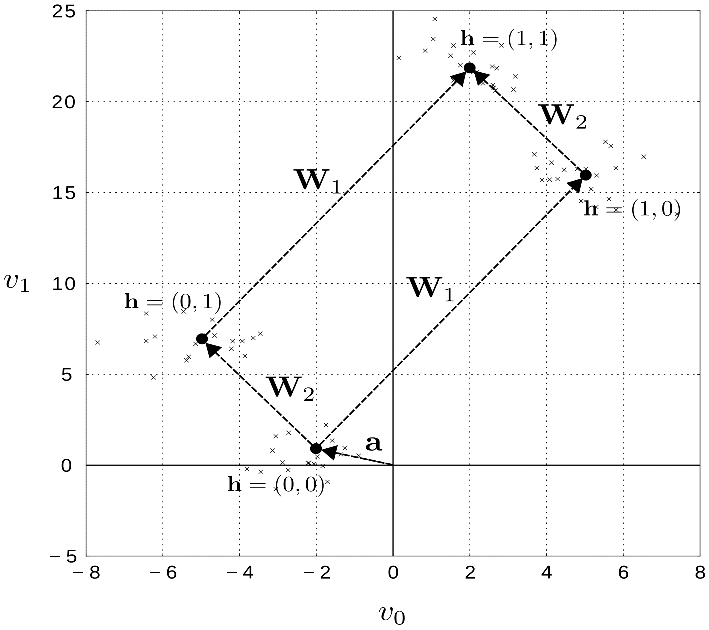


Figure 4.10: Relation between the modes of a Gaussian-Bernoulli RBM with two visible units and two hidden units. The bias $\mathbf{a}$ on the visible units gives the position of the mode for which all hidden units are set to 0. Each row $\mathbf{W}_i$ of the weight matrix $\mathbf{W}$ can then contribute an additive term to the mean of the Gaussian distribution if h_i is set to 1. The points × correspond to samples from each mode of the distribution.

4.8 Training RBMs with Contrastive Divergence

In the spirit of gradient descent, let us derive the partial derivatives of the log-likelihood for a general RBMs:

$$\begin{aligned}\frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} &= \mathbb{E}_{\mathbf{h}\sim P_\theta(\mathbf{h}|\mathbf{v})}\left[\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}\right] - \mathbb{E}_{\mathbf{v},\mathbf{h}\sim P_\theta(\mathbf{v},\mathbf{h})}\left[\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}\right] \\ &= \mathbb{E}_{P_{\text{data}}}\left[\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}\right] - \mathbb{E}_{P_{\text{model}}}\left[\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}\right]\end{aligned}$$

The first term corresponds to the expectation of $\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}$ when the visible units are set to the input vector $\mathbf{v}$ and the hidden variables are sampled according to the conditional distribution $P_\theta(\mathbf{h}|\mathbf{v})$. The second term is the expectation of $\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}$ when $\mathbf{v}$ and $\mathbf{h}$ are sampled according to the joint distribution of the RBM $P_\theta(\mathbf{v},\mathbf{h})$ and is intractable in general.

It can however be approximated with Gibbs sampling: starting from any configuration $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)})$, sample $\mathbf{h}^{(i+1)}$ according to $p_\theta(\mathbf{h}|\mathbf{v}^{(i)})$ and $\mathbf{v}^{(i+1)}$ according to $p_\theta(\mathbf{v}|\mathbf{h}^{(i+1)})$. In the limit of infinitely many iterations, the sample $(\mathbf{v}^{(n)}, \mathbf{h}^{(n)})$ is guaranteed to converge to the target distribution $P_\theta(\mathbf{v},\mathbf{h})$. In practice, it is possible to run Gibbs sampling for only one step by starting the Markov chain with a sample from the training dataset and assuming that the model is not too far from the target distribution. This is the idea behind the Contrastive-Divergence (CD) learning algorithm (Hinton, 2002; Carreira-Perpiñán and Hinton, 2005; Hinton, 2010). Although CD does not maximize the exact log-likelihood, experimental results show that gradient updates almost always improve the likelihood of the model (Hinton, 2002). Moreover, the improvements to the likelihood tend to zero as the length of the chain increases (Bengio and Delalleau, 2009), an argument which supports running the chain for a few steps only[3]. Note that training RBMs with CD is close to the training of auto-encoders in the sense that training auto-encoders with back-propagation corresponds to RBM training where only the biggest term of an expansion of the likelihood is kept (Bengio and Delalleau, 2009).

[3] CD-k refers to the algorithm where k steps of Gibbs sampling are used instead of one. As k gets bigger, the CD-k algorithm gets closer to the true maximum likelihood.

Summary

- An artificial neuron computes a dot product of the input with a weight vector which it maps through an activation function.
- Feed forward neural networks are neural networks which do not have cycles in their connections.
- Multi-layer neural networks are organized in layers of neurons and correspond to a matrix-vector multiplication mapped through an activation function.
- The back-propagation algorithm is an efficient way to implement gradient descent when training feed-forward neural networks.
- Auto-encoders are multi-layer neural networks which learn new representations by minimizing the reconstruction error.
- Boltzmann machines are a form of probabilistic neural network which defines a distribution in the input space with the help of latent variables.
- RBMs are universal approximators despite having less connections than general Boltzmann machines.
- RBMs can be trained efficiently with the CD algorithm.

This presentation of neural networks is by no means exhaustive but is centered on models used in the context of deep neural networks which are studied in the next chapter.

5

DEEP NEURAL NETWORKS

The models presented in the previous chapter have been applied successfully in many settings, however their application to large scale problems often leads to generalization and computational issues. To deal with these issues, deep learning considers architectures with many layers which scale more gracefully as problems become more complex[1].

In this section, we start by a definition of deep architectures and compare them with *shallow architectures*. We present supervised methods for deep learning, namely simple deep feed-forward networks and convolutional networks and then explain how deep learning can benefit from a *layer-wise approach* where layers are learned one at a time from the bottom up in an unsupervised fashion. Accordingly, we present stacked RBMs and stacked auto-encoders which are the most successful approaches to layer-wise deep learning. We then review recent works which may improve the performance of these models: variations of stacked RBMs and auto-encoders, better ways to estimate the log-likelihood and richer models. Finally, we show applications where deep learning has led to breakthroughs and discuss the possible limitations of the deep learning approach.

5.1 Shallow v.s. deep architectures

Deep architectures are designed by opposition with *shallow architectures* which have only one layer of hidden variables. In the supervised setting, this hidden layer is typically followed by a linear layer to produce the output (Bengio and LeCun, 2007). Many shallow architectures such as Gaussian mixtures, RBMs and neural networks with one non-linear hidden layer are universal approximators, thus they can represent any function in theory. However, there is an important restriction, namely they can only approximate any function *given a sufficient number of hidden variables*. In practice this assumption becomes unrealistic with highly varying functions as the number of parameters needed can scale exponentially in terms of the input dimension (Bengio et al., 2006; Bengio and LeCun, 2007; Bengio et al., 2007), a typical example of the curse of dimensionality. By contrast, deep architectures which allow for more layers, can lead to much more

[1] See (Bengio et al., 2012; Bengio, 2013) for recent reviews on deep learning.

efficient representations while still being universal approximators (Sutskever and Hinton, 2008). For example, the parity function in D dimensions requires $\mathcal{O}(2^D)$ parameters to be represented by a Gaussian Support Vector Machine (SVM) (Bengio et al., 2006), $\mathcal{O}(D^2)$ parameters for a neural network with one hidden layer and $\mathcal{O}(D)$ parameters for a network with $\mathcal{O}(\log_2 D)$ layers (Bengio et al., 2007). Because they may require less parameters, deep architectures have the potential to both improve generalization and reduce computational costs.

In practice, deep architectures seem to learn very interesting representations which can be invariant to several transformations of the input such as translations, and rotations (Goodfellow et al., 2009). These representations are almost always distributed[2] (which allows for *non-local generalization*, see Bengio and LeCun, 2007; Bengio, 2007), and sparse (either explicitly with a penalization term or implicitly when the hidden variables are binary). Furthermore, deep architecture are able to learn hierarchical representations (Lee et al., 2009a) which have remarkable similarities with the areas V1 and V2 (Lee et al., 2007) of the primate's visual cortex.

5.2 Deep feed-forward networks

Until recently, training huge traditional feed-forward networks with back-propagation was thought to be very difficult (Bengio and Glorot, 2010). Nonetheless, Ciresan et al. (2010) showed that such networks can be trained to achieve state of the art performance on the Mnist handwritten digit classification dataset. Their models use between 1.34 and 12.11 million parameters which in principle would lead to poor generalization and intractably high computational cost. In order to tackle the issue of generalization, Ciresan et al. generate new examples by deforming the original dataset. The transformations combine rotation, scaling, horizontal shearing and an elastic model of deformation particularly suited to handwritten data. This method allows the generation of new samples in real-time during training thus solving the over-fitting issue with large amounts of data. As for the computational issue, it is addressed by running the code on a Graphics Processing Unit (GPU) which speeds up the elastic model of deformation by a factor of 10 and the back-propagation algorithm by a factor of 40. Although this method achieves remarkable performance using a very specific model of deformations, generalization to other settings is problematic. Additionally, the Mnist dataset is concerned with the classification of digits from 0 to 9 which is arguably not very difficult, thus the computational cost may prevent the approach from scaling to more complicated problems.

Training deep feed-forward neural networks with the traditional back-propagation may yet be possible by using a different non-linearity which saturates less often

[2] A model which does not have a distributed representation cannot combine low level features to construct higher level features. Therefore, there does not seem to be any point in having deep architectures without distributed representations.

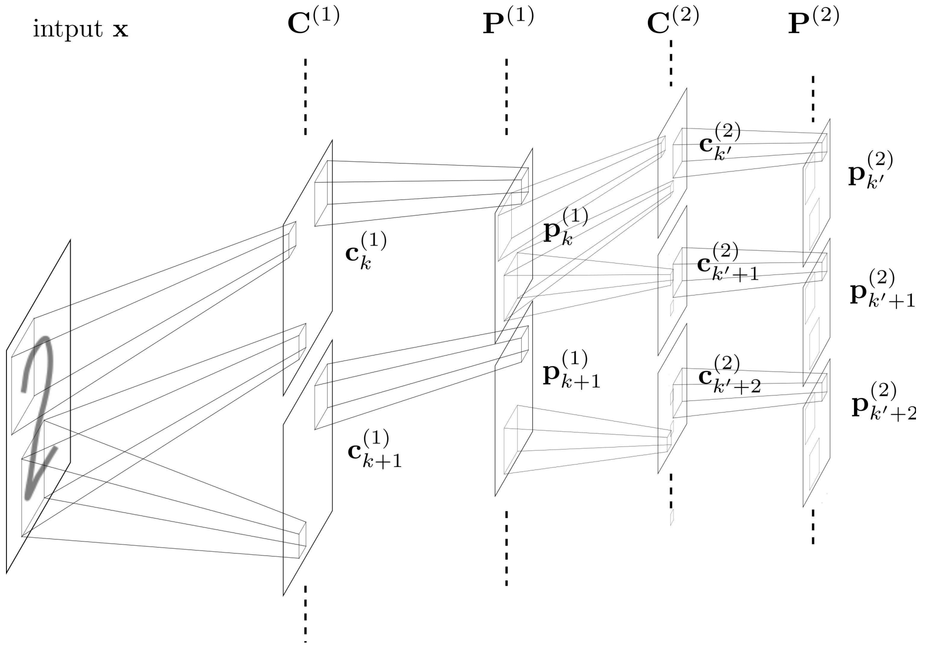


Figure 5.1: Convolutional and pooling layers of a convolutional network. New layers can be added by considering each pooling plane as the input of a new convolutional network.

and by using a different initialization of the weights (Bengio and Glorot, 2010). Additionally, the issue of generalization may be addressed by dropping out half of the hidden units randomly at each step of the training procedure (Hinton et al., 2012). Using dropout seems to prevent the co-adaptation of neurons since each neuron must carry enough information by itself to be relevant even when half of the other neurons are absent. At test time, all the weights of the networks are halved to account for the fact that neurons are twice as likely to be active. In the classification setting where the objective is to learn $p(\mathbf{y}|\mathbf{x})$, the model then gives the geometric mean of 2^N distributions corresponding to the 2^N possible models where different hidden units have been dropped. In practice, dropout greatly improves performance in settings such as speech and object recognition (Hinton et al., 2012).

5.3 Convolutional networks

Convolutional networks (Le Cun et al., 1990; LeCun et al., 1998a) are deep neural networks, particularly adapted to vision tasks, which improve generalization and decrease computational costs by reducing the number of parameters. In practice, the number of parameters is reduced by constraining many weights to share the same values (Rumelhart et al., 1986).

The model has a succession $\mathbf{C}^{(1)}, \mathbf{P}^{(1)}, \mathbf{C}^{(2)}\mathbf{P}^{(2)}, \ldots, \mathbf{C}^{(M)}\mathbf{P}^{(M)}$ of convolutional ($\mathbf{C}$) layers and pooling ($\mathbf{P}$) layers, inspired by the simple/complex cell organization of the visual cortex (Fukushima, 1980). The connections are feed-forward such that all the inputs of a convolutional layer are in the preceding pooling layer, and all the inputs of a pooling layer are in the preceding convolutional layer. Each convolutional layer $\mathbf{C}^{(m)}$ is composed of several feature planes $\mathbf{c}_1^{(m)}, \mathbf{c}_2^{(m)}, \ldots, \mathbf{c}_K^{(m)}$ which compute convolutions with the input $\mathbf{x}$ and a set of small features $\mathbf{f}_1^{(m)}, \mathbf{f}_2^{(m)}, \ldots, \mathbf{f}_K^{(m)}$ i.e. computing $\mathbf{c}_k^{(m)} = \mathbf{x} * \mathbf{f}_k^{(m)}$. See figure 5.1 for a visual representation of convolutional layers.

Each pooling layer $\mathbf{P}^{(m)}$ corresponds to an averaging/sub-sampling operation of the convolutional layer $\mathbf{C}^{(m)}$ where all weights are shared, event within a single neuron. A typical pooling operation reduces an image size by a factor of 2 along all directions by taking the averages of regions of 4×4 pixels which do not overlap. Each pooling plane can then be used as input for a new convolutional layer. See figure 5.1 for a visual representation of pooling layers.

Convolutional networks can be applied in any setting where the input is expected to be invariant by translation. For instance, convolutional networks have been very successful on the classification of handwritten digits (LeCun, 1989; Le Cun et al., 1990), object recognition (Jarrett et al., 2009), pedestrian detection (Kavukcuoglu et al., 2010b), speech and time series (LeCun and Bengio, 1995) and Natural Language Processing (Collobert and Weston, 2008). In addition, due to the reduced number of parameters, the optimization takes place in a search space of smaller dimension which is expected to speed up the learning procedure.

5.4 Layer-wise learning of deep representations

Although the previous two sections show how to directly apply deep learning to supervised problems, there may be an even more efficient way to train deep architectures by using unsupervised representation learning to learn *deep representations*. Once a deep representation has been learned, it may be used to solve the final task easily. An important insight for training deep architectures consists in the potentially recursive power of representation learning. Namely, suppose that an algorithm is capable of learning a better representation $\mathbf{h}^{(1)}$ for some data $\mathbf{x}$. In principle, the same algorithm may be used to learn an even better representation $\mathbf{h}^{(2)}$ from $\mathbf{h}^{(1)}$, and so on for many layers. This is the basis of *layer-wise* deep learning: iteratively train a better representation $\mathbf{h}^{(k+1)}$ given the previous representation $\mathbf{h}^{(k)}$ using an unsupervised algorithm.

Because representation learning and deep learning both have benefits w.r.t generalization[3], learning deep representations is an appealing approach. However layer-wise learning can lead to further improvements. For K layers and D neurons per layer, layer-wise learning transforms an optimization problem over $K \times D$ parameters to K subproblems over D parameters. This has two impli-

[3] These benefits come from the basis of representation learning on unsupervised learning (see Section 2.7) and the efficiency of deep learning in terms of the number of parameters (see the beginning of Chapter 5)

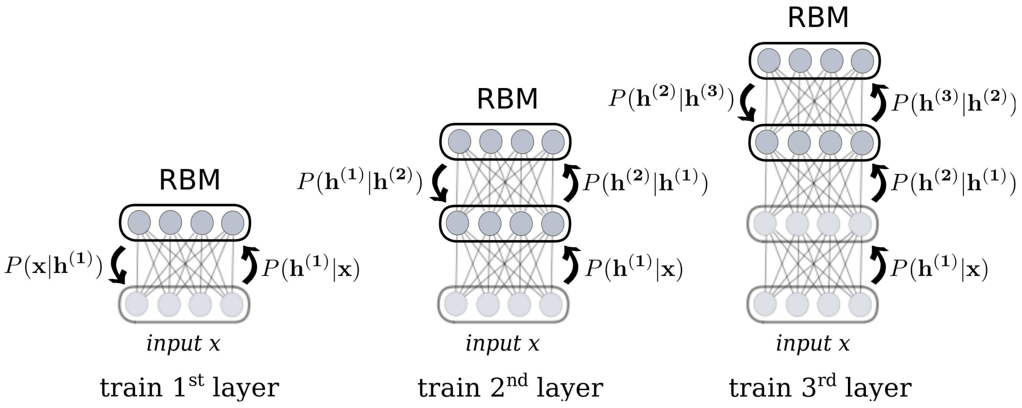

RBM
$P(\mathbf{x}|\mathbf{h}^{(1)})$
$P(\mathbf{h}^{(1)}|\mathbf{x})$
input x
train 1st layer
RBM
$P(\mathbf{h}^{(1)}|\mathbf{h}^{(2)})$
$P(\mathbf{h}^{(2)}|\mathbf{h}^{(1)})$
$P(\mathbf{h}^{(1)}|\mathbf{x})$
input x
train 2nd layer
RBM
$P(\mathbf{h}^{(2)}|\mathbf{h}^{(3)})$
$P(\mathbf{h}^{(3)}|\mathbf{h}^{(2)})$
$P(\mathbf{h}^{(2)}|\mathbf{h}^{(1)})$
$P(\mathbf{h}^{(1)}|\mathbf{x})$
input x
train 3rd layer

Figure 5.2: Illustration of the stacked RBMs training scheme.

cations: first, from an optimization perspective, optimization in dimension D is much more efficient and may lead to better results than in dimension $K \times D$; second, w.r.t. generalization, the smaller number of parameters makes each layer less prone to over-fitting which leads to better generalization overall. Finally, the representations obtained with a layer-wise learning procedure can be used as a form of *unsupervised pre-training* to initialize a supervised model and improve generalization (Erhan et al., 2009; Larochelle et al., 2009).

5.5 Stacked RBMs and deep belief networks

A first way to learn deep representations with a layer-wise approach is to use stacked RBMs (Hinton et al., 2006; Bengio et al., 2007). The layer-wise training procedure of stacked RBMs is given in Figure 5.2: Let a first RBM be trained on the input distribution $P_{\mathcal{D}}(\mathbf{x})$. This RBM then provides a way to compute the hidden representations $\mathbf{h}^{(1)}$ using the conditional distribution $P(\mathbf{h}^{(1)}|\mathbf{x})$. A second RBM can then be trained on the input distribution as represented by the latent variables $Q(\mathbf{h}^{(1)}) = \sum_{\mathbf{x}} P(\mathbf{h}^{(1)}|\mathbf{x})P_{\mathcal{D}}(\mathbf{x})$.

Once several layers have been trained, it is possible to consider stacked RBMs as a *deep belief network*. In this scheme, all RBMs except for the top one are seen as directed belief networks transmitting information downwards with the operation $P(\mathbf{h}^{(k)}|\mathbf{h}^{(k+1)})$. Samples can be generated from this model by sampling from the top RBM, with e.g. Gibbs sampling, and sampling sequentially with $P(\mathbf{h}^{(k)}|\mathbf{h}^{(k+1)})$ the state of layer $\mathbf{h}^{(k)}$ given the state of the layer above $\mathbf{h}^{(k+1)}$.

In the supervised setting, a deep belief network can be used to initialize a deep feed-forward network. In practice a new randomly initialized layer is added on top of the existing RBMs, linking the topmost layer to the target variable. All the layers can then be fine-tuned with back-propagation to maximize the supervised objective. In theory, tuning all the parameters at once may result in over-fitting but in practice the layer-wise unsupervised pre-training has a regularizing effect which prevents from falling into bad minima. Unsupervised pre-training with stacked RBMs can be seen as providing a good initialization for the supervised fine-tuning, placing the optimization procedure in a basin of attraction which has good generalization properties (Erhan et al., 2009). Deep belief networks have been shown to be state-of-the-art in several settings such as handwritten digit recognition (Hinton et al., 2006), object recognition (Nair and Hinton, 2009), phone recognition (rahman Mohamed et al., 2011), and modeling human motion (Taylor et al., 2007; Taylor and Hinton, 2009). If the input is known to be invariant with respect to translation, deep belief networks can be adapted to work with tied weights to form a *convolutional* deep belief network which is then even more efficient (Lee et al., 2009a; Krizhevsky, 2010; Lee et al., 2009b).

Formally, the layer-wise training procedure of stacked RBMs consists in the maximization of a variational lower bound on the probability of a deep belief network. Namely, let us rewrite the log-likelihood of a deep belief network associated with a probability P. For any distribution $Q(\mathbf{h}|\mathbf{x})$, we have:

$$\begin{aligned}\log P(\mathbf{x}) &= d_{KL}(Q(\mathbf{h}|\mathbf{x})||P(\mathbf{h}|\mathbf{x})) + H_{Q(\mathbf{h}|\mathbf{x})} + \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x})(\log P(\mathbf{x}|\mathbf{h}) + \log P(\mathbf{h})) \\ &\leq H_{Q(\mathbf{h}|\mathbf{x})} + \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x})(\log P(\mathbf{x}|\mathbf{h}) + \log P(\mathbf{h}))\end{aligned}$$

Let us denote by $R_1(\mathbf{x}, \mathbf{h})$ the probability of a single RBM and suppose for now that we have a deep belief network composed of this single layer, i.e. $P(\mathbf{x}) = \sum_{\mathbf{h}} R_1(\mathbf{x}, \mathbf{h})$. The above lower bound being defined for any $Q(\mathbf{h}|\mathbf{x})$, it holds for $Q(\mathbf{h}|\mathbf{x}) = R_1(\mathbf{h}|\mathbf{x})$. Using the fact that $P(\mathbf{x}|\mathbf{h}) = R_1(\mathbf{x}|\mathbf{h})$, the variational lower bound can be rewritten as:

$$H_{R_1(\mathbf{h}|\mathbf{x})} + \sum_{\mathbf{h}} R_1(\mathbf{h}|\mathbf{x})(\log R_1(\mathbf{x}|\mathbf{h}) + \log R_1(\mathbf{h}))$$

Let us now show that adding a layer $\mathbf{h}'$ can only improve performance. First we construct a new RBM R_2 consisting of the RBM R_1 upside down, i.e. $R_2(\mathbf{h}', \mathbf{h}) = R_1(\mathbf{x}, \mathbf{h})$. By construction $R_2(\mathbf{h})$ is equal to $R_1(\mathbf{h})$ which allows us to rewrite the bound as

$$H_{R_1(\mathbf{h}|\mathbf{x})} + \sum_{\mathbf{h}} R_1(\mathbf{h}|\mathbf{x})(\log R_1(\mathbf{x}|\mathbf{h}) + \log R_2(\mathbf{h}))$$

If a deep belief network is composed of a single RBM R_1 the bound is tight since $P(\mathbf{h}|\mathbf{x}) = R_1(\mathbf{h}|\mathbf{x})$ and therefore $d_{KL}(R_1(\mathbf{h}|\mathbf{x})||P(\mathbf{h}|\mathbf{x})) = 0$. When we replace $R_1(\mathbf{h})$ by $R_2(\mathbf{h})$, using the fact that $R_2(\mathbf{h}', \mathbf{h}) = R_1(\mathbf{x}, \mathbf{h})$ to create a deep belief network, we are then sure that the bound is tight as well:

$$P(\mathbf{x}) = \sum_{\mathbf{h}} R_1(\mathbf{x}|\mathbf{h}) \sum_{\mathbf{h}'} R_2(\mathbf{h}, \mathbf{h}')$$

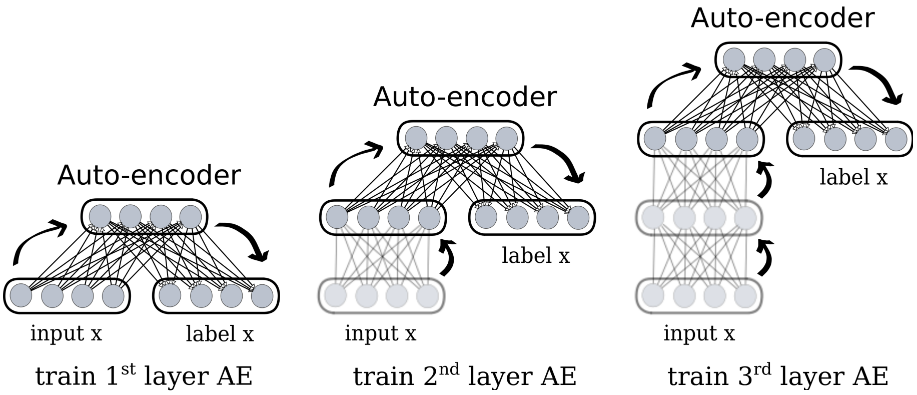


Figure 5.3: Illustration of the stacked auto-encoder training scheme.

The bound can then be increased by maximizing $\sum_{\mathbf{h}} R_2(\mathbf{h}) R_1(\mathbf{h}|\mathbf{x})$ with respect to the parameters of R_2 while never falling below the point at which the bound is tight.

Despite their empirical performance, the maximization of the above variational lower bound in deep belief networks poses several questions. First, the maximization of the lower bound could well be stuck at a point much lower than the true log-likelihood and therefore lead to an unsatisfactory solution. In addition, the proof of log-likelihood improvement is based on the initialization of a layer to an upside down version of the previous layer, a trick which is never used in practice and is likely to have unfortunate consequences when considering asymmetric models such as Gaussian-Bernoulli RBMs. Furthermore, the proof does not hold for more than two layers since the inference distribution for the deep model is unknown. Although a third layer can be guaranteed to improve the log-likelihood of $P(\mathbf{h})$ w.r.t. the target distribution $\sum_{\mathbf{x}} P(\mathbf{h}|\mathbf{x}) P_{\mathcal{D}}(\mathbf{x})$, this only guarantees an increase of the variational lower bound of $P(\mathbf{x})$, not an increase of the likelihood of $P(\mathbf{x})$. Finally, the method does not explicitly define what criterion for learning the first layer of a deep belief network. In practice, the first layer is trained to maximize the likelihood of a single RBM, hoping to set bound at the highest possible value before the next layer is trained, but this does not maximize the log-likelihood of the final deep belief network, nor even the variational lower bound w.r.t the parameters of the first RBM when the parameters of the second RBM have been trained. Because of all these factors, we expect that as more layers are added, the optimization of the variational lower bound may become less effective to optimize the log-likelihood of a deep belief network.

5.6 STACKED AUTO-ENCODERS AND DEEP AUTO-ENCODERS

Another way to learn deep representations is to use stacked auto-encoders Bengio (2007) to learn deep representations. First, an auto-encoder is trained to reconstruct its input $\mathbf{x}$ with one hidden layer $\mathbf{h}^{(1)}$, thus learning an encoder and a decoder performing the operation $\mathbf{x} \overset{\text{enc}}{\to} \mathbf{h}^{(1)} \overset{\text{dec}}{\to} \mathbf{x}$. A new auto-encoder can then be used to learn a representation $\mathbf{h}^{(k+1)}$ from a representation $\mathbf{h}^{(k)}$. Figure 5.3 gives a visual illustration of this training procedure.

Stacking auto-encoders can be seen as an alternative to stacking RBMs for pre-training a deep feed-forward neural network, an approach which has proved very successful in applications such as handwritten digit recognition and audio classification (Vincent et al., 2008, 2010; Larochelle et al., 2007, 2009). However, stacking auto-encoders has a troubling lack of theoretical justification[4]. Not only is there no formal guarantee that stacking auto-encoders improves performance compared to a single auto-encoder, but worse the theoretical encoder/decoder perspective leads to the (erroneous) conclusion that adding layers might decrease performance. As we consider adding a new layer $\mathbf{h}^{(1)} \to \mathbf{h}^{(2)} \to \mathbf{h}^{(1)}$ to an auto-encoder $\mathbf{x} \overset{\text{enc}}{\to} \mathbf{h}^{(1)} \overset{\text{dec}}{\to} \mathbf{x}$, the resulting deep auto-encoder corresponds the operation $\mathbf{x} \to \mathbf{h}^{(1)} \to \mathbf{h}^{(2)} \to \mathbf{h}^{(1)} \to \mathbf{x}$, which is expected to introduce additional errors on the representation $\mathbf{h}^{(1)}$. A better understanding of stacked auto-encoders may be found under the perspective of learning deep generative models. Since training auto-encoders is a close approximation to the RBM training procedure (Bengio and Delalleau, 2009), it may be interpreted as approximating a variational lower bound maximization for the log-likelihood of a deep generative model.

In practice, stacking auto-encoders can act as a good pre-training phase for a deep feed-forward neural network, improving supervised performance almost as effectively as stacked RBMs (Larochelle et al., 2007, 2009). Furthermore, the encoder/decoder approach is perfectly justified to increase performance in a deep auto-encoder which has already been pre-trained to perform the operation:

$$\mathbf{x} \to \mathbf{h}^{(1)} \to \cdots \to \mathbf{h}^{(K)} \to \cdots \to \mathbf{h}^{(1)} \to \mathbf{x}$$

When the pre-training step has not fully maximized performance, a global fine-tuning of the reconstruction error with back-propagation may improve the quality of hidden representations.

The combination of stacked RBMs having a more solid theoretical justification than stacked auto-encoders for pre-training, and deep auto-encoders having a good justification for fine-tuning deep representations, has led many practitioners to mix both approaches in their applications: using stacked RBMs for pre-training and the deep auto-encoder approach for fine-tuning. Note that auto-encoders are compatible with deep belief networks since they can be seen under a probabilistic perspective: learning a conditional probability $P(\mathbf{x}|\mathbf{h})$ in the decoder and an inference distribution $Q(\mathbf{h}|\mathbf{x})$ in the encoder[5]. This approach was shown to be very successful in applications such as dimensionality reduc-

[4] The situation on the justification of auto-encoders is evolving rapidly, see (Alain and Bengio, 2012; Bengio et al., 2013) for recent works on this subject.

[5] For this to work, both networks should of course use the same activation function.

tion (Hinton and Salakhutdinov., 2006), semantic hashing (Salakhutdinov and Hinton, 2009b), binary coding of speech spectrograms (Deng et al., 2010), and multimodal learning (Ngiam et al., 2011).

5.7 Variations on RBMs and stacked RBMs

After the success of deep architectures based on stacked RBMs, many studies have tried to improve RBMs by allowing continuous inputs and continuous hidden units, but also by designing new training procedures.

As the traditional RBM is only capable of dealing with binary values, several approaches propose ways to modify the energy function to obtain continuous valued units. To deal with continuous inputs, the most widely adopted approach is to use Gaussian Bernoulli RBMs (Bengio et al., 2007; Krizhevsky, 2010) where the Gaussian units have an activation centered on the pre-activation. Other possible choices include exponential and truncated exponential units (Bengio et al., 2007). RBMs can also be extended to allow for continuous hidden units for instance with rectified linear units (Nair and Hinton, 2010), rectifier networks (Glorot et al., 2010) and continuous sigmoidal belief networks (Frey, 1996; Frey and Hinton, 1999; Adams et al., 2010). Spike and slab RBMs (Courville et al., 2011a,b; Goodfellow et al., 2012) are an interesting approach which consists in decomposing the activations of hidden units into a binary *spike* component which determines whether a hidden unit should activate at all, and a *slab* component which corresponds to the amplitude of the activation. Spike and slab RBMs have proved very effective to discover interesting representations for images.

Other studies have examined ways to improve the training procedure for RBMs. Maximum likelihood is intractable in general because it requires samples from the model which are not easily obtained. However, approximations such as CD have shown to be quite effective in practice. The first approach to improve the RBM training procedure compared to CD is to generate samples which are distributed more closely to the model distribution. This can be done with *persistent contrastive divergence* (updating several Markov chains for the model distribution at each step as in Tieleman, 2008; Tieleman and Hinton, 2009; Breuleux et al., 2011), *tempered transitions* (running a single chain at increasing and decreasing temperatures, as in Salakhutdinov, 2009) or *parallel tempering* (running several chains at different temperatures, as in Desjardins et al., 2010). Another approach is to use inductive methods (Marlin et al., 2010) such as pseudo-likelihood, ratio-matching and score-matching (Hyvärinen, 2005; Swersky et al., 2011), which try to avoid the issues of maximum likelihood when dealing with intractable partition functions.

Although RBMs are typically used to estimate a density in the unsupervised setting, RBMs can also reconstruct incomplete inputs. This possibility can be used in the supervised setting with discriminative RBMs (Larochelle and Bengio, 2008; Schmah et al., 2008; Larochelle et al., 2012). By using the concatenation of an input $\mathbf{x}$ and a label $\mathbf{y}$ as the visible layer of a RBM, the conditional probability $p(\mathbf{y}|\mathbf{x})$ becomes tractable and can therefore be used to take samples or find the most probable value of the label $\mathbf{y}$ given $\mathbf{x}$.

5.8 Tractable estimation of the log-likelihood

Due to their intractable partition function, the evaluation of RBMs and deep belief networks according to the likelihood is a difficult problem.

Recent studies have shown that the partition function of RBMs can be estimated efficiently using Annealed Importance Sampling (AIS) (Neal, 1998; Salakhutdinov, 2008; Salakhutdinov and Murray, 2008), a variant of importance sampling where a Markov chain maintains samples of an RBM at different temperatures. Another possibility is to track the partition function during learning when the training procedure allows it, for instance in the case of parallel tempering (Desjardins et al., 2011).

However, estimating the log-likelihood for Deep Belief Networks poses a much more serious problem as every layer adds a new sum over an exponential number of states. A possibility is to consider the variational lower bound used to train deep belief networks as a proxy for the likelihood of the deep model (Salakhutdinov and Murray, 2008; Salakhutdinov, 2008). This approach has the advantage of being consistent with the training procedure, but it is exposed to the same issues. Namely, the assumption that the lower bound is close to the true likelihood does not seem justified in theory, especially for more than two layers. If it is true that using a lower bound has the advantage that it does not lead to over-estimating results, it cannot be used in most practical applications which involve comparing several models: since the lower bound under-estimates the true likelihood by an unknown amount, a model with the highest variational lower bound is not necessarily the one with the highest log-likelihood.

A more promising approach could be to employ an MCMC method to take reasonably well distributed samples from the model as in (Murray and Salakhutdinov, 2009). However, it is difficult to know whether the approach truly scales to larger models as it would require comparing the estimator to the true likelihood which, unfortunately, is intractable for larger models.

5.9 Variations on auto-encoders

Many studies are dedicated to improving auto-encoders. The first improvement which allowed auto-encoders to achieve performance similar to that of stacked RBMs was the introduction of a denoising criterion. Namely, instead of trying to reconstruct its input, the *denoising auto-encoder* (Vincent et al., 2008) is trained to recover the original input $\mathbf{x}$ given a corrupted version $\tilde{\mathbf{x}}$ where some percentage of bits have been set to 0. A first advantage is that the auto-encoder cannot learn the identity and does not need to be combined with other forms of regularization such as using a hidden layer smaller than the input, sparsity or tied weights. Additionally the approach can be justified under the manifold hypothesis in which the training points $\mathbf{x}$ are supposed to lie on a small dimensional sub-manifold of the input space. As it learns to map close variations $\tilde{\mathbf{x}}$ to the original point $\mathbf{x}$, a denoising auto-encoder tries to find a representation $\mathbf{h}$ which ignores directions of variation orthogonal to the manifold. Variations in directions which lie inside the manifold must still be taken into account to minimize the reconstruction error. Note that this capacity to accurately represent points inside the manifold while being invariant to variations orthogonal to the manifold is exactly what we would expect from a good coordinate system within the manifold. Interestingly, the denoising auto-encoder training procedure is closely linked with score-matching (Vincent, 2011; Swersky et al., 2011).

The same principle can be applied with *contractive auto-encoders* (Rifai et al., 2011): using the Frobenius norm $\sum_{i,j}\left(\frac{\partial h_j(\mathbf{x})}{\partial x_i}\right)^2$ of the Jacobian $J_{\mathbf{h}}$, as a penalty term during optimization. While this penalty term tries to make the representation $\mathbf{h}$ less sensitive to variations in all directions around the training examples, by combining it with a criterion such as the reconstruction error, $\mathbf{h}$ still has to be sensitive to directions which lie within the data manifold because it needs to give a different representation to different training examples. Contractive auto-encoders have been shown to improve performance on several tasks such as handwritten digit classification and object recognition (Rifai et al., 2011). Additionally, the invariance of the hidden representation w.r.t. directions which are orthogonal to the data manifold allow the construction of a Markov chain which moves along the data manifold with a first order approximation, essentially taking samples from the manifold (Rifai et al., 2012) by moving a representation $\mathbf{h}$ to a representation $\mathbf{h} + \boldsymbol{\epsilon} JJ^T$ where $\boldsymbol{\epsilon}$ is a small isotropic random variation.

5.10 Richer models for layers

Deep learning approaches might also benefit from using richer models for each layer.

A first possibility is to use higher order Boltzmann machines (Sejnowski, 1986) where the energy function includes interactions terms between more than two units. For instance, the energy function of a third order Boltzmann machines includes terms $\sum_{i,j,k} t_{ijk} x_i x_j x_k$ which account for interactions between three units x_i, x_j and x_k. The weight matrix is then replaced by a tensor which makes computational costs very prohibitive in the general case. However, it is possible to allow for some higher order interaction while keeping the model tractable, for example, using *Gated RBMs* (where hidden units mediate the interactions between visible units as in Memisevic and Hinton, 2007; Memisevic, 2008) or *factored RBMs* (where three-way interactions are approximated by a sum of factors $\sum_f (\sum_i x_i B_{if})(\sum_j x_j C_{if})(\sum_k x_k D_{if})$ as in Ranzato et al., 2010; Memisevic and Hinton, 2010).

Another interesting approach is to consider *deep Boltzmann machines* (Salakhutdinov and Hinton, 2009a). Just like deep belief networks, deep Boltzmann machines can be pre-trained with stacked RBMs (Salakhutdinov and Hinton, 2009a, 2012), but unlike deep belief networks, these RBMs are then combined to form an undirected graphical model. After pre-training, the network is fine-tuned to maximize the likelihood of the full model using a variational mean field approach to perform inference, and running persistent Markov chains to sample from the model. Even if it relies on several approximations, this global fine-tuning step tries to maximize the true likelihood of the deep model, contrary to deep belief networks which only maximize a variational lower bound. Deep Boltzmann machines have shown to be very efficient for object recognition (Salakhutdinov and Hinton, 2009a) and for multimodal deep learning (Srivastava and Salakhutdinov, 2012).

Several approaches propose to improve the training procedure of deep Boltzmann machines for instance with a different pre-training procedure (Cho et al., 2012; Salakhutdinov and Hinton, 2012) or avoiding pre-training altogether and using a novel way to train all layers jointly (Goodfellow et al., 2013).

5.11 Concrete breakthroughs

Since their introduction in 2006, deep learning approaches have been shown to outperform state-of-the-art methods in several settings. We now present several breakthroughs in ML which can be attributed to the deep learning approach.

The origin of deep learning can be traced back to the University of Toronto Deep Learning Group which, in 2006, was able to reduce the error rate by 11% compared to SVMs on the Mnist handwritten digit classification dataset using Deep Belief Networks. The RBMs on which the approach was based were found to be very efficient during the Netflix Prize[6]. Although the Korbell team which won the 2007 Netflix progress prize used an ensemble method based on 107 models (Bell et al., 2007), only the two best ones: RBMs and Singular Value Decomposition were put into production (they were still in use in 2012). The remaining 105 models only accounted for a 3% reduction in the error rate. In 2009, The Swiss AI Lab IDSIA won the ICDAR Arabic Connected Handwriting Competition and the ICDAR Handwritten Farsi/Arabic Character Recognition Competition and the ICDAR French Connected Handwriting Competition (Graves and Schmidhuber, 2008; Graves, 2012) with a deep neural network variant. In 2011, The Université de Montreal LISA laboratory and the INSA de Rouen LITIS laboratory showed the potential of unsupervised representation learning in the context of transfer learning by winning the final phase of the Unsupervised and Transfer Learning Challenge with contractive auto-encoder and spike & slab RBMs, outperforming the second place by 7% on average (Mesnil et al.,

[6] The corresponding Netflix blog-post can be found at http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html

2012). Also, the Swiss AI Lab IDSIA was able to win both the ICDAR 2011 Chinese handwriting recognition competition (Meier et al., 2011) and the IJCNN 2011 on-site Traffic Sign Recognition Competition with deep neural networks, outperforming human beings in the process (Ciresan et al., 2012b). In 2012, a deep learning approach to speech recognition proposed by Microsoft Research was able to reduce the error rate by 16% compared to traditional Gaussian mixture models (Dahl et al., 2012). Additionally, as part of a collaboration between Google and the University of Stanford, a single deep neural network trained on $16,000$ cores with 9 layers and 10^9 connections was able to outperform the state of the art by 15% (resp. 70%) on the $10,000$ (resp. $20,000$) object categories from the ImageNet dataset. Furthermore, this experiment showed that it is possible to obtain robust detectors for a supervised task (face detection) with only unsupervised training examples (Le et al., 2012), a result which might be interpreted as a demonstration that an ML algorithm can solve tasks for which it has not been trained specifically, an important step towards general purpose AI. The Swiss AI Lab IDSIA also won first place at the ICPR competition on mitosis detection in breast cancer histological images and the ISBI challenge on segmentation of neuronal structures (Ciresan et al., 2012a). Finally, the University of Toronto Deep Learning Group was able to win the Merck Drug Discovery Competition with deep neural networks which randomly drop out hidden units during learning to improve generalization (Hinton et al., 2012). The approach required no feature engineering and only minimal preprocessing and showed that deep neural networks may be applicable in settings which traditionally require a lot of feature engineering.

5.12 Principles of deep learning under question ?

Despite the impressive achievements given in the preceding section, a lot could be learned by looking at the limits of deep learning. Accordingly, a few publications spell out arguments against deep learning and layer-wise learning.

A first question concerns the usefulness of deep architectures compared to shallow architectures. In the work of Coates et al. (2011), an analysis of single layer networks on image classification reveals that the use of deep learning may be a less important factor than the choice of several orthogonal aspects of the learning algorithm: whitening, large numbers of features, and dense feature extraction. In their experiments, Coates et al. show that it is possible to outperform deep neural networks with Gaussian mixture models. If the use of whitening, large numbers of features, and dense feature extraction are effectively proven to be important aspects of a learning algorithm, the conclusion that deep learning is a less important factor can be misleading. Indeed, the number of features required by a deep architecture is expected to scale in $\mathcal{O}(D \log D)$ compared to $\mathcal{O}(2^D)$ for a mixture of Gaussian (Bengio and LeCun, 2007). By comparison, the effect of tuning important parameters of the learning algorithm is not expected to help scaling to bigger architectures. As a result, using deep architectures can probably be considered the most important factor in the long run, even if there are more efficient approaches when the number of hidden features is limited[7].

Another question concerns the efficiency of a layer-wise training procedure. The work of Ciresan et al. (2010) is often taken as a demonstration that layer-wise learning may be unnecessary and that training with back-propagation can be efficient enough to train deep neural networks. However as we discussed at the beginning of this chapter, the approach of Ciresan et al. is based on an elastic model of deformations for handwritten characters and cannot easily be generalized to other applications. Thus, this publication cannot be seen as a demonstration that layer-wise learning is unnecessary. The dropout approach (Hinton et al., 2012) on the other hand seems to confirm that it is possible to train deep feed-forward neural networks without pre-training in the general case. Nonetheless, layer-wise pre-training may still be necessary for larger models as it is expected to scale more gracefully.

Finally, Theis et al. (2011) discuss in their work the limitations of current deep learning approaches w.r.t. the maximization of the likelihood. Using a novel approach to evaluate the likelihood for deep belief networks, they show that deep belief networks can be outperformed by mixtures of Gaussian, and that adding layers only marginally improves the likelihood when each layer is trained thoroughly. Although these negative results may be caused by the limitations of training deep belief networks with a maximization of a variational lower bound (a valid concern) they may also be explained by the fact that the experiments only test models with less than 100 hidden features, a setting in which deep networks are expected to be at a disadvantage[8].

Even if attempts to find limits to the deep learning principles have not been quite successful, the deep learning methods used in practice still leave a lot of room for improvement. Accordingly, we now turn to what can be done to improve the current state of the art.

[7] In their study, Coates et al. consider up to 4,000 hidden features and only outperform deep neural networks by a small margin.

[8] As we will show in the following chapters, if the lower layers of a deep architecture do not have enough hidden variables, there may not be a point in adding more layers.

Summary

- Deep architectures are often capable of representing functions more efficiently and usually benefit from non-local generalization.
- Having less parameters, deep architectures lead to better generalization.
- Using dropout may be a promising way to train deep feed-forward neural networks in a supervised fashion.
- Convolutional networks and tied weights can be very efficient to deal with datasets which are invariant by translation.
- Learning deep representations combines the benefits of learning representations and of deep architectures w.r.t. generalization.
- Layer-wise deep learning has the potential to improve generalization further and to reduce training time by training one layer at a time.
- Stacking RBMs and stacking auto-encoders are the most popular algorithms for layer-wise deep learning.
- Stacking RBMs maximizes a variational lower bound of the log-likelihood of a deep generative model.
- Under the right conditions, maximizing this variational lower bound leads to a guarantee that adding layers improves the likelihood.
- There is not guarantee however that maximizing the variational lower bound leads to an optimal solution.
- Stacking auto-encoders seems to perform very well in practice but lacks theoretical justification.
- A final fine-tuning phase can be used to fine-tune representations either with unsupervised data (deep auto-encoder) or with supervised data (supervised fine-tuning).
- Stacked RBMs and deep auto-encoders already have many variants which can improve performance.
- Deep learning methods have resulted in concrete breakthroughs in recent years, winning several competitions without expert knowledge or feature engineering.
- Deep learning methods have a remarkable potential to solve supervised tasks with small amounts of labeled data or with no labeled data at all.

Having presented the general framework of deep learning and recent advances in the field, we now discuss possible directions of improvement.

6

WHAT CAN WE DO ?

As we have seen, deep learning and more precisely layer-wise deep learning, has led to many impressive results in several settings. Nevertheless, in this thesis, we have to consider what could be improved.

Stacked RBMs seem to have the best justification for layer-wise learning so far. However the maximization of a variational lower bound does not seem to result in the maximization of the likelihood of a full deep generative model since i/ the first layer is not trained to maximize the likelihood of the deep model, and ii/ the guarantee of improvement for the upper layers does not hold for more than two layers. Nevertheless, the layer-wise training principle has tremendous advantages in theory which leads us to ask:

> QUESTION 1: Is it possible to learn an optimal lower layer before learning the upper layers ?

If the answer is yes, then there exists a sound criterion for layer-wise training. In which case we would like to know:

> QUESTION 2: If it is possible to learn an optimal lower layer before learning the upper layers, what is the criterion to maximize at each step ?

If the answer to question 1 is no, then, any layer-wise method is bound to encounter issues such as those encountered in the maximization of the variational lower bound and will probably have some difficulty when scaling to many layers.

An other point concerns the generalization of layer-wise training to models other than RBMs and deep Boltzmann machines. In the case of stacking deep auto-encoders, the current theoretical justification seems to be that auto-encoders learn an approximation of RBMs and that training deep auto-encoders may therefore maximize an approximation of the variational lower bound[1]. This

[1] There are many theoretical justifications for auto-encoders (manifold hypothesis, relation to score-matching), however, here we are interested in a theoretical justification for the layer-wise procedure with many layers. The field is evolving very rapidly on this subject, see (Bengio and Thibodeau-Laufer, 2013).

has led to RBMs and deep Boltzmann machines being the first choices considered when faced with a deep learning problem. Nonetheless, a final step of unsupervised fine-tuning with back-propagation is often considered in practical applications whether the layer-wise method used auto-encoders or stacked RBMs. This leads us to consider the possibility that auto-encoders are in fact quite good at learning deep neural networks and to ask what could justify this performance.

QUESTION 3: Is there a justification for layer-wise deep learning which directly applies to models other than RBMs and deep Boltzmann machines ?

or more precisely in the case of stacked auto-encoders and fine-tuning:

QUESTION 4: Why does the layer-wise training of stacked auto-encoders result in deep learning ?

QUESTION 5: How can we justify the unsupervised fine-tuning of probabilistic models with back-propagation ?

Although probabilistic formulations have led to a better understanding of deep learning, the evaluation of deep generative models should ideally be done w.r.t. the log-likelihood of the deep model which is intractable.

QUESTION 6: Can we find a tractable performance measure for deep generative models ?

Measuring performance can be useful at the end of training to assess whether training is successful, but also during training as part of a model selection procedure. Provided we have a good criterion for training a deep model in a layer-wise manner, model selection could also be done with a layer-wise criterion in theory. This has the potential to greatly reduce the computational cost currently incurred when models are only compared after all their layers have been trained (Bergstra et al., 2011; Bergstra and Bengio, 2012). Additionally, being able to measure the performance at each level would give an empirical measure of what is gained each time a layer is added and could be used to decide when to stop adding layers.

QUESTION 7: Can we evaluate deep architectures in a layer-wise fashion to perform model selection ?

QUESTION 8: Does this lead to a sound criterion to stop adding layers ?

Finally, training each layer is a difficult optimization problem which usually involves millions of parameters or more. The impact of metrics and parametrization on optimization strategies such as gradient descent is well known, thus we ask:

QUESTION 9: What is the impact of metrics on the optimization of each layer ?

Incidentally, if there is an impact, we would be very interested in improving the current optimization methods to reduce computational time. This leads us to our final question, namely:

QUESTION 10: How can we improve layer optimization by considering the impact of metrics and parametrization ?

Having asked these questions, we now turn to the contributions which try to give some answers.

Part III

CONTRIBUTIONS

7

PRESENTATION OF THE FIRST ARTICLE

Ludovic Arnold, Hélène Paugam-Moisy, and Michèle Sebag. Unsupervised layer-wise model selection in deep neural networks. In *19th European Conference on Artificial Intelligence (ECAI 2010)*, Lisbon Portugal, August 2010. 915–920

7.1 Context

In deep learning, model selection poses a difficult problem because the large number of parameters per layer is then multiplied by the number of layers, thus increasing the size of the search space exponentially: an example of the curse of dimensionality.

When considering generative deep neural networks such as stacked RBMs or deep belief networks, the probabilistic perspective has the advantage of providing a comprehensive theoretical background. However the intractability of the likelihood which should be used to measure performance compounds the problem. This problem of model selection and evaluation of performance is in fact much more crucial than it appears. If we are capable of identifying a good criterion for evaluation, one such that better models according to this criterion are better models for a target task, we have found not only a performance measure, but also arguably a valid *training criterion* which we should maximize during training. Hyper-parameter selection can be seen as an integral part of the training process, aimed at maximizing the training criterion further over a larger search space which takes the hyper-parameters into account.

Following on this line of reasoning, we naturally come to ask the following question: If stacked RBMs are trained using a layer-wise criterion, why not evaluate them using the same kind of layer-wise criterion ? This is the subject of this first paper.

7.2 Contributions

Because deep neural networks are trained in a layer-wise fashion, we propose to evaluate them in a similar way, i.e. layer by layer: train several possible layers at each step i.e. with various hyper-parameters and choose the best one before considering training subsequent layers. This strategy, if successful allows the reduction of the search space from exponential to linear in the number of layers.

To evaluate this approach, we must compare it to the alternative approach considered intractable: evaluation of each deep model after it has been fully trained. This involves a very high computational cost which leads us to consider a more limited scope of study. In this regard, determining the optimal topology, i.e. choosing the number of layers and number of neurons per layer is an interesting sub-problem which allows us to test our hypothesis. The layer-wise criterion we propose for evaluating layers in deep belief networks is the reconstruction error, a natural criterion for auto-associators. Since deep auto-encoders can be interpreted under a probabilistic perspective, the reconstruction error must somehow be meaningful for the evaluation of deep generative models.

Although the results support our hypothesis, the mean field criterion we used to train RBMs is a close approximation to that of auto-associators which may limit the generality of the approach. Nevertheless, the study leads to several interesting results. First, the layer-wise selection of the number of neurons with the reconstruction error is successful, giving some substance to the claimed potential benefits of the approach. Second, the results show that upper-layers cannot recover losses resulting from an insufficient number of neurons in the lower layers. This is consistent with the interpretation of each layer as encoding information in terms of the explanatory concepts in the layer below, with higher layers encoding higher order concepts than lower layers. In this analogy, the understanding of a concept is conditional to the understanding of concepts of lower order.

Unsupervised Layer-Wise Model Selection in Deep Neural Networks

Arnold Ludovic[*] Paugam-Moisy Hélène[†] Sebag Michèle[‡]

June 11, 2013

Abstract

Deep Neural Networks (DNN) propose a new and efficient ML architecture based on the layer-wise building of several representation layers. A critical issue for DNNs remains model selection, e.g. selecting the number of neurons in each DNN layer. The hyper-parameter search space exponentially increases with the number of layers, making the popular grid search-based approach used for finding good hyper-parameter values intractable. The question investigated in this paper is whether the unsupervised, layer-wise methodology used to train a DNN can be extended to model selection as well. The proposed approach, considering an unsupervised criterion, empirically examines whether model selection is a modular optimization problem, and can be tackled in a layer-wise manner. Preliminary results on the MNIST data set suggest the answer is positive. Further, some unexpected results regarding the optimal size of layers depending on the training process, are reported and discussed.

1 INTRODUCTION

The general question of model selection − including the selection of hyper-parameter values for a Machine Learning (ML) algorithm − remains at the core of the Statistics and Machine Learning studies (Efron and Tibshirani, 1993). From a practitioner viewpoint, the best practice relies on variants of the cross-validation procedure (Dietterich, 1998): one should select the model and hyper-parameter setting yielding the best performance on average. From a theoretical viewpoint, although some intrinsic limitations of cross-validation have been pointed out in the ML literature, these appear to be negligible (Bengio and Grandvalet, 2004) comparatively to methodological errors (Hastie et al., 2001). From a computational viewpoint, one generally proceeds by exploring the hyper-parameter space using a grid search, possibly using racing-based approaches in order to decrease the overall computational cost (Mnih et al., 2008). Overall, theoreticians and practitioners would likely join and agree that the fewer hyper-parameters, the better.

[*] Université Paris Sud 11 – CNRS, LIMSI, Ludovic.Arnold@lri.fr

[†] Université de Lyon, TAO – INRIA Saclay, Helene.Paugam-Moisy@univ-lyon2.fr

[‡] TAO – INRIA Saclay, CNRS, LRI, Michele.Sebag@lri.fr

A new ML field, Deep Networks (Bengio et al., 2007; Hinton et al., 2006) however seems to go against such a parameter-light trend. The main claim behind Deep Networks can be schematized as: several levels of representations, stacked on top of each other, are required to represent complex concepts in a tractable way; a single-level representation, though in principle able to do the job, will not make it in practice. While the greater expressiveness and compactness obtained through the composition of representations had been known for decades, deep architectures were discarded as they could not be trained in an efficient way. The training bottleneck of deep architectures was overcome through an original, sequential approach (Bengio et al., 2007; Hinton et al., 2006) aimed at the greedy optimization of a seemingly irrelevant criterion : while the goal of a Deep Network is to achieve supervised learning, each layer is pre-trained using unsupervised learning criteria (more in section 2).

The issue of DNN hyper-parameter selection however remains critical, as the number of hyper-parameters (e.g. for each layer: number of neurons and learning rate) linearly increases with the number of layers, exponentially increasing the cost of a grid search.

This paper investigates whether the "Unsupervised learning first!" principle at the root of DNNs can be applied to hyper-parameter selection too. Accordingly, an unsupervised criterion based on the Reconstruction Error is proposed. The underlying question is whether hyper-parameter selection is a *modular* optimization problem, meaning that the optimal overall parameter setting can be obtained by i/ finding the optimal setting for layer 1; ii/ iteratively finding the optimal setting for layer $i + 1$, conditionally to the hyper-parameter values for layers $1 \dots i$.

The experimental validation of the approach for Restricted Boltzmann Machines (trained with Mean Field Contrastive Divergence (Welling and Hinton, 2002) on the MNIST dataset) suggests a positive answer to the modularity question (section 4). Furthermore, some unexpected findings, concerning the optimal size of the representation w.r.t the number of gradient updates of the training procedure, are reported and discussed, raising new questions for further study.

2 DEEP NEURAL NETWORKS

For the sake of completeness, this section introduces Deep Neural Networks, focusing on the Restricted Boltzmann Machine and Auto-Associator approaches. The interested reader is referred to (Hinton et al., 2006; Ackley et al., 1985; Larochelle et al., 2009) for a comprehensive presentation.

2.1 Restricted Boltzmann Machine (RBM)

While a Boltzmann Machine is a completely connected network made of a bag of visible and hidden units, Restricted Boltzmann Machines (RBMs) only involve connections between visible units on the one hand and hidden units on the other hand (Fig. 1).

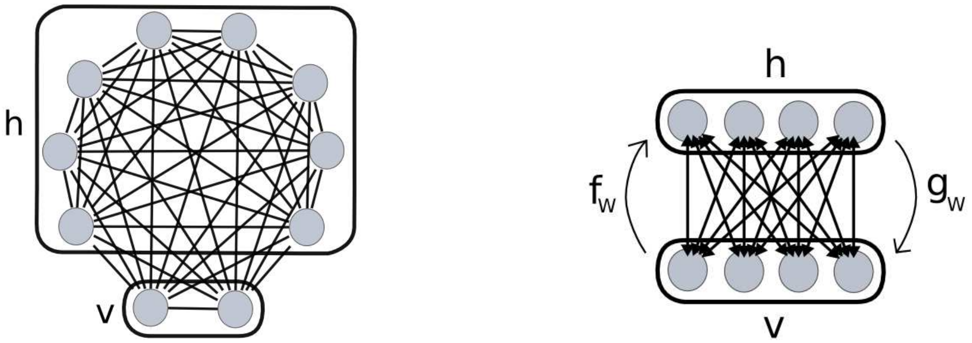


Figure 1: Left: Architecture of a Boltzmann Machine. Right: A Restricted Boltzmann Machine.

Let us denote $\mathbf{v} = v_1, \ldots, v_q$ (respectively $\mathbf{h} = h_1, \ldots, h_r$) the set of visible (resp. hidden) units. For notational simplicity, it is assumed that both the visible and the hidden layers involve a bias unit always set to 1. An RBM is described from its set of weights $\mathbf{W} \in \mathbb{R}^{q \times r}$, where w_{ij} stands for the weight on the connection between v_i and h_j. All units are boolean. Each visible unit v_i encodes the i-th domain attribute, while each hidden unit h_j encodes a hypothesis. Formally, the so-called energy of an RBM state $(\mathbf{v}, \mathbf{h})$ is defined as:

$$Energy(\mathbf{v}, \mathbf{h}) \;=\; -\mathbf{h}^\top \mathbf{W} \mathbf{v} \tag{1}$$

An RBM can be interpreted as a constraint satisfaction network, where the weight w_{ij} reflects the correlation between v_i and h_j (if $w_{ij} > 0$ a lower energy is obtained for $v_i = h_j$ everything else being equal). As such, an RBM induces a joint probability measure on the space of RBM states:

$$P_{\mathbf{W}}(\mathbf{v}, \mathbf{h}) = \frac{e^{-Energy(\mathbf{v},\mathbf{h})}}{Z} \tag{2}$$

where Z denotes as usual the normalization factor. Simple calculations show that visible (resp. hidden) units are independent conditionally to the hidden (resp. visible) units, and the conditional probabilities can be expressed as follows, where $sgm(t) = \frac{1}{1+e^{-t}}$:

$$P_{\mathbf{W}}(h_i | \mathbf{v}) = sgm(\sum_j w_{ij} v_j) \quad 1 \leq i \leq q \tag{3}$$

$$P_{\mathbf{W}}(v_j | \mathbf{h}) = sgm(\sum_i w_{ij} h_i) \quad 1 \leq j \leq r \tag{4}$$

An RBM thus defines a probabilistic generative model: Considering a uniform distribution on the visible units, a probability distribution on the hidden units is derived (Eq. (3)), which enables in turn to derive a probability distribution on the visible units (Eq. (4)) and this process can be iterated after the so-called Gibbs sampling (Monte-Carlo Markov Chain), converging toward $P_{\mathbf{W}}$. Let us consider the probability distribution P_D, defined from the empirical data D by taking a uniform sample $\mathbf{v}$ in D, and sampling $\mathbf{h}$ after $P_{\mathbf{W}}(\mathbf{h}|\mathbf{v})$. Intuitively, the RBM model best fitting the data is such that $P_{\mathbf{W}} = P_D$ (a sample $\mathbf{v}$ is equally likely generated after $P_{\mathbf{W}}$ or by uniformly sampling D).

Accordingly, an RBM is trained by minimizing the Kullback Leibler divergence $KL(P_D||P_{\mathbf{W}})$, or a tractable approximation thereof, the *Contrastive Divergence* (Hinton., 2002). Contrastive divergence can itself be approximated using a Mean Field approach (Welling and Hinton, 2002), yielding a deterministic and faster learning procedure, albeit with higher risk of overfitting.

2.2 Stacked RBMs

After the Deep Network principles (Hinton et al., 2006), stacked RBMs (SRBMs) are built in a layer-wise manner (Fig. 2). The first layer $\mathbf{h}$ is built from the training set and the visible units $\mathbf{v}$ as explained above, and the i-th layer RBM is built using the same approach, with the hidden units $\mathbf{h}_{i-1}$ in the $i-1$-th layer being used as the new RBM's visible layer.

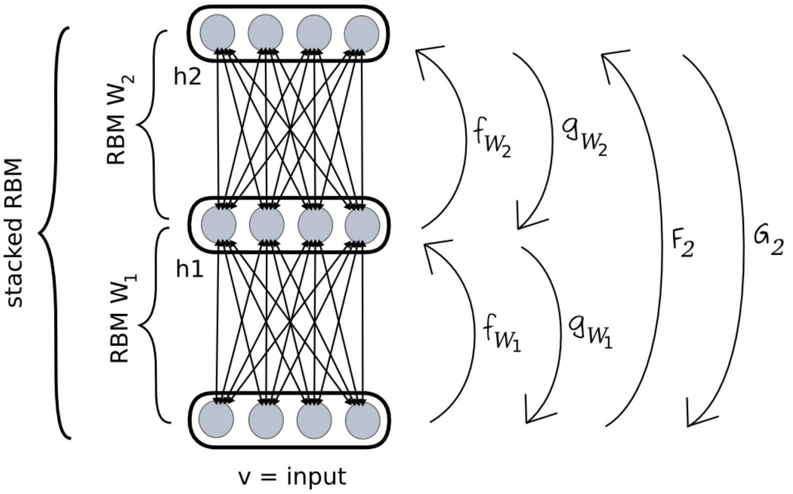


Figure 2: A Deep Architecture: Stacked RBMs

The rationale for iterating the RBM construction is that trained hidden units $\mathbf{h}_i$ are not independent; rather, they are independent conditionally to $\mathbf{v}$. A more refined generative model can thus in principle be defined by capturing the correlations between the hidden units $\mathbf{h}$, via a second RBM layer. More generally, the i-th layer in a stacked RBM aims at modelling the correlations between the hidden units in the previous layer. Letting $\mathbf{W}_1, \ldots \mathbf{W}_\ell$ denote the RBM parameters for layers $1 \ldots \ell$, with $\mathbf{h}_0 = \mathbf{v}$, comes the equation:

$$P(\mathbf{v}) = \sum_{\mathbf{h^1} \ldots \mathbf{h}^\ell} P_{\mathbf{W}_1}(\mathbf{v}|\mathbf{h^1}) P_{\mathbf{W}_2}(\mathbf{h^1}|\mathbf{h^2}) \ldots P_{\mathbf{W}_\ell}(\mathbf{h}^\ell) \tag{5}$$

Let $f_{\mathbf{W}}$ (respectively $g_{\mathbf{W}}$) denote the forward propagation of an input to the hidden layer according to $P_{\mathbf{W}}(\mathbf{h}|\mathbf{v})$ (resp. the backward propagation from the hidden layer from the input according to $P_{\mathbf{W}}(\mathbf{v}|\mathbf{h})$). Considering a ℓ-layer RBM with weights $\mathbf{W}_1, \ldots \mathbf{W}_\ell$ (Fig. 2), F_ℓ and G_ℓ are respectively defined as the forward propagation of the input to the ℓ-th layer, and the backward propagation from the ℓ-th layer to the input:

$$F_\ell = f_{\mathbf{W}_\ell} \circ \cdots \circ f_{\mathbf{W}_1} \qquad G_\ell = g_{\mathbf{W}_1} \circ \cdots \circ g_{\mathbf{W}_\ell} \tag{6}$$

2.3 Stacked Auto-Associators

Deep Neural Networks can also be built by stacking auto-associators (Larochelle et al., 2009). An auto-associator is a 1-hidden layer neural network aimed at reproducing its input; formally, it uses back propagation to minimize the Reconstruction error defined as

$$\sum_{\mathbf{x} \in D} ||x - \Phi(x)||^2$$

where Φ is the function corresponding to forward propagation through the network.

Stacking Auto-Associators proceeds by setting the i-th DNN layer to the hidden layer of the Auto-Associator built from the $(i-1)$-th DNN layer.

3 UNSUPERVISED MODEL SELECTION

The model selection approach is inspired from the DNN unsupervised layer-wise methodology. After discussing the position of the problem, this section describes an unsupervised criterion supporting the model selection task. The presented approach focuses on finding the optimal number of neurons in each layer of a stacked RBM. The choice of a SRBM architecture is motivated by their good empirical results and their strong theoretical background (Hinton et al., 2006; Larochelle et al., 2009, 2007).

3.1 Position of the problem

The goal is to optimize the size n_i of the i-th layer $\mathcal{L}_i$ conditionally to the size of the layers $1 \ldots i-1$; in this manner, model selection is brought to a sequence of ℓ scalar optimization problems, as opposed to a single optimization problem in $\mathbb{N}^\ell$.

A first question regards the type of optimization criterion to be used. After (Bengio et al., 2007), it is better that supervised learning criteria only be considered at a late stage when learning a deep architecture. Extensive empirical evidence suggests that unsupervised learning actually regularizes the supervised optimization problem (Erhan et al., 2009). The proposed model selection approach will accordingly be based on an unsupervised criterion.

Another question raised by a layer-wise approach regards its consistency, aka the modularity of the model selection problem. Formally, the question is whether the global optimum of the considered criterion is found with a layer-wise sequential optimization approach. Lastly and most importantly, the question is whether the layer-wise unsupervised optimization approach eventually enforces good performances w.r.t. supervised learning.

3.2 Reconstruction Error

A most natural unsupervised criterion relevant to RBMs is the log-likelihood of test data under the model. This criterion would however require computing the normalization factor in Eq. (2), which is intractable in the general case. An alternative is to consider the *Reconstruction Error* inspired from the auto-associator approach (section 2.3). Formally, the Reconstruction Error of an RBM is computed by clamping $\mathbf{v}$ to some data sample $\mathbf{x}$, computing the (real-valued) hidden unit configuration after $P(\mathbf{h}|\mathbf{v})$, backpropagating this hidden configuration onto the visible units, and computing the square Euclidean distance between $\mathbf{x}$ and the visible unit configuration obtained. For the sake of computational tractability, the Mean Field approximation is used.

With same notations as in section 2.2, let $\mathbf{W}^*_\theta$ denote the weights of the RBM trained from the dataset D using the hyper-parameters θ; then the associated Reconstruction Error is defined as:

$$\mathcal{L}(D,\theta) =_{def} \mathcal{L}(D,\mathbf{W}^*_\theta) = \sum_{\mathbf{x}\in D} ||\mathbf{x} - g_{\mathbf{W}^*_\theta} \circ f_{\mathbf{W}^*_\theta}(\mathbf{x})||^2 \tag{7}$$

The Reconstruction Error corresponding to the whole ℓ-layer RBM can be directly defined as:

$$\mathcal{L}(D,\mathbf{W}_1,\ldots\mathbf{W}_\ell) = \sum_{\mathbf{x}\in D} ||\mathbf{x} - G_\ell \circ F_\ell(\mathbf{x})||^2 \tag{8}$$

For reference, examples of reconstructed digits from the MNIST dataset are given in Fig.3.

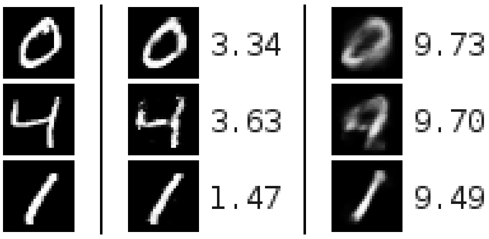


Figure 3: Examples of reconstructed digits from the MNIST dataset with an RBM trained on 60,000 examples for 1 epoch. Left: Original image. Middle: Reconstruction with a 300 hidden units RBM. Right: Reconstruction with a 10 hidden units RBM.

3.3 Optimum selection

How to use the reconstruction error to select the optimal number of neurons in each DNN layer raises the parsimony *vs* accuracy tradeoff. After (Le Roux and Bengio, 2008), the reconstruction error should decrease with the number of neurons, which suggests the use of a regularization term. Another possibility is to wait until the reconstruction error does not decrease any more (plateau).

4 EXPERIMENTAL VALIDATION

This section reports on the experimental validation of the proposed unsupervised layer-wise approach for hyper-parameter selection in stacked RBMs and discusses possible extensions.

4.1 Goals of experiments

The goal of the experiments is to answer the following questions:
Feasibility: Does the considered unsupervised criterion clearly and steadily support some selection of the optimal number of neurons ?
Stability: Does the proposed procedure offer some stability w.r.t. the experimental setting (number of samples, number of epochs) ?
Efficiency: How does the model selected in an unsupervised layer-wise manner affect the supervised classification accuracy ?
Consistency: Does the model globally optimized over ℓ layers coincide with the layer-wise optimization of the size of each layer ?
Generality: A last question concerns the extension of the proposed approach to RBMs trained with plain Contrastive Divergence, and Auto-Associators.

4.2 Experimental setting

The experiments consider the MNIST dataset including $60,000$ 28×28 images representing digits from 0 to 9 in grey level, intensively used in the DNN literature[1] (Hinton et al., 2006; Larochelle et al., 2009). The unsupervised learning stage considers $1,000$, $10,000$ or $60,000$ examples. A disjoint test set including 1,000 examples is used to assess the (supervised or unsupervised) generalization performance.

For the sake of computational tractability, the experimental validation was limited to a 2-layer RBM setting. The above experiment goals were investigated in this restricted experimental setting, based on a grid-search systematic exploration of the first and second layer size. The overall computational effort is 300 days CPU time on a 1.8GHz Opteron processor. Using multiple cores, several models were trained in parallel making the total training time about 15 days.

4.3 Feasibility and stability

The Reconstruction Error on the MNIST test set for the first RBM is reported vs the number of neurons (in log scale) in Fig.4. As expected, the Reconstruction Error is a monotonically decreasing function of the number of neurons. The **feasibility** of the approach however is empirically established as the Reconstruction Error displays a plateau when the number of neurons increases[2].

[1] Mean Field Contrastive Divergence is used in the unsupervised phase, and backpropagation with momentum is used in the supervised phase. The algorithm is available upon request.

[2] Experiments conducted on the cifar-10 dataset likewise show a decreasing reconstruction error as the number of neurons increases. Due to the higher complexity of the cifar-10 dataset comparatively to MNIST, the plateau however was not reached for the largest considered networks (up to 12,000 neurons).

After these results, the Reconstruction Error criterion suggests a clear and stable hyper-parameter selection of $n_1 \geq n_1^* = 300$ neurons.

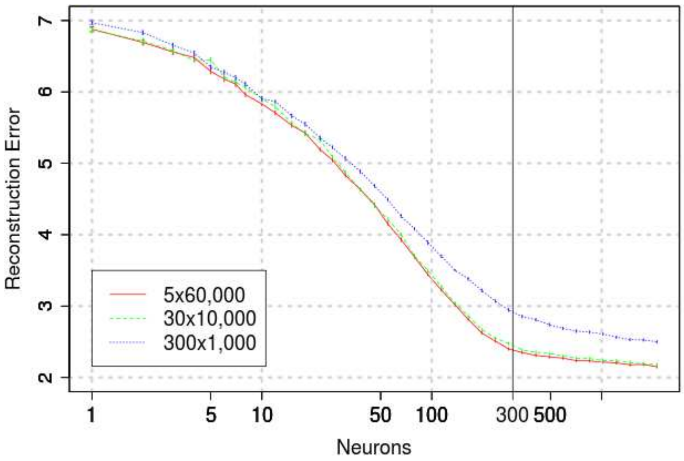


Figure 4: Reconstruction Error for different numbers of epochs and sizes of the training set (with the same overall number of gradient updates) in the first layer of a SRBM.

Following the proposed layer-wise approach (section 3.1), we now optimize the size of the second layer, conditionally to the selected size of the first layer. Setting the size of the first layer to $n_1 = 300$, the selection of the optimal number of neurons in the second layer after the Reconstruction Error criterion (Fig. 5), was conducted in the same way as above. Once again, the experimental results show a plateau for the Reconstruction Error criterion after a certain threshold value. The optimal number of neurons on the second layer conditionally to $n_1 = 300$ is obtained for $n_2 \geq n_2^* = 200$.

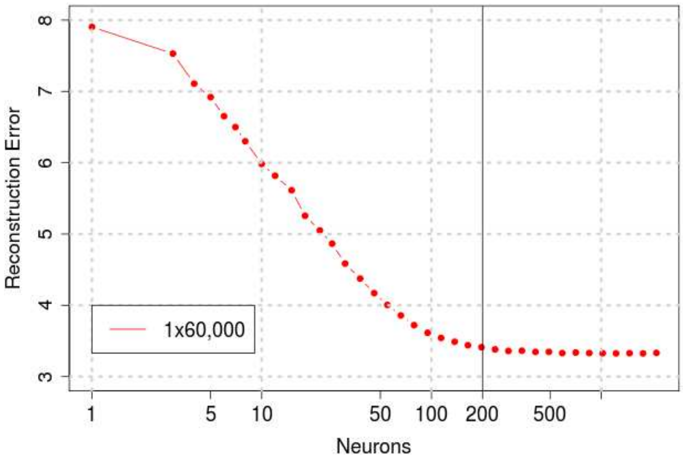


Figure 5: Reconstruction Error vs number of neurons in the second-layer of a SRBM, with $n_1 = 300$ neurons on the first layer.

The **stability** of the criterion with respect to experimental settings is illustrated on Fig. 4, which shows the Reconstruction Error for three different sizes of the training set and same number of gradient updates. This confirms that the optimal layer size does not depend on the size of the training set, conditionally to the number of gradient updates; this point will be discussed further in section 4.6.

4.4 Efficiency and consistency

The **efficiency** of the presented approach must eventually be assessed with the classification accuracy. In the previous experiments, RBMs with different hidden layer sizes were trained and evaluated against the Reconstruction Error criterion only. In order to assess their classification performance, a last layer with 10 output neurons is built on the top of the first RBM-layer (with weights uniformly initialized in $[-1, 1]$), and a backpropagation algorithm with early stopping is applied on the whole network (Bengio et al., 2007). Fig. 6 depicts the classification accuracy versus the number of neurons in the hidden layer. As shown in Fig. 7, the classification accuracy increases as the Reconstruction Error decreases. Overall, the unsupervised criterion used for model selection yields same performance as that of the best 1-hidden layer neural networks in the literature (1.9%) (Bengio et al., 2007).

A last question regards the **consistency** of the modular approach, that is, whether simultaneously optimizing several layers yields the same result as sequentially optimizing each layer conditionally to the optimal setting for the previous layers. The global Reconstruction Error on a 2-layer RBM is depicted in Fig. 8, where the first (resp. second) axis stands for the number of neurons in the first (resp. second) layer.

As could have been expected, the Reconstruction Error decreases as the number of units increases; and it is shown to plateau after a sufficient number of units in each layer. Furthermore, one cannot compensate for the insufficient number of units in layer 1 by increasing the number of neurons in layer 2. This behavior is unexpected in the perspective of the mainstream statistical learning theory (Vapnik, 1998), where the VC-dimension of the model space increases with the number of weights in the network, whatever the distribution of the neurons on the different layers, as confirmed empirically in the multi-layer perceptron framework (Paugam-Moisy, 1993).

The Reconstruction Error isolines are approximately rectangular-shaped: for a given number of units n_1 on the first layer, there exists a best Reconstruction Error reached for n_2 greater to a minimal value, referred to as $n_2^*(n_1)$. Likewise, the best Reconstruction Error for a given number of units n_2 on the second layer, is reached whenever the number of hidden units on the first layer is sufficient. Overall, the optimal Reconstruction Error w.r.t. the simultaneous optimization of n_1 and n_2, given in Fig. 8, leads to the same optimal setting as the layer-wise procedure in Fig.4 and Fig.5 ($(n_1, n_2)^* = (n_1^*, n_2^*(n_1^*))$), which empirically confirms the modularity of the optimization problem.

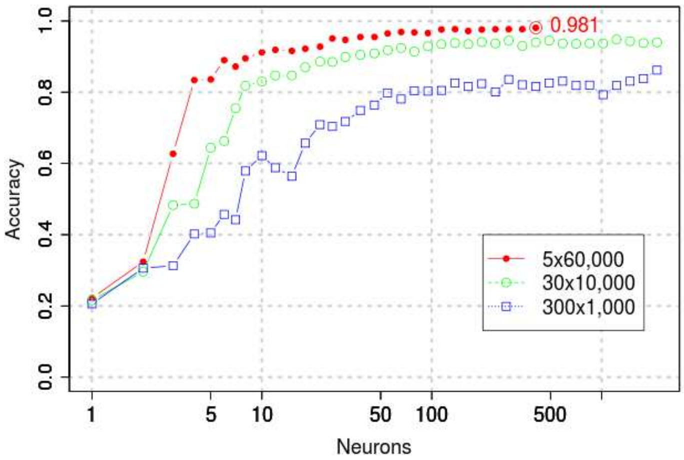


Figure 6: Classification accuracy *vs* number of neurons for different dataset sizes and numbers of epochs.

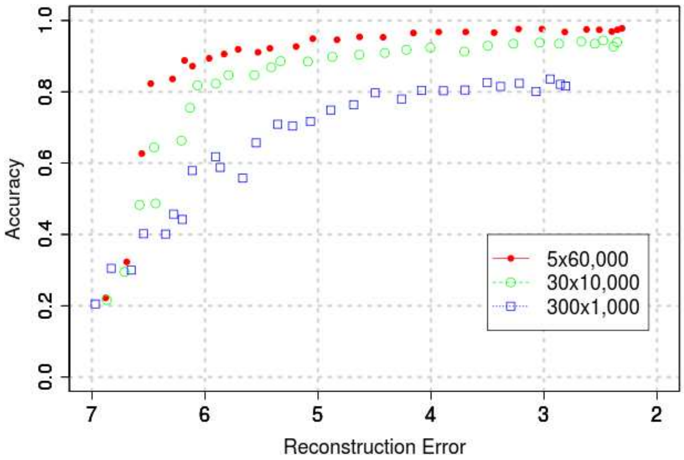


Figure 7: Classification accuracy *vs* Reconstruction Error (the higher and the rightmost the better).

4.5 Generality

Complementary experiments show that the presented approach hardly applies when considering Contrastive Divergence (instead of Mean Field); on the one hand, the Reconstruction Error very slowly decreases as the number of neurons increases, making the plateau detection computationally expensive. Furthermore, the Reconstruction Error displays a high variance between runs due to the stochastic nature of the learning procedure.

In the meanwhile, the same approach was investigated in the Auto-Associator (AA) framework, which naturally considers the Reconstruction Error as training criterion. Fig. 9 shows the AA Reconstruction Error on the MNIST dataset *vs* the number of neurons. Interestingly, the optimal layer size is larger than in the RBM case; interpreting this fact is left for further work.

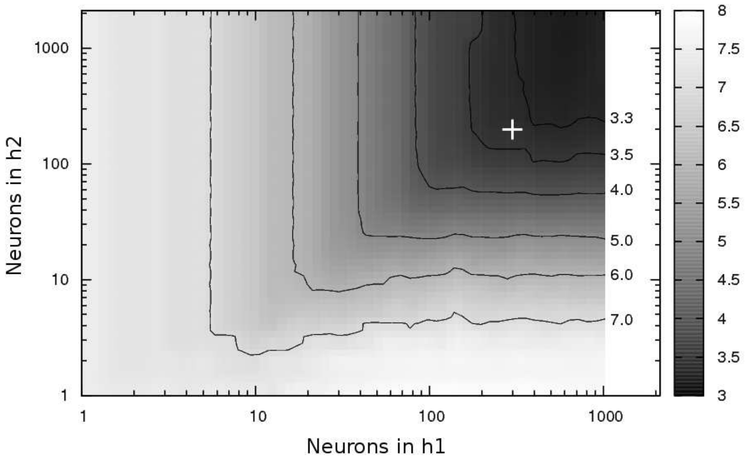


Figure 8: Overall Reconstruction Error in a 2-layer RBM *vs* the number of units in first layer (horizontal axis) and second layer (vertical axis). The white cross shows the optimal configuration found with the layer-wise procedure (300,200).

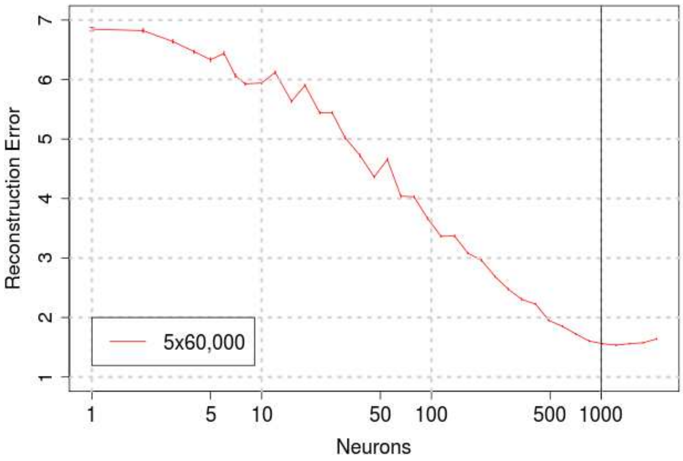


Figure 9: Reconstruction Error vs number of neurons for an Auto-Associator.

4.6 Model selection and training process

Intermediate experiments, aimed at reducing the computational cost and memory resources involved in the experiment campaign, led to consider datasets with sizes 60,000, 10,000 and 1,000 (Fig. 10) trained for 1 epoch. Quite interestingly, for the Reconstruction Error criterion, the optimal number of hidden units *decreases* with smaller datasets.

A second experiment was launched to see if the above results could be attributed to the increased diversity of the bigger datasets, or to the increased number of gradient updates (granted that 1 epoch was used for every dataset in Fig. 10).

For two datasets of 1,000 and 60,000 samples, the Reconstruction Error is evaluated at multiple points in the training process, each separated by 1,000

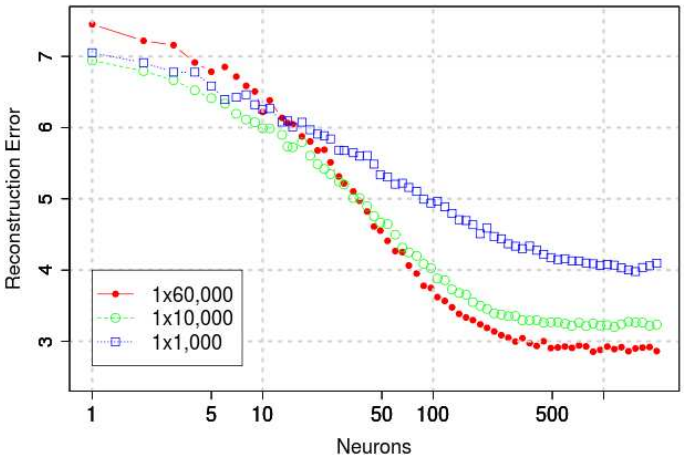


Figure 10: Reconstruction Error vs the number of hidden units in a 1-layer RBM, depending on the size of the training set.

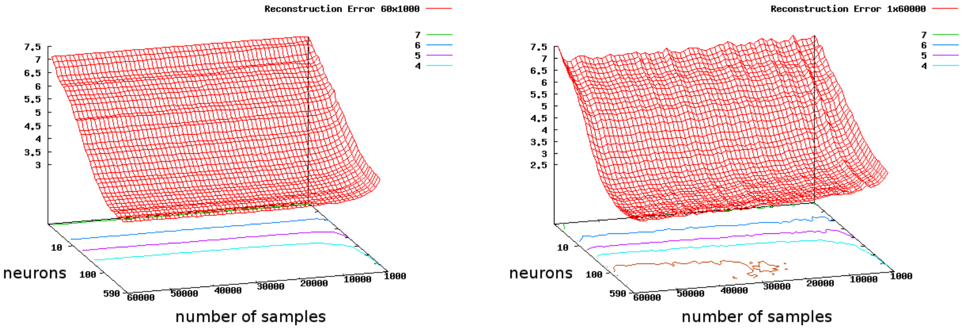


Figure 11: Reconstruction Error for RBM trained on 60x1,000 (Left) and 1x60,000 examples (Right) as a function of the number of neurons and gradient updates.

gradient updates. The results (Fig.11) show the Reconstruction Error w.r.t. the number of neurons for the two datasets as the training progresses from right to left. The Reconstruction Error isolines clearly show the decreasing number of neurons needed with the increasing number of gradient updates.

A tentative interpretation for this fact goes as follows. Firstly, the above results state that the optimal model size can decrease as training goes on. Secondly, a main claim at the core of DNNs (Bengio and LeCun, 2007; Hinton and Salakhutdinov., 2006) is that they capture a more abstract and powerful description of the data domain than shallow networks. Therefore it is conjectured that the network gradually becomes more able to construct key features as training goes on. Further work will aim at investigating experimentally this conjecture, by examining the features generated in the deep layers depending on the input distribution, as done by (Lee et al., 2009).

5 Conclusion and Perspectives

The contribution of the paper is twofold. Firstly, an unsupervised layer-wise approach has been proposed to achieve model selection in Deep Networks. The selection of hidden layer sizes is a critical issue potentially hindering the large scale deployment of DNN. A frugal unsupervised criterion, the Reconstruction Error, has been proposed. A proof of principle for the feasibility and stability of Reconstruction Error-based Model Selection has been experimentally given on the MNIST dataset, based on an extensive experiment campaign. The merits of the approach have also been demonstrated from a supervised viewpoint, considering the predictive accuracy in classification for the supervised DNN learned after the unsupervised layer-wise parameter setting.

After these results, the model selection tasks related to the different layers can be tackled in a modular way, iteratively optimizing each layer conditionally to the previous ones. This result is unexpected in the perspective of standard Neural Nets, where the complexity of the model is dominated by the mere size of the weight vector. Quite the contrary, it seems that deep networks actually depend on the sequential acquisition of different "skills", or representational primitives. If some skills have not been acquired at some stage, these can hardly be compensated at a later stage.

Lastly, the dependency between the model selection task and the training effort has been investigated. Experimental results suggest that more intensive training efforts lead to a more parsimonious model. Further work will investigate in more depth these findings, specifically examining the properties of abstraction of the hidden layers in an Information Theoretical perspective and taking inspiration from (Lee et al., 2009). Along the same lines, the choice of the examples (curriculum learning (Bengio et al., 2009)) used to train the RBM, will be investigated w.r.t. the unsupervised quality of the hidden units: The challenge would be to define an intrinsic, unsupervised, measure to order the examples and construct a curriculum.

Acknowlegements

This work was supported by the French ANR as part of the ASAP project under grant ANR_09_EMER_001_04. The authors gratefully acknowledge the support of the PASCAL2 Network of Excellence (IST-2007-216886).

References

D.H. Ackley, G.E. Hinton, and T.J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 5:1089–1105, 2004.

Y. Bengio and Y. LeCun. Scaling learning algorithms towards ai. In *Large-Scale Kernel Machines*. MIT Press, 2007.

Y. Bengio, P. Lamblin, V. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, Cambridge, MA, 2007.

Yoshua Bengio, Jerome Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*, 2009.

T.G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 1998.

B. Efron and R. Tibshirani. *An introduction to the bootstrap*, volume 57 of *Monographs on Statistic and Applied Probability.* Chapman & Hall, 1993.

Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, 2009.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics, 2001.

G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

G.E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

G.E. Hinton, S. Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Conputation*, 18:1527–1554, 2006.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 473–480, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3.

H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40, 2009.

Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural Computation*, 20(6):1631–1649, 2008.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, page 77, 2009.

V. Mnih, Cs. Szepesvari, and J.-Y. Audibert. Empirical bernstein stopping. In *25th International Conference on Machine Learning (ICML)*, 2008.

H. Paugam-Moisy. Parallel neural computing based on network duplicating. In I. Pitas, editor, *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*, pages 305–340. John Wiley, 1993.

V. N. Vapnik. *Statistical Learning Theory.* Wiley, 1998.

M. Welling and G.E. Hinton. A new learning algorithm for mean field boltzmann machines. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, 2002.

7.3 Discussion

This paper, the first of the author ever published, is understandably of a lesser quality than his subsequent work. It is nonetheless a necessary “first step” in the understanding of layer-wise training and evaluation for deep architectures.

A first result concerns the possibility to perform layer-wise model selection (Question 7). Although the generality of the approach may be questioned because of the mean-field training procedure, the approach supports the hypothesis that layer-wise evaluation with the reconstruction error is possible and considerably less expensive than evaluating networks after training all layers.

An even more interesting conclusion concerns the possibility to train lower layers first (Questions 1 and 2). Namely, it seems that in practice, there is such as thing as a “better lower layer” independently of what layers are added next. To be more precise, even though the reconstruction error of a lower layer cannot be used as a guarantee of performance for the whole network because subsequent layers may be poorly trained, it seems to maximize the *possible performance* of the future network.

This will prove to be a useful insight in the next paper: *Layer-wise learning of deep generative models*, which gives a comprehensive theoretical and empirical justification for a new criterion close to the reconstruction error which can be used for the layer-wise training of deep architectures.

8

PRESENTATION OF THE SECOND ARTICLE

L. Arnold and Y. Ollivier. Layer-wise learning of deep generative models. Technical report, ArXiv e-prints, December 2012. URL http://arxiv.org/abs/1212.1524 (submitted for publication, February 2013).

8.1 Context

Just as in the preceding paper, we pose the question of layer-wise evaluation and model selection. However, the main concern is the justification of layer-wise training for deep architectures.

Training methods for deep architectures usually fall into one of the following categories:

Method 1: Maximizing performance *in a supervised setting* with e.g. multi-layer neural networks. Usually considered intractable when the number of layers is too large but recently shown to be possible (Hinton et al., 2012; Bengio and Glorot, 2010; Ciresan et al., 2010). Very successful in domains where it is possible to encode invariances in the model as in e.g. convolutional neural networks (LeCun and Bengio, 1995). This method could arguably be used for arbitrary model sizes but in practice, it is expected to pose an increasingly difficult optimization problem.

Method 2: Maximizing *the likelihood* of a deep generative model. This approach should lead to optimal latent variables, but it is intractable and must therefore be approximated.

Method 3: Maximizing *a variational lower-bound of the likelihood* of a deep generative model. This method is applied to (pre-)train almost all deep generative models so far: stacked RBMs (Hinton et al., 2006; Bengio et al., 2007), stacked auto-associators (Vincent et al., 2008) since they approximate RBM training (Bengio and Delalleau, 2009), deep Boltzmann machines (Salakhutdinov and Hinton, 2009a) and other variations. Very successful in practice. Applicable in theory to any probabilistic model with latent variable which are not independent. This is in fact an approximation of method 2 (maximizing the likelihood of a deep generative model) with a variational lower bound. This comes with a theoretical guarantee that adding a layer on top of a shallow model can only increase the likelihood of the deep model. However, the guarantee breaks down for three layers or more and adding layers breaks down the inference mechanism.

Note that Method 2 being intractable, an essential question of deep learning is to ascertain if layer-wise learning which is the principle behind methods 3 is sound, i.e. : Can the maximum likelihood estimate of a deep generative model be found by maximizing a layer-wise criterion ?

However, for this to be possible, we need a layer-wise criterion which has to represent some notion of what it means for a layer to be optimal *before subsequent layers are trained*. If no such criterion exists, then the optimal lower layers always depend on the choice of upper layers and layer-wise learning is fundamentally flawed.

8.2 Contributions

In the following paper, we propose a criterion similar to the one suggested in (Le Roux and Bengio, 2008), which satisfies the above requirements and can in theory be applied to find an optimal solution to a deep learning problem in a layer-wise fashion.

This criterion: the *Best Latent Marginal (BLM) upper-bound*, comes with a guarantee of optimality for lower layers (Theorem 1) provided the rest of the training goes well. Namely, the *BLM* upper bound represents the maximum attainable log-likelihood for a given lower layer, while the upper part of the deep generative model is left unspecified. Assuming the BLM upper bound has been successfully maximized, the problem is then transfered to the upper layers in a similar way to Method 3 presented above.

When training of the upper layers is imperfect, as can be the case in practice, the overall error admits an upper bound which is exactly the criterion to be optimized for upper layers. An unintended result is the importance of having different parameters for the generative and inference parts of the model, and to allow the latter to be as rich as possible. This approach has a tight relationship with stacked RBMs and auto-encoders which is explored in detail. Importantly, the BLM upper bound provides a probabilistic justification for stacking auto-associators.

Experiments on two distinct deep datasets confirm the practicality of the approach for the layer-wise learning of deep generative models and for layer-wise model selection.

Layer-wise training of deep generative models

Ludovic Arnold ludovic.arnold@lri.fr
CNRS, LRI/LIMSI, INRIA Saclay – TAO
Bât. 490, Université Paris-Sud
91405 Orsay, France

Yann Ollivier yann.ollivier@lri.fr
CNRS & Université Paris-Sud, LRI, INRIA Saclay – TAO
Bât. 490, Université Paris-Sud
91405 Orsay, France

Editor: unknown

Abstract

When using deep, multi-layered architectures to build generative models of data, it is difficult to train all layers at once. We propose a layer-wise training procedure admitting a performance guarantee compared to the global optimum. It is based on an optimistic proxy of future performance, the *best latent marginal.* We interpret auto-encoders in this setting as generative models, by showing that they train a lower bound of this criterion. We test the new learning procedure against a state of the art method (stacked RBMs), and find it to improve performance. Both theory and experiments highlight the importance, when training deep architectures, of using an inference model (from data to hidden variables) richer than the generative model (from hidden variables to data).

Keywords: Deep Learning, Deep Belief Networks, Deep Neural Networks, Stacked Restricted Boltzmann Machines, Autoencoders

Introduction

Deep architectures, such as multiple-layer neural networks, have recently been the object of a lot of interest and have been shown to provide state-of-the-art performance on many problems (Bengio et al., 2012). A key aspect of deep learning is to help in learning better representations of the data, thus reducing the need for hand-crafted features, a very time-consuming process requiring expert knowledge.

Due to the difficulty of training a whole deep network at once, a so-called *layer-wise* procedure is used as an approximation (Hinton et al., 2006; Bengio et al., 2007). However, a long-standing issue is the justification of this layer-wise training: although the method has shown its merits in practice, theoretical justifications fall somewhat short of expectations. A frequently cited result (Hinton et al., 2006) is a proof that adding layers increases a so-called variational lower bound on the log-likelihood of the model, and therefore that adding layers *can* improve performance.

We reflect on the validity of layer-wise training procedures, and discuss in what way and with what assumptions they can be construed as being equivalent to the non-layer-wise, that is, whole-network, training. This leads us to a new approach for training deep generative models, using a new criterion for optimizing each layer starting from the bottom and for transferring the problem upwards to the next layer. Under the right conditions, this new layer-wise approach is equivalent to optimizing the log-likelihood of the full deep generative model (Theorem 1).

As a first step, in Section 1 we re-introduce the general form of deep generative models, and derive the gradient of the log-likelihood for deep models. This gradient is seldom ever considered because it is considered intractable and requires sampling from complex distributions. Hence the need for a simpler, layer-wise training procedure.

We then show (Section 2.1) how an optimistic criterion, the *BLM upper bound*, can be used to train *optimal* lower layers provided subsequent training of upper layers is successful, and discuss what criterion to use to transfer the learning problem to the upper layers.

This leads to a discussion of the relation of this procedure with stacked restricted Boltzmann machines (SRBMs) and auto-encoders (Sections 2.3 and 2.4), in which a new justification is found for auto-encoders as optimizing the lower part of a deep generative model.

In Section 2.7 we spell out the theoretical advantages of using a model for the hidden variable $\mathbf{h}$ having the form $Q(\mathbf{h}) = q(\mathbf{h}|\mathbf{x})P_{\text{data}}(\mathbf{x})$ when looking for hidden-variable generative models of the data $\mathbf{x}$, a scheme close to that of auto-encoders.

Finally, we discuss new applications and perform experiments (Section 3) to validate the approach and compare it to state-of-the-art methods, on two new deep datasets, one synthetic and one real. In particular we introduce auto-encoders with rich inference (AERIes) which are auto-encoders modified according to this framework.

Indeed both theory and experiments strongly suggest that, when using stacked auto-associators or similar deep architectures, the inference part (from data to latent variables) should use a much richer model than the generative part (from latent variables to data), in fact, as rich as possible. Using richer inference helps to find much better parameters for the *same* given generative model.

1. Deep generative models

Let us go back to the basic formulation of training a deep architecture as a traditional learning problem: optimizing the parameters of the whole architecture seen as a probabilistic generative model of the data.

1.1 Deep models: probability decomposition

The goal of generative learning is to estimate the parameters $\theta = (\theta_1, \ldots, \theta_n)$ of a distribution $P_\theta(\mathbf{x})$ in order to approximate a data distribution $P_\mathcal{D}(\mathbf{x})$ on some observed variable $\mathbf{x}$.

The recent development of deep architectures (Hinton et al., 2006; Bengio et al., 2007) has given importance to a particular case of *latent variable models* in which the distribution of $\mathbf{x}$ can be decomposed as a sum over states of latent variables $\mathbf{h}$,

$$P_\theta(\mathbf{x}) = \sum_{\mathbf{h}} P_{\theta_1,\ldots,\theta_k}(\mathbf{x}|\mathbf{h}) P_{\theta_{k+1},\ldots,\theta_n}(\mathbf{h})$$

with separate parameters for the marginal probability of $\mathbf{h}$ and the conditional probability of $\mathbf{x}$ given $\mathbf{h}$. Setting $I = \{1, 2, \ldots, k\}$ such that θ_I is the set of parameters of $P(\mathbf{x}|\mathbf{h})$ and $J = \{k+1, \ldots, n\}$ such that θ_J is the set of parameters of $P(\mathbf{h})$, this rewrites as

$$P_\theta(\mathbf{x}) \quad = \quad \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h}) P_{\theta_J}(\mathbf{h}) \tag{1}$$

In deep architectures, the same kind of decomposition is applied to $\mathbf{h}$ itself recursively, thus defining a layered model with several hidden layers $\mathbf{h}^{(1)}$, $\mathbf{h}^{(2)}$, …, $\mathbf{h}^{(k_{\max})}$, namely

$$\begin{aligned} P_\theta(\mathbf{x}) &= \sum_{\mathbf{h}^{(1)}} P_{\theta_{I_0}}(\mathbf{x}|\mathbf{h}^{(1)}) P_{\theta_{J_0}}(\mathbf{h}^{(1)}) & (2)\\ P(\mathbf{h}^{(k)}) &= \sum_{\mathbf{h}^{(k+1)}} P_{\theta_{I_k}}(\mathbf{h}^{(k)}|\mathbf{h}^{(k+1)}) P_{\theta_{J_k}}(\mathbf{h}^{(k+1)}), \;\; 1 \leqslant k \leqslant k_{\max} - 1 & (3) \end{aligned}$$

At any one time, we will only be interested in one step of this decomposition. Thus for simplicity, we consider that the distribution of interest is on the observed variable $\mathbf{x}$, with latent variable $\mathbf{h}$. The results extend to the other layers of the decomposition by renaming variables.

In Sections 2.3 and 2.4 we quickly present two frequently used deep architectures, stacked RBMs and auto-encoders, within this framework.

1.2 Data log-likelihood

The goal of the learning procedure, for a probabilistic generative model, is generally to maximize the log-likelihood of the data under the model, namely, to find the value of the parameter $\theta^* = (\theta_I^*, \theta_J^*)$ achieving

$$\begin{aligned} \theta^* &:= \arg\max_\theta \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} [\log P_\theta(\mathbf{x})] & (4)\\ &= \arg\min_\theta D_{\mathrm{KL}}(P_\mathcal{D} \,\|\, P_\theta), & (5) \end{aligned}$$

where $P_{\mathcal{D}}$ is the empirical data distribution, and $D_{\mathrm{KL}}(\cdot \,\|\, \cdot)$ is the Kullback–Leibler divergence. (For simplicity we assume this optimum is unique.)

An obvious way to tackle this problem would be a gradient ascent over the full parameter θ. However, this is impractical for deep architectures (Section 1.3 below).

It would be easier to be able to train deep architectures in a layer-wise fashion, by first training the parameters θ_I of the bottom layer, deriving a new target distribution for the latent variables $\mathbf{h}$, and then training θ_J to reproduce this target distribution on $\mathbf{h}$, recursively over the layers, till one reaches the top layer on which, hopefully, a simple probabilistic generative model can be used.

Indeed this is often done in practice, except that the objective (4) is replaced with a surrogate objective. For instance, for architectures made of stacked RBMs, at each level the likelihood of a single RBM is maximized, ignoring the fact that it is to be used as part of a deep architecture, and moreover often using a further approximation to the likelihood such as contrastive divergence (Hinton., 2002). Under specific conditions (i.e., initializing the upper layer with an upside-down version of the current RBM), it can be shown that adding a layer improves a lower bound on performance (Hinton et al., 2006).

We address in Section 2 the following questions: Is it possible to compute or estimate the optimal value of the parameters θ_I^* of the bottom layer, without training the whole model? Is it possible to compare two values of θ_I without training the whole model? The latter would be particularly convenient for hyper-parameter selection, as it would allow to compare lower-layer models before the upper layers are trained, thus significantly reducing the size of the hyper-parameter search space from exponential to linear in the number of layers.

We propose a procedure aimed at reaching the global optimum θ^* in a layer-wise fashion, based on an optimistic estimate of log-likelihood, the *best latent marginal (BLM) upper bound*. We study its theoretical guarantees in Section 2. In Section 3 we make an experimental comparison between stacked RBMs, auto-encoders modified according to this scheme, and vanilla auto-encoders, on two simple but deep datasets.

1.3 Learning by gradient ascent for deep architectures

Maximizing the likelihood of the data distribution $P_{\mathcal{D}}(\mathbf{x})$ under a model, or equivalently minimizing the KL-divergence $D_{\mathrm{KL}}(P_{\mathcal{D}} \,\|\, P_\theta)$, is usually done with gradient ascent in the parameter space.

The derivative of the log-likelihood for a deep generative model can be written as:

$$\frac{\partial \log P_\theta(\mathbf{x})}{\partial \theta} = \frac{\sum_{\mathbf{h}} \frac{\partial P_{\theta_I}(\mathbf{x}|\mathbf{h})}{\partial \theta} P_{\theta_J}(\mathbf{h}) + \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h}) \frac{\partial P_{\theta_J}(\mathbf{h})}{\partial \theta}}{P_\theta(\mathbf{x})} \tag{6}$$

$$= \sum_{\mathbf{h}} \frac{\partial \log P_{\theta_I}(\mathbf{x}|\mathbf{h})}{\partial \theta} P_\theta(\mathbf{h}|\mathbf{x}) + \sum_{\mathbf{h}} \frac{\partial \log P_{\theta_J}(\mathbf{h})}{\partial \theta} P_\theta(\mathbf{h}|\mathbf{x}) \tag{7}$$

by rewriting $P_\theta(\mathbf{h})/P_\theta(\mathbf{x}) = P_\theta(\mathbf{h}|\mathbf{x})/P_\theta(\mathbf{x}|\mathbf{h})$. The derivative w.r.t. a given component θ_i of θ simplifies because θ_i is either a parameter of $P_{\theta_I}(\mathbf{x}|\mathbf{h})$ when $i \in I$, or a parameter of $P_{\theta_J}(\mathbf{h})$ when $i \in J$:

$$\forall i \in I, \qquad \frac{\partial \log P_\theta(\mathbf{x})}{\partial \theta_i} = \sum_{\mathbf{h}} \frac{\partial \log P_{\theta_I}(\mathbf{x}|\mathbf{h})}{\partial \theta_i} P_{\theta_I,\theta_J}(\mathbf{h}|\mathbf{x}), \tag{8}$$

$$\forall i \in J, \qquad \frac{\partial \log P_\theta(\mathbf{x})}{\partial \theta_i} = \sum_{\mathbf{h}} \frac{\partial \log P_{\theta_J}(\mathbf{h})}{\partial \theta_i} P_{\theta_I,\theta_J}(\mathbf{h}|\mathbf{x}). \tag{9}$$

Unfortunately, this gradient ascent procedure is generally intractable, because it requires sampling from $P_{\theta_I,\theta_J}(\mathbf{h}|\mathbf{x})$ (where both the upper layer and lower layer influence $\mathbf{h}$) to perform inference in the deep model.

2. Layer-wise deep learning

2.1 A theoretical guarantee

We now present a training procedure that works successively on each layer. First we train θ_I together with a conditional model $q(\mathbf{h}|\mathbf{x})$ for the latent variable knowing the data. This step involves only the bottom part of the model and is thus often tractable. This allows to infer a new target distribution for $\mathbf{h}$, on which the upper layers can then be trained.

This procedure singles out a particular setting $\hat{\theta}_I$ for the bottom layer of a deep architecture, based on an optimistic assumption of what the upper layers may be able to do (cf. Proposition 3).

Under this procedure, Theorem 1 states that it is possible to obtain a validation that the parameter $\hat{\theta}_I$ for the bottom layer was optimal, provided the rest of the training goes well. Namely, *if* the target distribution for $\mathbf{h}$ can be realized or well approximated by some value of the parameters θ_J of the top layers, and if θ_I was obtained using a rich enough conditional model $q(\mathbf{h}|\mathbf{x})$, then (θ_I, θ_J) is guaranteed to be globally optimal.

Theorem 1 *Suppose the parameters θ_I of the bottom layer are trained by*

$$(\hat{\theta}_I, \hat{q}) := \underset{\theta_I, q}{\arg\max}\, \mathbb{E}_{\mathbf{x}\sim P_{\mathcal{D}}} \left[\log \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h})\, q_{\mathcal{D}}(\mathbf{h}) \right] \tag{10}$$

where the arg max runs over all *conditional probability distributions $q(\mathbf{h}|\mathbf{x})$ and where*

$$q_{\mathcal{D}}(\mathbf{h}) := \sum_{\tilde{\mathbf{x}}} q(\mathbf{h}|\tilde{\mathbf{x}}) P_{\mathcal{D}}(\tilde{\mathbf{x}}) \tag{11}$$

with $P_{\mathcal{D}}$ the observed data distribution.

We call the optimal $\hat{\theta}_I$ the best optimistic lower layer *(BOLL). Let $\hat{q}_{\mathcal{D}}(\mathbf{h})$ be the distribution on $\mathbf{h}$ associated with the optimal $\hat{q}$. Then:*

- *If the top layers can be trained to reproduce $\hat{q}_{\mathcal{D}}(\mathbf{h})$ perfectly, i.e., if there exists a parameter $\hat{\theta}_J$ for the top layers such that the distribution $P_{\hat{\theta}_J}(\mathbf{h})$ is equal to $\hat{q}_{\mathcal{D}}(\mathbf{h})$, then the parameters obtained are globally optimal:*

$$(\hat{\theta}_I, \hat{\theta}_J) = (\theta_I^*, \theta_J^*)$$

- *Whatever parameter value θ_J is used on the top layers in conjunction with the BOLL $\hat{\theta}_I$, the difference in performance (4) between $(\hat{\theta}_I, \theta_J)$ and the global optimum (θ_I^*, θ_J^*) is at most the Kullback–Leibler divergence $D_{\mathrm{KL}}(\hat{q}_{\mathcal{D}}(\mathbf{h}) \,\|\, P_{\theta_J}(\mathbf{h}))$ between $\hat{q}_{\mathcal{D}}(\mathbf{h})$ and $P_{\theta_J}(\mathbf{h})$.*

This theorem strongly suggests using $\hat{q}_{\mathcal{D}}(\mathbf{h})$ as the target distribution for the top layers, i.e., looking for the value $\hat{\theta}_J$ best approximating $\hat{q}_{\mathcal{D}}(\mathbf{h})$:

$$\hat{\theta}_J := \arg\min_{\theta_J} D_{\mathrm{KL}}(\hat{q}_{\mathcal{D}}(\mathbf{h}) \,\|\, P_{\theta_J}(\mathbf{h})) = \arg\max_{\theta_J} \mathbb{E}_{\mathbf{h}\sim\hat{q}_{\mathcal{D}}} \log P_{\theta_J}(\mathbf{h}) \tag{12}$$

which thus takes the same form as the original problem. Then the same scheme may be used recursively to train the top layers. A final fine-tuning phase may be helpful, see Section 2.6.

Note that when the top layers fail to approximate $\hat{q}_{\mathcal{D}}$ perfectly, the loss of performance depends only on the observed difference between $\hat{q}_{\mathcal{D}}$ and $P_{\hat{\theta}_J}$, and not on the unknown global optimum (θ_I^*, θ_J^*). Beware that, unfortunately, this bound relies on *perfect* layer-wise training of the bottom layer, i.e., on $\hat{q}$ being the optimum of the criterion (10) optimized over all possible conditional distributions q; otherwise it is a priori not valid.

In practice the supremum on q will always be taken over a restricted set of conditional distributions $q(\mathbf{h}|\mathbf{x})$, rather than the set of all possible distributions on $\mathbf{h}$ for each $\mathbf{x}$. Thus, this theorem is an idealized version of practice (though Remark 4 below mitigates this). This still suggests a clear strategy to separate the deep optimization problem into two subproblems to be solved sequentially:

1. Train the parameters θ_I of the bottom layer after (10), using a model $q(\mathbf{h}|\mathbf{x})$ as wide as possible, to approximate the BOLL $\hat{\theta}_I$.

2. Infer the corresponding distribution of $\mathbf{h}$ by (11) and train the upper part of the model as best as possible to approximate this distribution.

Then, provided learning is successful in both instances, the result is close to optimal.

Auto-encoders can be shown to implement an approximation of this procedure, in which only the terms $\mathbf{x} = \tilde{\mathbf{x}}$ are kept in (10)–(11) (Section 2.4).

This scheme is designed with in mind a situation in which the upper layers get progessively simpler. Indeed, if the layer for $\mathbf{h}$ is as wide as the layer for $\mathbf{x}$ and if

$P(\mathbf{x}|\mathbf{h})$ can learn the identity, then the procedure in Theorem 1 just transfers the problem unchanged one layer up.

This theorem strongly suggests decoupling the inference and generative models $q(\mathbf{h}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h})$, and using a rich conditional model $q(\mathbf{h}|\mathbf{x})$, contrary, e.g., to common practice in auto-encoders[1]. Indeed the experiments of Section 3 confirm that using a more expressive $q(\mathbf{h}|\mathbf{x})$ yields improved values of θ.

Importantly, $q(\mathbf{h}|\mathbf{x})$ is only used as an auxiliary prop for solving the optimization problem (4) over θ and is not part of the final generative model, so that using a richer $q(\mathbf{h}|\mathbf{x})$ to reach a better value of θ is not simply changing to a larger model. Thus, using a richer inference model $q(\mathbf{h}|\mathbf{x})$ should not pose too much risk of overfitting because the regularization properties of the model come mainly from the choice of the generative model family (θ).

The criterion proposed in (10) is of particular relevance to representation learning where the goal is not to learn a generative model, but to learn a useful representation of the data. In this setting, training an upper layer model $P(\mathbf{h})$ becomes irrelevant because we are not interested in the generative model itself. What matters in representation learning is that the lower layer (i.e., $P(\mathbf{x}|\mathbf{h})$ and $q(\mathbf{h}|\mathbf{x})$) is optimal for *some* model of $P(\mathbf{h})$, left unspecified.

We now proceed, by steps, to the proof of Theorem 1. This will be the occasion to introduce some concepts used later in the experimental setting.

2.2 The Best Latent Marginal Upper Bound

One way to evaluate a parameter θ_I for the bottom layer without training the whole architecture is to be optimistic: assume that the top layers will be able to produce the probability distribution for $\mathbf{h}$ that gives the best results if used together with $P_{\theta_I}(\mathbf{x}|\mathbf{h})$. This leads to the following.

Definition 2 *Let θ_I be a value of the bottom layer parameters. The* best latent marginal *(BLM) for θ_I is the probability distribution Q on $\mathbf{h}$ maximizing the log-likelihood:*

$$\hat{Q}_{\theta_I,\mathcal{D}} := \arg\max_Q \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}}\left[\log \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h})Q(\mathbf{h})\right] \tag{13}$$

where the arg max runs over the set of all probability distributions over $\mathbf{h}$. The BLM upper bound *is the corresponding log-likelihood value:*

$$\mathcal{U}_\mathcal{D}(\theta_I) := \max_Q \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}}\left[\log \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h})Q(\mathbf{h})\right] \tag{14}$$

[1] Attempts to prevent auto-encoders from learning the identity (which is completely justifiable) often result in an even more constrained inference model, e.g., tied weights, or sparsity constraints on the hidden representation.

The BLM upper bound $\mathcal{U}_{\mathcal{D}}(\theta_I)$ is the least upper bound on the log-likelihood of the deep generative model on the dataset $\mathcal{D}$ if θ_I is used for the bottom layer. $\mathcal{U}_{\mathcal{D}}(\theta_I)$ is only an upper bound of the actual performance of θ_I, because subsequent training of $P_{\theta_J}(\mathbf{h})$ may be suboptimal: the best latent marginal $\hat{Q}_{\theta_I,\mathcal{D}}(\mathbf{h})$ may not be representable as $P_{\theta_J}(\mathbf{h})$ for θ_J in the model, or the training of $P_{\theta_J}(\mathbf{h})$ itself may not converge to the best solution.

Note that the arg max in (13) is concave in Q, so that in typical situations the BLM is unique—except in degenerate cases such as when two values of $\mathbf{h}$ define the same $P_{\theta_I}(\mathbf{x}|\mathbf{h})$).

Proposition 3 *The criterion* (10) *used in Theorem 1 for training the bottom layer coincides with the BLM upper bound:*

$$\mathcal{U}_{\mathcal{D}}(\theta_I) = \max_q \mathbb{E}_{\mathbf{x}\sim P_{\mathcal{D}}}\left[\log \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h}) q_{\mathcal{D}}(\mathbf{h})\right] \tag{15}$$

where the maximum runs over all conditional probability distributions $q(\mathbf{h}|\mathbf{x})$*. In particular the BOLL* $\hat{\theta}_I$ *selected in Theorem 1 is*

$$\hat{\theta}_I = \arg\max_{\theta_I} \mathcal{U}_{\mathcal{D}}(\theta_I) \tag{16}$$

and the target distribution $\hat{q}_{\mathcal{D}}(\mathbf{h})$ *in Theorem 1 is the best latent marginal* $\hat{Q}_{\hat{\theta}_I,\mathcal{D}}$*.*

Thus the BOLL $\hat{\theta}_I$ is the best bottom layer setting if one uses an optimistic criterion for assessing the bottom layer, hence the name “best optimistic lower layer”.

Proof Any distribution Q over $\mathbf{h}$ can be written as $q_{\mathcal{D}}$ for some conditional distribution $q(\mathbf{h}|\mathbf{x})$, for instance by defining $q(\mathbf{h}|\mathbf{x}) = Q(\mathbf{h})$ for every $\mathbf{x}$ in the dataset. In particular this is the case for the best latent marginal $\hat{Q}_{\theta_I,\mathcal{D}}$.

Consequently the maxima in (15) and in (14) are taken on the same set and coincide. ■

The argument that any distribution is of the form $q_{\mathcal{D}}$ may look disappointing: why choose this particular form? In Section 2.7 we show how writing distributions over $\mathbf{h}$ as $q_{\mathcal{D}}$ for some conditional distribution $q(\mathbf{h}|\mathbf{x})$ may help to maximize data log-likelihood, by quantifiably incorporating information from the data (Proposition 7). Moreover, the bound on loss of performance (second part of Theorem 1) when the upper layers do not match the BLM crucially relies on the properties of $\hat{q}_{\mathcal{D}}$. A more practical argument for using $q_{\mathcal{D}}$ is that optimizing both θ_I and the full distribution of the hidden variable $\mathbf{h}$ at the same time is just as difficult as optimizing the whole network, whereas the deep architectures currently in use already train a model of $\mathbf{x}$ knowing $\mathbf{h}$ and of $\mathbf{h}$ knowing $\mathbf{x}$ at the same time.

Remark 4 *For Theorem 1 to hold, it is not necessary to optimize over all possible conditional probability distributions $q(\mathbf{h}|\mathbf{x})$ (which is a set of very large dimension). As can be seen from the proof above it is enough to optimize over a family $q(\mathbf{h}|\mathbf{x}) \in \mathcal{Q}$ such that every (non-conditional) distribution on $\mathbf{h}$ can be represented (or well approximated) as $q_{\mathcal{D}}(\mathbf{h})$ for some $q \in \mathcal{Q}$.*

Let us now go on with the proof of Theorem 1.

Proposition 5 *Set the bottom layer parameters to the BOLL*

$$\hat{\theta}_I = \arg\max_{\theta_I} \mathcal{U}_{\mathcal{D}}(\theta_I) \tag{17}$$

and let $\hat{Q}$ be the corresponding best latent marginal.

Assume that subsequent training of the top layers using $\hat{Q}$ as the target distribution for $\mathbf{h}$, is successful, i.e., there exists a θ_J such that $\hat{Q}(\mathbf{h}) = P_{\theta_J}(\mathbf{h})$.

Then $\hat{\theta}_I = \theta_I^*$.

Proof Define the *in-model* BLM upper bound as

$$\mathcal{U}_{\mathcal{D}}^{\text{model}}(\theta_I) := \max_{\theta_J} \mathbb{E}_{\mathbf{x}\sim P_{\mathcal{D}}} \left[\log \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h}) P_{\theta_J}(\mathbf{h})\right] \tag{18}$$

By definition, the global optimum θ_I^* for the parameters of the whole architecture is given by $\theta_I^* = \arg\max_{\theta_I} \mathcal{U}_{\mathcal{D}}^{\text{model}}(\theta_I)$.

Obviously, for any value θ_I we have $\mathcal{U}_{\mathcal{D}}^{\text{model}}(\theta_I) \leqslant \mathcal{U}_{\mathcal{D}}(\theta_I)$ since the argmax is taken over a more restricted set. Then, in turn, $\mathcal{U}_{\mathcal{D}}(\theta_I) \leqslant \mathcal{U}_{\mathcal{D}}(\hat{\theta}_I)$ by definition of $\hat{\theta}_I$.

By our assumption, the BLM $\hat{Q}$ for $\hat{\theta}_I$ happens to lie in the model: $\hat{Q}(\mathbf{h}) = P_{\theta_J}(\mathbf{h})$. This implies that $\mathcal{U}_{\mathcal{D}}(\hat{\theta}_I) = \mathcal{U}_{\mathcal{D}}^{\text{model}}(\hat{\theta}_I)$.

Combining, we get that $\mathcal{U}_{\mathcal{D}}^{\text{model}}(\theta_I) \leqslant \mathcal{U}_{\mathcal{D}}^{\text{model}}(\hat{\theta}_I)$ for any θ_I. Thus $\hat{\theta}_I$ maximizes $\mathcal{U}_{\mathcal{D}}^{\text{model}}(\theta_I)$, and is thus equal to θ_I^*. ■

The first part of Theorem 1 then results from the combination of Propositions 5 and 3.

We now give a bound on the loss of performance in case further training of the upper layers fails to reproduce the BLM. This will complete the proof of Theorem 1. We will make use of a special optimality property of distributions of the form $q_{\mathcal{D}}(\mathbf{h})$, namely, Proposition 7, whose proof is postponed to Section 2.7.

Proposition 6 *Keep the notation of Theorem 1. In the case when $P_{\theta_J}(\mathbf{h})$ fails to reproduce $\hat{q}_{\mathcal{D}}(\mathbf{h})$ exactly, the loss of performance of $(\hat{\theta}_I, \theta_J)$ with respect to the global optimum (θ_I^*, θ_J^*) is at most*

$$D_{\text{KL}}(P_{\mathcal{D}}(\mathbf{x}) \,\|\, P_{\hat{\theta}_I,\theta_J}(\mathbf{x})) - D_{\text{KL}}(P_{\mathcal{D}}(\mathbf{x}) \,\|\, \hat{q}_{\mathcal{D},\hat{\theta}_I}(\mathbf{x})) \tag{19}$$

where $\hat{q}_{\mathcal{D},\hat{\theta}_I}(\mathbf{x}) := \sum_{\mathbf{h}} P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h})\hat{q}_{\mathcal{D}}(\mathbf{h})$ *is the distribution on* $\mathbf{x}$ *obtained by using the BLM.*

This quantity is in turn at most

$$D_{\mathrm{KL}}(\hat{q}_{\mathcal{D}}(\mathbf{h}) \,\|\, P_{\theta_J}(\mathbf{h})) \tag{20}$$

which is thus also a bound on the loss of performance of $(\hat{\theta}_I, \theta_J)$ *with respect to* (θ_I^*, θ_J^*).

Note that these estimates do not depend on the unkown global optimum θ^*.

Importantly, this bound is *not* valid if $\hat{q}$ has not been perfectly optimized over all possible conditional distributions $q(\mathbf{h}|\mathbf{x})$. Thus it should not be used blindly to get a performance bound, since heuristics will always be used to find $\hat{q}$. Therefore, it may have only limited practical relevance. In practice the real loss may both be larger than this bound because q has been optimized over a smaller set, and smaller because we are comparing to the BLM upper bound which is an optimistic assessment.

Proof From (4) and (5), the difference in log-likelihood performance between any two distributions $p_1(\mathbf{x})$ and $p_2(\mathbf{x})$ is equal to $D_{\mathrm{KL}}(P_{\mathcal{D}} \,\|\, p_1) - D_{\mathrm{KL}}(P_{\mathcal{D}} \,\|\, p_2)$.

For simplicity, denote

$$\begin{aligned} p_1(\mathbf{x}) &= P_{\hat{\theta}_I,\theta_J}(\mathbf{x}) = \sum_{\mathbf{h}} P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h}) P_{\theta_J}(\mathbf{h}) \\ p_2(\mathbf{x}) &= P_{\theta_I^*,\theta_J^*}(\mathbf{x}) = \sum_{\mathbf{h}} P_{\theta_I^*}(\mathbf{x}|\mathbf{h}) P_{\theta_J^*}(\mathbf{h}) \\ p_3(\mathbf{x}) &= \sum_{\mathbf{h}} P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h}) \hat{q}_{\mathcal{D}}(\mathbf{h}) \end{aligned}$$

We want to compare p_1 and p_2.

Define the in-model upper bound $\mathcal{U}_{\mathcal{D}}^{\mathrm{model}}(\theta_I)$ as in (18) above. Then we have $\theta_I^* = \arg\max_{\theta_I} \mathcal{U}_{\mathcal{D}}^{\mathrm{model}}(\theta_I)$ and $\hat{\theta}_I = \arg\max_{\theta_I} \mathcal{U}_{\mathcal{D}}(\theta_I)$. Since $\mathcal{U}_{\mathcal{D}}^{\mathrm{model}} \leqslant \mathcal{U}_{\mathcal{D}}$, we have $\mathcal{U}_{\mathcal{D}}^{\mathrm{model}}(\theta_I^*) \leqslant \mathcal{U}_D(\hat{\theta}_I)$. The BLM upper bound $\mathcal{U}_D(\hat{\theta}_I)$ is attained when we use $\hat{q}_{\mathcal{D}}$ as the distribution for $\mathbf{h}$, so $\mathcal{U}_{\mathcal{D}}^{\mathrm{model}}(\theta_I^*) \leqslant \mathcal{U}_D(\hat{\theta}_I)$ means that the performance of p_3 is better than the performance of p_2:

$$D_{\mathrm{KL}}(P_{\mathcal{D}} \,\|\, p_3) \leqslant D_{\mathrm{KL}}(P_{\mathcal{D}} \,\|\, p_2)$$

(inequalities hold in the reverse order for data log-likelihood).

Now by definition of the optimum θ^*, the distribution p_2 is better than p_1: $D_{\mathrm{KL}}(P_{\mathcal{D}} \,\|\, p_2) \leqslant D_{\mathrm{KL}}(P_{\mathcal{D}} \,\|\, p_1)$. Consequently, the difference in performance between p_2 and p_1 (whether expressed in data log-likelihood or in Kullback–Leibler divergence) is smaller than the difference in performance between p_3 and p_1, which is the difference of Kullback–Leibler divergences appearing in the proposition.

Let us now evaluate more precisely the loss of p_1 with respect to p_3. By abuse of notation we will indifferently denote $p_1(\mathbf{h})$ and $p_1(\mathbf{x})$, it being understood that one is obtained from the other through $P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h})$, and likewise for p_3 (with the same $\hat{\theta}_I$).

For any distributions p_1 and p_3 the loss of performance of p_1 w.r.t. p_3 satisfies

$$\mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \log p_3(\mathbf{x}) - \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \log p_1(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \left[\log \frac{\sum_\mathbf{h} P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h}) p_3(\mathbf{h})}{\sum_\mathbf{h} P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h}) p_1(\mathbf{h})} \right]$$

and by the log sum inequality $\log(\sum a_i / \sum b_i) \leqslant \frac{1}{\sum a_i} \sum a_i \log(a_i/b_i)$ (Cover and Thomas, 2006, Theorem 2.7.1) we get

$$\begin{aligned}
&\mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \log p_3(\mathbf{x}) - \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \log p_1(\mathbf{x}) \\
&\leqslant \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \left[\frac{1}{\sum_\mathbf{h} P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h}) p_3(\mathbf{h})} \sum_\mathbf{h} P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h}) p_3(\mathbf{h}) \log \frac{P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h}) p_3(\mathbf{h})}{P_{\hat{\theta}_I}(\mathbf{x}|\mathbf{h}) p_1(\mathbf{h})} \right] \\
&= \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \left[\frac{1}{p_3(\mathbf{x})} \sum_\mathbf{h} p_3(\mathbf{x},\mathbf{h}) \log \frac{p_3(\mathbf{h})}{p_1(\mathbf{h})} \right] \\
&= \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \left[\sum_\mathbf{h} p_3(\mathbf{h}|\mathbf{x}) \log \frac{p_3(\mathbf{h})}{p_1(\mathbf{h})} \right] \\
&= \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \mathbb{E}_{\mathbf{h}\sim p_3(\mathbf{h}|\mathbf{x})} \left[\log \frac{p_3(\mathbf{h})}{p_1(\mathbf{h})} \right]
\end{aligned}$$

Given a probability p_3 on $(\mathbf{x},\mathbf{h})$, the law on $\mathbf{h}$ obtained by taking an $\mathbf{x}$ according to $P_\mathcal{D}$, then taking an $\mathbf{h}$ according to $p_3(\mathbf{h}|\mathbf{x})$, is generally not equal to $p_3(\mathbf{h})$. However, here p_3 is equal to the BLM $\hat{q}_D$, and by Proposition 7 below the BLM has exactly this property (which characterizes the log-likelihood extrema). Thus thanks to Proposition 7 we have

$$\mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}} \mathbb{E}_{\mathbf{h}\sim \hat{q}_\mathcal{D}(\mathbf{h}|\mathbf{x})} \left[\log \frac{\hat{q}_\mathcal{D}(\mathbf{h})}{p_1(\mathbf{h})} \right] = \mathbb{E}_{\mathbf{h}\sim \hat{q}_\mathcal{D}} \left[\log \frac{\hat{q}_\mathcal{D}(\mathbf{h})}{p_1(\mathbf{h})} \right] = D_{\mathrm{KL}}(\hat{q}_\mathcal{D}(\mathbf{h}) \,\|\, p_1(\mathbf{h}))$$

which concludes the argument. ■

2.3 Relation with Stacked RBMs

Stacked RBMs (SRBMs) (Hinton et al., 2006; Bengio et al., 2007; Larochelle et al., 2009) are deep generative models trained by stacking restricted Boltzmann machines (RBMs) (Smolensky, 1986).

A RBM uses a single set of parameters to represent a distribution on pairs $(\mathbf{x},\mathbf{h})$. Similarly to our approach, stacked RBMs are trained in a greedy layer-wise fashion: one starts by training the distribution of the bottom RBM to approximate the distribution of $\mathbf{x}$. To do so, distributions $P_{\theta_I}(\mathbf{x}|\mathbf{h})$ and $Q_{\theta_I}(\mathbf{h}|\mathbf{x})$ are learned jointly using a *single* set of parameters θ_I. Then a target distribution for $\mathbf{h}$ is defined as

$\sum_{\mathbf{x}} Q_{\theta_I}(\mathbf{h}|\mathbf{x})P_{\mathcal{D}}(\mathbf{x})$ (similarly to (11)) and the top layers are trained recursively on this distribution.

In the final generative model, the full top RBM is used on the top layer to provide a distribution for $\mathbf{h}$, then the bottom RBMs are used only for the generation of $\mathbf{x}$ knowing $\mathbf{h}$. (Therefore the $\mathbf{h}$-biases of the bottom RBMs are never used in the final generative model.)

Thus, in contrast with our approach, $P_{\theta_I}(\mathbf{x}|\mathbf{h})$ and $Q_{\theta_I}(\mathbf{h}|\mathbf{x})$ are not trained to maximize the least upper bound of the likelihood of the full deep generative model but are trained to maximize the likelihood of a single RBM.

This procedure has been shown to be equivalent to maximizing the likelihood of a deep generative model with infinitely many layers where the weights are all tied (Hinton et al., 2006). The latter can be interpreted as an assumption on the future value of $P(\mathbf{h})$, which is unknown when learning the first layer. As such, SRBMs make a different assumption about the future $P(\mathbf{h})$ than the one made in (10).

With respect to this, the comparison of gradient ascents is instructive: the gradient ascent for training the bottom RBM takes a form reminiscent of gradient ascent of the global generative model (7) but in which the dependency of $P(\mathbf{h})$ on the upper layers θ_J is ignored, and instead the distribution $P(\mathbf{h})$ is tied to θ_I because the RBM uses a single parameter set for both.

When adding a new layer on top of a trained RBM, if the initialization is set to an upside down version of the current RBM (which can be seen as "unrolling" one step of Gibbs sampling), the new deep model still matches the special infinite deep generative model with tied weights mentioned above. Starting training of the upper layer from this initialization guarantees that the new layer can only increase the likelihood (Hinton et al., 2006). However, this result is only known to hold for two layers; with more layers, it is only known that adding layers increases a *bound* on the likelihood (Hinton et al., 2006).

In our approach, the perspective is different. During the training of lower layers, we consider the best possible model for the hidden variable. Because of errors which are bound to occur in approximation and optimization during the training of the model for $P(\mathbf{h})$, the likelihood associated with an optimal upper model (the BLM upper bound) is expected to *decrease* each time we actually take another lower layer into account: At each new layer, errors in approximation or optimization occur so that the final likelihood of the training set will be smaller than the upper bound. (On the other way these limitations might actually improve performance on a test set, see the discussion about regularization in Section 3.)

In (Le Roux and Bengio, 2008) a training criterion is suggested for SRBMs which is reminiscent of a BLM with tied weights for the inference and generative parts (and therefore without the BLM optimality guarantee), see also Section 2.5.

2.4 Relation with Auto-Encoders

Since the introduction of deep neural networks, auto-encoders (Vincent et al., 2008) have been considered a credible alternative to stacked RBMs and have been shown to have almost identical performance on several tasks (Larochelle et al., 2007).

Auto-encoders are trained by stacking auto-associators (Bourlard and Kamp, 1988) trained with backpropagation. Namely: we start with a three-layer network $\mathbf{x} \mapsto \mathbf{h}^{(1)} \mapsto \mathbf{x}$ trained by backpropagation to reproduce the data; this provides two conditional distributions $P(\mathbf{h}^{(1)}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h}^{(1)})$. Then in turn, another auto-associator is trained as a three-layer network $\mathbf{h}^{(1)} \mapsto \mathbf{h}^{(2)} \mapsto \mathbf{h}^{(1)}$, to reproduce the distribution $P(\mathbf{h}^{(1)}|\mathbf{x})$ on $\mathbf{h}^{(1)}$, etc.

So as in the learning of SRBMs, auto-encoder training is performed in a greedy layer-wise manner, but with a different criterion: the reconstruction error.

Note that after the auto-encoder has been trained, the deep generative model is incomplete because it lacks a generative model for the distribution $P(\mathbf{h}^{k_{\max}})$ of the deepest hidden variable, which the auto-encoder does not provide[2]. One possibility is to learn the top layer with an RBM, which then completes the generative model.

Concerning the theoretical soundness of stacking auto-associators for training deep generative models, it is known that the training of auto-associators is an approximation of the training of RBMs in which only the largest term of an expansion of the log-likelihood is kept (Bengio and Delalleau, 2009). In this sense, SRBM and stacked auto-associator training approximate each other (see also Section 2.5).

Our approach gives a new understanding of auto-encoders as the lower part of a deep generative model, because they are trained to maximize a *lower bound* of (10), as follows.

To fix ideas, let us consider for (10) a particular class of conditional distributions $q(\mathbf{h}|\mathbf{x})$ commonly used in auto-associators. Namely, let us parametrize q as q_ξ with

$$q_\xi(\mathbf{h}|\mathbf{x}) = \prod_j q_\xi(h_j|\mathbf{x}) \tag{21}$$

$$q_\xi(h_j|\mathbf{x}) = \operatorname{sigm}(\textstyle\sum_i x_i w_{ij} + b_j) \tag{22}$$

where the parameter vector is $\xi = \{\mathbf{W}, \mathbf{b}\}$ and $\operatorname{sigm}(\cdot)$ is the sigmoid function.

Given a conditional distribution $q(\mathbf{h}|\mathbf{x})$ as in Theorem 1, let us expand the distribution on $\mathbf{x}$ obtained from $P_{\theta_I}(\mathbf{x}|\mathbf{h})$ and $q_\mathcal{D}(\mathbf{h})$:

$$P(\mathbf{x}) = \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h}) q_\mathcal{D}(\mathbf{h}) \tag{23}$$

$$= \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h}) \sum_{\tilde{\mathbf{x}}} q(\mathbf{h}|\tilde{\mathbf{x}}) P_\mathcal{D}(\tilde{\mathbf{x}}) \tag{24}$$

[2] Auto-associators can in fact be used as valid generative models from which sampling is possible (Rifai et al., 2012) in the setting of manifold learning but this is beyond the scope of this article.

where as usual $P_{\mathcal{D}}$ is the data distribution. Keeping only the terms $\mathbf{x} = \tilde{\mathbf{x}}$ in this expression we see that

$$P(\mathbf{x}) \geqslant \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h}) q(\mathbf{h}|\mathbf{x}) P_{\mathcal{D}}(\mathbf{x}) \tag{25}$$

Taking the sum of likelihoods over $\mathbf{x}$ in the dataset, this corresponds to the criterion maximized by auto-associators when they are considered from a probabilistic perspective[3]. Since moreover optimizing over q as in (10) is more general than optimizing over the particular class q_ξ, we conclude that *the criterion optimized in auto-associators is a lower bound on the criterion* (10) *proposed in Theorem 1.*

Keeping only $\mathbf{x} = \tilde{\mathbf{x}}$ is justified if we assume that inference is an approximation of the inverse of the generative process[4], that is, $P_{\theta_I}(\mathbf{x}|\mathbf{h}) q(\mathbf{h}|\tilde{\mathbf{x}}) \approx 0$ as soon as $\mathbf{x} \neq \tilde{\mathbf{x}}$. Thus under this assumption, both criteria will be close, so that Theorem 1 provides a justification for auto-encoder training in this case. On the other hand, this assumption can be strong: it implies that no $\mathbf{h}$ can be shared between different $\mathbf{x}$, so that for instance two observations cannot come from the same underlying latent variable through a random choice. Depending on the situation this might be unrealistic. Still, using this as a training criterion might perform well even if the assumption is not fully satisfied.

Note that we chose the form of $q_\xi(\mathbf{h}|\mathbf{x})$ to match that of the usual auto-associator, but of course we could have made a different choice such as using a multilayer network for $q_\xi(\mathbf{h}|\mathbf{x})$ or $P_{\theta_I}(\mathbf{x}|\mathbf{h})$. These possibilities will be explored later in this article.

2.5 From stacked RBMs to auto-encoders: layer-wise consistency

We now show how imposing a "layer-wise consistency" constraint on stacked RBM training leads to the training criterion used in auto-encoders with tied weights. Some of the material here already appears in (Le Roux and Bengio, 2008).

Let us call *layer-wise consistent* a layer-wise training procedure in which each layer determines a value θ_I for its parameters and a target distribution $P(\mathbf{h})$ for the upper layers which are mutually optimal in the following sense: if $P(\mathbf{h})$ is used a the distribution of the hidden variable, then θ_I is the bottom parameter value maximizing data log-likelihood.

[3] In all fairness, the training of auto-associators by backpropagation, in probabilistic terms, consists in the maximization of $P(\mathbf{y}|\mathbf{x})P_{\mathcal{D}}(\mathbf{x}) = o(\mathbf{x})P_{\mathcal{D}}(\mathbf{x})$ with $\mathbf{y} = \mathbf{x}$ (Buntine and Weigend, 1991), where o is the output function of the neural network. In this perspective, the hidden variable $\mathbf{h}$ is not considered as a random variable but as an intermediate value in the form of $P(\mathbf{y}|\mathbf{x})$. Here, we introduce $\mathbf{h}$ as an intermediate random variable as in (Neal, 1990). The criterion we wish to maximize is then $P(\mathbf{y}|\mathbf{x})P_{\mathcal{D}}(\mathbf{x}) = \sum_{\mathbf{h}} f(\mathbf{y}|\mathbf{h})g(\mathbf{h}|\mathbf{x})P_{\mathcal{D}}(\mathbf{x})$, with $\mathbf{y} = \mathbf{x}$. Training with backpropagation can be done by sampling $\mathbf{h}$ from $g(\mathbf{h}|\mathbf{x})$ instead of using the raw activation value of $g(\mathbf{h}|\mathbf{x})$, but in practice we do not sample $\mathbf{h}$ as it does not significantly affect performance.

[4] which is a reasonable assumption if we are to perform inference in any meaningful sense of the word.

The BLM training procedure is, by construction, layer-wise consistent.

Let us try to train stacked RBMs in a layer-wise consistent way. Given a parameter θ_I, SRBMs use the hidden variable distribution

$$Q_{\mathcal{D},\theta_I}(\mathbf{h}) = \mathbb{E}_{\mathbf{x}\sim P_{\mathcal{D}}}\, P_{\theta_I}(\mathbf{h}|\mathbf{x}) \tag{26}$$

as the target for the next layer, where $P_{\theta_I}(\mathbf{h}|\mathbf{x})$ is the RBM distribution of $\mathbf{h}$ knowing $\mathbf{x}$. The value θ_I and this distribution over $\mathbf{h}$ are mutually optimal for each other if the distribution on $\mathbf{x}$ stemming from this distribution on $\mathbf{h}$, given by

$$\begin{align} P^{(1)}_{\theta_I}(\mathbf{x}) &= \mathbb{E}_{\mathbf{h}\sim Q_{\mathcal{D},\theta_I}(\mathbf{h})}\, P_{\theta_I}(\mathbf{x}|\mathbf{h}) \tag{27}\\ &= \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h}) \sum_{\tilde{\mathbf{x}}} P_{\theta_I}(\mathbf{h}|\tilde{\mathbf{x}}) P_{\mathcal{D}}(\tilde{\mathbf{x}}) \tag{28}\end{align}$$

maximizes log-likelihood, i.e.,

$$\theta_I = \arg\min D_{\mathrm{KL}}(P_{\mathcal{D}}(\mathbf{x}) \,\|\, P^{(1)}_{\theta_I}(\mathbf{x})) \tag{29}$$

The distribution $P^{(1)}_{\theta_I}(\mathbf{x})$ is the one obtained from the data after one "forward-backward" step of Gibbs sampling $\mathbf{x} \to \mathbf{h} \to \mathbf{x}$ (cf. Le Roux and Bengio, 2008).

But $P^{(1)}_{\theta_I}(\mathbf{x})$ is also equal to the distribution (24) for an auto-encoder with tied weights. So the layer-wise consistency criterion for RBMs coincides with tied-weights auto-encoder training, up to the approximation that in practice auto-encoders retain only the terms $\mathbf{x} = \tilde{\mathbf{x}}$ in the above (Section 2.4).

On the other hand, stacked RBM training trains the parameter θ_I to approximate the data distribution by the RBM distribution:

$$\theta_I^{\mathrm{RBM}} = \arg\min_{\theta_I} D_{\mathrm{KL}}(P_{\mathcal{D}}(\mathbf{x}) \,\|\, P^{\mathrm{RBM}}_{\theta_I}(\mathbf{x})) \tag{30}$$

where $P^{\mathrm{RBM}}_{\theta_I}$ is the probability distribution of the RBM with parameter θ_I, i.e. the probability distribution after an infinite number of Gibbs samplings from the data.

Thus, stacked RBM training and tied-weight auto-encoder training can be seen as two approximations to the layer-wise consistent optimization problem (29), one using the full RBM distribution $P^{\mathrm{RBM}}_{\theta_I}$ instead of $P^{(1)}_{\theta_I}$ and the other using $\mathbf{x} = \tilde{\mathbf{x}}$ in $P^{(1)}_{\theta_I}$.

It is not clear to us to which extent the criteria (29) using $P^{(1)}_{\theta_I}$ and (30) using $P^{\mathrm{RBM}}_{\theta_I}$ actually yield different values for the optimal θ_I: although these two optimization criteria are different (unless RBM Gibbs sampling converges in one step), it might be that the optimal θ_I is the same (in which case SRBM training would be layer-wise consistent), though this seems unlikely.

The θ_I obtained from the layer-wise consistent criterion (29) using $P^{(1)}_{\theta_I}(\mathbf{x})$ will always perform at least as well as standard SRBM training if the upper layers match the target distribution on $\mathbf{h}$ perfectly—this follows from its very definition.

Nonetheless, it is not clear whether layer-wise consistency is always a desirable property. In SRBM training, replacing the RBM distribution over $\mathbf{h}$ with the one obtained from the data seemingly breaks layer-wise consistency, but at the same time it always *improves* data log-likelihood (as a consequence of Proposition 7 below).

For non-layer-wise consistent training procedures, fine-tuning of θ_I after more layers have been trained would improve performance. Layer-wise consistent procedures may require this as well in case the upper layers do not match the target distribution on $\mathbf{h}$ (while non-layer-wise consistent procedures would require this even with perfect upper layer training).

2.6 Relation to fine-tuning

When the approach presented in Section 2 is used recursively to train deep generative models with several layers using the criterion (10), irrecoverable losses may be incurred at each step: first, because the optimization problem (10) may be imperfectly solved, and, second, because each layer was trained using a BLM assumption about what upper layers are able to do, and subsequent upper layer training may not match the BLM. Consequently the parameters used for each layer may not be optimal with respect to each other. This suggests using a fine-tuning procedure.

In the case of auto-encoders, fine-tuning can be done by backpropagation on all (inference and generative) layers at once (Figure 1). This has been shown to improve performance[5] in several contexts (Larochelle et al., 2009; Hinton and Salakhutdinov., 2006), which confirms the expected gain in performance from recovering earlier approximation losses. In principle, there is no limit to the number of layers of an auto-encoder that could be trained at once by backpropagation, but in practice training many layers at once results in a difficult optimization problem with many local minima. Layer-wise training can be seen as a way of dealing with the issue of local minima, providing a solution close to a good optimum. This optimum is then reached by global fine-tuning.

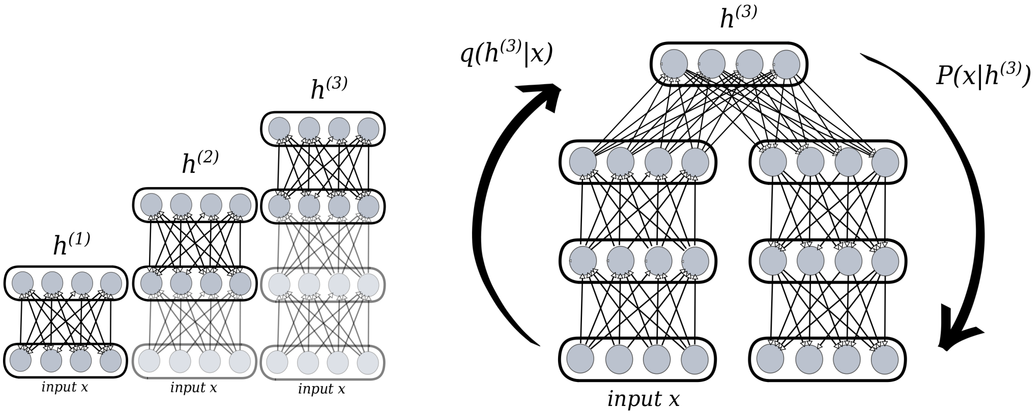
Fine-tuning can be described in the BLM framework as follows: fine-tuning is the maximization of the BLM upper bound (10) where all the layers are considered as one single complex layer (Figure 1). In the case of auto-encoders, the approximation $\mathbf{x} = \tilde{\mathbf{x}}$ in (10)–(11) is used to help optimization, as explained above.

Note that there is no reason to limit fine-tuning to the end of the layer-wise procedure: fine-tuning may be used at intermediate stages where any number of layers have been trained.

This fine-tuning procedure was not applied in the experiments below because our experiments only have one layer for the bottom part of the model.

As mentioned before, a generative model for the topmost hidden layer (e.g., an RBM) still needs to be trained to get a complete generative model after fine-tuning.

[5] The exact likelihood not being tractable for larger models, it is necessary to rely on a proxy such as classification performance to evaluate the performance of the deep network.



Layer-wise deep learning for several layers

Fine-tuning of all generative and inference layers

Figure 1: Deep training with fine-tuning.

2.7 Data Incorporation: Properties of $q_{\mathcal{D}}$

It is not clear why it should be more interesting to work with the conditional distribution $q(\mathbf{h}|\mathbf{x})$ and then define a distribution on $\mathbf{h}$ through $q_{\mathcal{D}}$, rather than working directly with a distribution Q on $\mathbf{h}$.

The first answer is practical: optimizing on $P_{\theta_I}(\mathbf{x}|\mathbf{h})$ and on the distribution of $\mathbf{h}$ simultaneously is just the same as optimizing over the global network, while on the other hand the currently used deep architectures provide both $\mathbf{x}|\mathbf{h}$ and $\mathbf{h}|\mathbf{x}$ at the same time.

A second answer is mathematical: $q_{\mathcal{D}}$ is defined through the dataset $\mathcal{D}$. Thus by working on $q(\mathbf{h}|\mathbf{x})$ we can concentrate on the correspondence between $\mathbf{h}$ and $\mathbf{x}$ and not on the full distribution of either, and hopefully this correspondence is easier to describe. Then we use the dataset $\mathcal{D}$ to provide $q_{\mathcal{D}}$: so rather than directly crafting a distribution $Q(\mathbf{h})$, we use a distribution which automatically incorporates aspects of the data distribution $\mathcal{D}$ even for very simple q. Hopefully this is better; we now formalize this argument.

Let us fix the bottom layer parameters θ_I, and consider the problem of finding the best latent marginal over $\mathbf{h}$, i.e., the Q maximizing the data log-likelihood

$$\arg\max_Q \mathbb{E}_{\mathbf{x}\sim P_{\mathcal{D}}}\left[\log \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h})Q(\mathbf{h})\right] \tag{31}$$

Let $Q(\mathbf{h})$ be a candidate distribution. We might build a better one by "reflecting the data" in it. Namely, $Q(\mathbf{h})$ defines a distribution $P_{\theta_I}(\mathbf{x}|\mathbf{h})Q(\mathbf{h})$ on $(\mathbf{x},\mathbf{h})$. This distribution, in turn, defines a conditional distribution of $\mathbf{h}$ knowing $\mathbf{x}$ in the standard way:

$$Q^{\mathrm{cond}}(\mathbf{h}|\mathbf{x}) := \frac{P_{\theta_I}(\mathbf{x}|\mathbf{h})Q(\mathbf{h})}{\sum_{\mathbf{h}'} P_{\theta_I}(\mathbf{x}|\mathbf{h}')Q(\mathbf{h}')} \tag{32}$$

We can turn $Q^{\mathrm{cond}}(\mathbf{h}|\mathbf{x})$ into a new distribution on $\mathbf{h}$ by using the data distribution:

$$Q^{\mathrm{cond}}_{\mathcal{D}}(\mathbf{h}) := \sum_{\mathbf{x}} Q^{\mathrm{cond}}(\mathbf{h}|\mathbf{x}) P_{\mathcal{D}}(\mathbf{x}) \tag{33}$$

and in general $Q^{\mathrm{cond}}_{\mathcal{D}}(\mathbf{h})$ will not coincide with the original distribution $Q(\mathbf{h})$, if only because the definition of the former involves the data whereas Q is arbitrary. We will show that this operation is always an improvement: $Q^{\mathrm{cond}}_{\mathcal{D}}(\mathbf{h})$ always yields a better data log-likelihood than Q.

Proposition 7 *Let* data incorporation *be the map sending a distribution $Q(\mathbf{h})$ to $Q^{\mathrm{cond}}_{\mathcal{D}}(\mathbf{h})$ defined by* (32) *and* (33)*, where θ_I is fixed. It has the following properties:*

- *Data incorporation always increases the data log-likelihood* (31)*.*
- *The best latent marginal $\hat{Q}_{\theta_I,\mathcal{D}}$ is a fixed point of this transformation. More precisely, the distributions Q that are fixed points of data incorporation are exactly the critical points of the data log-likelihood* (31) *(by concavity of* (31) *these critical points are all maxima with the same value). In particular if the BLM is uniquely defined (the arg max in* (13) *is unique), then it is the only fixed point of data incorporation.*
- *Data incorporation $Q \mapsto Q^{\mathrm{cond}}_{\mathcal{D}}$ coincides with one step of the expectation-maximization (EM) algorithm to maximize data log-likelihood by optimizing over Q for a fixed θ_I, with $\mathbf{h}$ as the hidden variable.*

This can be seen as a justification for constructing the hidden variable model Q through an inference model $q(\mathbf{h}|\mathbf{x})$ from the data, which is the basic approach of auto-encoders and the BLM.

Proof Let us first prove the statement about expectation-maximization. Since the EM algorithm is known to increase data log-likelihood at each step (Dempster et al., 1977; Wu, 1983), this will prove the first statement as well.

For simplicity let us assume that the data distribution is uniform over the dataset $\mathcal{D} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$. (Arbitrary data weights can be approximated by putting the same observation several times into the data.) The hidden variable of the EM algorithm will be $\mathbf{h}$, and the parameter over which the EM optimizes will be the distribution $Q(\mathbf{h})$ itself. In particular we keep θ_I fixed. The distributions Q and P_{θ_I} define a distribution $P(\mathbf{x}, \mathbf{h}) := P_{\theta_I}(\mathbf{x}|\mathbf{h})Q(\mathbf{h})$ over pairs $(\mathbf{x}, \mathbf{h})$. This extends to a distribution over n-tuples of observations:

$$P((\mathbf{x}_1, \mathbf{h}_1), \ldots, (\mathbf{x}_n, \mathbf{h}_n)) = \prod_i P_{\theta_I}(\mathbf{x}_i|\mathbf{h}_i)Q(\mathbf{h}_i)$$

and by summing over the states of the hidden variables

$$P(\mathbf{x}_1, \dots, \mathbf{x}_n) = \sum_{(\mathbf{h}_1, \dots, \mathbf{h}_n)} P((\mathbf{x}_1, \mathbf{h}_1), \dots, (\mathbf{x}_n, \mathbf{h}_n))$$

Denote $\vec{\mathbf{x}} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ and $\vec{\mathbf{h}} = (\mathbf{h}_1, \dots, \mathbf{h}_n)$. One step of the EM algorithm operating with the distribution Q as parameter, is defined as transforming the current distribution Q_t into the new distribution

$$Q_{t+1} = \arg\max_{Q} \sum_{\vec{\mathbf{h}}} P_t(\vec{\mathbf{h}}|\vec{\mathbf{x}}) \log P(\vec{\mathbf{x}}, \vec{\mathbf{h}})$$

where $P_t(\vec{\mathbf{x}}, \vec{\mathbf{h}}) = P_{\theta_I}(\vec{\mathbf{x}}|\vec{\mathbf{h}}) Q_t(\vec{\mathbf{h}})$ is the distribution obtained by using Q_t for $\mathbf{h}$, and P the one obtained from the distribution Q over which we optimize. Let us follow a standard argument for EM algorithms on n-tuples of independent observations:

$$\begin{aligned}
\sum_{\vec{\mathbf{h}}} P_t(\vec{\mathbf{h}}|\vec{\mathbf{x}}) \log P(\vec{\mathbf{x}}, \vec{\mathbf{h}}) &= \sum_{\vec{\mathbf{h}}} P_t(\vec{\mathbf{h}}|\vec{\mathbf{x}}) \log \prod_i P(\mathbf{x}_i, \mathbf{h}_i) \\
&= \sum_i \sum_{\vec{\mathbf{h}}} P_t(\vec{\mathbf{h}}|\vec{\mathbf{x}}) \log P(\mathbf{x}_i, \mathbf{h}_i)
\end{aligned}$$

Since observations are independent, $P_t(\vec{\mathbf{h}}|\vec{\mathbf{x}})$ decomposes as a product and so

$$\begin{aligned}
\sum_i \sum_{\vec{\mathbf{h}}} (\log P(\mathbf{x}_i, \mathbf{h}_i)) P_t(\vec{\mathbf{h}}|\vec{\mathbf{x}}) &= \sum_i \sum_{\mathbf{h}_1, \dots, \mathbf{h}_n} (\log P(\mathbf{x}_i, \mathbf{h}_i)) \prod_j P_t(\mathbf{h}_j|\mathbf{x}_j) \\
&= \sum_i \sum_{\mathbf{h}_i} (\log P(\mathbf{x}_i, \mathbf{h}_i)) P_t(\mathbf{h}_i|\mathbf{x}_i) \prod_{j \neq i} \sum_{\mathbf{h}_j} P_t(\mathbf{h}_j|\mathbf{x}_j)
\end{aligned}$$

but of course $\sum_{\mathbf{h}_j} P_t(\mathbf{h}_j|\mathbf{x}_j) = 1$ so that finally

$$\begin{aligned}
\sum_{\vec{\mathbf{h}}} P_t(\vec{\mathbf{h}}|\vec{\mathbf{x}}) \log P(\vec{\mathbf{x}}, \vec{\mathbf{h}}) &= \sum_i \sum_{\mathbf{h}_i} (\log P(\mathbf{x}_i, \mathbf{h}_i)) P_t(\mathbf{h}_i|\mathbf{x}_i) \\
&= \sum_{\mathbf{h}} \sum_i (\log P(\mathbf{x}_i, \mathbf{h})) P_t(\mathbf{h}|\mathbf{x}_i) \\
&= \sum_{\mathbf{h}} \sum_i (\log P_{\theta_I}(\mathbf{x}_i|\mathbf{h}) + \log Q(\mathbf{h})) P_t(\mathbf{h}|\mathbf{x}_i)
\end{aligned}$$

because $P(\mathbf{x}, \mathbf{h}) = P_{\theta_I}(\mathbf{x}|\mathbf{h}) Q(\mathbf{h})$. We have to maximize this quantity over Q. The first term does not depend on Q so we only have to maximize $\sum_{\mathbf{h}} \sum_i (\log Q(\mathbf{h})) P_t(\mathbf{h}|\mathbf{x}_i)$.

This latter quantity is concave in Q, so to find the maximum it is sufficient to exhibit a point where the derivative w.r.t. Q (subject to the constraint that Q is a probability distribution) vanishes.

Let us compute this derivative. If we replace Q with $Q + \delta Q$ where δQ is infinitesimal, the variation of the quantity to be maximized is

$$\sum_{\mathbf{h}} \sum_i (\delta \log Q(\mathbf{h})) P_t(\mathbf{h}|\mathbf{x}_i) = \sum_{\mathbf{h}} \frac{\delta Q(\mathbf{h})}{Q(\mathbf{h})} \sum_i P_t(\mathbf{h}|\mathbf{x}_i)$$

Let us take $Q = (Q_t)^{\mathrm{cond}}_{\mathcal{D}}$. Since we assumed for simplicity that the data distribution $\mathcal{D}$ is uniform over the sample this $(Q_t)^{\mathrm{cond}}_{\mathcal{D}}$ is

$$Q(\mathbf{h}) = (Q_t)^{\mathrm{cond}}_{\mathcal{D}}(\mathbf{h}) = \frac{1}{n} \sum_i P_t(\mathbf{h}|\mathbf{x}_i)$$

so that the variation of the quantity to be maximized is

$$\sum_{\mathbf{h}} \frac{\delta Q(\mathbf{h})}{Q(\mathbf{h})} \sum_i P_t(\mathbf{h}|\mathbf{x}_i) = n \sum_{\mathbf{h}} \delta Q(\mathbf{h})$$

But since Q and $Q + \delta Q$ are both probability distributions, both sum to 1 over $\mathbf{h}$ so that $\sum_{\mathbf{h}} \delta Q(\mathbf{h}) = 0$. This proves that this choice of Q is an extremum of the quantity to be maximized.

This proves the last statement of the proposition. As mentioned above, it implies the first by the general properties of EM. Once the first statement is proven, the best latent marginal $\hat{Q}_{\theta_I,\mathcal{D}}$ has to be a fixed point of data incorporation, because otherwise we would get an even better distribution thus contradicting the definition of the BLM.

The only point left to prove is the equivalence between critical points of the log-likelihood and fixed points of $Q \mapsto Q^{\mathrm{cond}}_{\mathcal{D}}$. This is a simple instance of maximization under constraints, as follows. Critical points of the data log-likelihood are those for which the log-likelihood does not change at first order when Q is replaced with $Q + \delta Q$ for small δQ. The only constraint on δQ is that $Q + \delta Q$ must still be a probability distribution, so that $\sum_{\mathbf{h}} \delta Q(\mathbf{h}) = 0$ because both Q and $Q + \delta Q$ sum to 1.

The first-order variation of log-likelihood is

$$
\begin{aligned}
\delta \sum_i \log P(\mathbf{x}_i) &= \delta \sum_i \log \left(\sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}_i|\mathbf{h}) Q(\mathbf{h}) \right) \\
&= \sum_i \frac{\delta \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}_i|\mathbf{h}) Q(\mathbf{h})}{\sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}_i, \mathbf{h}) Q(\mathbf{h})} \\
&= \sum_i \frac{\sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}_i|\mathbf{h}) \delta Q(\mathbf{h})}{P(\mathbf{x}_i)} \\
&= \sum_{\mathbf{h}} \delta Q(\mathbf{h}) \sum_i \frac{P_{\theta_I}(\mathbf{x}_i|\mathbf{h})}{P(\mathbf{x}_i)} \\
&= \sum_{\mathbf{h}} \delta Q(\mathbf{h}) \sum_i \frac{P(\mathbf{x}_i, \mathbf{h})/Q(\mathbf{h})}{P(\mathbf{x}_i)} \\
&= \sum_{\mathbf{h}} \delta Q(\mathbf{h}) \sum_i \frac{P(\mathbf{h}|\mathbf{x}_i)}{Q(\mathbf{h})}
\end{aligned}
$$

This must vanish for any δQ such that $\sum_{\mathbf{h}} \delta Q(\mathbf{h}) = 0$. By elementary linear algebra (or Lagrange multipliers) this occurs if and only if $\sum_i \frac{P(\mathbf{h}|\mathbf{x}_i)}{Q(\mathbf{h})}$ does not depend on $\mathbf{h}$, i.e., if and only if Q satisfies $Q(\mathbf{h}) = C \sum_i P(\mathbf{h}|\mathbf{x}_i)$. Since Q sums to 1 one finds $C = \frac{1}{n}$. Since all along P is the probability distribution on $\mathbf{x}$ and $\mathbf{h}$ defined by Q and $P_{\theta_I}(\mathbf{x}|\mathbf{h})$, namely, $P(\mathbf{x}, \mathbf{h}) = P_{\theta_I}(\mathbf{x}|\mathbf{h})Q(\mathbf{h})$, by definition we have $P(\mathbf{h}|\mathbf{x}) = Q^{\mathrm{cond}}(\mathbf{h}|\mathbf{x})$ so that the condition $Q(\mathbf{h}) = \frac{1}{n} \sum_i P(\mathbf{h}|\mathbf{x}_i)$ exactly means that $Q = Q_{\mathcal{D}}^{\mathrm{cond}}$, hence the equivalence between critical points of log-likelihood and fixed points of data incorporation. ■

3. Applications and Experiments

Given the approach described above, we now consider several applications for which we evaluate the method empirically.

The intractability of the log-likelihood for deep networks makes direct comparison of several methods difficult in general. Often the evaluation is done by using latent variables as features for a classification task and by direct visual comparison of samples generated by the model (Larochelle et al., 2009; Salakhutdinov and Hinton, 2009). Instead, we introduce two new datasets which are simple enough for the true log-likelihood to be computed explicitly, yet complex enough to be relevant to deep learning.

We first check that these two datasets are indeed deep.

Then we try to assess the impact of the various approximations from theory to practice, on the validity of the approach.

We then apply our method to the training of deep belief networks using properly modified auto-encoders, and show that the method outperforms current state of the art.

We also explore the use of the BLM upper bound to perform layer-wise hyper-parameter selection and show that it gives an accurate prediction of the future log-likelihood of models.

3.1 Low-Dimensional Deep Datasets

We now introduce two new deep datasets of low dimension. In order for those datasets to give a reasonable picture of what happens in the general case, we first have to make sure that they are relevant to deep learning, using the following approach:

1. In the spirit of (Bergstra and Bengio, 2012), we train 1000 RBMs using CD-1 (Hinton., 2002) on the dataset $\mathcal{D}$, and evaluate the log-likelihood of a disjoint validation dataset $\mathcal{V}$ under each model.
2. We train 1000 2-layer deep networks using stacked RBMs trained with CD-1 on $\mathcal{D}$, and evaluate the log-likelihood of $\mathcal{V}$ under each model[6].
3. We compare the performance of each model at equal number of parameters.
4. If deep networks consistently outperform single RBMs for the same number of parameters, the dataset is considered to be deep.

The comparison at equal number of parameters is justified by one of the main hypotheses of deep learning, namely that deep architectures are capable of representing some functions more compactly than shallow architectures (Bengio and LeCun, 2007).

Hyper-parameters taken into account for hyper-parameter random search are the hidden layers sizes, CD learning rate and number of CD epochs. The corresponding priors are given in Table 1. In order not to give an obvious head start to deep networks, the possible layer sizes are chosen so that the maximum number of parameters for the single RBM and the deep network are as close as possible.

Cmnist dataset

The Cmnist dataset is a low-dimensional variation on the Mnist dataset (LeCun et al., 1998), containing 12,000 samples of dimension 100. The full dataset is split into training, validation and test sets of 4,000 samples each. The dataset is obtained by taking a 10 × 10 image at the center of each Mnist sample and using the values

[6] for stacked RBMs, the exact log-likelihood is computed by summing over the hidden states; the log-likelihood of RBMs is evaluated using Annealed Importance Sampling (AIS) (Salakhutdinov and Murray, 2008)

Parameter	Prior
RBM hidden layer size	1 to 19
Deep Net hidden layer 1 size	1 to 16
Deep Net hidden layer 2 size	1 to 16
inference hidden layer size	1 to 500
CD learn rate	$\log U(10^{-5}, 5 \times 10^{-2})$
BP learn rate	$\log U(10^{-5}, 5 \times 10^{-2})$
CD epochs	$20 \times (10000/N)$
BP epochs	$20 \times (10000/N)$
ANN init σ	$U(0, 1)$

Table 1: Search space for hyper-parameters when using random search for a dataset of size N.

in [0,1] as probabilities. The first 10 samples of the Cmnist dataset are shown in Figure 2.

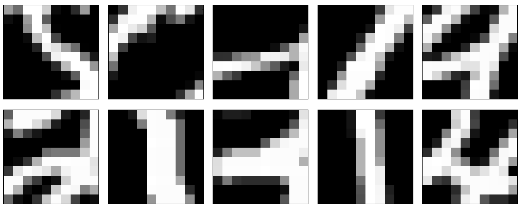


Figure 2: First 10 samples of the Cmnist dataset.

We propose two baselines to which to compare the log-likelihood values of models trained on the Cmnist dataset:

1. The uniform coding scheme: a model which gives equal probability to all possible binary 10 × 10 images. The log-likelihood of each sample is then -100 bits, or -69.31 nats.

2. The independent Bernoulli model in which each pixel is given an independent Bernoulli probability. The model is trained on the training set. The log-likelihood of the validation set is -67.38 nats per sample.

The comparison of the log-likelihood of stacked RBMs with that of single RBMs is presented in Figure 3 and confirms that the Cmnist dataset is deep.

Tea dataset

The Tea dataset is based on the idea of learning an invariance for the amount of liquid in several containers: a teapot and 5 teacups. It contains 243 distinct samples

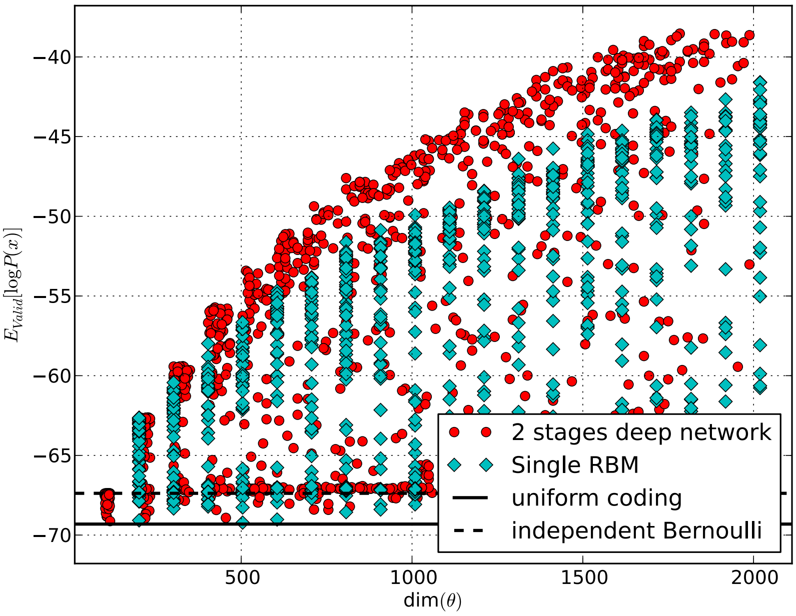


Figure 3: Checking that Cmnist is deep: log-likelihood of the validation dataset $\mathcal{V}$ under RBMs and SRBM deep detworks selected by hyper-parameter random search, as a function of the number of parameters dim(θ).

which are then distributed into a training, validation and test set of 81 samples each. The dataset consists of 10×10 images in which the left part of the image represents a (stylized) teapot of size 10×5. The right part of the image represents 5 teacups of size 2×5. The liquid is represented by ones and always lies at the bottom of each container. The total amount of liquid is always equal to the capacity of the teapot, i.e., there are always 50 ones and 50 zeros in any given sample. The first 10 samples of the Tea dataset are shown in Figure 4.

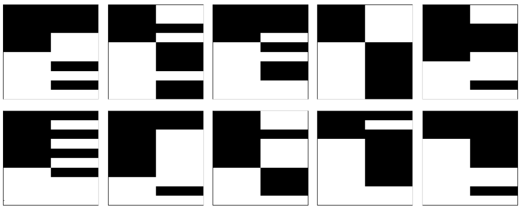


Figure 4: First 10 samples of the Tea dataset.

In order to better interpret the log-likelihood of models trained on the Tea dataset, we propose 3 baselines:

1. The uniform coding scheme: the baseline is the same as for the Cmnist dataset: -69.31 nats.

2. The independent Bernoulli model, adjusted on the training set. The log-likelihood of the validation set is -49.27 nats per sample.

3. The perfect model in which all 243 samples of the full dataset (consituted by concatenation of the training, validation and test sets) are given the probability $\frac{1}{243}$. The expected log-likelihood of a sample from the validation dataset is then $\log(\frac{1}{243}) = -5.49$ nats.

The comparison of the log-likelihood of stacked RBMs and single RBMs is presented in Figure 5 and confirms that the Tea dataset is deep.

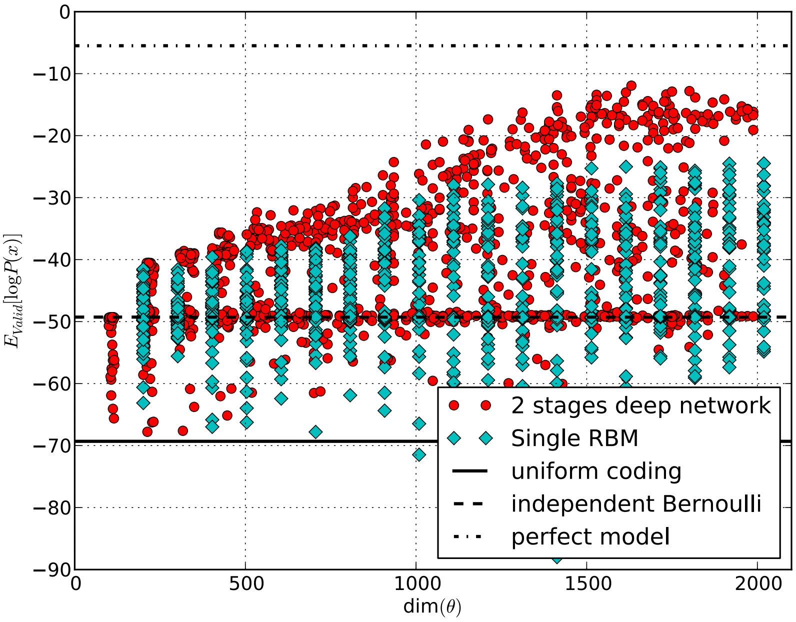


Figure 5: Checking that Tea is deep: log-likelihood of the validation dataset $\mathcal{V}$ under RBMs and SRBM deep networks selected by hyper-parameter random search, as a function of the number of parameters $\dim(\theta)$.

3.2 Deep Generative Auto-Encoder Training

A first application of our approach is the training of a deep *generative* model using auto-associators. To this end, we propose to train lower layers using auto-associators and to use an RBM for the generative top layer model.

We will compare three kinds of deep architectures: standard auto-encoders with an RBM on top (vanilla AEs), the new *auto-encoders with rich inference* (AERIes) suggested by our framework, also with an RBM on top, and, for comparison, stacked restricted Boltzmann machines (SRBMs). All the models used in this study use the same final generative model class for $P(\mathbf{x}|\mathbf{h})$ so that the comparison focuses on the training procedure, on equal ground. SRBMs are considered the state of the art (Hinton et al., 2006; Bengio et al., 2007)—although performance can be increased using richer models (Bengio et al., 2012), our focus here is not on the model but on the layer-wise training procedure for a given model class.

In ideal circumstances, we would have compared the log-likelihood obtained for each training algorithm with the optimum of a deep learning procedure such as the full gradient ascent procedure (Section 2). Instead, since this ideal deep learning procedure is intractable, SRBMs serve as a reference.

The new AERIes are auto-encoders modified after the following remark: the complexity of the inference model used for $q(\mathbf{h}|\mathbf{x})$ can be increased safely without risking overfit and loss of generalization power, because q is not part of the final generative model, and is used only as a tool for optimization of the generative model parameters. This would suggest that the complexity of q could be greatly increased with only positive consequences on the performance of the model.

AERIes exploit this possibility by having, in each layer, a modified auto-associator with two hidden layers instead of one: $\mathbf{x} \to \mathbf{h}' \to \mathbf{h} \to \mathbf{x}$. The generative part $P_{\theta_I}(\mathbf{x}|\mathbf{h})$ will be equivalent to that of a regular auto-associator, but the inference part $q(\mathbf{h}|\mathbf{x})$ will have greater representational power because it includes the hidden layer $\mathbf{h}'$ (see Figure 7).

We will also use the more usual auto-encoders composed of auto-associators with one hidden layer and tied weights, commonly encountered in the literature (vanilla AE).

For all models, the deep architecture will be of depth 2. The stacked RBMs will be made of two ordinary RBMs. For AERIes and vanilla AEs, the lower part is made of a single auto-associator (modified for AERies), and the generative top part is an RBM. (Thus they have one layer less than depicted for the sake of generality in Figures 6 and 7.) For AERIes and vanilla AEs the lower part of the model is trained using the usual backpropagation algorithm with cross-entropy loss, which performs gradient ascent for the probability of (25). The top RBM is then trained to maximize (12).

The competitiveness of each model will be evaluated through a comparison in log-likelihood over a validation set distinct from the training set. Comparisons are made for a given identical number of parameters of the generative model[7]. Each model will be given equal chance to find a good optimum in terms of the number of evaluations in a hyper-parameter selection procedure by random search.

When implementing the training procedure proposed in Section 2, several approximations are needed. An important one, compared to Theorem 1, is that the distribution $q(\mathbf{h}|\mathbf{x})$ will not really be trained over all possible conditional distributions for $\mathbf{h}$ knowing $\mathbf{x}$. Next, training of the upper layers will of course fail to reproduce the BLM perfectly. Moreover, auto-associators use an $\mathbf{x} = \tilde{\mathbf{x}}$ approximation, cf. (25). We will study the effect of these approximations.

Let us now provide more details for each model.

Stacked RBMs. For our comparisons, 1000 stacked RBMs were trained using the procedure from (Hinton et al., 2006). We used random search on the hyper-parameters, which are: the sizes of the hidden layers, the CD learning rate, and the number of CD epochs.

Vanilla auto-encoders. The general training algorithm for vanilla auto-encoders is depicted in Figure 6. First an auto-associator is trained to maximize the adaptation of the BLM upper bound for auto-associators presented in (25). The maximization procedure itself is done with the backpropagation algorithm and cross-entropy loss. The inference weights are tied to the generative weights so that $\mathbf{W}_{\text{gen}} = \mathbf{W}_{\text{inf}}^{\top}$ as is often the case in practice. An ordinary RBM is used as a generative model on the top layer.

1000 deep generative auto-encoders were trained using random search on the hyper-parameters. Because deep generative auto-encoders use an RBM as the top layer, they use the same hyper-parameters as stacked RBMs, but also backpropagation (BP) learning rate, BP epochs, and ANN init σ (i.e. the standard deviation of the gaussian used during initialization).

Auto-Encoders with Rich Inference (AERIes). The model and training scheme for AERIes are represented in Figure 7. Just as for vanilla auto-encoders, we use the backpropagation algorithm and cross-entropy loss to maximize the auto-encoder version (25) of the BLM upper bound on the training set. No weights are tied, of course, as this does not make sense for an auto-associator with different models for $P(\mathbf{x}|\mathbf{h})$ and $q(\mathbf{h}|\mathbf{x})$. The top RBM is trained afterwards. Hyper-parameters are the same as above, with in addition the size of the new hidden layer $\mathbf{h}'$.

[7] Because we only consider the generative models obtained, q is never taken into account in the number of parameters of an auto-encoder or SRBM. However, the parameters of the top RBM are taken into account as they are a necessary part of the generative model.

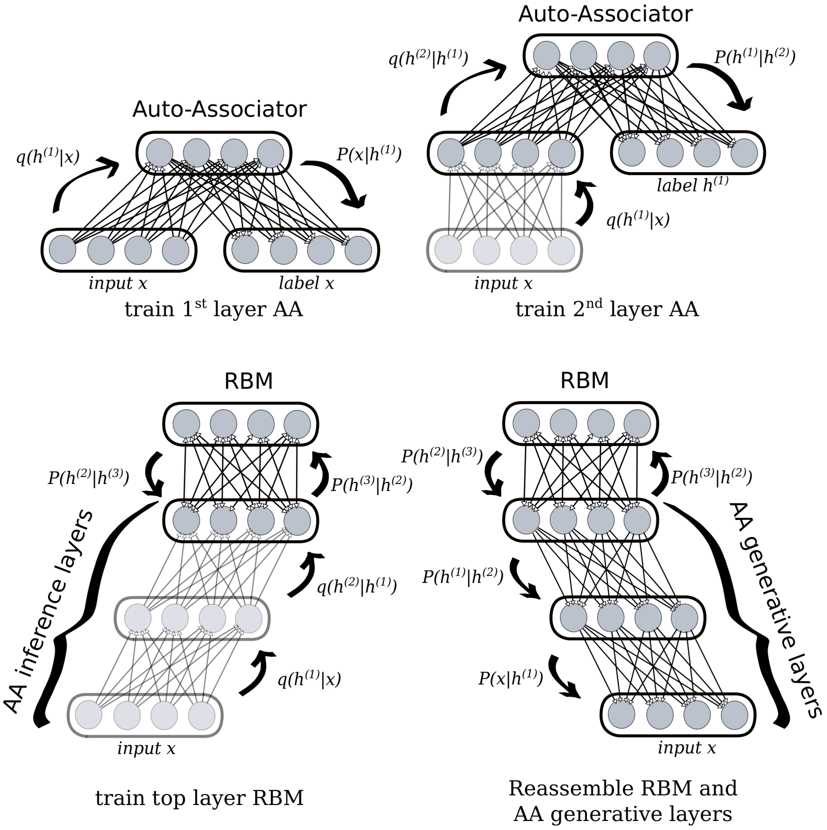


Figure 6: Deep generative auto-encoder training scheme.

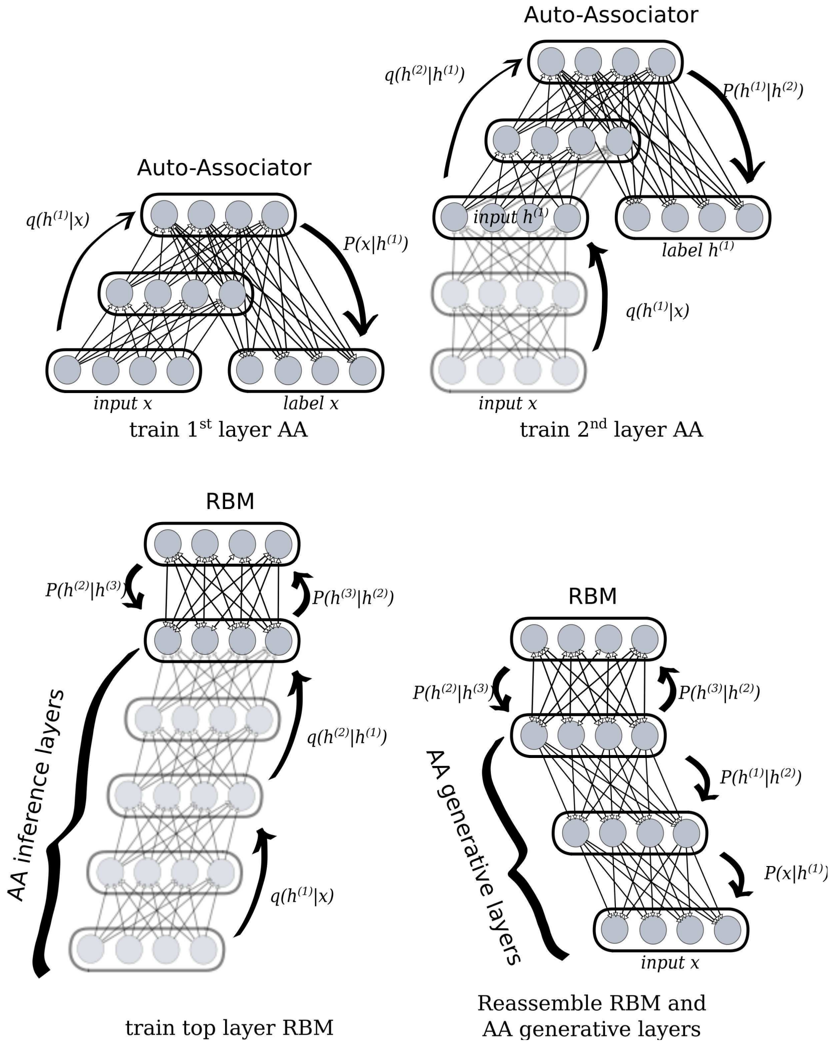


Figure 7: Deep generative modified auto-encoder (AERI) training scheme.

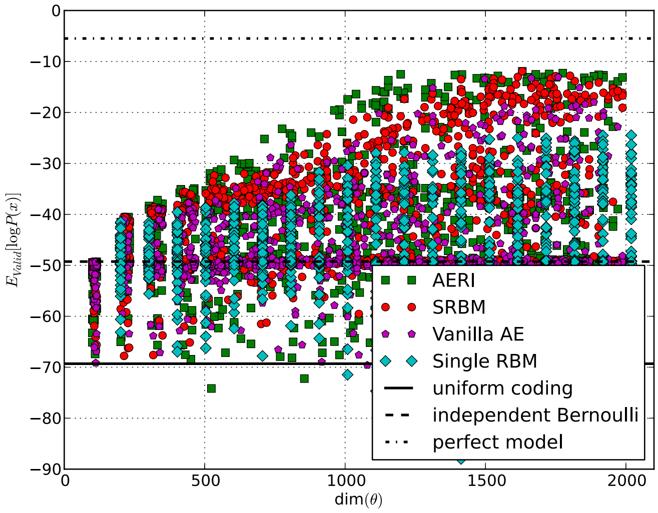


Figure 8: Comparison of the average validation log-likelihood for SRBMs, vanilla AE, and AERIes on the Tea dataset.

Results

The results of the above comparisons on the Tea and Cmnist *validation* datasets are given in Figures 8 and 9. For better readability, the Pareto front[8] for each model is given in Figures 10 and 11.

As expected, all models perform better than the baseline independent Bernoulli model but have a lower likelihood than the perfect model[9]. Also, SRBMs, vanilla AEs and AERIes perform better than a single RBM, which can be seen as further evidence that the Tea and Cmnist are deep datasets.

Among deep models, vanilla auto-encoders achieve the lowest performance, but outperform single RBMs significantly, which validates them not only as generative models but also as *deep* generative models. Compared to SRBMs, vanilla auto-encoders achieve almost identical performance but the algorithm clearly suffers from

[8] The Pareto front is composed of all models which are not subsumed by other models according to the number of parameters and the expected log-likelihood. A model is said to be subsumed by another if it has strictly more parameters and a worse likelihood.

[9] Note that some instances are outperformed by the uniform coding scheme, which may seem surprising. Because we are considering the average log-likelihood on a validation set, if even one sample of the validation set happens to be given a low probability by the model, the average log-likelihood will be arbitrarily low. In fact, because of roundoff errors in the computation of the log-likelihood, a few models have a measured performance of $-\infty$. This does not affect the comparison of the models as it only affects instances for which performance is already very low.

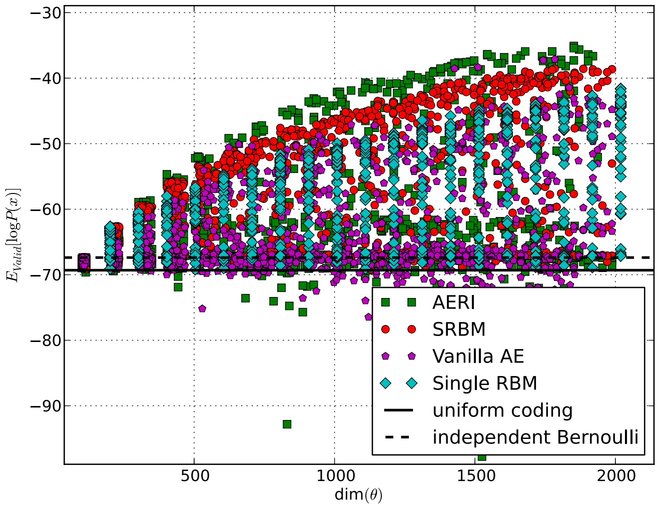


Figure 9: Comparison of the average validation log-likelihood for SRBMs, vanilla AE, and AERIes on the Cmnist dataset.

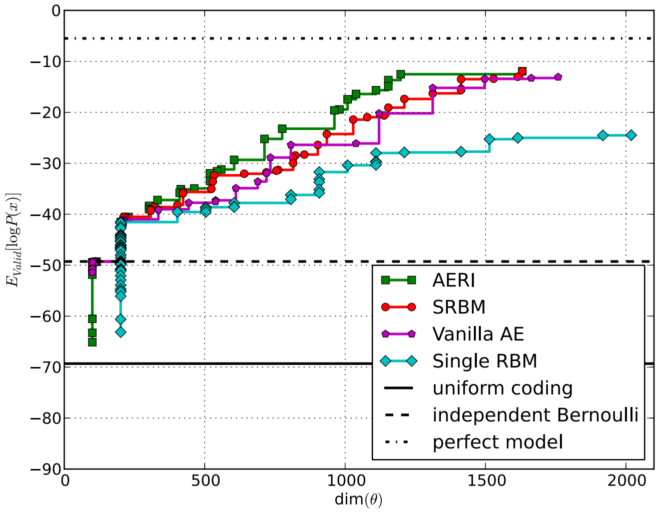


Figure 10: Pareto fronts for the average validation log-likelihood and number of parameters for SRBMs, deep generative auto-encoders, and modified deep generative auto-encoders on the Tea dataset.

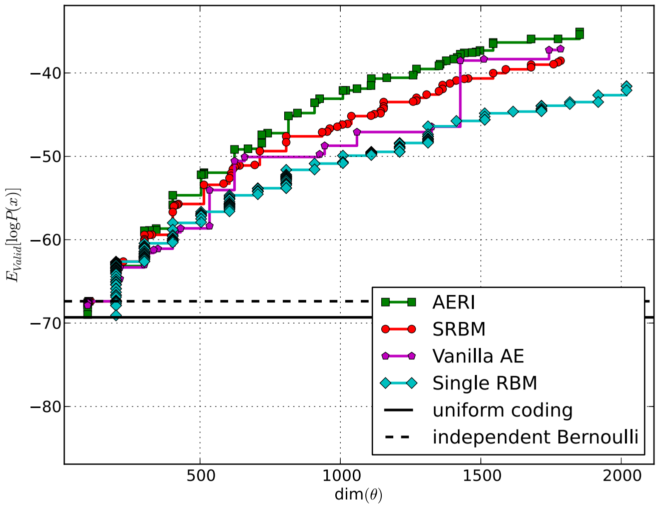


Figure 11: Pareto fronts for the average validation log-likelihood and number of parameters for SRBMs, deep generative auto-encoders, and modified deep generative auto-encoders on the Cmnist dataset.

local optima: most instances perform poorly and only a handful achieve performance comparable to that of SRBMs or AERIes.

As for the auto-encoders with rich inference (AERIes), they are able to outperform not only single RBMs and vanilla auto-encoders, but also stacked RBMs, and do so consistently. This validates not only the general deep learning procedure of Section 2, but arguably also the understanding of auto-encoders in this framework.

The results confirm that a more universal model for q can significantly improve the performance of a model, as is clear from comparing the vanilla and rich-inference auto-encoders. Let us insist that the rich-inference auto-encoders and vanilla auto-encoders optimize over exactly the *same* set of generative models with the same structure, and thus are facing exactly the same optimization problem (4). Clearly the modified training procedure yields improved values of the generative parameter θ.

3.3 Layer-Wise Evaluation of Deep Belief Networks

As seen in section 2, the BLM upper bound $U_{\mathcal{D}}(\theta_I)$ is the least upper bound of the log-likelihood of deep generative models using some given θ_I in the lower part of the model. This raises the question of whether it is a good indicator of the final performance of θ_I.

In this setting, there are a few approximations w.r.t. (10) and (12) that need to be discussed. Another point is the intractability of the BLM upper bound for models with many hidden variables, which leads us to propose and test an estimator in Section 3.3.4, though the experiments considered here were small enough not to need this unless otherwise specified.

We now look, in turn, at how the BLM upper bound can be applied to log-likelihood estimation, and to hyper-parameter selection—which can be considered part of the training procedure. We first discuss various possible effects, before measuring them empirically.

3.3.1 Approximations in the BLM upper bound

Consider the maximization of (14). In practice, we do not perform a specific maximization over q to obtain the BLM as in (14), but rely on the training procedure of θ_I to maximize it. Thus the q resulting from a training procedure is generally not the globally optimal $\hat{q}$ from Theorem 1. In the experiments we of course use the BLM upper bound with the value of q resulting from the actual training.

Definition 8 *For θ_I and q resulting from the training of a deep generative model, let*

$$\hat{\mathcal{U}}_{\mathcal{D},q}(\theta_I) := \mathbb{E}_{\mathbf{x}\sim P_{\mathcal{D}}}\left[\log \sum_{\mathbf{h}} P_{\theta_I}(\mathbf{x}|\mathbf{h}) q_{\mathcal{D}}(\mathbf{h})\right] \tag{34}$$

be the empirical BLM upper bound.

This definition makes no assumption about how θ_I and q in the first layer have been trained, and can be applied to any layer-wise training procedure, such as SRBMs.

Ideally, this quantity should give us an idea of the final performance of the deep architecture when we use θ_I on the bottom layer. But there are several discrepancies between these BLM estimates and final performance.

A first question is the validity of the approximation (34). The BLM upper bound $\mathcal{U}_{\mathcal{D}}(\theta_I)$ is obtained by maximization over all possible q which is of course untractable. The learned inference distribution q used in practice is only an approximation for two reasons: first, because the model for q may not cover all possible conditional distributions $q(\mathbf{h}|\mathbf{x})$, and, second, because the training of q can be imperfect. In effect $\hat{\mathcal{U}}_{\mathcal{D},q}(\theta_I)$ is only a lower bound of the BLM upper bound : $\hat{\mathcal{U}}_{\mathcal{D},q}(\theta_I) \leqslant \mathcal{U}_{\mathcal{D}}(\theta_I)$.

Second, we can question the relationship between the (un-approximated) BLM upper bound (14) and the final log-likelihood of the model. The BLM bound is optimistic, and tight only when the upper part of the model manages to reproduce the BLM perfectly. We should check how tight it is in practical applications when the upper layer model for $P(\mathbf{h})$ is imperfect.

In addition, as for any estimate from a training set, final performance on validation and test sets might be different. Performance of a model on the validation set is generally lower than on the training set. But on the other hand, in our situation there is a specific *regularizing effect of imperfect training of the top layers*. Indeed the BLM refers to a universal optimization over all possible distributions on $\mathbf{h}$ and might therefore overfit more, hugging the training set too closely. Thus if we did manage to reproduce the BLM perfectly on the training set, it could well decrease performance on the validation set. On the other hand, training the top layers to approximate the BLM within a model class P_{θ_J} introduces further regularization and could well yield higher final performance on the validation set than if the exact BLM distribution had been used.

This latter regularization effect is relevant if we are to use the BLM upper bound for hyper-parameter selection, a scenario in which regularization is expected to play an important role.

We can therefore expect:

1. That the ideal BLM upper bound, being by definition optimistic, can be higher that the final likelihood when the model obtained for $P(\mathbf{h})$ is not perfect.

2. That the empirical bound obtained by using a given conditional distribution q will be lower than the ideal BLM upper bound either when q belongs to a restricted class, or when q is poorly trained.

3. That the ideal BLM upper bound on the training set may be either higher or lower than actual performance on a validation set, because of the regularization effect of imperfect top layer training.

All in all, the relationship between the empirical BLM upper bound used in training, and the final log-likelihood on real data, results from several effects going in both directions. This might affect whether the empirical BLM upper bound can really be used to predict the future performance of a given bottom layer setting.

3.3.2 A method for single-layer evaluation and layer-wise hyper-parameter selection

In the context of deep architectures, hyper-parameter selection is a difficult problem. It can involve as much as 50 hyper-parameters, some of them only relevant conditionally to others (Bergstra et al., 2011; Bergstra and Bengio, 2012). To make matters worse, evaluating the generative performance of such models is often intractable. The evaluation is usually done w.r.t. classification performance as in (Larochelle et al., 2009; Bergstra et al., 2011; Bergstra and Bengio, 2012), sometimes complemented by a visual comparison of samples from the model (Hinton et al., 2006; Salakhutdinov and Hinton, 2009). In some rare instances, a variational lower-bound of the log-likelihood of the deep model is considered (Salakhutdinov and Murray, 2008; Salakhutdinov and Hinton, 2009).

These methods only consider evaluating the models after all layers have been fully trained. However, since the training of deep architectures is done in a layer-wise fashion, with some criterion greedily maximized at each step, it would seem reasonable to perform a layer-wise evaluation. This would have the advantage of reducing the size of the hyper-parameter search space from exponential to linear in the number of layers.

We propose to first evaluate the performance of the lower layer, after it has been trained, according to the BLM upper bound (34) (or an approximation thereof) on the validation dataset $\mathcal{D}_{\text{valid}}$. The measure of performance obtained can then be used as part of a larger hyper-parameter selection algorithm such as (Bergstra and Bengio, 2012; Bergstra et al., 2011). This results in further optimization of (10) over the hyper-parameter space and is therefore justified by Theorem 1.

Evaluating the top layer is less problematic: by definition, the top layer is always a "shallow" model for which the true likelihood becomes more easily tractable. For instance, although RBMs are well known to have an intractable partition function which prevents their evaluation, several methods are able to compute close approximations to the true likelihood (such as Annealed Importance Sampling (Neal, 1998; Salakhutdinov and Murray, 2008)). The dataset to be evaluated with this procedure will have to be a sample of $\sum_{\mathbf{x}} q(\mathbf{h}|\mathbf{x})P_{\mathcal{D}}(\mathbf{x})$.

In summary, the evaluation of a two-layer generative model can be done in a layer-wise manner:

1. Perform hyper-parameter selection on the lower layer using $\hat{\mathcal{U}}_{\theta_I}(\mathcal{D})$ as a performance measure (preferably on a validation rather than training dataset, see below), and keep only the best possible lower layers according to this criterion[10].

2. Perform hyper-parameter selection on the upper layer by evaluating the true likelihood of validation data samples transformed by the inference distribution, under the model of the top layer[11].

Hyper-parameter selection was not used in our experiments, where we simply used hyper-parameter random search. (This has allowed, in particular, to check the robustness of the models, as AERIes have been found to perform better than vanilla AEs on many more instances over hyper-parameter space.)

As mentioned earlier, in the context of representation learning the top layer is irrelevant because the objective is not to train a generative model but to get a better representation of the data. With the assumption that good latent variables make

[10] In practice an approximation must be used as discussed later

[11] This could lead to a stopping criterion when training a model with arbitrarily many layers: for the upper layer, compare the likelihood of the best upper-model with the BLM of the best possible next layer. If the BLM of the next layer is not significatively higher than the likelihood of the upper-model, then we do not add another layer as it would not help to achieve better performance.

good representations, this suggests that the BLM upper bound can be used directly to select the best possible lower layers.

3.3.3 Testing the BLM and its approximations

We now present a series of tests to check whether the selection of lower layers with higher values of the BLM actually results in higher log-likelihood for the final deep generative models, and to assess the quantitative importance of each of the BLM approximations discussed earlier.

For each training algorithm (SRBMs, RBMs, AEs, AERIes), the comparison is done using 1000 models trained with hyper-parameters selected through random search as before. The empirical BLM upper bound is computed using (34) above.

Training BLM upper bound vs training log-likelihood. We first compare the value of the empirical BLM upper bound $\hat{\mathcal{U}}_{\theta_I}(\mathcal{D}_{\text{train}})$ over the training set, with the actual log-likelihood of the trained model on the training set. This is an evaluation of how optimistic the BLM is for a given dataset, by checking how closely the training of the upper layers manages to match the target BLM distribution on $\mathbf{h}$. This is also the occasion to check the effect of using the $q(\mathbf{h}|\mathbf{x})$ resulting from actual learning, instead of the best q in all possible conditional distributions.

In addition, as discussed below, this comparison can be used as a criterion to determine whether more layers should be added to the model.

The results are given in Figures 12 and 13 for SRBMs, and 14 and 15 for AERIes. We see that the empirical BLM upper bound (34) is a good predictor of the future log-likelihood of the full model on the *training* set. This shows that the approximations w.r.t. the optimality of the top layer and the universality of q can be dealt with in practice.

For AERIes, a few models with low performance have a poor estimation of the BLM upper bound (estimated to be lower than the actual likelihood), presumably because of a bad approximation in the learning of q. This will not affect model selection procedures as it only concerns models with very low performance, which are to be discarded.

If the top part of the model were not powerful enough (e.g., if the network is not deep enough), the BLM upper bound would be too optimistic and thus significantly higher than the final log-likelihood of the model. To further test this intuition we now compare the BLM upper bound of the bottom layer with the log-likelihood obtained by a shallow architecture with *only one layer*; the difference would give an indication of how much could be gained by adding top layers. Figures 16 and 17 compare the expected log-likelihood[12] of the training set under the 1000 RBMs previously trained with the BLM upper bound[13] for a generative model using this

[12] The log-likelihood reported in this specific experiment is in fact obtained with Annealed Importance Sampling (AIS).

[13] The BLM upper bound value given in this particular experiment is in fact a close approximation (see Section 3.3.4).

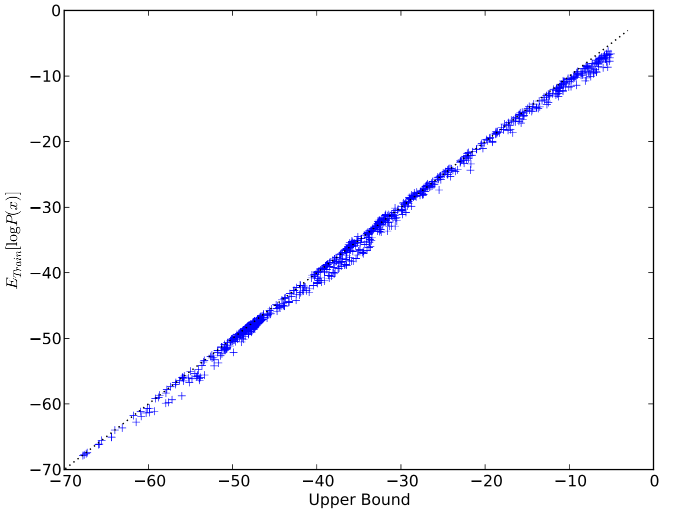


Figure 12: Comparison of the BLM upper bound on the first layer and the final log-likelihood on the Tea training dataset, for 1000 2-layer SRBMs

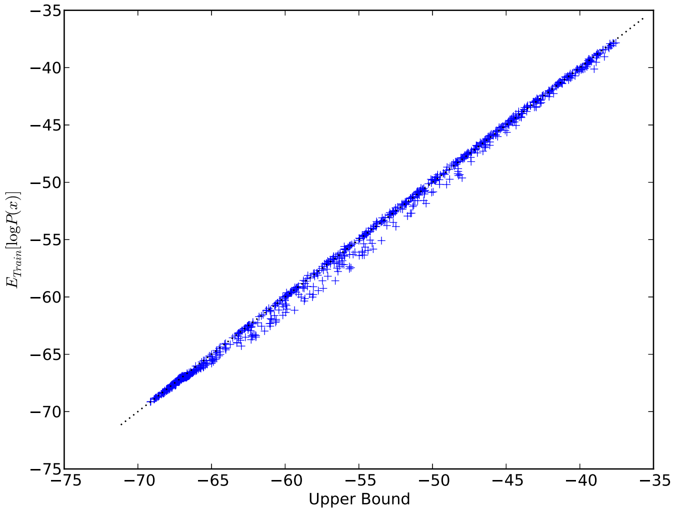


Figure 13: Comparison of the BLM upper bound on the first layer and the final log-likelihood on the Cmnist training dataset, for 1000 2-layer SRBMs

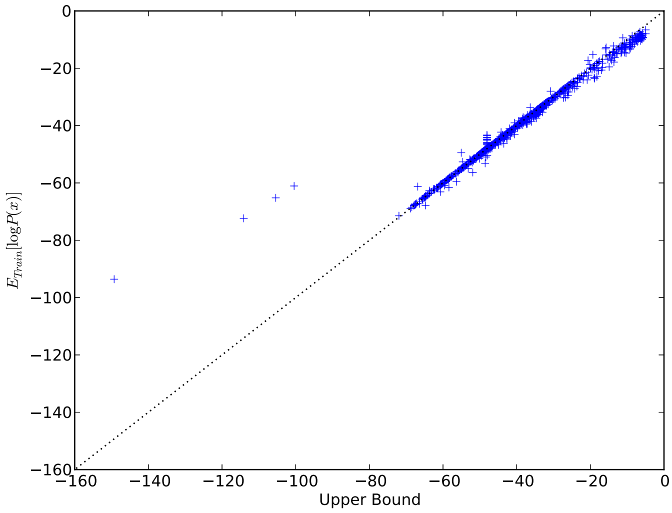


Figure 14: Comparison of the BLM upper bound on the first layer and the final log-likelihood on the Tea training dataset, for 1000 2-layer AERIes

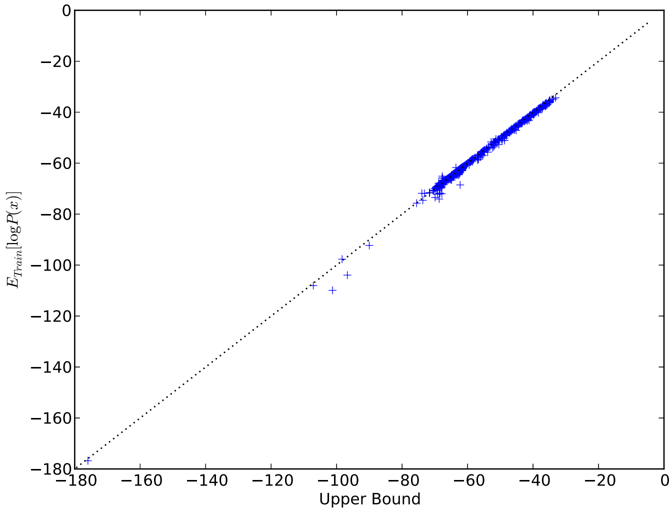


Figure 15: Comparison of the BLM upper bound on the first layer and the final log-likelihood on the Cmnist training dataset, for 1000 2-layer AERIes

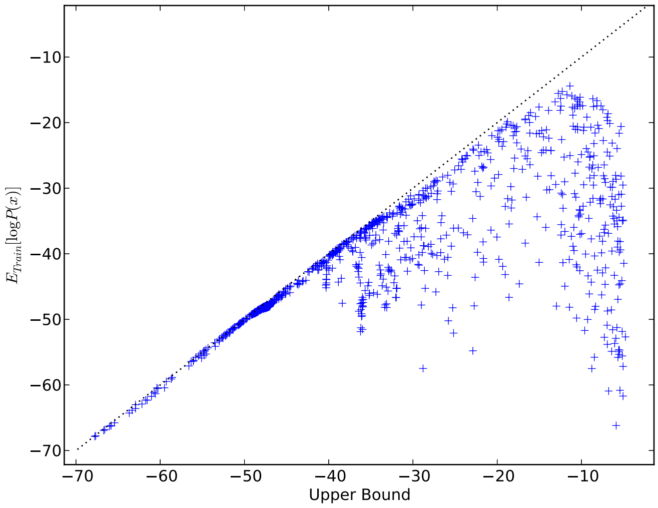


Figure 16: BLM on a too shallow model: comparison of the BLM upper bound and the AIS log-likelihood of an RBM on the Tea training dataset

RBM as first layer. The results contrast with the previous ones and confirm that final performance is below the BLM upper bound when the model does not have enough layers.

The alignment in Figures 12 and 13 can therefore be seen as a confirmation that the Tea and Cmnist datasets would not benefit from a third layer.

Thus, the BLM upper bound could be used as a test for the opportunity of adding layers to a model.

Training BLM upper bound vs validation log-likelihood. We now compare the training BLM upper bound with the log-likelihood on a *validation* set distinct from the training set: this tests whether the BLM obtained during training is a good indication of the final performance of a bottom layer parameter.

As discussed earlier, because the BLM makes an assumption where there is no regularization, using the training BLM upper bound to predict performance on a validation set could be too optimistic: therefore we expect the validation log-likelihood to be somewhat lower than the training BLM upper bound. (Although, paradoxically, this can be somewhat counterbalanced by imperfect training of the upper layers, as mentioned above.)

The results are reported in Figures 18 and 19 and confirm that the training BLM is an upper bound of the validation log-likelihood. As for regularization, we can see that on the Cmnist dataset where there are 4000 samples, generalization is not very

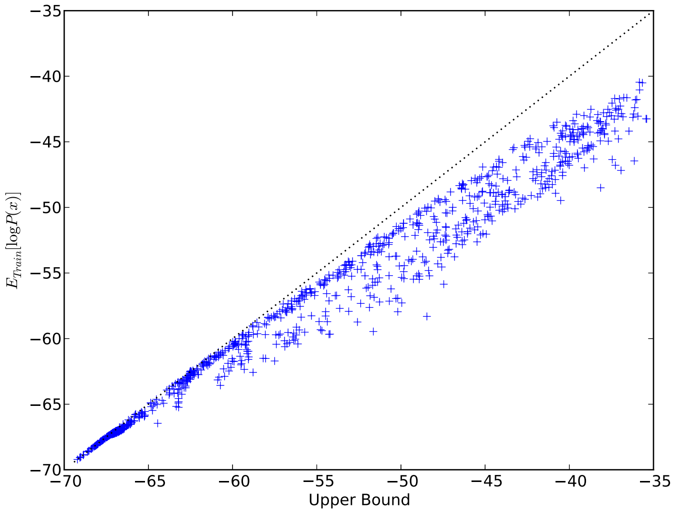


Figure 17: BLM on a too shallow model: Comparison of the BLM upper bound and the AIS log-likelihood of an RBM on the Cmnist training dataset

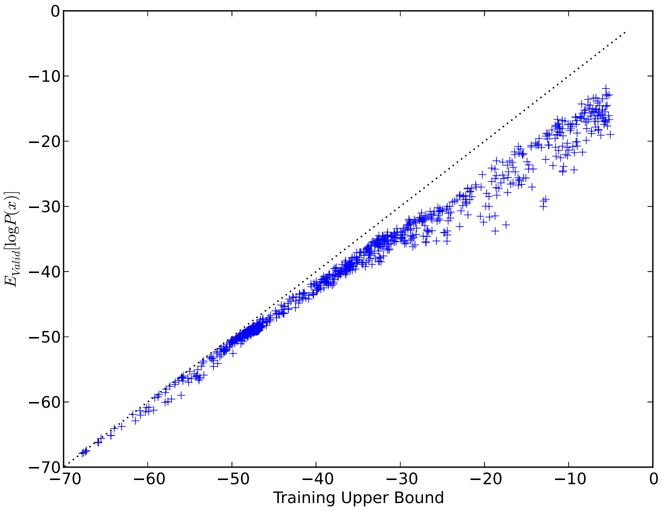


Figure 18: Training BLM upper bound vs validation log-likelihood on the Tea training dataset

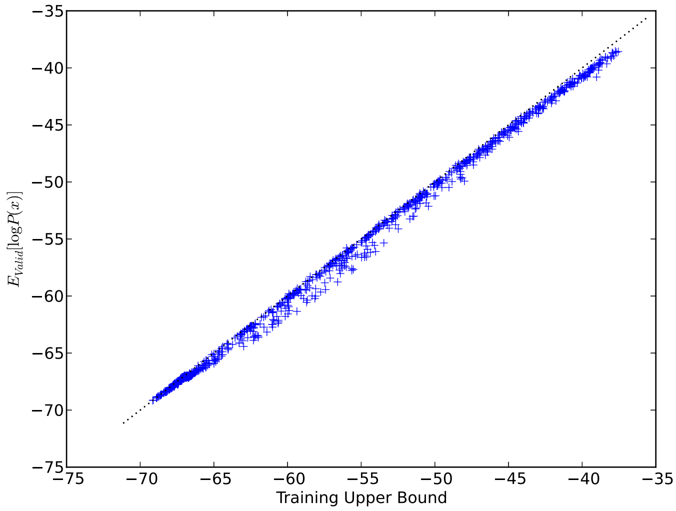


Figure 19: Training BLM upper bound vs validation log-likelihood on the Cmnist training dataset

difficult: the optimal $P(\mathbf{h})$ for the training set used by the BLM is in fact almost optimal for the validation set too. On the Tea dataset, the picture is somewhat different: there is a gap between the training upper-bound and the validation log-likelihood. This can be attributed to the increased importance of regularization on this dataset in which the training set contains only 81 samples.

Although the training BLM upper bound can therefore not be considered a good predictor of the validation log-likelihood, it is still a monotonous function of the validation log-likelihood: as such it can still be used for comparing parameter settings and for hyper-parameter selection.

Feeding the validation dataset to the BLM. Predictivity of the BLM (e.g., for hyper-parameter selection) can be improved by feeding the *validation* rather than training set to the inference distribution and the BLM.

In the cases above we examined the predictivity of the BLM obtained during training, on final performance on a validation dataset. We have seen that the training BLM is an imperfect predictor of this performance, notably because of lack of regularization in the BLM optimistic assumption, and because we use an inference distribution q maximized over the training set.

Some of these effects can easily be predicted by feeding the *validation* set to the BLM and the inference part of the model during hyper-parameter selection, as follows.

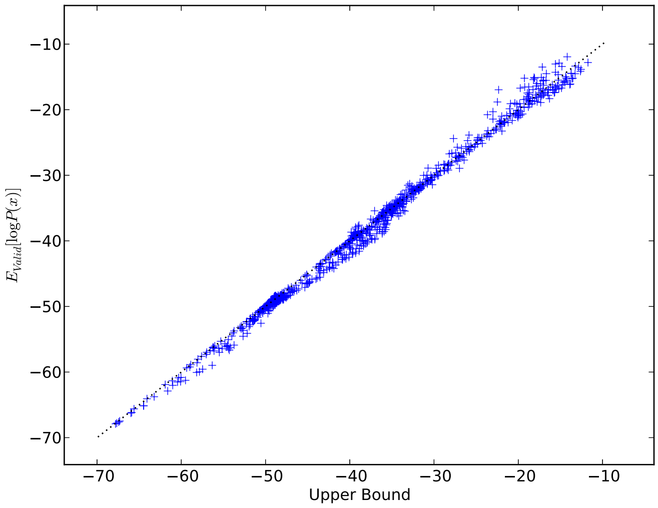


Figure 20: Validation upper bound vs log-likelihood on the Tea validation dataset

We call *validation BLM upper bound* the BLM upper bound obtained by using the validation dataset instead of $\mathcal{D}$ in (34). Note that the values q and θ_I are still those obtained from training on the training dataset. This parallels the validation step for auto-encoders, in which, of course, reconstruction performance on a validation dataset is done by feeding this same dataset to the network.

We now compare the validation BLM upper bound to the log-likelihood of the validation dataset, to see if it qualifies as a reasonable proxy.

The results are reported in Figures 20 and 21. As predicted, the validation BLM upper bound is a better estimator of the validation log-likelihood (compare Figures 18 and 19).

We can see that several models have a validation log-likelihood higher than the validation BLM upper bound, which might seem paradoxical. This is simply because the validation BLM upper bound still uses the parameters trained on the training set and thus is not formally an upper bound.

The better overall approximation of the validation log-likelihood seems to indicate that performing hyper-parameter selection with the *validation* BLM upper bound can better account for generalization and regularization.

3.3.4 Approximating the BLM for larger models

The experimental setting considered here was small enough to allow for an exact computation of the various BLM bounds by summing over all possible states of the hidden variable $\mathbf{h}$. However the exact computation of the BLM upper bound using

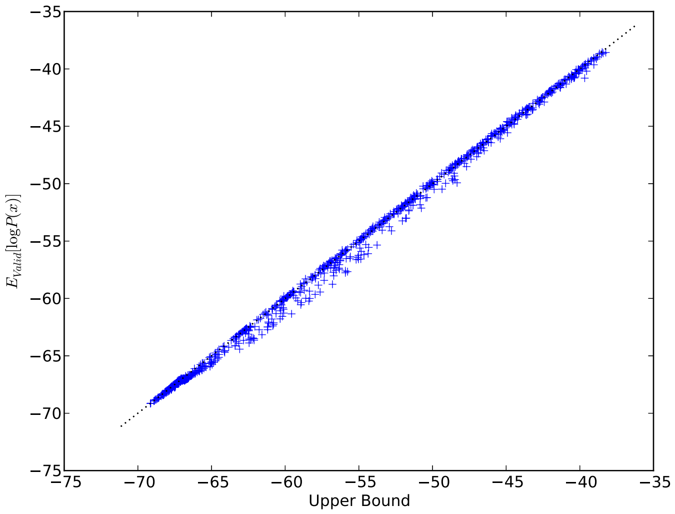


Figure 21: Validation upper bound vs log-likelihood on the Cmnist validation dataset

$\hat{\mathcal{U}}_{\mathcal{D},q}(\theta_I)$ as in (34) is not always possible because the number of terms in this sum is exponential in the dimension of the hidden layer $\mathbf{h}$.

In this situation we can use a sampling approach. For each data sample $\tilde{\mathbf{x}}$, we can take K samples from each mode of the BLM distribution $q_{\mathcal{D}}$ (one mode for each data sample $\tilde{\mathbf{x}}$) to obtain an approximation of the upper bound in $\mathcal{O}(K \times N^2)$ where N is the size of the validation set. (Since the practitioner can choose the size of the validation set which need not necessarily be as large as the training or test sets, we do not consider the N^2 factor a major hurdle.)

Definition 9 *For θ_I and q resulting from the training of a deep generative model, let*

$$\hat{\hat{\mathcal{U}}}_{\mathcal{D},q}(\theta_I) := \mathbb{E}_{\mathbf{x}\sim P_{\mathcal{D}}}\left[\log \sum_{\tilde{\mathbf{x}}} \sum_{k=1}^{K} P_{\theta_I}(\mathbf{x}|\mathbf{h}) P_{\mathcal{D}}(\tilde{\mathbf{x}})\right] \tag{35}$$

where for each $\tilde{\mathbf{x}}$ and k, $\mathbf{h}$ is sampled from $q(\mathbf{h}|\tilde{\mathbf{x}})$.

To assess the accuracy of this approximation, we take $K = 1$ and compare the values of $\hat{\hat{\mathcal{U}}}_{\mathcal{D},q}(\theta_I)$ and of $\hat{\mathcal{U}}_{\mathcal{D},q}(\theta_I)$, on the Cmnist and Tea training datasets. The results are reported in Figures 22 and 23 for all three models (vanilla AEs, AERIes, and SRBMs) superimposed, showing good agreement.

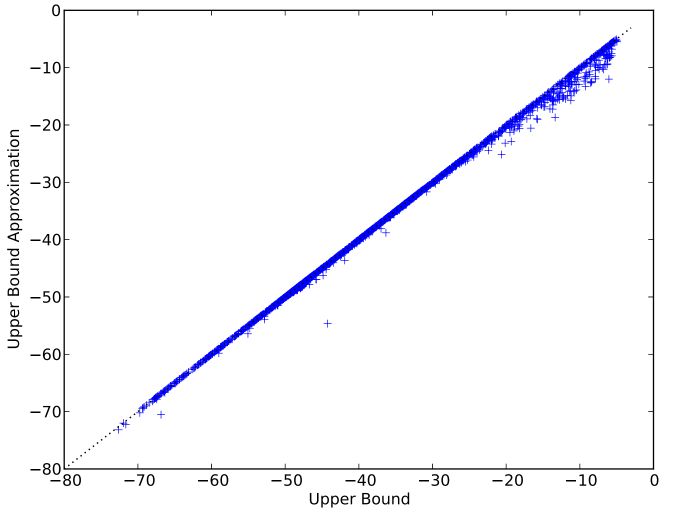


Figure 22: Approximation of the training BLM upper bound on the Tea training dataset

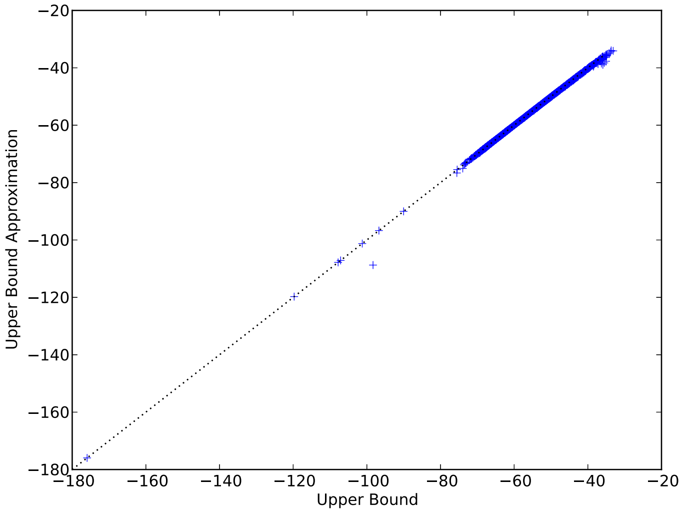


Figure 23: Approximation of the training BLM upper bound on the the Cmnist training dataset

Conclusions

The new layer-wise approach we propose to train deep generative models is based on an optimistic criterion, the BLM upper bound, in which we suppose that learning will be successful for upper layers of the model. Provided this optimism is justified a posteriori and a good enough model is found for the upper layers, the resulting deep generative model is provably close to optimal. When optimism is not justified, we provide an explicit bound on the loss of performance.

This provides a new justification for auto-encoder training and fine-tuning, as the training of the lower part of a deep generative model, optimized using a lower bound on the BLM.

This new framework for training deep generative models highlights the importance of using richer models when performing inference, contrary to current practice. This is consistent with the intuition that it is much harder to guess the underlying structure by looking at the data, than to derive the data from the hidden structure once it is known.

This possibility is tested empirically with auto-encoders with rich inference (AERIes) which are completed with a top-RBM to create deep generative models: these are then able to outperform current state of the art (stacked RBMs) on two different deep datasets.

The BLM upper bound is also found to be a good layer-wise proxy to evaluate the log-likelihood of future models for a given lower layer setting, and as such is a relevant means of hyper-parameter selection.

This opens new avenues of research, for instance in the design of algorithms to learn features in the lower part of the model, or in the possibility to consider feature extraction as a partial deep generative model in which the upper part of the model is temporarily left unspecified.

Acknowledgments

This work was partially supported by the French ANR as part of the ASAP project under grant ANR_09_EMER_001_04. The authors gratefully acknowledge the support of the PASCAL2 Network of Excellence (IST-2007-216886).

References

Y. Bengio and Y. LeCun. Scaling learning algorithms towards ai. In *Large-Scale Kernel Machines*. MIT Press, 2007.

Y. Bengio, P. Lamblin, V. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, Cambridge, MA, 2007.

Yoshua Bengio and Olivier Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6):1601–1621, 2009.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

James Bergstra, Rémy Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 23*, 2011.

H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294, 1988.

Wray L. Buntine and Andreas S. Weigend. Bayesian back-propagation. *Complex Systems*, 5:603–643, 1991.

Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, second edition, 2006.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39:1–38, 1977.

G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

G.E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

G.E. Hinton, S. Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Conputation*, 18:1527–1554, 2006.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 473–480, New York, NY, USA, 2007. ACM.

H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10: 1–40, 2009.

Nicolas Le Roux and Yoshua Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, 20:1631–1649, June 2008.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

R. M. Neal. Learning stochastic feedforward networks. Technical report, Dept. of Computer Science, University of Toronto, 1990.

Radford M. Neal. Annealed importance sampling. Technical report, University of Toronto, Department of Statistics, 1998.

Salah Rifai, Yoshua Bengio, Yann Dauphin, and Pascal Vincent. A generative process for sampling contractive auto-encoders. In *International Conference on Machine Learning, ICML'12*, 06 2012.

Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5, pages 448–455, 2009.

Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 872–879, New York, NY, USA, 2008. ACM.

P. Smolensky. Information processing in dynamical systems: foundations of harmony theory. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, MA, USA, 1986.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008.

C. F. Jeff Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11:95–103, 1983.

8.3 Discussion

This paper answers most of the questions presented in Chapter 6: How to train optimal lower layers? With what criterion? How to perform layer wise model selection? How to stop adding layers?

First, we show that it is theorerically possible to learn an optimal lower layer before learning the upper layers (Question 1), in the sense that we can maximize for a single layer the capacity of the model to achieve higher performance later. The criterion to maximize at each step (Question 2) is then the BLM upper bound[1]. As for the question of generality (Question 3), the BLM upper bound applies to any generative model which represents probabilities over the input $\mathbf{x}$ with distinct models for $p(\mathbf{x}|\mathbf{h})$ and for $p(\mathbf{h})$. In this setting, the BLM upper bound is the criterion to maximize w.r.t. the parameters of $p(\mathbf{x}|\mathbf{h})$, and the log-likelihood of the dataset transformed by an inference distribution $q_{cond}(\mathbf{h}|\mathbf{x})$ is the criterion to maximize w.r.t. the parameters of $p(\mathbf{h})$. In the case of auto-associators and unsupervised fine-tuning (Questions 4 and 5), the maximization of the reconstruction error can be seen as maximizing a reasonable lower bound of the BLM upper bound where each example corresponds to only one hidden representation.

This provides adittional justification for auto-associators, the unsupervised fine-tuning of deep auto-encoders and for sparse coding models (Kavukcuoglu et al., 2010a), as learning a partial deep generative model where the upper-part is left unspecified.

This paper also gives a possible approach to estimating the log-likelihood of deep generative models (Question 6). Although the BLM upper bound can only be used to estimate the lower part of a deep generative model reliably, in the case where the upper layer is effectively capable of learning its target distribution efficiently, the BLM upper bound gives a very good approximation of the log-likelihood of the complete model. As a good criterion for evaluating lower layers, the BLM upper bound can also be used to perform model selection in a layer-wise fashion (Question 7). The evaluation of the last layer can then be approached by evaluating the likelihood w.r.t. its target distribution which can be reliably approximated because it only concerns one hidden layer. The BLM upper bound also gives a criterion to stop adding layers (Question 8). Namely, if the BLM upper bound is not higher than the performance achieved with a shallow model, this shallow model can be chosen as the last layer with a guarantee that adding layers could not have improved performance.

This study also leads to many questions. First, the empirical evaluation of the approach poses a serious problem because the aim is to maximize (even approximately) an intractable quantity: the log likelihood of a deep generative model. This makes difficult the comparison of two competing approaches when

[1] Although the BLM upper bound is optimal in a theoretical sense, in practice it is intractable and must be approximated.

the performance measure cannot itself be easily approximated. In this paper, the solution proposed is to evaluate the log-likelihood under small models, and accordingly to train them on problems of limited dimension, while trying to make sure that the evaluation is still meaningful. This work would therefore benefit from a more comprehensive empirical study involving more complex datasets and based either on 1/ new ways to approximate the log-likelihood of deep generative models (as in e.g. Murray and Salakhutdinov, 2009); or 2/ evaluation w.r.t. a proxy measure of performance such as classification accuracy in several settings. An other unresolved issue concerns the absence of guarantee that the problem is simplified at each layer. In other words, when a problem is transfered to the upper layers, the problem could in theory be as difficult as the original problem or even harder. Fortunately this does not seem to be the case in practice. Nonetheless, this question is closely related to the matter of what constitutes a good hidden representation ? A representation which simplifies the problem would be desirable, but further study would be needed to better understand this issue. Finally, although we discuss which criterion to use in the optimization of each layer, the maximization is performed using the usual back-propagation algorithm which corresponds to stochastic gradient descent. In the next paper, we consider the possibility of using the natural gradient to learn the parameters of RBMs, hopefully improving the quality of the estimation.

9

PRESENTATION OF THE THIRD ARTICLE

Ludovic Arnold, Anne Auger, Nikolaus Hansen, and Yann Ollivier. Information-geometric optimization algorithms: A unifying picture via invariance principles. Technical report (*partial*), ArXiv e-prints, June 2011. URL http://arxiv.org/abs/1106.3708.

9.1 Context

The previous papers discussed the deep learning procedure itself, and more precisely dealt with the issue of solving a deep learning problem by sequentially solving simpler sub-problems.

However, the learning of one layer itself might be improved by a better optimization procedure. In several contexts such as deep feed-forward neural networks, or recurrent neural networks, the usual gradient descent procedure does not seem to lead to a satisfactory estimate of the optimum. This makes important the research for better optimization techniques such as second order methods (Martens, 2010; Martens and Sutskever, 2011; Sutskever et al., 2011), the natural gradient (Amari, 1998) and their possible combination (Le Roux and Fitzgibbon, 2010).

The natural gradient approach is arguably the best metric in which to perform gradient descent for maximizing the log-likelihood of a distribution because it is invariant w.r.t. re-parametrization and can also lead to more invariance with respect to transformations of the input. Nevertheless, it is seldom used in practice because it involves computing the Fisher matrix (see section 3.9), at each step. Even if computing the exact natural gradient will often be impractical, approximations might be fast enough to be applicable in practice while still leading to a better optimum than what is usually obtained with the vanilla gradient descent.

This leads us to ask what exactly happens when the vanilla gradient is used instead of the natural gradient and how this relates to parametrization and metrics.

As an exact implementation of the natural gradient is not expected to be competitive in an ML setting, we consider the setting of black-box optimization with EDAs (see section 1.8), where maximum likelihood gradient updates are used to move a proposal distribution towards better values of the objective function. In this setting, the computational cost is evaluated w.r.t. the number of evaluations of the target function which does not take into account the computation of the Fisher matrix used in the natural gradient.

9.2 Contributions

In this paper, the natural gradient is examined in the context of black-box optimization and EDAs. Experiments are performed to compare the vanilla and natural gradients when trying to optimize a simple bi-modal function using an RBM as proposal distribution.

The results clearly show the benefits of using the natural gradient in the context of EDAs. Even using the equivalent of a batch gradient update with 10,000 training samples at each step, the vanilla gradient suffers from a breach of symmetry, favoring some configurations at the expense of others. This breach of symmetry results in a premature loss of diversity as the gradient is incapable of moving the proposal RBM towards two modes at the same time. By comparison, the natural gradient restores symmetry and is consistently able to preserve diversity to account for the multimodal nature of the input. In the case of stochastic gradient updates where only 10 training samples are used at each step, the natural gradient is still able to move towards both modes of the objective function provided the learning rate is small enough.

These results suggest that the natural gradient may be more capable of dealing with multimodal distributions. In practical applications where the natural gradient may be considered too expensive, a possible way to mitigate the issues associated with the vanilla gradient may be to use a parametrization as symmetric as possible. This leads us to propose a centered energy function for RBMs.

The author is the main contributor of section 5 which concerns the experiments on the natural gradient with RBMs. Accordingly, sections 3 and 4, which do not directly concern the author's contribution, have been removed. The full paper can be found at http://arxiv.org/abs/1106.3708.

Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles

Yann Ollivier, Ludovic Arnold, Anne Auger, Nikolaus Hansen

Abstract

We present a canonical way to turn any smooth parametric family of probability distributions on an arbitrary search space X into a continuous-time black-box optimization method on X, the *information-geometric optimization* (IGO) method. Invariance as a major design principle keeps the number of arbitrary choices to a minimum. The resulting *IGO flow* is the flow of an ordinary differential equation conducting the natural gradient ascent of an adaptive, time-dependent transformation of the objective function. It makes no particular assumptions on the objective function to be optimized.

The IGO method produces explicit IGO algorithms through time discretization. It naturally recovers versions of known algorithms and offers a systematic way to derive new ones. In continuous search spaces, IGO algorithms take a form related to natural evolution strategies (NES). The cross-entropy method is recovered in a particular case with a large time step, and can be extended into a smoothed, parametrization-independent maximum likelihood update. When applied to the family of Gaussian distributions on $\mathbb{R}^d$, the IGO framework recovers a version of the well-known CMA-ES algorithm and of xNES. For the family of Bernoulli distributions on $\{0,1\}^d$, we recover the seminal PBIL algorithm. For the distributions of restricted Boltzmann machines, we naturally obtain a novel algorithm for discrete optimization on $\{0,1\}^d$. All these algorithms are natural instances of, and unified under, the single information-geometric optimization framework.

The IGO method achieves, thanks to its intrinsic formulation, maximal invariance properties: invariance under reparametrization of the search space X, under a change of parameters of the probability distribution, and under increasing transformation of the function to be optimized. The latter is achieved through an adaptive formulation of the objective.

Theoretical considerations strongly suggest that IGO algorithms are essentially characterized by a minimal change of the distribution over time. Therefore they have minimal loss in diversity through the course of optimization, provided the initial diversity is high. First experiments using restricted Boltzmann machines confirm this insight. As a simple consequence, IGO seems to provide, from information theory, an elegant way to spontaneously explore several valleys of a fitness landscape in a single run.

Introduction

Optimization problems are at the core of many disciplines. Given an objective function $f : X \to \mathbb{R}$, to be optimized on some space X, the goal of black-box optimization is to find solutions $x \in X$ with small (in the case of minimization) value $f(x)$, using the least number of calls to the function f. In a *black-box* scenario, knowledge about the function f is restricted to the handling of a device (e.g., a simulation code) that delivers the value $f(x)$ for any input $x \in X$. The search space X may be finite, discrete infinite, or continuous. However, optimization algorithms are often designed for a specific type of search space, exploiting its specific structure.

One major design principle in general and in optimization in particular is related to *invariance*, which allows to extend performance observed on a given function to its whole associated invariance class. Thus invariance hopefully provides better robustness w.r.t. changes in the presentation of a problem. For continuous search spaces, invariance under translation of the coordinate system is standard in optimization. Invariance under general affine-linear changes of the coordinates has been—we believe—one of the keys to the success of the *covariance matrix adaptation evolution strategy* (CMA-ES, (Hansen and Ostermeier, 2001)). While these relate to transformations in the search space, another important invariance concerns the application of monotonically increasing transformations to f, so that it is indifferent whether the function f, f^3 or $f \times |f|^{-2/3}$ is minimized. This way some non-convex or non-smooth functions can be as "easily" optimised as convex ones. Invariance under f-transformation is not uncommon, e.g., for evolution strategies (Schwefel, 1995) or pattern search methods (Hooke and Jeeves, 1961; Torczon, 1997; Nelder and Mead, 1965); however it has not always been recognized as an attractive feature.

Many stochastic optimization methods have been proposed to tackle black-box optimization. The underlying (often hidden) principle of those stochastic methods is to iteratively update a probability distribution P_θ defined on X, parametrized by a set of parameters θ. At a given iteration, the distribution P_θ represents, loosely speaking, the current belief about where solutions with the smallest values of the function f may lie. Over time, P_θ is expected to concentrate around the minima of f. The update of the distribution involves querying the function with points sampled from the current probability distribution P_θ. Although implicit in the presentation of many stochastic optimization algorithms, this is the natural setting for the wide family of *estimation of distribution algorithms* (EDA) (Larranaga and Lozano, 2002; Baluja and Caruana, 1995; Pelikan et al., 2002). Updates of the probability distribution often rely on heuristics (nevertheless in (Toussaint, 2004) the possible interest of information geometry to exploit the structure of probability distributions for designing better grounded heuristics is pointed out). In addition, in the EDA setting we can distinguish two theoretically founded approaches to update P_θ. First, the *cross-entropy* method consists in taking θ minimizing the Kullback–Leibler divergence between P_θ and the indicator of the best points according to f (de Boer et al., 2005). Second, one can transfer the objective function f to the space of parameters θ by taking the average of f under P_θ, seen as a function of θ. This average is a new function from a Euclidian space to $\mathbb{R}$ and is minimal when P_θ is concentrated on minima of f. Consequently, θ can be updated by following a gradient descent of this function with respect to θ. This has been done in various situations such as $X = \{0,1\}^d$ and the family of Bernoulli measures (Berny, 2000a) or of Boltzmann machines (Berny, 2002), or on $X = \mathbb{R}^d$ for the family of Gaussian distributions (Berny, 2000b; Gallagher and Frean, 2005).

However, taking the ordinary gradient with respect to θ depends on the precise way a parameter θ is chosen to represent the distribution P_θ, and does not take advantage of the Riemannian metric structure of families of probability distributions. In the context of machine learning, Amari noted the shortcomings of the ordinary gradient for families of probability distributions (Amari, 1998) and proposed instead to use the natural gradient with respect to the Fisher metric (Rao, 1945; Jeffreys, 1946; Amari and Nagaoka, 2000). In the context of optimization, the natural gradient with respect to the Fisher metric has been used for exponential families on $X = \{0,1\}^d$ (Malagò et al., 2008, 2011) and for the family of Gaussian distributions on $X = \mathbb{R}^d$ with so-called natural evolution strategies (NES) (Wierstra et al., 2008; Sun et al., 2009; Glasmachers et al., 2010).

However, none of the previous attempts using gradient updates captures the invariance under increasing transformations of the objective function, which is instead, in some cases, enforced *a posteriori* with heuristics arguments.

Building on these ideas, this paper overcomes the invariance problem of previous attempts and provides a consistent, unified picture of optimization on arbitrary search spaces via invariance principles. More specifically, we consider an arbitrary search space X, either discrete or continuous, and a black-box optimization problem on X. We assume that a family of probability distributions P_θ on X depending on a continuous multicomponent parameter $\theta \in \Theta$ has been chosen. A classical example is to take $X = \mathbb{R}^d$ and to consider the family of all Gaussian distributions P_θ on $\mathbb{R}^d$, with $\theta = (m, C)$ the mean and covariance matrix. Another simple example is $X = \{0,1\}^d$ equipped with the family of Bernoulli measures, i.e. $\theta = (\theta_i)_{1 \leqslant i \leqslant d}$ and $P_\theta(x) = \prod \theta_i^{x_i}(1-\theta_i)^{1-x_i}$ for $x = (x_i) \in X$.

From this setting, *information-geometric optimization* (IGO) can be defined in a natural way. At each (continuous) time t, we maintain a value θ^t of the parameter of the distribution. The function f to be optimized is transferred to the parameter space Θ by means of a suitable time-dependent transformation based on the *P_{θ^t}-quantiles* of f (Definition 2). The *IGO flow*, introduced in Definition 3, follows the natural gradient of the expected value of this function of θ^t in the parameter space Θ, where the natural gradient derives from the Fisher information metric. The IGO flow is thus the flow of an ordinary differential equation in space Θ. This continuous-time gradient flow is turned into a family of explicit *IGO algorithms* by taking an Euler time discretization of the differential equation and approximating the distribution P_{θ^t} by using samples. From the start, the IGO flow is invariant under strictly increasing transformations of f; we also prove that the sampling procedure is consistent. IGO algorithms share their final algebraic form with the *natural evolution strategies* (NES) introduced in the Gaussian setting (Wierstra et al., 2008; Sun et al., 2009; Glasmachers et al., 2010); the latter are thus recovered in the IGO framework as an Euler approximation to a well-defined flow, without heuristic arguments.

The IGO method also has an equivalent description as an *infinitesimal max-*

imum likelihood update; this reveals a new property of the natural gradient and does not require a smooth parametrization by θ anymore. This also establishes a link between IGO and the *cross-entropy method* (de Boer et al., 2005).

When we instantiate IGO using the family of Gaussian distributions on $\mathbb{R}^d$, we naturally obtain versions of the well-known *covariance matrix adaptation evolution strategy* (CMA-ES) (Hansen and Ostermeier, 2001; Hansen and Kern, 2004; Jastrebski and Arnold, 2006) and of *natural evolution strategies*. With Bernoulli measures on the discrete cube $\{0,1\}^d$, we recover the well-known *population-based incremental learning* (PBIL) (Baluja and Caruana, 1995; Baluja, 1994); this derivation of PBIL as a natural gradient ascent appears to be new, and sheds some light on the common ground between continuous and discrete optimization.

From the IGO framework, it is immediate (theoretically) to build new optimization algorithms using more complex families of distributions than Gaussian or Bernoulli. As an illustration, distributions associated with restricted Boltzmann machines (RBMs) provide a new but natural algorithm for discrete optimization on $\{0,1\}^d$ which is able to handle dependencies between the bits (see also (Berny, 2002)). The probability distributions associated with RBMs are multimodal; combined with the specific information-theoretic properties of IGO that guarantee minimal change in diversity over time, this allows IGO to reach multiple optima at once very naturally, at least in a simple experimental setup (Section 5).

The IGO framework is built to achieve maximal *invariance properties*. Invariance in the search space is related to invariance under θ-reparametrization which is the main idea behind *information geometry* (Amari and Nagaoka, 2000). First, the IGO flow is invariant under reparametrization of the family of distributions P_θ, that is, it only depends on P_θ and not on the way we write the parameter θ. For instance, for Gaussian measures it should not matter whether we use the covariance matrix or its inverse or a Cholesky factor as the parameter. This limits the influence of encoding choices on the behavior of the algorithm. Second, the IGO flow is invariant under a change of coordinates in the search space X, provided that this change of coordinates globally preserves the family of distributions P_θ. For instance, for Gaussian distributions on $\mathbb{R}^d$, this includes all affine changes of coordinates. This means that the algorithm, apart from initialization, does not depend on the precise way the data is presented. Last, IGO algorithms are invariant under applying a strictly increasing function to f. Contrary to previous formulations using natural gradients (Wierstra et al., 2008; Glasmachers et al., 2010; Akimoto et al., 2010), this invariance is achieved from the start. Such invariance properties mean that we deal with *intrinsic* properties of the objects themselves, and not with the way we encode them as collections of numbers in $\mathbb{R}^d$. It also means, most importantly, that we make a minimal number of arbitrary choices.

In Section 1, we define the IGO flow and the IGO algorithm. We begin with standard facts about the definition and basic properties of the natural gradient, and its connection with Kullback–Leibler divergence and diversity. We then proceed to the detailed description of the algorithm.

In Section 2, we state some first mathematical properties of IGO. These include monotone improvement of the objective function, invariance properties, the form of IGO for exponential families of probability distributions, and the case of noisy objective functions.

In Section 3 we explain the theoretical relationships between IGO, maximum likelihood estimates and the cross-entropy method. In particular, IGO is uniquely characterized by a weighted log-likelihood maximization property.

In Section 4, we derive several well-known optimization algorithms from the IGO framework. These include PBIL, versions of CMA-ES and other Gaussian evolutionary algorithms such as EMNA and xNES. This also illustrates how a large step size results in more and more differing algorithms w.r.t. the continuous-time IGO flow. We also study the IGO flow solution on linear functions for discrete and continuous search spaces.

In Section 5, we illustrate how IGO can be used to design new optimization algorithms. As a proof of concept, we derive the IGO algorithm associated with restricted Boltzmann machines for discrete optimization, allowing for multimodal optimization. We perform a preliminary experimental study of the specific influence of the Fisher information matrix on the performance of the algorithm and on diversity of the optima obtained.

In Section 6, we discuss related work, and in particular, IGO's relationship with and differences from various other optimization algorithms such as natural evolution strategies or the cross-entropy method. We also sum up the main contributions of the paper and the design philosophy of IGO.

1 Algorithm description

We now present the outline of the algorithm. Each step is described in more detail in the sections below.

The IGO flow can be seen as an *estimation of distribution algorithm*: at each time t, we maintain a probability distribution P_{θ^t} on the search space X, where $\theta^t \in \Theta$. The value of θ^t will evolve so that, over time, P_{θ^t} gives more weight to points x with better values of the function $f(x)$ to optimize.

A straightforward way to proceed is to transfer f from x-space to θ-space: define a function $F(\theta)$ as the P_θ-average of f and then do a gradient descent for $F(\theta)$ in space Θ (Berny, 2000a, 2002, 2000b; Gallagher and Frean, 2005). This way, θ will converge to a point such that P_θ yields a good average value of f. We depart from this approach in two ways:

- At each time, we replace f with an adaptive transformation of f representing how good or bad observed values of f are relative to other observations. This provides invariance under all monotone transformations of f.

- Instead of the vanilla gradient for θ, we use the so-called *natural gradient* given by the Fisher information matrix. This reflects the intrinsic geometry of the space of probability distributions, as introduced by Rao and

Jeffreys (Rao, 1945; Jeffreys, 1946) and later elaborated upon by Amari and others (Amari and Nagaoka, 2000). This provides invariance under reparametrization of θ and, importantly, minimizes the change of diversity of P_θ.

The algorithm is constructed in two steps: we first give an "ideal" version, namely, a version in which time t is continuous so that the evolution of θ^t is given by an ordinary differential equation in Θ. Second, the actual algorithm is a time discretization using a finite time step and Monte Carlo sampling instead of exact P_θ-averages.

1.1 The natural gradient on parameter space

About gradients and the shortest path uphill. Let g be a smooth function from $\mathbb{R}^d$ to $\mathbb{R}$, to be maximized. We first recall the interpretation of gradient ascent as "the shortest path uphill".

Let $y \in \mathbb{R}^d$. Define the vector z by

$$z = \lim_{\varepsilon \to 0} \operatorname*{arg\,max}_{z,\, \|z\| \leqslant 1} g(y + \varepsilon z). \tag{1}$$

Then one can check that z is the normalized gradient of g at y: $z_i = \frac{\partial g / \partial y_i}{\|\partial g / \partial y_k\|}$. (This holds only at points y where the gradient of g does not vanish.)

This shows that, for small δt, the well-known gradient ascent of g given by

$$y_i^{t+\delta t} = y_i^t + \delta t \, \tfrac{\partial g}{\partial y_i}$$

realizes the largest increase of the value of g, for a given step size $\|y^{t+\delta t} - y^t\|$.

The relation (1) depends on the choice of a norm $\|\cdot\|$ (the gradient of g is given by $\partial g / \partial y_i$ only in an orthonormal basis). If we use, instead of the standard metric $\|y - y'\| = \sqrt{\sum (y_i - y'_i)^2}$ on $\mathbb{R}^d$, a metric $\|y - y'\|_A = \sqrt{\sum A_{ij}(y_i - y'_i)(y_j - y'_j)}$ defined by a positive definite matrix A_{ij}, then the gradient of g with respect to this metric is given by $\sum_j A^{-1}_{ij} \frac{\partial g}{\partial y_i}$. This follows from the textbook definition of gradients by $g(y + \varepsilon z) = g(y) + \varepsilon \langle \nabla g, z \rangle_A + O(\varepsilon^2)$ with $\langle \cdot, \cdot \rangle_A$ the scalar product associated with the matrix A_{ij} (Schwartz, 1992).

It is possible to write the analogue of (1) using the A-norm. We then find that the gradient ascent associated with metric A is given by

$$y^{t+\delta t} = y^t + \delta t \, A^{-1} \tfrac{\partial g}{\partial y_i},$$

for small δt and maximizes the increment of g for a given A-distance $\|y^{t+\delta t} - y^t\|_A$—it realizes the steepest A-ascent. Maybe this viewpoint clarifies the relationship between gradient and metric: this steepest ascent property can actually be used as a definition of gradients.

In our setting we want to use a gradient ascent in the parameter space Θ of our distributions P_θ. The "vanilla" gradient $\frac{\partial}{\partial \theta_i}$ is associated with the metric $\|\theta - \theta'\| = \sqrt{\sum (\theta_i - \theta'_i)^2}$ and clearly depends on the choice of parametrization θ. Thus this metric, and the direction pointed by this gradient, are not intrinsic, in the sense that they do not depend only on the *distribution* P_θ. A metric depending on θ only through the distributions P_θ can be defined as follows.

Fisher information and the natural gradient on parameter space. Let $\theta, \theta' \in \Theta$ be two values of the distribution parameter. The most widely used way to define a "distance" between two generic distributions P_θ and $P_{\theta'}$ is the *Kullback–Leibler divergence* from information theory, defined (Kullback, 1997) as

$$\mathrm{KL}(P_{\theta'} \,||\, P_\theta) = \int_x \ln \frac{P_{\theta'}(x)}{P_\theta(x)} \, P_{\theta'}(\mathrm{d}x).$$

When $\theta' = \theta + \delta\theta$ is close to θ, under mild smoothness assumptions we can expand the Kullback–Leibler divergence at second order in $\delta\theta$. This expansion defines the Fisher information matrix I at θ (Kullback, 1997):

$$\mathrm{KL}(P_{\theta+\delta\theta} \,||\, P_\theta) = \frac{1}{2} \sum I_{ij}(\theta)\, \delta\theta_i \delta\theta_j + O(\delta\theta^3).$$

An equivalent definition of the Fisher information matrix is by the usual formulas (Cover and Thomas, 2006)

$$I_{ij}(\theta) = \int_x \frac{\partial \ln P_\theta(x)}{\partial \theta_i} \frac{\partial \ln P_\theta(x)}{\partial \theta_j} \, P_\theta(\mathrm{d}x) = -\int_x \frac{\partial^2 \ln P_\theta(x)}{\partial \theta_i \, \partial \theta_j} \, P_\theta(\mathrm{d}x).$$

The Fisher information matrix defines a (Riemannian) metric on Θ: the distance, in this metric, between two very close values of θ is given by the square root of twice the Kullback–Leibler divergence. Since the Kullback–Leibler divergence depends only on P_θ and not on the parametrization of θ, this metric is intrinsic.

If $g : \Theta \to \mathbb{R}$ is a smooth function on the parameter space, its *natural gradient* (Amari, 1998) at θ is defined in accordance with the Fisher metric as

$$(\widetilde{\nabla}_\theta \, g)_i = \sum_j I^{-1}_{ij}(\theta) \, \frac{\partial g(\theta)}{\partial \theta_j}$$

or more synthetically

$$\widetilde{\nabla}_\theta \, g = I^{-1} \, \frac{\partial g}{\partial \theta}.$$

From now on, we will use $\widetilde{\nabla}_\theta$ to denote the natural gradient and $\frac{\partial}{\partial \theta}$ to denote the vanilla gradient.

By construction, the natural gradient descent is intrinsic: it does not depend on the chosen parametrization θ of P_θ, so that it makes sense to speak of the natural gradient ascent of a function $g(P_\theta)$. The Fisher metric is essentially the only way to obtain this property (Amari and Nagaoka, 2000, Section 2.4).

Given that the Fisher metric comes from the Kullback–Leibler divergence, the "shortest path uphill" property of gradients mentioned above translates as follows (see also (Amari, 1998, Theorem 1)):

Proposition 1. *The natural gradient ascent points in the direction $\delta\theta$ achieving the largest change of the objective function, for a given distance between P_θ and $P_{\theta+\delta\theta}$ in Kullback–Leibler divergence. More precisely, let g be a smooth function* *on the parameter space Θ. Let $\theta \in \Theta$ be a point where $\widetilde{\nabla} g(\theta)$ does not vanish. Then, if*

$$\delta\theta = \frac{\widetilde{\nabla} g(\theta)}{\|\widetilde{\nabla} g(\theta)\|}$$

is the direction of the natural gradient of g, we have

$$\delta\theta = \lim_{\varepsilon \to 0} \frac{1}{\varepsilon} \underset{\substack{\delta\theta \text{ such that} \\ \mathrm{KL}(P_{\theta+\delta\theta} \,\|\, P_\theta) \leqslant \varepsilon^2/2}}{\arg\max} g(\theta + \delta\theta).$$

Here we have implicitly assumed that the parameter space Θ is non-degenerate and proper (that is, no two points $\theta \in \Theta$ define the same probability distribution, and the mapping $P_\theta \mapsto \theta$ is continuous).

Why use the Fisher metric gradient for optimization? Relationship to diversity. The first reason for using the natural gradient is its reparametrization invariance, which makes it the only gradient available in a general abstract setting (Amari and Nagaoka, 2000). Practically, this invariance also limits the influence of encoding choices on the behavior of the algorithm. More prosaically, the Fisher matrix can be also seen as an *adaptive learning rate* for different components of the parameter vector θ_i: components i with a high impact on P_θ will be updated more cautiously.

Another advantage comes from the relationship with Kullback–Leibler distance in view of the "shortest path uphill" (see also (Amari, 1998)). To minimize the value of some function $g(\theta)$ defined on the parameter space Θ, the naive approach follows a gradient descent for g using the "vanilla" gradient

$$\theta_i^{t+\delta t} = \theta_i^t + \delta t \tfrac{\partial g}{\partial \theta_i}$$

and, as explained above, this maximizes the increment of g for a given increment $\|\theta^{t+\delta t} - \theta^t\|$. On the other hand, the Fisher gradient

$$\theta_i^{t+\delta t} = \theta_i^t + \delta t I^{-1} \tfrac{\partial g}{\partial \theta_i}$$

maximizes the increment of g for a given Kullback–Leibler distance $\mathrm{KL}(P_{\theta^{t+\delta t}} \,\|\, P_{\theta^t})$.

In particular, if we choose an initial value θ^0 such that P_{θ^0} covers the whole space X uniformly (or a wide portion, in case X is unbounded), the Kullback–Leibler divergence between P_{θ^t} and P_{θ^0} is the Shannon entropy of the uniform distribution minus the Shannon entropy of P_{θ^t}, and so this divergence measures the loss of diversity of P_{θ^t} with respect to the uniform distribution. So following the natural gradient of a function g, starting at or close to the uniform distribution, amounts to optimizing the function g while staying as close as possible to uniform in Kullback–Leibler divergence, i.e., optimizing the function g with *minimal loss of diversity, provided the initial diversity is large.* (This is valid, of course, only at the beginning; once one gets too far from uniform, a better interpretation is minimal change of diversity.) On the other hand, the vanilla gradient descent optimizes g with minimal change in the numerical values of the parameter θ, which is of little interest.

So arguably this method realizes the best trade-off between optimization and loss of diversity. (Though, as can be seen from the detailed algorithm description below, maximization of diversity occurs only greedily at each step, and so there is no guarantee that after a given time, IGO will provide the highest possible diversity for a given objective function value.)

An experimental confirmation of the positive influence of the Fisher matrix on diversity is given in Section 5 below. This may also provide a theoretical explanation to the good performance of CMA-ES.

1.2 IGO: Information-geometric optimization

Quantile rewriting of f. Our original problem is to minimize a function $f : X \to \mathbb{R}$. A simple way to turn f into a function on Θ is to use the expected value $-\mathbb{E}_{P_\theta} f$ (Berny, 2000a; Wierstra et al., 2008), but expected values can be unduly influenced by extreme values and using them can be rather unstable (Whitley, 1989); moreover $-\mathbb{E}_{P_\theta} f$ is not invariant under increasing transformation of f (this invariance implies we can only compare f-values, not sum them up).

Instead, we take an adaptive, quantile-based approach by first replacing the function f with a monotone rewriting $W^f_{\theta^t}$, depending on the current parameter value θ^t, and then following the gradient of $\mathbb{E}_{P_\theta} W^f_{\theta^t}$, seen as a function of θ. A due choice of $W^f_{\theta^t}$ allows to control the range of the resulting values and achieves the desired invariance. Because the rewriting $W^f_{\theta^t}$ depends on θ^t, it might be viewed as an *adaptive* f-transformation.

The goal is that if $f(x)$ is "small" then $W^f_\theta(x) \in \mathbb{R}$ is "large" and vice versa, and that W^f_θ remains invariant under increasing transformations of f. The meaning of "small" or "large" depends on $\theta \in \Theta$ and is taken with respect to typical values of f under the current distribution P_θ. This is measured by the P_θ-quantile in which the value of $f(x)$ lies.

Definition 2. *The lower and upper P_θ-f-quantiles of $x \in X$ are defined as*

$$\begin{aligned} q^<_\theta(x) &= \Pr_{x' \sim P_\theta}(f(x') < f(x)) \\ q^\leqslant_\theta(x) &= \Pr_{x' \sim P_\theta}(f(x') \leqslant f(x)) \ . \end{aligned} \tag{2}$$

Let $w : [0;1] \to \mathbb{R}$ be a non-increasing function, the selection scheme*.*

The transform $W^f_\theta(x)$ of an objective function $f : X \to \mathbb{R}$ is defined as a function of the P_θ-f-quantile of x as

$$W^f_\theta(x) = \begin{cases} w(q^\leqslant_\theta(x)) & \text{if } q^\leqslant_\theta(x) = q^<_\theta(x), \\ \frac{1}{q^\leqslant_\theta(x) - q^<_\theta(x)} \int_{q=q^<_\theta(x)}^{q=q^\leqslant_\theta(x)} w(q)\,\mathrm{d}q & \text{otherwise.} \end{cases} \tag{3}$$

The quantile functions q reflect the probability to sample a better value than $f(x)$. They are monotone in f (if $f(x_1) \leqslant f(x_2)$ then $q^<_\theta(x_1) \leqslant q^<_\theta(x_2)$, and likewise for $q^\leqslant$) and invariant under strictly increasing transformations of f.

A typical choice for w is $w(q) = \mathbb{1}_{q \leqslant q_0}$ for some fixed value q_0, the *selection quantile*. In what follows, we suppose that a selection scheme has been chosen once and for all.

As desired, the definition of W_θ^f is invariant under a strictly increasing transformation of f. For instance, the P_θ-median of f gets remapped to $w(\frac{1}{2})$.

Note that $\mathbb{E}_{x\sim P_\theta} W_\theta^f(x)$ is always equal to $\int_0^1 w$, independently of f and θ: indeed, by definition, the P_θ-quantile of a random point under P_θ is uniformly distributed in $[0;1]$. In the following, our objective will be to maximize the expected value of $W_{\theta^t}^f$ over θ, that is, to maximize

$$\mathbb{E}_{P_\theta} W_{\theta^t}^f = \int W_{\theta^t}^f(x)\, P_\theta(\mathrm{d}x) \tag{4}$$

over θ, where θ^t is fixed at a given step but will adapt over time.

Importantly, $W_\theta^f(x)$ can be estimated in practice: indeed, the quantiles $\Pr_{x'\sim P_\theta}(f(x') < f(x))$ can be estimated by taking samples of P_θ and ordering the samples according to the value of f (see below). The estimate remains invariant under strictly increasing f-transformations.

The IGO gradient flow. At the most abstract level, IGO is a continuous-time gradient flow in the parameter space Θ, which we define now. In practice, discrete time steps (a.k.a. iterations) are used, and P_θ-integrals are approximated through sampling, as described in the next section.

Let θ^t be the current value of the parameter at time t, and let $\delta t \ll 1$. We define $\theta^{t+\delta t}$ in such a way as to increase the P_θ-weight of points where f is small, while not going too far from P_{θ^t} in Kullback–Leibler divergence. We use the adaptive weights $W_{\theta^t}^f$ as a way to measure which points have large or small values. In accordance with (4), this suggests taking the gradient ascent

$$\theta^{t+\delta t} = \theta^t + \delta t\, \widetilde{\nabla}_\theta \int W_{\theta^t}^f(x)\, P_\theta(\mathrm{d}x) \tag{5}$$

where the natural gradient is suggested by Proposition 1.

Note again that we use $W_{\theta^t}^f$ and not W_θ^f in the integral. So the gradient $\widetilde{\nabla}_\theta$ does not act on the adaptive objective $W_{\theta^t}^f$. If we used W_θ^f instead, we would face a paradox: right after a move, previously good points do not seem so good any more since the distribution has improved. More precisely, $\int W_\theta^f(x)\, P_\theta(\mathrm{d}x)$ is constant and always equal to the average weight $\int_0^1 w$, and so the gradient would always vanish.

Using the log-likelihood trick $\widetilde{\nabla} P_\theta = P_\theta\, \widetilde{\nabla} \ln P_\theta$ (assuming P_θ is smooth), we get an equivalent expression of the update above as an integral under the current distribution P_{θ^t}; this is important for practical implementation. This leads to the following definition.

Definition 3 (IGO flow). *The IGO flow is the set of continuous-time trajectories in space Θ, defined by the ordinary differential equation*

$$\frac{\mathrm{d}\theta^t}{\mathrm{d}t} = \widetilde{\nabla}_\theta \int W_{\theta^t}^f(x)\, P_\theta(\mathrm{d}x) \tag{6}$$

$$= \int W_{\theta^t}^f(x)\, \widetilde{\nabla}_\theta \ln P_\theta(x)\, P_{\theta^t}(\mathrm{d}x) \tag{7}$$

$$= I^{-1}(\theta^t) \int W_{\theta^t}^f(x)\, \frac{\partial \ln P_\theta(x)}{\partial \theta}\, P_{\theta^t}(\mathrm{d}x). \tag{8}$$

where the gradients are taken at point $\theta = \theta^t$, and I is the Fisher information matrix.

Natural evolution strategies (NES, (Wierstra et al., 2008; Glasmachers et al., 2010; Sun et al., 2009)) feature a related gradient *descent* with $f(x)$ instead of $W^f_{\theta^t}(x)$. The associated flow would read

$$\frac{\mathrm{d}\theta^t}{\mathrm{d}t} = -\widetilde{\nabla}_\theta \int f(x)\, P_\theta(\mathrm{d}x) \ , \tag{9}$$

where the gradient is taken at θ^t (in the sequel when not explicitly stated, gradients in θ are taken at $\theta = \theta^t$). However, in the end NESs always implement algorithms using sample quantiles, as if derived from the gradient ascent of $W^f_{\theta^t}(x)$.

The update (7) is a weighted average of "intrinsic moves" increasing the log-likelihood of some points. We can slightly rearrange the update as

$$\frac{\mathrm{d}\theta^t}{\mathrm{d}t} = \int \overbrace{W^f_{\theta^t}(x)}^{\text{preference weight}} \underbrace{\widetilde{\nabla}_\theta \ln P_\theta(x)}_{\text{intrinsic move to reinforce } x} \overbrace{P_{\theta^t}(\mathrm{d}x)}^{\text{current sample distribution}} \tag{10}$$

$$= \widetilde{\nabla}_\theta \int \underbrace{W^f_{\theta^t}(x) \ln P_\theta(x)}_{\text{weighted log-likelihood}} P_{\theta^t}(\mathrm{d}x). \tag{11}$$

which provides an interpretation for the IGO gradient flow as a gradient ascent optimization of the weighted log-likelihood of the "good points" of the current distribution. In a precise sense, IGO is in fact the "best" way to increase this log-likelihood.

For exponential families of probability distributions, we will see later that the IGO flow rewrites as a nice derivative-free expression.

IGO algorithms: time discretization and sampling. The above is a mathematically well-defined continuous-time flow in the parameter space. Its practical implementation involves three approximations depending on two parameters N and δt:

- the integral under P_{θ^t} is approximated using N samples taken from P_{θ^t};
- the value $W^f_{\theta^t}$ is approximated for each sample taken from P_{θ^t};
- the time derivative $\frac{\mathrm{d}\theta^t}{\mathrm{d}t}$ is approximated by a δt time increment.

We also assume that the Fisher information matrix $I(\theta)$ and $\frac{\partial \ln P_\theta(x)}{\partial \theta}$ can be computed (see discussion below if $I(\theta)$ is unknown).

At each step, we draw N samples $x_1, \ldots, x_N$ under P_{θ^t}. To approximate the quantiles, we rank the samples according to the value of f. Define $\mathrm{rk}(x_i) = \#\{j,\, f(x_j) < f(x_i)\}$ and let the estimated weight of sample x_i be

$$\widehat{w}_i = \frac{1}{N}\, w\left(\frac{\mathrm{rk}(x_i) + 1/2}{N}\right) , \tag{12}$$

using the selection scheme function w introduced above. (This is assuming there are no ties in our sample; in case several sample points have the same value of f, we define $\widehat{w}_i$ by averaging the above over all possible rankings of the ties[1].)

Then we can approximate the IGO flow as follows.

Definition 4 (IGO algorithms)**.** *The* IGO algorithm *associated with parametrization θ, sample size N and step size δt is the following update rule for the parameter θ^t. At each step, N sample points $x_1, \ldots, x_N$ are drawn according to the distribution P_{θ^t}. The parameter is updated according to*

$$\theta^{t+\delta t} = \theta^t + \delta t \sum_{i=1}^{N} \widehat{w}_i \ \widetilde{\nabla}_\theta \ln P_\theta(x_i)\Big|_{\theta=\theta^t} \tag{14}$$

$$= \theta^t + \delta t \, I^{-1}(\theta^t) \sum_{i=1}^{N} \widehat{w}_i \ \frac{\partial \ln P_\theta(x_i)}{\partial \theta}\Big|_{\theta=\theta^t} \tag{15}$$

where $\widehat{w}_i$ is the weight (12) obtained from the ranked values of the objective function f.

Equivalently one can fix the weights $w_i = \frac{1}{N} w\left(\frac{i-1/2}{N}\right)$ once and for all and rewrite the update as

$$\theta^{t+\delta t} = \theta^t + \delta t \, I^{-1}(\theta^t) \sum_{i=1}^{N} w_i \ \frac{\partial \ln P_\theta(x_{i:N})}{\partial \theta}\Big|_{\theta=\theta^t} \tag{16}$$

where $x_{i:N}$ denotes the i^{th} sampled point ranked according to f, i.e. $f(x_{1:N}) < \ldots < f(x_{N:N})$ (assuming again there are no ties). Note that $\{x_{i:N}\} = \{x_i\}$ and $\{w_i\} = \{\widehat{w}_i\}$.

As will be discussed in Section 4, this update applied to multivariate normal distributions or Bernoulli measures allows to neatly recover versions of some well-established algorithms, in particular CMA-ES and PBIL. Actually, in the Gaussian context updates of the form (15) have already been introduced (Glasmachers et al., 2010; Akimoto et al., 2010), though not formally derived from a continuous-time flow with quantiles.

When $N \to \infty$, the IGO algorithm using samples approximates the continuous-time IGO gradient flow. Indeed, the IGO algorithm, with $N = \infty$, is simply the Euler approximation scheme for the ordinary differential equation defining the IGO flow (6). The latter result thus provides a sound mathematical basis for currently used rank-based updates.

[1] A mathematically neater but less intuitive version would be

$$\widehat{w}_i = \frac{1}{\mathrm{rk}^{\leqslant}(x_i) - \mathrm{rk}^{<}(x_i)} \int_{u=\mathrm{rk}^{<}(x_i)/N}^{u=\mathrm{rk}^{\leqslant}(x_i)/N} w(u) \mathrm{d}u \tag{13}$$

with $\mathrm{rk}^{<}(x_i) = \#\{j,\, f(x_j) < f(x_i)\}$ and $\mathrm{rk}^{\leqslant}(x_i) = \#\{j,\, f(x_j) \leqslant f(x_i)\}$.

IGO flow versus IGO algorithms. The IGO *flow* (6) is a well-defined continuous-time set of trajectories in the space of probability distributions P_θ, depending only on the objective function f and the chosen family of distributions. It does not depend on the chosen parametrization for θ.

On the other hand, there are several IGO *algorithms* associated with this flow. Each IGO algorithm approximates the IGO flow in a slightly different way. An IGO algorithm depends on three further choices: a sample size N, a time discretization step size δt, and a choice of parametrization for θ in which to implement (15).

If δt is small enough, and N large enough, the influence of the parametrization θ disappears and all IGO algorithms are approximations of the "ideal" IGO flow trajectory. However, the larger δt, the poorer the approximation gets.

So for large δt, different IGO algorithms for the same IGO flow may exhibit different behaviors. We will see an instance of this phenomenon for Gaussian distributions: both CMA-ES and the maximum likelihood update (EMNA) can be seen as IGO algorithms, but the latter with $\delta t = 1$ is known to exhibit premature loss of diversity (Section 4).

Still, two IGO algorithms for the same IGO flow will differ less from each other than from a non-IGO algorithm: at each step the difference is only $O(\delta t^2)$ (Section 2). On the other hand, for instance, the difference between an IGO algorithm and the vanilla gradient ascent is, generally, not smaller than $O(\delta t)$ at each step, i.e. roughly as big as the steps themselves.

Unknown Fisher matrix. The algorithm presented so far assumes that the Fisher matrix $I(\theta)$ is known as a function of θ. This is the case for Gaussian distributions in CMA-ES and for Bernoulli distributions. However, for restricted Boltzmann machines as considered below, no analytical form is known. Yet, provided the quantity $\frac{\partial}{\partial\theta} \ln P_\theta(x)$ can be computed or approximated, it is possible to approximate the integral

$$I_{ij}(\theta) = \int_x \frac{\partial \ln P_\theta(x)}{\partial\theta_i} \frac{\partial \ln P_\theta(x)}{\partial\theta_j} \, P_\theta(\mathrm{d}x)$$

using P_θ-Monte Carlo samples for x. These samples may or may not be the same as those used in the IGO update (15): in particular, it is possible to use as many Monte Carlo samples as necessary to approximate I_{ij}, at no additional cost in terms of the number of calls to the black-box function f to optimize.

Note that each Monte Carlo sample x will contribute $\frac{\partial \ln P_\theta(x)}{\partial\theta_i} \frac{\partial \ln P_\theta(x)}{\partial\theta_j}$ to the Fisher matrix approximation. This is a rank-1 non-negative matrix[2]. So, for the approximated Fisher matrix to be invertible, the number of (distinct) samples x needs to be at least equal to, and ideally much larger than, the number of components of the parameter θ: $N_{\text{Fisher}} \geqslant \dim \Theta$.

For exponential families of distributions, the IGO update has a particular form which simplifies this matter somewhat. More details are given below (see Section 5) for the concrete situation of restricted Boltzmann machines.

[2] The alternative, equivalent formula $I_{ij}(\theta) = -\int_x \frac{\partial^2 \ln P_\theta(x)}{\partial\theta_i \, \partial\theta_j} P_\theta(\mathrm{d}x)$ for the Fisher matrix would not necessarily yield non-negative matrices through Monte Carlo sampling.

2 First properties of IGO (removed)

2.1 Consistency of sampling. 2.2 Monotonicity: quantile improvement. 2.3 The IGO flow for exponential families. 2.4 Invariance properties. 2.5 Speed of the IGO flow. 2.6 Noisy objective function. 2.7 Implementation remarks

3 IGO, maximum likelihood, and the cross-entropy method (removed)

4 CMA-ES, NES, EDAs and PBIL from the IGO framework (removed)

4.1 PBIL as IGO algorithm for Bernoulli measures. 4.2 Multivariate normal distributions (Gaussians). 4.3 Computing the IGO flow for some simple examples

5 Multimodal optimization using restricted Boltzmann machines

We now illustrate the behavior of IGO versus the vanilla gradient on an example: the probability distributions obtained from restricted Boltzmann machines for optimization on $\{0,1\}^d$. A purported advantage of such distributions for optimization (Berny, 2002) is to represent dependencies between the bits: contrary to, e.g., the Bernoulli measures, restricted Boltzmann machines can in principle handle a situation where good values of the objective function are obtained, for instance, if the first and second bit are both set to 0 or both set to 1 simultaneously. The goal of this section is threefold:

- To illustrate step by step how the IGO framework can be implemented in practice on new families of probability distributions, yielding new optimization algorithms. We discuss in particular the delicate problem of estimating the Fisher matrix.
- To illustrate the (sometimes striking) difference between the optimization trajectories obtained from the natural gradient or the vanilla gradient even in a simple situation.
- To illustrate the idea that the natural gradient tries to keep the diversity of the population unchanged, thanks to its relation with Kullback–Leibler divergence (Section 1.1). Since restricted Boltzmann machines are able to represent multimodal distributions on the search space, keeping a high diversity suggests that on a multimodal objective function, natural gradient algorithms will spontaneously tend to find several optima in a single run.

Let us stress that this is *not* a systematic study of the optimization performance of IGO and restricted Boltzmann machines: the function to be optimized in this experiment is extremely simple. We choose a simple setting to get a better understanding of the consequences of using the natural gradient rather than the vanilla gradient, and to illustrate the resulting difference in behavior.

5.1 IGO for restricted Boltzmann machines

The IGO method allows to build a natural search algorithm from an arbitrary family of probability distributions on an arbitrary search space. In particular, by choosing probability distributions that are richer than Gaussian or Bernoulli, one may hope to be able to optimize functions with complex shapes. Here we study how this might help optimize multimodal functions.

Both Gaussian distributions on $\mathbb{R}^d$ and Bernoulli distributions on $\{0,1\}^d$ are unimodal. So at any given time, a search algorithm using such distributions concentrates around a single point in the search space, looking around that point (with some variance). It is an often-sought-after feature for an optimization algorithm to handle multiple hypotheses simultaneously.

Here we apply the IGO framework to an example of multimodal distributions on $\{0,1\}^d$: restricted Boltzmann machines (RBMs) (Smolensky, 1986; Ackley et al., 1985). The precise definition is given below. In RBMs, values for various blocks of bits can be switched on or off depending on the activation state of *latent variables*, hence the possibility to represent multimodal distributions. Hopefully, the optimization algorithm derived from these distributions will explore several distant zones of the search space at any given time. Boltzmann machines, a superset of restricted Boltzmann machines, were used for optimization, e.g., in (Berny, 2002) (using the vanilla gradient) and found to perform better than PBIL on some functions.

We consider here the very simple situation of a fitness function with two distant optima and test whether IGO-based or vanilla-gradient-based algorithms are able to reach both optima simultaneously or only find one of them. This provides an empirical test of Proposition 1 stating that the natural gradient minimizes loss of diversity. Our study of a bimodal RBM distribution for the optimization of a bimodal function confirms that the natural gradient does indeed behave in a more natural way than the vanilla gradient: when initialized properly, the natural gradient is able to maintain diversity by fully using the RBM distribution to learn a distribution concentrating around the two modes, while the vanilla gradient always finds only one of the two modes.

Although these experiments support using a natural gradient approach, they also establish that complications can arise for estimating the inverse Fisher matrix in the case of complex distributions such as RBMs: estimation errors may lead to a singular or unreliable estimation of the Fisher matrix, especially when the distribution becomes singular or when the learning rate is large. Further research may be needed to work around this issue.

The experiments reported here, and the fitness function used, are extremely simple from an optimization viewpoint: both algorithms using the natural and vanilla gradient find an optimum in only a few steps. The emphasis here is on the specific influence of replacing the vanilla gradient with the natural gradient,

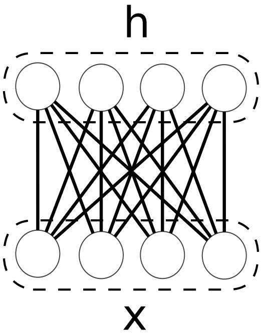


Figure 1: The RBM architecture with the observed ($\mathbf{x}$) and latent ($\mathbf{h}$) variables. In our experiments, a single hidden unit was used.

and the resulting influence on diversity and multimodality, in a simple situation.

Restricted Boltzmann machines. A restricted Boltzmann machine (RBM) (Smolensky, 1986; Ackley et al., 1985) is a probability distribution belonging to the family of undirected graphical models (also known as Markov random fields). A set of observed variables $\mathbf{x} \in \{0,1\}^{n_x}$ are given a probability using their joint distribution with unobserved latent variables $\mathbf{h} \in \{0,1\}^{n_h}$ (Ghahramani, 2004). The latent variables are then marginalized over. See Figure 1 for the graph structure of a RBM.

The probability associated with an observation $\mathbf{x} = (x_i) \in \{0,1\}^{n_x}$ and latent variable $\mathbf{h} = (h_j) \in \{0,1\}^{n_h}$ is given by

$$P_\theta(\mathbf{x}, \mathbf{h}) = \frac{e^{-E(\mathbf{x},\mathbf{h})}}{\sum_{\mathbf{x}',\mathbf{h}'} e^{-E(\mathbf{x}',\mathbf{h}')}}\,, \qquad P_\theta(\mathbf{x}) = \sum_{\mathbf{h}} P_\theta(\mathbf{x}, \mathbf{h}), \tag{17}$$

where

$$E(\mathbf{x}, \mathbf{h}) = -\sum_i a_i x_i - \sum_j b_j h_j - \sum_{i,j} w_{ij} x_i h_j \tag{18}$$

is the *energy function*. The distribution is fully parametrized by the parameter $\theta = (\mathbf{a}, \mathbf{b}, \mathbf{W})$ comprising the *bias on the observed variables* $\mathbf{a} = (a_i) \in \mathbb{R}^{n_x}$, the *bias on the latent variables* $\mathbf{b} = (b_j) \in \mathbb{R}^{n_h}$ and the *weights* $\mathbf{W} = (w_{ij}) \in \mathbb{R}^{n_x n_h}$ which account for pairwise interactions between observed and latent variables. Note that the biases can be viewed as weights, by introducing variables x_0 and h_0 always equal to one; thus in the sequel we will only write formulas involving w_{ij}, with the understanding that analogous formulas for $\mathbf{a}$ and $\mathbf{b}$ are readily obtained through this analogy.

RBM distributions are a special case of exponential family distributions with latent variables, thus the IGO equations for the RBM stem from those for general exponential families. In particular, for these distributions the gradient of the log-likelihood is well-known (Hinton., 2002). This gradient has then to be multiplied with the inverse Fisher matrix.

Both the gradient and Fisher matrix in the IGO update involve expectations over $(\mathbf{x},\mathbf{h})$. For instance the gradient of the log-likelihood is given by

$$\frac{\partial \ln P_\theta(\mathbf{x},\mathbf{h})}{\partial w_{ij}} \quad = \quad x_i h_j - \mathbb{E}_{P_\theta}[x_i h_j] \tag{19}$$

with analogous formulas for the derivatives w.r.t. a_i and to b_j. Although this quantity is often considered intractable in the context of machine learning where many variables are involved, it can be estimated accurately in the case of smaller RBMs: the expectations under P_θ can be estimated by the standard technique of Gibbs sampling (see for instance (Hinton., 2002)). We now discuss estimation of the Fisher matrix by this technique.

Estimating the Fisher matrix, and optimizing over $(\mathbf{x},\mathbf{h})$ or over $\mathbf{x}$. A restricted Boltzmann machine defines a distribution P_θ on both visible and hidden units $(\mathbf{x},\mathbf{h})$, whereas the function to optimize depends only on the visible units $\mathbf{x}$. Thus we are faced with a choice. A first possibility is to see the objective function $f(\mathbf{x})$ as a function of $(\mathbf{x},\mathbf{h})$ where $\mathbf{h}$ is a dummy variable; then we can use the IGO algorithm to optimize over $(\mathbf{x},\mathbf{h})$ using the distributions $P_\theta(\mathbf{x},\mathbf{h})$. A second possibility is to marginalize $P_\theta(\mathbf{x},\mathbf{h})$ over the hidden units $\mathbf{h}$ as in (17), to define the distribution $P_\theta(\mathbf{x})$; then we can use the IGO algorithm to optimize f over $\mathbf{x}$ using $P_\theta(\mathbf{x})$.

These two approaches yield slightly different algorithms. For instance, with $I_{w_{ij}w_{i'j'}}$ denoting the entry of the Fisher matrix corresponding to the components w_{ij} and $w_{i'j'}$ of the parameter θ, from the *dummy variable* perspective we get

$$I_{w_{ij}w_{i'j'}}(\theta) = \mathbb{E}_{P_\theta}[x_i h_j x_{i'} h_{j'}] - \mathbb{E}_{P_\theta}[x_i h_j]\,\mathbb{E}_{P_\theta}[x_{i'} h_{j'}] \tag{20}$$

whereas from the perspective of a marginalized distribution we get the same expression in which each h_j is replaced with its expectation $\bar{h}_j$ knowing $\mathbf{x}$ namely $\bar{h}_j = \mathbb{E}_{P_\theta}[h_j|\mathbf{x}] = \left(1 + e^{-b_j - \sum_i x_i w_{ij}}\right)^{-1}$ and likewise for $h_{j'}$. (Actually, since h_j and $h_{j'}$ are independent knowing $\mathbf{x}$ when $j \neq j'$, the only difference in the Fisher matrix is when $j = j'$.)

Both versions were tested on a small instance of the problem and found to be viable. However the version using $(\mathbf{x},\mathbf{h})$ is numerically more stable and requires fewer Gibbs samples to estimate the Fisher matrix, whereas the second one procudes non-invertible Fisher matrice estimates more frequently. Indeed, if $I_1(\theta)$ is the Fisher matrix at θ in the first approach and $I_2(\theta)$ in the second approach, we always have $I_1(\theta) \geqslant I_2(\theta)$ (in the sense of positive-definite matrices). This is because probability distributions on the pair $(\mathbf{x},\mathbf{h})$ carry more information than their projections on $\mathbf{x}$ only, and so the Kullback–Leibler distances will always be larger. In particular, there exist values of θ for which the Fisher matrix I_2 is not invertible whereas I_1 is.

For this reason we selected the first approach: we optimize f as a function of $(\mathbf{x},\mathbf{h})$ using IGO for the probability distributions $P_\theta(\mathbf{x},\mathbf{h})$.

Description of a step of the algorithm. The final implementation of the IGO algorithm for RBMs with step size δt and sample size N is as follows.

At each step, N samples $(\mathbf{x}_1, \mathbf{h}_1)$, ..., $(\mathbf{x}_N, \mathbf{h}_N)$ are drawn from the current distribution $P_\theta(\mathbf{x}, \mathbf{h})$ using Gibbs sampling. Each Gibbs sampling starts with an element $\mathbf{x}$ uniformly distributed in $\{0, 1\}^{n_x}$ and performs 50 Gibbs iterations for each sample.

Then the IGO update for a dataset with N sample points $(\mathbf{x}_1, \mathbf{h}_1)$, ..., $(\mathbf{x}_N, \mathbf{h}_N)$ is taken from (15):

$$\theta^{t+\delta t} = \theta^t + \delta t\, I^{-1}(\theta^t) \sum_{k=1}^{N} \widehat{w}_k \left. \frac{\partial \ln P_\theta(\mathbf{x}_k, \mathbf{h}_k)}{\partial \theta} \right|_{\theta=\theta^t} \tag{21}$$

where the $\widehat{w}_k$ are the selection weights of IGO (not to be confused with the weights of the RBM).

The gradient $\partial \ln P_\theta(\mathbf{x}_k, \mathbf{h}_k)/\partial\theta$ is estimated using (19) where the expectation is estimated by taking the average of the $T(\mathbf{x}, \mathbf{h})$ statistics over the N samples $(\mathbf{x}_1, \mathbf{h}_1)$, ..., $(\mathbf{x}_N, \mathbf{h}_N)$.

The Fisher matrix $I(\theta^t)$ is estimated using (20). The P_θ-expectation involved in this equation is estimated using a number N_s of Gibbs samples $(\mathbf{x}, \mathbf{h})$. These samples need not coincide with the IGO samples $(\mathbf{x}_1, \mathbf{h}_1)$, ..., $(\mathbf{x}_N, \mathbf{h}_N)$ and in practice we take $N_s \gg N$ as described below. Note that no f-call is needed on these samples, so in a typical optimization situation where computational cost comes from the objective function f, a large N_s may not cost too much.

Fisher matrix imprecision and invertibility. The Fisher matrix (see Eq. 20) was inverted using the QR-Algorithm, when invertible. However, the imprecision incurred by the limited sampling size sometimes leads to an unreliable or even singular estimation of the Fisher matrix (see p. 13 for a lower bound on the number of samples needed).

Having a singular Fisher estimation happens rarely at startup; however, it occurs very frequently when the probability distribution P_θ becomes too concentrated over a few points $\mathbf{x}$. Unfortunately this situation arises naturally when the algorithm is allowed to continue the optimization after the optimum has been reached; the experiments below confirm this. For this reason, in our experiments, each run using the natural gradient was "frozen" as soon as estimation of the Fisher matrix was deemed to be unreliable according to the criterion below. By "freezing" a run we mean that the value of the parameter θ is not updated anymore, but the run still contributes to the statistics reported for later times, using the frozen value of θ.

The unreliability criterion is as follows: either the estimated Fisher matrix is not invertible; or it is numerically invertible but fails the following statistical cross-validation test to check reliability of the estimate. Namely: we make two estimates $\hat{F}_1$ and $\hat{F}_2$ of the Fisher matrix on two distinct sets of $N_s/2$ Gibbs samples generated from P_θ. (The final estimate of the Fisher matrix using all N_s samples can then be obtained at little computational cost by combining $\hat{F}_1$ and $\hat{F}_2$; this has the advantage that the cross-validation procedure does not affect computational performance significantly, as the main cost is Fisher matrix estimation.) In the ideal case $\hat{F}_1$ and $\hat{F}_2$ are close.

At all steps we tested whether the rescaled squared Frobenius norms

$$\begin{aligned} &\frac{1}{\dim(\theta)}\|\hat{F}_2^{-1/2}\hat{F}_1\hat{F}_2^{-1/2} - \mathrm{Id}\|^2_{\text{Frobenius}} \\ \text{and } &\frac{1}{\dim(\theta)}\|\hat{F}_1^{-1/2}\hat{F}_2\hat{F}_1^{-1/2} - \mathrm{Id}\|^2_{\text{Frobenius}} \end{aligned}$$

which ideally are equal to 0, are both smaller than 1: this is a crude test to check eigenvalue explosion. Note that $\hat{F}_1$ and $\hat{F}_2$ represent quadratic forms on parameter space, and $\hat{F}_2^{-1/2}\hat{F}_1\hat{F}_2^{-1/2}$ represents one of them in an orthonormal basis for the other. This test is independent of θ-parametrization. The squared Frobenius norm of $\hat{F}_2^{-1/2}\hat{F}_1\hat{F}_2^{-1/2} - \mathrm{Id}$ is computed as the trace of $(\hat{F}_1\hat{F}_2^{-1} - \mathrm{Id})^2$.

If at some point during the optimization the Fisher estimation becomes singular or unreliable according to this criterion, the corresponding run is frozen.

The number of runs that were frozen before reaching an optimum in our experiments below is reported in Table 1. Choosing a small enough learning rate appears to limit the problem.

#runs	N	δt	#iters	O	S	CV	other
300	10,000	0.5	100	98.3	0.0	1.7	0.0
		1.	100	98.0	0.0	2.0	0.0
		2.	100	95.7	0.0	4.3	0.0
		0.002	25,000	100.0	0.0	0.0	0.0
		0.004	25,000	95.0	0.0	5.0	0.0
20	10	0.008	25,000	90.0	0.0	10.0	0.0
		0.1	500	5.0	25.0	70.0	0.0
		0.2	500	0.0	30.0	70.0	0.0
		0.4	500	0.0	20.0	80.0	0.0
		0.8	500	0.0	40.0	60.0	0.0

Table 1: Percentage of runs according to their state after #iter iterations: found at least one optimum (O), frozen before hitting an optimum because of a singular matrix (S) or cross-validation test failure (CV). The confidence margin (1σ) over the reported percentages is less than ±2.9 percent points for 300 runs and ±11.2 percent points for 20 runs.

Computational time. Contrary to usual optimization situations, in the experiments below the function f has a negligible cost. The complexity of the algorithm is then largely determined two factors: Gibbs sampling in the RBM and Fisher matrix computation given the samples. To ensure stability of the Fisher matrix estimate (see above), the number of samples N_s used in the estimation has to scale at least like $\dim(\theta)$. With this assumption, Gibbs sampling which essentially scales like $N_s \dim(\theta)$ then scales like $\dim(\theta)^2$, with a leading constant proportional to the number of Gibbs steps used. The computation of the Fisher matrix itself scales like $N_s \dim(\theta)^2$ because for each sample, we have to compute an additive term of the Fisher matrix which has $\dim(\theta)^2$ entries. This means that the Fisher matrix computation scales like $\dim(\theta)^3$ and can therefore be expected to be the dominating factor in the running-time of experiments involving larger models. The cost of the Fisher matrix inversion is expected to scale like $\dim(\theta)^3$ as the size of the RBM increases. In practice this cost was significantly smaller than that of matrix estimation (Figure 2).

Figure 2 gives empirical running times[3] (in log-log scale) for one natural gradient step update in the experimental setting described below, together with the corresponding Gibbs sampling, Fisher matrix computation, QR-inversion and cross-validation times. It shows that indeed Fisher matrix estimation and the associated Gibbs sampling are responsible for most of the computational cost.

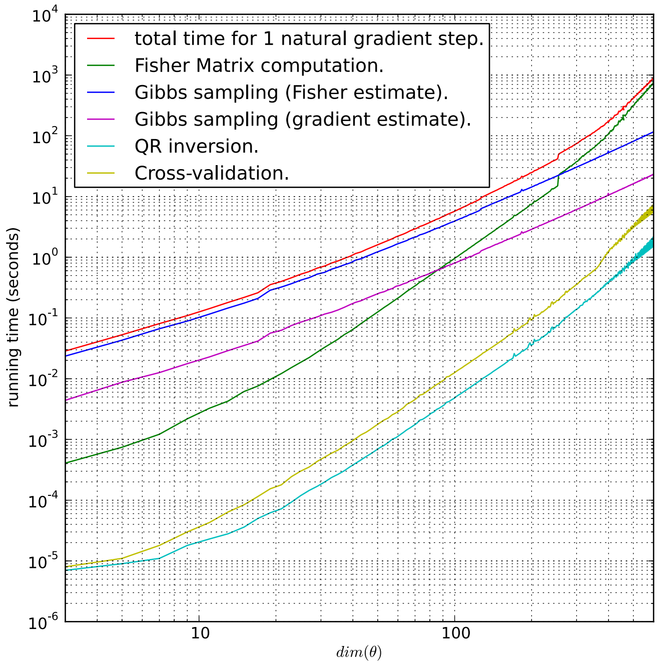


Figure 2: Log-log plot of the empirical running time in seconds of 1 step of the natural gradient algorithm, and corresponding times for Gibbs sampling, Fisher matrix computation, QR inversion and cross-validation, for a RBM using $\dim(\theta)$ parameters. The number of samples for estimating the Fisher matrix is assumed to scale like $100 \times \dim(\theta)$ to ensure stability of the estimate. We use 5 times fewer samples to compute the gradient estimate and those samples are distinct from those used to compute the Fisher matrix estimate.

Gibbs sampling is not the fastest known method for sampling in a large RBM: a method such as parallel tempering (Desjardins et al., 2010) has the

[3] on an Intel Xeon CPU E5420 @ 2.50GHz with 16Gbytes of memory

same asymptotic complexity $N_s \dim(\theta)$ for N_s samples, but usually converges faster and would therefore be more suitable for large RBMs.

A possible way to reduce the computational burden of Fisher matrix evaluation would be to use a larger learning rate together with a larger number of gradient samples at each step (e.g., using the same RBM samples for the gradient and the Fisher matrix estimation). Although this would increase the number of f-calls at each iteration, the experiments below suggest that it may be possible to achieve convergence in a smaller number of steps.

With this in mind, a naïve application of the natural gradient can be expected to be considerably costly for larger RBMs. Strategies already exist to deal with these issues (e.g., (Le Roux et al., 2007)): for instance, the Fisher matrix is not usually recomputed from scratch at each step, but a discount factor is applied to its previous value and a few new samples are incorporated; Fisher matrix inversion can then be done iteratively using the Woodbury formula; lower-rank approximations of the Fisher matrix can be used.

5.2 Experimental setup

Problem setting. In our experiments, we look at the optimization of the two-min function defined below with a bimodal RBM: an RBM with only one latent variable ($n_h = 1$). Such an RBM is bimodal because it has two possible configurations of the latent variable: $\mathbf{h} = 0$ or $\mathbf{h} = 1$, and given $\mathbf{h}$, the observed variables are independent and distributed according to two Bernoulli distributions. We used $n_x = 40$, and therefore $\dim(\theta) = 81$.

Set a parameter $\mathbf{y} \in \{0, 1\}^d$. The *two-min function based at* $\mathbf{y}$ is defined as:

$$f_{\mathbf{y}}(\mathbf{x}) = \min\left(\sum_i |x_i - y_i|, \sum_i |(1 - x_i) - y_i)|\right) \tag{22}$$

This function of $\mathbf{x}$ has two optima: one at $\mathbf{y}$, the other at its binary complement $\bar{\mathbf{y}}$.

We ran both the IGO algorithm as described above, and the version using the vanilla gradient instead of the natural gradient (that is, omitting the Fisher matrix in (21)).

Parameter setting. We used two different values $N = 10,000$ and $N = 10$ for the population size of the IGO algorithm. The rather comfortable setting $N = 10,000$ allows for a good illustration of the behavior of the theoretical IGO flow which corresponds to $N \to \infty$ and $\delta t \to 0$, whereas $N = 10$ is a much more realistic value for an optimization problem. The values of δt were chosen in a range as reported below.

The number of sample points used for estimating the Fisher matrix is set to $N_s = 10,000$: large enough to ensure the stability of the Fisher matrix estimates.

For the quantile rewriting of f (Section 1.2), we followed a selection scheme often applied in evolution strategies (Rechenberg, 1994) and set w to be $w(q) = \mathbb{1}_{q \leqslant 1/5}$ so that the best 20% of points in a sample are given weight 1 for the update, while other points are given weight 0. Ties were dealt with according to (13).

Initialization. For initialization of the RBMs, we choose random values of the parameters that guarantee that each variable (observed or latent) has a probability of activation close to 1/2 at startup, i.e. the initial distribution is almost uniform. This is in line with the discussion in Section 1.1 about starting with large initial diversity. Namely, the weights $\mathbf{w}$ are sampled from a normal distribution centered around zero and of standard deviation $1/\sqrt{n_x \times n_h}$, where n_x is the number of observed variables (dimension d of the problem) and n_h is the number of latent variables ($n_h = 1$ in our case), so that initially the energies E are not too large. Then the bias parameters are initialized as $b_j \leftarrow -\sum_i \frac{w_{ij}}{2}$ and $a_i \leftarrow -\sum_j \frac{w_{ij}}{2} + \mathcal{N}(\frac{0.01}{n_x^2})$: this setting guarantees initial probabilities of activation close to 1/2 for all variables.

For each run, the parameter $\mathbf{y}$ of the two-max function was sampled randomly in $\{0,1\}^{n_x}$ in order to ensure that the presented results are not dependent on a particular location of the optima.

Step size. The value of δt is indicated for each plot. Note that the values of the parameter δt for the two gradients used are not directly comparable from a theoretical viewpoint (they correspond to parametrizations of different trajectories in Θ-space, and identifying vanilla δt with natural δt is meaningless). For a given δt the natural gradient tends to move faster than the vanilla gradient. For this reason, we selected larger values of δt for the vanilla gradient: a factor 4 seems to yield roughly comparable moving speeds in practice. (This is consistent with the remark that at time 0, the largest terms of the Fisher matrix are equal to 1/4 and most non-diagonal terms vanish, as can be seen from (20) given that the parameters are initialized close to the uniform distribution.)

Reading the plots. The plots reported below are of two types: trajectories of single runs (such as Fig. 3), and aggregate over K runs (such as Fig. 4). For the aggregate plots, we present the median together with error bars indicating the 16th percentile and the 84th percentile over the K runs, namely, 16% of runs are below the lower error bar and likewise for the upper error bar (for a Gaussian variable this is the same as the mean and mean±stddev, but applies to non-Gaussian variables and is better behaved under f-reparametrization).

Source code. The code used for these experiments can be found on the internet at `http://www.ludovicarnold.com/projects:igocode` .

5.3 Experimental results

Approaching the optima. We now report how the vanilla and natural gradient approach the two optima of the objective function (remember the objective function is bimodal). We start with the large population case $N = 10,000$.

Figure 3 shows ten trajectories of the natural gradient (left column) and vanilla gradient (right column), using a large population size ($N = 10,000$). At each step, we report the smallest distance of the N sample points in the population to the closest optimum of the objective function (top row), as well as the smallest distance of the N sample points in the population to the *other* optimum of the objective function (bottom row). Figure 4 reports the same quantities aggregated over 300 independent runs (median over all runs, error bars as described above) for various settings of δt.

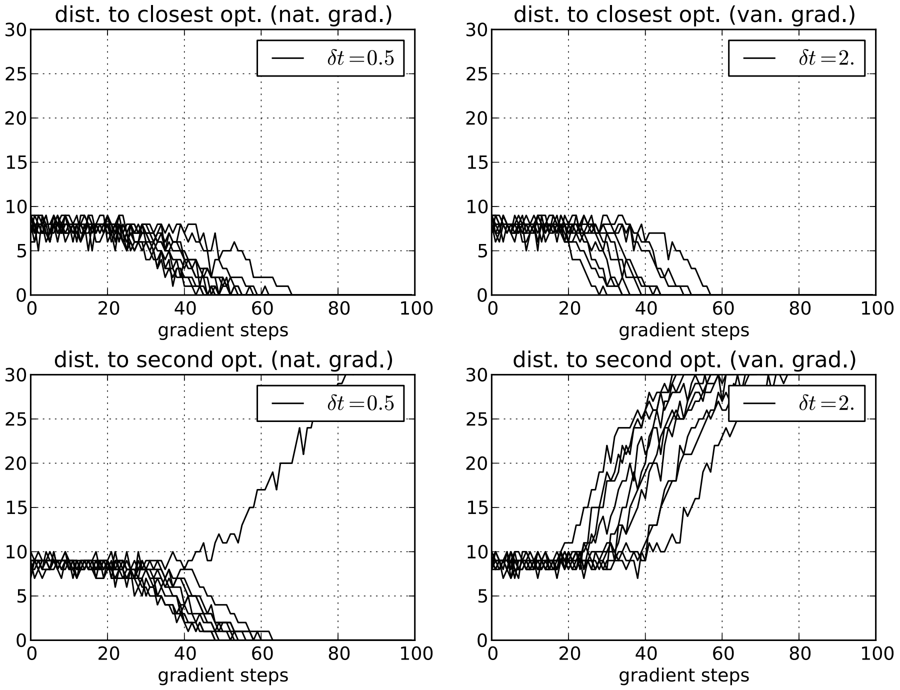


Figure 3: Distance to the two optima using a large population of 10,000 samples, during 10 IGO optimization runs and 10 vanilla gradient runs

The small population case $N = 10$ is illustrated in Figure 5 (single runs) and Figure 6 (aggregate over 20 runs). For small population sizes, the dynamics is noisier, and smaller step sizes δt have been used to average out the noise (resulting in roughly the same total number of f-calls, see the discussion of finite sample size in Section 6). The results are broadly similar to those with $N = 10,000$, as revealed by the aggregate plots (Figure 6), but individual runs can exhibit less regular behavior (see the "spike" on the bottom-left graph of Figure 5, presumably due to an unreliable estimate of the Fisher matrix, though the next iteration apparently cancels the effect). Smaller learning rates seem to behave better.

Larger step sizes δt have also been tested for the small population case (Figures 7 and 8). It appears that the natural gradient is more sensitive to large step sizes than the vanilla gradient: with the natural gradient for small population at large learning rates, many runs get frozen before they reach an optimum due

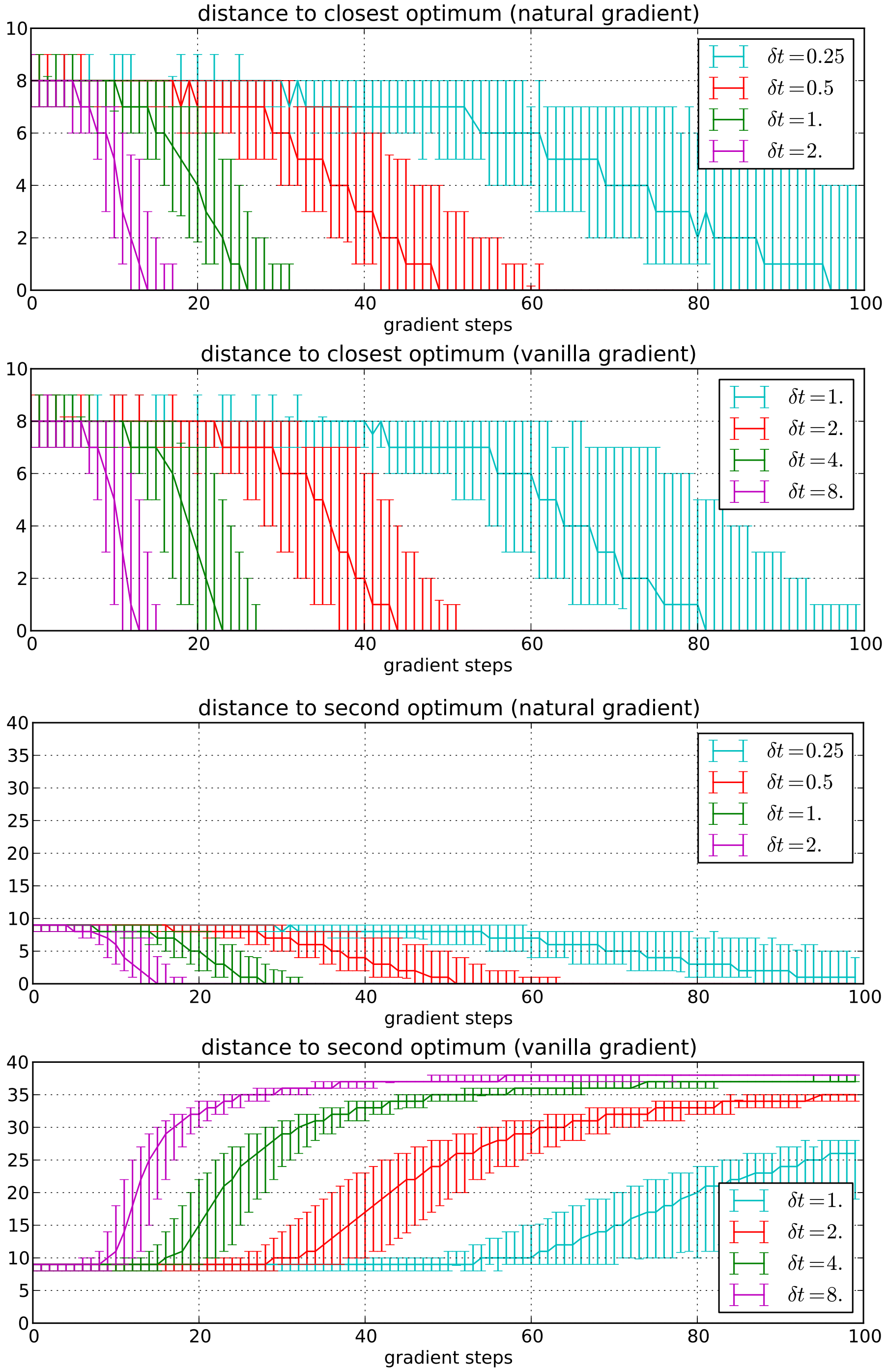


Figure 4: Median distance to an optimum with a large population of 10,000 samples over 300 optimization runs, respectively using IGO or the vanilla gradient. Top two figures: distance to the closest optimum; bottom two figures: distance to other optimum. Error bars indicate the 16th and 84th quantile over the runs.

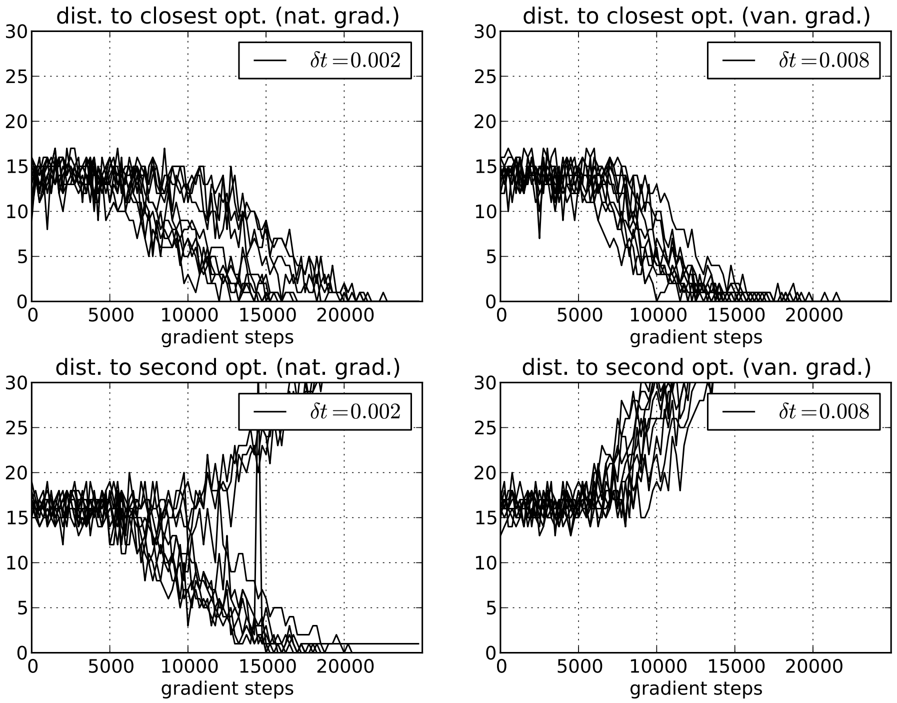


Figure 5: Distance to closest optimum using a small population of 10 samples, during 10 IGO optimization runs and 10 vanilla gradient runs. The “spike” on the bottom-left plot is presumably due to an unreliable estimate of the Fisher matrix at one step.

to the problem of estimating the Fisher matrix (Table 1).

Diversity and h-statistics. Predictably, both algorithms are able to find at least one optimum of the very simple two-min function in a few steps. However, the methods fare very differently when we look at the distance from the sample points to *both* optima (Figures 3, 4, 5 and 6).

Most runs using the natural gradient get close to both optima simultaneously, reflecting the fact that the distribution P_θ becomes bimodal, as allowed by the RBM. The two optima are generally reached within a few steps of each other.

This is consistent with the intuition of Section 1.1 about maintaining diversity in natural gradient optimization. This property of IGO depends, of course, on having initialized the RBM with enough diversity. When initialized properly so that each variable (observed and latent) has a probability 1/2 of being equal to 1, the initial RBM distribution has maximal diversity over the search space and is at equal distance from the two optima of the function. From this starting position, IGO is then able to increase the likelihood of the two optima at the same time.

By stark contrast, the vanilla gradient is *never* able to go towards both optima at the same time. In fact, the vanilla gradient only finds one optimum at the expense of the other: for all values of δt, the distance to the second optimum

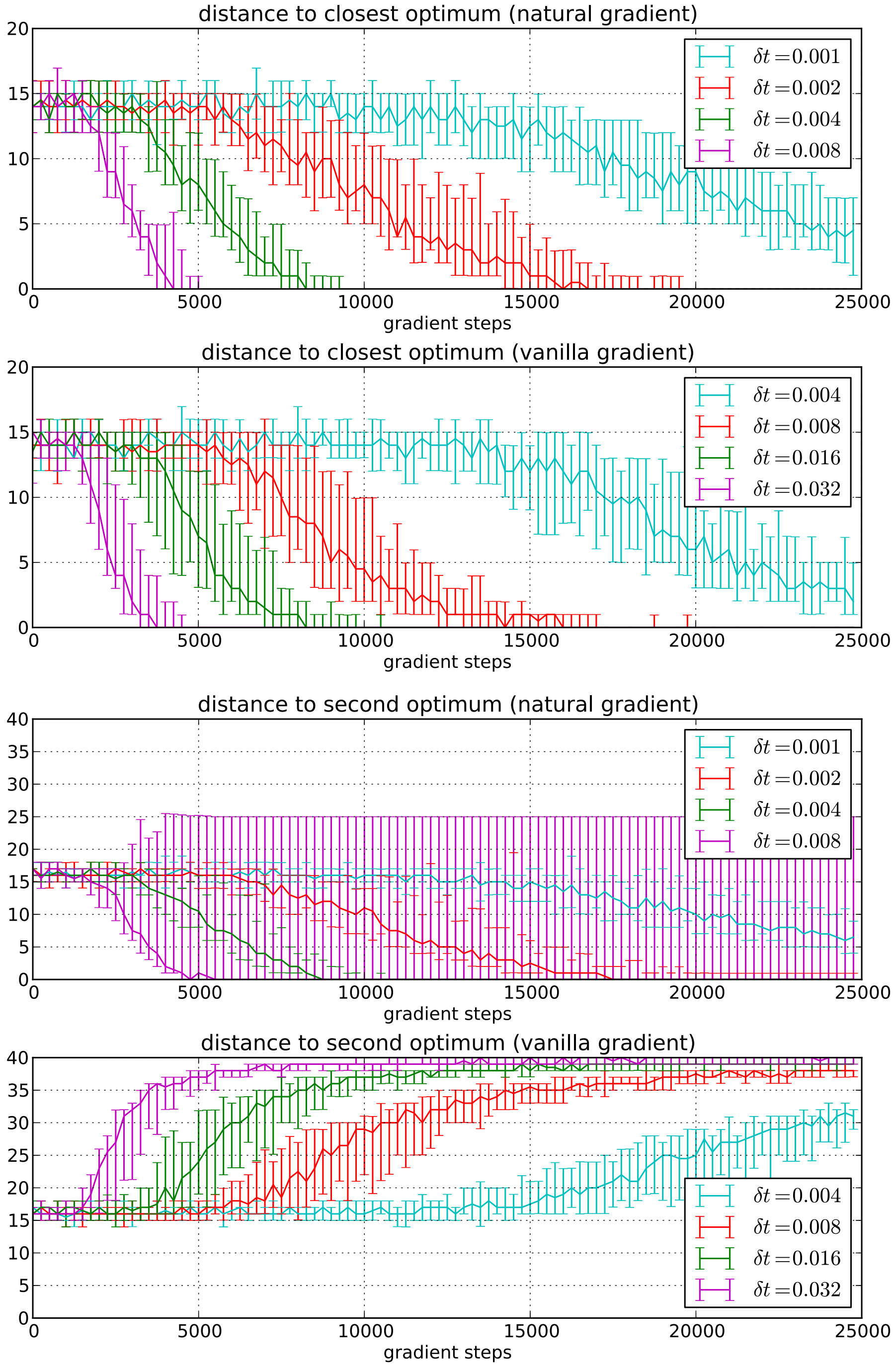


Figure 6: Median distance to an optimum using a small population of 10 samples in either 20 IGO optimization runs or 20 vanilla gradient runs, respectively. Top two figures: distance to the closest optimum; bottom two figures: distance to the other optimum. Error bars indicate the 16th and 84th quantile.

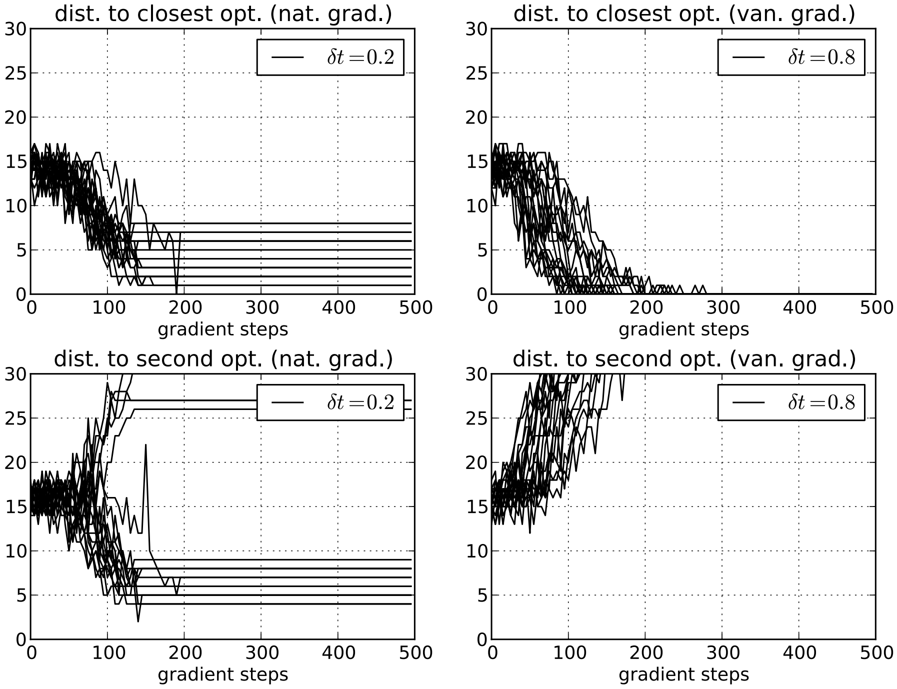


Figure 7: Distance to closest (above) and the other (below) optimum using a small population of 10 samples and learning rates too large to ensure convergence of IGO. 20 IGO optimization runs and 20 vanilla gradient runs.

increases gradually and approaches the maximum possible distance. So in these experiments the vanilla gradient never exploits the possibility offered by the RBM to create a bimodal probability distribution P_θ.

As mentioned earlier, each value 0 or 1 of the latent variable $\mathbf{h}$ corresponds to a mode of the distribution. To illustrate the evolution of uni- or bi-modality of P_θ, we plot in Figure 9 the average value of $\mathbf{h}$ in the population over time (aggregated over 300 runs). An average value close to 1/2 means that the distribution samples from both modes $\mathbf{h} = 0$ or $\mathbf{h} = 1$ with a comparable probability. Conversely, average values close to 0 or 1 indicate that the distribution gives most probability to one mode at the expense of the other and is thus essentially unimodal. We can see that with IGO, the average value of $\mathbf{h}$ stays remarkably centered during the whole optimization procedure: the distribution stays bimodal. As for the vanilla gradient, we see that the statistics for $\mathbf{h}$ quickly tend to converge to 1: one of the two modes of the distribution has been lost during optimization.

Hidden breach of symmetry by the vanilla gradient. The experiments reveal a curious phenomenon (Figure 9): the vanilla gradient loses multimodality by always setting the hidden variable h to 1, not to 0. (We detected no obvious asymmetry on the visible units x.) So the vanilla gradient for RBMs seems to favor $h = 1$.

Of course, exchanging the values 0 and 1 for the hidden variables in a

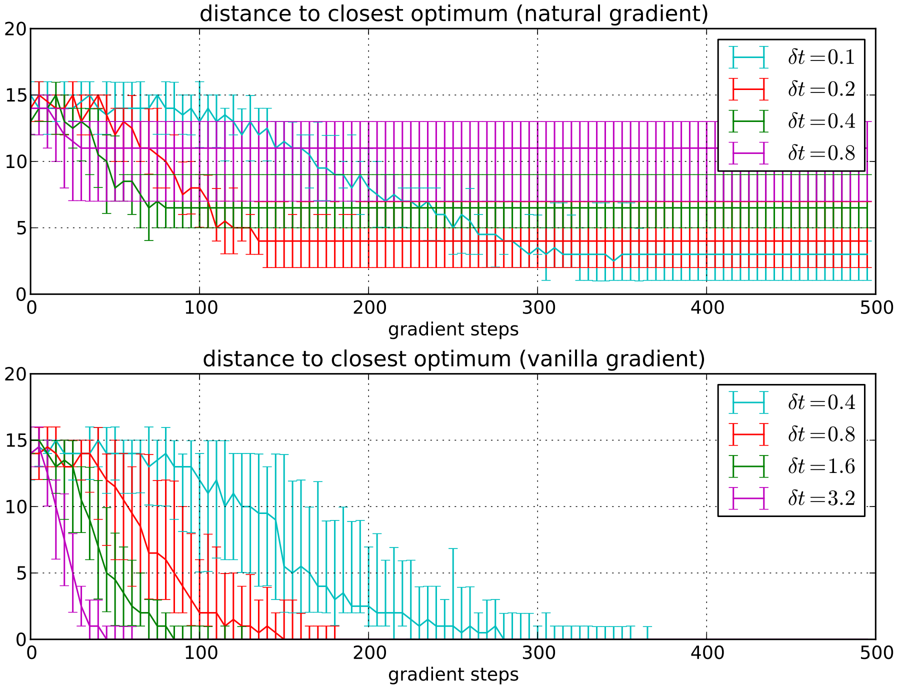


Figure 8: Median distance (over 20 runs) to closest optimum using a small population of 10 samples and learning rates too large to ensure convergence of IGO. Above: natural gradient, below: vanilla gradient. Error bars indicate the 16th and 84th quantile over the 20 runs.

restricted Boltzmann machine still gives a distribution of another Boltzmann machine. More precisely, changing h_j into $1 - h_j$ is equivalent to resetting $a_i \leftarrow a_i + w_{ij}$, $b_j \leftarrow -b_j$, and $w_{ij} \leftarrow -w_{ij}$. IGO and the natural gradient are impervious to such a change[4].

The vanilla gradient implicitly relies on the Euclidean norm on parameter space, as explained in Section 1.1. For this norm, the distance between the RBM distributions (a_i, b_j, w_{ij}) and (a'_i, b'_j, w'_{ij}) is simply $\sum_i |a_i - a'_i|^2 + \sum_j \left|b_j - b'_j\right|^2 + \sum_{ij} \left|w_{ij} - w'_{ij}\right|^2$. However, the change of variables $a_i \leftarrow a_i + w_{ij}$, $b_j \leftarrow -b_j$, $w_{ij} \leftarrow -w_{ij}$ does *not* preserve this Euclidean metric. Thus, exchanging 0 and 1 for the hidden variables will result in two different vanilla gradient ascents. The observed asymmetry on h is a consequence of this implicit asymmetry.

The same asymmetry actually exists for the visible variables x_i; but this does not prevent convergence to an optimum in our situation, since any gradient descent eventually reaches some local optimum.

Of course it is possible to use parametrizations for which the vanilla gradient will be more symmetric: for instance, using $-1/1$ instead of $0/1$ for the variables,

[4] see section 2.

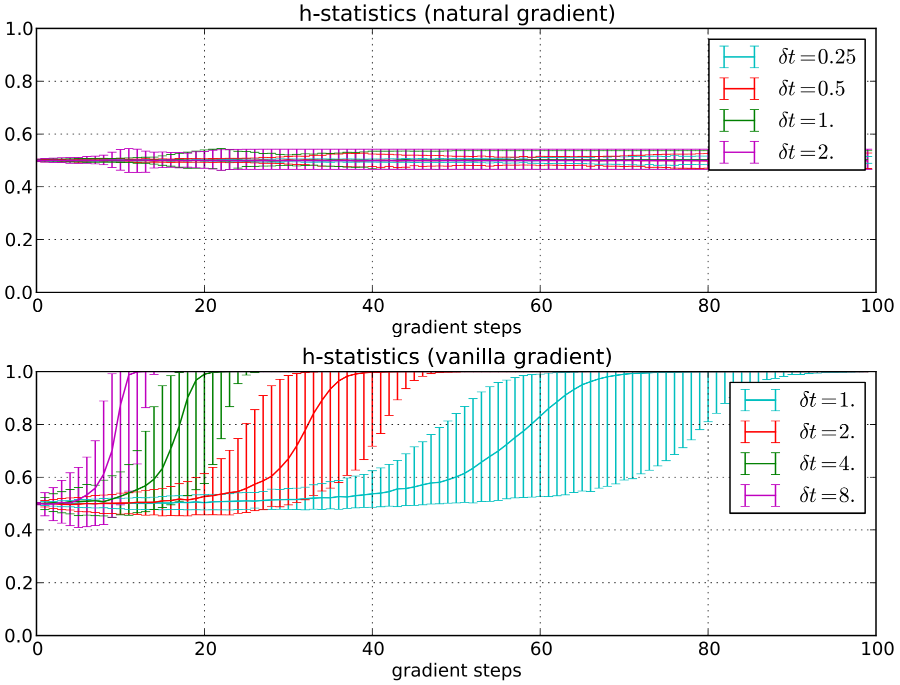


Figure 9: Median of average h-statistics in 300 IGO optimization runs and 300 vanilla gradient runs using a large population of 10,000 samples. 2,000 samples are used for selection at each step. Error bars indicate the 16th and 84th quantile over the runs.

or defining the energy by

$$E(\mathbf{x}, \mathbf{h}) = -\textstyle\sum_i A_i(x_i - \frac{1}{2}) - \sum_j B_j(h_j - \frac{1}{2}) - \sum_{i,j} W_{ij}(x_i - \frac{1}{2})(h_j - \frac{1}{2}) \qquad (23)$$

with “bias-free” parameters A_i, B_j, W_{ij} related to the usual parametrization by $w_{ij} = W_{ij},\ a_i = A_i - \frac{1}{2}\sum_j w_{ij}\ b_j = B_j - \frac{1}{2}\sum_i w_{ij}$. The vanilla gradient might perform better in this parametrization.

However, we adopted the approach of using a family of probability distributions found in the literature, with the parametrization commonly found in the literature. We then used the vanilla gradient and the natural gradient on these distributions—and indeed the vanilla gradient or an approximation thereof is routinely applied to RBMs in the literature to optimize the log-likelihood of data (Hinton., 2002; Hinton et al., 2006; Bengio et al., 2007). It was not obvious a priori (at least for us) that the vanilla gradient ascent favors $h = 1$.

This directly illustrates the specific influence of the chosen gradient (the two implementations only differ by the inclusion of the Fisher matrix): the natural gradient offers a systematic way to recover symmetry from a non-symmetric gradient update.

Note that symmetry alone does not explain the fact that IGO reaches the two optima simultaneously: indeed, a symmetry-preserving stochastic algorithm could very well end up on either single optimum with 50% probability in each run. The diversity-preserving property of IGO offers a reasonable interpretation of why this does not happen.

5.4 Convergence to the continuous-time limit

In the previous figures, it looks like changing the parameter δt only results, to some extent, in a time speedup of the plots.

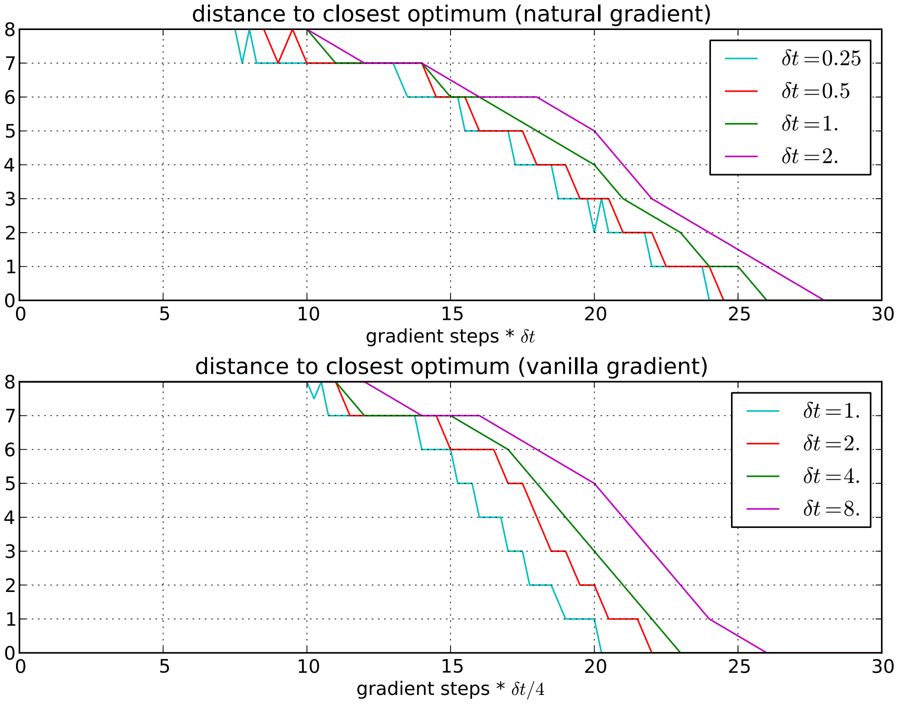


Figure 10: Median distance to closest optimum during 300 IGO optimization runs and 300 vanilla gradient runs using a large population of 10,000 samples plotted in intrinsic time. 2,000 samples are used for selection at each step.

This can be checked on Figure 10, where we plot the distance to the closest optimum as a function of $t = (\delta t \times \text{number of steps})$ instead of the number of steps. An asymptotic trajectory seems to emerge when δt decreases.

This is because update rules of the type $\theta \leftarrow \theta + \delta t\, \nabla_\theta g$ (for either gradient) are Euler approximations of the continuous-time ordinary differential equation $\frac{\mathrm{d}\theta}{\mathrm{d}t} = \nabla_\theta g$, with each iteration corresponding to an increment δt of the time t. Consequently, for small enough δt, the algorithm after k steps approximates the IGO flow or vanilla gradient flow at time $t = k.\delta t$. Thus for the natural gradient, the asymptotic trajectory can be interpreted as the fitness of samples of the continuous-time IGO flow.

So on one hand, for this kind of optimization algorithms it would make theoretical sense to plot the results according to the "intrinsic time" $t = k.\delta t$ of the underlying continuous-time object, to illustrate properties that do not depend on the setting of the parameter δt. Still, the raw number of steps is more directly related to algorithmic cost.

6 Further discussion and perspectives

A single framework for optimization on arbitrary spaces. A strength of the IGO viewpoint is to automatically provide a unique, distinct, and arguably optimal optimization algorithm from any family of probability distributions on any given space, discrete or continuous. This has been illustrated with restricted Boltzmann machines. IGO algorithms feature good invariance properties and make a least number of arbitrary choices.

In particular, IGO describes several well-known optimization algorithms within a single framework. For instance, to the best of our knowledge, PBIL has never been described as a natural gradient ascent in the literature[5].

For Gaussian measures, algorithms of the same form (15) had been developed previously (Hansen and Ostermeier, 2001; Wierstra et al., 2008) and their close relationship with a natural gradient ascent had been recognized (Akimoto et al., 2010; Glasmachers et al., 2010).

The wide applicability of natural gradient approaches seems not to be widely known in the optimization community (though see (Malagò et al., 2008)).

About quantiles. The IGO flow in (6) has, to the best of our knowledge, never been defined before. The introduction of the quantile-rewriting (3) of the objective function provides the first rigorous derivation of quantile- or rank- or comparison-based natural optimization from a gradient ascent in θ-space.

NES and CMA-ES have been claimed to maximize $-\mathbb{E}_{P_\theta} f$ via natural gradient ascent (Wierstra et al., 2008; Akimoto et al., 2010). However, we have proved that when the number of samples is large and the step size is small, the NES and CMA-ES updates converge to the IGO flow, not to the similar flow with the gradient of $\mathbb{E}_{P_\theta} f$. So we find that in reality these algorithms maximize $\mathbb{E}_{P_\theta} W^f_{\theta^t}$, where $W^f_{\theta^t}$ is a decreasing transformation of the f-quantiles under the current sample distribution.

Also, in practice, maximizing $-\mathbb{E}_{P_\theta} f$ is a rather unstable procedure and has been discouraged, see for example (Whitley, 1989).

About choice of P_θ: learning a model of good points. The choice of the family of probability distributions P_θ plays a double role.

First, it is analogous to choosing the variation operators (namely *mutation* or *recombination*) as seen in evolutionary algorithms: indeed, P_θ encodes possible moves according to which new sample points are explored.

Second, optimization algorithms using distributions can be interpreted as learning a probabilistic model of where the points with good values lie in the search space. With this point of view, P_θ describes *richness of this model*: for

[5] Thanks to Jonathan Shapiro for an early argument confirming this property (personal communication).

instance, restricted Boltzmann machines with h hidden units can describe distributions with up to 2^h modes, whereas the Bernoulli distribution used in PBIL is unimodal. This influences, for instance, the ability to explore several valleys and optimize multimodal functions in a single run.

More generally, the IGO framework makes it tempting to use more complex models of where good points lie, inspired, e.g., from machine learning, and adapt them for optimization. The restricted Boltzmann machines of Section 5 are a first step in this direction. The initial idea behind these machines is that each hidden unit controls a block of coordinates of the search space (a block of features), so that the optimization algorithm hopefully builds a good model of which features must be activated or de-activated together to obtain good values of f. This is somewhat reminiscent of a crossover operator: if observation of good points shows that a block of features go together, this information is stored in the RBM structure and this block may be later activated as a whole, thus effectively transferring blocks of features from one good solution to another. Inspired by models of deep learning (Bengio et al., 2012), one might be tempted to stack such models on top of each other, so that optimization would operate on a more and more abstract representation of the problem. IGO and the natural gradient might help in exploiting the added expressivity that comes with richer models (as in our simple experiment the vanilla gradient ignores the additional expressivity of RBMs with respect to Bernoulli distributions).

Natural gradient and parametrization invariance. Central to IGO is the use of natural gradient, which follows from θ-invariance and makes sense on any search space, discrete or continuous.

While the IGO flow is exactly θ-invariant, for any practical implementation of an IGO algorithm, a parametrization choice has to be made. Still, since all IGO algorithms approximate the IGO flow, two parametrizations in combination with IGO will differ less than the same two parametrizations in combination with another algorithm (such as the vanilla gradient or the smoothed CEM method)—at least if the learning rate δt is not too large. The chosen parametrization becomes more relevant as the step size δt increases.

On the other hand, natural evolution strategies have never strived for θ-invariance, but instead, the chosen parametrization (Cholesky, exponential) has been deemed a relevant feature. We believe the term natural evolution strategy should rather be used independently of the chosen parameterization, thereby referring to the usage of the natural gradient as the main principle for the update of distribution parameters.

IGO, maximum likelihood and cross-entropy. The cross-entropy method (CEM) (de Boer et al., 2005) can be used to produce optimization algorithms given a family of probability distributions on an arbitrary space, by performing a jump to a maximum likelihood estimate of the parameters.

We have seen that the standard CEM is an IGO algorithm *in a particular parametrization*, with a learning rate δt equal to 1. However, it is well-known, both theoretically and experimentally (Branke et al., 2007; Hansen, 2006;

Wagner et al., 2004), that standard CEM loses diversity too fast in many situations. The usual solution (de Boer et al., 2005) is to reduce the learning rate (smoothed CEM), but this breaks the reparametrization invariance of non-smoothed CEM.

On the other hand, the IGO flow can be seen as a *maximum likelihood update with infinitesimal learning rate*. This interpretation allows to define a particular IGO algorithm, the IGO-ML: it performs a maximum likelihood update with an arbitrary learning rate, and keeps the reparametrization invariance. It coincides with CEM when the learning rate is set to 1, but it differs from smoothed CEM by the exchange of the order of argmax and averaging. We argue that this new algorithm may be a better way to reduce the learning rate and achieve smoothing in CEM.

Standard CEM can lose diversity, yet is a particular case of an IGO algorithm: this illustrates the fact that reasonable values of the learning rate δt depend on the parametrization. We have studied this phenomenon in detail for various Gaussian IGO algorithms.

Why would a smaller learning rate perform better than a large one in an optimization setting? It might seem more efficient to jump directly to the maximum likelihood estimate of currently known good points, instead of performing a slow gradient ascent towards this maximum.

However, optimization faces a “moving target”, contrary to a learning setting in which the example distribution is often stationary. Currently known good points heavily depend on the current distribution and are likely not to indicate the *position* at which the optimum lies, but, rather, the *direction* in which the optimum is to be found. After an update, the next elite sample points are going to be located somewhere new. So the goal is certainly not to settle down around these currently known points, as a maximum likelihood update does: by design, CEM only tries to reflect status-quo (even for $N = \infty$), whereas IGO tries to move somewhere. When the target moves over time, a progressive gradient ascent is more reasonable than an immediate jump to a temporary optimum, and realizes a kind of time smoothing.

This phenomenon is most clear when the number of sample points is small. Then, a full maximum likelihood update risks losing a lot of diversity; it may even produce a degenerate distribution if the number of sample points is smaller than the number of parameters of the distribution. On the other hand, for smaller δt, the IGO algorithms do, by design, try to maintain diversity by moving as little as possible from the current distribution P_θ in Kullback–Leibler divergence. A full ML update disregards the current distribution and tries to move as close as possible to the elite sample in Kullback–Leibler divergence (de Boer et al., 2005), thus realizing maximal diversity loss. This makes sense in a non-iterated scenario such as batch learning but is unsuited for optimization.

Diversity and multiple optima. The IGO framework emphasizes the relation of natural gradient and diversity: we argued that IGO provides minimal diversity change for a given objective function increment. In particular, provided the initial diversity is large, diversity is kept at a maximum at startup.

This theoretical relationship has been confirmed experimentally for restricted Boltzmann machines.

On the other hand, using the vanilla gradient does not lead to a balanced distribution between the two optima in our experiments. Using the vanilla gradient introduces hidden arbitrary choices between those points (more exactly between moves in Θ-space). This results in unnecessary and unwelcome loss of diversity, and might also be detrimental at later stages in the optimization. This may reflect the fact that the Euclidean metric on the space of parameters, implicitly used in the vanilla gradient, becomes less and less meaningful for gradient descent on complex distributions.

IGO and the natural gradient are certainly relevant to the well-known problem of exploration-exploitation balance: as we have seen, arguably the natural gradient realizes the largest improvement in the objective with the least possible change of diversity in the distribution.

More generally, like other distribution-based optimization algorithms, IGO tries to learn a model of where the good points are. This is typical of machine learning, one of the contexts for which the natural gradient was studied. The conceptual relationship of IGO and IGO-like optimization algorithms with machine learning is still to be explored and exploited.

We now present some ideas which we believe would be worth exploring.

Adaptive learning rate. Comparing consecutive updates to evaluate a learning rate or step size is an effective measure. For example, in back-propagation, the update sign has been used to adapt the learning rate of each single weight in an artificial neural network (Silva and Almeida, 1990). In CMA-ES, a step size is adapted depending on whether recent steps tended to move in a consistent direction or to backtrack. This is measured by considering the changes of the mean m of the Gaussian distribution.

For a probability distribution P_θ on an arbitrary search space, in general no notion of mean may be defined. However, it is still possible to define “backtracking” in the evolution of θ as follows.

Consider two successive updates $\delta\theta^t = \theta^t - \theta^{t-\delta t}$ and $\delta\theta^{t+\delta t} = \theta^{t+\delta t} - \theta^t$. Their scalar product in the Fisher metric $I(\theta^t)$ is

$$\langle \delta\theta^t, \delta\theta^{t+\delta t} \rangle = \sum_{ij} I_{ij}(\theta^t)\, \delta\theta^t_i\, \delta\theta^{t+\delta t}_j .$$

Dividing by the associated norms will yield the cosine $\cos\alpha$ of the angle between $\delta\theta^t$ and $\delta\theta^{t+\delta t}$.

If this cosine is positive, the learning rate δt may be increased. If the cosine is negative, the learning rate probably needs to be decreased. Various schemes for the change of δt can be devised; for instance, inspired by step-size control commonly used in evolution strategies, one can multiply δt by $\exp(\beta(\cos\alpha))$ or $\exp(\beta\,\mathrm{sign}(\cos\alpha))$, where $\beta \approx \min(N/\dim\Theta, 1/2)$.

As before, this scheme is constructed to be robust w.r.t. reparametrization of θ, thanks to the use of the Fisher metric. However, for large learning rates δt, in practice the parametrization might well become relevant.

A consistent direction of the updates does not necessarily mean that the algorithm is performing well: for instance, when CEM/EMNA exhibits premature convergence (see above), the parameters consistently move towards a zero covariance matrix and the cosines above are positive. The desired target value for the cosine is zero.

Geodesic parametrization. While the IGO flow is fully invariant under θ-reparametrization, an IGO algorithm does depend on the choice of parametrization for θ, even if for small δt the difference between two IGO algorithms is $O(\delta t^2)$, one order of magnitude smaller than between IGO and vanilla gradient in general.

Still one can wonder how to discretize time in the IGO flow in a fully intrinsic way, not depending at all on a parametrization for θ. A first possibility is given by the IGO-ML algorithm —this means, for exponential families, that we can decide to single out the parametrization by expectation parameters.

Another, more geometric solution is to use *geodesics* on the statistical manifold. This means we approximate the trajectories of the IGO flow by successive geodesic segments of length δt in the Fisher metric, where the initial direction of each segment is given by the direction of the IGO flow.

More precisely, if $Y = \sum_{i=1}^{N} \widehat{w}_i \; \widetilde{\nabla}_\theta \ln P_\theta(x_i)\Big|_{\theta=\theta^t} = I^{-1}(\theta^t) \sum_{i=1}^{N} \widehat{w}_i \; \frac{\partial \ln P_\theta(x_i)}{\partial \theta}\Big|_{\theta=\theta^t}$ is the direction of the IGO update (14) at θ^t, one can define

$$\theta^{t+\delta t} = \exp_{\theta^t}(\delta t.Y)$$

where exp is the exponential map of the Riemannian manifold Θ equipped with the Fisher information metric.

This defines an approximation to the IGO flow that depends on the step size δt and sample size N, but *not* on any choice of parametrization.

Practical implementation will depend on being able to compute the geodesics of the Fisher metric. The equation of geodesics may be computed explicitly in some particular cases (Burbea, 1986), (Amari et al., 1987, Chapter 5), such as Bernoulli distributions or Gaussian distributions with a fixed mean or with a fixed covariance matrix. Interestingly, for Gaussian distributions with a fixed mean, the geodesic update resembles the one in xNES.

When no closed formula for geodesics is available, $\theta^{t+\delta t}$ can always be found by numerically integrating the geodesic equation starting at θ^t with initial speed Y. This is, of course, an added computational cost, but it does not require any calls to the objective function f.

Finite sample size and noisy IGO flow. The IGO flow is an ideal model of the IGO algorithms. But the IGO flow is deterministic while IGO algorithms are stochastic, depending on a finite number N of random samples. This might result in important differences in their behavior and one can wonder if there is a way to reflect stochasticity directly into the definition of the IGO flow.

The IGO update (14) is a stochastic update

$$\theta^{t+\delta t} = \theta^t + \delta t \sum_{i=1}^{N} \widehat{w}_i \ \widetilde{\nabla}_\theta \ln P_\theta(x_i)\Big|_{\theta=\theta^t}$$

because the term $\sum_{i=1}^{N} \widehat{w}_i \ \widetilde{\nabla}_\theta \ln P_\theta(x_i)\Big|_{\theta=\theta^t}$ involves a random sample. As such, this term has an expectation and a variance. So for a fixed N and δt, this random update is a weak approximation with step size δt (Kloeden and Platen, 1992, Chapter 9.7) of a stochastic differential equation on θ, whose drift is the expectation of the IGO update (which tends to the IGO flow when $N \to \infty$), and whose noise term is $\sqrt{\delta t}$ times the square root of the covariance matrix of the update applied to a normal random vector.

Such a stochastic differential equation, defining a *noisy IGO flow*, might be a better theoretical object with which to compare the actual behavior of IGO algorithms, than the ideal noiseless IGO flow.

For instance, this strongly suggests that if we have $\delta t \to 0$ while N is kept fixed in an IGO algorithm, noise will disappear (compare Remark 2 in (Akimoto et al., 2012)).

Second, for large N, one expects the variance of the IGO update to scale like $\frac{1}{N}$, so that the noise term will scale like $\sqrt{\delta t/N}$. This formally suggests that, within reasonable bounds, multiplying or dividing both N and δt by the same factor should result in similar behavior of the algorithm, so that for instance it should be reasonable to reset N to 10 and δt to $10\delta t/N$. (Note that the cost in terms of f-calls of these two algorithms is similar.)

This dependency is reflected in evolution strategies in several ways, with typical values for N ranging between ten and a few hundred. First, theoretical results on the function $f(x) = \|x\|$ indicate that the optimal step-size δt for the mean vector is proportional to N, provided the weighting function w reflects truncation selection with a fixed truncation ratio (Beyer, 2001) or optimal weights (Arnold, 2006). Second, the learning rate δt of the covariance matrix in CMA-ES is chosen proportional to $\left(\sum_{i=1}^{N} \widehat{w}_i\right)^2 / \sum_{i=1}^{N} \widehat{w}_i^2$ which is again proportional to N (Hansen and Kern, 2004). For small enough N, the progress per f-call is then in both cases rather independent of the choice of N.

Influence of the Fisher geometry of the statistical manifold. The global Riemannian geometry of the statistical manifold P_θ might have a bearing on the behavior of stochastic algorithms exploring this manifold. For instance, the Fisher metric identifies the set of 1-dimensional normal distributions $\mathcal{N}(m, \sigma^2)$ with the two-dimensional hyperbolic plane. The latter has negative curvature. The sign of curvature has a strong influence on the behavior of random walks in a Riemannian manifold: in particular, in negative curvature, successive random errors tend to not compensate as much as in the Euclidean case (because geodesics diverge more quickly); this might be relevant to the settings of a stochastic optimization algorithm, suggesting to use larger sample size (or smaller steps) when curvature is negative. This is speculative and remains to be explored.

Summary and conclusion

We sum up:

- The information-geometric optimization (IGO) framework derives from invariance principles and allows to build optimization algorithms from any family of distributions on any search space. In some instances (Gaussian distributions on $\mathbb{R}^d$ or Bernoulli distributions on $\{0,1\}^d$) it recovers versions of known algorithms (CMA-ES or PBIL); in other instances (restricted Boltzmann machine distributions) it produces new, hopefully efficient optimization algorithms.

- The use of a quantile-based, time-dependent transform of the objective function (Equation (3)) provides a rigorous derivation of rank-based update rules currently used in optimization algorithms.

- The IGO flow is singled out by its equivalent description as an infinitesimal weighted maximal log-likelihood update. In a particular parametrization and with a step size of 1, it recovers the cross-entropy method.

- Theoretical arguments suggest that the IGO flow minimizes the change of diversity in the course of optimization. In particular, starting with high diversity and using multimodal distributions may allow simultaneous exploration of multiple optima of the objective function. Preliminary experiments with restricted Boltzmann machines confirm this effect in a simple situation.

Thus, the IGO framework is an attempt to provide sound theoretical foundations to optimization algorithms based on probability distributions. In particular, this viewpoint helps to bridge the gap between continuous and discrete optimization.

The invariance properties, which reduce the number of arbitrary choices, together with the relationship between natural gradient and diversity, may contribute to a theoretical explanation of the good practical performance of those currently used algorithms, such as CMA-ES, which can be interpreted as instantiations of IGO.

We hope that invariance properties will acquire in computer science the importance they have in mathematics, where intrinsic thinking is the first step for abstract linear algebra or differential geometry, and in modern physics, where the notions of invariance w.r.t. the coordinate system and so-called gauge invariance play a central role.

Acknowledgements

The authors would like to thank Michèle Sebag for the acronym and for helpful comments. We also thank Youhei Akimoto for helpful feedback. Y. O. would like to thank Cédric Villani and Bruno Sévennec for helpful discussions on the

Fisher metric. A. A. and N. H. would like to acknowledge the Dagstuhl Seminar No 10361 on the Theory of Evolutionary Computation[6] for inspiring their work on natural gradients and beyond. This work was partially supported by the ANR-2010-COSI-002 grant (SIMINOLE) of the French National Research Agency.

Appendix: Proofs (removed)

References

D.H. Ackley, G.E. Hinton, and T.J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

Youhei Akimoto, Yuichi Nagata, Isao Ono, and Shigenobu Kobayashi. Bidirectional relation between CMA evolution strategies and natural evolution strategies. In *Proceedings of Parallel Problem Solving from Nature - PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 154–163. Springer, 2010.

Youhei Akimoto, Anne Auger, and Nikolaus Hansen. Convergence of the continuous time trajectories of isotropic evolution strategies on monotonic $\mathcal{C}^2$-composite functions. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *PPSN (1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 42–51. Springer, 2012.

S.-I. Amari, O. E. Barndorff-Nielsen, R. E. Kass, S. L. Lauritzen, and C. R. Rao. *Differential geometry in statistical inference.* Institute of Mathematical Statistics Lecture Notes—Monograph Series, 10. Institute of Mathematical Statistics, Hayward, CA, 1987.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Comput.*, 10:251–276, February 1998.

Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 2000. Translated from the 1993 Japanese original by Daishi Harada.

D.V. Arnold. Weighted multirecombination evolution strategies. *Theoretical computer science*, 361(1):18–37, 2006.

Shumeet Baluja. Population based incremental learning: A method for integrating genetic search based function optimization and competitve learning. Technical Report CMU-CS-94-163, Carnegie Mellon Report, 1994.

Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. In *Proceedings of ICML'95*, pages 38–46, 1995.

[6] http://www.dagstuhl.de/10361

Y. Bengio, P. Lamblin, V. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, Cambridge, MA, 2007.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

A. Berny. Selection and reinforcement learning for combinatorial optimization. In Marc Schoenauer, Kalyanmoy Deb, Günther Rudolph, Xin Yao, Evelyne Lutton, Juan Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 601–610. Springer Berlin Heidelberg, 2000a.

A. Berny. An adaptive scheme for real function optimization acting as a selection operator. In *Combinations of Evolutionary Computation and Neural Networks, 2000 IEEE Symposium on*, pages 140 –149, 2000b.

Arnaud Berny. Boltzmann machine for population-based incremental learning. In *ECAI*, pages 198–202, 2002.

H.-G. Beyer. *The Theory of Evolution Strategies.* Natural Computing Series. Springer-Verlag, 2001.

J. Branke, C. Lode, and J.L. Shapiro. Addressing sampling errors and diversity loss in umda. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 508–515. ACM, 2007.

Jacob Burbea. Informative geometry of probability spaces. *Exposition. Math.*, 4 (4):347–378, 1986.

Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, second edition, 2006.

Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals OR*, 134(1):19–67, 2005.

G. Desjardins, A. Courville, Y. Bengio, P. Vincent, and O. Dellaleau. Parallel tempering for training of restricted Boltzmann machines. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Marcus Gallagher and Marcus Frean. Population-based continuous optimization, probabilistic modelling and mean shift. *Evol. Comput.*, 13(1):29–42, January 2005.

Zoubin Ghahramani. Unsupervised learning. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 72–112. Springer Berlin / Heidelberg, 2004.

Tobias Glasmachers, Tom Schaul, Yi Sun, Daan Wierstra, and Jürgen Schmidhuber. Exponential natural evolution strategies. In *GECCO*, pages 393–400, 2010.

N. Hansen. The CMA evolution strategy: a comparing review. In J.A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In X. Yao et al., editors, *Parallel Problem Solving from Nature PPSN VIII*, volume 3242 of *LNCS*, pages 282–291. Springer, 2004.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

G.E. Hinton, S. Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Conputation*, 18:1527–1554, 2006.

R. Hooke and T.A. Jeeves. "direct search" solution of numerical and statistical problems. *Journal of the ACM*, 8:212–229, 1961.

G.A. Jastrebski and D.V. Arnold. Improving evolution strategies through active covariance matrix adaptation. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 2814–2821. IEEE, 2006.

Harold Jeffreys. An invariant form for the prior probability in estimation problems. *Proc. Roy. Soc. London. Ser. A.*, 186:453–461, 1946.

Peter E. Kloeden and Eckhard Platen. *Numerical solution of stochastic differential equations*, volume 23 of *Applications of Mathematics (New York)*. Springer-Verlag, Berlin, 1992.

Solomon Kullback. *Information theory and statistics*. Dover Publications Inc., Mineola, NY, 1997. Reprint of the second (1968) edition.

P. Larranaga and J.A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Netherlands, 2002.

Nicolas Le Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *NIPS*, 2007.

Luigi Malagò, Matteo Matteucci, and Bernardo Dal Seno. An information geometry perspective on estimation of distribution algorithms: boundary analysis. In *GECCO (Companion)*, pages 2081–2088, 2008.

Luigi Malagò, Matteo Matteucci, and Giovanni Pistone. Towards the geometry of estimation of distribution algorithms based on the exponential family. In Hans-Georg Beyer and William B. Langdon, editors, *FOGA, Proceedings*, pages 230–242. ACM, 2011.

John Ashworth Nelder and R Mead. A simplex method for function minimization. *The Computer Journal*, pages 308–313, 1965.

M. Pelikan, D.E. Goldberg, and F.G. Lobo. A survey of optimization by building and using probabilistic models. *Computational optimization and applications*, 21(1):5–20, 2002.

Calyampudi Radhakrishna Rao. Information and the accuracy attainable in the estimation of statistical parameters. *Bull. Calcutta Math. Soc.*, 37:81–91, 1945.

I. Rechenberg. *Evolutionsstrategie '94*. Frommann-Holzboog Verlag, 1994.

Laurent Schwartz. *Analyse. II*, volume 43 of *Collection Enseignement des Sciences [Collection: The Teaching of Science]*. Hermann, Paris, 1992. Calcul différentiel et équations différentielles, With the collaboration of K. Zizi.

H.-P. Schwefel. *Evolution and Optimum Seeking*. Sixth-generation computer technology series. John Wiley & Sons, Inc. New York, NY, USA, 1995.

F. Silva and L. Almeida. Acceleration techniques for the backpropagation algorithm. *Neural Networks*, pages 110–119, 1990.

P. Smolensky. Information processing in dynamical systems: foundations of harmony theory. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, MA, USA, 1986.

Yi Sun, Daan Wierstra, Tom Schaul, and Juergen Schmidhuber. Efficient natural evolution strategies. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 539–546, New York, NY, USA, 2009. ACM.

Virginia Torczon. On the convergence of pattern search algorithms. *SIAM Journal on optimization*, 7(1):1–25, 1997.

M. Toussaint. Notes on information geometry and evolutionary processes. eprint arXiv:nlin/0408040, 2004.

Michael Wagner, Anne Auger, and Marc Schoenauer. EEDA : A New Robust Estimation of Distribution Algorithms. Research Report RR-5190, INRIA, 2004.

D. Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the third international conference on Genetic algorithms*, pages 116–121, 1989.

Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. In *IEEE Congress on Evolutionary Computation*, pages 3381–3387, 2008.

9.3 Discussion

Although the results of this paper are given in the context of optimization with EDAs, they also give useful insights for training generative models such as RBMs with gradient descent in the ML setting.

First, the dependence of the vanilla gradient on parametrization (Question 9), is shown to affect the potential of an RBM to learn multimodal distributions. Indeed, if the vanilla gradient tends to loose diversity as it progresses towards two modes with a bimodal RBM, the problem is expected to become more severe when it needs to fit more than 10^{300} modes[1] as is usually the case in the ML setting.

As for the question of how to approach these issues (Question 10), a first possibility would be to use an approximation of the natural gradient (Le Roux et al., 2007; Desjardins et al., 2013) to recover some degree of symmetry. However, the exact natural gradient update is only truly invariant to re-parametrization in the limit of an infinitesimal learning rate. An interesting course of action is then to directly restore symmetry in the parametrization of the model, for instance using a centered energy function (Montavon and Müller, 2012) such as the one proposed in section 5:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_i a_i(x_i - \frac{1}{2}) - \sum_j b_j(h_j - \frac{1}{2}) - \sum_{i,j} w_{ij} a_i(x_i - \frac{1}{2})(h_j - \frac{1}{2})$$

Even if a centered energy does not have all the advantages of an exact natural gradient with an infinitesimal learning rate, this should allow for a better trajectory in parameter space with almost no computational overhead.

[1] This corresponds approximatively to the number of modes for an RBM with 1000 hidden units.

CONCLUSION AND PERSPECTIVES

The ML approach to AI allows to learn and generalize from data by posing a learning problem as an optimization one.

The objective is then to find a model which maximizes performance. However, as we consider tasks which require more and more intelligence, the corresponding models need more parameters which makes optimization increasingly difficult: as the number of parameter increases, so does the dimensionality of the search space. Aside from the necessity for additional computational power, optimization in higher dimensions usually requires large amounts of data which can be difficult or impossible to find in the supervised setting where labels are obtained with human intervention. A possible way to circumvent the issue may be to *learn representations* with unsupervised learning to leverage the large amounts of unlabeled data which are almost always available. The original learning problem is then solved in two steps: i/ learning a suitable representation and ii/ solving the final problem given this representation. That being said, there is still the matter of how to learn an interesting representation. This can be done with unlabeled data, which is an improvement, but more parameters may be needed since unsupervised learning will tend to take all aspects of the input distribution into account and not only those which may be needed for the final problem.

In this setting, deep learning may lead to a practical answer. First, deep architectures are capable of performing complex operations with much less parameters than shallow ones, which makes them easier to train. Second, the possibility of using a layer-wise training procedure decreases the computational cost further by making the optimization sequentially separable: instead of learning all the parameters of the model at once, the optimization is done one layer at a time.

Consistency of layer-wise training The current state-of-the-art justification for layer-wise training of deep generative models is based on the maximization of a variational lower bound. This approach leads to a guarantee that the likelihood improves when a layer is added on top of a shallow model, but the guarantee does not hold for more than two layers. Additionally, optimizing the variational lower bound is not consistent as it does not lead to the same optimum as the optimization of all layers at once. In this thesis, we propose a new criterion for training deep generative models in a layer-wise fashion: the BLM upper-bound. We prove that maximizing this criterion at each step leads to an optimal deep generative model, provided the upper layers can be trained successfully. The BLM upper-bound corresponds to the maximum log-likelihood attainable by adding layers. This results in a new paradigm for layer-wise deep learning: maximizing the potential log-likelihood of a deep generative model even as upper-layers are not yet specified. Although the BLM is intractable in general, we provide several approximations. The BLM upper-bound also has close ties with encoder-decoder models. Namely, the training procedure of auto-encoders with the reconstruction error corresponds to an approximation of the BLM upper-bound. This leads to a new justification for stacked auto-associators as an approximate method for training the lower part of a deep generative model and suggests that the encoder part of the model should be as rich as possible. This is confirmed by our experiments in which Auto-Encoders with Rich Inference (AERIes) are shown to outperform the current state-of-the-art variational approach on two distinct datasets.

Tractability of performance evaluation Having a tractable measure of performance is critical for model selection where several models must be compared to choose the fittest. Current approaches based on the evaluation w.r.t. a supervised task or based on the computation of the likelihood on a validation set typically incur a high computational cost. In this thesis, we propose to use the BLM upper-bound to evaluate the potential performance of lower layers before the upper part of the model has been trained. In the context of learning representations, the BLM upper bound gives an accurate measure of the quality of a representation even when a model is incomplete. We show in our experiments that the BLM upper-bound is a good estimator of the final likelihood and we describe a sound procedure for model selection: select each layer in turn according to the BLM upper-bound, and select the final generative layer according to the log-likelihood.

Efficiently training layers Training a single layer can be a difficult optimization problem. Current approaches often rely on a gradient descent with the Euclidean metric to perform optimization in parameter space. When the goal is to estimate a distribution, our experiments show that choosing the Euclidean metric introduces a spurious dependence on parametrization which can lead to a breach of symmetry and a difficulty to account for multimodal distributions. In this thesis, we show the importance of considering metrics in optimization and suggest that the natural gradient or a centered parametrization can be used to improve the trajectory of a gradient descent procedure.

These contributions lead to several new avenues of research.

Empirical validation of the BLM on larger models The properties of the BLM could only be tested on relatively small models due to the intractability of the likelihood. Recent advances in MCMC sampling may allow the computation of the likelihood of larger models in a reasonable time. This would allow a comparison of the BLM upper-bound and the traditional maximization of the variational upper bound on deeper datasets where the BLM is predicted to have a greater advantage.

Maximizing the BLM upper bound for richer models Several models such as spike and slab RBMs and deep Boltzmann machines have been shown to perform better than RBMs and auto-associators with the help of a variational layer-wise pre-training. Further improvements may result from maximizing the BLM upper bound with these richer models.

Deep learning without neural networks Although several attempts have been made to apply the deep learning principles to other models, most successful attempts concern neural networks. In this thesis we introduce a general form for deep generative models which is not restricted to neural networks and may apply to other classes of models.

Richer inference Using richer inference in the encoder part of an encoder-decoder has shown to be a good way to increase performance when learning deep neural networks. This principle is widely applicable and may lead to increasing the performance of other models.

Layer-wise simplification The success of deep learning methods suggests that auto-associators and RBMs are able to simplify a problem for upper layers. However, there is no theoretical justification for these simplification properties. Understanding how a distribution can be simplified may lead to a better understanding of what constitutes a good representation for deep learning.

optimization metrics This thesis highlights the importance of choosing a good metric to perform optimization. Although in several applications, the exact natural gradient itself may have a prohibitive computational cost, better metrics may be found by changing parametrization or by approximating the natural gradient.

PUBLICATIONS

[1] Ludovic Arnold, Hélène Paugam-Moisy, and Michèle Sebag. Optimisation de la topologie pour les réseaux de neurones profonds. In *17e congrès francophone AFRIF–AFIA Reconnaissance des Formes et Intelli- gence Artificielle (RFIA 2010)*, Caen France, Jan 2010.

[2] Ludovic Arnold, Hélène Paugam-Moisy, and Michèle Sebag. Unsupervised layer-wise model selection in deep neural networks. In *19th European Conference on Artificial Intelligence (ECAI 2010)*, Lisbon Portugal, Aug 2010, 915–920.

[3] Ludovic Arnold, Sebastien Rebecchi, Sylvain Chevallier, and Hélène Paugam-Moisy. An introduction to deep learning. In European Symposium on Artificial Neural Networks (ESANN 2011), 2011.

[4] Ludovic Arnold, Anne Auger, Nikolaus Hansen, and Yann Ollivier. Information-geometric optimization algorithms: A unifying picture via invariance principles. Technical report, ArXiv e-prints, June 2011. URL http://arxiv.org/abs/1106.3708.

[5] Ludovic Arnold and Yann Ollivier. Layer-wise learning of deep generative models. Technical report, ArXiv e-prints, December 2012. URL http://arxiv.org/abs/1212.1524.

BIBLIOGRAPHY

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985. (Cited on pages 72 and 73.)

Ryan Prescott Adams, Hanna M. Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. *Journal of Machine Learning Research - Proceedings Track*, 9:1–8, 2010. (Cited on page 87.)

Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data generating distribution. *ArXiv e-prints*, November 2012. (Cited on page 86.)

Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998. (Cited on pages 57 and 173.)

Shun-ichi Amari, Hyeyoung Park, and Kenji Fukumizu. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12:1399–1409, June 2000. (Cited on page 57.)

Shummet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, Pittsburgh, PA, USA, 1994. (Cited on page 19.)

Robert M. Bell, Yehuda Koren, and Chris Volinsky. The bellkor solution to the netflix prize. Technical report, AT&T Labs – Research, 2007. (Cited on page 90.)

Yoshua Bengio. Learning deep architectures for ai. Technical report, Université de Montréal, Dept. IRO, 2007. (Cited on pages 80 and 85.)

Yoshua Bengio. Deep learning of representations: Looking forward. *ArXiv e-prints*, May 2013. (Cited on page 79.)

Yoshua Bengio and Olivier Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6):1601–1621, 2009. (Cited on pages 76, 86, and 121.)

Yoshua Bengio and Xavier Glorot. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS 2010*, volume 9, pages 249–256, 2010. (Cited on pages 80, 81, and 121.)

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards ai. In *Large-Scale Kernel Machines*. MIT Press, 2007. (Cited on pages 79, 80, and 91.)

Yoshua Bengio and Éric. Thibodeau-Laufer. Deep generative stochastic networks trainable by backprop. *ArXiv e-prints*, June 2013. (Cited on page 95.)

Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. In *Advances in Neural Information Processing Systems 18*, volume 18, 2006. (Cited on pages 79 and 80.)

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, Cambridge, MA, 2007. (Cited on pages 79, 80, 83, 87, and 121.)

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012. (Cited on page 79.)

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. *ArXiv e-prints*, May 2013. (Cited on page 86.)

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. (Cited on pages 16 and 96.)

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 23*, 2011. (Cited on page 96.)

Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995. (Cited on page 63.)

Sean Borman. The expectation maximization algorithm: A short tutorial. 2004. (Cited on page 53.)

Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294, 1988. (Cited on page 71.)

Olivier Breuleux, Yoshua Bengio, and Pascal Vincent. Quickly generating representative samples from an rbm-derived process. *Neural Computation*, 23(8): 2053–2073, August 2011. (Cited on page 87.)

Miguel A. Carreira-Perpiñán and Geoffrey E. Hinton. On contrastive divergence learning. In *Artificial Intelligence and Statistics*, 2005. (Cited on page 76.)

Kyunghyun Cho, Tapani Raiko, Alexander Ilin, and Juha Karhunen. A two-stage pretraining algorithm for deep boltzmann machines. In *Proceedings of the NIPS 2012 Workshop on Deep Learning and Unsupervised Feature Learning*, 2012. (Cited on page 90.)

Dan C. Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *NIPS*, pages 2852–2860, 2012a. (Cited on page 91.)

Dan C. Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32: 333–338, 2012b. (Cited on page 91.)

Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition, 2010. (Cited on pages 80, 92, and 121.)

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. (Cited on pages 91 and 92.)

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008. (Cited on page 82.)

Aarron Courville, James Bergstra, and Yoshua Bengio. Unsupervised models of images by spike-and-slab rbms. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1145–1152, New York, NY, USA, June 2011a. ACM. (Cited on page 87.)

Aarron Courville, James Bergstra, and Yoshua Bengio. The spike and slab restricted boltzmann machine. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 233–241, 2011b. (Cited on page 87.)

Richard Threlkeld Cox. Probability, frequency and reasonable expectation. *American Journal of Physics*, 14:1–13, 1946. (Cited on page 39.)

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314–314, December 1989. (Cited on page 69.)

George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech & Language Processing*, 20(1):30–42, 2012. (Cited on page 91.)

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39:1–38, 1977. (Cited on page 53.)

Li Deng, Michael L. Seltzer, Dong Yu, Alex Acero, Abdel rahman Mohamed, and Geoffrey E. Hinton. Binary coding of speech spectrograms using a deep auto-encoder. In *INTERSPEECH*, pages 1692–1695, 2010. (Cited on page 87.)

Guillaume Desjardins, Aaron Courville, Yoshua Bengio, Pascal Vincent, and Olivier Dellaleau. Parallel tempering for training of restricted boltzmann machines. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010. (Cited on page 87.)

Guillaume Desjardins, Aaron Courville, and Yoshua Bengio. On tracking the partition function. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2501–2509, 2011. (Cited on page 88.)

Guillaume Desjardins, Razvan Pascanu, Aaron Courville, and Yoshua Bengio. Metric-free natural gradient for joint-training of boltzmann machines. *CoRR*, abs/1301.3545, 2013. (Cited on page 217.)

Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009. (Cited on pages 83 and 84.)

Brendan J. Frey. Continuous sigmoidal belief networks trained using slice sampling. In *NIPS*, pages 452–458, 1996. (Cited on page 87.)

Brendan J. Frey and Geoffrey E. Hinton. Variational learning in nonlinear gaussian belief networks. *Neural Computation*, 11(1):193–213, 1999. (Cited on page 87.)

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. (Cited on page 81.)

Zoubin Ghahramani. Unsupervised learning. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 72–112. Springer Berlin / Heidelberg, 2004. (Cited on page 52.)

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In David Dunson Geoffrey Gordon and eds Miroslav DudÃk, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15 (draft) of *W&CP*. JMLR, 2010. (Cited on page 87.)

Ian J. Goodfellow, Quoc Le, Andrew Saxe, Honglak Lee, and Andrew Ng. Measuring invariances in deep networks. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 646–654. MIT Press, 2009. (Cited on page 80.)

Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio. Spike-and-slab sparse coding for unsupervised feature discovery. *CoRR*, abs/1201.3382, 2012. (Cited on page 87.)

Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio. Joint training of deep boltzmann machines for classification. *ArXiv e-prints*, January 2013. (Cited on page 90.)

Alex Graves. Offline arabic handwriting recognition with multidimensional recurrent neural networks. In Volker MÃ€rgner and Haikal El Abed, editors, *Guide to OCR for Arabic Scripts*, pages 297–313. Springer London, 2012. (Cited on page 90.)

Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, pages 545–552, 2008. (Cited on page 90.)

Maya R. Gupta and Yihua Chen. Theory and use of the em algorithm. *Found. Trends Signal Process.*, 4:223–296, 2011. (Cited on page 53.)

Nikolaus Hansen. The CMA evolution strategy: A tutorial. Technical report, Université Paris Sud, 2008. (Cited on page 19.)

Suzana Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, 3(00031):31, 2009. (Cited on page 63.)

Geoffrey E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989. (Cited on page 71.)

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002. (Cited on page 76.)

Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. Technical report, Department of Computer Science, University of Toronto, 2010. (Cited on page 76.)

Geoffrey E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006. (Cited on page 87.)

Geoffrey E. Hinton, S. Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Conputation*, 18:1527–1554, 2006. (Cited on pages 83, 84, and 121.)

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. (Cited on pages 81, 91, 92, and 121.)

Alan L. Hodgkin and Andrew F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, August 1952. (Cited on page 63.)

Kurt Hornik, Maxwell Stinchcombe, and Halber White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, 1989. ISSN 0893-6080. (Cited on page 69.)

Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *J. Mach. Learn. Res.*, 6:695–709, 2005. (Cited on page 87.)

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*. IEEE, 2009. (Cited on page 82.)

Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. Fast inference in sparse coding algorithms with applications to object recognition. *CoRR*, abs/1010.3467, 2010a. (Cited on page 171.)

Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Michael Mathieu, and Yann Le Cun. Learning convolutional feature hierarchies for visual recognition. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1090–1098, 2010b. (Cited on page 82.)

Scott Kirkpatrick, C. Daniel Gelatt Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983. (Cited on page 73.)

Alex Krizhevsky. Convolutional deep belief networks on cifar-10. Technical report, Department of Computer Science, University of Toronto, 2010. (Cited on pages 84 and 87.)

Alex Krizhevsky and Geoffrey E. Hinton. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, 2009. (Cited on page 74.)

Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 536–543, New York, NY, USA, 2008. ACM. (Cited on page 87.)

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 473–480, New York, NY, USA, 2007. ACM. (Cited on page 86.)

Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40, 2009. (Cited on pages 83 and 86.)

Hugo Larochelle, Michael Mandel, Razvan Pascanu, and Yoshua Bengio. Learning algorithms for the classification restricted boltzmann machine. *J. Mach. Learn. Res.*, 13:643–669, 2012. (Cited on page 87.)

Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012. (Cited on page 91.)

Yann Le Le Cun, Bernhard Boser, John S. Denker, Richard E. Howard, Wayne E. Habbard, Lawrence D. Jackel, and Donnie Henderson. Handwritten digit recognition with a back-propagation network. In David S. Touretzky, editor, *Advances in neural information processing systems 2*, pages 396–404, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc. (Cited on pages 81 and 82.)

Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural Computation*, 20:1631–1649, June 2008. (Cited on pages 74 and 122.)

Nicolas Le Roux and Andrew W. Fitzgibbon. A fast natural newton method. In *ICML*, pages 623–630, 2010. (Cited on page 173.)

Nicolas Le Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Top-moumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems 20*. MIT Press, 2007. (Cited on page 217.)

Yann LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier. (Cited on page 82.)

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995. (Cited on pages 82 and 121.)

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, November 1998a. (Cited on page 81.)

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*. Springer Berlin / Heidelberg, 1998b. (Cited on page 69.)

Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing Systems 20*. MIT Press, 2007. (Cited on page 80.)

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML '09: Proceedings of the 26th international conference on Machine learning*, page 77, 2009a. (Cited on pages 80 and 84.)

Honglak Lee, Yan Largman, Peter Pham, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems 22*, pages 1096–1104. MIT Press, 2009b. (Cited on page 84.)

Benjamin M. Marlin, Kevin Swersky, Bo Chen, and Nando de Freitas. Inductive principles for restricted boltzmann machine learning. *Journal of Machine Learning Research - Proceedings Track*, 9:509–516, 2010. (Cited on page 87.)

James Martens. Deep learning via hessian-free optimization. In Johannes Fürnkranz and Thorsten Joachims, editors, *ICML '10: Proceedings of the 27th Annual International Conference on Machine Learning*, pages 735–742, Haifa, Israel, June 2010. Omnipress. (Cited on pages 14 and 173.)

James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In Lise Getoor and Tobias Scheffer, editors, *ICML '11: Proceedings of the 28th Annual International Conference on Machine Learning*, ICML '11, pages 1033–1040, New York, NY, USA, June 2011. ACM. (Cited on pages 14 and 173.)

Ueli Meier, Dan Claudiu Ciresan, Luca Maria Gambardella, and Jürgen Schmidhuber. Better digit recognition with a committee of simple neural nets. In *ICDAR*, pages 1250–1254, 2011. (Cited on page 91.)

Roland Memisevic. *Non-linear latent factor models for revealing structure in high-dimensional data*. PhD thesis, University of Toronto, 2008. (Cited on page 89.)

Roland Memisevic and Geoffrey Hinton. Unsupervised learning of image transformations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007. (Cited on page 89.)

Roland Memisevic and Geoffrey E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, June 2010. (Cited on page 89.)

Grégoire Mesnil, Yann Dauphin, Xavier Glorot, Salah Rifai, Yoshua Bengio, Ian Goodfellow, Erick Lavoie, Xavier Muller, Guillaume Desjardins, David Warde-Farley, Pascal Vincent, Aaron Courville, and James Bergstra. Unsupervised and transfer learning challenge: a deep learning approach. In Isabelle Guyon, G. Dror, V Lemaire, G. Taylor, and D. Silver, editors, *JMLR W&CP: Proceedings of the Unsupervised and Transfer Learning challenge and workshop*, volume 27, pages 97–110, 2012. (Cited on page 90.)

Marvin L. Minsky and Seymour Papert. *Perceptrons: An introduction to computational geometry*. MIT press Cambridge, Mass, 1969. (Cited on page 68.)

Grégoire Montavon and Klaus-Robert Müller. *Deep Boltzmann Machines and the Centering Trick*, volume 7700 of *LNCS*. Springer, 2nd edn edition, 2012. (Cited on page 217.)

Iain Murray and Ruslan Salakhutdinov. Evaluating probabilities under high-dimensional latent variable models. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21, 2009. (Cited on pages 88 and 172.)

Vinod Nair and Geoffrey E. Hinton. 3d object recognition with deep belief nets. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1339–1347. MIT Press, 2009. (Cited on page 84.)

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML '10: Proceedings of the 27th international conference on Machine learning*, pages 807–814, 2010. (Cited on page 87.)

Radford M. Neal. Probabilistic inference using markov chain monte carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993. (Cited on page 41.)

Radford M. Neal. Annealed importance sampling. Technical report, University of Toronto, Department of Statistics, 1998. (Cited on page 88.)

Andrew Ng. Sparse autoencoder. CS294A Lecture notes, 2011. (Cited on page 72.)

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal deep learning. In *ICML*, pages 689–696, 2011. (Cited on page 87.)

Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, 2006. (Cited on page 14.)

Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996. (Cited on page 37.)

Bruno A. Olshausen and David J. Field. Sparse coding with an overcomplete basis set: a strategy employed by v1? *Vision research*, 37:3311–3325, 1997. (Cited on page 37.)

Abdel rahman Mohamed, Tara N. Sainath, George E. Dahl, Bhuvana Ramabhadran, Geoffrey E. Hinton, and Michael A. Picheny. Deep belief networks using discriminative features for phone recognition. In *ICASSP*, pages 5060–5063, 2011. (Cited on page 84.)

Marc'Aurelio Ranzato, Alex Krizhevsky, and Geoffrey E. Hinton. Factored 3-way restricted boltzmann machines for modeling natural images. *Journal of Machine Learning Research - Proceedings Track*, 9:621–628, 2010. (Cited on page 89.)

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, pages 833–840, 2011. (Cited on pages 72 and 89.)

Salah Rifai, Yoshua Bengio, Yann Dauphin, and Pascal Vincent. A generative process for sampling contractive auto-encoders. In *International Conference on Machine Learning, ICML'12*, 06 2012. (Cited on page 89.)

Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. (Cited on page 41.)

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. (Cited on pages 69 and 81.)

Ruslan Salakhutdinov. Learning and evaluating Boltzmann machines. Technical Report UTML TR 2008-002, Department of Computer Science, University of Toronto, June 2008. (Cited on page 88.)

Ruslan Salakhutdinov. Learning in markov random fields using tempered transitions. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1598–1606. MIT Press, 2009. (Cited on page 87.)

Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5, pages 448–455, 2009a. (Cited on pages 90 and 121.)

Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009b. (Cited on page 87.)

Ruslan Salakhutdinov and Geoffrey E. Hinton. A better way to pretrain deep boltzmann machines. In *NIPS*, pages 2456–2464, 2012. (Cited on page 90.)

Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 872–879, New York, NY, USA, 2008. ACM. (Cited on page 88.)

Tanya Schmah, Geoffrey E. Hinton, Richard S. Zemel, Steven L. Small, and Stephen C. Strother. Generative versus discriminative training of rbms for classification of fmri images. In *NIPS*, pages 1409–1416, 2008. (Cited on page 87.)

Terrence J. Sejnowski. Higher-order boltzmann machines. In J. Denker, editor, *Neural Networks for Computing*, pages 398–403. American Institute of Physics, 1986. (Cited on page 89.)

Paul Smolensky. Information processing in dynamical systems: foundations of harmony theory. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, MA, USA, 1986. (Cited on page 73.)

Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep boltzmann machines. In *NIPS*, pages 2231–2239, 2012. (Cited on page 90.)

Ilya Sutskever and Geoffrey E. Hinton. Deep, narrow sigmoid belief networks are universal approximators. *Neural Comput.*, 20:2629–2636, November 2008. (Cited on page 80.)

Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1017–1024, New York, NY, USA, June 2011. ACM. (Cited on pages 14 and 173.)

Kevin Swersky, Marc'Aurelio Ranzato, David Buchman, Benjamin Marlin, and Nando Freitas. On autoencoders and score matching for energy based models. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1201–1208, New York, NY, USA, 2011. ACM. (Cited on pages 87 and 89.)

Graham W. Taylor and Geoffrey E. Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1025–1032, New York, NY, USA, 2009. ACM. (Cited on page 84.)

Graham W. Taylor, Geoffrey E. Hinton, and Sam T. Roweis. Modeling human motion using binary latent variables. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1345–1352. MIT Press, Cambridge, MA, 2007. (Cited on page 84.)

Lucas Theis, Sebastian Gerwinn, Fabian Sinz, and Matthias Bethge. In all likelihood, deep belief is not enough. *Journal of Machine Learning Research*, 12:3071–3096, Nov 2011. (Cited on page 92.)

Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1064–1071, New York, NY, USA, 2008. ACM. (Cited on page 87.)

Tijmen Tieleman and Geoffrey E. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1033–1040, New York, NY, USA, 2009. ACM. (Cited on page 87.)

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, July 2011. (Cited on page 89.)

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. (Cited on pages 72, 86, 88, and 121.)

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010. (Cited on page 86.)

David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67 –82, apr 1997. (Cited on page 16.)

C. F. Jeff Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11:95–103, 1983. (Cited on page 53.)

Colophon

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

`http://code.google.com/p/classicthesis/`

Final Version as of July 6, 2013 (`classicthesis` version 1.0).