Study on training methods and generalization performance of deep learning for image classification

Michaël Blot

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Michaël Blot

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Sujet de la thèse :

Etude de l'apprentissage et de la généralisation des réseaux profonds en classification d'images

Study on training methods and generalization performance of deep learning for image classification

soutenue le XX/XX/XX

devant le jury composé de :

M. Alain Rakotomamonjy	Rapporteur
M. Christian Wolf	Rapporteur
Mme. Elisa Fromont	Examinatrice
M. Aurélien Bellet	Examinateur
M. Liva Ralaivola	Examinateur
M. Nicolas Thome	Examinateur
Mme Véronique Serfaty	Invitée
M. Matthieu Cord	Directeur de thèse

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Contents

1.1 Context

Computer vision has for a long time a great interaction with artificial intelligence. Today, some of the more complex tasks in image processing, such as facial recognition, object tracking, action detection, image captioning or semantic segmentation are being automatized by computers. Hence, using algorithms to retrieve semantic information from image content is a promising mechanism for many industries and computer vision applications such as medical imaging, self driving cars, astronomy, video surveillance, and so on.

From a broader perspective, general artificial intelligence is sometimes referenced as the next technology revolution and its ubiquity in our society seems inevitable in the near future. A particular branch of artificial intelligence that is under the spotlight is machine learning. This domain of computer science aims at automating the treatment of information using algorithms that have the ability to "learn" from data. In other words, a machine learning model uses statistical techniques to adapt an algorithm according to available data in order to respond correctly to a given task. Big companies like the GAFA (Google, Amazon, Facebook, Apple), are making unprecedented investments in research in this area. In addition, the number of start-ups that market machine learning innovations is exploding. Such enthusiasm has risen recently due to impressive performance of machine learning algorithms in several applications. Sometimes the algorithms even beat humans in very skillful tasks like playing Atari video games [Mni+13], playing Go or chess [Sil+16; DNW17], recognizing handwritten digits [LeC+98], recognizing faces [BRM16], etc. Those popular achievements are not going to stay isolated in a domain that progresses every day, with a potential that seems to be limitless.

Two principal factors can plead for those major advances in artificial intelligence. First, machine learning is a statistical discipline that is data driven; and therefore it benefits from

diversified, organized, and clean data. The current big data context is a perfect opportunity to explore machine learning capability and to extend its applications. The second factor is the substantial progresse made in the the field of statistical learning during the last two decades and specifically with the emergence of the promising "deep learning". Deep learning, which is a subset of machine learning models, will be our focus in this thesis.

1.1.1 Big Data and Supervised Learning

Among the reasons for such devotion around machine learning is the soaring volume of available data. Big data is a new paradigm in computer science which has emerged because of increased storage capacity and the facility to collect information from a super connected world. Gathering massive amounts of data is way easier today than it was decades ago. For instance, big databases of images (Flickr, Instagram, Facebook,...), of texts in different languages (Wikipedia), of videos (Youtube), or even of source-codes (Github) are today accessible to everyone. Moreover, many sources of data also provide side information like meta-data, contextual information, user information, commentary, etc. This additional information has proved to be very useful to create "training" data for machine learning models. Indeed, those rich databases have been cleaned and organized to construct task specific sets of pertinent data with multiple annotations. Those big datasets containing annotations constitute essential tools to train and apply machine learning models efficiently. In fact, when they contain semantic information, those annotations assist machine learning models by guiding them to fill what is called, the **semantic gap**. It is the difference between raw data and the semantic information it contains, which is interpretable by humans. For instance, an image, seen as a matrix of pixels, does not inform about the nature of the object it represents and it cannot be discussed. When the studied task requires to extract abstract semantic information, the gap has to be reduces by the algorithms. Even if most sources of data provide poor semantic information about the data, many efforts were done to create big datasets with task specific annotations (companies are even dedicated to this activity), enabling scientists and researchers to work on abundant, informative, and clean data.

Supervised learning. Above all, supervised learning, which is the training method studied in the thesis, is a particular machine learning paradigm that strongly benefits from increasing annotated datasets. Indeed, those annotated databases permit to construct **training sets** where all raw data is associated to a **label**. When given a piece of data as input, the associated label is the expected/targeted answer for a machine learning algorithm. For instance, in order to translate a sentence, the label could be the same text in another language. In order to translate music into a genre, the labels could be the style of the songs, if you are interested in music recognition. In order to recognize speech, the label could be the text corresponding to the sound. For a given raw data input, the machine learning algorithm outputs an answer due to respond to the task for which the database has been labelled. This output is called **prediction** because it is computed without being given any information about the expected answer. The objective of supervised training is then to obtain an algorithm whose predictions on the training set match the labels. This method ensures a good behavior of the predictive algorithm, at least for the training

data. Indeed, the training phase enforces the predictive model to return output values close to the labels when given data from the training set. For an input, noted X, from input space noted $\mathcal{X}$, the machine learning algorithm outputs a prediction noted $\hat{C}(X)$. $\hat{C}$ is a function of the space $\mathcal{X}$ to the space $\mathcal{C}$, sometimes called the **hypothesis**. In order to quantify the difference between the target, noted C and the output of the algorithm $\hat{C}(X)$ for a given input X, an error function or loss function, noted ℓ, must be defined. If the target space $\mathcal{C}$ is continuous, the loss function can be a distance between both points like the squared error: $\ell(C, \hat{C}(X)) = \|C - \hat{C}(X)\|_2^2$. If $\mathcal{C}$ is discreet, it can be the simple zero-one loss: $\ell(C, \hat{C}(X)) = \mathbb{1}(C \neq \hat{C}(X))$. In our formalism, the couple (C, X) is a random variable with unknown joint probability p. $\hat{C}(X)$ is naturally a random variable as well, with a probability depending on $p(X)$ and on the hypothesis $\hat{C}$. A training set $S^N = (x^{(n)}, c^{(n)})_{1 \leq n \leq N}$ is composed of N samples of couples (input label), draw according to p, and supposed independent. The points $x^{(n)}$ are realizations of X and the points $c^{(n)}$ are the labels, which are realizations of C.

A typical supervised training method that is studied in this thesis, is the Risk Expectancy Minimization principle (REM) which consists in minimizing the average error found on the training set:

$$\hat{\varepsilon rr}_N = \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{C}(x^{(n)}), c^{(n)}). \tag{1.1}$$

It is the average error value made on the dataset. $\hat{\varepsilon rr}_N$ is in fact, a proxy to estimate the expected value of the error function $\varepsilon rr = \mathbb{E}_p[\ell(\hat{c}(X), C)]$. For instance if the error function is the zero-one loss with discreet target space, $\hat{\varepsilon rr}_N$ is the proportion of bad prediction made on the training set. In this case, the training **accuracy** is obtained with $1 - \hat{\varepsilon rr}_N$. In this configuration, finding a hypothesis $\hat{C}$ that minimizes the training error $\hat{\varepsilon rr}_N$ can be $\mathcal{NP}$-hard. When the target space is big, the minimization strategy usually resorts to a surrogate function that is continuous, enabling standard optimization methods such as gradient descent [LY08]. Then, a machine learning model is properly trained if its average error function is low on the training data. However, this achievement does not provide any guarantees that the model will continue to behave correctly on new data.

Generalization. The generalization gap measures the difference between the expected error εrr of the algorithm, according to the natural data probability p, and the empirical average error $\hat{\varepsilon rr}_N$, computed from the training set: $\varepsilon rr - \hat{\varepsilon rr}_N$. Recall that, a task is correctly realized by an algorithm if it has a low expected error which is the sum of the empirical error with the generalization gap: $\varepsilon rr = \hat{\varepsilon rr}_N + (\varepsilon rr - \hat{\varepsilon rr}_N)$.

Practically, it is not possible to compute εrr because the natural distribution p is unknown. It is instead estimated by Monte Carlo sampling, from a side set called **test set**, that is not used during training. Because models are selected by REM to have small training errors $\hat{\varepsilon rr}_N$, it is important to control the generalization gap, which is usually positive. Statistical learning is interested at minimizing this gap, which is the challenge of generalization theory [Hau90; Val13; NY72].

A first requirement to generalize correctly is to provide a training set that is representative enough of the natural distribution of the data. When the input variable X lives in a

large space with high dimensionality, a lot of samples are required to properly embody the true distribution. This flaw is called the **curse of dimensionality**. Computer vision applications suffer from this inconvenience and former state of the art machine learning models had to consider a featuring process in order to reduce the dimensionality of the input data.

Standard supervised learning methods. Among popular machine learning methods, the Support Vector Machine (SVM) [Vap98] is a suitable model for supervised learning. It trains a linear predictor to maximize a margin criterion. Not only the criterion indirectly permits to reduce the training error, but it is also due to find a hypothesis that generalizes well, according to theoretical generalization bounds [Vap98; NY72]. In order to enable a non linear predictor, the SVM can be coupled with kernel methods [Sch01].

The overall method is usually associated to a featuring processing in order to reduce the dimension of the input variable and avoid the curse of dimensionality. This process applies different transformations to the raw data with the objective to extract the information about the target in a variable of lower dimension. If the feature variable size is still high, the more informative features can then be elected to feed the predictive algorithm via feature selection techniques [GE03].

Other generic methods such as boosting [Sch99; CG16] or bagging [Büh04] can be associated to the training to enhance the model performances by reducing its generalization gap. Both techniques combine the predictions of several trained algorithms by aggregating their outputs and producing a global prediction supposed to be more accurate. These methods have been applied with success on many applications.

Computer vision and image classification. Frequently, supervised learning brings the best results because it directly relates the algorithm behavior with the task for which it is trained. Unfortunately, finding big labelled datasets can be difficult because annotating big databases is not automatized easily and is thus expensive. In the field of computer vision, however, several databases have been created associating a particular type of label to the images in the dataset with the ambition to enable supervised learning applications. The nature of the information provided in different datasets is miscellaneous. For instance it can be pixel segmentation of the different items on the images [Eve+15; Lin+15], some textual descriptions of the images [Ant+15], recipes associated to pictures of meals [Wan+15], etc. The attention of this thesis will be image classification, according to their content (object, scene,...) onto different predefined classes. In this context, the input variable is an image and the label is the class of the image. For this task, existing datasets [Kri09; Lec+98; Net+11; Lin+15; JDe+09] facilitate the study and the comparison of different machine learning models. For instance, one of the most popular datasets in image classification is ImageNet [JDe+09] which is an evaluation reference for large scale models. For this dataset there are one thousand predefined classes (airplane, dog, car, truck, ...). Every image is a picture of an object or animal and the goal is to design an algorithm that predicts to which class it belongs. Example of images from ImageNet can be found on Figure 1.1.

The task of image classification attracts our attention for various reasons. A principal one is the quantity and the quality of the existing datasets. Unlike many other computed vision tasks, image classification rarely has to cope with small, noisy and biased datasets.

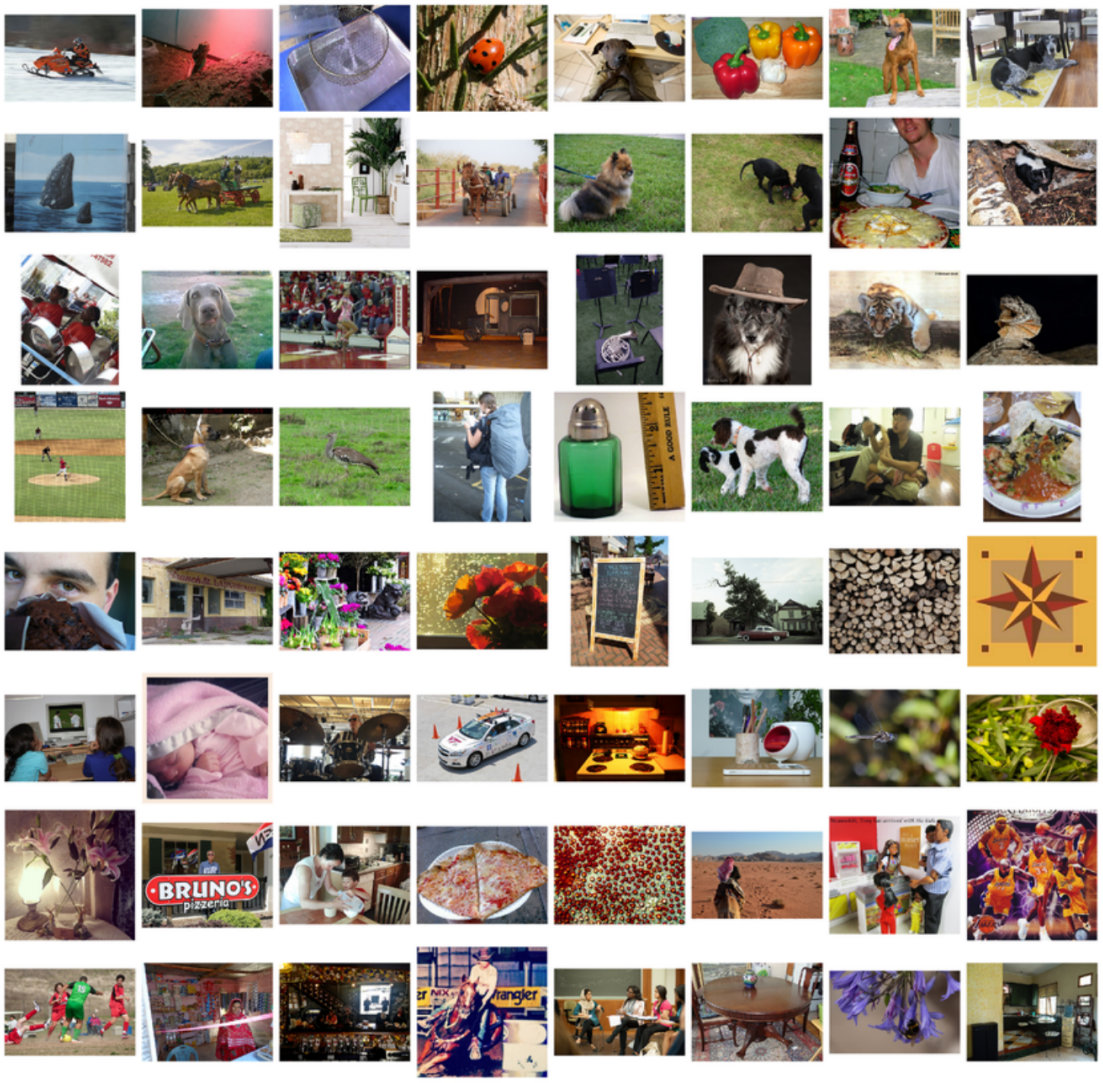


Figure 1.1.: Example of image from ImageNet dataset. They are considered as high resolution pictures with a size of approximately 256×256 pixels with RGB channels. The dataset is composed of 1.3M training images and 50K testing images equally spread into 1000 classes.

Requiring cheap annotations, supervised learning for image classification benefits from clean and bulky datasets. This advantage facilitates the focus on the machine learning models and their properties. This is not by any chance that it is on this particular exercise that a major breakthrough has favored the exhibition of deep learning potential. Indeed, for a while, state of the art algorithms for image classification were based on bag of words (BoW) models [BS97; Siv+05; FCP01]. Those algorithms build a "visual dictionary" from local image descriptors. For any image, local detection values of each "word" are pooled together to represent the final image feature, which is used as input for a SVM classifier. Today, those algorithms have been replaced by deep learning models, that are introduced in next section.

1.1.2 Deep learning for computer vision

Simultaneously with the evolution of big data, favouring supervised learning application, the spectacular progress in machine learning has raised this exercise to the status of theory [Vap98]. While generalization theory was bringing analytical guidances to design machine learning models [NY72; BLM02; Vap98], other tools borrowed from other mathematical fields such as signal processing, information theory, algebra, optimization, differential analysis, etc, where bringing their contributions. Many scientists, from many different domains are getting interested and are sharing their knowledge to the sake of machine learning. In the sphere of computer vision, a reviving family of models, called today **deep learning**, has empowered astonishing improvement on several challenging benchmarks, specifically on image classification. Deep learning defines the machine learning methods that use Deep Neural Networks (DNN) as predictive algorithms (or hypothesis). They have been applied and studied on different kinds of input (text, speech, temporal series,...) but their application on image is particularly convincing [KSH12; He+16; DTC16]. An appealing characteristic of deep learning is the ability to learn representations from raw pixels input. Unlike classical machine learning models, they do not require a handcrafted featuring operation to tackle the dimensionality issue and DNNs can be fed with raw data.

1.1.2.1 DNN architectures.

A DNN is a finite graph of operations that can be described as a sequence of **layers**. For instance, a DNN h, with L layers, and parametrized by the vector $\mathbf{w}$, outputs a value $\mathbf{y}^L = h(\mathbf{x}, \mathbf{w})$ for the input $\mathbf{x} \in \mathcal{X}$, that can be computed in the following sequential way: $\mathbf{y}^L = h^L_{\mathbf{w}_L}(h^{L-1}_{\mathbf{w}_{L-1}}(...h^1_{\mathbf{w}_1}(\mathbf{x})...))$ where $h^\ell_{\mathbf{w}_\ell}$ defines a layer's operation parametrized by $\mathbf{w}_\ell$. The parameters $\mathbf{w} = [\mathbf{w}_1; \mathbf{w}_2; ..; \mathbf{w}_L]$ are the learnable parameters, sometimes called **weights**. Unlike BoW, DNNs construct their own featuring system, from layer to layer. In fact, a layer ℓ of the DNN can be regarded as a featuring step, that takes as input the features vector output by the precedent layer $y^{\ell-1}$, and producing the feature vector $y^\ell = h^\ell_{\mathbf{w}_\ell}(y^{\ell-1})$. These sequential operations introduce a semantic hierarchy between features of different layers. The outputs of the layers closer to the input are called low level features. The outputs of the layers closer to the DNN's output are called high level features. The higher level features are supposed to capture abstract concepts (such as shape, object, visual texture, etc) and to provide semantic information about the input. In the rest of the thesis, we

call interchangeably a layer's output, a layer feature variable. A **neuron** is defined as a particular coordinate of a feature variable.

The architecture of a DNN is defined by the number (L) and the nature of its layers. In modern architectures, a layer's operation is usually the composition of a parametrized linear transformation, with an element wise, non-linear function, called the **activation function**.

Linear Layers. If the linear operation simply applies a (biased) matrix multiplication to the layer's input, it is called a **fully connected layer**. In this case, the coefficient of the matrix and the bias are the learned parameters. A DNN with only fully connected layers is called a Multi-Layer Perceptron (MLP). Note that MLPs were tested without much success on image classification. In a convolutional layer, the linear operation is a convolution of the input with several filters [LeC+98]. The filters have a small compact support, called filter window, and their window coefficients are learned. For two dimensional images, the output of a convolutional filter is a two dimensional matrix, called a filter map. In image applications, the output of a convolutional layer is a three dimensional **tensor** that concatenates the different filter maps and that we call feature maps. Exploiting the natural decorrelation between distant pixels in the image, the convolutional filters have small windows and then, few parameters to learn, which is a particular advantage in machine learning. On the other side, the convolution operation takes advantage of the fact that the image probability distribution has translation invariances. A DNN that possesses convolutional layers is called a Convolutional Neural Network (CNN). CNNs have been introduced first by [Fuk80] and then popularized by [LeC+98]. In 2012, [KSH12], whose CNN architecture named AlexNet is pictured in Figure 1.2, has demonstrated deep learning aptitudes on image classification. In fact, AlexNet has beat all other models by a large margin, on the famous ILSVR2012 competition [AF10]. CNNs are currently the state of the art in image classification and used in most tasks of computer vision in general.

Activation Functions. Concerning the activation function that comes right after the linear operation, the bio inspired sigmoid or hyperbolic tangent were prominent activation functions in the early 2000's. They can be found on the famous architecture LeNet [LeC+98], that was standing as a proof of concept for CNNs, on a handwritten digits dataset, MNIST [Lec+98]. Currently, the most popular activation function remains ReLU: $ReLU(y) = max(y, 0)$. In the following, unless specified, all considered DNNs are using ReLU as activation function. One great benefit of this activation is its capacity to speed up the training time by avoiding saturation issues [BL16].

A DNN with only ReLU activation defines a linear function per piece. It is known that any continuous function can be approximated on a compact set by a DNN, as closely as desired [Aro+18a]. This provides to the deep learning hypothesis space, the property of *universal approximation*. This property is important because it partially explains deep learning aptitude to fit the training data.

Softmax and prediction. In image classification, the final layer L of the DNN is usually a fully connected layer combined with a Softmax activation function. The linear operation projects its input in a space with same dimension as the number of classes. Then, the

Softmax transforms this vector into a probability distribution on the class space. The predicted class is the one achieving the maximum probability:

$$\hat{C}_{\mathbf{w}}(\mathbf{x}) = \arg\max_{i \in \mathcal{C}} h(\mathbf{x}, \mathbf{w})_i. \tag{1.2}$$

where we define $\mathcal{C} = \{1, 2, .., |\mathcal{C}|\}$ as the indexed class space with finite number of classes $|\mathcal{C}|$.

Invariant architecture and generalization. In order to guarantee a good generalization, it is important for a machine learning model to be robust to natural variations of the input. Those variations can be reported on image by transformations, such as small translation, changes in brightness, or small object rotation. For simple transformations, images in the dataset can be artificially transformed to complete the training set. This method is called **data augmentation** and is almost systematically used in image classification. Unfortunately, more complex transformations, such as local elasticity or shooting viewpoint changes, are difficult to model. A simple approach to ensure that the trained model will be robust is to design an architecture that is invariant by design to those transformations, and this for any set of weights. We recall that a machine learning predictive algorithm is invariant to a transformation, if it returns the same prediction for any input and its transformation. For instance, some specific layer can be added to DNNs to provide invariance to some transformations. The **max pooling** operator [KSH12], that is optionnaly set after a convolution layer, is a good example. Max pooling returns the maximum activation value of small spatial window densely distributed among the feature maps. By aggregating information from a local neighborhood, this layer ensures to CNN the invariance to small translations and to local variations in general. However, this operation also filters a lot of spatial information because it summarizes the information of a window into a single value. For instance, AlexNet adds max pooling to the convolutional layers 1, 2 and 5. Pooling operations are also responsible for dividing each spatial dimension per two, as can be observed in Figure 1.2.

Modern architecture and DNN depth. The question of DNN representability occupies many researches and better understanding the architecture properties remains an matter of concern. The CNNs used for image classification have standard architectures. Some convolutional layers with optional max pooling operation precede some fully connected layers and a Softmax projection. However, since [KSH12], CNN architectures did not stop maturing, making constant progresses in term of generalization. The more remarkable evolution in the architectures is the depth that has constantly grew until now, by stacking more convolutional layers and having less fully connected layers. Until now, machine learning intuitions for better generalization, led by theoretical analysis, were privileging low complexity models. For example, the number of trainable parameters was maintained as low as possible because it is a factor of over-fitting. In deep learning, on the contrary, augmenting the complexity and the expressivity of CNNs architecture by increasing their depth has not necessarily a bad influence on the generalization performance. Indeed, right after AlexNet, came vgg architecture [Kar15] with its 19 layers and GoogLeNet [Sze+15] with its 22 layers. This last CNN differs from standard architecture by allowing non linear convolutional filters using Inception blocks [LCY14], to increase the expressivity of convolutional layer. Another noticeable difference in GoogLeNet is the class projection, which is not done by a fully connected layer but with an ad hoc average pooling which saves a significant amount of parameters. Today, DNNs exploiting new convolutional layers using a residual function can be very deep and are the state of the art, like the 151 layer CNN, ResNet [He+16]. Furthermore, all these modern architectures are extremely deep, and contain huge amounts of parameters to optimize. For instance vgg contains 160M parameters for 19 layers, GoogLeNet contains approximately 7M parameters for 21 layers and ResNet contains around 7M parameters for 151 layers. These figures could be frightening when considering to optimize the parameters. However, as explained below, DNN can be trained very efficiently on big datasets.

1.1.2.2 DNN training

A first factor enabling deep learning to obtain excellent performance in image classification is that DNNs can learn properly a training set even of substantial size. This training is effected by optimizing an objective function which enforces the DNN good predictions when minimized. The objective function, that we further describe in the following, is efficiently minimized by gradient descent methods.

DNN objective function. In image classification, the error function ℓ used to train DNNs, is usually the cross entropy loss function [KSH12]. For an input x, labelled with class c, the cross entropy value is $\ell(h(\mathbf{x}, \mathbf{w}), c) = -\log h(\mathbf{x}, \mathbf{w})_c$. It corresponds to the Kullback Liebler divergence between the deterministic probability distribution associating probability one to the class c, and the DNN's output $h(\mathbf{x}, \mathbf{w})$, which defines a probability distribution on the class space. When this cross entropy is low, the output coordinate indexed by c has a high value compared to other coordinates. This ensures a good classification as the prediction of the DNN will be the class giving the maximum probability. Following the REM principle,

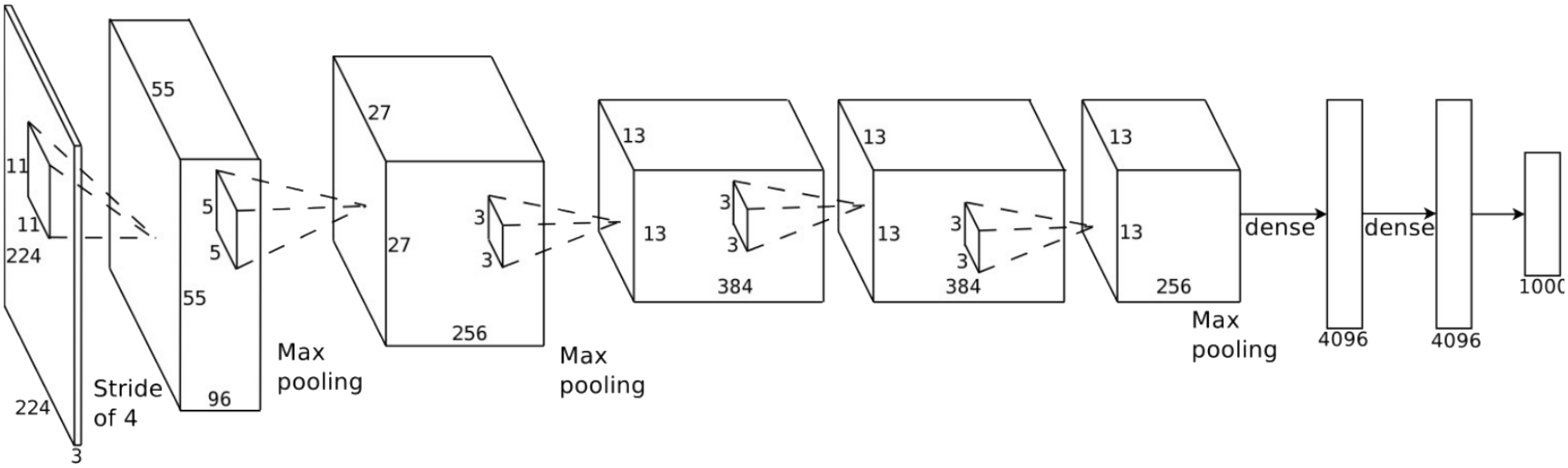


Figure 1.2.: Architecture of AlexNet CNN. It begins with five convolutional layers, with max pooling function after convolutinonal layers 1,2 and 5. It ends with three fully connected layers before the Softmax projection. The CNN takes 224×224 pixel images as inputs and it outputs probability distributions over 1000 classes. It has been trained on the ImageNet dataset for which it was state of the art in 2012.

the training objective function $\mathcal{L}(\mathbf{w})$ is the averaging of the cross entropy loss function over the training set, to ensure small prediction error in average:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} -\log h(\mathbf{x}^{(n)}, \mathbf{w})_{c^{(n)}}$$

The objective function is convenient because it is differentiable with respect to the DNN's parameters $\mathbf{w}$ and gradient descent methods can be contemplated. It is today the standard loss function used in deep learning for image classification.

DNN optimization. The most popular gradient method used in deep learning for image classification is Stochastic Gradient Descent (SGD)[Bot10]. The SGD iteratively applies a descent steps on the DNN's weights. The descent direction is computed from the gradient of the loss function ℓ, averaged over a random sub-sample of couple (input, label), draw from the training set. This sub-sample is different for every descent step and is called **mini-batch** or simply **batch**. Since the computed gradients at each step only involve a small amount of the data, it is much faster than a classical gradient descent on $\mathcal{L}$, that requires to compute the loss' gradients for the whole dataset. Even if the computed gradient is then different from the objective function gradients, this method converges in reasonable time toward very small local minimum for $\mathcal{L}$. Although the objective function is non-convex, the local minimum found by the SGD for different initializations, are almost equivalent in terms of value. Many works tend to justify that all local minimum of the objective function of a DNN using ReLU, are global minimum [HV17; LL17]. This property remains unproven in the general case, but is observed empirically. In practice, for many DNNs, it is possible to get an objective function extremely low, ensuring good prediction on almost all the training set.

The SGD and its variant [Sut+13; KB15] have proved to be very efficient at optimizing DNN objective functions. However, the SGD is not able to find very low minimum for every given architecture. For instance, very deep architectures are difficult to train. Whereas the DNN depth has a positive impact on generalization performance, it also exacerbates unfortunate phenomena such as exploding of vanishing gradients. Some modification on the architectures had to be done in order to continue to study deeper DNNs. First, residual layers [He+16] were particularly helping the SGD to reduce the objective function without impacting the representability of the DNN. Nevertheless, Batchnorm [IS16] is also an essential layer to train very deep DNNs. This layer, placed before the activation function, aims at normalizing the feature variable of the convolutional layers to zero mean and unit variance. This operation eases the optimization by controlling the "covariate shift" but is also responsible for enhancement in generalization performance. For this reason it is also considered as a regularization method.

Regularization. As explained above, DNN optimization can find weights giving very high accuracy on the training set, sometimes reaching zero classification error. Observing this aptitude to learn easily the training set, one can fear that DNNs simply overfit the data and are guilty of big generalization gaps. In fact, for a long time, MLPs were demonstrating high ability to over fit the data without being able to generalize correctly. The question of

DNN generalization is essentially a question of architecture as illustrated in [Zha+17]. For instance, CNNs have a structural advantage compared to MLPs, which makes them more relevant for images applications where they have demonstrated small generalization gaps. Interpretations of convolutional layers show that they are equivalent to fully connected layers with a strong prior on the weights which has proven to be adequate for images.

Furthermore, one can wonder if for a given architecture, all sets of weight with similar training error have automatically equivalent generalization errors. Experiments in [Kes+16] have demonstrated that this is not true. Indeed, local minima found using large mini-batches in the optimization generalize poorly compared to minima found with small batches. They demonstrate that using small batches is a regularization technique because it influences the optimization toward local minima that are not sharp. This illustrates that for a given architecture, it is possible to regularize the training to obtain a better generalization gap. Regularization is in fact, a common practice in deep learning.

For instance, a classical method to regularize the training is to add another loss term to the objective function, aiming at influencing the optimization toward a local minimum with desired properties. A famous example is **weight decay** [KH92], which penalizes weights with high variations by adding to the objective function the euclidean norm of the parameters vector $\|\mathbf{w}\|_2^2$. This method can also be interpreted as a Bayesian prior on the weights. Even if the benefits of this method seem to be negligible in modern architectures [Zha+17], it is still widely used because it also helps to stabilize the training. Dropout [Hin+12] is also a famous regularization method that aims at prioritizing DNNs which decision is not strongly dependent on a subset of learned features. However, this method is not systematically used with modern architectures because batchnorm lower its positive effect [Li+18].

The way to regularize deep learning training seems wide open, but one major difficulty to study new methods is the lake of theoretical interpretation of DNN mechanism. Indeed, the factors influencing deep learning generalization are not yet sufficiently understood to be optimized. Indeed, there is no consensus about the theory of deep learning generalization, which is a new exciting research topic. For instance, recent attempts have helped to understand deep learning performance with an information theory approach [AM17; JSO18; NN15].

1.1.2.3 Programming on GPU

It is not possible to talk about the rise of deep learning and CNN performance without mentioning the role played by GPU (Graphic Processus Unit) computing. Cuda is a language extending C and C++, that permits to execute computation on a GPU. GPU computing is well adapted to parallel CNN's three dimensional convolutional operations. Implementing a framework enabling GPU computing to train CNNs was first proposed by [KSH12] who successfully trained a height layers deep CNN on the ImageNet dataset. Before [KSH12], CNNs had obtained promising performances on small and simple datasets such as MNIST [Lec+98], but were failing to demonstrate their full potential on large scale image classification tasks. Among the factors to blame is the training time of deep CNN that was way to long. Studying properly different architectures and different training method was not possible on big datasets. With GPU computing, [KSH12] was able to properly train a CNN within a week and to obtain convincing performance compared to the state of the art.

Today, GPU computing is omnipresent in deep learning for computer vision. Universities and companies have open sourced their frameworks, which have been quickly developed in many popular languages. For example, matconvnet for Matlab [VL15], torch 7 for lua [CBM02], pytorch and tensorflow for Python [Aba+16], among many others tools to apply deep learning. Since [KSH12], GPU computing for CNN has evolved very fast. For instance, the company Nvidia has developed and is maintaining CuDNN, a CUDA library especially made to optimize DNN computations on GPU. Furthermore, the GPU hardware has also improved, getting more memory and more cores to train big CNNs with tons of parameters. Today, it takes only a dozen of hours to train an AlexNet CNN on ImageNet with tensorflow.

1.2 Motivations and Contributions

As explained earlier, deep learning performance is impressive in many computer vision applications. For image classification, CNNs' abilities to generalize is very appealing and their use has been systematized on existing datasets where they are currently state of the art. With such encouraging results, both industries and academia will consider applying deep learning on new challenging applications. As a consequence, deep learning models will be asked to perform well on more complex tasks like classifying blurry images [Che+15] or having a lot of classes to recognize [Zha+18]. Extending the human level results obtained on elementary datasets, such as CIFAR-10 or MNIST, is not obvious and improving deep learning general performance is still on the table.

We place our researches in this context, with the objective to strengthen deep learning productivity in image classification. With this motivation we focus our research on characteristics that impact the generalization capacity of deep learning. We bring the attention on three axes of interest that are identified as leverage to improve CNN performance. First, because the training time is critical in deep learning, we have inspected the way deep DNNs can accurately learn considerably-sized datasets with SGD in a reasonable time. With DNN depth constantly growing and datasets with consistently increasing number of images, this question is essential. Second, we have studied the influence of the architecture of a DNN on its generalization performances. By deciphering a DNN's internal treatment of the information, we have studied alternative architectures. Finally, we have focused our research on regularizing DNN training. By modifying the objective function in the SGD optimization we studied how regularization can impact the DNN performance.

These three points of view offer a large panorama of deep learning methods that we intend to improve with the following contributions.

1.2.1 Training very deep DNN on massive datasets

Increasing the number of images in the training set is a straightforward solution to improve models generalization. Indeed, with more data, the natural image distribution is better represented, which allows higher quality learning. In general, training a machine learning model with a lot of data guarantees the robustness of the statistical evaluations.

Unfortunately, the training time of a DNN simultaneously increases with the number of images. Furthermore, modern state of the art architectures never cease to get deeper,

which slow down the SGD. Having a very long training phase is an issue because it reduces the number of architectures that can be tested and training methods cannot be evaluated adequately. For instance, before GPU computing, it was not possible to train a DNN on a large scale dataset such as ImageNet. Being able to properly learn a very big dataset while controlling the training time is a major issue in deep learning.

With the ambition to accelerate the optimization of DNNs' weights, we have studied the possibility to distribute the training over several GPUs. In fact, the sequential mini-batch optimization can be distributed in order to process several mini-batches at a time. It is important to watch that distributed methods do not deteriorate the overall optimization. Indeed, the difference in the optimization process implied by a distributed method must be studied. It is important to identify the properties a distributed strategy should ensure in order not to degrade the training quality. A formalism covering existing methods is thus necessary and will be part of our first contribution in this thesis:

- **Distributed framework**: Chapter 2 begins with the development of a framework that encompasses within the same formalism a particular family of distributed optimization techniques. Those methods parallelize the mini-batch optimization on different GPU processes. The framework allows us to observe important properties ensuring an efficient optimization and enable to compare distributed methods.

Still with the objective to speed up DNNs training, the Chapter 2 proceeds to proposes two distributed methods. We study them from our framework and confirm their good properties on various experiments:

- **PerSyn** distributes the optimization in a synchronized and centralized way. Despite these two constraints the algorithm is efficient, although very simple.
- **GoSGD** is an asynchronous and decentralized algorithm based on gossip exchanges [Boy+06]. The algorithm has the advantage to only involve inter GPU-communication, making it very efficient.

The framework, as well as the two distributed methods, are developed in Chapter 2 and published in [Blo+18a].

1.2.2 CNN architecture and activation function

Apart from the number and the quality of the images in the training set, a major characteristic that determines a DNN generalization performance is its architecture. For instance, CNNs usually generalize way better than MLPs on image tasks. As already discussed, not all architectures can be tested and exploring the architecture space randomly is quite inefficient. Our strategy to consider better architectures is to address the question of layers expressivity and filtering of information. The max pooling layer and the ReLU activation function have been recently popularized in the deep learning community but they have not been extensively studied in the context of image classification. They are both responsible for filtering information, making the DNN invariant to some transformations. However, there is a possibility that the eliminated information is correlated with the target class and could

help to predict it more accurately. With this intuition, we have inspected on an activation function that permits to filter little information and that does not impact the expressivity of DNNs:

- **MaxMin** is an activation function we propose, which permits convolutional layers to filter less information from their inputs. It separates the positive part from the negative part of the linear convolutional filters' output. It then transmits both information values to the following layer. At equivalent architecture complexity and number of parameters, the CNNs using MaxMin as activation function, generalize better than the ones using ReLU. We also demonstrate that MaxMin activation can be seen as a new pooling function.

MaxMin activation function is developed in Chapter 3 and has been published [BCT16].

1.2.3 Regularizing the training

Finally, for a given dataset and for a given architecture, regularizing the training is a solution to improve the generalization of a DNN. By modifying the loss function, we intend to influence the optimization toward local a minimum with better generalization properties. In order to determine our regularization loss function, we first introduce a new criterion based on information theory, that is due to guarantee good generalization performance.

- **Entropy criterion**: To define our criterion, we assert that models with variable representations that have low entropy conditionnaly to the class variables, generalize well. The new regularization methods we propose in Chapter 4 and Chapter 5, are based on this criterion, which is close to the one proposed in the Information Bottleneck framework [TPB99]. We justify the betterment of our criterion and demonstrate that it guarantees invariant models, thus implying good generalization.

Implementing a loss function to minimize the proposed criterion during the optimization is not a simple task, because of the difficulty to estimate entropy measures. We propose two methods in chapter 4 and 5:

- **SHADE** loss function can be declined in a layer wise regularization. It estimates our criterion making some intuitive assumptions about the class information contained in each neuron. The method has been extensively tested on various datasets, demonstrating encouraging results.

  SHADE is described in Chapter 4 and has been published recently [Blo+18b].

- **REVE** loss function reduces our criterion, only for the last layer of the DNN. However, this work slightly modifies the proposed criterion, by estimating the entropy for the variable that effectively gives the prediction. This way, the regularization is more efficient at regularizing the DNN. It has been extensively tested on various datasets for similar architectures as SHADE. The results are systematically positive and the method presents several axes of improvement. It is described in Chapter 5 and is under review at AAAI [BSC18].

Publications

Published

- *MaxMin convolutional neural networks for image classification*, Michael Blot and Matthieu Cord and Nicolas Thome, IEEE International Conference on Image Processing (ICIP), 2016
- *Gossip training for deep learning*, Michael Blot and David Picard and Nicolas Thome and Matthieu Cord, NIPS Optimization workshop, Barcelona, 2016
- *SHADE: Information-Based Regularization for Deep Learning*, Michael Blot and Thomas Robert and Nicolas Thome and Matthieu Cord, IEEE International Conference on Image Processing (ICIP), 2018, Best paper award
- *Information-Based Regularization for Deep Learning*, Michael Blot and Thomas Robert and Nicolas Thome and Matthieu Cord, ICML Theory of Deep Learning workshop, Stockholm, 2018,
- *Distributed Optimization for Deep Learning with Gossip Exchange*, Michael Blot and David Picard and Nicolas Thome and Matthieu Cord, accepted with minor revisions at Neurocomputing, 2018

Under Review

- *Reve: Regularizing deep learning with variational entropy bound*, Antoine Saporta and Michael Blot and Matthieu Cord, AAAI 2019

CHAPTER 2

DISTRIBUTED SGD FOR DEEP LEARNING

Contents

Chapter abstract

In this Chapter, we investigate distributed methods to accelerate deep learning training on big datasets. More specifically, we study algorithms to parallelize the mini-batch processing of SGD over several threads and GPU. We develop a framework based on a formalism that incorporates popular distributed algorithms from the literature. From this framework, we propose two methods that give encouraging results when tested on challenging image datasets. A first method is PerSyn, that is centralized but limits the number of communication. The second one is GoSGD, which exploits gossip protocols to enable decentralized and asynchronous exchanges between threads.

The work in this chapter has led to the publication of a journal paper:

- *Distributed Optimization for Deep Learning with Gossip Exchange*, Michael Blot, David Picard, Nicolas Thome and Matthieu Cord, accepted with minor revisions at Neurocomputing, 2018

2.1 Introduction

As explained in Chapter 1, a DNN is trained by optimizing an objective function with gradient descents. For each descent step, the objective function and its gradient are estimated on random mini-batches according to [Bot10]. This method, called Stochastic Gradient Descent (SGD), has proven to be very efficient to train deep learning models in general in reasonable time.

However, this optimization time remains a matter of concern for researchers, because it prevents to test a lot of architectures and new training methods that are difficult to evaluate. It takes days to train the deepest state of the art architectures on ImageNet classification [CV16] and comparing different methods on this kind of DNN can be a true hurdle and a daunting task. Despite the fact that GPUs are developing toward substantial memory cache and more computing cores, the inconvenience is not going to disappear in the near future as training databases are growing and architectures are getting deeper.

Indeed, we explain in Chapter 1, that machine learning models benefit a lot from big collections of annotated images like ImageNet [JDe+09] or COCO [Lin+15]. As experimented in [Sun+17], bringing DNNs to the top of their potential requires the use of a maximum of available images. In order to boost deep learning performance, bigger databases are required. Public datasets used to train DNNs are developing in this direction, constantly increasing the number of labelled images. Simultaneously, modern architectures are getting deeper because of the positive effect it imparts on the generalization gap. Not long ago, the DNN number of layers L, was limited by the difficulty to train very deep DNN. Too many layers aggravates phenomena like vanishing or exploding gradient. Today, DNNs using batchnorm [IS16] and residual layers [He+16] can be trained efficiently even if very deep. For instance, a state of the art architecture such as ResNet embodies 151 layers and contains around 7M parameters to updated at each gradient descent step.

It is thus important to find new mechanisms to accelerate the training time. In this chapter, we investigate the possibility to use distributed methods to train DNNs. In SGD, the mini-batch optimization seems suitable for distributing the training among several threads. Many methods have been studied with success, like [ZCL15; Dea+12], which propose that several threads, called **workers**, compute gradients on different mini-batches, at the same time. To compute the gradients, a workers hosts a local set of weights that is updated by applying SGD descents according to the mini-batches it receives. The workers periodically exchange information with the overall system via a central variable in order to synchronize their parameters. The information communicated by the workers can imply the gradients computed locally, or the set of weights they are in charge of.

The major issue of these methods is that they induce waits in all computing nodes due to the synchronization problems with the critical resource that is the central variable. A popular way for tackling this issue in distributed optimization involves exchanging information in a peer to peer fashion. In this category, Gossip protocols [Boy+06] have been successfully applied on many machine learning problems such as kernel methods [Col+16], PCA [FPG15], and k-means clustering [Di +11; FPG13]. We thus consider here, to combine Gossip protocols with SGD to efficiently train deep models without the need of a central node.

Proposing new distributed algorithms that have the properties required to speed up the training is not evident. In order to study the different methods in term of training

acceleration, it is important to develop a formalism that enables to compare them. With this intention, a framework presented in this chapter, exhibits the pertinent properties of a distributed algorithm. This framework shows that distributing the mini-batches among workers is somehow equivalent to use bigger batches in a standard SGD. Another indication from the framework is that the efficiency of the distributed method largely relies on its ability to control a consensus between the workers' sets of weights. We are then able to propose two distributed SGD algorithms for training DNNs. The first one, called PerSyn, uses a central node in a very efficient way. The second one, called GoSGD, is based on Gossip exchanges and can thus be scaled to an arbitrarily large number of nodes.

The remaining of this chapter is as follows. In the next section, we recall the principles of SGD applied to deep learning as well as the relevant literature about distributing its computation. Next, in Section 2.3, we detail our formalization of distributed SGD which allows use to compare existing algorithms and write our first proposition PerSyn. In Section 2.4, we present a decentralized alternative based on Gossip algorithms, that we call GoSGD, which fits nicely in our framework. Finally, we present experiments showing the efficiency of distributing SGD in Section 2.5 before we conclude in Section 2.6.

2.2 SGD and Related Work

In the following we note the variable couple (input, class) $\mathbf{S} = (X, C)$, which follows the joint probability $p(S) = p(X, C)$. $\mathbf{w}$ is the vector with DNN parameters which is the variable to optimize. Because the following does not depend on the loss function $\ell(h(X, \mathbf{w}), C)$ as long a it is differentiable with respect to $\mathbf{w}$, we note the precedent $L(\mathbf{S}, \mathbf{w}) = \ell(h(X, \mathbf{w}), C)$ for simplification.

2.2.1 SGD for deep learning

In image classification, training deep learning models follow the empirical risk minimization (ERM) principle [Vap95] on the error function L. The underlying objective function to minimize is:

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{\mathbf{S}\sim p}[L(\mathbf{S}, \mathbf{w})] \tag{2.1}$$

Despite the objective function usually being non convex, gradient descent has been shown to be a successful optimization method for the problem. In general gradient descent algorithm [LY08], the update operation

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \mathcal{L}(\mathbf{w}) \tag{2.2}$$

is sequentially applied to the optimized variables until a convincing local minimum for $\mathcal{L}$ is found or some stopping criteria are reached. η is the descent step size, called the **learning rate**. Assuming some properties on the objective function and some constraints on the learning rate, the gradient descent is guaranteed to asymptotically converge toward local minimum [RSS12].

In image classification, it is possible to compute neither the expected value $\mathcal{L}(\mathbf{w})$, nor its gradient $\nabla\mathcal{L}(\mathbf{w})$ because the distribution p is unknown (and not properly modeled). However it is possible to approximate the gradient update term using a Monte Carlo method[1]:

$$\begin{aligned}\nabla\mathcal{L}(\mathbf{w}) &= \nabla_{\mathbf{w}}\big(\mathbb{E}_{\mathbf{S}\sim p}[L(\mathbf{S},\mathbf{w})]\big)\\ &= \mathbb{E}_{\mathbf{S}\sim p}[\nabla_{\mathbf{w}}L(\mathbf{S},\mathbf{w})]\end{aligned}$$

The last quantity is approximated by a Monte Carlo estimator:

$$\widehat{\nabla}\mathcal{L}(\mathbf{w}) = \frac{1}{K}\sum_{k=1}^{K}\nabla_{\mathbf{w}}L(\mathbf{s}^{(k)},\mathbf{w})$$

The gradient estimator, $\widehat{\nabla}\mathcal{L}(\mathbf{w})$, defines the descent direction used at each update of the SGD. The set $(\mathbf{s}^{(k)})_{k=1..K}$ of independent samples drawn from p is called a mini-batch and K is the batch size. In practice, the images in a mini-batch are drawn randomly from the training set and are different for every descent step. Since popular GPU enables the parallelism of the loss gradients ($\nabla_{\mathbf{w}}L(\mathbf{S},\mathbf{w})$) computation within a batch, the computational time is not really linear with the batch size anymore but still grows with it.

Concerning the precision of the gradient indicator, we point out that it is unbiased. It can also be shown that the approximation with respect to the true gradient depends on the batch size. In fact, we have the following relation demonstrated in Appendix A.1,

$$\mathbb{E}_{(\mathbf{S}^{(k)})_{k=1..K}\,\sim\,p^K}||\nabla\mathcal{L}(\mathbf{w}) - \widehat{\nabla}\mathcal{L}(\mathbf{w})||_2^2 \propto \frac{1}{K}tr\big(\mathbb{C}\mathrm{ov}_{\mathbf{S}\sim p}[\nabla L(\mathbf{S},\mathbf{w})]\big)$$

where $\mathbb{C}\mathrm{ov}_{\mathbf{S}\sim p}[\nabla L(\mathbf{S},\mathbf{w})]$ is the covariance matrix of the gradient and tr denotes the trace of the matrix. Therefore the bigger the batch size K, the more accurate the gradient estimator.

It is worth mentioning that the dimension of the gradient direction is immense (around 7M for ResNet) and the descent direction is very sensitive to noise. DNN architectures having a very high number of parameters require a high batch size to be optimize. For instance, the vgg architecture [Kar15], containing around 130M parameters, necessitates a minimum batch size of 256 image to be trained. More generally, the bigger are the batches, the more accurate are the gradients and the faster is the optimization. This is illustrated in Figure 2.1, that highlights the benefit of using bigger batches in the convergence speed. We see on the figure that with bigger batch size the optimization converges faster toward a lower minimum.

Unfortunately, GPU memory is limited and bounds the number of images that can be processed in a batch. For instance, a GPU card with 12Go of memory cannot process batches bigger than 16 images of resolution 256×256 RGB pixels for a ResNet CNN with 151 layers. Indeed, at each update step, the GPU needs to host the whole model intermediate computations for all images in the batch. As such, using a single GPU limits the precision of the stochastic gradient descent and thus hinders its convergence.

From the precedent, we identify two leverages on the general training time of a DNN. First the **convergence rate**, which measures the objective function value at constant number of

1 The properties allowing to swap the derivative and the integral are supposed to be respected and will not be discussed any further.

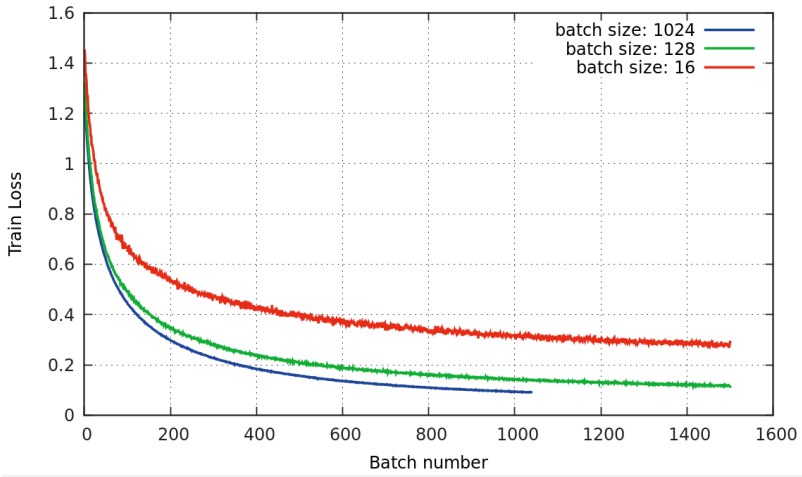


Figure 2.1.: Evolution of the objective function –estimated on training batches– at constant number of SGD update for different batch size: 16 (blue), 128 (green), and 1024 (red). The objective function is computed for a CNN of same architecture trained on CIFAR 10. The values are compared at constant number of gradient descent update. It shows that with the same number of descent step, the optimization with 1024 images per batch is faster. This is because the gradients are more precise with 1024 images than with 128 or 16 images.

descent step. It benefits from efficient descent direction at each update. Then, comes the computational time of the descent direction which grows with the batch size. The time to update the parameters is supposed to be constant because the architecture of the DNN is fixed in the training. In order to ensure a high convergence rate, with small computational time, the SGD has to balance the batch size adequately.

2.2.2 Related work

Among the alternatives to accelerate the convergence rate of SGD, one of which is to optimize the learning rate η for each gradient descent iteration as aimed at by second order gradient descent methods [LY08]. For deep learning, the parameters space is of high dimension and second order methods are computationally too heavy to be applied efficiently. Examples of first order gradient descent that adapt the learning rate with success can be found in [KB15; Hin13].

Regarding the descent direction $\widehat{\nabla}\mathcal{L}(\mathbf{w})$, a second way to accelerate the SGD is to reduce its computation time or its precision. In this chapter we deal with both aspects. Augmenting the batch size and thus the precision of $\widehat{\nabla}\mathcal{L}(\mathbf{w})$ is a solution to boost the convergence rate. However this option increases the computational time and has GPU memory limitations. Based on the availability of several GPUs, different methods that overcome the batch size constraints have been proposed by distributing gradient computation among the different GPUs. There are two fundamental approaches to distribute the computation of the gradients and updates, which are either splitting the model and distributing parts of it among the different GPUs, or splitting the mini-batch of images in subsets and distributing them

among the GPUs. A famous technique for distributing the model, used by [KSH12] and presented in [KI14], consists in paralleling the batch losses and gradients computations over two GPUs. The convolutional part of the DNN is split into two independent channels that are only aggregated with a fully connected layer at the top of the network. This conception enables to load the two parallel parts on two different GPUs, liberating space for more images in batches. However, the DNN models claiming current state of the art performances in image classification like [He+16; Hua+17], benefit a lot from the inter-correlation between features of each layer and do not allow independent partitioning within a layer. Thus, if the network is loaded in several GPU cards they would have to communicate during gradient computations. The communication being time-consuming, it is preferable to restrain a whole DNN on a single GPU.

The second manner, which consists in distributing subsets of a mini-batch to the different GPUs, is becoming increasingly popular. We use [ZCL15] notations to describe such methods: There are M threads, called worker, each owning a GPU. Every GPU hosts an independent set of weights noted $\mathbf{w}_m$ for worker m. When provided a mini-batch of image, a worker is able to compute a gradient estimate from it. The set of parameters used for inference is noted $\tilde{\mathbf{w}}$ and is specifically the model's weights we intend to optimize. The easiest way of distributing mini-batches among workers is presented in [MMT16] and consists in computing the average of all the workers weights after every descent step as in Algorithm 1.

Algorithm 1 Fully synchronous SGD: pseudo-code

Input: M: number of threads, η : learning rate
Initialize: $\mathbf{w}$ is initialized randomly, $\mathbf{w}_m = \mathbf{w}$
repeat
    $\forall m,\ v_m = \widehat{\nabla}^m \mathcal{L}(\mathbf{w}_m)$
    $v = \frac{1}{M} \sum_{m=1}^{M} v_m$
    $\forall m, \mathbf{w}_m = \mathbf{w}_m - \eta v$
until Maximum iteration reached
return $\frac{1}{M} \sum_{m=1}^{M} \mathbf{w}_m$

We stress the fact that after each iteration, all workers host the same value of the variable $\tilde{\mathbf{w}}$ to optimize (i.e., $\forall m, \mathbf{w}_m = \tilde{\mathbf{w}}$). The mini-batches used to compute $v_m = \widehat{\nabla}\mathcal{L}_m(\mathbf{w}_m)$ are sampled from the training set independently from a worked to another. This is the reason we index the gradient with the worker's index m. Averaging the workers' gradient estimates is equivalent at using M times the number of image to estimate the gradient at point $\tilde{\mathbf{w}}$ as demonstrated by the following equation:

$$
\begin{aligned}
v &= \frac{1}{M} \sum_{m=1}^{M} \widehat{\nabla}^m \mathcal{L}(\tilde{\mathbf{w}}) \\
&= \frac{1}{M} \sum_{m=1}^{M} \frac{1}{K} \sum_{k=1}^{K} \nabla_{\mathbf{w}} L(\mathbf{S}^{(k,m)}, \tilde{\mathbf{w}}) \\
&= \frac{1}{MK} \sum_{m=1}^{M} \sum_{k=1}^{K} \nabla_{\mathbf{w}} L(\mathbf{S}^{(k,m)}, \tilde{\mathbf{w}})
\end{aligned}
$$

The descent direction that updates $\tilde{\mathbf{w}}$ is the averaging[2] of the loss gradients of $M \times K$ random and independent sampled images. The method is thus equivalent at using M times bigger batches for the gradients estimation in a single threaded SGD. Consequently, it is possible to alleviate the single GPU memory constraint that limits the number of examples in a batch when possessing several GPUs. However, in comparison with the classical single threaded SGD there are three additional phases of communication and aggregation that impose incompressible times:

1. The transmission to the master of the local gradient estimates computed by the threads at line 4.
2. The aggregation of the gradients at line 5.
3. The transmission of the aggregated gradient $\widehat{\nabla}\mathcal{L}(\mathbf{w})$ to all threads before local updates at line 6.

In large scale image classification, the communications involved at lines 4 and 6 imply a DNN with millions of parameters and are thus very costly. Moreover, to perform line 5, the master has to wait for all workers to have completed line 4. The additional time involved by communications and synchronizations can have a significant impact on the global training time and this naive synchronous distributed method can be very inefficient.

The goal of our study here is to reduce these communication and aggregation costs. Different algorithms have been proposed to reduce the amount of communication between the central variable and the workers with periodical exchange of information. In [ZCL15], when a worker exchanges information with the central node, it replaces its parameters variable with an elastic averaging of its own weight variable and the central node variable. The master thread does the symmetrical update on the central node variable. In [Dea+12], the workers accumulate in an additional buffer the gradients computed at the last steps, and applies this aggregated gradient to the central node parameters after a fixed number of descent steps.

As with existing strategies such as [ZCL15; Dea+12], the main idea is to control the amount of information exchanged between the nodes. This amount is a key factor in making all workers contribute to the optimization of the test weights $\tilde{\mathbf{w}}$. Indeed, if no information is ever exchanged, the whole distributed system is equivalent at training M independent models. As a consequence of the symmetry property of DNN explained in [Cho+15], these independent models are likely to be very different and almost impossible to combine in a single model $\tilde{\mathbf{w}}$. Note that we are not interested here in training several DNNs, but to accelerate the training of a single one. As such, the distributed training would not improve over using a single GPU with a reduced batch size. The main challenge of distributed methods is thus to allow as little information exchange as possible to ensure reduced communication costs, while still optimizing a single aggregated model $\tilde{\mathbf{w}}$.

In the next section, we introduce a matrix framework that allows to explicitly control the amount of exchanged information, and thus to explore different optimization strategies.

2 In [MMT16] they use the sum instead of the averaging of the gradients but both are equivalent by adjusting the learning rate by a factor of M.

2.3 SGD Distributed Framework

In this section we propose a general framework for distributed SGD methods. We start from the single threaded SGD update and we develop the equivalent centralized and synchronous distributed procedure. Then, we show that relaxing some equality constraints among workers is equivalent to consider different communication strategies. We are thus able to write methods found in the literature using a simple matrix expression.

2.3.1 SGD Recursive formulae

First, we unroll the recursion of the classic SGD update of Equation 2.2 to obtain the value of $\tilde{\mathbf{w}}$ after $T+1$ gradient descent steps:

$$\tilde{\mathbf{w}}^{(T+1)} = \tilde{\mathbf{w}}^{(0)} - \eta \sum_{t=0}^{T} \hat{\nabla} \mathcal{L}(\tilde{\mathbf{w}}^{(t)}) \tag{2.3}$$

With $\tilde{\mathbf{w}}^{(t)}$ being variable $\tilde{\mathbf{w}}$ a time t (after t gradient descent iterations)[3]. The equivalent batch distribution method described in Algorithm 1 is the following:

$$\tilde{\mathbf{w}}^{(T+1)} = \tilde{\mathbf{w}}^{(0)} - \eta \sum_{t=0}^{T} \frac{1}{M} \sum_{m=1}^{M} \hat{\nabla}^m \mathcal{L}(\tilde{\mathbf{w}}^{(t)})$$

In order to make the communication between the workers and the central node more visible, we can use the equivalent recursive rule that introduces local workers' variable $\mathbf{w}_m$ and consensus constraints. By doing so, we obtain the standard distributed SGD method:

$$\tilde{\mathbf{w}}^{(T+1)} = \tilde{\mathbf{w}}^{(0)} - \eta \sum_{t=0}^{T} \frac{1}{M} \sum_{m=1}^{M} \hat{\nabla}^m \mathcal{L}(\mathbf{w}_m^{(t)}) \tag{2.4}$$

$$\text{s.t. } \forall t,\ \forall m,\ \mathbf{w}_m^{(t)} = \tilde{\mathbf{w}}^{(t)} \tag{2.5}$$

We can replace the consensus equality constraints (2.5) by equivalent local update rules obtained from (2.4):

$$\forall r,\ \mathbf{w}_r^{(T+1)} = \tilde{\mathbf{w}}^{(0)} - \eta \sum_{t=0}^{T} \frac{1}{M} \sum_{m=1}^{M} \hat{\nabla}^m \mathcal{L}(\mathbf{w}_m^{(t)})$$

This formulation is a first step towards decentralized and communication efficient algorithms since it makes the communications between workers visible. Indeed, at each step, each worker gathers the gradients from all other workers and applies them to its local variable. The main drawback of this formulation is that the number of messages exchanged

3 For simplicity, we consider here a constant learning rate η. The general case can be expressed by indexing the learning rate with the time t, η^t, and include it in the sum.

by workers at each step is now $M(M-1)$ instead of $2M$ in the classical distributed version in Algorithm 1.

To allow for more efficient communication, we relax those consensus constraints by allowing local variables to differ slightly. This can be done efficient by introducing non-uniform weights $\alpha_{r,m}$ in the combination of the local gradients:

$$\tilde{\mathbf{w}}^{(T+1)} = \tilde{\mathbf{w}}^{(0)} - \eta \sum_{t=0}^{T} \sum_{m=1}^{M} \alpha_{0,m} \hat{\nabla}^m \mathcal{L}(\mathbf{w}_m^{(t)}) \tag{2.6}$$

$$\forall r,\ \mathbf{w}_r^{(T+1)} = \tilde{\mathbf{w}}^{(0)} - \eta \sum_{t=0}^{T} \sum_{m=1}^{M} \alpha_{r,m} \hat{\nabla}^m \mathcal{L}(\mathbf{w}_m^{(t)}) \tag{2.7}$$

$$\text{s.t. } \forall r,\ \sum_{m=1}^{M} \alpha_{r,m} = 1 \tag{2.8}$$

For coherency with the synchronous algorithm, the coefficients $\alpha_{r,m}$ have to sum to one ($\forall r, \sum_{m=1}^{M} \alpha_{r,m} = 1$). This ensures that the equivalent gradient estimator remains unbiased like in the synchronous method, where all coefficient are equal to $1/M$.

Remark about the workers consensus: The major consequence of the relaxation introduced by the coefficients $\alpha_{r,m}$ and the removal of constraint 2.5 is that the workers' weights can now be different from the test variable $\tilde{\mathbf{w}}^{(t)}$. Regarding that a worker's gradient, $\hat{\nabla}^m \mathcal{L}(\mathbf{w}_m^{(t)})$ contributes to the optimization of the test variable through the formula (2.6), its direction must be close to the direction of the true gradient at point $\tilde{\mathbf{w}}^{(t)}$. If $-\hat{\nabla}^m \mathcal{L}(\mathbf{w}_m^{(t)})$ is a increasing direction for $\mathcal{L}$ at point $\tilde{\mathbf{w}}^{(t)}$, its impact on the optimization is unfavorable. Consequently, having a consensus between the workers' variable and the inference variable is important. This requirement holds for every iteration t. The initial optimization problem is thus coupled with a consensus problem.

We point out that, at constant number of gradient descent iteration T, the more efficient method would be the synchronous algorithm 1 presented above because the consensus is always absolute and all local gradient estimators have expectancy equal to the gradient of $\mathcal{L}$ at point $\tilde{\mathbf{w}}^{(t)}$. The goal of our study is to find algorithms with much fewer communication, and with comparable convergence rate.

2.3.2 Matrix notation

We introduce a matrix notation over $\alpha_{r,m}$ to simplify the expression of the update rules. We note $\mathbf{W}^{(t)} = [\tilde{\mathbf{w}}^{(t)}; \mathbf{w}_1^{(t)}; \dots; \mathbf{w}_M^{(t)}]$ the matrix[4] concatenating the central variable $\tilde{\mathbf{w}}^{(t)}$ and all local variables $\mathbf{w}_m^{(t)}$ at time step t. Similarly, we note $\mathbf{V}^{(t)} = [0_{dim(\tilde{\mathbf{w}})}; \hat{\nabla}^1 \mathcal{L}(\mathbf{w}_1^{(t)}); \dots; \hat{\nabla}^M \mathcal{L}(\mathbf{w}_M^{(t)})]$ the matrix concatenating all the local gradients $\hat{\nabla}^m \mathcal{L}(\mathbf{w}_m^{(t)})$ at time step t, with a leading vector $0_{dim(\tilde{\mathbf{w}})}$ to maintain the same size as $\mathbf{W}$. In fact, gradients are only computed locally on variables $\mathbf{w}_m$. At initialization, every worker holds the same value of the weights

4 The weights $\mathbf{w}$ and the associated gradients $\hat{\nabla}\mathcal{L}(\mathbf{w})$ are considered as column vectors.

and $\mathbf{W}^{(0)} = [\mathbf{w}^{(0)}; \mathbf{w}^{(0)}; \ldots; \mathbf{w}^{(0)}]$. The coefficients $\alpha_{r,m}$ of the gradient combinations are enclosed in a matrix $K = (\alpha_{r,m})_{0 \leq r,m \leq M}$, which leads to the following matrix update rule:

$$\begin{aligned}\mathbf{W}^{(T+1)} &= \mathbf{W}^{(T)} - \eta K \mathbf{V}^{(T)} \\ &= \mathbf{W}^{(0)} - \eta \sum_{t=0}^{T} \left(\prod_{\tau=t}^{T} K \right) \mathbf{V}^{(t)}\end{aligned}$$

K is the communication matrix and is responsible for the mixing of the local updates at iteration t. Because its rows sum to one, K is called a **stochastic matrix** and $\left(\prod_{t=0}^{T} K\right) \mathbf{W}^{(0)} = \mathbf{W}^{(0)}$, giving:

$$\mathbf{W}^{(T+1)} = \left(\prod_{t=0}^{T} K \right) \mathbf{W}^{(0)} - \eta \sum_{t=0}^{T} \left(\prod_{\tau=t}^{T} K \right) \mathbf{V}^{(t)}$$

If K is sparse, there is few communication and a good consensus is difficult to get. On the extremes, if K is diagonal ($K = Id$) (no information exchanged between workers and with central node), the update rule is equivalent to M different models being independently trained. In the fully synchronous case, where the consensus is perfect, K is the very dense matrix $\frac{1}{M}\mathbf{1}_M$, where $\mathbf{1}_M$ is the $M + 1 \times M + 1$ matrix with all coefficients equal to one, except on the first column and the first raw which are zeros. Hence, the choice of K is key in the trade-off between low communication costs and good consensus between workers. An important remark, is that the convergence rate of the distributed algorithm is bounded. It is upper bounded by the convergence rate of single threaded SGD with a batch size M times bigger (perfect consensus) and lower bounded by the consensus rate of a single threaded SGD with similar batch size (worse case with no exchanges).

To further allow for the optimization of the matrix K, we introduce time dependent communication matrices $K^{(t)}$. This allows for the design of very sparse communication matrices most of the time, balanced with the use of a dense $K^{(t)}$ to enforce better consensus whenever it is required. We note $P_t^T = \prod_{\tau=t}^{T} K^{(\tau)}$, and obtain:

$$\mathbf{W}^{(T+1)} = P_0^T \mathbf{W}^{(0)} - \eta \sum_{t=0}^{T} P_t^T \mathbf{V}^{(t)}$$

This corresponds to the following recursion:

$$\mathbf{W}^{(T+1)} = K^{(T)} \mathbf{W}^{(T)} - \eta K^{(T)} \mathbf{V}^{(T)}$$

Note that since the update is linear, mixing the local gradients and mixing the local variables are equivalent. As such, we can define an intermediate variable $\mathbf{W}^{(t+\frac{1}{2})} = \mathbf{W}^{(t)} - \eta \mathbf{V}^{(t)}$ that considers only the local updates and obtain the following update rules:

$$\mathbf{W}^{(T+\frac{1}{2})} = \mathbf{W}^{(T)} - \eta \mathbf{V}^{(T)} \tag{2.9}$$

$$\mathbf{W}^{(T+1)} = K^{(T)} \mathbf{W}^{(T+\frac{1}{2})} \tag{2.10}$$

Since these rules make a clear distinction between local computation (step $T + \frac{1}{2}$) and communication (step $T + 1$), they are the ones we will use in the remaining of this chapter.

Note that our framework incorporates other gradient descent strategies, like momentum SGD [Lan12; Sut+13]. In fact the general notation $\mathbf{V}^{(T)}$ for the descent direction, allows us to consider any update rule.

In the presented formalism, the different distributed strategy can be compared by studying the properties of the exchange matrix K they define. In the following, we show how several existing distributed SGDs can be specified by their sequence of communication matrices $K^{(t)}$.

2.3.3 Exemples

We first start with a naïve, yet communication efficient method, we call *PerSyn* (Periodically Synchronous) that considers exchanging information only once every few steps. Then, we show how EASGD [ZCL15] and Downpour [Dea+12] compare to PerSyn.

2.3.3.1 PerSyn

As a first illustration of our framework we introduce a new communication and aggregation strategy for distributed SGD that is very simple to implement and yet surprisingly effective. The main idea is to relax the synchronization of all local variables with the master variable to occur only once every τ steps. This is done by defining the following time dependent communication matrices:

$$K^{(t)} = \begin{cases} \begin{pmatrix} 0 & \cdots 0 \\ \mathbb{1}_M & \cdots 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & \frac{1}{M}\mathbb{1}_M^\top \\ \vdots & I_M \end{pmatrix}, & \text{if } t \bmod \tau = 0 \\ \begin{pmatrix} 0 & \cdots \\ \vdots & I_M \end{pmatrix}, & \text{else} \end{cases}$$

I_M being the identity matrix of size $(M \times M)$, and $\mathbb{1}_M$ the column vector of size M with 1 for every component. The corresponding algorithm is described in Algorithm 2. The matrix for $t \bmod \tau = 0$ is presented as the product of two matrices in order to exhibit two communications phases: the aggregation of the local variables and the distribution to workers of the averaged variable. Note that it is very similar to the standard distributed SGD Algorithm 1 and thus can be very easily implemented.

PerSyn involves no communication at all $1 - \rho$ $(= \frac{\tau-1}{\tau})$ percent of the time. However, during that time, models are driven by their local gradients only and can diverge from one another leading to incoherent distributed optimization. The trade-off with PerSyn is then to choose ρ such as to minimize the communication costs while preventing models to diverge from one to another during the time where no communication is done.

One major limitation of PerSyn is that the communication matrix $K^{(t)}$ is very dense when $t \bmod \tau = 0$. Because all workers exchange at the same time, the centralized operation can imply very long delay when using a lot of workers.

Algorithm 2 PerSyn SGD: Pseudo-code

Input: M: number of threads, η : learning rate
Initialize: $\mathbf{w}$ is initialized randomly, $\mathbf{w}_m = \mathbf{w}$, $t = 0$
repeat
    for all m, $\mathbf{w}_m^{(t+\frac{1}{2})} \leftarrow \mathbf{w}_m^{(t)} - \eta \widehat{\nabla}^m \mathcal{L}(\mathbf{w}_m^{(t)})$
    $t = t + 1$
    if $t \bmod \tau = 0$ **then**
        $\tilde{\mathbf{w}}^{(t+1)} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{w}_m^{(t+\frac{1}{2})}$
        $\forall m$, $\mathbf{w}_m^{(t+1)} \leftarrow \frac{1}{M} \sum_{m=1}^{M} \mathbf{w}_m^{(t+\frac{1}{2})}$
    else
        $\forall m$, $\mathbf{w}_m^{(t+1)} \leftarrow \mathbf{w}_m^{(t+\frac{1}{2})}$
    end if
until Maximum iteration reached
return $\frac{1}{M} \sum_{m=1}^{M} \mathbf{w}_m$

2.3.3.2 EASGD

In the synchronous version of EASGD in [ZCL15], the local models are periodically averaged like in PerSyn, however, the averaging is performed in an elastic way. The corresponding matrix is then:

$$K^{(t)} = \begin{cases} \begin{pmatrix} 1 - M\alpha & \alpha\mathbb{1} \\ \alpha\mathbb{1} & (1-\alpha)I \end{pmatrix}, & \text{if } t \bmod \tau = 0 \\ \begin{pmatrix} 0 & \dots \\ \vdots & I \end{pmatrix}, & \text{else} \end{cases}$$

With α being a mixing weight.

Every τ iterations, an averaging synchronization is performed to keep the local models from diverging from one another. We note that, contrarily to PerSyn, which replaces all models (local and central) by the newly averaged weights, only an elastic averaging is done here. With a similar number of communication as PerSyn, the consensus is not absolute after the communication phase making the algorithm sub-optimal. However, an asynchronous version of EASGD is proposed in [ZCL15] which allows the workers to proceed to individual elastic updates with the central variable independently, avoiding synchronization issues that can be encountered with PerSyn.

2.3.3.3 Downpour SGD

In Downpour SGD [Dea+12], another asynchronous update scheme is discussed. By asynchronous, the authors mean that each worker is endowed with its own clock and process gradient at a rate that is completely independent from the other workers. This can be easily described in our matrix framework by considering a universal clock that ticks each time one of the workers clock ticks. This corresponds to considering the finest time

resolution possible, when only one worker is awake at any given time step. In particular, this redefines $\mathbf{V}^{(t)} = [0, \dots, 0, \hat{\nabla}^m \mathcal{L}(\mathbf{w}_m^{(t)}), 0, \dots]$ to have zeros everywhere but on the component corresponding to the awaken worker m.

Whenever they awake, each worker has the option to communicate with the master while performing its gradient update. These communications can either be fetching the global model from the master or sending the current update to the master. This is expressed by the following communication matrices:

$$K^{(\text{send})} = \begin{pmatrix} 1 & \mathbf{e}_m \\ 0 & I \end{pmatrix}, \quad K^{(\text{receive})} = \begin{pmatrix} 1 & 0 \\ \mathbf{e}_m & I - \mathbf{e}_m \mathbf{e}_m^\top \end{pmatrix}$$

with $\mathbf{e}_m$ being the vector of zeros with a 1 on component m.

When nodes are neither sending nor receiving, the communication matrix is simply the identity and thus the only ongoing operation is a computation which involves a single worker that performs a local gradient update. The main drawback of Downpour is that it involves a master that stores the most up-to-date model. This single entry point can be a source of weaknesses in the training process since it acts as the communication bottleneck (all workers have to communicate with the master) and is also a critical point of failure. We intend to overcome this drawback by introducing in next section, new communication matrices that lead to fully decentralized training algorithms.

2.4 Gossip Stochastic Gradient Descent (GoSGD)

To remove the burden of using a master, several key modifications to the communication process have to be made. First, the absence of a master means that each worker is expected to converge to the same value $\tilde{\mathbf{w}}$, which corresponds to ensuring a strict consensus among workers, at least asymptotically. Furthermore, this absence of a master is reflected by the communication matrix having its first row and first column to be 0. For simplification we will omit them in our notations in this section. Therefore, all communication are performed in a peer to peer way, which is reflected in the communication matrix where the columns are the senders and the rows are the receivers.

Second, the absence of a master also implies the absence of a global clock. This means that workers are performing their updates at any time, independently of the others. To reflect this, we consider the same clock model as with Downpour, where at each time step t, only a single worker s is awaken. Consequently, this also redefines $\mathbf{V}^{(t)} = [0, \dots, 0, \hat{\nabla}_s \mathcal{L}(\mathbf{w}_s^{(t)}), 0, \dots, 0]$ to have zeros everywhere but on the component corresponding to the awaken worker s. Furthermore, since the active worker is random, the communications have to be randomized too, which leads to random communication matrices $K^{(t)}$. These random peer to peer communication systems are known as Randomized Gossip Protocols [Boy+06], and are known to converge to the consensus exponentially fast in the absence of perturbations (which in our case corresponds to $\forall t, \mathbf{V}^{(t)} = 0$). To control the communication cost, the frequency at which a worker emits messages is set by using a random variable deciding whether the awaken node shares its variables with neighbors.

Finally, we want a communication protocol where no worker is waiting for another, so as to maximize the time they spend at computing their local gradients. Waiting occurs when

a worker is expecting a message from another worker and has to idle until this message arrives. This is the case in symmetrical communication where workers expect replies to their messages. It is well known in the Randomized Gossip literature that such local blocking waits can cause global synchronization issues where most of the workers spend their time waiting for the needed resources to be available. To overcome this problem, asymmetric gossip protocols have been developed such as in [KDG03]. Asymmetric means that no worker is both sending and receiving messages at the same time, which translates in constraints on the non-zero coefficients of the communication matrix $K^{(t)}$. To assure convergence to the consensus, a coefficient $\lambda_m^{(t)}$ has to be associated with every variable $\mathbf{w}_m^{(t)}$ and is shared by workers using the same communications as the variables $\mathbf{w}_m^{(t)}$. These communication protocols are known as Sum-Weight Gossip protocols because of introduction of the weights $\lambda_m^{(t)}$.

Using these assumptions, we define the Gossip Stochastic Gradient Descent (GoSGD) as follows: Let s be the worker awaken at time t, which is our potential sender. Let $B \sim \mathcal{B}(\rho)$ be a random variable following a Bernoulli distribution with probability of success ρ. Let r be a random variable sampled from the uniform distribution in $\{1, \dots, M\} \setminus \{s\}$. r represents our potential receiver. The communication matrix $K^{(t)}$ is then:

$$K^{(t)} = \begin{cases} I + \frac{\lambda_s^{(t)}}{\lambda_s^{(t)}+\lambda_r^{(t)}} \mathbf{e}_r \mathbf{e}_s^\top + \left(\frac{\lambda_s^{(t)}}{\lambda_s^{(t)}+\lambda_r^{(t)}} - 1 \right) \mathbf{e}_s \mathbf{e}_s^\top, \text{ if } B \\ I, \text{ else} \end{cases} \tag{2.11}$$

If B is successful, the weights of the sender s and the receiver r are updated as follows:

$$\lambda_s^{(t+1)} = \frac{\lambda_s^{(t)}}{2}, \quad \lambda_r^{(t+1)} = \lambda_r^{(t)} + \frac{\lambda_s^{(t)}}{2} \tag{2.12}$$

Notice that this update involves a single message from s to r just as for the update of $\mathbf{w}_r^{(t)}$ using $K^{(t)}$. In practice, both $\mathbf{w}_s^{(t)}$ and $\lambda_s^{(t)}$ are encapsulated in a single message and sent together.

Remark also, that contrarily to what is found in the sum-weight gossip literature, we do not perform explicit ratio of the local variables and their associated coefficient. Our implicit ratio is performed by including the coefficients in the communication matrix $K^{(t)}$. This leads to more complicated matrices than what is common in the literature and hinders the theoretical analysis of the communication process (since $K^{(t)}$ are no longer i.i.d.). Furthermore, since the updates are equivalent, our proposed communication matrix keeps all the exponential convergence to consensus properties of standard sum-weights gossip protocols.

2.4.1 GoSGD Algorithm

The procedure run by each worker to perform GoSGD is described in Algorithm 3. Each worker is endowed with a queue q_m which can be concurrently accessed by all workers. Each worker repeats the same loop which consists in a sequence of 3 operations, namely processing incoming messages, performing a local gradient descent update and possibly sending a message to a neighbor worker.

Remark that all messages are processed in a delayed fashion in the sense that several messages can be received and the corresponding variables may have been updated by their respective workers before the receiving worker applies the reception procedure. From this point of view, it seems the workers are taking into account outdated information.However, as we show in the experiment, even very low probabilities of communication ($\rho = 0.01$) yield very good convergence properties and satisfying consensus.

Algorithm 3 GoSGD: worker Pseudo-code

Input: ρ: probability of exchange, M: number of workers, η : learning rate, $\forall m, q_m$: message queue associated with worker m, s current worker id.
Initialize: $\mathbf{w}$ is initialized randomly, $\mathbf{w}_s = \mathbf{w}$, $\lambda_s = \frac{1}{M}$
repeat
  processMessages(q_s)
  $\mathbf{w}_s \leftarrow \mathbf{w}_s - \eta^{(t)} \hat{\nabla}_s \mathcal{L}^{(t)}$
  if $B \sim B(\rho)$ **then**
    $r = Random(M)$
    pushMessage(q_r, $\mathbf{w}_s$, λ_s)
  end if
until Maximum iteration reached
return $\mathbf{w}_s$

Algorithm 4 Gossip update functions

function PUSHMESSAGE(queue q_r, $\mathbf{w}_s$, λ_s)
  $\mathbf{w}_s \leftarrow \mathbf{w}_s$
  $\lambda_s \leftarrow \frac{\lambda_s}{2}$
  q_r.push($(\mathbf{w}_s, \lambda_s)$)
end function
function PROCESSMESSAGES(queue q_r)
  repeat
    $(\mathbf{w}_s, \lambda_s) \leftarrow q_r$.pop()
    $\mathbf{w}_r \leftarrow \frac{\lambda_r}{\lambda_s+\lambda_r}\mathbf{w}_r + \frac{\lambda_s}{\lambda_s+\lambda_r}\mathbf{w}_s$
    $\lambda_r \leftarrow \lambda_r + \lambda_s$
  until q_r.empty()
end function

2.4.2 Distributed optimization interpretation

We now show that our proposed GoSGD solves a consensus augmented version of the distributed ERM principle. As shown in Section 2.3, the easiest way of tackling the distributed ERM principle is to consider local variables constrained to be equal:

$$\min_{\mathbf{w}_1,\dots,\mathbf{w}_M} \sum_{m=1}^{M} \mathcal{L}(\mathbf{w}_m)$$
$$\text{s.t. } \forall m, \mathbf{w}_m = \hat{\mathbf{w}}$$
$$\hat{\mathbf{w}} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{w}_m$$

As used in [ZCL15] and [Boy+11], the consensus constraints can be relaxed which leads to the following augmented problem:

$$\min_{\mathbf{w}_1,\dots,\mathbf{w}_M} \sum_{m=1}^{M} \mathcal{L}(\mathbf{w}_m) + \frac{\gamma}{2}\|\mathbf{w}_m - \hat{\mathbf{w}}\|_2^2 \tag{2.13}$$

$$\text{s.t. } \hat{\mathbf{w}} = \frac{1}{M}\sum_{m=1}^{M} \mathbf{w}_m \tag{2.14}$$

We can rewrite this loss in order to make the equivalent gossip communications visible:

$$\min_{\mathbf{w}_1,\dots,\mathbf{w}_M} \sum_{s=1}^{M} \left(\mathcal{L}(\mathbf{w}_s) + \frac{\gamma}{4M} \sum_{r=1}^{M} \|\mathbf{w}_s - \mathbf{w}_r\|_2^2 \right) \tag{2.15}$$

The gradient of this objective function contains 2 terms, the first being related to the empirical risk and the second being the pairwise distance between local models. We prove in Appendix A.2 that the update rules of GoSGD are in expectation equivalent to this two gradient terms and thus that GoSGD performs a stochastic alternate gradient descent on this augmented problem.

2.5 Experiments

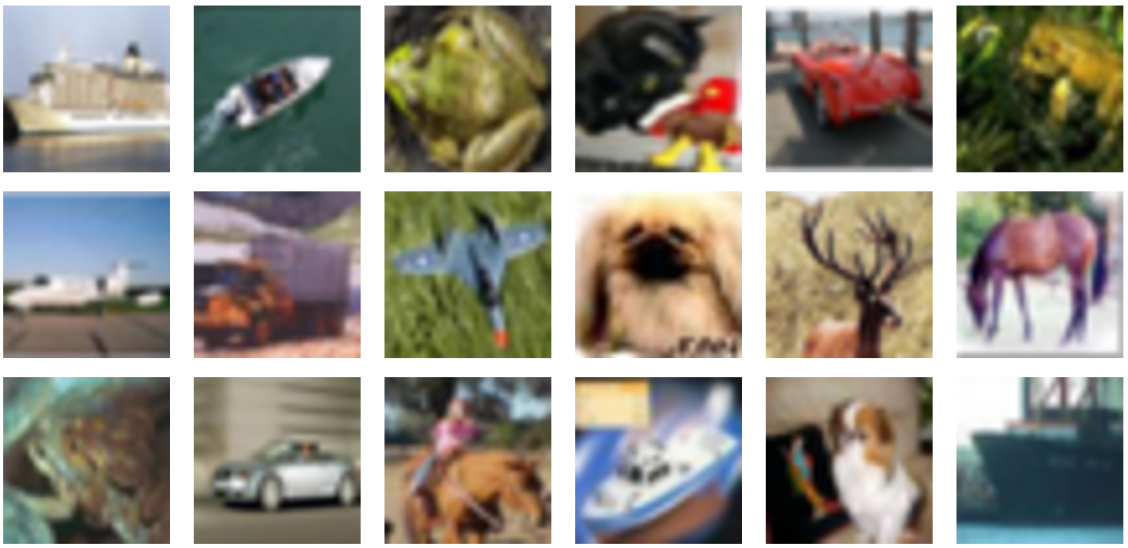


Figure 2.2.: Example of CIFAR-10 images. The images have a resolution of 32×32 RGB color pixels.

We conducted the experiment using a basic CNN described in [Wan+13a]. The CNN has three convolutional layers with max pooling each and two fully connected layers before the softmax prediction for a total of 200K parameters. This CNN is the one used in [ZCL15] experiments. In the first part of our experiments, we focus on the consensus ability of the

different communication strategies. In the second part, we test the convergence rate of different distributed algorithms, namely PerSyn, GoSGD and EASGD with updates driven by natural image distribution.

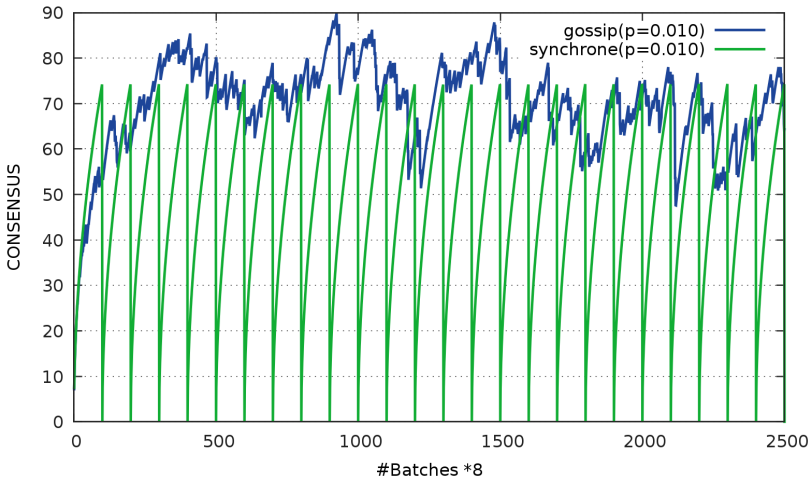
For distributed experiments we always use $M = 8$ workers, loaded on 8 different Tesla K-20 GPU cards with 5Go of RAM each. For all the experiments ρ represents the frequency/probability of communication per batch for one worker. In order to get a fair comparison the methods are always compared at equal frequency/probability of exchange. For all experiments, we use the deep learning framework torch 7 [CBM02], implemented in lua.

2.5.1 Consensus with random updates

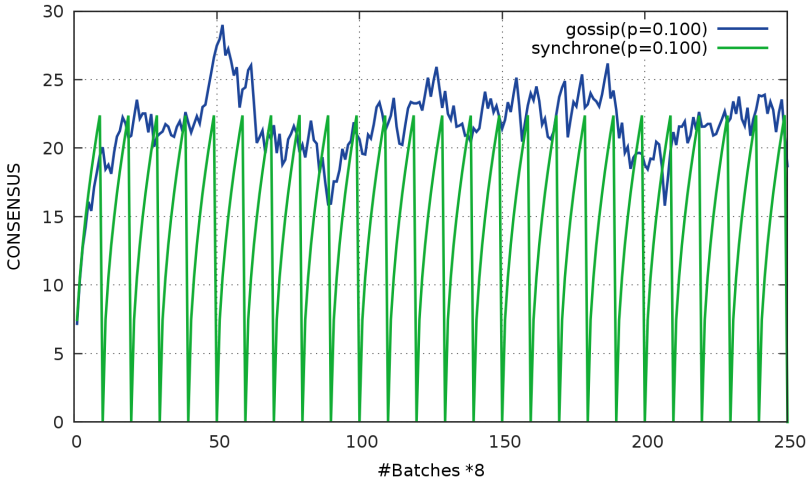
Our framework demonstrates that the convergence rate of different distributed algorithms is tightly related to the quality of the consensus they enable. In this experiment, we test the ability of GoSGD and PerSyn to control a consensus. We consider a worst-case scenario where the local updates are not correlated and as such, we replace the gradient term by a random variable sampled from $\mathcal{N}(0_{dim(\mathbf{w})}, I_{dim(\mathbf{w})})$, independently and identically distributed on each worker. We show on Figure 2.3 the following consensus error for different values of the frequency/probability of exchange ρ:

$$\varepsilon(t) = \sum_{m=1}^{M} \|\mathbf{w}_m^{(t)} - \tilde{\mathbf{w}}^{(t)}\|^2$$

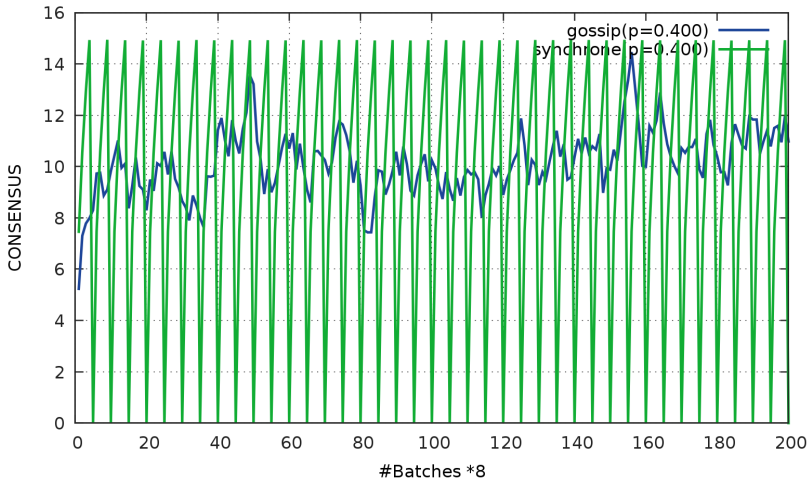
It can be observed on the Figure 2.3c, that for high frequency/probability of exchange ($\rho = 0.4$), GoSGD and PerSyn are equivalent on average. For low frequency/probability $\rho = 0.01$ on Figure 2.3a, the Gossip strategy has a consensus error in the range of the higher values of PerSyn, as both share the same magnitude. The main difference between the 2 strategies is that PerSyn exhibits the expected periodicity in its behavior which leads to big variation of the consensus error, while GoSGD seems to have much less variation. Overall, GoSGD is able to maintain consensus properties comparable to synchronous algorithms despite its random nature and the fact that the number of communication is divided per two for GoSGD for the same probability of communication ρ.



(a) $\rho = 0.01$



(b) $\rho = 0.1$



(c) $\rho = 0.4$

Figure 2.3.: Evolution of the consensus error $\varepsilon(t)$ against SGD ster for different communication frequency/probability ρ for GoSGD (in blue) and PerSyn (in green).

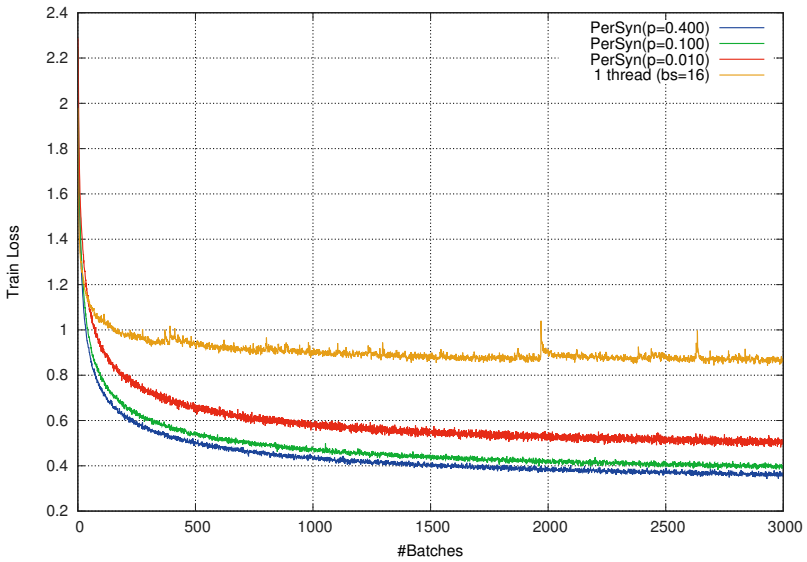
The same experiment has been repeated with noise variables of different variance value. The results are similar and are plotted in Appendix A.3

2.5.2 Training speed and generalization performances

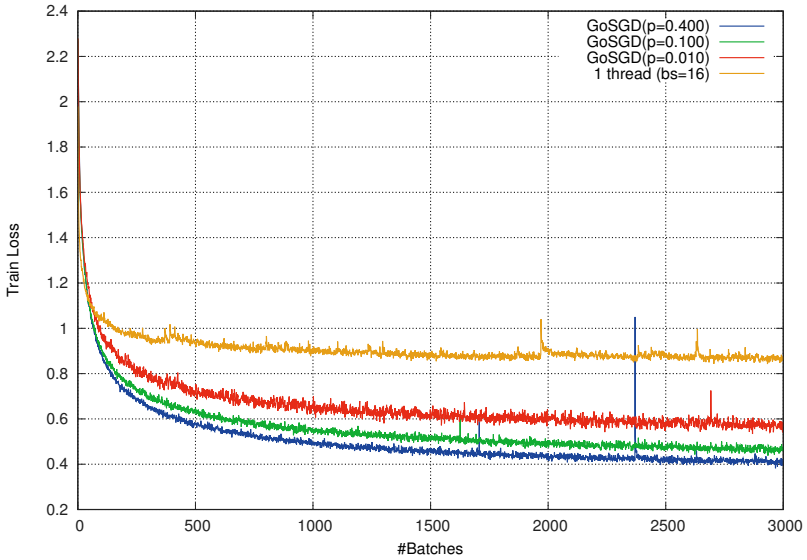
In the following experiments we study the behaviour of the distributed algorithms during training on CIFAR-10. This dataset is composed of 50K images in the training set and 10K images in the test set. They are of resolution 32×32 RBG pixels, split into ten balanced classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Images from the dataset can be find in Figure 2.2 and more information in [Kri09].

We use the same data augmentation strategy as in [ZCL15] by randomly coping a 28×28 image within the 32×32 pixels images and we randomly flip the image horizontally. During training, the learning rate is set to 0.1 and the weight decay is set to 10^{-4}. We still use eight workers.

We show on Figure 2.4 the evolution of the training loss for both PerSyn and GoSGD for different values of the frequency/probability of exchange ρ. As we can see, PerSyn seems to be slightly faster in terms of convergence rate (as measured by the number of iterations required to reach a specific loss value). This is expected as PerSyn seems to ensure a better consensus that GoSGD. However, remember that even not taking into account synchronization issues, PerSyn requires double the amount of message of GoSGD for the same frequency/probability since workers have to wait for the master to answer.



(a) Training loss evolution for PerSyn



(b) Training Loss evolution for GoSGD

Figure 2.4.: Training loss evolution for PerSyn and GoSGD for different frequency/probability of exchange ρ.

Remark. We stress that although the batch size is equal for each worker, the implicit global batch size of the algorithms are not equivalent for all curves. Indeed, the single thread run has a batch size of 16 while each of the 8 workers of the distributed runs has a batch size of 16. Therefore the distributed runs have an equivalent batch size of $16 \times 8 = 128$ as demonstrated in 2.3. These batch sizes are chosen so has to have equal hardware requirements per available machine, which we believe is the most fair and practical use case. In section 2, we recall that having a bigger batch size leads to a much more accurate gradient direction which may significantly speed up the convergence rate. This is indeed observed in Figure 2.4.

Similarly, we compare the training loss of GoSGD with EASGD on figure 2.5 using a real world clock. As we can see, GoSGD is significantly faster than EASGD, which can be explain by the non blocking updates of GoSGD as well as by the fact that GoSGD requires less messages to keep the same consensus error.

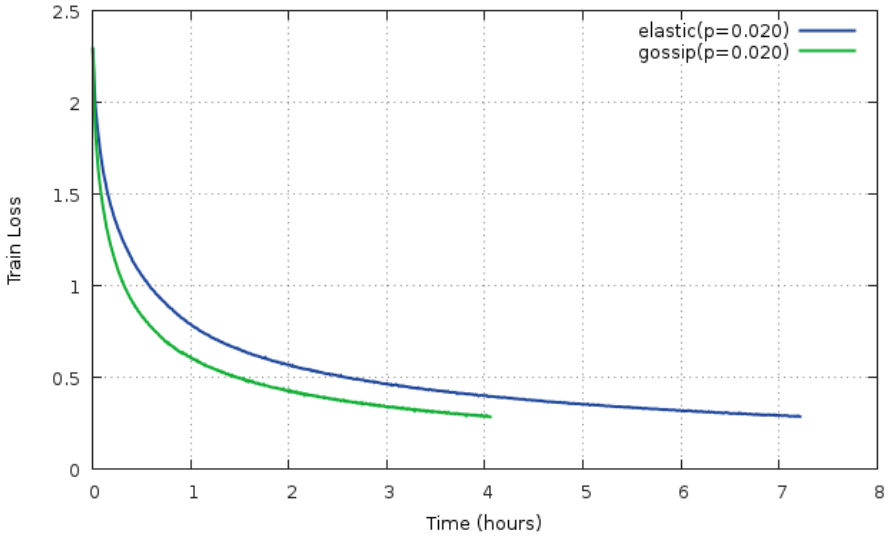
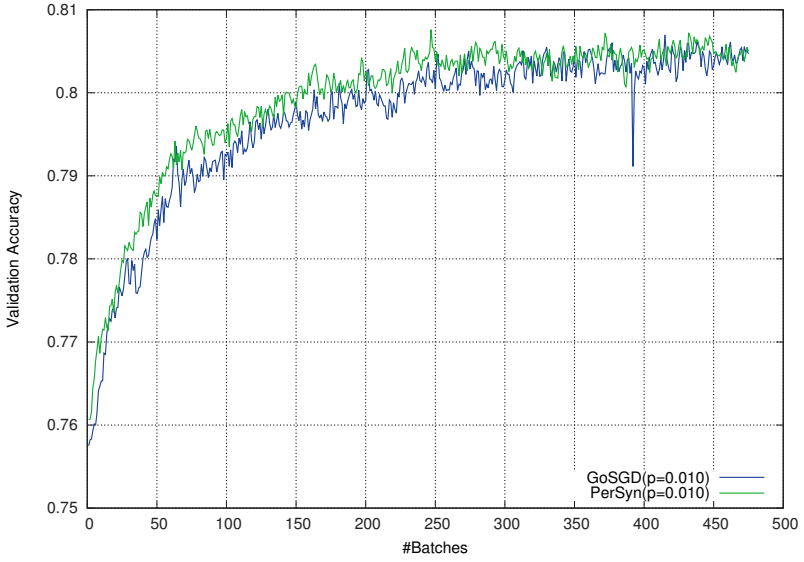


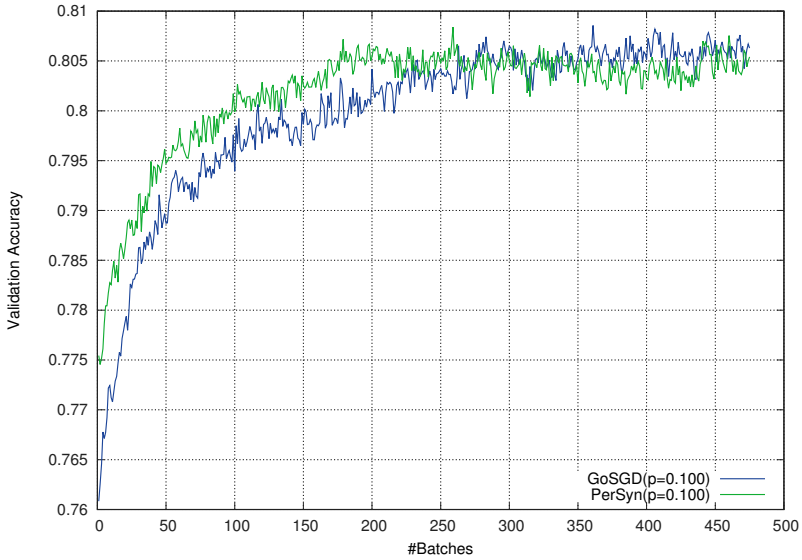
Figure 2.5.: Convergence speed comparison between GoSGD and EASGD for similar frequency/probability of exchange $\rho = 0.02$.

finally, we show on Figure 2.6 the evolution of the validation accuracy for both PerSyn and GoSGD for different values of the frequency/probability of communication ρ. For low frequency $\rho = 0.01$, despite PerSyn being faster in terms on training convergence rate, both PerSyn and GoSGD models obtain equivalent validation accuracy. However, GoSGD is expected to be at least twice as fast in term of real world clock since it uses half the number of messages for the same rate ρ.

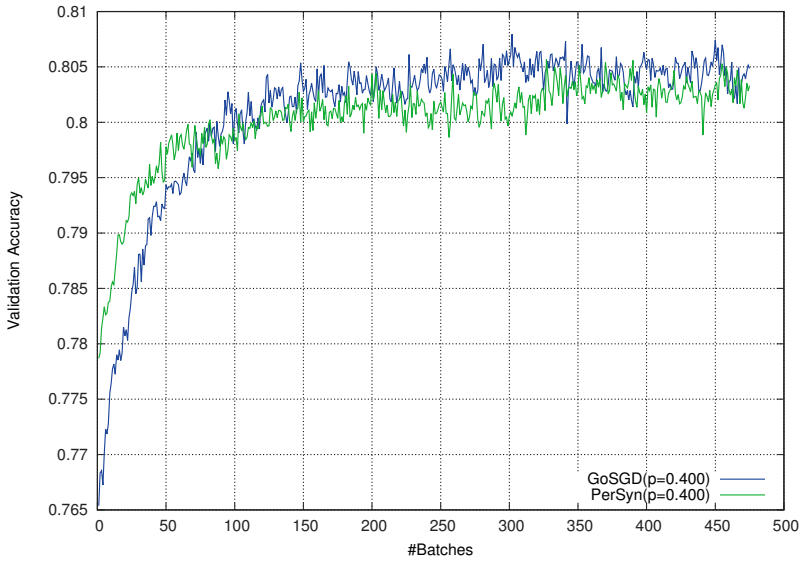
At higher exchange rate $\rho = 0.4$, we can see that GoSGD obtains better validation accuracy than PerSyn, despite having higher training loss, which means that GoSGD is in practice more robust to overfitting. This can be explained by the randomized nature of the gossip exchanges, which perform a stochastic exploration of the parameter space, similarly to what is done by [Wan+13a].



(a) $\rho = 0.01$



(b) $\rho = 0.1$



(c) $\rho = 0.4$

Figure 2.6.: Validation accuracy evolution for PerSyn and GoSGD for different frequency/probability of exchange ρ.

2.6 Conclusion

In this chapter, we have discussed the distributed computing strategies for training deep DNN on large datasets. The matrix framework we have developed, provides two results for the methods it includes:

> The convergence rate of a distributed method is upper bounded by the convergence rate of a single threaded SGD with bigger batches.
>
> The convergence rate of a distributed method is related to the consensus between workers enabled by its communication strategy.

From these insights, we are able to compare different strategies by studying their communication matrix only, and more specifically their capacity to maintain a consensus, like was done in Section 2.5.1. There is indeed, a crucial trade-off between low communication costs and sufficient communication to ensure that all workers contribute to the same variable optimization.

The two methods we propose are compared in term of consensus quality and in terms of convergence rate on an image dataset. Our experiments confirm the fact that a higher consensus guarantees a higher convergence rate. However, these conclusions cannot be reported on the validation error and further studies have to be done in this direction.

Further more, the consensus properties of GoSGD could be further improved helped with the well developed theoretical gossip framework. Never the less, the peer to peer communication protocols propose in GoSGD permits efficient and scalable applications of distributed SGD.

CHAPTER 3

UNDERSTANDING AND IMPROVING CNN ARCHITECTURE

Contents

Chapter abstract

In this chapter, we study the influence of the DNN architecture on its generalization performance. More specifically, we discuss the question of invariance and information filtering. With the intention to use convolutional layers that filter less information, we propose a new activation function. The positive results obtained in our experiments when replacing the ReLU validate the intuition that the ReLU filtering is sub-optimal.

The work in this chapter has led to the publication of a conference paper:

- *MaxMin convolutional neural networks for image classification*, Michael Blot and Matthieu Cord and Nicolas Thome, IEEE International Conference on Image Processing (ICIP), 2016

3.1 Introduction

In the previous chapter, we have presented distributed methods to accelerate the training of DNNs. Reducing the training time is important to ease the study of deep learning on

very big datasets. Another motivation for developing distributed methods is the trend in DNN architecture to get deeper, which is a factor of slowness for the SGD. The reason is that deeper architecture implies higher time and memory consumption to compute the loss gradients. Besides, in image classification, empirical results tend to validate that deep architectures generalize better than shallow ones. This is why, since AlexNet, state of the art DNN architectures never ceased to get deeper despite the negative effect on the training time. This highlights on the prevalence of the architecture in DNN performance. The importance of the architecture is also illustrated in [Zha+17], where it is shown that different architectures enjoy different generalization properties even if they are all able to make zero error on the training data.

However, augmenting the depth of DNNs has limitations. The difficulty to train very deep architecture limits the depth and a DNN with substantial memory and computation complexity such as vgg [Kar15], is hard to exploit. Improving deep learning generalization through architectures without augmenting much DNN complexity remains a matter of concern and seems feasible. For instance, on computer vision tasks, CNNs have demonstrated generalization performance that has never been attained by MLPs before, even for equivalent computation complexity, number of parameters, and depth.

Above all, in CNN architectures, the activation function may be questioned. With ReLU, all negative activations from the convolutional map are set to zero. By introducing a non linear operation, this activation increases the representational power of the CNN but is also a source of information filtering. Studying on the nature of this unexploited information, we remark that ignoring it for the prediction can be somehow sub-optimal. With further inspection, we assume that strong negative activations contain discriminant information about the image class. Considering the possibility to exploit it, we propose a new activation function, that is one to one, meaning that there is no loss in the information processing. We call this activation function **MaxMin** because of its implication with pooling layers that is presented later in this chapter. Our experiments show that MaxMin activation enables to improve the CNN generalization performance at constant depth and equivalent number of parameters.

The remaining of this chapter is as follows. In the next section, we discuss related work about DNN architecture and more specifically, we focus on invariance by filtering information to introduce our new activation function. Next, in Section 3.3, we motivate our new activation function and present its implementation along with various discussions about its implication in a CNN. In Section 3.4, we present experiments demonstrating the benefits of using MaxMin instead of ReLU activation function before we conclude the chapter.

3.2 Related work

State of the art architectures in image classification are pretty standard. They are composed of a succession of convolutional layers with ReLU activations. Optionally a pooling layer and/or a local contrast normalization [KSH12] follow the convolution operations. Then, come some fully connected layers before the final Softmax activation. We present here, attempts to optimize the DNN architecture in order to improve deep learning performance.

3.2.1 Designing DNN for representability and invariance

Since AlexNet, the studies that are responsible for improving deep learning performances mainly focus on the expressivity of CNNs. Other approaches, which focus instead on making the architectures invariant, are also presented below.

Representation power. As already mentioned and pointed out in [He+16], the depth of the architecture has a positive effect of the DNN generalization. The reason for such observations is still unclear and many researches are trying to decipher the influence of the depth parameter [NN15; JSO18; Aro+18b; He+16]. Nevertheless, very deep DNNs are difficult to optimize, even though they have higher expressive power, as noticed by [He+16]. To alleviate this drawback, two new characteristics on the architecture seem today mandatory to permit very deep DNN optimization. The first one is the use of residual operations [He+16] for the convolutional layers. The second one is the use of a layer wise normalization of the feature maps, called batch normalization [IS16]. The implementation of both alternatives do not impact the expressiveness of the DNN but modify positively its gradients and enables efficient optimization with SGD. Note that the batch normalization is also responsible for improvement in the generalization error, probably due to implicit stochasticity. In fact, in the article [IS16], the authors explain that the mini-batches estimations of the neurons' variance and expected value is noisy. The resulting stochasticity in the normalization could increase the robustness of the trained DNN.

On another hand, [ZK16] studies the influence of the layers' width. The width defines the numbers of filters in a convolutional layer, which is also an important parameters of the representability of the architecture. They show that the widths also have an important impact on the DNN generalization performance and should be optimized accordingly.

Regarding the convolutional filters themselves, improvement have been done by modifying the size of their support. For instance, vgg uses a filter window size of 3×3 for all its convolutional layers. For comparison, AlexNet first convolutional layer has filters of size 11×11. Small filters necessitate few parameters to optimize but are less expressive than bigger ones. However, vgg counterbalances this diminution by using much more filters than AlexNet, increasing the layers' width. Another strategy is tested in GoogLeNet [Sze+15], that adopts convolutional filters of different sizes within a same layer. This particularity enables to capture patterns of different scales more easily. Moreover, following [LCY14] implementation, this architecture uses convolutional filters with higher expressivity than standard linear filters.

Those studies mainly imply the representability of the architecture. It seems important to consider a DNN architecture with sufficient expressiveness to get satisfying generalization performances as illustrated in [Zha+17]. This particularity is in contradiction with many machine learning intuitions and theoretical analysis which recommend models with small representative power [NY72; BLM02]. This machine learning guidance is thus helpless to improve deep learning performance. From a different perspective, in order to obtain good generalization performances, another classical guideline is to design invariant models. Being invariant to natural transformations can also guarantee an architecture to have satisfying generalization.

Invariance. A natural transformation denotes an alteration of the input due to natural variations of the objects in the image which do not change its class. Those variations can affect the color of objects, their position, the viewpoint of the image like the distance to the objects or the angle of shooting, the brightness of the scene, etc. We recall that a model is invariant to a transformation if it outputs the same value, being provided an input or its transformation. Considering the limited number of training sample, a model invariant to many natural transformations will be able to generalize on unexplored area of the image distribution.

This motivates the work of [MBC03; Che+18], who have imagined an architecture invariant to local object or texture rotations in the images by allowing filters rotations. Other researches concerned about invariant architecture can be found in [Mal12], proposing scattering networks, inspired by wavelet transforms. Obtaining architectures invariant to intuitive transformations, such as local object rotation or scale changes, is not straightforward. The task is even harder for complex transformations that are difficult to model. To ease the research about invariant architectures, some efforts have been done to model the set of natural transformations under a symmetric group formalism [Ans+18; Bor15]. However, the application to deep learning remains unexplored.

Another approach to obtain invariant architectures is to use layers that eliminate information appropriately. Next section presents some precedent strategies to filter information and make DNNs invariant.

3.2.2 Filtering and pooling

We state that a layer filters information if it is not one to one on the distribution support of its input. This way, for some of the layer's output, it is not possible to recover the input the output has been computed from. Such a layer induces invariance to the DNN architecture but also reduces its representability power. However, the invariance has to be to natural transformation. If it is not the case, there is a risk that the filtered information could have been discriminant for the prediction. Within classic architectures, the association of convolutional filters with ReLU and the pooling layers are responsible for filtering information and are discussed below.

Pooling layers. In standard CNNs, the pooling layer is responsible for filtering local spatial information. It summarizes into a single value, the information contained within a small window. The pooled windows are densely spread on the feature maps, as described in [LeC+98]. Within a DNN, such a layer provides inviance to local transformations on the image, such as local elasticity of objects, small translation of the image, etc. In modern architectures, **max pooling**, that returns the maximum value of the pooled window, is the most popular pooling function which has proved to be very efficient on image applications. This operation filters a lot of information, because, except from one well chosen neuron value from the pooled window, all the information is deleted by the pooling operation.

Other pooling strategies that would filter less information have been studied. For instance, the average pooling which returns the average value of the window, is not performing as well as max pooling when tested on image classification. In [BPL10], they investigate theoretical reasons for such a difference. In the article, they also propose a new parametric pooling

function, that generalizes average and max pooling operations. By varying parameters they can find a balance between max pooling, that filters a lot of spatial information, and average pooling that aggregates the spatial information. However, they did not find better results than with max pooling. In [Sze+15] on the contrary, they report that the average pooling is performing better when applied on the last feature maps of the DNN, right before the fully connected layer. In the presented architecture, this last pooling layer considers the whole feature maps as pooling windows. Another pooling strategy for the last feature maps, introduced in [DTC16], proposes a pooling function that is somehow between max and average pooling. This methods has demonstrated impressive results on image classification and object detection, and is state of the art on several datasets.

Those pooling examples illustrate the difficulty to balance the filtering of information within the DNN architecture. Selecting layers that filter a lot of information is efficient to make the DNN invariant and help to generalize on the whole image distribution. On the other hand, it is important to exploit as much discriminant information as possible, to obtain accurate predictions.

Convolutionnal filters and activation function. Regarding the convolutional layers, a convolutional filter is due extract a specific discriminant information from the image or from the precedent feature maps. Before transmitting the detection value to the following layer, the convolutional filter is complemented by a ReLU activation function. This function is supposed to eliminate the information from negative detections, by setting them all to zero. The intuition behind this activation function, is that a convolutional filter acts like a detector which useful information remains in positive detections only. This ReLU filtering is supposed to facilitate the exploitation of discriminative information, by de-noising the convolutional detections.

In this configuration, a strong negative detection provides a similar information as a weak negative detection. Indeed, they are both set to zero by the ReLU. However, there is good chances that those highly negative values contain discriminant information about the class and should be exploited in the prediction process. With this intention, [He+15] introduces the PReLU as an extension of ReLU. This function has two linear parts; the identity function on the positive values; and a learned linear mapping on the negative values. This activation function does not reduce the representative power of the DNN because the linear part on negative values can be set to zero. Popular variant of the PReLU, transmitting information about the negative detection, are the Leaky ReLU [Xu+15], the ELU [CUH15], and the recent SELU [Kla+17].

Concerning these activation functions, one can wonder if scaling negative values differently from positive values makes sens. Moreover, with those strategies, the transmitted detection is treated the same way by the following layer, whether it is negative or positive. Considering that the sign of the detection can provide semantic information, these functions can be inappropriate for image classification.

In this chapter, we also study the possibility to transmit information about the negative detections to following layers. However, to alleviate the described issues of PReLU and its variants, we adopt a different strategy by considering an architecture allowing both positive and negative activations to be transmitted in different channels. Next section further presents the MaxMin activation function.

3.3 MaxMin CNN

We explore in this chapter a different strategy to improve convolutional layers by studying alternative activation functions. We propose to keep high positive and high negative values obtained by the convolutional filter in a double ReLU scheme (MaxMin) that we detail in the following. Similar researches have been conducted concurrently in [Sha+16], resulting in the same activation function, named CReLU.

3.3.1 Motivation

We consider the intuition that linear convolutional filters are well designed to detect geometrical patterns, especially for low level layers. In this case the semantic information provided by a detection is mainly about a shape or a texture. Moreover, learned shapes and textures usually are invariant to the sign inversion of the detection. Unfortunately, the ReLU filters all negative detection of the patterns, removing the possibility to transmit information about the negative version of the patterns. This particularity could drive the CNN optimization to learn a filter, and its negative version in order to complete the information about the pattern.

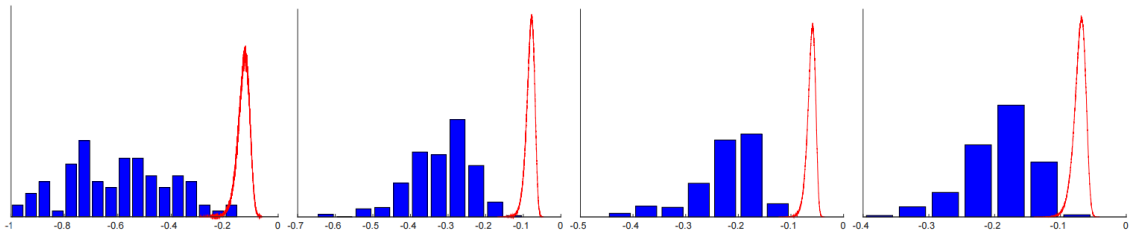


Figure 3.1.: From [Sha+16]. Histograms of minimum negative match between filters of a same layer. In blue are the histogram for filters learned by AlexNet CNN, and in red histogram for random filters. For conv1, the distribution of AlexNet filters- is negatively centered, which significantly differs from that of random filters, whose center is very close to zero. The center gradually shifts towards zero when going deeper into the network.

This intuition is confirmed with the observation in [Sha+16], who report that the learned filters of AlexNet present unexpected negative correlations. For every filters of a layer, the experiment consists in computing its scalar product with other filters (of the same layer) and selecting the minimum value obtained. On Figure 3.1, is plotted the histograms of the described value for the first four layers of a trained AlexNet. The histograms are compared to the ones obtained with random filters. Clearly, the filters learned by AlexNet present a negative biased correlation confirming the intuition that a CNN with ReLU activation has to learn two versions of a filter. The phenomena seems to fade on higher level layers (layer 4), for which the patterns detected become more abstract and less geometrical.

By transmitting to following layers the information about the negative detections, a CNN could avoid this redundancy in the learned filters. We present in next section our strategy to exploit the negative values obtained by the convolutional filters.

3.3.2 MaxMin function

In the following, we note the convolutional operation on $\mathbf{x}$ of the individual filter h, $h * \mathbf{x}$. Our method aims at transmitting directly the negative detections from the convolutional filter h in order to prevent the DNN to learn the opposite filter h^-. Indeed, when convoluted with any input $\mathbf{x}$, h^- verifies, $h^- * \mathbf{x} = (-h) * \mathbf{x} = -(h * \mathbf{x})$.

Then, if the pattern filtered by h^- is strongly detected on $\mathbf{x}$, $h^- * \mathbf{x}$ will be high and positive while $h * \mathbf{x}$ will be symmetrically low and negative. Thus, conserving the information from the convolution with h, either the result is positive or negative, will provide the information about h^- and enables to use less filters in the architecture.

To implement our strategy, we duplicate the convolutional filter maps (represented is blue in Fig. 3.2) and multiply the copy by $\times -1$ resulting in the negative version of the detections (in red in Fig. 3.2). We then concatenate the original maps, and their negative copies as shown on Fig. 3.2. This operation increases the width of the convolutional layer's output by two compared to classical CNN. We then, commonly apply the ReLU to the concatenated output and optionally process a pooling operation. This way, negative values are not filtered and can be exploited independently by following layers.

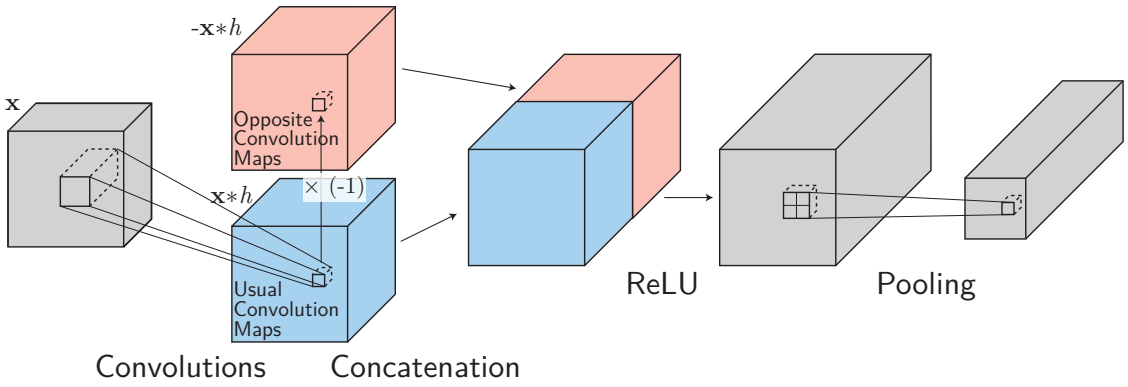


Figure 3.2.: MaxMin scheme. After the convolutional layer, the resulting maps (in blue) are duplicated negatively (in red). Both stacked maps are concatenated to get the MaxMin output that will pass through the ReLU before pooling.

This new activation function can be written $MaxMin(x) = \big(ReLU(x), ReLU(-x)\big)$ and is one-to-one, permitting to the global convolutional layer to filter less information from the input.

3.3.3 Relation with max pooling

In the case a max pooling operation is applied after the convolutional layer, our following analysis demonstrates that an equivalent architecture can be found by only modifying the pooling function. The resulting pooling provides more information to the next layer than the classical max pooling as demonstrated here. Let first remark that $ReLU \circ max = max \circ ReLU$. It is possible to commute the pooling layer and the activation layer. Regarding our method, we notice that

$$\Big(max(ReLU(\mathbf{x})),\ max(ReLU(-\mathbf{x}))\Big) = \Big(ReLU(max(\mathbf{x})),\ ReLU(-min(\mathbf{x}))\Big)$$

where $\mathbf{x}$ is a vector. Thus, our method can be interpreted as a new bi-dimensional pooling function, that is applied before the activation function, and that returns the maximum and the minimum of the pooled window, the two values being anti-correlated. Using MaxMin activation with max pooling is thus equivalent, at using instead the ReLU activation with a different pooling function, that provides more information about the local window.

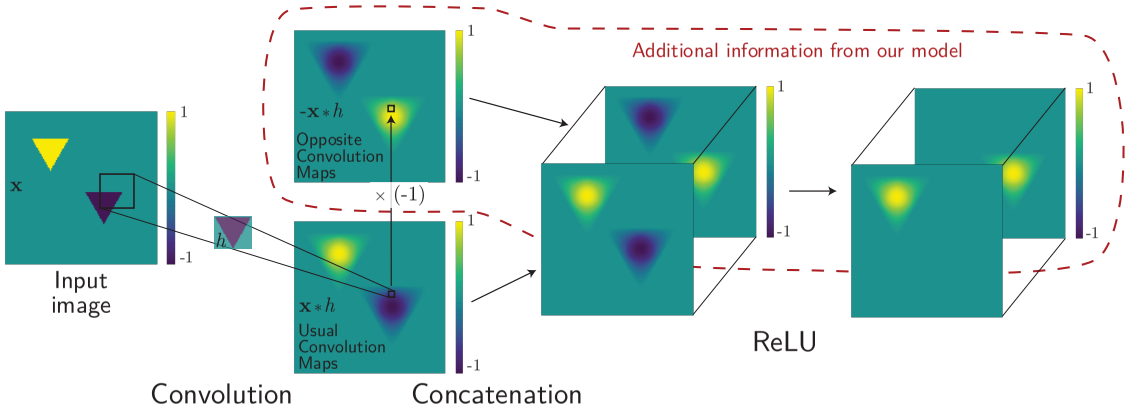


Figure 3.3.: MaxMin CNN block for one input image with specific patterns. When considering an input with 2 specific triangles and a convolution with the filter h, we obtain a map with 2 strong positive and negative answers. Thanks to our MaxMin, we get additional information (red dotted line) that makes it possible to keep both extreme information after ReLU in a joint double map.

3.3.4 Discussion

Here we discuss the method in term of modeling and generalization.

Number of parameters. Compared to classical ReLU, MaxMin doubles the size of the output. Consequently, it necessitates to double the size of the following convolutional filters. This has the immediate effect of doubling the number of parameters of the following layer. It is worth mentioning that, if those additional parameters are all set to zero the CNN is equivalent to a CNN with ReLU and our method does not reduce the CNN expressivity.

Moreover, the intuition behind MaxMin activation prescribes to use half the number of filters, canceling this dimensional effect. After a convolutional layer, MaxMin activation enforces the transmission of a filter detection along with the detection of the negative version of the same filter. If the negative version was learned with standard CNNs, it is not necessary anymore with MaxMin. This is illustrated in Figure 3.3, were the MaxMin activation transmits the information of the presence of a triangle with a single filter, whether the triangle is yellow or blue. The same information requires two filters to be transmitted by a standard CNN with ReLU activations. It seems then reasonable, when using MaxMin activations, to reduce the number of filter of the convolutional layers in order to control the number of parameters.

Generalization. On a spatial window of a feature map, the maximum and the minimum pooling outputs are rarely simultaneously positively high and negatively low. Therefore, the ReLU filters very often one of the two values (min or max). This ensures a sparse activation of the CNN's neurons. This property is known to enhance the generalization performance of DNN in computer vision as studied in [OF96; Goh+13]. Moreover, we claim that our method is able to learn more efficiently the convolutional filters than standard CNN. MaxMin method enforces discriminant patterns to be learned by gradient descent on the error function from both positive detections and negative ones. Learning a pattern from more occurrences enables to both learn faster and generalize better.

3.4 Experiments

In our experiments, we test our method on two well-known datasets, MNIST and CIFAR-10, and we compare our results with classical CNN.

Learning protocol. All the models and the learning framework are implemented in Lua using Torch 7 library [CBM02]. For the training, we use the same momentum SGD as in [KSH12] with 0.9 of momentum and a fixed weight decay (10^{-3} for MNIST and 10^{-4} for CIFAR). We use an equal learning rate for all layers starting at 0.1, that is being manually reduced (divided by 10) when validation error stops decrease. All weights of convolutional filters are initialized from a zero-mean Gaussian distribution with standard deviation 0.01. The number of training epochs depends on the CNN but is between 30 and 120 for CIFAR-10 and around 250 for MNIST. Top-1 accuracies on test sets are computed to evaluate performances.

3.4.1 MNIST

MNIST is a 60,000 images dataset, containing handwritten versions of the nine digits. Images are of 28×28 pixels resolution with one color channel (see [LeC+98] for more information). We use the same protocol as [DS12]: 50,000 images for training and the 10,000 remaining images are used for testing. To compare our method, we use a LeNet like CNN. It is composed of three convolutional layers with ReLU activation function, all followed by a max pooling and a local contrast normalization layers [KSH12]. The filters are of size 5×5 with a stride of 1 pixel, for the convolutional steps. There is 64 of them in each convolutional layer. The pooling windows are of size 3×3 for a stride of 2. Those layers are followed by one fully connected layer, projecting the output onto the class space of dimension ten, before a Softmax is applied. For the MaxMin DNN setting, we simply replace ReLUs with MaxMin activation functions. We keep the same number of filters paying attention to double their size when needed.

The baseline network obtains a score of 99.34% accuracy. With our MaxMin DNN, we obtain an accuracy of 99.41%. Our strategy has 29 errors on the validation set against 36 errors for the baseline CNN. It corresponds to a relative improvement of 13.9%. Note that the performance scores on this dataset are very high, and any small performance gain is difficult to obtain.

3.4.2 CIFAR-10

In this section we present experiments on the image dataset CIFAR-10 [Kri09], presented in Chapter 2 experiments Section 2.5. The baseline CNN is the one implemented in [VL15] where we have replaced the average pooling layers with max pooling. The CNN has three convolutional layers with $ReLU$ activation, all followed by max pooling. All filters have size 5×5 and stride 1 with 32 filters for the first two layers and 64 for the last one. The pooling windows have size 3×3 and stride 2 for all layers. The last pooling layer is followed by two fully connected layers before a Softmax is applied.

The accuracy score obtained using this simple CNN is 74.44% on the test set. When we train an equivalent MaxMin network, we reach an accuracy of 78.62%, showing an improvement of more than 4% on the test error.

Robustness to parameter number. As mentioned earlier, our MaxMin activation imposes to double up the size of following filters compared to standard CNNs. For a fixed number of filter, this implies to increase the number of parameters when including MaxMin activation to CNN. As the number of parameters directly impacts the performances of a CNN, we now compare both activation functions at constant number of parameters. We use the previous CNN and modify the number of filters on different convolutional layers in order to vary the total number of parameters. For the MaxMin CCN, we adapt the number of filters to get comparable amount of parameters as the standard CNN. As we intend to compare the quality of the exploitation of the convolutional filters information by CNN using different activation functions, we pay attention to get the same amount of neurons in the fully connected layers. Only the number of convolutional filters varies from a CNN to another. We report in table 3.1 the accuracies obtained on the test set for different

# Parameters	MaxMin-CNN	Simple-CNN
$\approx 30K$	73.81	69.98
$\approx 15K$	78.13	74.44
$\approx 60K$	80.03	77.01
$\approx 2M$	81.68	78.11
$\approx 5M$	82.07	79.48
$\approx 15M$	82.69	80.13
$\approx 45M$	82.98	80.41

Table 3.1.: CIFAR-10 results for simple CNN and our MaxMin CNN. Accuracy scores are reported for different architectures distinct by their number of parameters. MaxMin CNN has systematically better performances than simple CNN for equal complexity (# Parameters).

numbers of parameters for both standard CNN and CNN using MaxMin. We observe that the MaxMin CNN systematically outperforms the standard CNN whatever the number of parameters, with a best performance of 82.98%. This shows the robustness of our method to different network parametrization and demonstrates the ability of MaxMin to better exploit convolutional filter information.

Boosting performances. Current deep models on CIFAR-10 use several kind of learning tricks such as data augmentation, image preprocessing, or dropout, to improve final classification performances.

Our MaxMin CNN may benefit of the same tricks. As the learning becomes quite more complex and much more time consuming, we do perform only once the optimization for our MaxMin architecture on CIFAR-10. We thus apply some translations and flips on the images for data augmentation, a zca whitening referenced in [Goo+13] to preprocess the input, and dropout on the fully connected layer to boost the performance. We use the same CNN described above with additional local contrast normalization layer after each pooling operation.

With this setup, we obtain the accuracy of 90.03% on the test set. This score beats the results reported by [KSH12] that uses a deeper architecture (89%) and the ones reported by [DS12] that uses 8 networks for prediction (88.79%). On the contrary, our MaxMin CNN scores are obtained with only one network.

3.5 Conclusion

The MaxMin activation we propose in this chapter, transmits on different channels, the positive and negative activations values from the convolutional maps. It permits to following layers to exploit the discriminant information from patterns detected negatively by convolutional filters, avoiding some redundancies in the learned parameters. We evaluate and compare our strategy with classical architectures using ReLU, on two benchmarks CIFAR-10 and MNIST. Results demonstrate that our MaxMin CNNs perform better than standard

CNNs. Architectures using MaxMin may be less invariant and filter less noise, but exploit better the discriminant information from an input to make a robust prediction.

In order to generalize our results, it would be interesting to test MaxMin activation on different architectures and on large scale tasks. Besides, more experiments can be found in [Sha+16], where the activation function is tested on ImageNet. Still, our experiments permit to highlight the difficulty to balance the information filtering in DNN. In fact, the question of information extraction and invariance remains unanswered in the context of deep learning. For instance, while pooling layers and ReLU activations are still present in state of the art architectures and seem important to provide invariance to the architectures, [JSO18] intends to demonstrate that filtering information before the class projection layer, preceding the Softmax, is not necessary to obtain good generalization performance.

In the next chapter, we will address this question of information filtering and invariance, but we rather focus on the parameters of a fixed architecture. We will adopt an information theory point of view, in order to find DNN parameters with good generalization properties.

CHAPTER 4

INFORMATION-BASED REGULARIZATION FOR DEEP LEARNING

Contents

Chapter abstract

In this chapter, we address the question of regularization in deep learning. Studies on generalization performance of machine learning algorithms under the scope of information theory suggest that compressed representations can guarantee good generalization. With this guidance, many regularization methods, aimed at compressing neural networks intermediate representations were proposed. We present here a new information based criterion to minimize: the entropy of the representation variable conditioned to the class variable. This criterion is due to influence the optimization toward compressed and invariant representations. We also propose a tractable implementation of this criterion that incorporates easily on the SGD process. We test our regularization on several datasets with various architectures. We report consistent improvements compared to standard regularizations.

The work in this chapter has led to the publication of a conference paper:

- *SHADE: SHAnnon DEcay Information-Based Regularization for Deep Learning*, Michael Blot and Thomas Robert and Nicolas Thome and Matthieu Cord, IEEE International Conference on Image Processing (ICIP), 2018, Best paper award

4.1 Introduction

In the previous chapter, we have talked about the influence of the architectures on deep learning generalization performance. Still in the context of image classification, we address, in this chapter, the question of improving the generalization performance for a DNN of a given architecture.

The objective function of a DNN is a non-convex function of the DNN parameters and it holds several local minima. Moreover, it was demonstrated in [Kes+16], that distinct local minima obtaining similar loss values on the train set, can show different behaviors on the test set. To compare those minima, they introduce a measure to quantify how flat is a local minima. According to their measure, they show that flatter minima generalize better. They also present experiments showing that this measure is tightly correlated with the batch size used during the training. Consequently, the training method has an impact on the generalization of the optimized DNN. Even though [Din+17] has proved that sharp minimum can also generalize well, making the condition on the measure of [Kes+16] unnecessary, it opens the way to explore new regularization methods. The ambition is to influence the optimization toward local minima, that present a smaller generalization gap.

However, the difficulty to exhibit predominant factors impacting DNNs generalization, complicates the task to study new regularization methods. In fact, despite constant progress since [KSH12], DNN generalization ability is still largely misunderstood. As illustrated in [Zha+17], theoretical tools such as PAC based analysis seem limited to help identifying the characteristics in DNNs that explain deep learning good performance.

Information theory has been applied successfully on many research areas, and recently, its application to generalization theory is winning interest [AM17; SST10]. For instance, several attempts to interpret deep learning performance are using information theory tools [NN15; AS17; GG17].

In this chapter, we study a regularization scheme based on a new information theory criterion. For a machine learning model, the entropy of its intermediate representation variable, conditionally to the class variable, is the criterion we propose to apprehend the generalization potential of a model. This criterion instantiates the **Minimum Description Length Principle** [Ris78], a machine learning interpretation of the Occam Razor.

As information measures are usually difficult to estimate when the number of data is low, compared to the size of the variable support space, we propose an implementation of a tractable loss. This loss, called SHADE for SHAnnon DEcay, has proven to reduce our criterion when minimized. It has the advantage to be layer-wise and more particularly neuron-wise.

The remaining of this chapter is as follow. The next Section 4.2 presents the relevant literature about regularization and information theory in deep learning. Then, we present how we develop the loss function SHADE and instantiate it in Section 4.3. Finally, in Section 4.4 we demonstrate the relevance of our criterion and our loss with experiments on several datasets, with various architectures before we conclude in Section 4.5.

4.2 Related work and motivation

We recall the notations used for this chapter. $X \in \mathcal{X}$ is the input variable, $C \in \mathcal{C}$ is the class variable and $\mathbf{w}$ is the model set of parameters giving $Y = h(\mathbf{w}, X)$ the (deep) representation of the input leading to the class prediction. Y is naturally a random variable depending on the distribution of X. SHADE criterion is noted $H(Y \mid C)$, with H standing for the Shannon entropy measure (see [CT91] for definition).

In the following, we present the relevant literature about DNN regularization and generalization.

Regularization in deep learning. For classification tasks, DNN training schemes usually use an objective function which linearly combines the expectancy of a classification loss $\ell_{\text{cls}}(\mathbf{w}, X, C)$ –generally cross-entropy– and a regularization term $\Omega(\mathbf{w}, X, C)$, that aims at influencing the optimization toward a local minimum with good generalization properties:

$$\mathcal{L}(w) = \mathbb{E}_{(X,C)}\big[\ell_{\text{cls}}(\mathbf{w}, X, C) + \beta \cdot \Omega(\mathbf{w}, X, C)\big] \tag{4.1}$$

with $\beta \in \mathbb{R}^+$. For example, the weight decay (WD) loss [KH92] is supposed to penalize the complexity of the model through the euclidean norm of its weights. There is strong motivations to use WD on linear models, in term of margins [Vap98] or in term of Lipschitz coefficient. WD is also equivalent to a Bayesian prior on the training [GG17]. However, the extension of those interpretation to deep learning is not straightforward and the effects of WD on DNN's generalization performances is still not clear as demonstrated in [Zha+17]. Other regularization methods like [ZLL16], add a reconstruction cost to the classification loss. This strategy is essentially applied on semi-supervised tasks, when not all labels are available.

Our work adopts a similar strategy, as we intend to construct a loss $\Omega_{SHADE}(h(\mathbf{w}, X), C)$, that will be added to the classification objective function like in Equation 4.1. The regularization loss aims at influencing the optimization toward representations with low, class conditioned, entropy. We show in the experiments, that SHADE loss has a positive effect on our theoretical criterion $H(Y \mid C)$, resulting in enhanced generalization performances.

Other popular regularization approaches, like [Hin+12; Wan+13b], randomly deactivate part of the neurons during the training, by setting their values to zero. These methods, which intend to lower the dependency of the prediction to a reduced set of features, are backed by some theoretical interpretations like [AS16a; GG17; AS16b]. Other practices, such as batch normalization [IS16] or stochastic poolings [DF13; Ben14], induce stochasticity to the training. Beside being comparable to sophisticated data-augmentation, these strategies result in the addition of noise to the gradients during optimization. This property would enable the model to converge toward flatter local minimum, that are known to generalize well as shown in [Kes+17] and observed in our experiments in Chapter 2. More generally, stochastic methods tend to make the learned parameters less dependant on the training data, which guarantee tighter generalization bounds [AM17]. In this chapter, we focus on another strategy to make deep learning models less dependant on the training data. By focusing on representation that are invariant to many input transformations, the training becomes less

dependant on the training set. Favoring invariant representation is the motivation for our criterion $H(Y \mid C)$.

Quantifying invariance. Designing DNN models that are robust to variations on the training data and that preserve class information is the main motivation of this work. In the same direction, Scattering decompositions [Mal12] are appealing transforms. They have been incorporated into adapted network architectures like [BM13]. However, as mentioned in Chapter 3, for tasks like image recognition, it is very difficult to design an explicit modelling of all transformations a model should be invariant to.

Differently, a measure such as $H(X|Y)$ is suitable to quantify how globally invariant is the model's representation. More specifically, it is agnostic to the transformations the representation is invariant to. Indeed, a model that is invariant to many transformations will produce the same representation for different inputs and inversely. It is then impossible to guess the input that has produced a given representation value. When trying to reconstruct the input from its representation, a high invariance means a high reconstruction error, even for the best strategy. Inequalities such as Fano's inequality [CT91], demonstrate that this reconstruction error is lower bounded by a decreasing function of the measure $H(X|Y)$ (A general discussion illustrated with new bounds on the reconstruction error with $H(X \mid Y)$ can be found in Appendix B.1). The higher is $H(X|Y)$, the higher is the reconstruction error. Indeed, $H(X \mid Y)$ represents the uncertainty in the variable X knowing its representation $Y = h(\mathbf{w}, X)$ and is therefore, a good measure of invariance.

Regarding the entropy of the representation variable $H(Y)$, when X is discrete and for a deterministic mapping $Y = h(\mathbf{w}, X)$, we obtain:

$$H(Y) = I(X,Y) + \underbrace{H(Y \mid X)}_{=0 \text{ (determ.)}} = I(X,Y) = H(X) - H(X \mid Y). \tag{4.2}$$

$H(X)$ being fixed by the problem, $H(Y)$ is inversely related to $H(X \mid Y)$ making $H(Y)$ also an equivalent good measure of invariant. This explains the search for compressed and sparse representations in [OF96; Goh+13; HMD15] and for pruning methods [LW17] for regularization.

Latter, considering the targeted classification task, we do not want two inputs of different classes to have the same representation but rather want to focus on intra-class invariance. Therefore, all this reasoning should be done conditionally to the class C, explaining final choice of $H(Y \mid C)$[1] as a measure of intra-class invariance.

Information-theory-based regularization. Many works like [Per+17; AS17] use information measures as regularization criterion. Still with the objective of making the trained model less dependant on the data, [AS17] builds a specific weight regularization. However the regularization resorts to a prior on weight distribution which could be discussed. The information criterion that is closer to ours is the one defined in the Information Bottleneck framework (IB) proposed in [TPB99], that suggests to use mutual information $I(X,Y)$ ([CT91])[2] as a regularization criterion. [Ale+17] extends it in a variational context, VIB,

1 $H(Y \mid C) = H(X \mid C) - H(X \mid Y, C)$

2 for all variable X, Y, $I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$

by constructing a variational upper-bound of the criterion which is minimized by the training. Along with IB also come some theoretical investigations, with the definition of generalization bounds in [SST10]. Using mutual information as a regularizer is also closely connected to invariance since $I(X, Y)$ attempts at compressing as much information as possible from input data. In the case X is discrete and the representation mapping is deterministic ($H(Y \mid X) = 0$), our criterion is related to IB's trough the following development $I(X, Y) = H(Y) = I(C, Y) + H(Y \mid C)$. In a context of optimization with SGD, minimizing $H(Y \mid C)$ appears to be more efficient to preserve the term $I(C, Y)$, which represents the mutual information of the representation variable with the class variable and that must stay high to predict accurately C from Y.

Compressing the representation, without damaging the class information seems in fact to be a holy grail in machine learning. Our work, resulting in SHADE, goes in this direction.

4.3 SHADE: A new Regularization Method Based on $H(Y \mid C)$

In this section, we will further describe SHADE, a new regularization loss based on the conditional entropy $H(Y \mid C)$, designed to drive the optimization toward more invariant representations. We first show how to derive a layer-wise, neuron-wise criterion before developing a proper tractable upper bound of the neuron-wise criterion. In order to properly develop entropy inequalities it is necessary to suppose that X is a discrete variable of the space $\{0, ..., 255\}^{\mathcal{H} \times \mathcal{W} \times 3}$ with $\mathcal{H}$ and $\mathcal{W}$ respectively the height and width of the images. However it is common to consider X as a discrete quantization of a continuous natural variable, enabling to exploit some properties verified by continuous variables such as gradient computing[3].

3 Regarding information measures, a discussion on this topic can be found in [AG18]

4.3.1 A neuron-wise criterion

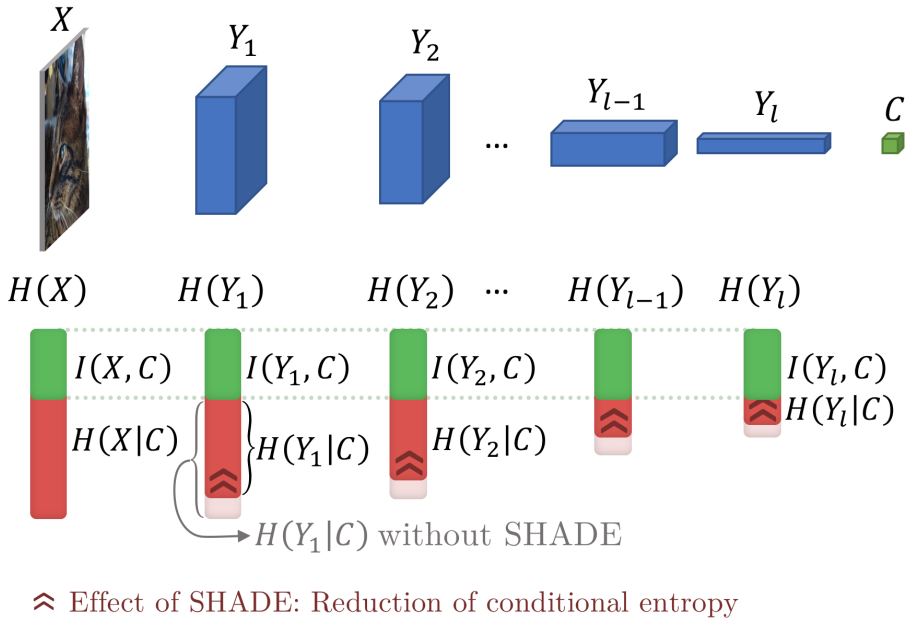


Figure 4.1.: DNN architecture with corresponding layers' entropies, showing the layer-wise action of SHADE. Given that $H(Y_\ell) = I(Y_\ell, C) + H(Y_\ell \mid C)$, SHADE minimizes $H(Y_\ell \mid C)$ without affecting $I(Y_\ell, C)$.

Layer-wise criterion. A DNN is composed of a number L of layers that sequentially transform the input. Each layer output can be seen as an intermediate representation variable, noted Y_ℓ for layer $\ell \in \{1, .., L\}$, that is determined by the output of the previous layer $Y_{\ell-1}$ and a the layer set of parameters $\mathbf{w}_\ell$: $Y_\ell = h_\ell(Y_{\ell-1}, \mathbf{w}_\ell)$. Each layer filters out a certain part of the information from the initial input X. Thus, the following inequalities can be derived from the data processing inequality in [CT91]:

$$H(Y_L \mid C) \leq H(Y_{L-1} \mid C) \leq ... \leq H(Y_1 \mid C) \leq H(X \mid C). \tag{4.3}$$

The conditional entropy of every layer is upper bounded by the conditional entropy of the previous layer. Similarly to the recommendation of [NN15], we apply this regularization to all the layers, using a layer-wise criterion $H(Y_\ell \mid C)$, and producing a global criterion[4]:

$$\Omega_{\text{layers}} = \sum_{\ell=1}^{L} \beta_\ell H(Y_\ell \mid C). \tag{4.4}$$

Where β_ℓ is a weighting term differentiating the importance of the regularization between layers. Those coefficient will be omitted in the following as in our experiments, all β_ℓ are identical. Adjusting the values of the variables β_ℓ remains a lead for further research. This

4 Confirming the intuition, in our experiments, regularizing all layer has proved to be more efficient than regularizing the final layer representation only

layer wise action is illustrated in Fig. 4.1 where we see that $I(Y_l, C)$ (in green) remains constant while $H(Y_l \mid C)$ (in red) decreases.

Neuron-wise criterion. Examining one layer ℓ, its representation variable is a random vector of coordinates noted $Y_{\ell,i}$ and of dimension D_ℓ: $Y_\ell = (Y_{\ell,1}, ..., Y_{\ell,D_\ell})$. The classical upper bound $H(Y_\ell \mid C) \leq \sum_{i=1}^{D_\ell} H(Y_{\ell,i} \mid C)$ enables to define a neuron-wise criterion that SHADE seeks to minimize. For each unit i of every layer ℓ we design a loss $\Omega_{\text{unit}}(Y_{\ell,i} \mid C) = H(Y_{\ell,i} \mid C)$ that will be part of the global regularization loss:

$$\Omega_{\text{layers}} \leq \sum_{l=1}^{L} \sum_{i=1}^{D_\ell} \underbrace{H(Y_{\ell,i} \mid C)}_{\Omega_{\text{unit}}(Y_{\ell,i}|C)}. \tag{4.5}$$

For the rest of the chapter, since the layers and coordinates are all considered independently to define $\Omega_{\text{unit}}(Y_{\ell,i} \mid C)$, we use the notation Y instead of $Y_{\ell,i}$ for simplicity.

4.3.2 Estimating the Entropy

In this section, we describe how to define a loss based on the measure $H(Y \mid C)$ with Y being one coordinate variable of one layer output. Defining this loss is not obvious as the gradient of $H(Y \mid C)$ with respect to the layer's parameters may be computationally intractable. Y has an unknown distribution and without modeling it properly it is not possible to compute $H(Y \mid C)$. Since $H(Y \mid C) = \sum_{c \in \mathcal{C}} p(c) H(Y \mid c)$, a naive approach would consist in computing $|\mathcal{C}|$ different entropies $H(Y \mid c)$, $1 \leq c \leq |\mathcal{C}|$. This means that, given a mini-batch, the number of samples used to estimate one of these entropies is divided by $|\mathcal{C}|$ on average. This becomes particularly problematic when dealing with a large number of classes such as the 1,000 classes of ImageNet. Furthermore, entropy estimators are extremely inaccurate considering the number of samples in a batch. For example, LME estimators of entropy in [Pan03a] converge in $\mathcal{O}((\log K)^2/K)$ for K the number of samples. Finally, most estimators require to discretise the space in order to approximate the distribution *via* a histogram. This raises issues on the bins definition considering that the variable distribution is unknown and varies during the training in addition to the fact that having a histogram for each neuron is computationally and memory consuming. Moreover, entropy estimators using kernel density estimation [KLS11; SZS05], or nearest neighbors methods [Pan03b], usually have a too high complexity ($\approx \mathcal{O}(K^2)$) to be applied efficient on deep learning models.

To tackle these drawbacks we propose the two following solutions: the introduction of a latent variable that enables to use more examples to estimate the conditional entropies; and a bound on the entropy of the variable by an increasing function of its variance to avoid the issue of entropy estimation with a histogram, making the computation tractable and scalable.

Latent code. First, considering a single neuron Y (before ReLU or Softmax), the ReLU activation induces a detector behavior toward a certain pattern. If the pattern is absent from the input, the signal is around zero; if it is present, the activation quantifies the resemblance

with the pattern. We therefore propose to associate a binomial variable B with each unit variable Y (before ReLU). This variable B indicates if a particular pattern is present on the input ($B = 1$ when $Y \gg 0$) or not ($B = 0$ when $Y \leq 0$). B acts like a latent code from which the input X is generated like in graphic models (*e.g.* [KW14]) or in generative models (*e.g.* [Che+16]).

Furthermore, it is very likely that most intermediate features of a DNN can take similar values for inputs of different classes – this is especially true for low-level features. The semantic information provided by a feature is thus more about a particular pattern than about the class itself. Only the association of features by the following layer, allows determining the class. So B represents a semantically meaningful factor about the class C. The feature value Y is then a quantification of the possibility for this semantic attribute B to be present in the input or not. We thus assume the Markov chain $C \to B \to X \to Y$. Indeed, during the training, the distribution of Y varies in order to get as close as possible to a sufficient statistic of X for C (see definition in [CT91]). Therefore, we expect B to be such that Y draws near a sufficient statistic of B for C as well. By assuming the sufficient statistic relation $I(Y, C) = I(Y, B)$ we get the equivalent equality $H(Y \mid C) = H(Y \mid B)$, and finally obtain:

$$H(Y \mid C) = H(Y \mid B) = \sum_{b \in \{0,1\}} p(b) H(Y \mid B = b). \tag{4.6}$$

This modeling of B as a Bernoulli variable (one for each unit) has the advantage of enabling good estimators of conditional entropy since we only divide the batch into two sets for the estimation ($b = 0$ and $b = 1$) regardless of the number of classes. The fact that most of Y information about C is contained in such a variable B is validated in the experiments Section 4.4.4.

Variance bound. The previous trick allows computing fewer entropy estimates to obtain the global conditional entropy, therefore increasing the sample size used for each entropy estimation. Unfortunately, it does not solve the bin definition issue. To address this, we propose to use the following bound on $H(Y \mid B)$, that does not require the definition of bins:

$$H(Y \mid B) \leq \frac{1}{2} \ln \big(2\pi e \mathbb{V}\mathrm{ar}(Y \mid B)\big). \tag{4.7}$$

This bound holds for any continuous distributions on Y and there is equality if the distribution is Gaussian. For many other distributions such as the exponential one, the entropy is also equal to an increasing function of the variance. In addition, one main advantage is that variance estimators are much more robust than entropy estimators, converging in $\mathcal{O}(1/K)$ for K samples instead of $\mathcal{O}(\log(K)^2/K)$ for entropy estimators. The use of this bound is well justified in our case because the variable Y is the quantization of a continuous variable. Moreover, even if Y is discrete, this inequality still holds with respect to a constant term depending on the quantization steps.

Algorithm 5 Moving average updates: for $b \in \{0,1\}$, p^b estimates $p(B = b)$ and μ^b estimates $\mathbb{E}(Y \mid B = b)$

Initialize: $\mu^0 = -1$, $\mu^1 = 1$, $p^0 = p^1 = 0.5$, $\lambda = 0.8$
for each mini-batch $\{y^{(k)}, k \in 1..K\}$ **do**
  for $b \in \{0,1\}$ **do**
    $p^b \leftarrow \lambda p^b + (1-\lambda)\frac{1}{K}\sum_{k=1}^{K} p(b \mid y^{(k)})$
    $\mu^b \leftarrow \lambda \mu^b + (1-\lambda)\frac{1}{K}\sum_{k=1}^{K} \frac{p(b \mid y^{(k)})}{p^b} y^{(k)}$
  end for
end for

The ln function being one-to-one and increasing, we only keep the simpler term $\mathbb{V}\mathrm{ar}(Y \mid B)$ to design our final loss:

$$\Omega_{\text{SHADE}} = \sum_{\ell=1}^{L}\sum_{i=1}^{D_\ell}\sum_{b\in\{0,1\}} p(B_{\ell,i} = b \mid Y_{\ell,i})\mathbb{V}\mathrm{ar}(Y_{\ell,i} \mid B_{\ell,i} = b). \tag{4.8}$$

In the next section, we detail the definition of the differential loss, computed on a mini-batch, using $\mathbb{V}\mathrm{ar}(Y \mid B)$ as a criterion.

4.3.3 Instantiating SHADE

For one neuron of one layer, the previous criterion writes:

$$\mathbb{V}\mathrm{ar}(Y \mid B) = \int_{\mathcal{Y}} p(y) \sum_{b\in\{0,1\}} p(b \mid y)\big(y - \mathbb{E}(Y \mid b)\big)^2 \,\mathrm{d}y \tag{4.9}$$

$$\approx \frac{1}{K}\sum_{k=1}^{K}\left[\sum_{b\in\{0,1\}} p(b \mid y^{(k)})\big(y^{(k)} - \mathbb{E}(Y \mid b)\big)^2\right]. \tag{4.10}$$

The quantity $\mathbb{V}\mathrm{ar}(Y \mid B)$ can be estimated with Monte-Carlo sampling on a mini-batch of K input-target pairs $\left\{(x^{(k)}, c^{(k)})\right\}_{1\le k\le K}$ of intermediate representations $\left\{y^{(k)}\right\}_{1\le k\le K}$ as in Eq. (4.10).

$p(B \mid y)$ is interpreted as the probability of presence of attribute B on the input, and should clearly be modeled such that $p(B = 1 \mid y)$ increases with y. The more similarities between the input and the pattern represented by y, the higher the probability of presence for B. We suggest using[5]:

$$p(B = 1 \mid y) = 1 - e^{-ReLU(y)} \qquad p(b = 0 \mid y) = e^{-ReLU(y)}.$$

For the expected values $\mu^b = \mathbb{E}(Y \mid b)$, we use a classic moving average that is updated after each batch as described in Algorithm 5. Note that the expected values are not modified by the optimization since translating a variable has no influence on its entropy.

5 Other functions have been experimented with similar results

The concrete behavior of SHADE can be interpreted by analyzing its gradient as described in Appendix B.2.

For this proposed instantiation, our SHADE regularization penalty takes the final form:

$$\Omega_{\text{SHADE}} = \sum_{\ell=1}^{L} \sum_{i=1}^{D_\ell} \sum_{k=1}^{K} \sum_{b\in\{0,1\}} p\left(B_{\ell,i} = b \,\middle|\, y_{\ell,i}^{(k)}\right) \left(y_{\ell,i}^{(k)} - \mu_{\ell,i}^{b}\right)^2. \tag{4.11}$$

We have presented a regularizer that is applied neuron-wise and that can be integrated into the usual optimization process of a DNN. The additional computation and memory usage induced by SHADE is almost negligible (computation and storage of two moving averages per neuron). For comparison, SHADE adds half as many parameters as batch normalization does.

4.4 Experiments

All our experiments have been implemented on the framework TensorFlow [Aba+16].

4.4.1 Image Classification with Various Architectures on CIFAR-10

Table 4.1.: Classification accuracy (%) on CIFAR-10 test set.

	MLP	AlexNet	Inception	ResNet
No regul.	64.68	83.25	91.21	92.95
Weight decay	66.52	83.54	92.87	93.84
Dropout	66.70	85.95	91.34	93.31
SHADE	**70.45**	**85.96**	**93.56**	**93.87**

We perform image classification on the CIFAR-10 dataset. Following [Sze+14] and [ZK16], we used the following architectures:

- Mlp: it is a multi-layer perceptron described in [Zha+17], composed of three fully connected layers with ReLU activations.
- AlexNet: this architecture is an AlexNet like model. It possesses three convolutional layers of 64 filters of window size 3×3 and stride 1. All convolutional layers are associated to a max pooling operation with window of size 3×3 and stride 2. The architecture ends with two fully connected layers, with intermediate representation of 1000 neurons.
- **Inception**: this architecture is an inception like model described in [Sze+14].
- **ResNet**: We also use a ResNet architecture from [ZK16] (k=10, N=4).

Those architectures have been selected because they represent a large family of DNN and some have been well studied in [Zha+17] and [ZK16] within the generalization scope.

They also illustrate the evolution of architectures in image classification, with some of them approaching the state of the art on CIFAR-10. Contrarily to the precedent chapter, our objective is not to study architectures. We are rather interested in the results of the use SHADE relatively to other regularization for different family of architectures.

We compare SHADE with two popular regularization methods, namely *weight decay* (WD)[KH92] and *dropout* [Hin+12]. For all architectures, the regularizations parameter has been cross-validated to find the best ones for each method and the obtained accuracies on the test set are reported in Table 4.1.

We obtain the same trends as [Zha+17], which shows a small improvement of 0.29% with weight decay on AlexNet. The improvement with weight decay is slightly more important with ResNet and Inception (0.79% and 1.66%). In our experiments dropout improves significantly generalization performances for AlexNet and MLP only. It is known that the use of batch normalization and only one fully connected layer for classification, lowers the benefit of dropout, which is in fact not used in [He+16].

We first notice that for all kind of architectures, the use of SHADE significantly improves the generalization performances, 5.77% for MLP, 2.71% for AlexNet, 2.35% for Inception and 0.92% for ResNet. It demonstrates the ability of SHADE to regularize the training of deep architectures. Finally, SHADE shows better performances than dropout and weight decay on all architectures.

4.4.1.1 Experiments details on CIFAR-10

We provide here the detailed protocol for the CIFAR-10 experiments.

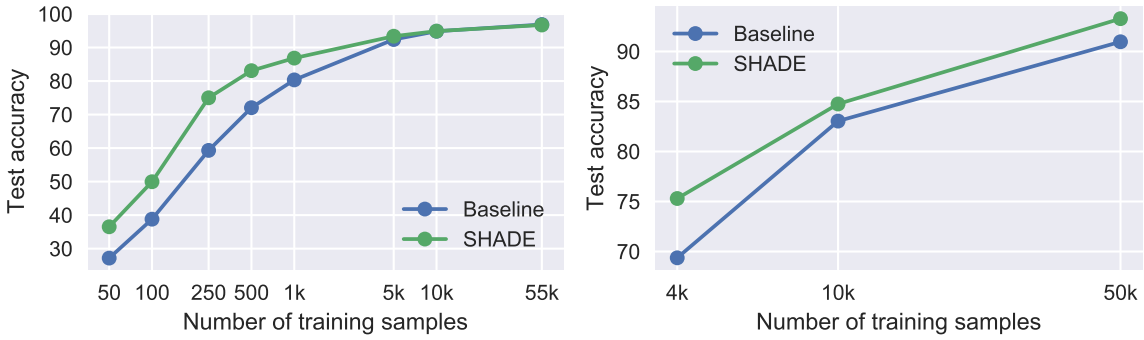
Images. For training, we use randomly cropped images of size 28×28 with random horizontal flips. For testing, we simply center-crop 28×28 images.

Optimization. We use momentum SGD for optimization (momentum = 0.9). For all experiments the learning rate is initialize and a multiplicative decay is apply on it after every batches. We detail here the initial learning rate and the decay for every architecture used in the format (initial learning rate value, decay): mlp (0.1, 0.99), alexnet (0.1, 0.99), inception (0.8, 0.98), resnet(0.7, 0.97).

Hyperparameter tuning For WD and SHADE, the optimal regularization weight of each model has been chosen to maximize the accuracy on the validation sets. We tried the values $\{[1, 5].10^{-i}, i = 1..7\}$. For the dropout we have apply it on the two last layer of every networks. The optimal activation probabilities for each model has been chosen among $\{(n/10, m/10), n = 1..7,\ m = 3..10\}$ to maximize the accuracy on the validation sets.

4.4.2 Large Scale Classification on ImageNet

In order to experiment SHADE regularization on a large scale dataset, we train on ImageNet [JDe+09] a WELDON network from [DTC16] adapted from ResNet-101. This architecture changes the last pooling strategy by using the network in a fully-convolutional way and adding a max+min pooling, thus improving the performance of the baseline network.



(a) Results for MNIST-M

(b) Results for CIFAR-10

(c) Examples of MNIST-M images misclassified by the baseline and correctly classified using SHADE, both trained with 250 samples.

Figure 4.2.: Results when training with a limited number of samples in the training set for MNIST-M and CIFAR-10 with and without SHADE.

The fine tuning in the experiment has been done with momentum-SGD with a learning rate of 10^{-5} and a momentum of 0.9 and a batch size of 16 images. It took 8 epochs to converge.

We used the **pre-trained weights of ResNet-101** (from the torchvision package of PyTorch) giving performances on the test set of **77.56%** for top-1 accuracy and 93.89% for top-5 accuracy. Provided with the pre-trained weights, the **WELDON architecture** obtains **78.51%** for top-1 accuracy and 94.65% for top-5 accuracy. After fine tuning the network using **SHADE** for regularization, we finally obtained **80.14%** for top-1 accuracy and 95.35% for top-5 accuracy for a concrete improvement. This demonstrates the ability to apply SHADE on very large scale image classification tasks successfully.

4.4.3 Training with a Limited Number of Samples

When datasets are small, DNN tend to overfit quickly and regularization becomes essential. Because it tends to filter out information and make the network more invariant, SHADE seems to be well fitted for this task. To investigate this, we propose to train DNN with and without SHADE on CIFAR-10 and MNIST-M with different numbers of samples in the training set.

First, we tested this approach on the digits dataset MNIST-M [GL15]. This dataset consists of the MNIST digits where the background and digit have been replaced by colored and textured information (see Fig. 4.2c for examples). The interest of this dataset is that it contains lots of unnecessary information that should be filtered out, and is therefore well adapted to measure the effect of SHADE. A simple convolutional network has been trained with different numbers of samples of MNIST-M and the optimal regularization weight for SHADE have been determined on the validation set (see training details in the following

sub section 4.4.3.1). The results can be seen on Figure 4.2a. We can see that especially for small numbers of training samples (< 1000), SHADE provides an important gain of 10 to 15% points over the baseline. This shows that SHADE helped the model in finding invariant and still discriminative patterns using less data samples.

Additionally, Figure 4.2c shows samples that are misclassified by the baseline model but correctly classified when using SHADE. These images contain a large amount of intra-class variance (color, texture, etc.) that is not useful for the classification tasks, explaining why adding SHADE, that encourages the model to discard information, allows important performance gains on this dataset and especially when only few training samples are given.

Finally, to confirm this behavior, we also applied the same procedure in a more conventional setting by training an Inception model on CIFAR-10. Figure 4.2b shows the results in that case. We can see that once again SHADE helps the model gain in performance and that this behavior is more noticeable when the number of samples is limited, allowing a gain of 6% when using 4000 samples.

4.4.3.1 Experiments details on MNIST-M

Dataset splits and creation. To create MNIST-M, we kept the provided splits of MNIST, so we have 55,000 training samples, 5,000 validation samples, and 10,000 test samples. Each digit of MNIST is processed to add color and texture by taking a crop in images from BST dataset. This procedure is explained in [GL15].

Subsets of limited size. To create the training sets of limited size N, we keep $N/10$ (since there are 10 classes) randomly picked samples from each class. When increasing N we keep the previously picked samples so the training samples for $N = 100$ are a subset of the ones for $N = 250$. The samples chosen for a given value of N are the same across all models trained using this number of samples.

Image preprocessing. The only preprocessing applied to the input images is that their values are rescaled from $[0, 1]$ to $[-1, 1]$.

Optimization. For the training, we use mini-batch of size 50 and use Adam optimizer with the recommended parameters, *i.e.* $\lambda_r = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$.

Hyperparameter tuning. For weight decay and SHADE, the optimal regularization weight of each model (for each value of N) has been chosen to maximize the accuracy on the validation sets. We tried the values $\{10^{-i}, i = 1..7\}$.

Model architecture. The model have the following architecture:

- 2D convolution ($64 \times 5 \times 5$ kernel, padding 2, stride 1) + ReLU
- MaxPooling 2×2
- 2D convolution ($64 \times 3 \times 3$ kernel, padding 1, stride 1) + ReLU
- MaxPooling 2×2

- 2D convolution ($64 \times 3 \times 3$ kernel, padding 1, stride 1) + ReLU
- MaxPooling 2×2
- Fully connected (1024 inputs, 10 outputs) + SoftMax

4.4.4 Further experiments: exploration on the latent variable

Table 4.2.: Classification accuracy (%) on CIFAR-10 test set with binarized activation.

MLP	baseline	layer 3	layer 2	layer 1
	64.68	64.92	62.45	61.13
AlexNet	baseline	last 4	layer 2	layer 1
	83.25	82.71	82.38	82.01
Inception	baseline	layer 21	layer 10	layer 1
	91.34	91.41	90.88	90.21
ResNet	baseline	layer 56	layer 25	layer 1
	93.24	92.67	92.09	91.99

SHADE is based on the intuition that the class information encoded within a neuron is mostly contained in a latent Bernoulli variable B. To justify this assumption we propose the following experiments on pre-trained DNN's neuron variables.

For one chosen layer of a pre-trained DNN, we replace the ReLU activation function with a binary activation function. The used binary function is $\overline{Y^+}.\mathbb{1}(Y \geq \overline{Y^+})$ where $\overline{Y^+}$ stands for the average value of the positive part of the neuron detection (before activation function). After replacing the activation function, we fine tune the weights of the layers coming after the chosen layer, in order to adapt the top of the DNN to the new activation values. The experiments is done on CIFAR-10 dataset with the same architectures used in Section 4.4.1 and we report the obtained accuracies on the Table 4.2 for different chosen layers.

We observe that the accuracies do not differ much with ReLU and with the binary activation. This means that for a given layer, most of the information used for the prediction can be summarized in a binary variable. This validates the existence of a latent binary code for neurons activation, that contains the class information. The observation is valid for every layers of the tested DNNs, approving the layer wise application of SHADE. Further researches on this binary activation could help to better model the variable B.

4.5 Conclusion

Regarding deep learning generalization, the criterion we propose in this chapter aims at compressing the intermediate representations variables of a DNN, without impacting the information about the class. In fact, there is close relationship between compressed variables and invariant representation. The good behavior on all our experiments of the instantiation SHADE consolidates the relevance of the entropy criterion.

The overall method could be further studied and improved from many angles, which makes it an exciting topic for future researches. First, regarding the criterion, it should be theoretically studied in the context of deep learning. The way, the information is encoded and exploited within a DNN should be deciphered in order to understand how our criterion influences the DNN generalization. On the other hand, the efficiency of SHADE instantiation can be improved by optimizing the model. Further studies could help to better approximate the conditional entropy and permit the regularization loss to minimize more efficiently the criterion. More particularly, the distance between the criterion and SHADE bound remains unclear. For instance, the Markov chain hypothesis on the binary variable has not been challenged much and it would be interesting to consider the dependencies between neurons of a same layers to better estimate the representation variable entropy.

Finally, the novelty of our approach opens the way to many researches and improvement of the method. The next chapter is an example. Indeed, we study there the same criterion, applied on a slightly different variable that is more pertinent to regularize. Moreover, to bound our entropy criterion by a tractable loss, we resort to a variation framework.

CHAPTER 5

REGULARIZING DEEP REPRESENTATION ENTROPY WITH VARIATIONAL BOUNDS

Contents

Chapter abstract

In this chapter, we introduce a new regularization scheme, still based on information measures. Noting that compressing the representation variable can be sub-optimal, our first focus is to identify a variable in the representation that is really responsible for the final class prediction. We describe this selective variable for a DNN. Similar to SHADE, we aim at compressing the class conditioned entropy of this latter variable. To do so, we introduce a variational upper bound on this conditional entropy term. Finally, we propose a scheme to instantiate a tractable loss that is integrated within the training procedure of the DNN. We demonstrate the efficiency of our method on different families of CNN for different datasets.

The work in this chapter has led to the application paper under review at AAAI:

- *Reve: Regularizing deep learning with variational entropy bound*, Antoine Saporta, Michael Blot and Matthieu Cord, under review at AAAI 2019

5.1 Introduction

In the previous chapter, we have investigated on a regularization methods aiming at minimizing the entropy of layers representation variable conditionaly to the class variable. This criterion is instantiated as a layer wise regularization loss, SHADE, that has proved to improve DNN generalizations.

In this chapter, we investigate a different approach, that differs from SHADE in three aspects:

- *Single layer regularization:* First, in the new method, we focus on regularizing the last representation variable, just before the projection on the class space (before the last fully connected layer + Softmax). This specific variable is noted Y in all the chapter. Regularizing on the penultimate layer of the DNN enables to lower the complexity of the regularization.

- *Identifying the predictive variable:* Moreover, we study a criterion that slightly differ from SHADE's one. In fact, in the regularization, we propose to ignore the part of Y information that is filtered out by the final fully connected layer. We study the other part, noted Z, and build our regularization framework around it. As Z is responsible for the prediction, it also conditions the generalization power of the DNN. This is why we claim that it is more efficient to directly compress Z instead of Y which is somehow sub-optimal. We still focus on the class conditioned entropy, but of the selected new variable: $H(Z \mid C)$.

- *Variational bound:* To minimize our criterion, we study a variational upper bound on the entropy measure explaining the name of our method: Reve for REgularizing with Variational Entropy bound. This upper bound enables to define a loss function that will be minimized during the optimization.

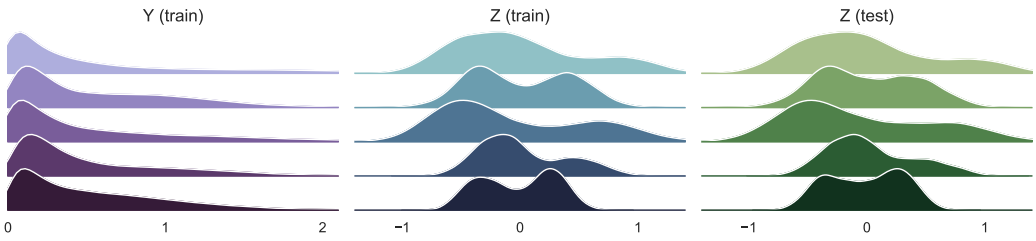
In order to instanciate the loss function, it is necessary to establish a model for Z. However, the quality of the model only impacts the tightness of the variational bound and never invalidates it, contrarily to SHADE, which model can alter the Markov chain hypothesis. In particular, for Reve, we adopt a bimodal approximation for modelling the variable Z, following the empirical observations.

In order to validate our instancitation and the new criterion, we extensively experiment the regularization loss on different DNN architectures and datasets, to demonstrate the capacity of the method to improve DNNs generalization.

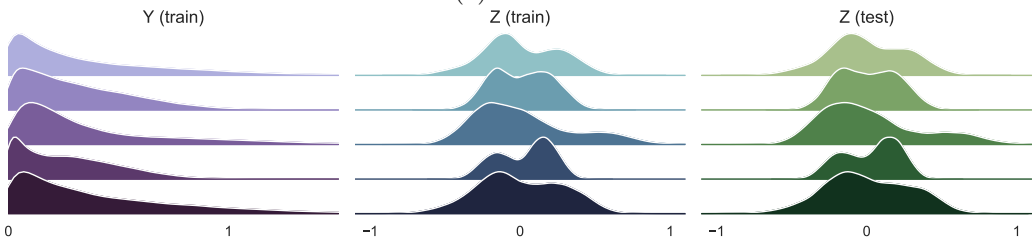
The chapter is organized as follow: Section 5.2 describes Reve objective function, necessitating to define Z for a DNN and to develop variational upper bounds on our criterion. Section 5.3 instantiates our objective function as a tractable loss. Section 5.4 presents the experiments. We conclude in Section 5.5.

5.2 Reve Regularization

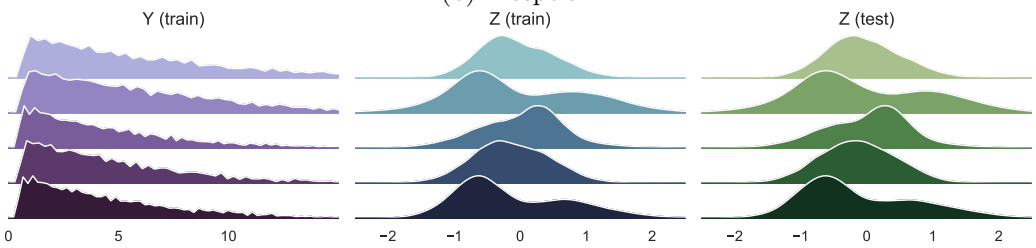
5.2.1 Reve Variable



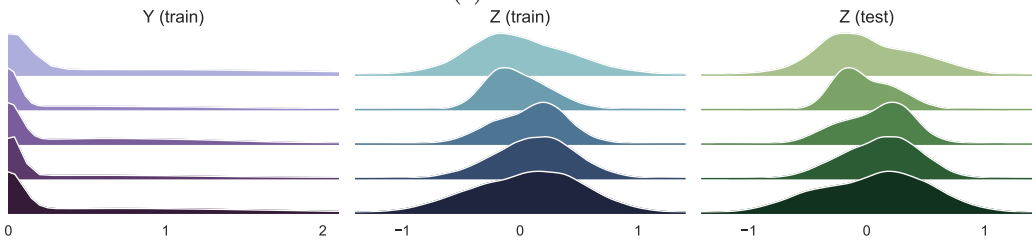
(a) ResNet



(b) Inception



(c) Alexnet



(d) MLP

Figure 5.1.: For four different DNNs, the Figure shows an estimation of the density distributions for some coordinates of the variables Z. The DNNs are ordered by increasing test error, from top to bottom: ResNet (see Section 4), Inception (see Section 4), Alexnet (see Section 4) and MLP, a three layer perceptron presented in [Zha+17] that performs 35.45% error on CIFAR-10 test set. All the DNNs are trained on CIFAR-10 with the standard protocol described in Section 4, and the density distributions are estimated from the images of CIFAR-10 as well. In every plot, each coordinate of Z seems to follow a mixture of two uni-modal distributions. This observation gets clearer as the DNN performs better on the test set. The depth of the DNN may benefit to the quality of the separation between the two modes. We leave this hypothesis for further empirical and theoretical investigation.

In this section, we develop the regularization objective function that is implemented by REVE. We first present the informative variable Z that is extracted from Y and used in our regularization scheme. Then, we derive an objective function from the regularization criterion $H(Z \mid C)$ with a variational approximation. This objective function is implemented as a loss in Section 5.3.

Suppose Y is any encoding of the input X, possibly stochastic. If the decoder that maps $\mathcal{Y}$ onto the class space is linear, with corresponding multiplicative matrix $\boldsymbol{W}_d$, then it is possible to uniquely decompose Y in the following way: $Y = Z + Y^{\text{ker}}$, where Y^{ker} is the projection of Y on the kernel of $\boldsymbol{W}_d$ ($\boldsymbol{W}_d Y^{\text{ker}} = 0$) and Z is the projection on the orthogonal complement of the kernel in the representation space. Thus $\boldsymbol{W}_d Y = \boldsymbol{W}_d Z$ and the term Y^{ker} is not taken into account in the prediction. Consequently, there is no point to compress it and we prefer to focus on the other part Z. We stress that the class space is generally of much lower dimension (the number of classes) than the input space and the representation space. Thus, the kernel of $\boldsymbol{W}_d$ is not reduced to zero and the kernel variable Y^{ker} is very likely to contain a lot of information about the input X. This last observation may be transposed in deep learning, as current DNNs last layer is usually a linear layer before the Softmax mapping $\mathcal{Y}$ onto the class probabilities space $\mathcal{P}(\mathcal{C})$.

We study here a compression scheme on the part Z of Y, the intermediate representation of the DNN, just before the last (fully connected) layer that maps the representation space onto the class space. Note that our motivation is not in contradiction with the recent work of [JSO18], which demonstrates that compressing the representation Y is not of key importance for achieving good generalization.

Regarding the informative part of the neuron activation, we expect Z to exhibit interesting properties. As suggested and studied in [OMS17], neurons may be interpreted as detectors of specific physical elements (object, shape, texture,etc.) that would be discriminant for the task. This intuition also matches the hypothesis in the last chapter 4, where we model the informative variable of a neuron as a binomial variable. Figure 5.1 shows the plotted estimations of the density for some coordinates of Z. The variable has been computed from CIFAR-10 images using different DNNs, further described in Section 5.4 and trained on this same dataset without the proposed regularization. Two modes seem to emerge in each distribution. This bimodality may justify the behavior of neurons as detectors for discriminant patterns. There is detection if the corresponding coordinate of Z is positive, and absence of detection if it is negative. Furthermore, by comparing the density of Z coordinates for different DNN architectures (see Figure 5.1), it seems that the better the modes are separated (distinguishable), the better the DNN generalizes. The separation of the two modes of Z could benefits from the depth of the DNN.

5.2.2 Reve Regularization Function

We adopt the conditional entropy $H(Z \mid C)$ proposed in Chapter 3 ([Blo+18b]) as regularization criterion, for which we develop a general variational upper bound. First, notice that $H(Z \mid C)$ can be decomposed into simpler terms: $H(Z \mid C) = H(Z) - I(Z;C) = H(Z) - H(C) + H(C \mid Z)$. The entropy of the class variable $H(C)$ is entirely determined

by the problem and is independent of any optimization. Therefore, we ignore it and define the following regularization loss:

$$\mathcal{L}_{\text{REVE}}(Z, C) = H(Z) + H(C \mid Z). \tag{5.1}$$

for $C \in \mathcal{C}$ and with any variable $Z \in \mathcal{Z}$ such that the quantities defined below exist.

The definitions of the two terms in $\mathcal{L}_{\text{REVE}}(Z, C)$ are as follow:[1]

$$H(Z) = -\int_{\mathcal{Z}} p(\boldsymbol{z}) \log p(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} \qquad \text{and} \qquad H(C \mid Z) = -\iint_{\mathcal{Z}\,\mathcal{C}} p(\boldsymbol{z}, \boldsymbol{c}) \log p(\boldsymbol{c} \mid \boldsymbol{z}) \mathrm{d}\boldsymbol{c}\mathrm{d}\boldsymbol{z}. \tag{5.2}$$

Variational Approximation: For any space $\mathcal{Z}$, the Kullback-Leibler divergence between any two probability distributions $p(\boldsymbol{z})$ and $q(\boldsymbol{z})$ both defined on $\mathcal{Z}$ is always non-negative: $D_{KL}[p||q] = \int_{\mathcal{Z}} p(\boldsymbol{z}) \log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z})} \mathrm{d}\boldsymbol{z} \geq 0$. Thus, if p is the true distribution of a variable Z, any q can be used to upper bound the entropy of Z: $H(Z) \leq -\int_{\mathcal{Z}} p(\boldsymbol{z}) \log q(\boldsymbol{z}) \mathrm{d}\boldsymbol{z}$. Another important property is that this inequality is an equality if, and only if, $p = q$ almost everywhere. It is good to keep this property in mind since it ensures that the closer p and q are in the sense of the Kullback-Leibler divergence, the tighter our bound gets. Such a q is said to be a *variational approximation* of p.

It is possible to upper bound the REVE criterion by defining variational approximations $q(Z)$ and $r(C \mid Z)$ of the true distributions $p(Z)$ and $p(C \mid Z)$, respectively:

$$\mathcal{L}_{\text{REVE}}(Z, C) \leq -\int_{\mathcal{Z}} p(\boldsymbol{z}) \log q(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} - \iint_{\mathcal{Z}\,\mathcal{C}} p(\boldsymbol{z}, \boldsymbol{c}) \log r(\boldsymbol{c} \mid \boldsymbol{z}) \mathrm{d}\boldsymbol{z}\mathrm{d}\boldsymbol{c}. \tag{5.3}$$

1 On classification tasks, the target C lives in a discrete space, meaning that the integral on $\mathcal{C}$ is a finite sum. Nevertheless, we prefer to keep the general formulation for the derivation of the bound.

5.3 Instantiating Reve

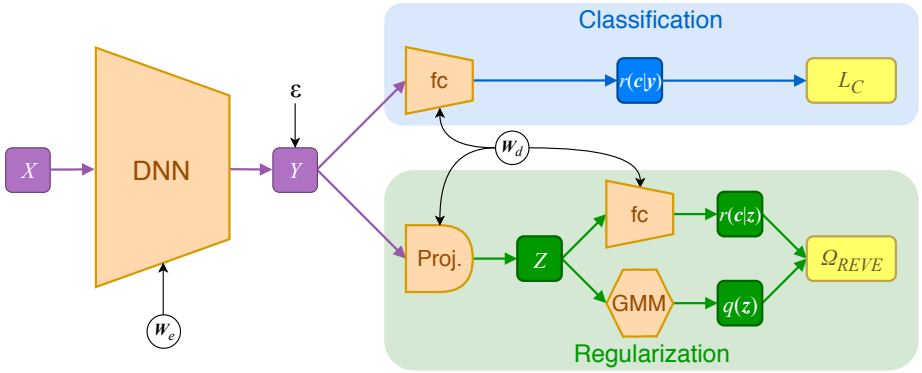


Figure 5.2.: Overview of the Reve learning framework. A stochastic representation Y is computed from the input X by the DNN parameterized by $\boldsymbol{W}_e$. A random variable $\boldsymbol{\epsilon}$ enforces the stochasticity of Y for deterministic DNNs. – The classification branch (in blue) maps $\mathcal{Y}$ onto the class probability space by a fully connected layer (parameterized by $\boldsymbol{W_d}$), followed by a Softmax layer giving an estimated probability $r(C \mid Y)$. The classification loss $\mathcal{L}_\mathrm{C}$ is computed from this distribution. – The regularization branch (in green) starts by projecting the representation onto the orthogonal complement of the kernel of $\boldsymbol{W}_d$, leading to a variable Z. One subbranch gives an estimated probability $r(C \mid Z)$. The other models Z by a Gaussian Mixture Model (GMM) to get an estimated probability $q(Z)$. The Reve loss Ω_Reve is computed from these two distributions. – The two losses $\mathcal{L}_\mathrm{C}$ and Ω_Reve are linearly combined and minimized by SGD.

Similarly to [Ale+17], implementing Reve method necessitates to examine $Y \in \mathcal{Y}$ as a stochastic encoding of the input X. Its distribution follows the probability density $p(\boldsymbol{y} \mid \boldsymbol{x}) = p(\boldsymbol{y}|h(\boldsymbol{W}_e, \boldsymbol{x}))$ where $h(\boldsymbol{W}_e, \boldsymbol{x})$ is the output for input $\boldsymbol{x}$, of a DNN of given architecture and with corresponding parameters $\boldsymbol{W}_e$. We also introduce the decoder that maps $\mathcal{Y}$ onto the class probability space $\mathcal{P}(\mathcal{C})$: $\hat{C} = \mathrm{Softmax}(\boldsymbol{W}_d Y + \boldsymbol{b})$ with $\boldsymbol{W}_d \in \mathbb{R}^{|\mathcal{C}| \times \dim(\mathcal{Y})}$ and $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{C}|}$. When embedding Reve on a deterministic baseline, the decoder takes the expectancy of the variable Y as input to compute the prediction and the classification loss.

When experimenting with deterministic baselines, we use $p(\boldsymbol{y} \mid \boldsymbol{x}) \sim \mathcal{N}(h(\boldsymbol{W}_e, \boldsymbol{x}), \sigma^2)$ as stochastic encoding, that is to say, $Y = h(\boldsymbol{W}_e, X) + \boldsymbol{\varepsilon}$ with $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \boldsymbol{\sigma}^2)$. Contrarily to [Ale+17], the variance $\boldsymbol{\sigma}^2$ is not learned but is a parameter of the method. This has the advantage to cut the dimension of the DNN output by a factor of two. Moreover, since this $\boldsymbol{\varepsilon}$ is averaged during testing for computing the predicted class, the choice of its law is somewhat arbitrary. As a consequence, different values of $\boldsymbol{\sigma}^2$ lead to different evaluations of entropy and only affect the regularization efficiency.

Obtaining Z from Y. The *Singular-Value Decomposition* (SVD) is the natural factorization of a matrix for extracting both its kernel and the orthogonal complement of the kernel. The matrix of parameters $\boldsymbol{W}_d$ can be factorized as $\boldsymbol{W}_d = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, where $\boldsymbol{\Sigma}$ is a diagonal matrix of same rank as $\boldsymbol{W}_d$, containing all its non-zero singular values, and the columns

of $\boldsymbol{U}$ (resp. $\boldsymbol{V}$) form an orthogonal set. Since Z is the projection of Y on the orthogonal complement of the kernel of $\boldsymbol{W}_d$, the compact singular-value decomposition where only the r column vectors of $\boldsymbol{U}$ and $\boldsymbol{V}$ corresponding to the non-zero singular values are computed. As a consequence, $\boldsymbol{V}^\top$ maps the representation space (of Y) onto the orthogonal complement of the kernel and $\boldsymbol{V}\boldsymbol{V}^\top$ is the orthogonal projection onto the orthogonal complement of the kernel. The part Z is simply computed from Y by the matrix multiplication: $Z = \boldsymbol{V}\boldsymbol{V}^\top Y$.

5.3.1 Reve Objective Function

We develop here the previous bound (5.3) by using a Monte Carlo method on the terms of the right-hand side of the inequality. By applying the Bayes rule we get[2]:

$$H(Z) \leq -\int_{\mathcal{Z}} p(\boldsymbol{z}) \log q(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} = -\iint_{\mathcal{X}\,\mathcal{Z}} p(\boldsymbol{x}) p(\boldsymbol{z} \mid \boldsymbol{x}) \log q(\boldsymbol{z}) \mathrm{d}\boldsymbol{x} \mathrm{d}\boldsymbol{z}. \tag{5.4}$$

Similarly:

$$H(C \mid Z) \leq -\iint_{\mathcal{Z}\,\mathcal{C}} p(\boldsymbol{z}, \boldsymbol{c}) \log r(\boldsymbol{c} \mid \boldsymbol{z}) \mathrm{d}\boldsymbol{z} \mathrm{d}\boldsymbol{c} \tag{5.5}$$

$$= -\iiint_{\mathcal{X}\,\mathcal{Z}\,\mathcal{C}} p(\boldsymbol{x}) p(\boldsymbol{c} \mid \boldsymbol{x}) p(\boldsymbol{z} \mid \boldsymbol{x}) \log r(\boldsymbol{c} \mid \boldsymbol{z}) \mathrm{d}\boldsymbol{x} \mathrm{d}\boldsymbol{z} \mathrm{d}\boldsymbol{c}. \tag{5.6}$$

To pass from line (5.5) to line (5.6), we exploit the Markov chain hypothesis $C \leftrightarrow X \leftrightarrow Z$ of the model that implies that $p(\boldsymbol{c} \mid \boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{c} \mid \boldsymbol{x})$ and thus $p(\boldsymbol{z}, \boldsymbol{c}) = \int_{\mathcal{X}} p(\boldsymbol{x}) p(\boldsymbol{c} \mid \boldsymbol{x}) p(\boldsymbol{z} \mid \boldsymbol{x}) \mathrm{d}\boldsymbol{x}$.

We obtain:

$$\mathcal{L}_{\textsc{Reve}}(Z, C) \leq -\iiint_{\mathcal{X}\,\mathcal{Z}\,\mathcal{C}} p(\boldsymbol{x}) p(\boldsymbol{c} \mid \boldsymbol{x}) p(\boldsymbol{z} \mid \boldsymbol{x}) \Big(\log q(\boldsymbol{z}) + \log r(\boldsymbol{c} \mid \boldsymbol{z}) \Big) \mathrm{d}\boldsymbol{x} \mathrm{d}\boldsymbol{z} \mathrm{d}\boldsymbol{c} \tag{5.7}$$

Then, we can use the empirical distribution of (X, C) given by the mini-batch sample $(\boldsymbol{x}_n, \boldsymbol{c}_n)_{n=1..N}$ to use the Monte Carlo method. Finally, the objective function to minimize we can extract from the upper bound (5.7) can be written:

$$\Omega_{\textsc{Reve}}\big(X; C; \boldsymbol{W}_e; \boldsymbol{W}_d\big) = -\frac{1}{N} \sum_{n=1}^{N} \left(\int_{\mathcal{Z}} p(\boldsymbol{z} \mid \boldsymbol{x}_n) \big(\log q(\boldsymbol{z}) + \log r(\boldsymbol{c}_n \mid \boldsymbol{z}) \big) \mathrm{d}\boldsymbol{z} \right). \tag{5.8}$$

In order to estimate $\int_{\mathcal{Z}} p(\boldsymbol{z} \mid \boldsymbol{x}) \ \ldots \ \mathrm{d}\boldsymbol{z}$, we again use the Monte Carlo method by sampling Z from chosen probability $p(\boldsymbol{z}|\boldsymbol{x_n}) = p(\boldsymbol{z} \mid h(\boldsymbol{W}_e, \boldsymbol{x_n}))$. To sample Z, S samples $\boldsymbol{\varepsilon}_s$ of $\boldsymbol{\varepsilon}$ are drawn leading to the representation samples $\boldsymbol{y_{n,s}} = h(\boldsymbol{W}_e, \boldsymbol{x_n}) + \boldsymbol{\varepsilon}_s$ from which we deduce the samples $\boldsymbol{z_{n,s}}$ of Z.

2 We consider without further justifications that the conditions of the Fubini-Tonelli theorem allowing to invert the integrals are respected.

Finally, we approximate the right-hand side term in (5.8) to define our REVE loss function Ω_{REVE}:

$$\Omega_{\text{REVE}}\big((\boldsymbol{x}^{(k)}, \boldsymbol{c}^{(k)})_{k=1..K}; \boldsymbol{W}_e; \boldsymbol{W}_d\big) = -\frac{1}{KS}\sum_{k=1}^{K}\sum_{s=1}^{S}\Big(\log q(\boldsymbol{z}^{(k,s)}) + \log r(\boldsymbol{c}^{(k)} \mid \boldsymbol{z}^{(k,s)})\Big). \tag{5.9}$$

As suggested by Alemi et al. [Ale+17], we choose to use $S = 12$ which seemed to capture well enough the noisy distribution of Y in our experiments.

In everything that follows, we consider the approximation (5.9) as the objective function we want to minimize for regularization.

5.3.2 Model for q

Since the dimension of Z can be arbitrarily large depending on the considered architecture, we start by adopting the *mean field approximation* in order to limit the complexity of our model:

$$q(\boldsymbol{z}) = \prod_{i=1}^{\dim(\mathcal{Z})} q(z_i) \tag{5.10}$$

This factorization means that we consider independent the coordinates of Z.

We have already mentioned in Section 5.2 the intuition of a bimodal variable related to two modes, a neuron acting as a detector of a discriminative attribute. Thus, we choose to model $q(z_i)$ by a Gaussian Mixture Model (GMM) with two modes, $M_i = 1$ and $M_i = 0$:

$$q(z_i) = \alpha_i \mathcal{N}(z_i \mid \mu_{1,i}, \sigma^2_{1,i}) + (1 - \alpha_i)\mathcal{N}(z_i \mid \mu_{0,i}, \sigma^2_{0,i}) \tag{5.11}$$

where the parameters $\boldsymbol{\alpha}, \boldsymbol{\mu}_1, \boldsymbol{\sigma}^2_1, \boldsymbol{\mu}_0, \boldsymbol{\sigma}^2_0$ must be properly estimated.

While *Expectation-Maximization* is the standard algorithm for computing the parameters of a Gaussian Mixture Model, it is practically not applicable because the size of the mini-batches is in general too small. To circumvent this problem, we suggest to skip the E-step by directly approximating $\pi_i(z_i) = p(M_i = 1 \mid z_i)$ by the sigmoid function: $\pi_i(z_i) = \frac{1}{1+\exp(-z_i)}$. Indeed, we observe empirically that one mode tends to be positive while the other one tends to be negative. Using this approximation, we compute the parameters of the two Gaussians on the mini-batch through the M-step.

5.3.3 Model for r

Similarly to [Ale+17], the approximation $r(\boldsymbol{c} \mid \boldsymbol{z})$ can be defined by the multinomial logistic regression model at the end of the DNN using the Softmax:

$$r(\boldsymbol{c} \mid \boldsymbol{z}) = Softmax(\boldsymbol{W}_d \boldsymbol{z} + \boldsymbol{b})_{\boldsymbol{c}} \tag{5.12}$$

where, for any $\boldsymbol{u}$, $u_{\boldsymbol{c}}$ is the coordinate of $\boldsymbol{u}$ related to the class $\boldsymbol{c}$. Unlike for $q(\boldsymbol{z})$, all the parameters of this variational approximation are defined by and optimized with the DNN.

5.3.4 Reve Regularization Loss

Combining the results of the previous subsections, the REVE regularization loss is computed as:

$$\Omega_{\text{REVE}}\big((\boldsymbol{x}^{(k)}, \boldsymbol{c}^{(k)})_{k=1..K}; \boldsymbol{W}_e; \boldsymbol{W}_d\big) = -\frac{1}{KS}\sum_{k=1}^{K}\sum_{s=1}^{S}\Bigg[\log Softmax(\boldsymbol{W}_d\boldsymbol{z}^{(k,s)} + \boldsymbol{b})_{\boldsymbol{c}^{(k)}} + \sum_{i=1}^{\dim(\mathcal{Z})} \log\Big(\alpha_i \mathcal{N}(z_i^{(k,s)} \mid \mu_{1,i}, \sigma_{1,i}^2) + (1-\alpha_i)\mathcal{N}(z_i^{(k,s)} \mid \mu_{0,i}, \sigma_{0,i}^2)\Big)\Bigg]. \quad (5.13)$$

Figure 5.2 summarizes the architecture of REVE and how it is embedded to a DNN.

5.4 Experiments

In this section, we present various experiments, done on several databases, which demonstrate the benefits of using REVE during the training of DNNs, thus supporting the theory. In all our experiments we use the cross entropy loss as classification loss. We also use a weight decay of 10^{-3} in every experiment, mainly to stabilize the training. Note nevertheless that the use of weight decay does not seem to impact REVE in any way. The DNNs are trained using SGD with 0.9 momentum and an exponentially decaying learning rate with a default value set for each considered architecture. Unless stated otherwise, the batch size used for the experiments is 128.

We experiment with our regularization strategy on three DNNs, constituting a variety of architectures: AlexNet and Inception from the previous chapter. We recall : **AlexNet**: an AlexNet-like model [KSH12] with three 5×5 convolution + 2×2 max-pooling layers, then two fully connected layers with 1000 neurons as intermediate representation;
Inception Net: a small Inception model [Sze+14] similar to the one used in [Zha+17];
And **ResNet**, an architecture from [ZK16] which varies from the architecture of the previous chapter. In fact, for the experiments here, we have chosen the ResNet configuration, depth $= 28$ and $k = 10$ which give better accuracies.

Please note that we use our own implementation of these DNNs on TensorFlow [Aba+16], thus explaining that the observed results may slightly differ from published results on those architectures. In particular, the preprocessing we apply to the datasets may be different to those used in the original papers. Please also note that we use the same three architectures on every dataset in order to better understand the influence of REVE on various data. The aim of these experiments is not so much to achieve state-of-the-art performance but to prove the efficiency of our regularization. Thus, for fair comparison, the training with and without REVE is performed with the exact same experimental protocol.

We would like to point out that REVE does not add additional parameters to the DNN training except for the two hyper-parameters $\boldsymbol{\sigma}^2$ and the Lagrange multiplier of the regularization loss β_{REVE}. Most of REVE's computation remains in the SVD that is computed once per batch, the rest of the computation being negligible for the studied architectures.

For instance for the ResNet architecture, on CIFAR-10, involving a SVD on a matrix of size 640×10, REVE increases by less than 10% the training time[3].

5.4.1 CIFAR Experiments

Table 5.1.: Classification error (%) results on CIFAR-10 and CIFAR-100 test sets. REVE regularization is applied on three different architectures on CIFAR-10 and its performance is compared to the baseline performance of these models. REVE systematically reduces the error on both datasets and every network architecture, even the most challenging.

	CIFAR-10			CIFAR-100		
	AlexNet	Inception Net	ResNet	AlexNet	Inception Net	ResNet
Baseline	15.62	6.10	5.39	48.29	27.36	22.84
REVE on Y	14.80	6.38	5.71	48.56	27.10	22.80
REVE	**13.92**	**5.92**	**5.18**	**48.07**	**26.94**	**22.63**

We compare here the test error of DNNs trained with and without the use of REVE on CIFAR [Kri09]. Some classic data augmentation techniques are applied to the images: the 32×32 training images are padded to 40×40 and randomly cropped to 32×32 and a random horizontal flip is applied. For preprocessing, we normalize the data using the channel means and standard deviations of the training set. We make sure that every DNN achieves zero percent error on the training set. The two hyperparameters $\boldsymbol{\sigma}^2$ and the Lagrange multiplier of the regularization loss β_{REVE} are cross validated to maximize the performance of REVE. However, we notice that the regularization is fairly stable to slight changes of these parameters ($\boldsymbol{\sigma}^2 \sim 10^{-2}$ and $\beta_{\text{REVE}} \sim 10^{-4}$).

CIFAR-100 The CIFAR- 100 dataset is, similarly to CIFAR-10, composed of 32×32 color image, representing 100 different classes. There is also 50K images for training and 10K images for test.

Generalization Improvement The classification errors obtained on the test set of CIFAR are reported on Table 5.1. Our experiments show that the application of REVE systematically improves the performances over the baseline on both CIFAR-10 and CIFAR-100. Especially, we insist on the fact that REVE improves the generalization of every architecture, even the best engineered and performing of them. To further emphasize the benefit provided by applying our method on the variable Z, we also present in Table 5.1 the classification error obtained by applying the REVE paradigm on the representation Y (referenced as REVE on Y). This strategy never outperforms the proposed method and sometimes increases the error over the baseline. This experiment highlights the fact that a focused compression on Z is definitely beneficial to DNNs generalization. We also experiment with another variant

3 To speedup the method, an option consists in updating the SVD every few batches without many consequences on the regularization.

of REVE using a basic approximation for $q(\boldsymbol{z})$: a normal model instead of a GMM. This variant gives poorer results compared to regular REVE on CIFAR-10 (14.24%, 6.17% and 5.34% on AlexNet, Inception Net and ResNet, respectively), proving that the choice of a bimodal model to describe Z is experimentally validated.

Evolution of the Loss Functions Figure 5.3 presents a comparison of the losses evolution with and without application of REVE. It appears that REVE significantly reduces the classification loss on the test set compared to the baseline, further highlighting the good generalization of REVE-regularized DNNs. Moreover, the application of REVE drastically reduces the loss Ω_{REVE} which bounds the conditional entropy $H(Z \mid C)$, proving that our regularization strategy by introduction of a loss is quite effective. The plots of the regularization loss on the baseline show that the SGD naturally reduces our criterion. This observation supports the claim that the criterion of REVE benefits to the generalization of a DNN. It is also interesting to note that the REVE loss shows consistent generalization, achieving similar values on both the train and test sets.

Best Performance on CIFAR-10 To achieve the best performance possible on CIFAR-10 with REVE, we adopt an ensemble strategy: six ResNets are trained with REVE and a batch size of 16, then are stacked together by averaging their Softmax predictions. Applying this strategy, we achieve **3.29**% test error on CIFAR-10. To contextualize, DenseNet-BC [Hua+17], the known published state-of-the-art architecture on CIFAR-10, achieves 3.46% test error with a single model, showing that REVE allows us to achieve competitive results using this strategy.

5.4.2 SVHN Experiments

Table 5.2.: Classification error (%) results on SVHN test set. REVE regularization is applied on three different architectures on CIFAR-10 and its performance is compared to the baseline performance of these models. REVE systematically reduces the error of every network architecture, even the most challenging.

	SVHN		
	AlexNet	Inception Net	ResNet
Baseline	7.68	3.78	3.27
REVE on Y	6.78	3.52	3.43
REVE	**6.55**	**3.29**	**3.21**

To show that REVE performance generalizes to other datasets, we conduct similar experiments on SVHN [Net+11] using the same architectures. The dataset includes color images of cropped digits of size 32×32 pixels. It is composed of 73257 images in the training set, and 26032 images for the test set. The same data augmentation by padding and cropping is applied on the training set. The accuracy results presented in Table 5.2 show a behavior similar to the results on CIFAR with a systematic improvement by applying REVE.

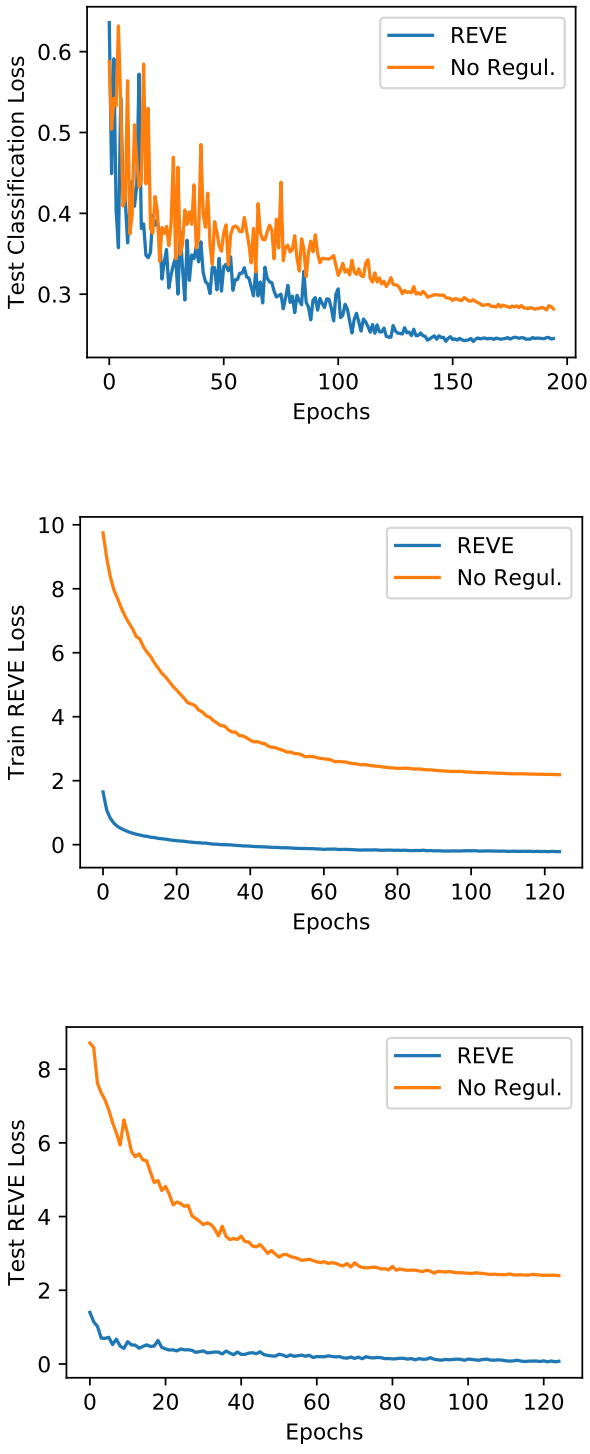


Figure 5.3.: Comparison of the losses evolution with and without application of REVE with ResNet.
(a) Classification loss on the test set; (b) REVE loss on the training set; (c) REVE loss on the test set.

5.4.3 Large Scale Experiments on ImageNet

REVE application is not limited to architectures that end with a fully connected layer. As an illustration we study here the model WELDON [DTC16] in the context of large size images with ImageNet dataset [JDe+09]. WELDON architecture modifies the classical ResNet-101 [He+16] by replacing the final average pooling and the following fully connected layer by a 1×1 convolutional layer followed by a max + min pooling described in [DTC16]. Apart from these modifications, the architecture of the DNN leading to the considered representation Y stays identical. WELDON last 1×1 convolution layer can be seen as a linear operation applied on each spatial coordinate of the map Y. The corresponding matrix is thus used to compute the SVD providing the projection matrix $VV^\top$. This last matrix parameters are used for a 1×1 convolution on Y in order to obtain Z on which REVE is applied.

The experiment consists in fine-tuning by regularizing with REVE the WELDON architecture initialized with pre-trained ResNet-101 parameters. The accuracies obtained without fine-tuning are 78.51% for the top-1 accuracy and 94.65% for the top-5 accuracy on the validation set. After only one epoch of fine-tuning we obtain a top-1 accuracy of 79.87% and a top-5 accuracy of 95.43% on the validation set. In addition to demonstrating the efficiency of REVE on large scale tasks, this experiment gives an insight on the adaptation of REVE for different kind of DNN architecture.

5.5 Conclusion

The regularization loss studied in this chapter, REVE, is a alternative to SHADE, based on a similar criterion. To be more efficient, REVE removes from the representation variable the information that is naturally filtered by the class projection layer. The conditioned entropy of the obtained variable is reduced by minimizing a variational entropy bound.

The positive results on the generalization error obtained for all tested DNN architectures, on several datasets, is a second confirmation for our class conditioned entropy criterion. More generally, our work on REVE and on SHADE, has demonstrated that information measures can be pertinent to study deep learning generalization performances.

Moreover, the exhibition of a predictive sub-variable Z, from the representation variable, is a promising technique to better decipher deep learning mechanisms. By restricting DNN study to a simpler variable, responsible for the class prediction, it is easier to extract the deep learning properties enabling good generalization.

CHAPTER 6

CONCLUSION AND PERSPECTIVES

In this thesis, we have investigated several options to improve deep learning generalization performance in the context of image classification. We have addressed the problem from three axes. The first direction studies the SGD training optimization through distributed methods, in order to accelerate the optimization of DNNs on huge datasets. Speeding up the training is especially important for DNNs that are extremely deep. A second approach consists in studying the architecture of CNNs, in order to optimize the processing and filtering of information through the layers. The last angle explores the possibility to modify the training objective function, to regularize the optimization.

Distributed methods. Concerning the first point, we have introduced in Chapter 2, a framework that enables to interpret distributed methods in term of batch size. It demonstrates that the speed up is limited by a convergence rate of a single threaded optimization with bigger batches. The distance to this upper bound is related of the consensus quality between workers' variable, which is determined by the quality and the frequency of communication. Knowing that communication time is not negligible, it is important to exploit an efficient strategy to limit the communication impact. Distributed methods are then distinguishable with a ratio consensus versus quantity of information exchanged. Within this framework, we also introduce a matrix notation, that represents the exchanges of information in the system. A distributed method, can thus be studied from its update matrix only.

This framework encompasses several of the popular existing distributed methods like EASGD [ZCL15] and Downpour [Dea+12], and we have proposed two additional methods. A first one, PerSyn, is centralized and periodically synchronous. It periodically synchronizes the workers' variable, by forcing a perfect consensus once in a while. A second one, GoSGD, gets inspiration from Gossip protocols, that are known to perform well at controlling a consensus. By only requesting peer to peer communication, it is decentralized and asynchronous avoiding synchronization latencies. It demonstrates a slightly higher average consensus error than PerSyn, but for fewer and more efficient communications.

An extension of our work could be to explore a distributed method, that would update the communication matrix after every batch processed. The update on the matrix aims at optimizing the consensus. It can be described as a optimization problem under constraints. The objective function would be the distance from an ideal matrix, for instance: $K^t = \arg\min_K \|KP^{(t-1)} - 1/M \mathbb{1}_{M\times M}\|$ and the constraints would define the methods characteristics (centralized or not, synchronous or not) as well as some communication restrictions.

More generally, we have studied distributed methods in term of optimization speed and convergence rate. Accelerating the optimization enables testing more methods and using

more images in the training set. A promising and alternative research direction would be the study of distributed methods under the scope of generalization. We have observed that our distributed methods are able to improve the generalization error of CNNs while being similar to using bigger batches. This phenomenon is counter intuitive because small batches are usually better for generalization. It would be interesting to inspect further this aspect of the distributed SGD.

Architecture. In Chapter 3, we have introduced a new activation function, MaxMin. Contrarily to ReLU, MaxMin does not filter any information but separates the positive part from the negative part of neuron activations after linear convolutions. In fact, machine learning predictive algorithms have to balance between filtering information in order to be invariant and robust to noise, and transmitting sufficient discriminant information in order to make accurate predictions. In the case of CNN, the pooling layer filters spatial information making the model invariant to local natural transformations. For the activation function, the benefit of the filtering is not evident. The fact that the convolutional filters of a same layer are negatively correlated demonstrates that negative activations contain relevant information for the prediction. With MaxMin, both negative and positive parts of the neuron activation are transmitted to the following layer and can be treated differently contrarily to PReLU or LReLU. In our experiments, CNN with MaxMin activation are able to significantly outperform CNN with ReLU activation for equivalent number of parameters. By not filtering negative detections, MaxMin activation function is able to exploit them with fewer parameters and then to improve the global generalization of CNNs.

The intuition leading to MaxMin could be reported to other types of transformation. If a transformation T correlates the filters, it means that there is redundancy in the learning. It could be avoided by transmitting the information about the filters and their transformation. As such, the learned filters would benefit for more occurrences in the training data. The transformation T can be defined by intuition, like filter rotations for instance, or it can be learned. Moreover, the question of redundancy in the learned parameters can be addressed from a theoretical point of view.

Regularization. Regarding regularization, we have inspected in Chapter 4 and Chapter 5, the possibility to regularize the DNN training, for a fixed architecture, on a given dataset. We have first introduced a new regularization criterion: $H(Y|C)$. This information theory criterion is inspired from the Information Bottleneck Framework. We claim that it is a good measure of invariance for any machine learning model.

In Chapter 4, we have instantiated this criterion in a layer wise and neuron wise fashion in order to regularize efficiently all layers of the DNN. We have used statistical modeling and information theory inequalities to properly upper bound our criterion and construct a loss that can be evaluated on a mini-batch. For all layers and all neurons, this loss is added to the objective function in order to be minimized. The resulting regularization scheme is called SHADE. Experimental results show that SHADE is more efficient at improving generalization performance of a DNN than existing regularization methods. The fact that it systematically improves the DNN generalization performance for several architectures, on many datasets is very convincing and validates both the criterion and its implementation. Despite being already competitive, the fact that the method can be challenged in many

aspects to improve its efficiency at minimizing the proposed criterion is very encouraging. For instance, the model of the latent variable can be adjusted, or some inequalities can be optimized to obtain a sharper bound.

In Chapter 5, we have presented an alternative to SHADE, based on a similar criterion. Unlike SHADE, the new regularization focuses on the penultimate variable layer of the DNN. From this variable, our method extracts a sub-variable that is not filtered by the class projection layer. This variable, named Z, is prevalent in the prediction and is thus more efficient at regularizing. In order to minimize the criterion $H(Z|C)$, the regularization scheme, called REVE, exploits a variational upper bound on the conditionnal entropy. This upper bound uses a bimodal model for Z, with a mixture of two Gaussians. This model is motivated both by intuitions and empirical observations. Contrary to SHADE, this bound is general and does not require any approximation. It can be estimated on a mini-batch and minimized with SGD when added to the global objective function. The new regularization method is also able to outperform SHADE generalization error, as well as other information based regularizations for all architectures it has been tested on, and on several datasets. Moreover, many improvements of the method should be expected. For instance, some works in progress are considering the use of a decomposed matrix system $U\Sigma V$ of the last projection matrix, in order to avoid computing the SVD at every batches. The decomposition $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ of $\boldsymbol{W}_d$ is directly learned by SGD, with $\boldsymbol{U}$ and $\boldsymbol{V}$ projected on orthogonal matrices after each step. This method trades the SVD computation for the addition of $C^2 + C$ parameters and has already demonstrated positive results.

Furthermore, in both Chapter 4 and Chapter 5, we have studied the new entropy criterion supposed to be correlated with the generalization performance of a machine learning model. Our experiments validate the intuition but there is still few strong theoretical demonstrations to legitimate it in the context of machine learning. Developing generalization bounds depending on this entropy, in a PAC fashion, could be very convincing to deploy much more energy on this criterion.

From a broader perspective, an essential question that came all along with our researches, remains unanswered. It is the role of the depth on the good generalization performance of deep learning. State of the art DNNs are still growing and the main limitation for deeper architectures seems to be the optimization constraints as mentioned in Chapter 2.

As seen previously, A DNN layer with ReLU activation filters information about its input. As explained in Chapter 4 and illustrated in Chapter 3, this filtering is not efficient if it applies on information correlated with the class. One benefit of the depth could then be the increased capacity conveyed to the DNN, to construct a top representation variable that has small dependency with the input variable, while conserving high mutual information with the class variable. This is the hypothesis exposed in [NN15], which authors interpret the effect of the depth from the Information Bottleneck Framework point of view. However, CNN architectures with reversible layers (that do not filter any information) and that generalize well have been exhibited in [JSO18], contradicting the precedent hypothesis.

If most of the generalization theory principles do not apply on deep learning models' representation, it would be interesting to rather focus on the underlying variables that have no information filtered before the prediction. For instance, the approach in Chapter 5, that exhibits an underlying variable from the penultimate representation, with no information

filtered by the class projection layer, could be sharpened and generalized to other layers of the DNN. Understanding how the depth affects the properties of these variables could lead to better understand deep learning generalization performance.

APPENDIX

A

GOSGD

a.1 Estimator variance

The error introduced by the Monte Carlo estimator of the gradient is proportional to the batch size.

Proof. The relation between the approximation error and the size of the sample batch is given by:

$$\begin{aligned}\mathbb{E}_{(\mathbf{S}_k)_{k=1..K} \sim p^K} ||\nabla\mathcal{L}(\mathbf{w}) - \widehat{\nabla}\mathcal{L}(\mathbf{w})||_2^2 &= \mathbb{E}_{(\mathbf{S}_k)_{k=1..K} \sim p^K} \left(\sum_{i=1}^{dim(\mathbf{w})} |\nabla\mathcal{L}(\mathbf{w})_i - \widehat{\nabla}\mathcal{L}(\mathbf{w})_i|^2 \right) \\ &= \sum_{i=1}^{dim(\mathbf{w})} \mathbb{E}_{(\mathbf{S}_k)_{k=1..K} \sim p^K} |\nabla\mathcal{L}(\mathbf{w})_i - \widehat{\nabla}\mathcal{L}(\mathbf{w})_i|^2\end{aligned}$$

$\widehat{\nabla}\mathcal{L}(\mathbf{w})_i$ being an unbiased estimator of $\nabla\mathcal{L}(\mathbf{w})_i$, $\mathbb{E}_{(\mathbf{S}_k)_{k=1..K} \sim p^K}\left(\widehat{\nabla}\mathcal{L}(\mathbf{w})_i\right) = \nabla\mathcal{L}(\mathbf{w})_i$ and $\mathbb{E}_{(\mathbf{S}_k)_{k=1..K} \sim p^K} |\nabla\mathcal{L}(\mathbf{w})_i - \widehat{\nabla}\mathcal{L}(\mathbf{w})_i|^2 = \mathbb{V}\mathrm{ar}_{(\mathbf{S}_k)_{k=1..K} \sim p^K}(\widehat{\nabla}\mathcal{L}(\mathbf{w})_i)$. For Monte-Carlo estimators we have that $\mathbb{V}\mathrm{ar}_{(\mathbf{S}_k)_{k=1..K} \sim p^K}(\widehat{\nabla}\mathcal{L}(\mathbf{w})_i) = \mathbb{V}\mathrm{ar}_{(\mathbf{S}_k)_{k=1..K} \sim p^K}(\frac{1}{N}\sum_{k=1}^{n} \nabla\ell(\mathbf{w}, \mathbf{S}_k)_i) = \frac{1}{N}\mathbb{V}\mathrm{ar}_{(\mathbf{S} \sim p}(\nabla\ell(\mathbf{w}, \mathbf{S})_i)$, the $\mathbf{S}_k$ being *iid*.

$$\begin{aligned}\mathbb{E}||\nabla\mathcal{L}(\mathbf{w}) - \widehat{\nabla}\mathcal{L}(\mathbf{w})||_2^2 &= \sum_{i=1}^{dim(\mathbf{w})} \frac{1}{N}\mathbb{V}\mathrm{ar}(\nabla\ell(\mathbf{w}, \mathbf{S})_i) \\ &= \frac{1}{N} tr(\mathbb{C}\mathrm{ov}_{\mathbf{S}}[\nabla\ell(\mathbf{w}, \mathbf{S})])\end{aligned}$$

□

a.2 Alternate Gradient

The update rules of GoSGD are in expectation equivalent to the empirical risk and the consensus loss parts of the gradient of the consensus augmented distributed problem.

For the empirical risk part, since at each time step t one of the workers performs a local stochastic gradient descent update with probability $1/M$, it is in expectation equivalent to performing the gradient descent on the empirical risk part with a learning rate divided by M.

For the consensus part, let us recall that the update for a receiving worker is:

$$\mathbf{w}_r^{(t+1)} = \frac{\lambda_r}{\lambda_s + \lambda_r}\mathbf{w}_r^{(t)} + \frac{\lambda_s}{\lambda_s + \lambda_r}\mathbf{w}_s^{(t+)}$$

This update is occurring with probability $\frac{p}{M(M-1)}$ for all pairs $(s,\ r)$ of workers. First we demonstrate the following lemma:

Lemma 1. $\mathbb{E}\left[\frac{\lambda_r}{\lambda_r+\lambda_s}\right] = \frac{1}{2}$ *for all possible pairs* $(r,\ s)$.

Proof. The weights update can be written with a random communication matrix $A^{(t)}$ with the following definition:

$$A^{(t)} = \begin{cases} I, \text{with probability } p \\ I - \frac{1}{2}e_s e_s^\top + \frac{1}{2}e_r e_s^\top, \text{with probability } 1-p \end{cases}$$

The sequence of $A^{(t)}$ are i.i.d. and we note $\mathbb{E}\left[A^{(t)}\right] = A$:

$$\begin{aligned} A &= pI + (1-p)\sum_{s,r\neq s}\frac{1}{M(M-1)}\left(I - \frac{1}{2}e_s e_s^\top + \frac{1}{2}e_r e_s^\top\right) \\ &= \left(1 - \frac{1-p}{2M}\right)I + \frac{1-p}{2M(M-1)}\mathbb{1}\mathbb{1}^\top \end{aligned}$$

Using A and denoting $\Lambda^{(t)}$ the vector concatenating all coefficients at time t, we have the following recursion:

$$\begin{aligned} \mathbb{E}\left[\Lambda^{(t+1)}|\Lambda^{(t)}\right] &= \mathbb{E}\left[A^{(t)}\right]\Lambda^{(t)} \\ &= A\Lambda^{(t)} \end{aligned}$$

Since all weights are initialized to 1, unrolling the recursion leads to:

$$\mathbb{E}\left[\Lambda^{(t+1)}\right] = A^t\mathbb{1}$$

Since $\mathbb{1}$ is a right eigenvector of A (associated with eigenvalue λ), we have:

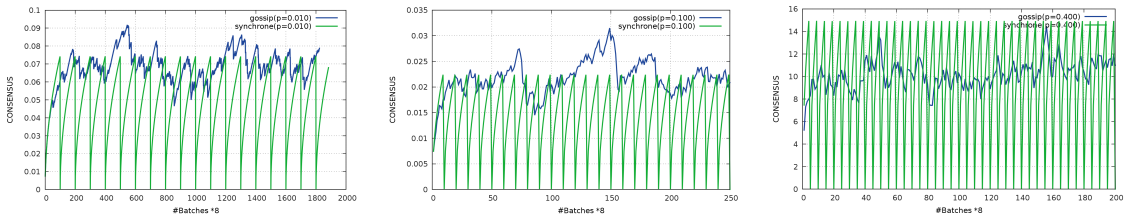
$$\mathbb{E}\left[\Lambda^{(t+1)}\right] = \Lambda^t\mathbb{1}$$

Which means that all weights are equal in expectation. □

This lemma leads to the following expected update step:

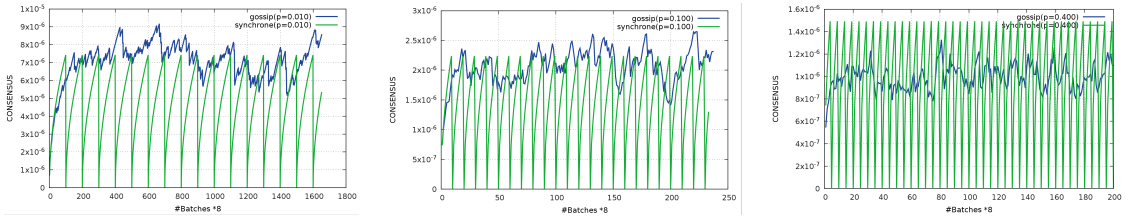
$$\mathbb{E}\left[\mathbf{w}_r^{(t+1)} - \mathbf{w}_r^{(t+\frac{1}{2})}\right] = \frac{p}{2M(M-1)}\left(\mathbf{w}_s^{(t)} - \mathbf{w}_r^{(t+\frac{1}{2})}\right)$$

Which corresponds in expectation to a gradient descent performed on the consensus part with a learning rate of $\frac{p}{2M(M-1)}$.



(a) $\rho = 0.01$ (b) $\rho = 0.1$ (c) $\rho = 0.4$

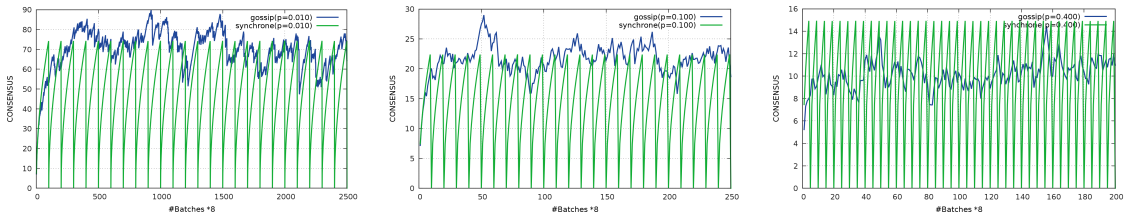
Figure A.2.: Evolution of the consensus error $\varepsilon(t)$ against time for different communication frequency/probability ρ for both GoSGD and PerSyn for updates of variance 4.



(a) $\rho = 0.01$ (b) $\rho = 0.1$ (c) $\rho = 0.4$

Figure A.3.: Evolution of the consensus error $\varepsilon(t)$ against time for different communication frequency/probability ρ for both GoSGD and PerSyn for updates of variance 8.

a.3 Consensus measures



(a) $\rho = 0.01$ (b) $\rho = 0.1$ (c) $\rho = 0.4$

Figure A.1.: Evolution of the consensus error $\varepsilon(t)$ against time for different communication frequency/probability ρ for both GoSGD and PerSyn for updates of variance 1.

As expected, the higher the variance, the larger the consensus error. This seems intuitive, however, it appears that it is not the variance of the update terms that differentiates the algorithms in term of consensus, but it is mostly the frequency of exchange. Indeed for low probability of exchange ($p = 0.01$) we note that the consensus distance with gossip exchanges is almost always above the consensus distance with PerSyn exchange. For bigger probability ($p = 0.4$) the consensus distance of GoSGD is somewhere between the worst and the best consensus error of the PerSyn. However the average consensus distance of GoSGD is still above PerSyn.

APPENDIX B

SHADE

b.1 Entropy bounding the reconstruction error

In Section 4.2 we highlight a link between the entropy $H(X \mid Y)$ and the difficulty to recover the input X from its representation Y. Here we exhibit a concrete relation between the reconstruction error, that quantifies the error made by a strategy that predicts X from Y, and the conditional entropy. This relation takes the form of an inequality, bounding the error measure in the best case (with the reconstruction strategy that minimizes the error) by an increasing function of the entropy. We note $\hat{x}(Y) \in \mathcal{X}$ the reconstruction model that tries to guess X from Y.

The discrete case In case the input space is discrete, we consider the zero-one reconstruction error for one representation point Y: $\varepsilon(Y) = E_X[\mathbb{1}(\hat{x}(Y) \neq X)]$. This is the probability of error when predicting $\hat{x}(Y)$ from Y. The function that minimizes the expected error $\mathcal{E} = \mathbb{E}_Y(\varepsilon(Y))$ is $\hat{x}(Y) = \arg\max_{x \in \mathcal{X}} p(x \mid Y)$ as shown in Proof 1. We derive the following inequality:

$$\frac{H(X \mid Y) - 1}{\log |\mathcal{X}|} \leq \mathcal{E} \leq 1 - 2^{-H(X|Y)}. \tag{B.1}$$

The left side of the inequality uses Fano's inequality in [CT91], the right one is developed in proof 2. This first inequalities show how the reconstruction error and the entropy are related. For very invariant representations, it is hard to recover the input from Y and the entropy of $H(X \mid Y)$ is high.

Besides, there can be an underlying continuity in the input space and it could be unfair to penalize predictions close to the input as much as predictions far from it. We expose another case below that takes this proximity into account.

The continuous case In the case of convex input space and input variable with continuous density, we consider the 2-norm distance as reconstruction error: $\varepsilon(Y) = \mathbb{E}_{X|Y}\left[||X - \hat{x}(Y)||_2^2\right]$. This error penalizes the average distance of the input and its reconstruction from Y. The function that minimizes the expected error $\mathcal{E} = \mathbb{E}_Y[\varepsilon(Y)]$ is the conditional expected value: $\hat{x}(y) = \mathbb{E}[X \mid Y = y]$. Then $\mathcal{E} = \mathbb{V}\mathrm{ar}(X \mid Y)$. Helped by the well-known inequality $H(X \mid Y) \leq \frac{1}{2}\ln(2\pi e \mathbb{V}\mathrm{ar}(X \mid Y))$ we obtain:

$$\frac{e^{2H(X|Y)}}{2\pi e} \leq \mathcal{E}. \tag{B.2}$$

Here again, notice that a high entropy $H(X \mid Y)$ implies a high reconstruction error in the best case.

b.1.1 Proof 1

We have

$$\mathcal{E} = \int_{\mathcal{Y}} p(y)\varepsilon(y)\,\mathrm{d}y \tag{B.3}$$

$$= \int_{\mathcal{Y}} p(y)p(X \neq \hat{x}(y) \mid y)\,\mathrm{d}y \tag{B.4}$$

$$= \int_{\mathcal{Y}} p(y)(1 - p(\hat{x}(y) \mid y))\,\mathrm{d}y. \tag{B.5}$$

Since

$$p(\hat{x}(y) \mid y) \leq p(\arg\max_{x \in X} p(x \mid y) \mid y), \tag{B.6}$$

the reconstruction that minimizes the error is $\hat{x}(y) = \arg\max_{x \in X} p(x \mid y)$. However, this is theoretical because in most cases $p(x \mid Y)$ is unknown.

b.1.2 Proof 2

We have:

$$\log(1 - \mathcal{E}) = \log\left(\int_{\mathcal{Y}} p(y)(1 - \varepsilon(y))\,\mathrm{d}y\right) \tag{B.7}$$

$$= \log\left(\int_{\mathcal{Y}} p(y)p(\hat{x}(y) \mid y)\right)\,\mathrm{d}y \tag{B.8}$$

$$\geq \int_{\mathcal{Y}} p(y)\log(p(\hat{x}(y) \mid y))\,\mathrm{d}y \tag{B.9}$$

$$= \int_{\mathcal{Y}} p(y)\int_{\mathcal{X}} p(x \mid y)\log(p(\hat{x}(y) \mid y))\,\mathrm{d}x\,\mathrm{d}y \tag{B.10}$$

$$\geq \int_{\mathcal{Y}} p(y)\int_{\mathcal{X}} p(x \mid y)\log(p(x \mid y))\,\mathrm{d}x\,\mathrm{d}y \tag{B.11}$$

$$= -H(X \mid Y). \tag{B.12}$$

The Eq. (B.9) is obtained using Jensen inequality and Eq. (B.11) is obtained using the result of Proof 1.

Thus,

$$\mathcal{E} \leq 1 - 2^{-H(X|Y)}. \tag{B.13}$$

b.2 SHADE Gradients

Here is studied the influence of SHADE on a gradient descent step for a single neuron Y of a single layer and for one training sample X. The considered case of a linear layer, we have: $Y = \mathbf{w}^\top X + b$.

The gradient of Ω_{SHADE} with respect to $\mathbf{w}$ is:

$$\begin{aligned}\nabla_{\mathbf{w}}\Omega_{\text{SHADE}} &= (\delta_1 + \delta_2)\mathbf{x} \\ \text{with } \delta_1 &= \sigma'(y)\big((y-\mu^1)^2 - (y-\mu^0)^2\big) \\ \text{and } \delta_2 &= 2\sigma(y)(y-\mu^1) + 2(1-\sigma(y))(y-\mu^0).\end{aligned}$$

With $\sigma(y) = p(B = 1|y)$ which has positive derivative. We can interpret the direction of this gradient by analyzing the two terms δ_1 and δ_2 as follows:

- $\boldsymbol{\delta_1}$: If $(y-\mu^0)^2$ is bigger than $(y-\mu^1)^2$ that means that y is closer to μ^1 than it is to μ^0. Then δ_1 is positive and it contributes to increasing y meaning that it increases the probability of B being from mode 1. In a way, it increases the average margin between positive and negative detections. Note that if there is no ambiguity about the mode of B meaning that $\sigma(y)$ or $1-\sigma(y)$ is very small then this term has negligible effect.
- $\boldsymbol{\delta_2}$: This term moves y toward the μ^b of the mode that presents the bigger probability. This has the effect of concentrating the outputs around their expectancy depending on their mode to get sparser activation.

BIBLIOGRAPHY

[Aba+16] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "TensorFlow: A System for Large-scale Machine Learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. URL: http://dl.acm.org/citation.cfm?id=3026877.3026899 (cit. on pp. 13, 64, 79).

[AF10] J. Deng A. Berg and L. Fei-Fei. "Large scale visual recognition challenge 2010". In: *www.imagenet.org/challenges* (2010) (cit. on p. 7).

[AG18] Rana Ali Amjad and Bernhard C. Geiger. "How (Not) To Train Your Neural Network Using the Information Bottleneck Principle". In: *ArXiv* (2018) (cit. on p. 59).

[Ale+17] A. A. Alemi, I. Fischer, J. V Dillon, and K. Murphy. "Deep Variational Information Bottleneck". In: *International Conference on Learning Representations*. 2017 (cit. on pp. 58, 76, 78).

[AM17] Xu Aolin and Raginsky Maxim. "Information-theoretic analysis of generalization capability of learning algorithms". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 2524–2533. URL: http://papers.nips.cc/paper/6846-information-theoretic-analysis-of-generalization-capability-of-learning-algorithms.pdf (cit. on pp. 12, 56, 57).

[Ans+18] F. Anselmi, G. Evangelopoulos, L. Rosasco, and T. Poggio. "Symmetry-adapted representation learning". In: 2018 (cit. on p. 44).

[Ant+15] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. "VQA: Visual Question Answering". In: *International Conference on Computer Vision (ICCV)*. 2015 (cit. on p. 4).

[Aro+18a] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. "Understanding Deep Neural Networks with Rectified Linear Units". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=B1J_rgWRW (cit. on p. 7).

[Aro+18b] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. "Understanding Deep Neural Networks with Rectified Linear Units". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=B1J_rgWRW (cit. on p. 43).

[AS16a] Achille A. and Soatto S. "Information Dropout: learning optimal representations through noisy computation". In: *ArXiv* (2016) (cit. on p. 57).

[AS16b] A. Achille and S. Soatto. "Information Dropout: learning optimal representations through noisy computation". In: *ArXiv* (2016) (cit. on p. 57).

[AS17] Alessandro Achille and Stefano Soatto. "Emergence of Invariance and Disentanglement in Deep Representations". In: *ArXiv*. 2017 (cit. on pp. 56, 58).

[BCT16] Michael Blot, Matthieu Cord, and Nicolas Thome. "Max-min convolutional neural networks for image classification". In: *IEEE International Conference on Image Processing (ICIP)*. 2016 (cit. on p. 15).

[Ben14] Graham Benjamin. "Fractional max-pooling". In: *arXiv preprint arXiv:1412.6071* (2014) (cit. on p. 57).

[BL16] Ruitong Huang Bing Xu and Mu Li. "Revised Saturated Activation Functions". In: *ArXiv* (2016) (cit. on p. 7).

[BLM02] Bartlett, Peter L, and Shahar Mendelson. "Rademacher and Gaussian complexities: Risk bounds and structural results". In: *Journal of Machine Learning Research* (2002) (cit. on pp. 6, 43).

[Blo+18a] Michael Blot, David Picard, Nicolas Thome, and Matthieu Cord. "Distributed Optimization for Deep Learning with Gossip Exchange". In: *Under review at Neuro Computing*. 2018 (cit. on p. 14).

[Blo+18b] Michael Blot, Thomas Robert, Nicolas Thome, and Matthieu Cord. "SHADE: SHAnnon DEcay Information-Based Regularization for Deep Learning". In: *IEEE International Conference on Image Processing (ICIP)*. 2018. URL: https://arxiv.org/pdf/1805.05814.pdf (cit. on pp. 15, 74).

[BM13] Joan Bruna and Stephane Mallat. "Invariant Scattering Convolution Networks". In: *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* (2013) (cit. on p. 58).

[Bor15] Nicolas Borie. "Effective Invariant Theory of Permutation Groups using Representation Theory". In: *ArXiv*. 2015 (cit. on p. 44).

[Bot10] Leon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010 (cit. on pp. 11, 18).

[Boy+06] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. "Randomized Gossip Algorithms". In: *IEEE transaction on information theory* (2006) (cit. on pp. 14, 18, 29).

[Boy+11] S Boyd, N Parikh, E Chu, B Peleato, and J. Eckstein. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Found. Trends Mach. Learn., 3(1):1–122* (2011) (cit. on p. 32).

[BPL10] Y-Lan Boureau, Jean Ponce, and Yann Lecun. "A Theoretical Analysis of Feature Pooling in Visual Recognition". In: *ICML* (2010) (cit. on p. 44).

[BRM16] Tadas Baltrusaitis, Peter Robinson, and Louis-Philippe Morency. "OpenFace: An open source facial behavior analysis toolkit." In: *WACV*. IEEE Computer Society, 2016, pp. 1–10. URL: http://dblp.uni-trier.de/db/conf/wacv/wacv2016.html#BaltrusaitisOM16 (cit. on p. 1).

[BS97] Manjunath B.S. "NeTra: a toolbox for navigating large image databases". In: *ICIP* (1997) (cit. on p. 6).

[BSC18] Michael Blot, Antoine Saporta, and Matthieu Cord. "Reve: Regularizing deep learning with variational entropy bound". In: *Under review at NIPS*. 2018 (cit. on p. 15).

[Büh04] Peter Bühlmann. *Bagging, boosting and ensemble methods*. Papers 2004,31. Humboldt University of Berlin, Center for Applied Statistics and Economics (CASE), 2004. URL: https://EconPapers.repec.org/RePEc:zbw:caseps:200431 (cit. on p. 4).

[CBM02] Ronan Collobert, Samy Bengio, and Johnny Marithoz. *Torch: A Modular Machine Learning Software Library*. 2002 (cit. on pp. 13, 33, 50).

[CG16] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. URL: http://doi.acm.org/10.1145/2939672.2939785 (cit. on p. 4).

[Che+15] Marion Chevalier, Nicolas Thome, Matthieu Cord, Jerome Fournier, Gilles Henaff, and Elodie Dusch. "LR-CNN For Fine-grained Classification with Varying Resolution". In: *International Conference on Image Processing (ICIP)*. 2015 (cit. on p. 13).

[Che+16] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. 2016 (cit. on p. 62).

[Che+18] Xiuyuan Cheng, Qiang Qiu, A. Robert Calderbank, and Guillermo Sapiro. "RotDCF: Decomposition of Convolutional Filters for Rotation-Equivariant Deep Networks". In: *CoRR* abs/1805.06846 (2018). arXiv: 1805.06846. URL: http://arxiv.org/abs/1805.06846 (cit. on p. 44).

[Cho+15] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gerard Ben Arous, and Yann LeCun. "The Loss Surfaces of Multilayer Networks". In: *AISTATS*. 2015 (cit. on p. 23).

[Col+16] Igor Colin, Aurelien Bellet, Joseph Salmon, and Stephan Clemencon. "Gossip Dual Averaging for Decentralized Optimization of Pairwise Functions". In: *ICML*. 2016 (cit. on p. 18).

[CT91] T. Cover and J. Thomas. "Elements of information theory". In: *Wiley New York* (1991) (cit. on pp. 57, 58, 60, 62, 93).

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)." In: *CoRR* abs/1511.07289 (2015). URL: http://dblp.uni-trier.de/db/journals/corr/corr1511.html#ClevertUH15 (cit. on p. 45).

[CV16] S. Ioffe C. Szegedy and V. Vanhoucke. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *arXiv:1602.07261*. 2016 (cit. on p. 18).

[Dea+12] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. "Large Scale Distributed Deep Networks". In: *NIPS* (2012) (cit. on pp. 18, 23, 27, 28, 85).

[DF13] Zeiler Matthew D and Rob Fergus. "Stochastic pooling for regularization of deep convolutional neural networks". In: *arXiv preprint arXiv:1301.3557* (2013) (cit. on p. 57).

[Di +11] Giuseppe Di Fatta, Francesco Blasa, Simone Cafiero, and Giancarlo Fortino. "Epidemic k-means clustering". In: *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*. IEEE. 2011, pp. 151–158 (cit. on p. 18).

[Din+17] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. "Sharp Minima Can Generalize For Deep Nets". In: *arXiv e-prints* 1703.04933 (2017). URL: https://arxiv.org/abs/1703.04933 (cit. on p. 56).

[DNW17] Eli David, Nathan S. Netanyahu, and Lior Wolf. "DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess". In: *CoRR* abs/1711.09667 (2017) (cit. on p. 1).

[DS12] Ueli Meier Dan Cires an and Jurgen Schmidhuber. "Multi-column Deep Neural Networks for Image Classification". In: *CVPR* (2012) (cit. on pp. 51, 52).

[DTC16] Thibaut Durand, Nicolas Thome, and Matthieu Cord. "WELDON: Weakly Supervised Learning of Deep Convolutional Neural Networks". In: *CVPR*. 2016 (cit. on pp. 6, 45, 65, 83).

[Eve+15] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (2015), pp. 98–136 (cit. on p. 4).

[FCP01] J. Fournier, M. Cord, and S. Philipp-Foliguet. "RETIN: A Content-Based Image Indexing and Retrieval System". In: *Pattern Analysis and Applications Journal, Special issue on image indexation* 4.2/3 (2001), pp. 153–173. URL: http://publi-etis.ensea.fr/2001/FCP01a (cit. on p. 6).

[FPG13] Jerome Fellus, David Picard, and Philippe-Henri Gosselin. "Decentralized K-means using randomized Gossip protocols for clustering large datasets". In: *IEEE 13th International Conference on Data Mining Workshops*. IEEE. 2013, pp. 599–606 (cit. on p. 18).

[FPG15] Jerome Fellus, David Picard, and Philippe-Henri Gosselin. "Asynchronous gossip principal components analysis". In: *Neurocomputing, Elsevier, 2015* (2015) (cit. on p. 18).

[Fuk80] K. Fukushima. "Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics, 36(4):193–202* (1980) (cit. on p. 7).

[GE03] I. Guyon and A. Elisseeff. "An Introduction to Variable and Feature Selection". In: *J. Mach. Learn. Res., vol. 3, pp. 1157–1182* (2003) (cit. on p. 4).

[GG17] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of the International Conference on Machine Learning*. 2017 (cit. on pp. 56, 57).

[GL15] Yaroslav Ganin and Victor Lempitsky. "Unsupervised Domain Adaptation by Backpropagation". In: *ICML*. 2015, pp. 1180–1189 (cit. on pp. 66, 67).

[Goh+13] Hanlin Goh, Nicolas Thome, Matthieu Cord, and Joo-Hwee Lim. "Top-down Regularization of Deep Belief Networks". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 1878–1886. URL: http://dl.acm.org/citation.cfm?id=2999792.2999822 (cit. on pp. 50, 58).

[Goo+13] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout Networks". In: *JMLR* (2013) (cit. on p. 52).

[Hau90] David Haussler. "Probably Approximately Correct Learning". In: *Proceedings of the Eighth National Conference on Artificial Intelligence*. 1990 (cit. on p. 3).

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *arXiv:1502.01852v1* (2015) (cit. on p. 45).

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *CVPR*. 2016 (cit. on pp. 6, 9, 11, 18, 22, 43, 65, 83).

[Hin+12] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012) (cit. on pp. 12, 57, 65).

[Hin13] Geoffrey Hinton. "Overview of mini-batch gradient descent". In: *GNeural Networks for Machine Learning Lecture 6a* (2013) (cit. on p. 21).

[HMD15] Song Han, Huizi Mao, and William J. Dally. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". In: (2015). cite arxiv:1510.00149Comment: Published as a conference paper at ICLR 2016 (oral). URL: http://arxiv.org/abs/1510.00149 (cit. on p. 58).

[Hua+17] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on pp. 22, 81).

[HV17] Benjamin D. Haeffele and Rene Vidal. "Global Optimality in Neural Network Training". In: *CVPR* (2017) (cit. on p. 11).

[IS16] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Journal of Machine Learning Research* (2016) (cit. on pp. 11, 18, 43, 57).

[JDe+09] J.Deng, W. Dong, R. Socher, L.-J. Li, and L. Fei-Fei. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR* (2009) (cit. on pp. 4, 18, 65, 83).

[JSO18] Jörn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. "i-RevNet: Deep Invertible Networks". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=HJsjkMb0Z (cit. on pp. 12, 43, 53, 74, 87).

[Kar15] Andrew Zisserman Karen Simonyan. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *ICLR* (2015) (cit. on pp. 9, 20, 42).

[KB15] D. P. Kingma and J. L. Ba. "Adam: a Method for Stochastic Optimization". In: *International Conference on Learning Representations* (2015) (cit. on pp. 11, 21).

[KDG03] David Kempe, Alin Dobra, and Johannes Gehrke. "Gossip-based computation of aggregate information". In: *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*. IEEE. 2003, pp. 482–491 (cit. on p. 30).

[Kes+16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: (2016). arXiv: 1609.04836 (cit. on pp. 12, 56).

[Kes+17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large batch training for deep learning: generalization gap and sharp minima". In: *ICLR*. 2017 (cit. on p. 57).

[KH92] Anders Krogh and John A. Hertz. "A Simple Weight Decay Can Improve Generalization". In: *Advances in Neural Information Processing Systems*. 1992 (cit. on pp. 12, 57, 65).

[KI14] Alex Krizhevsky and Google Inc. *One weird trick for parallelizing convolutional neural networks*. Tech. rep. 2014 (cit. on p. 22).

[Kla+17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. "Self-Normalizing Neural Networks." In: *CoRR* abs/1706.02515 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1706.html#KlambauerUMH17 (cit. on p. 45).

[KLS11] Matej Kristan, Aleš Leonardis, and Danijel Skočaj. "Multivariate Online Kernel Density Estimation with Gaussian Kernels". In: *Pattern Recogn.* 44.10-11 (Oct. 2011), pp. 2630–2642. URL: http://dx.doi.org/10.1016/j.patcog.2011.03.019 (cit. on p. 61).

[Kri09] A. Krizhevsky. "Learning multiple layers of features from tiny images". PhD thesis. Computer Science Department University of Toronto, 2009 (cit. on pp. 4, 35, 51, 80).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *NIPS*. 2012 (cit. on pp. 6–9, 12, 13, 22, 42, 50–52, 56, 79).

[KW14] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations*. 2014 (cit. on p. 62).

[Lan12] Guanghui Lan. "An optimal method for stochastic composite optimization". In: *Mathematical Programming* 133.1 (2012), pp. 365–397 (cit. on p. 27).

[LCY14] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: *ICLR* (2014) (cit. on pp. 9, 43).

[LeC+98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradientbased learning applied to document recognition. Proceedings". In: *IEEE* (1998) (cit. on pp. 1, 7, 44, 51).

[Lec+98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient based learning applied to document recognition". In: *IEEE*. 1998 (cit. on pp. 4, 7, 12).

[Li+18] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. "Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift". In: *CoRR* abs/1801.05134 (2018) (cit. on p. 12).

[Lin+15] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollar. "Microsoft COCO: Common Objects in Context". In: *arxiv* (2015) (cit. on pp. 4, 18).

[LL17] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf (cit. on p. 11).

[LW17] Jian-Hao Luo and Jianxin Wu. "An Entropy-based Pruning Method for CNN Compression". In: (2017). eprint: arXiv:1706.05791. URL: https://arxiv.org/abs/1706.05791 (cit. on p. 58).

[LY08] David G Luenberger and Yinyu Ye. *Linear and nonlinear programming*. 2008 (cit. on pp. 3, 19, 21).

[Mal12] Stephane Mallat. "Group invariant scattering". In: *Communications on Pure and Applied Mathematics*. 2012 (cit. on pp. 44, 58).

[MBC03] Ramchandra Manthalkar, P. K. Biswas, and B. N. Chatterji. "Rotation Invariant Texture Classification Using Even Symmetric Gabor Filters". In: *Pattern Recogn. Lett.* 24.12 (Aug. 2003), pp. 2061–2068. URL: http://dx.doi.org/10.1016/S0167-8655(03)00043-6 (cit. on p. 44).

[MMT16] He Ma, Fei Mao, and Graham W. Taylor. "Theano-MPI: a Theano-based Distributed Training Framework". In: *arXiv* (2016). arXiv: 1605.08325v1 (cit. on pp. 22, 23).

[Mni+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013. 2013. URL: http://arxiv.org/abs/1312.5602 (cit. on p. 1).

[Net+11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. "Reading Digits in Natural Images with Unsupervised Feature Learning". In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. 2011 (cit. on pp. 4, 81).

[NN15] Tishby Naftali and Zaslavsky Noga. "Deep learning and the information bottleneck principle". In: *Information Theory Workshop (ITW)*. IEEE. 2015 (cit. on pp. 12, 43, 56, 60, 87).

[NY72] Vapnik Vladimir N and Chervonenkis A Ya. "On the uniform convergence of relative frequencies of events to their probabilities". In: *Measures of Complexity*. Springer, 1972 (cit. on pp. 3, 4, 6, 43).

[OF96] Bruno A. Olshausen and David J. Field. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images?" In: *Nature* (1996) (cit. on pp. 50, 58).

[OMS17] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization". In: *Distill* (2017). https://distill.pub/2017/feature-visualization (cit. on p. 74).

[Pan03a] Liam Paninski. "Estimation of Entropy and Mutual Information". In: *Neural Computation* (2003) (cit. on p. 61).

[Pan03b] Liam Paninski. "Estimation of Entropy and Mutual Information". In: *Neural Comput.* 15.6 (June 2003), pp. 1191–1253. URL: http://dx.doi.org/10.1162/089976603321780272 (cit. on p. 61).

[Per+17] Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton. "Regularizing Neural Networks by Penalizing Confident Output Distributions". In: *International Conference on Learning Representations Workshop*. 2017 (cit. on p. 58).

[Ris78] J. Rissanen. "Modeling by shortest data description". In: *Automatica* (1978) (cit. on p. 56).

[RSS12] Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. "Making Gradient Descent Optimal for Strongly Convex Stochastic Optimization". In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 1571–1578. URL: http://dl.acm.org/citation.cfm?id=3042573.3042774 (cit. on p. 19).

[Sch01] Bernhard Schölkopf. "Learning with kernels". In: 2001 (cit. on p. 4).

[Sch99] Robert E. Schapire. "A Brief Introduction to Boosting". In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'99. Stockholm, Sweden: Morgan Kaufmann Publishers Inc., 1999, pp. 1401–1406. URL: http://dl.acm.org/citation.cfm?id=1624312.1624417 (cit. on p. 4).

[Sha+16] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. "Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units". In: *ICLM* (2016) (cit. on pp. 46, 53).

[Sil+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (2016), pp. 484–489 (cit. on p. 1).

[Siv+05] J. Sivic, B.C. Russell, A. Efros A.A.and Zisserman, and W.T Freeman. "Discovering objects and their location in images". In: *ICCV* (2005) (cit. on p. 6).

[SST10] O. Shamira, S. Sabato, and N. Tishby. "Learning and generalization with the information bottleneck". In: *Theoretical Computer Science* (2010) (cit. on pp. 56, 59).

[Sun+17] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: *IEEE International Conference on Computer Vision (ICCV)*. 2017 (cit. on p. 18).

[Sut+13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. Ed. by Sanjoy Dasgupta and David Mcallester. Vol. 28. 3. JMLR Workshop and Conference Proceedings, 2013, pp. 1139–1147 (cit. on pp. 11, 27).

[Sze+14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842 (cit. on pp. 64, 79).

[Sze+15] Christian Szegedy, Yangqing Jia Wei Liu, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions". In: *CVPR* (2015) (cit. on pp. 9, 43, 45).

[SZS05] Sarit Shwartz, Michael Zibulevsky, and Yoav Y. Schechner. "Fast kernel entropy estimation and optimization". In: *Signal Processing* 85 (2005), pp. 1045–1058 (cit. on p. 61).

[TPB99] N. Tishby, F. C. Pereira, and W. Bialek. "The information bottleneck method". In: *Annual Allerton Conference on Communication, Control and Computing* (1999) (cit. on pp. 15, 58).

[Val13] Leslie Valiant. *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World*. New York, NY, USA: Basic Books, Inc., 2013 (cit. on p. 3).

[Vap95] V Vapnik. *The Nature of Statistical Learning Theory*. 1995 (cit. on p. 19).

[Vap98] V. Vapnik. "Statistical Learning Theory". In: *Wiley-Interscience* (1998) (cit. on pp. 4, 6, 57).

[VL15] Andrea Vedaldi and Karel Lenc. "MatConvNet: Convolutional Neural Networks for MATLAB". In: *Proceedings of the 23rd ACM International Conference on Multimedia*. MM '15. Brisbane, Australia: ACM, 2015, pp. 689–692. URL: http://doi.acm.org/10.1145/2733373.2807412 (cit. on pp. 13, 51).

[Wan+13a] L Wan, M. D Zeiler, S Zhang, Y LeCun, and R Fergus. "Regularization of neural networks using dropconnect". In: *ICML* (2013) (cit. on pp. 32, 37).

[Wan+13b] Li Wan, Matthew D. Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. "Regularization of Neural Networks using DropConnect." In: *ICML (3)*. Vol. 28. JMLR Proceedings. JMLR.org, 2013, pp. 1058–1066. URL: http://dblp.uni-trier.de/db/conf/icml/icml2013.html#WanZZLF13 (cit. on p. 57).

[Wan+15] Xin Wang, Devinder Kumar, Nicolas Thome, Matthieu Cord, and Frederic Precioso. "Recipe recognition with large multimodal food dataset". In: *ICME Workshops*. IEEE Computer Society, 2015, pp. 1–6 (cit. on p. 4).

[Xu+15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. "Empirical Evaluation of Rectified Activations in Convolutional Network." In: *CoRR* abs/1505.00853 (2015). URL: http://dblp.uni-trier.de/db/journals/corr/corr1505.html#XuWCL15 (cit. on p. 45).

[ZCL15] Sixin Zhang, Anna Choromanska, and Yann LeCun. "Deep learning with Elastic Averaging SGD". In: *NIPS*. 2015 (cit. on pp. 18, 22, 23, 27, 28, 32, 35, 85).

[Zha+17] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. "Understanding deep learning requires rethinking generalization". In: *International Conference on Learning Representations* (2017) (cit. on pp. 12, 42, 43, 56, 57, 64, 65, 73, 79).

[Zha+18] Wenjie Zhang, Junchi Yan, Xiangfeng Wang, and Hongyuan Zha. "Deep Extreme Multi-label Learning". In: *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. ICMR '18. Yokohama, Japan: ACM, 2018, pp. 100–107. URL: http://doi.acm.org/10.1145/3206025.3206030 (cit. on p. 13).

[ZK16] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks". In: *arXiv*. 2016 (cit. on pp. 43, 64, 79).

[ZLL16] Yuting Zhang, Kibok Lee, and Honglak Lee. "Augmenting supervised neural networks with unsupervised objectives for large-scale image classification". In: *International Conference on Machine Learning*. 2016, pp. 612–621 (cit. on p. 57).