Machine learning under budget constraints

Gabriella Contardo

UNIVERSITÉ PIERRE ET MARIE CURIE

DOCTORAL THESIS

Machine Learning
Under Budget Constraints

Author:
Gabriella CONTARDO

Supervisor:
Pr. Ludovic DENOYER
Pr. Thierry ARTIÈRES

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Computer Science

in the

Machine Learning and Information Access team
Laboratoire d'Informatique de Paris 6
École doctorale Informatique, Télécommunications et Électronique (Paris)

Defended July 10th, 2017, with the following jury:

Pr. Olivier PIETQUIN	Reviewer
Pr. Marc SEBBAN	Reviewer
Pr. Anne DOUCET	Examiner
Dr. Balàzs KEGL	Examiner
Dr. Nicolas USUNIER	Examiner
Pr. Thierry ARTIERES	Supervisor
Pr. Ludovic DENOYER	Supervisor

Acknowledgements

Je tiens à remercier particulièrement mes deux directeurs, qui m'ont donné l'opportunité de faire cette thèse et qui m'ont encadrée pendant ces trois années, neuf mois et quelques jours (bon et encore un peu après j'espere). Merci pour vos conseils, votre soutien et votre patience, et aussi pour m'avoir permis de partir en conférence dans pleins d'endroits sympa. D'une manière plus générale, merci pour avoir fait de ces quatres années un peu plus qu'un simple diplôme à passer ou un boulot où aller le matin (même si je rale parfois). Et désolée pour mon hypothétique responsabilité dans ta perte de cheveux Ludovic :)
Merci également à Olivier Pietquin et Marc Sebban pour le temps qu'ils ont consacré à raporter ce manuscrit et leurs remarques, ainsi qu'à Anne Doucet, Balazs Kegl et Nicolas Usunier pour leur participation à mon jury de soutenance.

Merci aussi aux copains et copines du labo, pour l'ambiance, le soutien moral, les conversations plus ou moins dérisoires, les bières, et toutes ces choses qui ont également fait le fun de ces quatre années ici. Merci aussi aux ami-e-s de l'exterieur, à Evrard et à ma famille.

Je remercie également l'Etat français pour avoir financé ma thèse (et ma scolarité d'une manière plus générale), et enfin merci au personnel administratif et ingés du laboratoire pour leur travail, leur aide et leur patience (!).

Abstract

A machine learning method relies on a variety of information, such as examples, inputs' features and labels. These different types of information can be costly to acquire (human resources, time, money...). We propose to focus on *cost-sensitive methods for prediction*. The goal of this thesis is to develop methods that predict *under a budget*. The main contributions presented in this manuscript consider the **features' cost**.

We begin with an overview of the different problems in *budgeted learning*, and a more thorough study of related works on **feature acquisition** for cost-sensitive problems at test-time. We also provide a brief review of active-learning and meta-learning techniques, and a more technical report on recurrent neural architectures which are a key component of most of our models.

Our first contribution presents a **static feature acquisition** method. The goal is to find the best subset of features to acquire, for any input, in order to predict correctly. We focus on the user cold-start problem in recommender system applications. In this setting, the problem translates into finding the best questions to ask a new user to provide the more relevant recommendations afterward. We propose to use a representation-learning approach, where a user is shifted in the latent space depending on his answers. We learn the best-subset of questions to ask through a set of continuous weights. The cost constraint is directly integrated into the loss of our model. We demonstrate on classical recommender datasets the advantages of our approach.

We continue by studying the more generic problem of **adaptive feature acquisition**, where we aim at designing systems that have a sequential acquisition process. This allows the model to adapt its acquisition behavior depending on the values of the features acquired before. Thus, it should provide a better trade-off between acquisition cost and prediction quality. We present a framework that relies on a latent representation space. The key idea is that the system sequentially builds a representation of the input, by aggregating the acquired features. This representation guides the acquisition process. As before, the cost constraint is integrated explicitly in the loss. We propose a continuous relaxation that allows us to use fast gradient descent algorithms for learning. We illustrate on datasets of various sizes the efficiency of this model.

We propose in a second time to remove the continuous relaxation. We define an upper bound of the loss we defined originally for the problem. We present a stochastic

instantiation of this framework for this loss, using policy-gradient inspired techniques to learn in such setting. Comparison of experimental results with the continuous model shows that the stochastic method gives competitive results but requires more training iterations due to the Monte-Carlo sampling.
The last part of this thesis studies the **labels' costs**. We present preliminary works on **meta active learning**. Active-learning aims at designing strategies to choose which examples is best to label during training. We propose a meta-learning approach to the problem, inspired by recent works in one-shot learning. In this setting, the goal is to design systems that *learn to actively learn*, on several tasks. We define a model that considers all examples of a dataset before predicting which example should be labeled. Results on artificial and real datasets show encouraging performance and open several leads for further research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Classical supervised machine learning

Machine learning techniques are now recognized as reliable and efficient methods to tackle a large spectrum of tasks. They have proven their ability to solve problems in various domains, from visual computing (image classification, captioning and labeling, object recognition) to natural language processing (e.g translation). They have also been successfully applied on research fields unrelated to machine learning at first sight, such as medical care, biological analysis or even asteroids detection and classification.
A classical approach is to describe the generic supervised machine learning problem as a loss-based approach. The goal is to learn a model M based on a dataset $\mathcal{D}$. This dataset is usually composed of a certain number of examples x_i, taken as *inputs* by the model, and their corresponding supervision y_i. This supervision can be labels of categories (classification) or real numbers (regression). The purpose of the model is, given x_i, to predict y_i as accurately as possible. Thus, training can usually be written as finding the model M^* that minimizes the loss of the model as follow, where Δ computes the error between the model's prediction and the expected output y_i:

$$M^* = \arg\min_M \sum_{x_i \in \mathcal{D}} \Delta(y_i, M(x_i))$$

The various models in machine learning then differ on several aspects, such as the use of parameters (e.g potentially non-parametric methods such as decision tree), the learning strategies and algorithms (e.g various algorithms for gradient descent in neural networks) or the architecture used (e.g neural networks).
However, the prediction performance is usually the only component of the learning criterion, and it is also often the only aspect considered to evaluate a machine learning method. This illustrates how these approaches frequently rely on assumptions about the "freeness" of the information used and its associated costs. For instance, they often consider that all necessary data and resources come for free. The quantity of information, the computation time or the memory used, are not considered as impacting the quality of a model.

1.2 Budgeted learning

We can distinguish several aspects in the learning and inference processes of a machine learning model that may be costly for different reasons. First, the examples are crucial for machine learning approaches. They are supposedly representative of the current problem, and they guide the model to find some underlying principles (e.g statistics) which help to solve the task. They are therefore necessary to learn correctly. Some methods, especially (deep) neural networks, require a lot of examples to be trained and to obtain good performances. This prevents the use of these methods on several tasks where *examples* are costly. Typically, medical tasks such as "brain-reading" from fMRI, or genetic mutation detection on genome sequences, can have costly examples, in terms of money (e.g extracting genome sequences), time (e.g finding voluntary patients to conduct the experiments), or unquantifiable cost (e.g invasiveness of the experiment for a patient). Similarly, not only each example may have a cost, but its associated label can also be expensive. Indeed, supervision may be done through human and manual labeling, which induce time and money costs.
Furthermore, when we consider the examples, we can notice that they are also composed of bits of information, the features, which can likewise have a cost. For instance, we mentioned above the problem of medical diagnosis. For a patient, each result of a medical test is a "feature". This highlights the need for constraining a number of features a system needs to predict on an input at test-time. Indeed, medical tests have a cost (time, money, or even risk, invasiveness, pain), and it is not possible to take all possible tests on a patient. Obtaining features is not trivial in various applications and can induce a cost associated to *feature acquisition*.
In parallel of these information required for learning and predicting, a model may use other resources, such as time, electricity, computing memory or capacity. These are often ignored, but they prove crucial in various domain, especially with the development of more and more complex models in the field of deep learning.

All these aspects illustrate the need for **budgeted learning methods**, which integrate these costs in their processes. The overall goal is to provide systems that can interact with the data and resources, that develop some acquisition strategy in order to perform well on a problem while keeping on a restricted budget, as illustrated schematically in Figure 1.1.
We propose in this thesis to focus on two specific budgeted aspects, with two types of costly information: features and labels. Our goal is to design systems that learn how to adapt their decision process with regard to the cost of information.
Most contributions presented in this manuscript focuses on **cost-sensitive feature acquisition**. We keep a "classical" machine learning prediction setting, however, the goal is to have a system that "interacts" with the example to predict on during inference, by acquiring the relevant features. We also provide in a second time

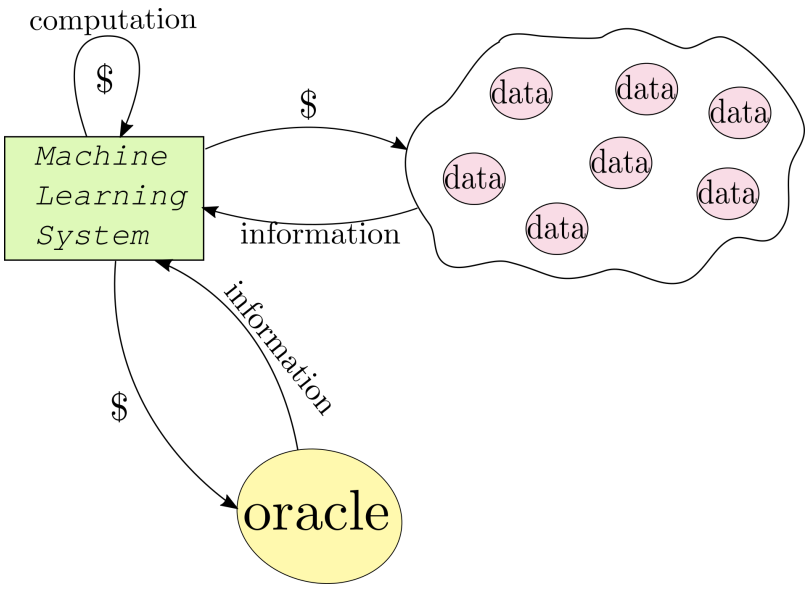


FIGURE 1.1: Schematic illustration of the various costly components a machine learning system interacts with: data information (e.g. examples, features), information coming from an oracle (e.g. labels), computation resources (e.g. memory or CPU resources to learn).

a preliminary work for the problem of **costly labels** (usually referred to as *active learning*). We choose to follow a novel *meta-learning* approach for this problem, to learn how to select the best examples to label when faced with a new problem.

1.3 Thesis structure

1.3.1 Chapter 3 : Static Feature Selection

We study in Chapter 3 the problem of static feature selection. The goal is to find the best subset of features (common to all inputs) that provides the better trade-off between cost and prediction quality. While being restrained by the non-adaptive ability, it can prove useful for some applications, such as recommender systems, more specifically for the user *cold-start problem*. A new user enters a recommendation system (e.g IMDb), and the system has no information about him or her. An efficient way of providing rapidly relevant recommendations is to ask him a few questions to learn more about his tastes. However, it has been shown that users don't like long interviews with several steps. This leads to the need of a "single shot" interview process with few questions. This translates easily within this framework in a static feature selection problem, where a feature is an answer to a question, typically the rating a user gives to an item.
We propose a model that learns conjointly the best subset of features and a prediction model. Previous approaches usually focus on designing the features' subset

only. This difference allows us to have a robust model, which is efficient for cold-start but can also be used in the classical "warm" recommendation setting.We rely on a **representation learning approach**, and the framework can be instantiated as a fully differentiable model. Thus, we can learn our approach with efficient and scalable methods based on gradient descent.
Our work on static feature selection for recommendation and the user cold-start problem has led to a publication in an international conference in the poster track:

- Contardo Gabriella, Denoyer Ludovic and Artieres Thierry, (2015a) "Representation Learning for cold-start recommendation", In *International Conference in Learning Representation (poster) ICLR 2015*.

1.3.2 Chapters 4 and 5 : Adaptive Feature acquisition

While appropriate for the specific problem of cold-start in recommender systems, a static selection process suffers from a hard constraint to provide good performance. Indeed, the subset is the same for all the possible inputs. It doesn't have the ability to refine its perception, as it can not choose one specific feature for an input instead of another. This highlights the need for **adaptive methods**, where feature acquisition is done **sequentially**, to provide a better trade-off between cost and performance.
We present in Chapters 4 and 5 two models for this problem. They are also based on representation learning, and can be instantiated as **recurrent neural networks** (RNN) architectures. The RNN sequentially interacts with the input, by acquiring features at each step of the process, and predicts an output at the final step. Here again, we learn the acquisition process along with the prediction process. Both of our models allow **batch acquisition** of features, i.e the system can ask for several features at a given time step, which is novel compared to state of the art techniques in this regard. Both models can also consider non-uniform costs.
Our work on adaptive feature acquisition has led to two international conference proceedings:

- Contardo Gabriella, Denoyer Ludovic and Artieres Thierry, (2016a) "Recurrent neural networks for adaptive feature acquisition", In *International Conference on Neural Information Processing*, pages 591–599, Springer, Best student paper award.
- Contardo Gabriella, Denoyer Ludovic and Artieres Thierry, (2016b) "Sequential Cost-Sensitive Feature Acquisition", In *International Symposium on Intelligent Data Analysis*, pages 284–294, Springer.

1.3.3 Chapter 6 : Meta Active Learning

Finally, Chapter 6 focuses on the active learning problem, where the goal is to constrain the number of labels used for learning. More precisely, we provide an introductory work for a meta-learning approach on this problem. We propose first to define the several fields related to the problem. We then define our learning-setting. It is inspired by one-shot meta-learning techniques that have been recently proposed. The key idea is that the model is trained, at each learning step, on a complete classification problem. However, unlike previous one-shot methods, the system can observe all the examples of the current problem, close to a *pool-based* scenario in active learning. We design a model for this framework that relies on specific **recurrent architecture** that allows the network to consider all the elements of the current "data-point" (i.e the dataset of the current problem) to predict which examples to label. We suggest that this additional, while unsupervised information, may help the system to obtain better results. We show preliminary results to illustrate the relevance of our approach. We also provide insights to design a hybrid model, which takes labeling decision sequentially.
This work is still in progress and has not been published in a conference yet.

1.3.4 Other contributions

This thesis has also lead to other contributions that are not presented in this manuscript. We first worked on representation-learning for reinforcement learning on partially observable environments. The key idea was to propose a dynamical representation model, which aims at learning the underlying "physics" of the environment. More precisely, the representations are built in a "transductive" fashion: they are not solely predicted by a function but are optimized at each step to integrate new observations. The dynamical model is learned in an unsupervised fashion, and is then used for policy learning through the representations. One interesting aspect of the model is that it results in a model that can be "blind": it can still take decisions (based on the representations and the dynamical model of the environment) even if it receives no observation. This is tied with the **budgeted learning** topic of this thesis, however, in this model there is no control to acquire information or not. This work was then enhanced and extended to time-series completion and prediction. It led to the following publications:

- Gabriella Contardo, Ludovic Denoyer, Thierry Artieres, Patrick Gallinari (2014) "Learning States Representations in POMDP", In *International Conference on Learning Representations (poster) ICLR 2014* .

- Ali Ziat, Gabriella Contardo, Nicolas Baskiotis and Ludovic Denoyer, (2015b) "Car-Traffic Forecasting: A Representation Learning Approach,"" In *Proceedings of the 2nd International Workshop on Mining Urban Data, co-located with 32nd International Conference on Machine Learning (ICML)*,pages 85–87.
- Ali Ziat, Gabriella Contardo, Nicolas Baskiotis, Ludovic Denoyer (2016c) "Learning Embeddings for Completion and Prediction of Relationnal Multivariate Time-Series", In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning - ESANN*

Chapter 2

Supervised learning under budget constraints: Background

Abstract: We provide an overview of the different techniques and domains this thesis is linked to. The first part of this chapter focuses on the problem of budgeted learning. We detail the different aspects the budget can take, be it during learning or inference, or at different levels of information (features, examples, labels). We then focus our review on the two aspects we propose to study in this thesis : (i) integrating the budget of features acquisition in the learning process to limit the inference cost (*cost-sensitive inference*), (ii) the cost of acquiring labeled examples when learning a new task (*cost-sensitive active (meta) learning*). Since several models presented in this manuscript are based on recurrent neural networks (RNN) architectures, which have proven over the years their efficiency, especially for sequential tasks, the last part of the chapter provides a description of various RNN models.

2.1 Budgeted Learning

The classical setting to tackle a learning problem considers only the prediction performance as an optimization criterion, be it in classification or regression. The general assumption is that all data comes at no cost: we can access all the examples labeled during training, and all inputs' features. However, this is a strong assumption. Indeed, data needed to learn a machine learning system have a cost. For example, labeling a big dataset of pictures requires human resources. Obtaining examples or specific features can also be costly, for instance if they require extensive computation. This observation has lead to a research domain that can be broadly named *budgeted learning*. It aims at taking these types of information costs into the development of learning systems.

2.1.1 An overview of the different types of cost

We now define the various types of costs in a machine learning system. We choose to divide the approaches along two axes: the nature of the costly information considered, and the time when the constraint is considered (i.e training or inference).

Nature of information We discern three "levels" of information in a learning system, each of them being potentially "costly": (i) the examples, for instance in some medical application e.g. genomic sequencing, where obtaining one sample is very expensive. (ii) The features describing these examples, e.g still in medical applications, some test might be more expensive (e.g fMRI vs blood sampling) or risky for a patient. (iii) The labels associated with each example, which often come from human knowledge, thus require time and money resources.

Period of constraint We differentiate between methods aiming at **learning** under a budget constraint *vs* methods aiming at **inferring** under a budget. The first approaches will rely on specific learning strategies to optimize a trade-off between the information used and the performance. This happens for instance in the *active learning* case, when labeling examples is costly. Designing algorithms that choose specifically which examples to label while achieving good performance allow to reduce the overall cost of the system. In the second case, the cost constraint is considered **at test-time**, where the information used for prediction during inference is "budgeted". The system will have no budget limit at train-time, but it still may have to consider the cost constraint during learning. For example, one can design a specific heuristic for the considered cost and force the model to learn under certain "rules". Differently, the cost could be integrated into the decision process, as an additional element of the optimized criterion.

Note that the **nature of cost** can also be considered as a third relevant aspect, depending if the cost is specific for each element of information (see Section 2.1.2 where each feature has a specific cost, or if examples from a particular class are harder to obtain), or is uniform (e.g classical active learning where all labels have the same cost).

Taxonomy of budgeted learning methods

We now provide a brief taxonomy of the different methods which cover these two axes. A summary of these settings is illustrated in Figure 2.1.

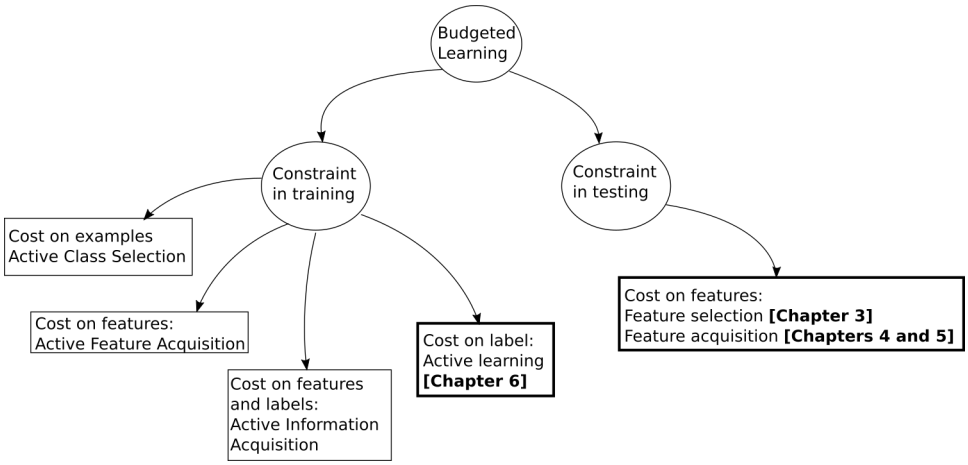


FIGURE 2.1: Graphical representation of the "budgeted learning" taxonomy.

Integration of budget constraint during training

Active Learning This field of research focuses on algorithms that select the instances to label during the learning phase. Different settings exist, where the instances are presented in a pool (one fixed set of examples, all available at the beginning of the process) or in a stream (one or several examples at a time in a sequential fashion), where the selection is made in a single step (static) or iteratively, one element at a time or by batch. Generally speaking, the system is thus composed of two main parts: a *learner* (updated with the selected labeled examples) and a *"selector"* that selects which example to label (through an oracle). A survey of active learning techniques is presented in [Settles, 2010]. We provide more details for this setting in Section 2.2.1.

Active Class Selection In this setting, the learning system can ask for an instance of a specific class. Several strategies are defined in [Lomasky et al., 2007; Wu and Parsons, 2011; Wu, Lance, and Parsons, 2013], based for example on the original distribution of the categories or the examples, accuracy improvement, current accuracy (e.g ask for class with low accuracy), or more refined strategy such as "redistricing" which aims at targeting "unstable" classes and volatile boundary. Note that these methods consider that there is no constraint on the generation of instances for any class, which may make them inapplicable on problems such as fraud detection. Moreover, they don't necessarily integrate "budget" constraint explicitly (e.g the number of examples used). Their main goal is to optimize and improve the learning by influencing the distribution of the training dataset.

Active Feature Acquisition (AFA) This setting considers the case where, during learning, some features are missing. These features can be acquired at some cost (uniform or not). The goal is thus to design a learner that conjointly learn to acquire

additional useful information for its prediction and learn to predict. The proposed methods rely for example on expectation of improvement (expected utility) as in [Melville et al., 2005] or information-theoretic criteria (e.g mutual information) in [Krishnapuram et al., 2005].

Active Information Acquisition (AIA) AIA methods consider both features and labels as information that may be acquired during training, thus leading to the design of systems able to acquire potentially both of them. The previously mentioned paper [Krishnapuram et al., 2005] integrates such specificity. Provost, Melville, and Saar-Tsechansky, 2007 also study the different types of "data acquisition" problems, in the specific case of an e-commerce application, and propose an algorithm based on a utility function designed for the "multi-type information" (labels and features) case. In [Raghavan, Madani, and Jones, 2006], the author also present a method for active learning interleaved with feature acquisition, where feature acquisition and label acquisition are done in tandem.

Integration of the budget constraint during inference:

Feature Selection / Feature Acquisition This setting considers the case where, during inference, the inputs have missing features, and most of the time, no given feature beforehand. The system can ask for the features at some cost (uniform or not) to an oracle. However, during training, the features' examples are often considered to be fully available. The goal is to learn a model able to acquire some features for the smaller possible cost and predict conjointly. We provide further details in the next section. Note that this setting can also be referred to as **active classification** (e.g in the taxonomy presented in [Settles, 2010]).

We choose in this thesis to focus on two types of costly information: features, and labels. More precisely, we propose to tackle the budgeted learning problem by learning acquisition strategies for this specific information. We provide in the next section a more detailed overview of the methods proposed for **feature acquisition**. We present the more particular case of meta active learning (i.e learning to acquire the right labels) in Section 2.2.

2.1.2 Cost-sensitive inference : feature selection and acquisition

This Section presents an overview of the models proposed to limit the cost of features used at test-time (i.e inference), as it is one of the two problems tackled in this thesis, notably in Chapter 3, 4 and 5. This problem can first be divided into two settings: uniform costs vs non-uniform costs of features. The first setting implies that each feature costs the same "price". The goal resumes to reduce the number

of features necessary to provide an accurate prediction. The second setting aims at integrating different costs between the features (e.g a medical test like blood sampling is cheaper than conducting an fMRI). Obviously, methods able to handle non-uniform costs can be straightforwardly used on tasks with uniform costs, making these methods more generic.

We also distinguish in the following two families of approaches: static methods (often referred to the problem of *feature selection*) *vs* adaptive models (*feature acquisition*).

Static methods : feature selection

A static feature selection method mainly means that the subset of features chosen for prediction is the same for all inputs, and is therefore set once and for all at the end of learning. Acquiring the features is done in a single step, and prediction is made right afterward. The problem thus becomes to find the optimal subset of features such that it maximizes the prediction quality (w.r.t budget).
Several types of static approaches exist, we refer to [Guyon and Elisseeff, 2003] for a more detailed survey on variable selection and we provide here only a brief overview of the different families of methods.

Filter models (following the classification of Kohavi and John, 1997): these methods are designed as a pre-processing step, and are independent of the choice of the predictor. For example, variable ranking (Duda, Hart, and Stork, 2001) is a filter method, which was often used as a baseline in several papers on variable selection. The principle is to compute a score for all the variables so that the highest score denotes the most valuable feature. If ones consider a ranking criterion defined for individual variables (i.e not considering subset or combination), it can be a computationally efficient method as it only requires to compute a score for each feature. It may be optimal w.r.t a given predictor in some very specific case. For instance, if the covariance matrix of the problem is diagonal, then using Fisher's criterion is optimal for Fisher's linear discriminant classifier (Duda, Hart, and Stork, 2001). Various criterion exist, such as correlation, single variable classifiers (ranking w.r.t the ability of prediction of classifiers built with a single feature), or information theoretic ranking (Bekkerman et al., 2003; Dhillon, Mallela, and Kumar, 2003; Forman, 2003; Torkkola, 2003). Other filter methods have been proposed, such as the Relief algorithm which is close to ranking as it aims at assigning a relevance weight to each feature w.r.t the "target concept", or the FOCUS algorithm (Almuallim and Dietterich, 1991), which finds the minimal subset that agrees on the features for all examples, by examining all possible subsets.

Wrappers approaches: These methods use a predictor learned on the complete inputs (i.e all the features) as a black box. In such case, the algorithm has to search for a good subset using the black box as an evaluation of the quality of the subset. The problem thus becomes how to search the space of possible subsets of features. Different strategies have been proposed, which are reviewed in [Kohavi and John, 1997], where the authors compare greedy search vs best-first algorithm. Branch-and-bound, simulated annealing and genetic algorithms have also been studied.

Embedded methods: Instead of considering the prediction algorithm as a pre-learned black box, these approaches integrate the reduction of the feature space in the learning of the prediction process, instead of "wrapping" around an inductive algorithm and trying to estimate the relevance of a selected subset of features, which can be computationally expensive when the number of features grows. This was already done for instance in [Breiman et al., 1984], where the CART algorithm for decision tree has a built-in mechanism to perform variable selection [1]. In [Bi et al., 2003], the authors use sparse linear SVMs with l_1 norm to identify the features to select, and a final non-linear SVM for prediction. They rely on the fact that both models employ the same loss function and are therefore tightly coupled. It can be related to the LASSO (least absolute shrinkage and selection operator) method (Tibshirani, 1996; Tibshirani, 1994) but designed for SVM regression. Directly incorporating a term favoring feature selection in the objective function have been proposed, mainly for SVMs. In [Weston et al., 2000], a selection parameter vector is integrated into the decision function and is optimized using gradient descent on the continuous relaxation of this parameter vector, instead of using more classical methods of search like forward or backward selection (greedy search). In [Weston et al., 2003], the authors propose a novel method to minimize the zero norm for linear models and kernel methods. Different formulations are given depending on the task/model at hand (e.g two-class linear models, non-linearly separable problem, multi-class linear models, kernel methods) and for the feature selection problem. A method to handle the selection of block of features has been proposed in [Yuan and Lin, 2005], where instead of considering the features as individuals, the authors can integrate subsets (*block*) of features in the selection process. However, the blocks have to be known and/or defined beforehand. It can be useful to integrate a priori knowledge on the data, but it is not clear if it is efficient when there is no particular assumption on the possible relevant blocks of features. Neural networks architectures have not really been studied, except in [Verikas and Bacauskiene, 2002], where the authors propose to train a neural network with a constraint on the output's derivatives. The features are selected based on the change in cross-validation error when removing the feature in the input. Note that these approaches generally don't handle specific costs per feature.

[1] Methods using decision trees for feature acquisition are detailed later in this section.

Adaptive methods: feature acquisition

One of the main limits of static methods are that they can not *adapt* depending on the current inputs, the selected subset of features will always be the same for all inputs. Having an adaptive process to choose which features to use would allow for more robust and complex decisions, as well as reducing the overall cost. Indeed, an "easy" example would only require a few features to be classified for example, while a more "difficult" one could benefit from more information (spending more budget on it) to ensure a correct prediction. Such methods have been proposed in the literature, integrating more often the variability of the cost features. We detailed some of them below. We propose to distinguish four "families" of approaches: those relying on the estimation of the information value, methods relying on cascade and decision trees, reinforcement-learning oriented approaches and attentions models.

Estimation of the information value: Some methods proposed to tackle the feature selection problem by estimating the information gain of the features in order to acquire the more informative ones. It can be seen as the sequential extension of ranking feature selection, where the ranking criterion is a particular function of information value, which can be sometimes "weighted" by the cost of feature's acquisition. For example, a specific data structure (VOILA) is proposed in [Bilgic and Getoor, 2007], which estimates the information values of subsets of features (a combinatorial problem) by exploiting the dependency between the features, resulting in a graph where each node is a subset (of varying size) of features. However, this method relies on the very strong assumption that one already knows the relationships between the features.
In [Chai et al., 2004], the authors present a strategy to integrate sequential acquisition of features to learn a test-cost sensitive naive Bayes classifier. When a new example arrives, the system computes the utility of asking for a feature as a mixture between the cost of the feature and the gain between expected misclassification with and without this new feature. They also propose a strategy for a static acquisition, namely taking the decision sequentially but "blindly", i.e without knowing the value of the feature asked for. These strategies are greedy and can become computationally expensive if the number of features is high. It is also not clear how the difference between expected misclassification with and without the feature compares with the features' costs, even if the costs are rescaled.
In [Weiss and Taskar, 2013], the authors propose to estimate a value-function of the information gain with reinforcement learning to guide the acquisition strategy. They first learn a predictor *a priori* on subsets of features. Note that they make some assumptions on the nature of the inputs which guide the choice of the different features combinations used for the sparse inputs. Then they use reinforcement learning to learn an acquisition process, based on the extraction of meta-features which guide the policy. The acquisition is stopped when the budget is met.

Another example of such approach is [Gao and Koller, 2011]. In this setting, a set of weak classifiers is learned, and an information gain is computed for each one of them depending on the currently observed information. However, in this setting, it is not the raw features that are acquired and constrained but the number of weak classifiers used.
These approaches are sequential and cost-sensitive, but do not allow acquisition per block without some assumptions or knowledge on the inputs, and will be limited when applied to larger datasets due to their greedy and/or combinatorial nature.

Cascades and Tree-based models: Decision trees are often used in feature acquisition models, as they will naturally reduce the number of features selected. In [Xu, Weinberger, and Chapelle, 2012; Xu et al., 2014b], several weak decision trees are learned with a constraint on the features used globally, and an additional constraint is used on which weak learners to use. Nan, Wang, and Saligrama, 2015 propose a new algorithm for random forest that integrates feature costs by using greedy minimax cost-weighted impurity splits to grow the trees.
In parallel, an important part of the literature relies on learning *cascade* of classifiers. One of the first approaches that proposed a cascade architecture was [Viola and Jones, 2001], where acquisition was made on high-level features of images for visual object detection. In their setting, the cascade structure allows to reject some features (sub-windows in the image) and do expensive processing only on the most useful patches of pixels. Other methods relying on weak classifiers cascade were presented afterward, as in [Trapeznikov, Saligrama, and Castañón, 2013], where the use of the cascade structure is slightly different. It is used so that each stage can either classify early (from the few features acquired up to now) or go further in the acquisition process.
In [Raykar, Krishnapuram, and Yu, 2010], groups of features are pre-assigned at each stage of the cascade, while in [Chen et al., 2012] the set of features are learned globally using additive regression method to re-weight and re-order the set of classifiers in a chain of cascades. Recently, Xu et al., 2014a presented a method to learn a tree of classifiers that allows an individual path for each input and an extension to cascade architecture.
The main limitation of cascade and decision trees based models lies in dealing with datasets with a high number of features, especially if one wants to cover a broader range of costs. However, the cascade architecture often induces the possibility of *early-stopping*, which is an interesting aspect.

Reinforcement learning (RL) based approaches: The problem of feature acquisition can be seen as a sequential decision process, and it has been studied under the Markov Decision Process (MDP) and RL framework. One of the first proposition in this regard was from Ji and Carin, 2007, who propose to solve the cost-sensitive

classification problem with a partially observable MDP alongside a myopic algorithm. Benbouzid, Busa-Fekete, and Kégl, 2012 proposed to design a controller that chooses between evaluating (a base classifier/a feature), skipping it or classifying from what have been seen, by modeling it as a Markov Decision Directed Acyclic Graph. Similarly, in [Trapeznikov and Saligrama, 2013], a sequential classifier using MDP-framework is presented, which chooses at each step to classify or to get the "next feature"[2]. Other reinforcement-learning approaches for sequential acquisition have been proposed, such as in [Dulac-Arnold et al., 2012], where a vector of the acquired features is built following a pre-defined heuristic and used to represent the current state and thus learn/follow an acquisition policy. Learning acquisition policies has also been tackled through imitation learning in [He, Daumé III, and Eisner, 2012]. However, this method requires an oracle to guide learning which is often not available in real-life tasks. Most of the reinforcement-learning based models suffer from a high computational learning cost and cannot easily scale with larger datasets when the number of features to choose from increases. It is also interesting to note that they can usually only consider classification problems.

Visual Attention Models: Models based on (visual) attention have been recently proposed for different image-related task, such as [Mnih, Heess, and Graves, 2014; Dulac-Arnold et al., 2013] for classification, [Xu et al., 2015] for text generation based on image, or for image generation [Gregor et al., 2015]. They are closely related to sequential acquisition. The principle of visual attention was presented to tackle tasks such as multiple object recognition [Ba, Mnih, and Kavukcuoglu, 2014], and was motivated by the comparison with humans behaviours in this case : when confronted with a new "scene", a human detects and distinguish the various object by moving the fovea from one specific area to another. The authors' goal was thus to mimic this process, by feeding the input (context image) to a deep recurrent neural network, which focuses at each step of the process on a particular location of the image (*glimpse*), possibly provided with better resolution, e.g in [Sermanet, Frome, and Real, 2014]. The internal state is updated w.r.t. the observed glimpse. This sequential process can be observed in Figure 2.2, which is the recurrent attention model as presented in [Ba, Mnih, and Kavukcuoglu, 2014], where each $\hat{l}_i$ is the predicted location of the glimpse to acquire. The attention strategy usually relies on adapted reinforcement learning techniques. Note that the literature has distinguish between *"soft attention"* vs *"hard attention"*. Soft attention is based on continuous weights on all the input with more or less "attention" (higher weights) on some areas. This has the advantage of keeping the model fully differentiable. Hard attention consists in sampling a specific part of the considered object.
The key idea of attention has then been extended to other applications, notably natural language processing, e.g translation task (Bahdanau, Cho, and Bengio, 2014;

[2]Here the authors doesn't focus on choosing which feature has to be acquired: it is based on the indices of the features.

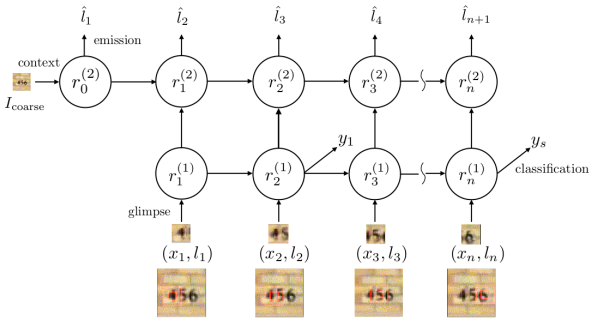


FIGURE 2.2: Deep recurrent attention model as illustrated in Ba, Mnih, and Kavukcuoglu, 2014. The system receives an image as input, and at each step of the process predicts some "patch" to acquire. This glimpse is then fed to the RNN which update its internal state with this additional refined information.

Luong et al., 2014; Luong, Pham, and Manning, 2015). Attention is also used to interact with the memory in *memory networks*, which we detail in Section 2.3.2.
The competitive performances of these methods motivate to design models that not only receives raw inputs and process it, but "interact" (e.g acquire specific information) with it. However, these models assume that there is a first observation of the input, where the attention process only helps to improve the final prediction by focusing on parts of the input. Moreover, they do not assume that there is a cost associated with additional information. Thus, the setting and assumptions quite differ from classical *cost-sensitive feature acquisition*.

Regarding these different methods, the methods presented in this thesis differ on several points. We present, to the best of our knowledge, the first framework that fully models the adaptive cost-sensitive acquisition problem with differentiable components (Chapters 4 and 5), making it learnable with (scalable) gradient descent algorithms. It is also one of the rare methods that allows batch selection of features, i.e acquiring several features at a given step, without suffering from the combinatorial problem when the number of features increases and without making assumptions on the nature of the input and the batch to acquire. Relying on a latent representation space that is learned conjointly with the two tasks, with a RNN-like architecture, seems also to be a novel aspect in the field.

2.2 Meta Active Learning

We mentioned above the problem of *active learning*, where the labels are costly. We define now the meta active learning problem, where one aims at **learning an active-learning strategy**. In this setting, during inference, the system is confronted with new problems, and it has to acquire the more useful labels for prediction. Here,

each data point is a complete prediction problem, with training and testing examples, and possibly labels or some oracle that provides labels, which has a cost. This setting is inspired by **one-shot learning** methods (e.g in [Santoro et al., 2016]). To the best of our knowledge, only one recent paper focuses on this problem (Woodward and Finn, 2017).
Note that transfer active learning has also been studied (Wang, Huang, and Schneider, 2014; Zhao et al., 2013; Yan et al., 2012). However, this field studies how active learning techniques can help *transfer learning*. In this case, the system has a cheap, often with a lot of examples, *source* dataset, and an expensive, smaller, *target* dataset. The goal is to use efficiently the *source* dataset to solve the *target* problem. Thus, active learning methods can help to better select a few labels on the *target* domain.

We present in this section an overview of the different axes of research that are closely related to **meta active learning**. First, we provide more details on the classical **active learning** setting, with various methods proposed in the field. We then present a brief survey of **meta learning** key ideas and different approaches. At last, we focus on the problem of **one-shot learning**, which has known recently a surge of interest and has motivated us to propose a meta-learning protocol for the active learning problem. A detailed and formal definition of the meta active learning protocol is provided in Chapter 6, where we propose different leads to tackle this problem.

2.2.1 Active learning

Active-learning techniques aim at constraining the amount of labeled examples used during training, by designing acquisition strategies that provide good final prediction performance. We propose to distinguish in our review two families of approaches in active learning, based on the nature of this acquisition process. We separate static methods, where the acquisition is made in a single shot, with sequential methods, with several steps of acquisition.

Static active learning These methods select the subset of examples to label (typically from a pool of unlabeled instances) in a single step, without considering the feedback of the oracle on the labels of each chosen example. In such setting, it is therefore not possible to rely on an estimation of the oracle, and the proposed methods will solely be based on what can be extracted from the dataset of unsupervised examples (e.g distribution of data). For example, in [Gu et al., 2012], the authors approach the problem as *selective labeling*, with a fixed budget, where they consider the out-of-sample error for a Laplacian Regularized Least Squares as a semi-supervised learner. Yu, Bi, and Tresp, 2006 present an approach for regression based on *transductive experimental design*: they aim at selecting examples that are hard to predict but also representative with regard to the test data (supposed to be

given beforehand). *Off-line* single batch active learning has also been proposed for specific graphs-based tasks, e.g in [Guillory and Bilmes, 2009], where the authors aim at selecting sets of vertices to label.

Sequential active learning In this setting, the models can send some instances to label to the oracle and observe the answer(s) of the oracle before asking maybe for more labels. We distinguish the case where there is a finite, observable, **pool of instances** *vs* the **"stream" setting**, where the data arrives through time, one example after the other. In [Tong and Koller, 2001], the authors proposed to tackle the pool based sequential single-instance-mode active learning problem with SVM, where label selection is driven by the resulting reduction of the size of the version space. Other "single-instance-mode" methods have been proposed, e.g in [Zhang and Oles, 2000] who demonstrate the utility of the Fisher information matrices for such problems. In [Collet and Pietquin, 2014], the authors present how to use a multi-armed bandit setting for active learning in binary classification. Guo and Schuurmans, 2008 propose an approach for batch-mode active learning, where a subset of examples is selected at each step of labeling. The authors define the performance of a model as high likelihood on the labeled examples and low uncertainty on the unlabeled ones and propose an approximation for the NP-hard optimization problem. Previously, Hoi et al., 2006 presented a method where the informativeness of a subset of examples is computed w.r.t the Fisher information matrix, using a greedy algorithm to overcome the combinatorial aspect of the problem. The same authors propose in [Hoi et al., 2009] an approach relying on SVM, where they first learn a kernel using labeled and unlabeled data [3], and then use this kernel for batch active learning.
In the case of stream of instances, the sequentiality is obviously required, as a decision has to be made at each step when a new example arrives. Freund et al., 1997 present a selective sampling approach based on a query by committee algorithm in a bayesian model of concept learning.

2.2.2 Meta learning

While learning to predict is the main goal of machine learning, meta-learning is a domain of research that focuses more on how to *learn to learn*. Here, the goal is not only to correctly classify on a single task but to use this knowledge across different tasks (with or without similar target domains). As we aim in part of this thesis at extending active learning to new unseen tasks (by learning the active label acquisition process), this typically falls under the meta-learning paradigm. We present here a brief overview and taxonomy of meta-learning and active-learning based partially on the surveys of Vilalta and Drissi, 2002 and Pan and Yang, 2010.

[3] Note that generally it is assumed that some labeled data are given beforehand and that the pool of unlabeled data on which to select is big

In [Vilalta and Drissi, 2002], meta-learning is defined according to region of tasks and learning algorithms. Each learning algorithm gives a specific performance on each task, and some perform well on a subspace of tasks and bad on other. The authors describe one goal of meta-learning as learning what causes a learning algorithm L to dominate in some region tasks R_L. From there, the target is to design techniques that adapt or select the best learning strategy for the tasks. Usually, the approaches distinguish two parts of a meta-learning system: a base-learner, which outputs predictions on the current task, and the meta-learner, which aims at using the knowledge of the previous problems to modify or guide the base-learner.
The survey distinguish different types of techniques for this problem. They first give an overview of the methods focused on the **meta-learning of base learners**, for example using *stacked generalization* (Wolpert, 1992). Another type of approaches is the **dynamic selection of bias**. It is based on moving the region of expertise of the system along the tasks, thus relying on some structures along the tasks. Utgoff, 1986 propose to do so by changing the representation of the feature space. In [Rendell, Seshu, and Tcheng, 1987], the authors propose to use algorithm-selection, while in [Rendell, Sheshu, and Tcheng, 1987] the meta-learner learns relationships between the tasks characteristics (e.g number of features) and biases embedded in the learning algorithm. This is closer to **meta-rules matching domains with algorithm performance**, where the idea is to define a set of domains related meta-features that are linked to the performance of a learning algorithm as in [Aha, 1992]. In parallel, methods based on **inductive transfer** (Pratt and Thrun, 1997) have been presented to transfer the knowledge gathered from one task to another, in order to improve the learning through time. Thrun and Pratt, 1998 illustrate how learning from multiple tasks improve generalization while Pratt and Jennings, 1996 study how learning on related tasks can benefit neural networks in the *learning to learn* scheme. Schmidhuber, Zhao, and Wiering, 1996 propose a meta-reinforcement learning strategy, that considers learning algorithms as possible actions of the RL system. Schmidhuber, 2004 also presents a hybrid approach, that mixes inductive transfer and dynamic selection.

In parallel, different methods have been proposed to **learn the optimization algorithm** (or the update rule). In [Schmidhuber, 1995] a reinforcement-learning system is designed "self-improves". The system improves the way it learns but also improves the way it improves the way it learns. More recently, Andrychowicz et al., 2016 propose to follow the idea in [Hochreiter, Younger, and Conwell, 2001] that use a RNN as a base-learner, with an approach that scale to larger neural networks optimization problems. They propose to rewrite the classical update rule for differentiable functions ($\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t)$) with a function that predicts the updating value (here the $\alpha_t \nabla f(\theta_t)$). This function is defined by its own parameters Ψ and takes as input the gradient $\nabla(f\theta_t)$. Thus, the updating function of the parameters

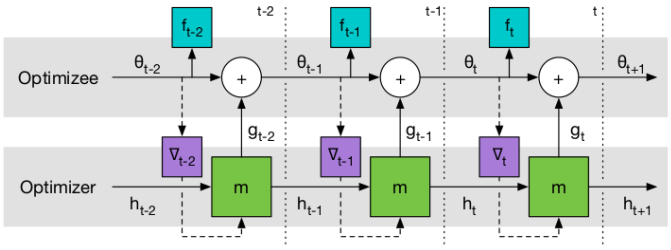


FIGURE 2.3: Architecture of the meta-learning architecture proposed by Andrychowicz et al., 2016, with a joint RNN. The upper part (*optimizee*) outputs at each steps t a prediction f_t and associated gradient ∇_t. This gradient is taken as input by the lower network, *optimizer*, which then outputs g_t, the predicted update of the optimizee weights.

is rewritten as:

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \Psi)$$

They model the update rule g with a RNN, as illustrated in Figure 2.3 from their work. They show that their network manages to learn some optimization strategy that converges faster than classical gradient techniques and obtains better performances[4]. Similarly, Li and Malik, 2016 present a reinforcement learning approach to the problem, where they redefine the generic optimization algorithm as a policy (sequence) of updates. Thus, an action is a step vector used to update the parameters, partially depending on the gradient. Munkhdalai and Yu, 2017 propose to design an architecture that combines both "fast weights" (similar to Andrychowicz et al.) and "slow weights" (more classical updates), and present several experiments including one-shot learning tasks. , as it uses the gradients and loss of a network as the input of another network (LSTM) to predict the new set of weights θ_t.

2.2.3 One-shot learning

We present in this section an overview of methods proposed for the problem of *one-shot learning*, first described in [Yip and Sussman, 1997]. The goal is to design algorithms capable of predicting correctly using very few examples for each class (typically between one and five). It is inspired by the ability of humans, especially young children, to learn rapidly new concepts and generalize from few examples of a category (studied in cognitive science e.g in [Berko, 1958)]. Usually, the approaches assume that data of similar nature is available beforehand, which will guide the learning, in an unsupervised or semi-supervised fashion. We describe in the following section methods based on various methodologies, which can differ in the assumption w.r.t. the available data and the protocol used during training.

[4]Note that the paper by [Ravi and Larochelle, 2017] mentioned in the next section on one-shot learning also relies on this idea. It has been published on arxiv shortly after.

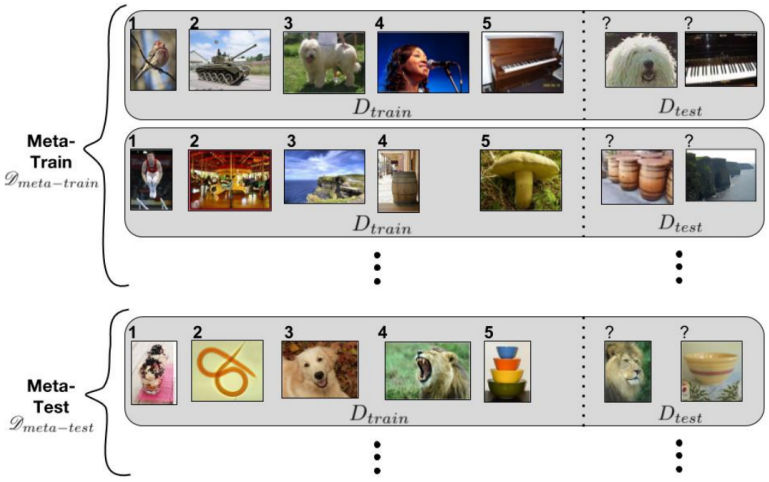


FIGURE 2.4: Example of the one-shot "meta-learning" protocol, illustration from Ravi and Larochelle, 2017. The **Meta-train** $\mathcal{D}_{meta-train}$ is the training set of the overall system : each line is a data-point, composed of 5 training examples, one for each class (left of the doted line), for example the categories are "bird, tank, dog, singer and piano" for the first problem. Left of the doted line are the testing examples of the problem. There are several problems in the meta-train dataset. The meta-test dataset is built similarly, on categories unseen in the meta-train.

Generative methods One type of approaches proposed for this problem relies on learning a *generative model* to better understand and represent the data at hand, e.g in [Rezende et al., 2016; Lake, Salakhutdinov, and Tenenbaum, 2013; Lake et al., 2014]. For example, in [Lake et al., 2011], the authors learn a generative model of the strokes composing a character, learned in an unsupervised fashion on various alphabets[5]. Prediction is based on the computation of the probability that two characters are explained by the same sequence of strokes.
The work of Edwards and Storkey, 2016 is also of interest while taking a different approach on the problem. They propose to design a model able to learn representations on a whole dataset, in an unsupervised fashion. The underlying idea is that these representations should encapsulate some statistics linked to the observed problem.

Bayesian framework In [Fe-Fei, 2003; Fei-Fei, Fergus, and Perona, 2006; Salakhutdinov, Tenenbaum, and Torralba, 2012], the authors use (variational) Bayesian framework to integrate the information from the previously seen and learned class for future prediction. The "prior knowledge" is expressed as a probability density function on the parameters of these models. The prior can then be updated w.r.t one (or few) new observations for a new category. In these approaches, it is assumed that one has access to categories with a lot of examples.

[5]The model is evaluated -in classification- on unseen alphabets.

Representation learning based methods Fink, 2004 presents the first approach for one-shot learning based on metric learning (Bellet, Habrard, and Sebban, 2013; Kulis, 2013). He proposes to learn a representation function so that distance between instances of the same class is smaller than the distance between different classes' examples. He learns this metric on examples of similar nature but different categories. The representations are then used with a nearest neighbours algorithm to predict. In [Koch, 2015], the authors propose to design a Siamese (Convolutional) Network that will learn to discriminate between pairs of examples (i.e decides if two images are similar or different), for which there is a lot of supervised data. Then, this network is used on new categories with few examples during inference, by simply measuring the similarity between labeled instances and examples to classify, and predicting the most similar class.

Learning to predict from few examples Inspired by Hochreiter, Younger, and Conwell, 2001, several recent methods rely on a specific learning scheme which mimics the final task at hand, in some "meta-learning" fashion. Instead of solely using instances of other categories to learn a representation model as before, they define a "training point" as a one-shot classification problem. This is illustrated in Figure 2.4. They then design specific models able to learn on such data. For example, Santoro et al., 2016 propose to use the recent memory-augmented neural network, to integrate and store the new examples. Similarly, Vinyals et al., 2016 propose to rely on external memories for neural networks, bidirectional LSTM and attention LSTM. One key aspect of their approach is their aim at representing an instance w.r.t. the current memory (i.e observed labeled examples). Note that these approaches design a "one-shot learning problem" (e.g training point/inference point) as a sequential problem, where one instance arrives after the other. Additionally, the system can receive some afterward feedback on the observed instances. For example in Santoro et al., 2016, the system receives the true label of an instance at the next step of the sequence.

Finally, one can also cite the work of [Bertinetto et al., 2016], where the authors aim at achieving dynamic parameters predictions to tackle the one-shot learning problem. They propose a method that learns the parameters of a deep model (generally not suited for one-shot task) in a single shot, by designing a "learnet", a deep network which predicts the parameters of a "pupil" network used for prediction.

2.2.4 Meta Active Learning

The problem that we propose to approach in the Chapter 6 of this manuscript is at the junction of active learning, meta-learning and one-shot learning. We observe

that all the approach proposed for one-shot learning consider that the labeled examples are picked randomly. Moreover, they do not necessarily integrate the unsupervised information of the future examples to classify. This leads us to the idea that integrating active learning properties in such system should improve the final performance. Moreover, it would present a novel way of tackling active learning, as instead of designing strategies beforehand, the active policy could be learned on training problems, in a similar fashion as presented before.
To the best of our knowledge, only one recent paper studies this problem. Woodward and Finn, 2017 propose to use reinforcement learning in their system so that it is able to decide which examples are "worth to label". They still consider each problem as a sequence of inputs. However, they transform slightly the original problem: instead of having a first part of the sequence composed of labeled examples, and then of unlabeled instances on which to predict, they consider all elements of the sequence similarly. For each input, the system can either classify or ask for a label (thus falling in the *sequential active learning* scheme). They do so by extending the model of Santoro et al., 2016, but where the true label of the previous instance is withheld unless the system asks for it. The system gets a high reward for accurate prediction and is penalized when acquiring label or giving false prediction[6]. They design the action-value function to learn as a LSTM.
The main limitation we observe in this method is that it suffers from the same drawbacks as one-shot methods, mainly as it does not consider the whole dataset at hand. This is one major aspect that we aim at studying in the last part of this thesis.

2.3 Recurrent Neural Networks Architectures

Several methods that are presented in this manuscript share a common idea with the *(visual) attention models* presented in Section 2.1.2 : the key idea is to design systems that acquire information on an input (a data-point) sequentially, in several steps. We chose to use recurrent architectures as well to build such systems in Chapters4, 5 and 6. We present in this section a description of various recurrent neural networks architectures designed for particular problems and inputs.
Neural networks systems for machine learning have known several evolutions since the perceptron in the late 50's (Rosenblatt, 1958). Notably, the integration of non-linear functions, the multiplication of layers and the development of the back-propagation algorithm (Werbos, 1974; Rumelhart, Hinton, and Williams, 1985), have helped to obtain better and better results with these systems. The design of particular functions for image processing (e.g convolutional layers [Fukushima and Miyake, 1982; LeCun et al., 1998]) and more recently the general growth of deep learning (Salakhutdinov, Mnih, and Hinton, 2007) have known in the last decade a pick of

[6] At each step the model can either classify or ask for a label but not both.

popularization, notably due to the good performance achieved on various challenging tasks from image classification, object recognition, to game playing and natural language processing.

Recurrent neural network (RNN) is a specific type of neural networks, commonly used to handle sequential inputs or sequential problem such as speech recognition (Sak, Senior, and Beaufays, 2014), handwriting recognition (Graves et al., 2009), or language translation (Sutskever, Vinyals, and Le, 2014). The main particularity of RNN is that they contain cyclic connections, based on recurrent weights (duplicated weights, see Figure 2.5a). This allows these architectures to build an internal state with **memory** ability, through a *feedback loop*. These systems are intrinsically dynamical, and their flexibility allows a variety of application setting. We present in this thesis several models that rely on these type of architectures to design sequential, adaptive and interactive systems for our budgeted tasks. This section thus proposes a non-exhaustive technical review on recurrent neural networks. We first define the various settings RNN can be adapted to. We then provide details on the more specific architectures Long-Short Term Memory (LSTM), memory network, bi-directional RNN, and attention RNN.

2.3.1 Overview of the various settings

As we mentioned, a recurrent network is based on cyclic connections on their internal state. The very basic architecture can be illustrated as in the Figure 2.5a, where an input x is integrated to the current hidden state s with weights U. This state is "updated" sequentially w.r.t. its previous value with an update function of weights W. Prediction is made based on the current state through function of weights V. If both x and y are non-sequential, this resume to a vanilla feed-forward network, as the "feedback loop" W is not used. Generally speaking, we can thus compute the update state s_t at step t as follow:

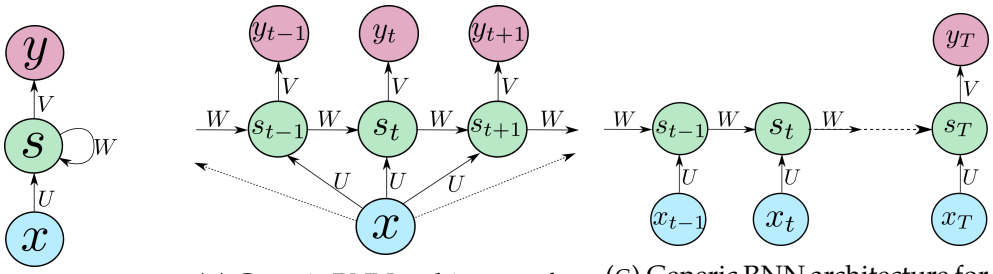
$$s_t = g_K(f_W((s_{t-1}), f_U(x_t))$$

And the expected output y_t as:

$$y_t = f_V(s_t)$$

However, the flexibility of the RNN allows various "shapes" for both inputs x and outputs y, which we detail now.

Static input to sequential output This setting corresponds to the case where the input is non-sequential, and the output sequential, for example image caption generation, where from an image one aims at predicting a sentence describing the picture. One corresponding architecture is illustrated in Figure 2.5b. In this case, the



(A) Basic recurrent architecture.

(B) Generic RNN architecture for a single input and sequential output.

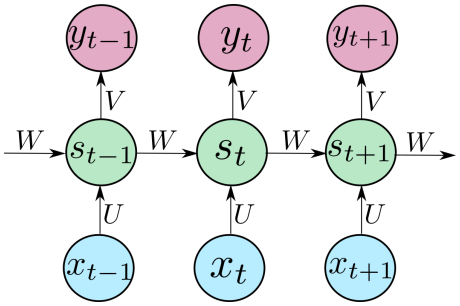
(C) Generic RNN architecture for sequential input and a single output.

FIGURE 2.5: Schematic RNN architectures for different input/output settings.

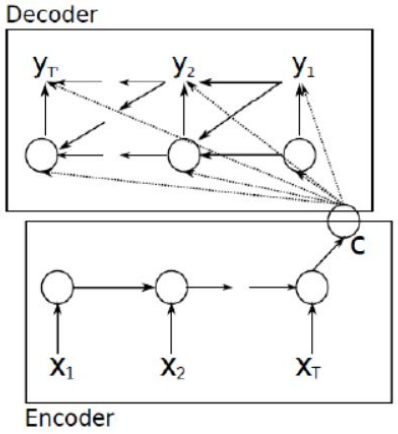
input x (e.g image) is fed to the RNN at each step of the process, and each y_t corresponds to a word, leading to a full sentence at the end of the process. Note that the input can be fed only at the first step and that one can consider re-integrating the predicted y_t in the next hidden state s_{t+1} (as in Figure 2.7b).

Sequential input to static input This setting corresponds to the case where the input is sequential, and the output non-sequential. Sentiment prediction is one application of such settings, as the goal is to predict for instance if a sentence is positive or negative. As shown in Figure 2.5c, the state s_t is updated at each step of the process w.r.t. the t-th element of the input sequence (e.g a word) and its previous state s_{t-1}, however prediction is made only at the end of the input. Thus, supervision is back-propagated through all states s_t from the last state s_T (limitations related to this -e.g vanishing gradient- are discussed later on this section).

Sequential input and sequential output This setting corresponds to the case where both inputs are sequential. Here, we distinguish two cases: (i) both sequences have the same length, for example video labeling at the frame level, where at each step (one frame of the video), a label is expected. (ii) The sequences are of different corresponds to translation tasks, as translation doesn't necessarily match one word to another, but a whole sentence. Figure 2.6a shows an architecture for the first case, where for each input x_t, the internal state is updated and predicts an output y_t. Figure 2.7b illustrates the second case, where the input sequence has a size T, and output sequence has a size T', and where Cho et al., 2014a propose two RNN as an encoder and a decoder to tackle the translation problem. Both RNN are "linked" through a common layer c.



(A) Generic architecture of a RNN for sequential inputs and outputs of same length.



(B) Sequential inputs and sequential outputs of different size T and T', architecture and illustration from [Cho et al., 2014a]. A first RNN takes the original sequence (here sentence) as input, which leads to an internal state c. This state is forwarded as input to the "decoder" RNN that outputs a sequence of size T'.

FIGURE 2.6: Two possible RNN architectures for sequential inputs and outputs.

2.3.2 Different specific RNN architectures

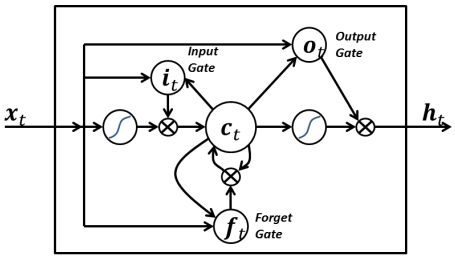
These various settings can thus be implemented in a "vanilla" fashion, with single layer hidden state and non-linear function such as hyperbolic tangent, but more complex layers can be integrated, such as convolutional ones. However, as it is, RNN architectures suffer from one major drawback with the problem of vanishing gradient. Indeed, for long sequences, and more particularly if the supervision is not done at each step of the sequence, the back-propagated gradient will become smaller and smaller going back through the time steps. As a consequence, the networks have a limited memory through time. We present in this section several types of architectures that try to overcome these limitations, first with specific "gated" cells, then with specific approaches that integrate an explicit memory in the network. Afterward, we present two techniques that aim at overcoming the "myopic" hard sequential aspect of RNN, as it considers inputs as ordered sequences and usually can not integrate "future" knowledge as it is considered "unseen".

Long-Short Term Memory and Gated Recurrent Unit

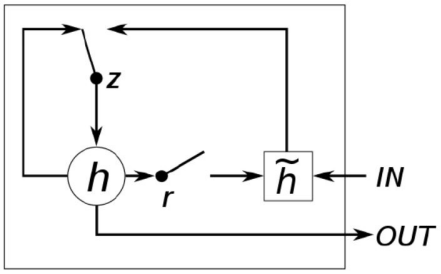
Long-Short Term Memory (LSTM) [7] is an architecture for RNN that has been proposed in [Hochreiter and Schmidhuber, 1997]. It has known in the recent years various success on several tasks on sequential data, and various variations have

[7] LSTM can designate both the cell or the RNN composed of the cells

been proposed (see [Greff et al., 2016] for an overview). The key idea that motivated the LSTM design is to have one part of the cell which can keep its state over time (memory cell), and one part that update/forget depending on the incoming information (gating unit). Figure 2.7a illustrates this specific cell. It contains three (or four) gates which control how the information "flows" in or out of the memory. The "forget" gate typically allows to "start afresh", by forgetting the previously seen inputs. The "input" gate allows integrating more or less information depending on what is observed. The Gated Recurrent Unit (Cho et al., 2014a; Cho et al., 2014b) is close to LSTM, as it also relies on gates that enable "forgetting", and on additive functions for the internal state. The use of additive function implies that the internal state is not "squashed" at each step and thus allows the gradient to flow through the steps when back-propagating. This, therefore, prevents from the vanishing gradient problem mentioned earlier.



(A) LSTM cells as illustrated in Greff et al., 2016. The input is forwarded to three "gates", and the internal state c. The $\otimes$ can be considered hadamard product. The input gate, considering c_{t-1} and x_t, "weights" how much of the input information will influence the hidden state c_t. The forget gate, also taking c_{t-1} into account, outputs a vector of the size of c which controls the "resetting" the hidden state. The output gate controls how much the internal state weights on the final prediction. Each gate acts as a "mask", respectively on the input (input gate) and the hidden state (forget and output gate).



(B) Gated Recurrent Unit illustration from Cho et al., 2014b; Cho et al., 2014a. Similarly to LSTM, there is here two gates. A reset gate r, controlled by the current input and the previous hidden state h. An update gate z, controlled by the input and the hidden state h. The actual update is done through a weighting between h_{t-1} and $\tilde{h}_t$.

FIGURE 2.7: Diagrams of two "gated" cells for RNN: LSTM and Gated Recurrent Unit.

Memory Networks

Several extensions of RNN have been proposed to integrate an **explicit** memory. It is based on the similar idea driving the LSTM, which is to provide a way to remember or forget what has been seen in the sequence. However, in the case of LSTM, the "memory" is contained and represented in the hidden state of the cell, thus it is not explicit and can lack power, as it's a "small" memory. As mentioned in [Weston, Chopra, and Bordes, 2014], these networks, unfortunately, don't benefit from one

of the major aspects of a modern day computer: the memory. The authors propose to this hand a class of models that integrates a memory component which can be read and written to. They define four additional components: (i) an input feature map which transform the raw input into latent feature representation, (ii) a "generalization" component, which updates the memory w.r.t. the current input, (iii) the "output feature map" which outputs a representation vector given the input and current memory, (iv) a module that transforms the output representation vector into the expected format. The authors present a specific implementation of this framework and experiments on textual and language tasks. Very close to this model, and published on arxiv shortly before this one, [Graves, Wayne, and Danihelka, 2014] presents Neural Turing Machine, a fully differentiable architecture that integrates a similar process of reading and writing in a "memory bank" (illustrated in Figure 2.8). They motivate their work on both psychology, neuroscience and cognitive science and the concept of "working memory" (Miller, 1956). Writing and reading is done through a set of (normalized) weights, which controls the degree of importance (to read or write into) of each part of the memory (thus giving "blurry" read and write operations, as mentioned by the authors, in order to keep the architecture differentiable and thus trainable with gradient descent). The authors distinguish two types of **addressing mechanisms** (which can be considered as "attention" mechanisms, as the read or write heads "focus" on some part of the memory): (i) content-based addressing, where the attention is based on similarity measures between elements of the memory and the current value to store, (ii) location-based addressing, using a rotational shift of the attention weights. This addressing strategy is particularly useful to handle problems such as arithmetic problem solving[8].

It is interesting to note that these models are closely related to attention models, to drive how the memory is addressed. However, some recent methods propose to use **active memory** (Kaiser and Sutskever, 2015), to replace attention techniques. In this case, the model can access and change all its memory. A more extensive comparison between attention and active memory is provided in [Kaiser and Bengio, 2016].

Bidirectional RNN

Bi-directional RNN have first been proposed by Schuster and Paliwal, 1997. It is motivated by the observation that some prediction at a given step could benefit from future observations. While the prediction can be delayed for a few steps, it can only be for a fixed number of steps. Moreover, several applications work with sequences that are *fully observable*, thus it is not necessary to constraint a prediction on only part of the available information. Yet, the RNNs have more powerful ability than

[8]The authors note that content-based is more general than location-based as some information regarding location could be included in the content of a memory cell.

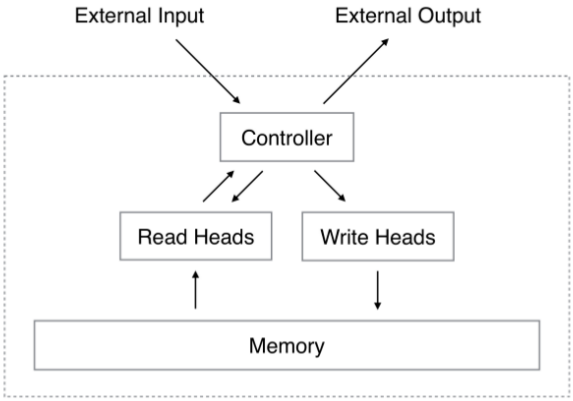


FIGURE 2.8: Neural Turing Machine Architecture, from Graves, Wayne, and Danihelka, 2014, integrating a memory "bank" and a controller to read and write on this memory.

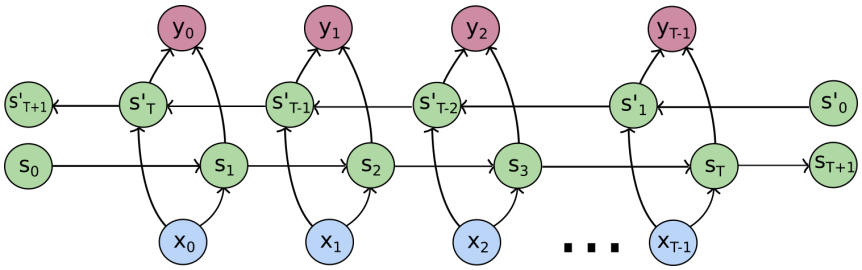


FIGURE 2.9: Bi-directional RNN illustration. Each prediction y_t is made w.r.t. both internal states s_{t+1} and s'_{T-t}, where s_{t+1} (resp. s'_{T-t}) which has "seen" examples x_0 to x_t (resp. examples x_{T-1} to x_t. Thus, the whole sequence needs to be processed before the system is able to compute any prediction.

classical feed-forward network to handle sequences. The authors thus proposed an adapted RNN architecture where each predicted output depends on two internal states. These two internal states are respectively computed in the "positive" time direction and the "negative" time direction (i.e from T to 0). This "two-way" process is illustrated in Figure 2.9. With similar notations, we can write the computation of each prediction y_t as the combination of s_t, s'_t, where s_t is computed w.r.t. inputs $x_0, \dots, x_t$ and previous states $s_0, \dots, s_{t-1}$, while s'_t is computed w.r.t $x_t, \dots, x_T$ and previous states $s'_0, \dots, s'_{T-t}$ (with T the size of the sequence).

Order-invariant RNN

While bi-directional RNN architectures (and its variations) allows making predictions by considering the complete sequences, it is still dependent on the order of the sequences. While it is one of its advantages when dealing with actual ordered data such as sentences or video, it becomes a limitation when applying these types of methods on sets. For example for the task of sorting a set of numbers, or in our setting in Chapter 6, where we consider a complete dataset as input, the system should

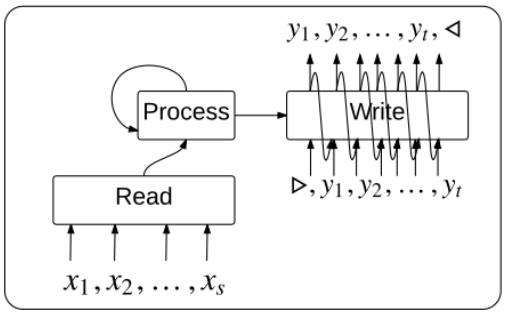


FIGURE 2.10: Read-Process and Write model illustration presented by Vinyals, Bengio, and Kudlur, 2015 to provide an order-invariant (w.r.t inputs) system.

be resilient w.r.t the order of the input sequence. Vinyals, Bengio, and Kudlur, 2015 study how the order influences the results of "sequence to sequence" models, both in inputs and outputs. To this matter, the authors propose two strategies to provide order-invariant method, one with respect to the inputs, one w.r.t. the outputs. To handle inputs sets, they design a recurrent system integrating associative memories with a content based attention. The model has three components: (i) a reading block which transforms each input x_i into a memory vector m_i, (ii) a process block, which is a LSTM working solely on the memories vectors m_i, and does not take inputs or outputs. It performs T steps of computations on the memory to build its internal state. (iii) A write block, a LSTM that takes the final state of the process block. In their case, they propose to use a LSTM pointer network (Vinyals, Fortunato, and Jaitly, 2015) that outputs a pointer to an element of the memory m_i, as their task is to re-order a sequence of inputs, however it seems possible to use any RNN architecture. The overall system is depicted in Figure 2.10. Note that this method can be seen as a special case of Neural Turing Machine (Graves, Wayne, and Danihelka, 2014) or Memory Network (Weston, Chopra, and Bordes, 2014).

2.4 Closing remarks

We provided in this chapter an overview of the different fields of research this thesis is linked to. The first section was dedicated to the research in **budgeted learning**. It is composed by a global survey of the various costs existing in machine learning, and in a second part by a specific focus on methods for **feature acquisition**, which is the main problem studied in this thesis (Chapters 3, 4 and 5). We distinguished two main types of approaches: static methods, which select a unique subset of features to acquire, the same for all inputs, *vs* adaptive methods, which sequentially gather features in several steps. The second section presented an overview of **active learning** and **meta-learning** works, which are two research domains tied to our **meta active learning** goal, studied in Chapter 6. Finally, the third section provided

a more technical review related to recurrent neural architectures, as this family of networks is a key component of several of our models.

Chapter 3

Feature Selection - Cold-start application in Recommender Systems

Abstract: We present in this chapter a static feature selection model and its specific application for *cold-start* in recommender systems. We first provide a reminder of the problem at hand, as well as a related work section on recommender systems and on the cold-start problem. The method we propose learns at the same time an interview set (which questions to ask a new user, i.e which features to select) and a set of representations used for prediction. We propose to rely on a particular representation learning scheme compared to what is usually done in recommender systems, and we integrate specific weights and regularization that control the interview learning. This chapter is partially based on the work presented in [Contardo, Denoyer, and Artieres, 2015].

3.1 Introduction

Representation learning has become of crucial importance to provide robust and efficient methods in machine learning. Most of these approaches tend to assume that the processed data is fully observed. However, there are many applications where the system has access only to partial information. For example, applications that relies on streams of inputs, where a decision should be made at each step without knowing the future incoming information (e.g on-line video tagging). Some applications also need models able to infer on only a partial view of the input, i.e a subset of features. Typically if the features are costly, it could be useful to reduce the amount used for prediction. One typical example of such case is the medical diagnosis, where all features (results of medical tests) can not be known as it would take too much time and money, and would not be necessary useful. In such setting, one would ideally want to choose which information to acquire on the inputs in order to improve the final decision. This is the problem we study in this chapter.

More particularly, we propose to focus on recommender systems and on the *cold-start* recommendation problem. Recommender systems have become an active field of research and are now used in an increasing variety of applications, such as e-commerce, social networks or participative platforms. They aim at suggesting the most relevant items (e.g products) to each user, in order for example to improve their experience (e.g Netflix). To recommend relevant items, recommender systems can rely on different types of data, such as users' explicit and/or implicit feedbacks (e.g rating a movie on a scale of stars, buying an item, or listening to a song), or informative features about users (age, postcode) or items (type of movie, actors). Users may all provide different information (rate different items and not rate all of them). Thus, the recommendation problem is typically relying on partial (incomplete) information (e.g ratings) of the inputs (e.g users). This is generally handled through the use of specific representation learning algorithms.

However, when a *new user* arrives in the system, there is no information about him or her provided, implicit nor explicit feedback. This is called the *cold-start* problem. A majority of the proposed methods can not make any personalized prediction in this case unless using pre-determined heuristics. It is considered to be a crucial problem in recommender systems, as one seeks to please the user as quickly as possible to ensure that he or she stays in the system. Therefore, waiting for information about the user to arrive through time is not a viable option. In such a setting, one needs to gather the information directly, for example by asking questions to the user about his or her tastes regarding some items. This is what we propose to study in this chapter. We present a model that can build relevant representations from a few information (e.g right after the interview, with only a few answers) to more "stable" settings (e.g *warm* context). The model also learns at the same time a subset of items on which to conduct the initial interview. The proposed method provides a *static* feature selection, as the subset will be the same for all inputs (in our case users). This is motivated by a study (Golbandi, Koren, and Lempel, 2011) illustrating that users prefer to have a finite and unique set of questions, appearing on one webpage, instead of having the questions showing one after another (thus allowing adaptiveness in the interview process), even if the total number of questions is the same.

We first provide an overview of related works on recommender systems and the cold-start problem in Section 3.2. Then we define more formally the cold-start problem and describe our model in Section 3.3, as well as how the approach can be used in different settings of the recommendation process. Experimental results are shown in Section 3.4.

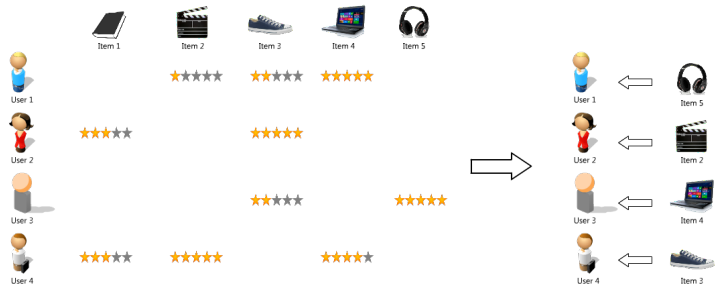


FIGURE 3.1: Collaborative filtering process: from a matrix of ratings, where each row is the ratings of a particular user, the goal is to recommend relevant items, using knowledge of the ratings given by all users on different items.

3.2 Related work on recommender systems and cold start

The recommendation problem has been intensively studied in the recent years, more particularly since the Netflix challenge in 2009 (Bennett and Lanning, 2007). The overall goal is to suggest the most relevant items (movies, videos, advertisements, products) so that the user will watch, listen, click or buy the item. The setting can, therefore, vary a lot between one application to another. Indeed, some systems, such as *imdb* or *amazon*, can access *ratings* of users on some items (e.g in Figure 3.1), while others will have more implicit data like sequence of songs listened to. We propose to focus our approach on cases where explicit feedback (e.g ratings) is obtained on items.

In such setting, the recommendation problem is often seen as accurately predicting the ratings given by each user on the items. Then, the recommendation is done by suggesting the item with the highest predicted score. Thus, distance prediction error (RMSE) is generally used in this case. However this topic has been widely discussed, and several other performance measures can be used, such as top-K.

3.2.1 Collaborative Filtering

One of the main family of approaches are *Collaborative Filtering* methods, which rely only on the interactions between users and items, such as ratings or purchase history (see [Shi, Larson, and Hanjalic, 2014; Koren and Bell, 2015] for recent and extensive surveys). Other families of approaches exist, such as *Content-Based* methods, which use informative features on users and items (Pazzani and Billsus, 2007), and hybrid methods that mix ratings and informative features (Basilico and Hofmann, 2004). We do not give details here as these approaches rely on different information as ours, and are thus not easily comparable.
Collaborative filtering techniques can be distinguished into two categories. *Memory-based* methods, such as Neighbour-based collaborative filtering Resnick et al., 1994,

compute *weights* between pairs of items (Sarwar et al., 2001) or users (Herlocker et al., 1999), based on similarities or correlations between them. To overcome the scaling limitation of these methods (which need to compute all pairwise similarities), Koren, 2010 proposes to factor the neighborhood model and learn these neighborhood relations through the minimization of a global cost function. *Model-based* methods, such as *Latent Factor Models*, have rather a *representation learning* approach, where representations vectors for each user and item are inferred from the matrix of ratings using matrix factorization techniques (Koren, Bell, and Volinsky, 2009). Let us define the rating matrix $\mathcal{R} \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$, $\mathcal{U}$ being the set of users, $\mathcal{I}$ the set of items. The notation $|\cdot|$ is the cardinality of a set. Note that this matrix $\mathcal{R}$ is incomplete, as all ratings (couple user and item) are not observed. Matrix factorization based methods aim at learning a decomposition based on two sub-matrices, such that the rating matrix can be approximated as $\mathcal{R} \approx PQ$ (see Figure 3.2), $Q \in \mathbb{R}^{N\times|\mathcal{I}|}$ and $P \in \mathbb{R}^{|\mathcal{U}|\times N}$, where N is the size of the latent space. Each of these sub-matrices can be interpreted as latent profiles of respectively the users ($p_u \in \mathbb{R}^N$, i.e the u-th line of matrix P, is the representation of user u, $u \in \mathcal{U}$) and the items ($q_i \in \mathbb{R}^N$ resp. the i-th column of matrix Q, the representation of item i, $i \in \mathcal{I}$), and the product between the representations of user u and item i should predict the expected rating the user would give to the item ($r_{u,i} \simeq q_i p_u$). Intuitively, a feature l of the item representation ($q_{i,l}$) could be seen as "how much this item has a particular characteristic" (e.g a movie is funny / a comedy), while the feature $p_{u,l}$ would respectively represent how much the user u appreciates such feature in an item.

Following the previous notations, and denoting $\mathcal{O}$ the set of actually observed pairs (u, i) such that a rating on item i has been made by user u (as $\mathcal{R}$ is incomplete), the representations $p_u, q_i, \forall u \in \mathcal{U}, \forall i \in \mathcal{I}$ (hence the matrices P and Q) are usually learned by minimizing an objective loss function $\mathcal{L}(\mathbf{p}, \mathbf{q})$ which measures the difference between observed ratings $r_{u,i}$ and predicted ones. The loss is usually defined as a L_2 objective:

$$\mathcal{L}(\mathbf{p}, \mathbf{q}) = \sum_{(u,i)\in\mathcal{O}} (r_{u,i} - q_i p_u)^2 + \lambda(\sum_{i\in\mathcal{I}} ||q_i||^2 + \sum_{u\in\mathcal{U}} ||p_u||^2) \tag{3.1}$$

The coefficient λ is a manually defined regularization coefficient. Different optimization algorithms have been proposed such as alternated least squares or stochastic gradient descent (Koren, Bell, and Volinsky, 2009). Moreover, several extensions have been proposed regarding the shape of P and Q, by adding for example constraints for non-negativity (non-negative matrix factorization e.g [Lee and Seung, 1999; Paatero and Tapper, 1994; Luo et al., 2014]), sparsity (Kim and Park, 2007), or both (Kim and Park, 2008).

It is interesting to note that this family of methods is built such that it computes representations over a set of *a priori* known users and items. Integration of new ratings for existing users is not straightforward, as it requires partial or complete retraining

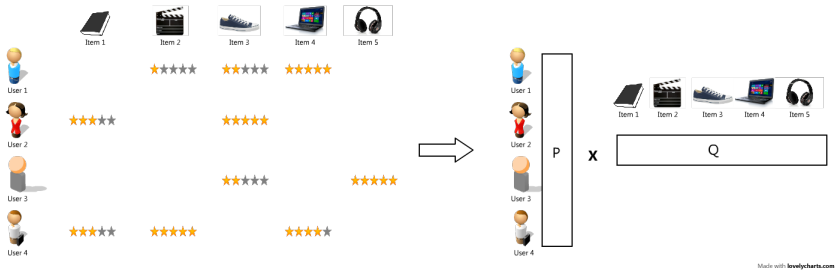


FIGURE 3.2: Matrix factorization in the recommendation problem : the goal is to find a decomposition of the rating matrix in two sub-matrices U and V, respectively the latent profile of the users (U_i the representation vector of user i) and of the items (V_j the representation vector of item j).

(strategies for updating $\mathcal{P}$ and $\mathcal{Q}$ w.r.t continuous incoming information have been studied for example in [Rendle and Schmidt-Thieme, 2008]). It is an interesting problem as recommender systems are typically expected to change through time, to gain new users and to recommend new items.

3.2.2 Cold-start recommendation

Collaborative filtering models generally suffer from a major drawback when they are confronted with completely new users with no rating. This setting is called in the literature the (user) *cold-start*[1]. While specific similarities have been designed to address this problem for memory-based models (Bobadilla et al., 2012, Ahn, 2008), a classical and intuitive approach is to use an interview process, where few questions are asked to a user to gather information (i.e ratings). Thus, the problem is then of choosing the more relevant questions to ask. Several papers have proposed different methods to tackle this problem.

Static approaches Static seed sets are constructed following a selection criterion fixed among all users, and results in a unique set of questions, the same for all new users incoming in the system. Rashid, Karypis, and Riedl, 2008 present a comparative study of different criteria :

- **Popularity** (items with higher numbers of ratings), which improves the chance the user will know the item and be able to rate it.
- **Entropy**, characterizing the dispersion of opinions of user on items.
- **Coverage**, which emphasizes items more heavily co-rated with other items.
- **Entropy0**, which considers missing ratings by setting a corresponding value for it (0). This palliates the tendency of Entropy to select "obscure" items.

[1]Note that the symmetric problem exists obviously with regard to new items, however, it is not studied in this thesis, it raises interesting questions as to how to adapt the following methods

- **HELF** (Harmonic mean of Entropy and Logarithm of Frequency) for an item i:

$$HELF(i) = \frac{2 * LF'_i * H'(i)}{LF'_i + H'(i)}$$

where $LF'(i) = log(|i|)/log(|\mathcal{U}|)$, the normalized logarithm of item i's rating frequency, and $H'(i) = -\sum_{k \in rangeratings} p_{i,k} log(p_{i,k})/log(5)$ the normalized entropy of the item.

- **GreedyExtend** :Golbandi, Koren, and Lempel, 2010 propose a greedy algorithm to construct a set : greedily add items to the seed set such that the item minimizes the error metric of the prediction performed with the actual seed set.

Adaptive approaches have also been proposed, where the interview process considers the user's previous answers to choose the next questions. For example, Rashid, Karypis, and Riedl, 2008 fit a decision tree to find a set of clusters of users, while Golbandi, Koren, and Lempel, 2011 use a ternary tree where each node is an item and branch corresponds to eventual answers (*like, dislike, unknown*). Zhou, Yang, and Zha, 2011 present *functional matrix factorization*, a decision tree based method which also associate a *latent profile* to each node of the tree. The closest model to our approach is from Sun et al., 2013, who learn a ternary tree allowing multiple questions at each node, each node containing a (learned) regressor and translations functions on selected items. Our model can be seen as one node of their tree. However, their approach does not seem to allow a bridge between cold start and warm context as ours does.
Another possible strategy is the use of *Active Learning* approaches (see [Rubens, Kaplan, and Sugiyama, 2011] for a general "foray" into Active Learning in recommender system). For example, Jin and Si, 2004 proposed a Bayesian Selection approach which has been extended by Harpale and Yang, 2008 to tackle the user cold start problem of selecting the ratings to ask for.
It is also interesting to note that these approaches while being usually more efficient, suffer from one major drawback for real-life applications: users usually dislike having to rate item *sequentially* and prefer rating several items at once in a single step, as shown by Golbandi, Koren, and Lempel, 2011 and Rashid et al., 2002.

Compared to these approaches, our model falls in the *static* category. The advantage and novelty of our approach is to propose not only a way of finding a subset of items for the interview process but also a global system for the recommendation problem, which helps at the same time to achieve better results in the cold-start setting, but to "survive" in the more usual setting with more ratings ("warm") and still be efficient compared to classical methods for this context. We propose to do so by conjointly learn a specific prediction model and the interview subset.

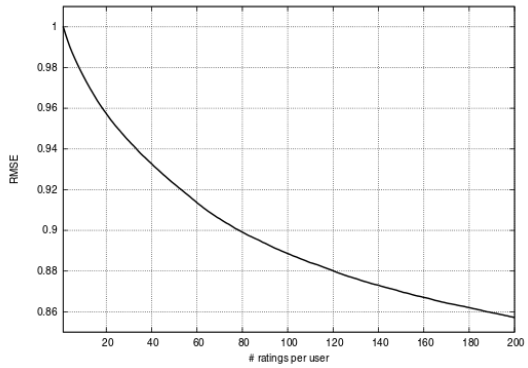


FIGURE 3.3: Test error rate vs. number of train ratings per user on the Netflix data for a model item-item factor, from [Golbandi, Koren, and Lempel, 2010]. Lower y-axis values represent more accurate predictions. The x-axis describes the number of ratings taken for each user.

3.3 Formulation of representation learning problem for user cold start

We presented in the previous section a brief overview of classical approaches in recommender system on one hand, and of the specific problem of cold-start on the other hand, where methods usually focus on how to best choose the questions of an interview process. As we noted, latent factor techniques are however somehow limited when dealing with ever-changing settings (new users, new items, and new ratings for already known users and items, arriving continuously in the system), as they will need to retrain often, which can be computationally expensive. Other collaborative filtering methods don't seem much more resilient to the small amount of information available on new users after an interview process (e.g 10 ratings). Golbandi, Koren, and Lempel, 2010 illustrate this in Figure 3.3, where one can see how the item-item factor method (Koren, 2010) performs w.r.t the number of ratings available for a user. This highlights how much the models are usually dependent from a minimum of ratings, thus making the cold-start problem difficult.

This motivates us to propose a model that not only learns the *interview* set of questions, but also rely on building representations in an *inductive* fashion. Changing the representation-learning paradigm to propose inductive methods has especially not been studied in the case of cold-start. However, it seems a powerful approach as it would provide similar prediction ability (in terms of RMSE) and allow to use simple methods to integrate interview learning. Moreover, we will see that this paradigm also allows us to provide a hybrid method that can cover all the "life" of the system, from the cold-start setting with a new user, to a more common *warm* setting, close to the classical problem of rating prediction, with several ratings available per user.

We first rewrite the cold-start problem coupled with representation learning in a

generic way. We then present the inductive approach that we propose for the representation learning part. Finally, we show how the model can be used in the cold-start setting as well as the warm-setting, thus tying the two situations together.

First, let us rewrite the objective function detailed in Equation 3.1 in a more general form that will allow us to integrate the user cold-start problem as a representation-learning problem. As seen above, we still consider that each item will have its own learned representation denoted $q_i \in \mathbb{R}^N$ and we focus on building a user representation. When facing any new user, our model will first collect a set of ratings by asking a set of queries during a static interview process. This process is composed of a set of items that are selected during the training phase. For each item in the interview, the new user can provide a rating, but can also choose to not answer if he has no opinion. This is typically the case for example with recommendation of movies, where users are only able to provide ratings on movies they have seen. The model will thus have to both select relevant items to include in the interview, but also to learn how (incomplete) collected ratings will be used to build a user representation.

Let us denote $\mathcal{Q} \subset \mathcal{I}$ the subset of items that will be used in the interview. The representation of a new incoming user u will thus depend on the ratings of u over $\mathcal{Q}$ that we note $\mathcal{Q}(u)$. This representation will be given by a function $f_\Psi(\mathcal{Q}(u))$ whose parameters, to be optimized, are denoted Ψ. These Ψ parameters are global, i.e shared by all users. The objective function of the cold-start problem (finding the parameters Ψ, the items' representations and the interview questions conjointly) can then be written as:

$$\begin{aligned}\mathcal{L}^{cold}(\mathbf{q}, \Psi, \mathcal{Q}) = & \overbrace{\sum_{(u,i)\in\mathcal{O}} (r_{u,i} - q_i^T f_\Psi(\mathcal{Q}(u)))^2}^{\text{prediction quality on all observed ratings}} \\ & + \underbrace{\lambda_1(\sum_i ||q_i||^2 + \sum_u ||f_\Psi(\mathcal{Q}(u))||^2)}_{\text{regularization on representations}} \\ & + \underbrace{\lambda_2|\mathcal{Q}|}_{\text{constraint on the size of the interview}}\end{aligned} \tag{3.2}$$

The difference between this loss and the classical CF loss is twofold: (i) first, the learned representations p_u are not free parameters, but computed by using a parametric function $f_\Psi(\mathcal{Q}(u))$, whose parameters Ψ are learned; this means that the representation of a user u is computed directly from his or her ratings; (ii) the loss includes an additional term $\lambda_2|\mathcal{Q}|$ which controls the trade-off between the quality of the prediction, and the size of the interview noted $|\mathcal{Q}|$; λ_1 and λ_2 are manually chosen hyper-parameters - by changing their values, one can obtain more robust models, and models with more or fewer interview questions. Note that solving this problem aims at simultaneously learning the items representations, the set of items in the interview, and the parameters of the representation building function.

3.3.1 Inductive Additive Model (IAM)

The generic formulation presented above cannot easily be optimized with any representation function. Particularly, the use of a transductive model in this context is not trivial and, when using matrix factorization-based approaches in that case, we only obtained very complex solutions with a high computation complexity. We thus need to use a more appropriate representation-learning function f_Ψ that is described below.

The Inductive Additive Model (IAM) is based on two ideas concerning the representation of users we want to build: (i) First, one has to be able to provide good recommendations to any user that does not provide ratings during the interview process $\mathcal{Q}$. (ii) Second, we want the user representation to be easily enriched as new ratings are available. This feature makes our approach suitable for the particular cold-start setting but also for the standard collaborative filtering setting as well.

Based on the first idea, IAM considers that any user without answers will be mapped to a representation denoted $\Psi_0 \in \mathbb{R}^N$. Moreover, the second idea naturally led us to build an additive model where a user representation is defined as a sum of the particular items' representations. This means that providing a rating will yield a **translation** of the user representation in the latent space. This translation will depend on the item i but also on the rating value. This translation will be learned for each possible rating value and item, and denoted Ψ_i^r, where r is the value of the rating. More precisely, in case of binary ratings *like* and *dislike*, the *like* over a particular item will correspond to a particular translation Ψ_i^{+1}, and a *dislike* to the translation Ψ_i^{-1}. The fact that the two rating values correspond to two different unrelated translations is interesting since, for some items, the *dislike* rating can provide no additional information represented by a null translation, while the *like* rating can be very informative, modifying the user representation - see Section 3.4 for a qualitative study over Ψ. The resulting model f_Ψ can thus be written as:

$$f_\Psi(u, \mathcal{Q}) = \Psi_0 + \sum_{(u,i) \in \mathcal{O}/i \in \mathcal{Q}} \Psi_i^{r_{u,i}} \tag{3.3}$$

where the set $\{(u, i) \in \mathcal{O}/i \in \mathcal{Q}\}$ is the set of items selected in the interview on which user u has provided a rating.

Note that this additive translation model for representing the users could be topped with a non-linear function (e.g hyperbolic tangent) as long as it is differentiable. This provides not only more complexity to the learned representations but also helps to tie the representations in a limited space. We provide further details on this regard in Section 3.3.3 where we study the case of going from the cold-start to the warm setting, where the number of ratings can vary a lot between users.

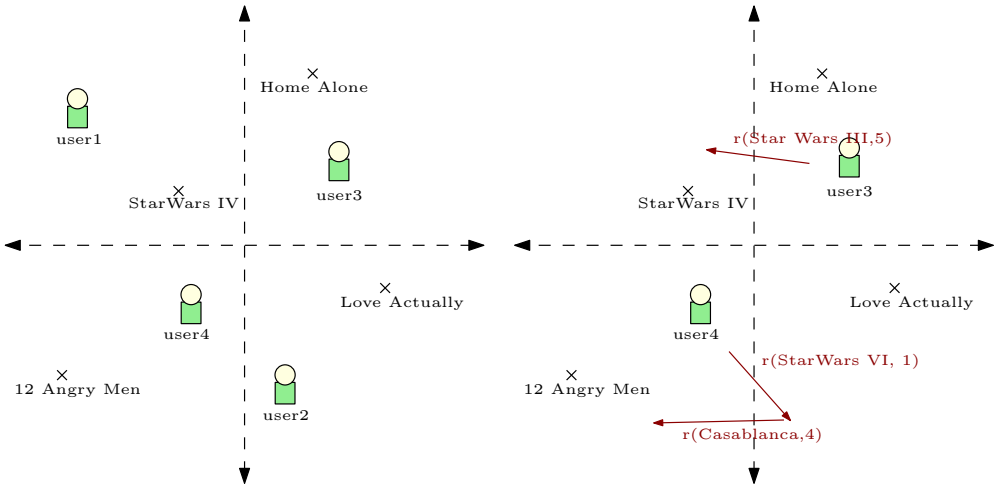


FIGURE 3.4: Difference between Matrix-Factorization (left) and IAM (right) w.r.t representations of users : while MF will learn simultaneously (final) representations for users and items, IAM learns the corresponding *translation* of a rating on item in the latent space. If a new rating is received (e.g *user3* rates *Star Wars III* with 5 stars), the user's representation simply moves in the latent space without needing to re-optimize.

Continuous Learning Problem

Now, let us describe how the objective function described in Equation 3.2 with IAM model described in Equation 3.3 can be optimized. Minimizing $\mathcal{L}^{cold}(\mathbf{q}, \Psi, \mathcal{Q})$ over $\mathbf{q}, \Psi$ and $\mathcal{Q}$ is a combinatorial problem since $\mathcal{Q}$ is a subset of the items. This combinatorial nature prevents us from using classical optimization methods such as gradient-descent methods and involves an intractable number of possible combinations of items. We propose to use a L_1 relaxation in order to transform this problem into a continuous one. Let us denote $\alpha \in \mathbb{R}^I$ a weight vector, one weight per item, such that if $\alpha_i = 0$ then item i will not be in the interview. The cold-start loss can be rewritten with α's as:

$$\mathcal{L}^{cold}(\mathbf{q}, \Psi, \alpha) = \sum_{(u,i)\in\mathcal{O}} (r_{u,i} - q_i^T f_\Psi(u, \alpha))^2 + \lambda|\alpha| \tag{3.4}$$

Where $|\alpha|$ is a L_1-norm. Note that the L_2 regularization term over the computed representation of users and items is removed here for sake of clarity. The representation of a user thus depends on the ratings made by this user for items i that have a non-null weight α_i, restricting our model to compute its prediction on a subset of items which compose the interview. If we rewrite the proposed model as:

$$f_\Psi(u, \alpha) = \Psi_0 + \sum_{(u,i)\in\mathcal{O}} \alpha_i \Psi_i^{r_{u,i}} \tag{3.5}$$

then we obtain the following loss function:

$$\mathcal{L}^{cold}(\mathbf{q}, \Psi, \alpha) = \sum_{(u,i)\in\mathcal{O}} (r_{u,i} - q_i^T(\Psi_0 + \sum_{(u,i)\in\mathcal{O}} \alpha_i \Psi_i^{r_{u,i}}))^2 + \lambda|\alpha| \tag{3.6}$$

which is now continuous. Note that, in that case, the translation resulting from a rating over an item corresponds to $\alpha_i \Psi_i^{r_{u,i}}$ rather than to $\Psi_i^{r_{u,i}}$. As mentioned above, one can choose to use a non-linear function on top of the additive model, therefore writing f_Ψ using for example a hyperbolic tangent function as:

$$f_\Psi(u, \alpha) = tanh(\Psi_0 + \sum_{(u,i)\in\mathcal{O}} \alpha_i \Psi_i^{r_{u,i}}) \tag{3.7}$$

which would result in the following loss:

$$\mathcal{L}^{cold}(\mathbf{q}, \Psi, \alpha) = \sum_{(u,i)\in\mathcal{O}} (r_{u,i} - q_i^T tanh(\Psi_0 + \sum_{(u,i)\in\mathcal{O}} \alpha_i \Psi_i^{r_{u,i}}))^2 + \lambda|\alpha| \tag{3.8}$$

Cold-Start IAM (CS-IAM) Learning Algorithm

This objective loss (Equation 3.8) can be optimized by using stochastic gradient descent methods, such as the one detailed in Algorithm 1. Given a set of training users and their observed ratings, we are going to "simulate" the interview process using the α weights and make predictions. More precisely, we consider for each user a subset of his or her ratings as available for the interview, and use the rest of the ratings to evaluate the prediction (more details regarding the experimental protocol is given in Section 3.4.1). Thus, at each iteration, a user is selected randomly (Line 3). His "train" ratings are used as inputs to the "interview" (for now it corresponds to the product of the sparse vector of training ratings with the weights α, as the null weights α_i will ignore some of the ratings), and from this we compute the corresponding representation of the user, as well as the predictions of the ratings. The loss is calculated with regard to the "evaluation" ratings and used to compute the gradient for each parameter $(\mathbf{q}, \Psi, \alpha)$ and to update them accordingly (Lines 6-9). Since the loss contains a L_1 term that is not derivable on all points, we propose to use the same idea than proposed in Carpenter, 2008, which consists in first making a gradient step without considering the L_1 term, and then applying the L_1 penalty to the weight to the extent that it does not change its sign. In other words, a weight α_i is clipped when it crosses zero. This corresponds to the lines 9-18 in Algorithm 1.

Algorithm 1 Learning algorithm for CS-IAM

Require: $\mathcal{O}$: set of observed ratings.
Require: ϵ : gradient step.
Require: λ_{l_1} for l_1-regularization.
1: Initialize $\mathbf{q}, \Psi, \alpha$ randomly.
2: **repeat**
3: Select a random user u, where r_u are the ratings of user u in $\mathcal{O}$.
4: Compute predicted ratings $\hat{r}_u = q^T tanh(f_\Psi(\alpha, r_u))$ ▷ See Eq. 3.7
5: Update parameters accordingly with gradient descent :
6: $\mathbf{q} \leftarrow \mathbf{q} - \epsilon \nabla \mathcal{L}^{cold}(\mathbf{q}, \Psi, \alpha)$
7: $\Psi \leftarrow \Psi - \epsilon \nabla \mathcal{L}^{cold}(\mathbf{q}, \Psi, \alpha)$
8: $\alpha \leftarrow \alpha - \epsilon \nabla$
9: L1-regularization on α using clipping.
10: **until** stopping criterion **return** $\mathbf{q}, \Psi, \alpha$

3.3.2 IAM and classical warm collaborative filtering

The IAM model, which is particularly well-fitted for user cold-start recommendation, can also be used in the classical collaborative filtering problem, without constraining the set of items. In that case, the objective function can be written as:

$$\mathcal{L}^{warm}(\mathbf{q}, \Psi) = \sum_{(u,i)\in\mathcal{O}} (r_{u,i} - q_i^T tanh(\Psi_0 + \sum_{(u,i)\in\mathcal{O}} \Psi_i^{r_{u,i}}))^2 \tag{3.9}$$

which can be easily optimized through gradient descent. This model is a simple alternative to matrix factorization-based approaches, which is also evaluated in the experimental section. This model have some nice properties in comparison to transductive techniques, mainly it can easily update users' representations when faced with new incoming ratings.

3.3.3 IAM from cold-start to warm collaborative filtering

We presented in the previous sections how the IAM can be used to tackle each problem of cold-start and warm CF separately. From this, the model can be easily adapted to bridge cold and warm context without additional retraining. This is a crucial and interesting aspect, as it prevents from having two distinct models (as it is often done in literature). The IAM model can be used in a unified way, with the ability to handle the transition between one setting (cold-start) to the other (warm). This can be done by simply changing the learning paradigm to a two-steps process as follow:

1. Learn the $\mathbf{q}$ and Ψ parameters on a set of training users (and their ratings) by optimizing the $\mathcal{L}^{warm}(\mathbf{q}, \Psi)$ from Equation 3.9, thus using all available ratings

without constraint. This is done in order to ensure learning of each item's translation in the representation space[2].

2. Consider the function $\mathcal{L}^{cold}_{\mathbf{q},\Psi}(\alpha)$ being defined as $\mathcal{L}^{cold}(\mathbf{q}, \Psi, \alpha)$ in Equation 3.8, but with parameters $\mathbf{q}, \Psi$ being fixed. Learn the alpha parameters for the cold-start interview process by optimizing on α the newly defined $\mathcal{L}^{cold}_{\mathbf{q},\Psi}(\alpha)$. This results in the optimal α (e.g interview process) for this specific representation model optimized for the warm setting.

In the cold-start case, the number of items selected is constrained by the α parameters, which naturally prevents any extreme variation of the representation's norm. In the *cold to warm* case, the number of items from which the representation is computed is expected to vary through time, and its norm might be unbounded, which could have undesirable effects. In such "mixed" setting, we recommend using the non-linear version of the representation model to limit the representation norm.

After the interview, each new incoming rating modifies the user representation as explained in Equation 3.7, resulting in a system that is naturally able to take into account new information.

3.4 Experiments

3.4.1 Experimental protocol

We evaluate our models on four benchmark datasets - Table 3.1- of various size in terms of the number of users, the number of items and with various ratings' sparsity. The datasets are classical datasets used in the recommender system literature (Zhou, Yang, and Zha, 2011,Golbandi, Koren, and Lempel, 2010). **ML1M** corresponds to the *MovieLens 1 million* dataset and **Yahoo** corresponds to the *Yahoo! Music* benchmark. **Flixter** and **Jester** are also classical recommendation datasets respectively on movies and jokes. As our main goal is mainly to evaluate the quality of our approach in the context of *new users* arriving in the system, we define the following protocol in order to simulate a realistic interview process on incoming users and to evaluate different models. We proceed as follow:

1. We randomly divide each dataset along users, to have a pool of *training* users denoted $\mathcal{U}^{train}$, representing 50% of the users of the complete dataset, in which we learn all the parameters of our model. The 50% users left are split into two sets, a pool of users for validation ($\mathcal{U}^{valid}$, 25% of total users) and a pool for testing ($\mathcal{U}^{test}$, 25% of total users).

[2]In cold-start setting, as the α force to ignore a large number of items, their translations are not learned as they are not used.

DataSet	Users	Items	Ratings
ML1M	5,954	2,955	991,656
Flixter	35,657	10,247	7,499,706
Jester	48,481	100	3,519,324
Yahoo	15,397	1000	311,672

TABLE 3.1: Number of users, items and ratings for each considered datasets.

2. The ratings corresponding to subsets $\mathcal{U}^{test}$ and $\mathcal{U}^{valid}$ are each randomly split into two subsets of ratings to simulate the possible known answers: we keep 50 % of the ratings for each subset as the possible answers to the interview questions (*Answer Set*).
3. The 50% of ratings left for both $\mathcal{U}^{test}$ and $\mathcal{U}^{valid}$ will be used for evaluating our models (*Evaluation Set*).

We illustrate the split in Figure 3.5. We keep the model with best results in validation (computed on ratings of subset $\mathcal{U}^{valid}$) and show the corresponding performance in test (computed on ratings of subset $\mathcal{U}^{test}$). Ratings have been binarized for each datasets, such that a rating of -1 (resp. 1) stands for a "dislike" (resp. a "like"). The quality of the different models is evaluated by two different measures. The *root mean squared error* (RMSE) measures the average ratings' prediction precision measured as the difference between predicted and actual ratings $(\hat{r}_{u,i} - r_{u,i})^2$. As we work with binary ratings, we also use the accuracy as a performance evaluation. In this context, it means that we focus on the overall prediction, i.e on the fact that the system has rightly predicted *like* or *dislike*, rather than on its precision regarding the "true" rating. The accuracy is calculated as the average "local" accuracy along users. These measures are computed over the set of missing ratings i.e the *Evaluation Set*.

We explore the behavior of our approach on both the classical CF context using the IAM Model (Equation 3.9) and on the cold-start problem using the CS-IAM model defined in Equation 3.6.

Baselines We compare our models with two collaborative filtering methods: Matrix Factorization (MF) that we presented earlier and the Item-KNN with Pearson correlation measure (Koren, 2010) which does not compute representations for users nor items but is a state-of-the-art CF method. Note that the inductive models (IAM and CS-IAM) are trained using only the set of training users $\mathcal{U}^{train}$. The ratings in the *Answer Sets* of $\mathcal{U}^{test}$ and $\mathcal{U}^{valid}$ are only taken as inputs during the testing phase (inference), but not used during training. Transductive models are trained using both the ratings of training users $\mathcal{U}^{train}$, but also the *Answer sets* of ratings defined over the testing users. It is a crucial difference as our model has significantly less information during training.

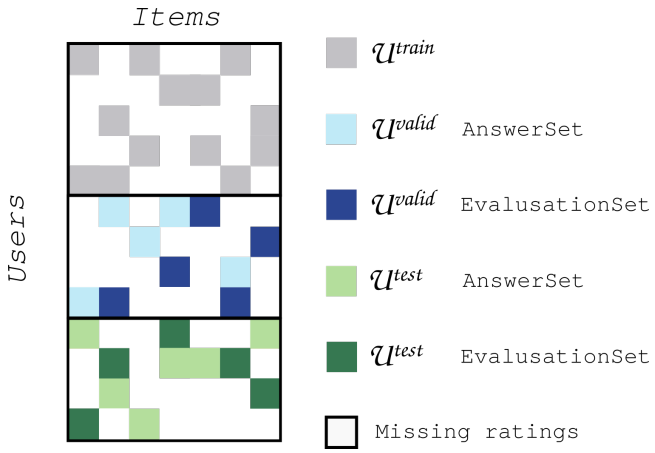


FIGURE 3.5: Experimental protocol: this represents the sparse matrix of ratings. The white parts represent unavailable ratings. A first subset of users is used as a training set (grey). The available ratings of a second set of users is split in two to create a validation set with ratings used as answers (light blue) and ratings used for evaluation (dark blue). A similar split is done on the last subset of users, with answers in light green and evaluation ratings in dark green.

The baselines used regarding the cold-start problem, therefore the strategy used to select a subset of items for the interview process, are the popularity criterion, where items are ranked with regard to the number of ratings (without considering the actual ratings), and the Harmonic mean of Entropy and Logarithm of Frequency (HELF), defined for an item i as:

$$HELF(i) = \frac{2 * LF'_i * H'(i)}{LF'_i + H'(i)}$$

where $LF'(i) = log(|i|)/log(|\mathcal{U}|)$, the normalized logarithm of item i's rating frequency, and $H'(i) = -\sum_{k \in rangeratings} p_{i,k} log(p_{i,k})/log(5)$ the normalized entropy of the item.

Each model has its own hyper-parameters to be tuned: the learning-rate of the gradient descent procedure, the size N of the latent space, the different regularization coefficients... The evaluation is thus made as follows: models are evaluated for several hyper-parameters values using a grid-search procedure, the performance being averaged over 3 different randomly initialized runs. The models with the best average performance on validation set are selected and the respective results on the test set are presented in the next figures and tables. All models have been evaluated over the same dataset splits.

3.4.2 Results

Collaborative Filtering

First, we evaluate the ability of our model to learn relevant representations in a classical CF context. In that case, the IAM model directly predicts ratings based on the ratings provided by a user. Results for the four different datasets are presented in Table 3.2. We can observe that, despite having much less information during the learning phase, IAM obtains competitive results, attesting the ability of the additive model to generalize to new users. More precisely, IAM is better than MF on three out of four datasets. For example, on the MovieLens-1M dataset, IAM obtains 72.7% in terms of accuracy while MF's accuracy is only 68.9%. Similar scores are observed for Jester and Yahoo. Although Item-KNN model gives slightly better results for one dataset, one should note that this method do not rely on nor provide any representations for users or items and belongs to a different family of approach, which can be a drawback if one wants to use the users/items representations for further (qualitative) analysis (e.g study behaviours of particular clusters of users or items). Moreover, ItemKNN - which is based on a KNN-based method - has a high complexity, and is thus very slow to use, and unable to deal with large-scale datasets like Flixter on which many days are needed in order to compute performance. Beyond its nice performance, IAM is able to predict over a new user in a very short time, on the contrary to MF and ItemKNN.

DataSet	MF	IAM	ItemKNN
Jester	0.723	**0.737**	0.725
ML1M	0.689	**0.727**	0.675
Yahoo	0.675	0.719	**0.726**
Flixter	**0.766**	0.758	NA

TABLE 3.2: Accuracy performance of different models in the classical Collaborative Filtering context (i.e without cold-start). NA (Not Available) means that, due to the complexity of ItemKNN, results were not computed over the Flixter dataset.

Cold-start Setting

We now study the ability of our approach to predict ratings in a realistic cold-start situation. As MF and ItemKNN do not provide a way to select a set of items for the interview, we use two benchmark static selection methods used in the literature (Rashid et al., 2002). The **POP** method selects the most popular items - i.e the items with the highest number of ratings in the training set - and the **HELF** (*Harmonic mean of Entropy and Logarithm of rating Frequency*) method which selects items based on both their popularity but also using an entropy criterion, which focus on the *informativeness* of items (e.g a controversial movie can be more informative than a

movie liked by everyone) (Rashid, Karypis, and Riedl, 2008). Our model is learned solely on the $\mathcal{U}^{train}$ set. Baselines are computed on a dataset composed of the original $\mathcal{U}^{train}$ ratings with the additional ratings of the *AnswerSets* of $\mathcal{U}^{valid}$ and $\mathcal{U}^{test}$ that lie into the set of items selected by the POP or the HELF approach. As before, transductive approaches use more information during training that our inductive model.

Figures 3.6 and 3.7 respectively show RMSE and accuracy results for all models on the Yahoo dataset as a function of the interview size. The size of the interview is straightforward to choose with POP and HELF, as you can select manually how many items you want in the interview, from the ranking obtain depending on the criterion. However, with CS-IAM, the size of the interview corresponds to the number of non-null α_i parameters, which is not adjustable directly, but through the value of the L_1 regularization coefficient. Therefore, we tested several values for this coefficient, in order to obtain different sizes of interviews.

The Figures 3.6 and 3.7 first illustrate that ItemKNN approach does not provide good results for RMSE-evaluation, as it is not a *regression*-based method, but is better than MF in terms of accuracy. It also shows that HELF criterion does not seem to be specifically better on this dataset than the POP criterion, while it is a slightly more complex and specific heuristic (and is presented as such in related work). For both evaluations, CS-IAM gives better results, for all sizes of interviews. It can also be noted that CS-IAM may give good results even if no item is selected thanks to the Ψ_0 parameters that correspond to a default representation (Equation 3.7). The model with 0 items also expresses the base performance obtained on users unable to provide ratings during the interview.

Detailed accuracy results for the four datasets are summarized in Table 3.3, for different reasonable sizes of interviews. Similar observations can be made on the results, where CS-IAM managed to have the best or competitive accuracy for all datasets and all number of questions allowed while using less information in train.

At last, when comparing the performance of CS-IAM with a version of IAM where items have been selected by the POP criterion -IAM-Pop, Figure 3.8 - one can see that the CS-IAM outperforms the other approaches. It interestingly shows that (i) IAM managed to give better results than MF with the same information selection strategy (POP) (ii) CS-IAM with all its parameters learned, managed to select more useful items for the interview process, illustrating that the performance of this model is due to both its expressive power and on its ability to simultaneously learn representations, and select relevant items.

We have shown that our approach gives significantly good quantitative results. We now focus our interest on a *qualitative* analysis of the results performed over the MovieLens dataset. First, we compare the items selected by the three selection methods (CS-IAM, POP and HELF). These items are presented in Table 3.4. First,

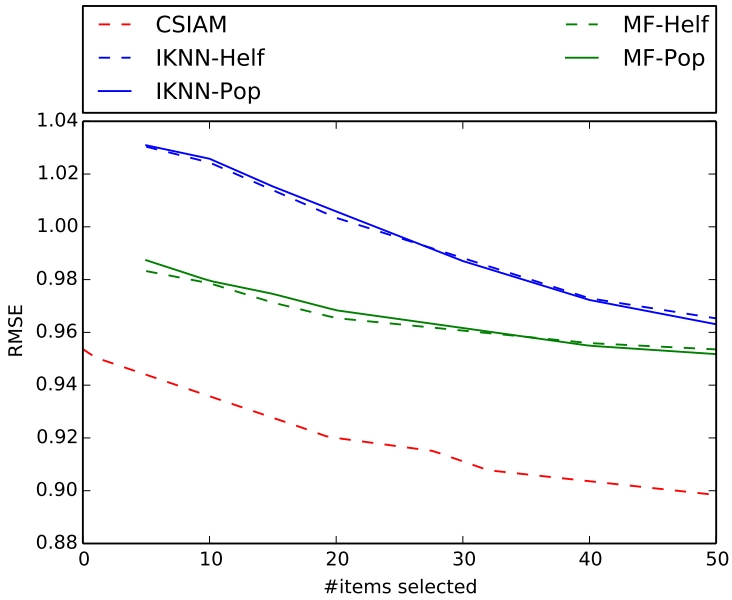


FIGURE 3.6: RMSE performance on **Yahoo** dataset for all models, regarding the size of the interview (number of questions/items asked)

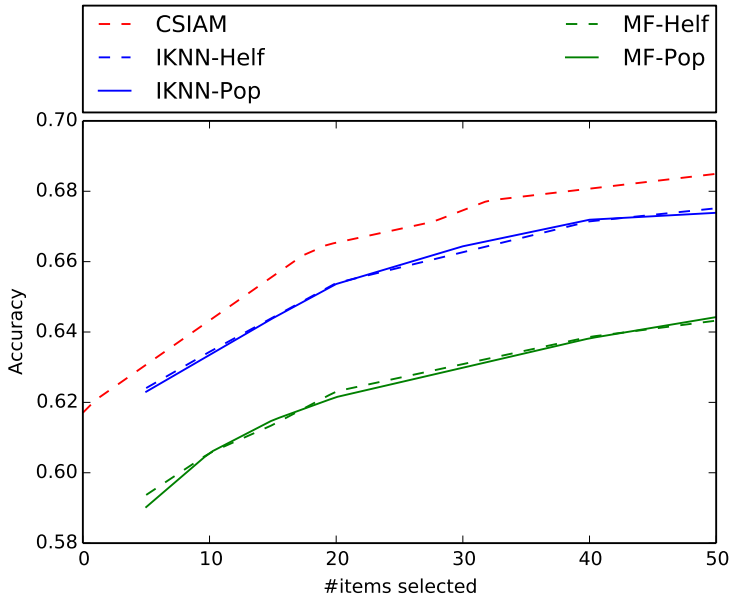


FIGURE 3.7: Accuracy performance on **Yahoo** dataset for all models, regarding the size of the interview (number of questions/items asked)

when using the POP criterion, one can see that many redundant movies are selected - e.g the three last episodes of Star Wars on which the ratings are highly correlated in the dataset: users usually like or dislike all three of them-. The same effect seems to appear also with CS-IAM which selects *Back to the future I* and *Back to the future III*. But, in fact, the situation is different since the ratings on these two movies have fewer correlations. Half of the users that like *Back to the future I* dislike *Back to the future III*.

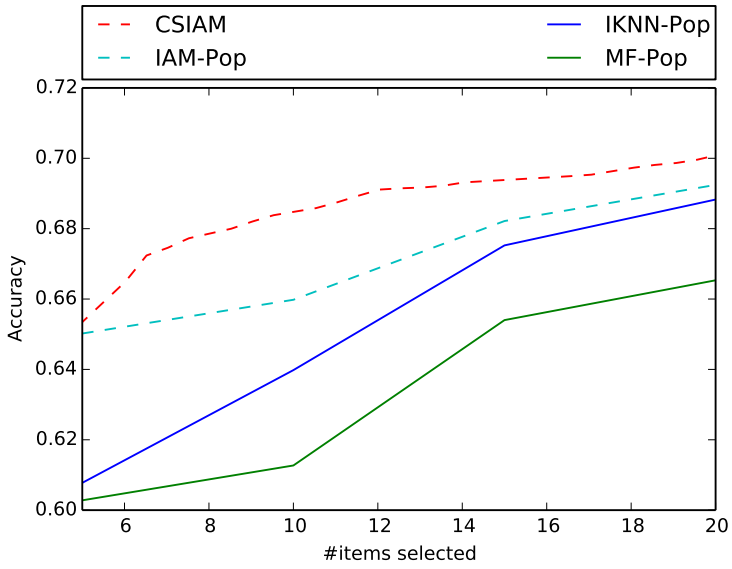


FIGURE 3.8: Accuracy performance on **Jester** dataset comparing the **Pop** selection criteria and the CS-IAM selection.

Figure 3.9 shows the translations $\alpha_i \Psi_i$ after having performed a PCA in order to obtain 2D representations. What we can see is that depending on the movie, the fact of having a positive rating or a negative rating does not have the same consequences in term of representation: For example, liking or disliking *Saving Private Ryan* is different than liking or disliking *Star Wars*; the translation concerning these two movies are almost perpendicular and thus result in a very different modification of the representation of the user. *Schindler's List* has fewer consequences concerning the user representation i.e the norm of $\alpha_i \Psi_i^r$ is lower than the others.

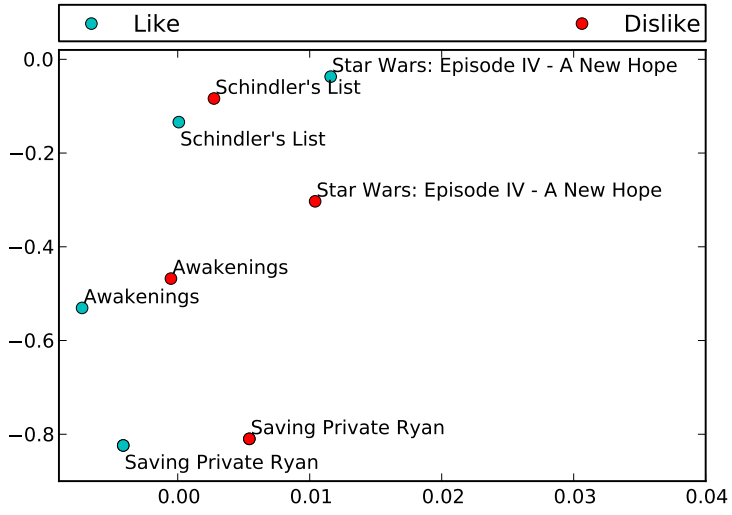


FIGURE 3.9: Visualization of some $\alpha_i \Psi_i$ after a PCA, on dataset MovieLens-1M

DataSet	NbItems	MF POP	MF HELF	IKNN POP	IKNN HELF	CS-IAM
Jester	5	0.603	0.589	0.608	0.634	**0.667**
	10	0.613	0.609	0.640	0.608	**0.686**
	20	0.665	0.641	0.688	0.676	**0.701**
MovieLens 1M	5	0.629	0.617	0.649	0.647	**0.690**
	10	0.634	0.620	0.651	0.653	**0.695**
	20	0.648	0.621	0.663	0.638	**0.696**
Yahoo	5	0.590	0.594	0.623	0.624	**0.638**
	10	0.601	0.610	0.633	0.634	**0.647**
	20	0.621	0.623	0.654	0.654	**0.665**
Flixter	5	0.719	0.722	NA	NA	**0.723**
	10	0.720	0.726	NA	NA	**0.727**
	20	0.727	**0.739**	NA	NA	0.735

TABLE 3.3: Accuracy performance of models on four datasets regarding the number of questions asked. NA (Not Available) means that, due to the complexity of ItemKNN, results were not computed over the Flixter dataset. Bold results corresponds to best accuracy.

CS-IAM	**Popularity**	**HELF**
American Beauty	American Beauty	Jurassic Park
Being John Malkovich	Star Wars: Episode I	Independence Day
Lion King	Star Wars: Episode V	Men in Black
Ghost	Star Wars: Episode IV	Total Recall
Superman	Star Wars: Episode VI	Mission: Impossible
Back to the Future	Jurassic Park	Speed
Fargo	Terminator 2	Face/Off
Armageddon	Matrix	Who Framed Roger Rabbit?
Get Shorty	Back to the Future	Abyss
Splash	Saving Private Ryan	Austin Powers
20 000 Leagues Under the Sea	Silence of the Lambs	Beetlejuice
Back to the Future Part III	Men in Black	Titanic

TABLE 3.4: MovieLens 1M - Selected items for the interview process by the three selection methods.

Mixing Cold-start and Warm Recommendation

We propose in this section to use our approach in a context where one want to move from *cold-start* to *warm* recommendation. After having answered the interview, the new user will start interacting with the system, eventually providing new ratings on its own. We follow the two-steps learning process presented in Section 3.3.3 to address this specific task. This approach is evaluated on the **Yahoo** dataset with the following experimental protocol being applied after learning:

1. Performance is firstly measured in the cold-start setting, using the items with non-null α's values for the interview process, as in Section 3.2.

2. We calculate the performance of this model when adding an increasing amount

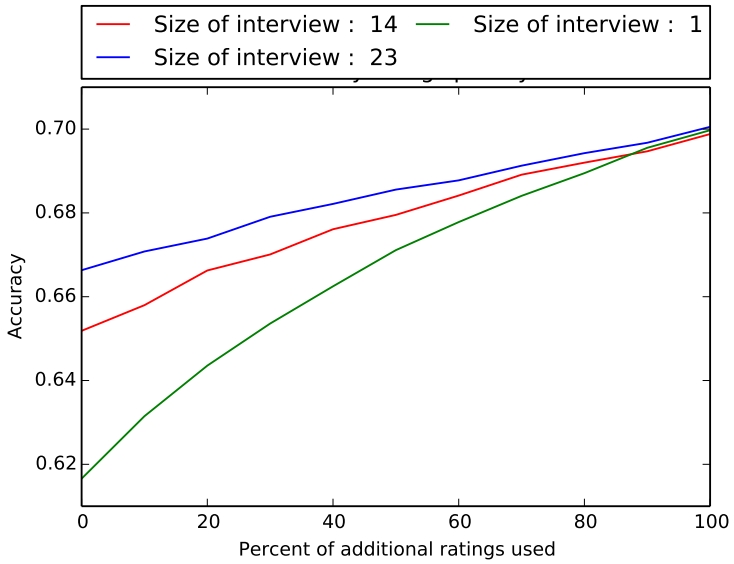


FIGURE 3.10: Accuracy regarding percentage of ratings added after the interview (from cold-start to warm setting) on dataset **Yahoo!**.

of "new" ratings sampled uniformly from the set of real available ratings left in the *Answer Set*.

The results are shown in Figure 3.10, which illustrates the accuracy given the percentage of additional ratings taken in the ratings left after the interview process, for three different sizes of initial interviews. This shows that this strategy starts (0% of additional ratings used) with a good accuracy performance, consistent with the results obtained in Table 3.3 on the strict cold-start context. The accuracy then increases as new ratings are added and almost reaches the one obtain for the classical warm setting (see Table 3.2).

This extension of our approach makes the link between the cold-start and the warm settings, which is an original and promising feature, and is novel compared to other classical approaches, which usually focus on either the cold-start or the classical warm context, and potentially need (extensive) additional computation to go from one to the other (typically for transductive models such as MF) without relearning.

3.5 Closing remarks

We presented in this chapter a method to tackle the problem of features selection for the cold-start in recommender systems, by proposing a different approach for representation learning and integrating the learning of static *interview set*. We showed that our approach performs well compared to different techniques in various settings, classical collaborative filtering as well as "budgeted" situations in cold-start.

However, we restricted ourselves here to *static* feature selection, where a unique subset of features will be acquired for each input. In the more general case of applications where features come at a cost, one intuitively tends to seek for *adaptive* acquisition methods instead, as it would provide more complex power of expression (by being able to choose different subset of features depending on what has been observed), thus more robust and efficient prediction ability. We propose to study in the following two chapters such adaptive approaches for features acquisition, with the presentation of two sequential models based also on a particular representation-learning scheme.

Chapter 4

Adaptive cost-sensitive feature acquisition - Recurrent Neural Network Approach

Abstract: The next two chapters present our contributions for the problem of *adaptive feature acquisition*. We focused on the previous chapter on static selection, where a unique subset of features was acquired for all inputs, based on representation learning and a specific architecture. This chapter introduces a model that integrates an adaptive acquisition process, by expanding the previously used techniques into a recurrent neural network architecture. We first situate the problem we choose to tackle, then we provide definitions of our model and finally we illustrate its ability on a variety of experiments. This chapter is based on the work presented in [Contardo, Denoyer, and Artières, 2016a].

4.1 Introduction

We propose now to study the problem of **cost-sensitive feature acquisition** more generally. We consider a target task (e.g classification), which should be solved using inputs x. The goal of feature acquisition is to choose a subset of features for every input that will guide the prediction. Ideally, the chosen subset of features should reduce the associated **acquisition cost** (e.g time or money), but without losing too much **prediction ability** [1]. In such more generic setting, the interest of choosing a static selection method is not relevant anymore. Indeed, for the majority of applications were budgeted prediction would be of use, it seems that an adaptive process of acquisition would be more efficient for the trade-off between cost and accuracy. For example, in one of the most commonly cited use-case for feature acquisition, medical diagnosis, the optimal acquisition strategy is obviously adaptive: the doctor decides the future tests (e.g a pet scan) to be conducted by observing some preliminary tests (e.g blood sampling). Figure 4.1 illustrates such sequential processes for acquisition of information : a doctor starts with similar tests for two patients (here

[1] Note that the cold-start problem can be "included" in this family of problems.

checking fever and blood pressure), and depending on the result, will choose to ask for blood sampling for the first patient (top sequence), but will take an electrocardiogram for the second patient (bottom sequence), and will then proceed with different tests after observing the respective results.

Therefore, an ideal machine learning model for such problem should ideally adapt its behavior (acquisition strategy) to what is currently observed on the input. Moreover, this example illustrates another aspect of the feature acquisition problem: each feature can potentially have a particular cost. To follow our medical example, a blood sampling is cheaper, faster to take and "easier" than an fMRI, which should ideally be taken into account in the overall acquisition and prediction process.

We can thus define the problem we propose to study in this chapter with the following key properties:

1. The optimality of a model is characterized as a trade-off between accuracy and inference budget.
2. Accurate prediction may be performed from a limited subset of features
3. Feature acquisition has a cost which is feature dependent: each feature can have a specific cost.
4. The optimal subset of features to acquire depend on the input example.

(Note that the previous method presented in Chapter 3 matches only the first two points.)

Our goal is also to propose methods able to deal with real problems that have potentially large sizes (e.g an important number of features to choose from). We present in this chapter a model that tackles this adaptive problem, using a recurrent neural network architecture. The rest of the chapter is organized as follow: we define more formally the problem at hand in Section 4.2.1, and then present the model and its generic components in the remaining of the Section. Section 4.2.3 specifies these components and gives further details about learning and inference algorithms. We study the quality of our approach on a variety of experiments in Section 4.3.

4.2 Adaptive feature acquisition with a recurrent neural network architecture

4.2.1 Definition of the problem

Let us first rewrite the learning problem under feature acquisition constraint, with a static selection scheme. We propose to express the corresponding loss of the problem for an example x as the trade-off between prediction ability (a loss Δ between the prediction of the model and the expected output y), and the cost of acquisition.

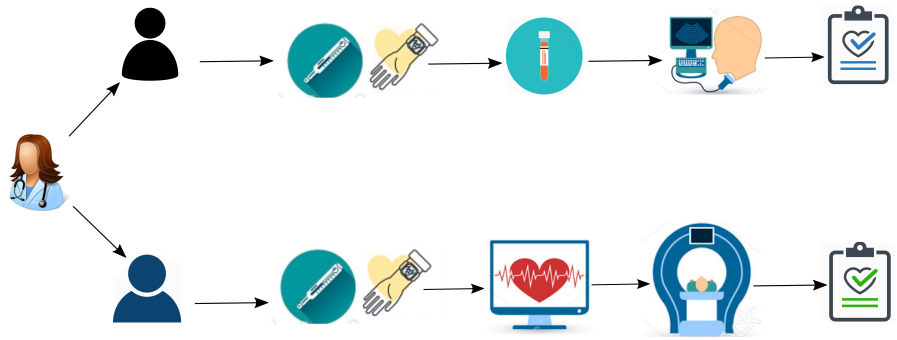


FIGURE 4.1: Adaptive acquisition process : the optimal information to acquire may depend on the current input. Here, a doctor performs on two patients different medical tests depending on their results.

For this cost, we keep the notations similar to the previous chapter, where we considered a binary vector α corresponding to $\alpha_i = 1$ if feature i is acquired by the system, 0 otherwise. We consider that this vector is predicted from a parametric function we write $f_\beta(.)$, thus β are the parameters to learn to predict α. We write $x[\alpha]$ the observed features of x w.r.t. the acquisition vector α (thus a partially observed x).
We define the prediction model with the generic notation $d_\theta(x[\alpha])$, which characterizes any parametric model that outputs a prediction based on the acquired features (w.r.t α) on input x. Thus, the input in itself is not completely observed during testing. This results in the following functions and generic loss:

$$\begin{aligned} \alpha &= f_\beta(.) \\ \hat{y} &= d_\theta(x[\alpha]) \\ \mathcal{L}(\theta, \beta) &= \Delta(d_\theta(x[\alpha]), y) + \lambda|\alpha| \end{aligned} \tag{4.1}$$

Where $|\alpha|$ is the L_1-norm of α. This loss can be rewritten to integrate an adaptive acquisition process instead of a static selection. To do so, we express the adaptive acquisition as a sequential process, therefore unrolling into several steps, where features can be acquired at each time-step t. Thus, instead of considering a unique acquisition vector α, we consider several vectors α_t, indexed by the time-step t. The prediction function is then expressed w.r.t its prediction parameters θ and all acquisitions vectors α_t that are predicted, as the final prediction will be made observing all the features acquired during the process. The cost is also modified to be the norm of the sum of all α_t for all steps. Here this means that the cost of acquiring a feature is counted only once, thus acquiring the same feature a second time comes for free[2]. When considering a finite and fixed number of acquisition steps T, the generic loss

[2]Note that this way of expressing cost is open to discussion as it can depend on the reality of application.

of adaptive acquisition of features then resumes to:

$$\begin{aligned} \alpha_t &= f_{\beta_t}(x[\alpha_{1:t-1}]) \\ \mathcal{L}(\theta, \beta_1, \ldots, \beta_T) &= \Delta(d_\theta(x[\alpha_{1:T}]), y) + \lambda |\sum_{t=1}^{T} \alpha_t| \end{aligned} \tag{4.2}$$

A specific cost per feature can be integrated in the second component of the loss. Let us note the vector of features' costs c, where c_i is the cost for the feature i. Then, the problem of adaptive cost-sensitive feature acquisition that we propose to study can be resumed by the following loss:

$$\mathcal{L}(\theta, \beta_1, \ldots, \beta_T) = \Delta(d_\theta(x[\alpha_{1:T}]), y) + \lambda \sum_{i=1}^{n} |\sum_{t=1}^{T} \alpha_{t,i}| c_i \tag{4.3}$$

We present now the generic aspects of the model we propose to learn such problem following this loss, and then describe more precisely its components.

4.2.2 Generic aspects of the model

We described on Chapter 2 an overview of the methods proposed for the (cost-sensitive) adaptive acquisition of features. We distinguished in the adaptive approaches different "families" of models, based on the ideas they were built upon : Reinforcement learning based methods, such as [Dulac-Arnold et al., 2012], which consider the acquisition process as a policy to learn, Cascade and Decision Trees based approaches, such as [Xu et al., 2014a], which often rely on weak decision trees that naturally reduce the amount of features used, Estimation of information value, such as [Bilgic and Getoor, 2007], which aims at estimating the information gain of subset of features to decide if it should be acquired, and in a more specific setting, Visual Attention models, such as [Ba, Mnih, and Kavukcuoglu, 2014], where acquisition mimics the focal eye attention on images, but are restricted to images inputs.These different families have various limitations, namely scaling ability with regard to the number of features (mostly reinforcement learning methods), the ability to easily learn models covering various acquisition costs (using a few features in average or a lot, or somewhere in between), and the combinatorial problem for approaches that propose to consider subsets of features, as they consider each possible subsets as a "particular feature" to acquire.

We present in this chapter a new kind of approach to tackle the cost-sensitive adaptive feature acquisition problem, under a framework that has not been studied before to the best of our knowledge. We propose a recurrent neural network generic architecture, with the motivation of providing a more scalable approach. The closest methods in terms of architecture are attention methods -presented in Chapter

2-, however, they assume that the input is completely observed at the beginning of the process, and rely on acquisition only to improve their prediction, to focus on some part of the input. They do not consider a cost nor constrain the amount of information used. The main key ideas behind our approach are the following:

1. **Differentiable components**: the method is designed within a framework that involves differentiable components only. This allows deriving a learning approach that relies on efficient gradient descent algorithms.

2. **Per-block feature acquisition**: Our method allows feature acquisition to be performed *per-block*, i.e several features are acquired at each step, without predefining the blocks before learning but nonetheless without suffering from the potential combinatorial problem that could arise. It is a novel aspect w.r.t state of the art models, where per-block approaches rely on manually pre-set blocks. Such "batch" acquisition is an interesting property as it helps to overcome some drawbacks regarding scalability when dealing with high dimensional data, and can be quite useful when dealing for example with computationally costly features (where acquiring several features at one given step allows to compute them in parallel, thus gaining time in overall).

3. **Representation learning**: a consequence of the adaptive property of the acquisition process is that the final prediction function (e.g a classifier), as well as the acquisition process, must be able to cope with any partially observed data (i.e the features gathered so far). Our approach overcomes this problem by relying on learning a shared latent space for representing an input whatever its observed features are. A representation is built sequentially for each input regarding the values of its previously acquired features. This representation is used to both drive the acquisition process and make the final prediction (e.g classify), tying the two parallel tasks together.

In the previous chapter, we defined a model with several components: a representation function for the users (user's ratings being the input) denoted f_Ψ, which can be interpreted as translation function for each item, items representations (denoted q_i), and a specific set of weights α for the feature selection parameters.
To integrate some adaptiveness, we now have to consider the acquisition as a **sequential process**, where the acquisition is driven by the currently seen observations. The generic idea here is to use a recurrent neural network architecture to do so. A representation will be built and updated through the sequential acquisition process, by adding the newly acquired features at each step. This representation will also guide the acquisition: the weights α will not be learned directly in the network but will be predicted by a specific layer taking the current representation as input. The generic process is illustrated in Figure 4.2, with T steps of acquisition. Note the sequence of α_t for the acquisition process, similarly as Equation 4.2.

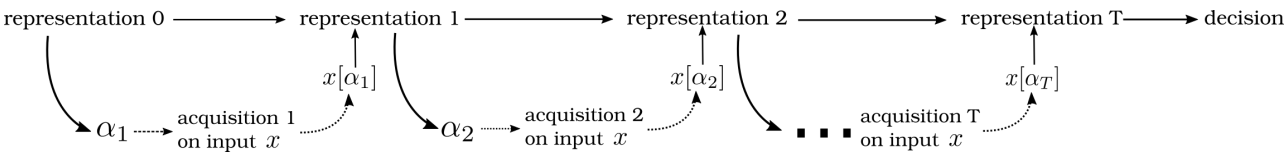


FIGURE 4.2: The generic process of adaptive features acquisition relying on a representation built in a recurrent fashion, from the previous representation and the observed features at a given step. The acquisition and the prediction are driven by this representation.

Note also that in our approach, we consider a number of acquisitions steps T fixed before learning. This number of steps will be the same for all examples, although it is possible for the system to acquire no feature at any step. This allows us to avoid the problem of "learning when to stop", which can be difficult in many cases, and is not really crucial and needed here. Let us now describe the model more formally.

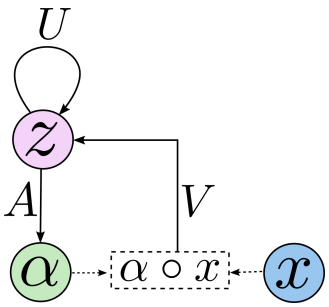
4.2.3 Recurrent ADaptive AcquisitIon Network (RADIN)

We consider a number of acquisition steps T fixed beforehand. One key aspect of our framework relies on the ability to predict which features to acquire next in step $t \leq T$ of the algorithm and on the ability to perform a prediction after T steps from any partially observed input sample (i.e. whatever the subset of features that have been observed on an input sample). We propose to rely on a representation space where one can express any partially observed input. More precisely, we consider that at each time step t, the partially acquired input $x[\alpha_{1:t}]$, is represented as a representation vector z_t lying in a p-dimensional latent space[3] $z_t \in \mathbb{R}^N$. This continuous vector z_t is the internal state of the recurrent neural network and is used to encode the information gathered (i.e the acquired features) on an input sample $x \in \mathbb{R}^n$. The updating process of the representation z_t involve two components, the *acquisition layer* in charge of choosing which features to acquire, and the *aggregation layer* in charge of aggregating the newly acquired information to the currently known one. The overall specific recurrent architecture we propose is illustrated in Figure 4.3b unfolded over T steps. We now describe each component of the architecture.

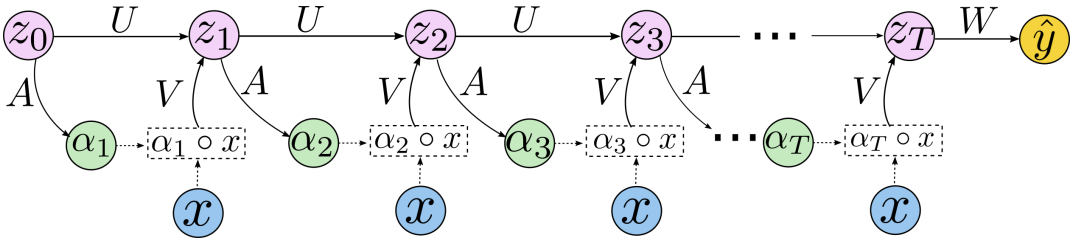
Components of RADIN

Acquisition Layer: While in classical RNN, the input at time t is usually a predetermined piece of the input (an element of an input sequence for example), in our case, this input is chosen by the model as a function of the previous state z_{t-1} in the following way: A specific *acquisition layer* (which can be compared to "attention" layer in visual attention models described in Chapter 2, e.g Mnih, Heess, and Graves, 2014) computes a vector $\alpha_t = h(A \times z_{t-1}) \in [0,1]^n$ whose component i

[3]For sake of clarity, we do not include x and a in the notation, using z_t instead of $z(x, t, a_1, ..., a_{t-1})$.



(A) The recurrent RADIN network, with acquisition parameters A, aggregation parameters U and V.



(B) Schema of RADIN's architecture, unfolded on T acquisition steps.

FIGURE 4.3: Architecture of the recurrent network for adaptive acquisition RADIN, folded (on top) and unfolded (bottom).

denoted $\alpha_{t,i}$ stands for the usefulness of feature i of the input denoted x_i. α_t is an attention vector that aims at selecting the features to acquire i.e the features i such that $\alpha_{t,i} > 0$. This vector is computed based on the previous representation z_{t-1} and different inputs will thus produce different representations, thus leading to different values of the acquisition layer resulting in an adaptive acquisition model. h is typically a non-linear activation function and $A \in \mathbb{R}^{n \times p}$ corresponds to the parameters of the acquisition layer. In order to compute the input of the hidden layer, the attention vector is then "mixed" with the original input x by using the Hadamard product. The acquisition layer acts as a filter on the features of x. This input is denoted $x\,[\alpha_t] = \alpha_t \circ x$ in the following. Note that in the particular case where a_t would be a binary vector, this stands for a copy of x where features that should not be acquired are set to 0. This vector $x\,[\alpha_t]$ is an additional input that is used to update the internal state, i.e. to compute z_t. The overall goal is to drop outputs (a lot or some, depending on the budget constraint) of the acquisition layer to zero, as it acts as a mask and acquire only features if $\alpha_{t,i}$ is positive. Therefore the use of an appropriate non-linear function is crucial to obtain such sparse outputs. Note that in this approach, we consider that the acquisition layer can acquire features that have already been gathered on previous steps.

Aggregation layer: Once the features have been acquired, the internal state z_t has to be updated, according to these features and the previous internal state. A possible

aggregation function is to update s.t $z_t = f(U \times z_{t-1} + V \times x\,[\alpha_t])$ (with U and V two weight matrices of size $p \times p$ and $p \times n$) as in classical RNN cells. The internal state layer z_t is thus an aggregation of the information gathered from all previous acquisition steps up to step t. It is also possible to use Gated Recurrent Unit (Cho et al., 2014b) or Long-Short Term Memory cells Hochreiter and Schmidhuber, 1997.

Decision Layer: The final representation z_T, which is obtained after the T-acquisition step, is used to perform classification $o(x) = g(W \times z_T) \in R^Y$ with g a non-linear function and W a weight matrix of size $Y \times p$, z_T being the representation of the input x at the end of the acquisition process.

Note that this architecture can be implemented as fully recurrent (where all weights U, V and A are shared), *semi*-recurrent, if one chooses to distinguish each acquisition steps, by learning specific matrices of weights A_t per acquisition step, or non-recurrent, where the weights of the aggregation layer (U and V) will be duplicated for every step (thus needing to learn matrices $U_t, V_t \forall t \in [1, T]$).
We now describe the corresponding loss equations of the model and its learning algorithm.

Loss and learning

Noting c_i the acquisition cost for feature i, $c_i \geq 0$ and $c \in \mathbb{R}^n$ the vector of all feature costs, the quantity $\sum_{t=1}^{T} \alpha_t^\intercal . c$ stands for the actual acquisition cost, provided that $a_{t,i}$ are actually binary values and that a feature cannot be acquired twice. Otherwise, it can be computed as $\sum_{i=1}^{n} | \sum_{t=1}^{T} \alpha_{t,i} | c_i$ (as in Equation 4.3). So ideally, the α weights should be binary. However, predicting real binary values or integrating a L_0 norm in the actual recurrent neural network would prevent learning as these are not differentiable components. Therefore, to handle the constraint on the predicted α_t to reduce the number of acquired features, we propose first to consider a continuous relaxation of the output of the *acquisition layer*. The acquisition function $f(A \times z_{t-1})$ thus outputs continuous weights α_t , ideally using activation functions providing stable and true 0 outputs (e.g Rectified Linear Unit (ReLU), or sigmoid using a threshold). Moreover, we propose to compute the evaluation of the cost as the product between the overall sum of each acquisition vectors α_t and the vector of cost c : $\sum_{t=1}^{T} a_t^\intercal . c$. This means that if a feature is acquired several times during the process, its cost will be counted each time. Howevern the cost will be weighted by the corresponding $\alpha_{t,i}$. Note that it corresponds to the "true" loss (Eq 4.3) if the system predicts binary acquisition value and can not re-acquire already seen features. While we do not have these constraints in our system, this loss induces the same effect as the original loss.

Algorithm 2 Learning algorithm for RADIN in a full recurrent setting.

Require: $\mathcal{D}$: training dataset of couple (x, y).
Require: ϵ : gradient step.
Require: λ : regularization of the acquisition constraint parameter.
Require: T : number of acquisition steps.
1: Initialize weights U, V, A, W. Initialize z_0 as a random vector.
2: **repeat**
3: Randomly select an input x and its expected output y.
4: **for** $t = 1, \ldots, T$ **do**
5: Compute $\alpha_t = h(A \times z_{t-1})$
6: Simulate acquisition by computing Hadamard product :
7: $acquisition_t = x \circ \alpha_t$
8: Update internal state : $z_t \leftarrow f(U \times z_{t-1} + V \times acquisition_t)$
9: **end for**
10: Compute prediction: $\hat{y} = g_W(z_T)$
11: Compute prediction error : $e = \Delta(\hat{y}, y)$
12: Update weights w.r.t gradient error (using back-propagation):
13: $U \leftarrow U - \epsilon \nabla \mathcal{L}_{RADIN}(A, U, V, W)$
14: $V \leftarrow V - \epsilon \nabla \mathcal{L}_{RADIN}(A, U, V, W)$
15: $W \leftarrow W - \epsilon \nabla \mathcal{L}_{RADIN}(A, U, V, W)$
16: $A \leftarrow A - \epsilon \nabla \mathcal{L}_{RADIN}(A, U, V, W)$
17: **until** stopping criterion
18: Return A, U, V, W, z_0

The corresponding loss of the model for an example x resumes to:

$$\mathcal{L}_{RADIN}(A, U, V, W) = \Delta(g_W(z_T), y) + \lambda \sum_{t=1}^{T} a_t^\intercal . c \tag{4.4}$$

With $z_T = f_{U,V}(z_{T-1}, x, \alpha_T)$, and $\alpha_t = h_A(z_{t-1})^\intercal, \forall t = 1, \ldots, T$, following the recurrence. The first term of the loss is a data fit term that measures how well prediction is performed on training samples and the second term is related to the constraint on the feature acquisition *budget*. This loss if fully differentiable and can therefore be used to learn using gradient descent optimization algorithms such as stochastic gradient descent or ADAM (Kingma and Ba, 2014). We describe in Algorithm 2 how learning is achieved. The loop in Lines 4 to 9 correspond actually to forward x in the recurrent neural network:the initial representation z_0 is forwarded in the acquisition layer, i.e computing the first set of acquisition weights $\alpha_1 = h(A \times z_0)$ (where for example h is a ReLU function). The corresponding features are "acquired" and re-injected in the process through the Hadamard product ($acquisition_1$), and used to update the internal state, leading to $z_1 = f(U \times z_0 + V \times acquisition_1)$, where f is for example a GRU. This process is repeated T times, thus leading to T acquisition vectors α_t and the final representation z_T. After the loop, prediction is computed (line 10), as well as the corresponding error. Using back-propagation of the error, the weights are updated following gradient error. One can observe that during learning we consider the inputs x to be fully observable for free.

Algorithm 3 Inference algorithm for RADIN

Require: A, U, V, W : weights of the network and z_0 common starting representation.

Require: x : a new input, technically "void", on which to acquire the features..

1: **for** $t = 1, \ldots, T$ **do**
2: Compute $\alpha_t = h(A \times z_{t-1})$
3: Acquire the features i on x s.t $\alpha_{t,i} > 0$:
4: Create vector $x[\alpha_t]$ s.t $x[\alpha_t]_i = x_i$ if the feature is acquired, $x[\alpha_t]_i = 0$ otherwise.
5: Compute weighted partial acquisition of x : $acquisition_t = x[\alpha_t] \circ \alpha_t$
6: Update internal state : $z_t \leftarrow f(U \times z_{t-1} + V \times acquisition_t)$
7: **end for**
8: Compute prediction: $\hat{y} = g_W(z_T)$
9: Return $\hat{y}$ and a binary vector of the acquired features (i.e $| \sum_{t=1}^{T} \alpha_t |$).

It is worth mentioning here that the architecture we present allows feature acquisition to be performed **per block**, i.e. the system can ask to acquire several features at a given time-step, as the output of the acquisition layer is not constraint w.r.t the number of non-null features per step, and during learning acquisition is done by using the weights α_t as a mask with Hadamard product.

Inference

We explain now how inference is done on a new example using RADIN. The process is describe in pseudo-code in Algorithm 3. It is similar to the learning process, with a loop for the T steps of acquisition, where the system starts with the initial representation z_0, which leads to the first acquisition vector α_1. However, during inference, instead of using α_1 as a mask on the whole input x, here acquisition is processed, for all non-null features of α_t. We thus consider a sub-observation of the current input, $x[\alpha_1]$, which corresponds to the partial view of x following acquisition at the current step 1. This sparse partially observed vector is then integrated to update the representation as during learning, with $z_1 = f(U \times z_0 + V \times acquisition_1)$, for the first step. Note that it is the observation $x[\alpha_1]$ weighted by α_1 (as it is not a binary vector) that is used to update the representation, in order to keep eventual weighting between features. This is repeated T times, for each acquisition step, leading to the final representation z_T used for final prediction (line 9). The binary vector of acquired features can be easily computed from all α_t (line 10) and used to compute the real acquisition cost.

4.3 Experiments

This section provides results of various experiments on feature-acquisition problems with different cost-settings. We study the ability of our approach on several

mono-label classification datasets [4].

4.3.1 Experimental protocol

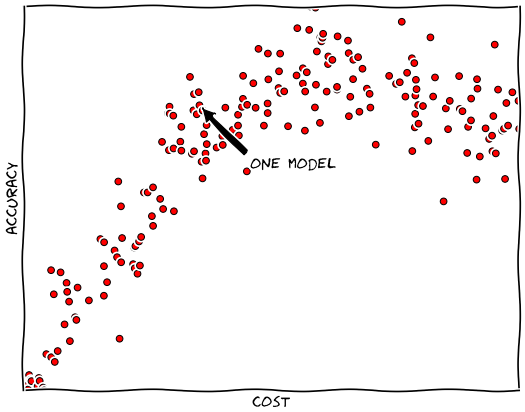
The main goal of the experiments is to determine the ability of our models to give good predictions (i.e accuracy) while constraining the amount of acquired information (i.e cost). We have in this regard a double-target task to validate and evaluate: accuracy *and* cost. We propose the following experimental validation protocol to tune and assess the results of all the models in our experiments, where each dataset has been split into training, validation and testing sets (the splits are common to all models), each split corresponding to one third[5] of the examples:

1. A set of models is learned **on the training set** with various hyper-parameters values.
2. We compute for each model the resulting accuracy and cost on **the validation set**. This yields a two dimension point (accuracy,cost) of performance for each model. This is illustrated in Figure 4.4a (artificial points), where each red dot corresponds to the average performance of a particular model on validation dataset.
3. To select the best models in validation, we compute the *Pareto front* of this set of points : we pick all models that are not dominated by another model on both criteria, i.e with a higher accuracy and a lower cost. These models correspond to various trade-offs between accuracy and cost. The Pareto front is illustrated in Figure 4.4b, where the blue dots corresponds to the model that are not dominated by any other in the set of results.
4. For each selected Pareto-models (blue dots), we evaluate the average accuracy/cost **on the test set**. These new points are the green rectangle in Figure 4.4c and are used to produce the test curve results, which are the curves we present in the remaining of the Section. Note that you can observe in the Figure that not only the average accuracy vary from validation to test, but also the average cost of acquisition. These curves have been used to extract results for a given precise cost (e.g in the Tables), using interpolation to provide potentially missing accuracy value for a particular cost.

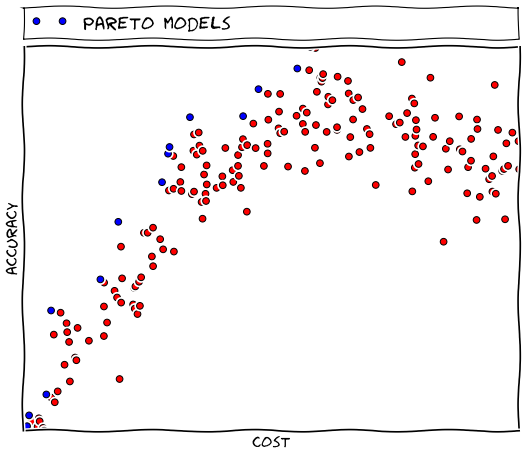
Model: Results shown in this manuscript are obtained with a semi-recurrent architecture, with a specific set of weights for each acquisition step (as explained in Section 4.2.3).

[4] Note that our approach also handles other problems such as multi-label classification, regression or ranking as long as the loss function Δ is differentiable.

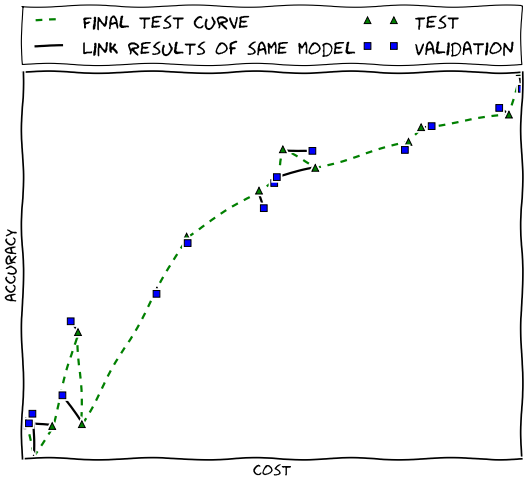
[5] Except for MNIST, where the train/validation/test split correspond resp. to 15%,5%,80% of the data.



(A) Performance (average accuracy/cost) of models on validation dataset



(B) Selection of Pareto models (blue circle) in the set of models:all models that are non-dominated (i.e there is no model with higher accuracy and lower cost) are selected.



(C) Computation of performance for each Pareto models on the test dataset : green triangle points, linked by a black solid line corresponds to the performance (average accuracy/cost) of the respective model that obtained the "blue squared" performance. Dashed green curve is the final resulting curve, equivalent to what the curves we show in this Section.

FIGURE 4.4: Illustration of the different steps of our validation protocol

We used linear functions for g_W throughout all our experiments. We tested two non-linear functions as aggregation function $f_{U,V}$:

- Classical RNN cell: $f_{U,V}^{RNN}(z_t, a_t, x) = tanh(U \times z_t + V \times (a_t \circ x))$
- Gated Recurrent Units (GRUs) as presented in [Cho et al., 2014b] and Chapter2, which is composed of a *reset gate* r_t and an *update gate* h_t. More formally, the output of a GRU is a linear interpolation between the previous representation (z_{t-1}) and the candidate representation :

$$z_t = (1 - update_t)z_{t-1} + update_t\hat{z}_t$$

Where the update gate $update$ decide how much the unit updates its activation. The update gate is computed as

$$update_t = \sigma_{update}(V_{update}x_t + U_{update}z_{t-1})$$

The candidate representation is computed similarly to recurrent unit with a reset gate :

$$\hat{z}_t = tanh(V_z x_t + U(reset_t \circ z_{t-1}))$$

Where the reset gate is computed in a similar fashion as the update gate :

$$reset_t = \sigma_{reset}(V_{reset}x_t + U_{reset}z_{t-1})$$

Final output is thus computed as

$$z_t = (1 - update_t) \circ z_{t-1} + update_t \circ \sigma_z(V_z x_t + U_z(reset_t \circ z_{t-1}))$$

We used a hard logistic activation function for the acquisition layer, and a mean-square error as Δ but other functions could be used. Different sizes of latent spaces have been tested ($N \in \{10, 25, 50, 100\}$), and different learning rates ($\eta \in \{00.1, 0.01, 0.1\}$) have been used with a large number of iterations. The model has been launched with different numbers of steps : $T \in \{1, 2, 3, 5\}$ At last, different sizes of mini-batches have been tested $\{1, 10, 100\}$.

Baseline models: our three variants are compared with different features selection approaches:

- **SVM** L_1 is a L_1 regularized linear SVM. We used the implementation available in the *scikit-learn* package in Python and optimized on hyper-parameter C.
- **Decision Trees** can be seen as particular cases of sequential adaptive predictive models. We used the implementation available in the *scikit-learn* package

in Python. [6] We tested various maximum depth value (up to not limiting the depth) and criterion (*gini* / *entropy*).

- **Greedy Miser** (Xu, Weinberger, and Chapelle, 2012) is a cost-sensitive model that relies on several weak classifiers (Decision Trees) where the acquisition cost is integrated as a local and a global constraint. We used the MATLAB implementation provided by the authors[7]. We conducted grid-search for the following set of parameters: number of trees, lambda (regularization parameter), learning rate and depth of trees(for CART algorithm).

Note that other models like RL-based approaches (e.g Dulac-Arnold et al., 2012) have not been tested since their complexity is too high to handle datasets with dozens of features, but the results obtained by our models on small datasets are competitive with the results previously published with these methods.

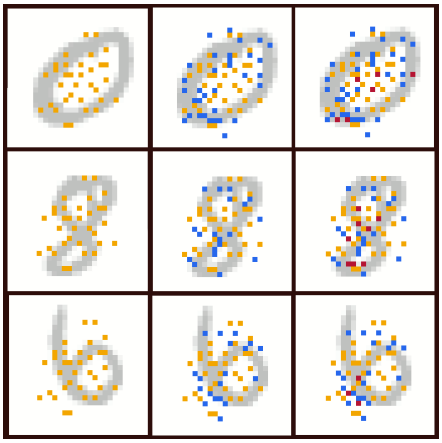
4.3.2 Results

Illustration of the adaptive acquisition process

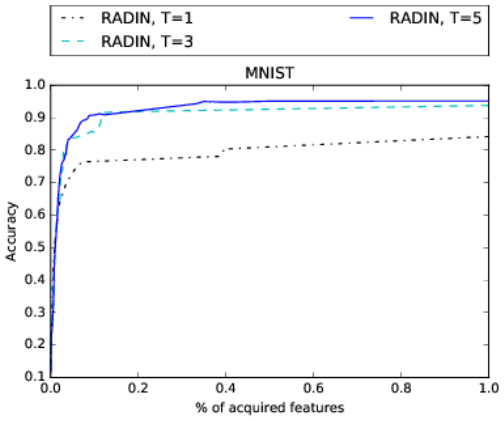
We propose first to illustrate on a visual task how our approach behaves, on the dataset MNIST. Figure 4.5a shows the pixels that are acquired at each step of the process for different inputs. Here, we show the acquisition of pixels for three images of numbers 0, 8, and 6, at test time, for a model RADIN with $T = 3$ steps of acquisition. On the left column, in yellow, are shown the pixels acquired at the first steps: the subset of pixels is the same for all inputs, as it depends only on the initial representation z_0. The second subset of pixels acquired is shown in blue, in the middle column, and the third in red, in the far right column. We can observe here the adaptiveness of the process: 6 pixels are acquired for input "0" at the third step, 8 pixels for input "8" and 3 for input "6", at a different location. In this case, each pixel has the same cost, but the system can gain on the overall budget by using fewer pixels for easier input (apparently "6") and a bit more for unclear input (here "8", where we can observe that the acquisition happens in the "middle", maybe to distinguish between a close number, like "0"). Note that we do not make any assumption regarding the shape or nature of the input in the model, therefore the input is processed as a vector (by flattening the images), unlike visual attention model (Gregor et al., 2015, see Section 2.1.2). This explains why the acquisition is not made by "patch", and seems less visually intuitive than the acquisition provided by methods designed for image inputs, which rely on their specific nature.

The Figure 4.5b illustrates more quantitatively the advantage of the adaptive process by plotting the accuracy/cost curves obtained for models RADIN learned with

[6] Note that these two baselines don't allow to integrate a specific cost per feature during learning.
[7] http://www.cse.wustl.edu/~xuzx/research/code/code.html



(A) Illustration of the adaptive behavior of RADIN with three acquisition steps on three different MNIST inputs. The first pixels acquired are pictured in yellow on the left, the second step in blue in the middle, and the last pixels acquired are in red, on the right.



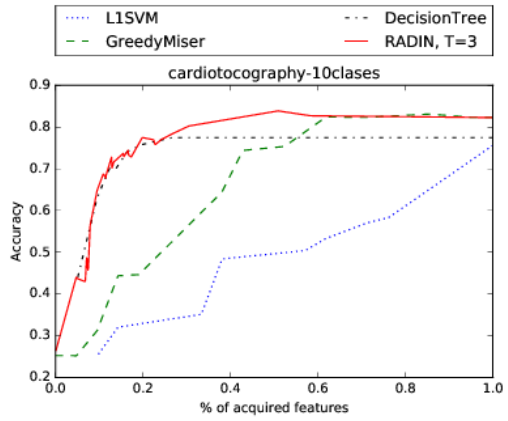
(B) RADIN with various numbers of acquisition steps (T) on *MNIST*. Abscissa is the percentage of acquired features, ordinate is the obtain accuracy. This illustrates the advantage of having an adaptive process (solid and dashed curves for 3 and 5 steps) *versus* a static selection (dotted black curve).

FIGURE 4.5: Adaptive acquisition of features: quantitative and qualitative advantages.

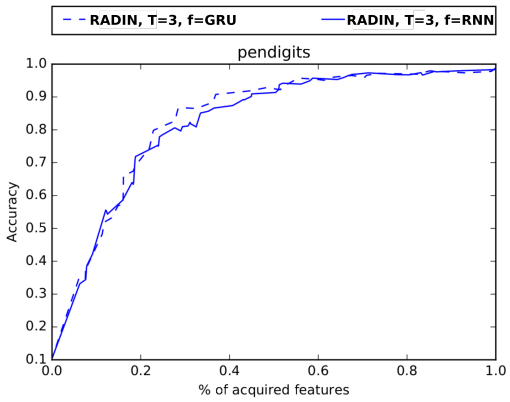
a different number of acquisition steps T: 1,3 and 5. We see here the gain in accuracy between a static process ($T = 1$) or an adaptive one ($T = 3$ or 5), for all the cost range (here the ratio of pixels acquired: 0.2 in abscissa means that 20 percent of the features have been acquired on average on the test inputs). The difference of accuracy performance when all features (100%, 1 in abscissa) are acquired can be explained by the non-linearity of the aggregation function, which provides a more complex, in some way "deeper", network.

Uniform cost

We propose now to further study the results on experiments with **uniform cost**, i.e $c_i = 1, \forall i$, on more diverse datasets. As mentioned above, in this case, the acquisition cost is directly the number of features gathered. The cost is thus expressed as the percentage of features acquired w.r.t the total number of available features. Figure 4.6a illustrates the overall accuracy-cost curves for the dataset *cardio*. One can see for example that the GreedyMiser approach yields an accuracy of about 68 % for a cost of 0.4, i.e acquiring 40 % of the features in average on test inputs, while our model RADIN obtains approximately 82 % of accuracy for the same amount of acquired features. The results in Table 4.1 show the ability of our method to give competitive or better results on a variety of datasets from UCI. These datasets have various size (number of features from 8 to 780, number of examples in training from 541 to 10000) and cover problem of classification from 2 classes to 26 classes.



(A) Curves "accuracy/cost" on *cardio* for all considered methods. Abscissa is the percentage of acquired features, ordinate is the obtain accuracy.



(B) Comparison of performance on *pendigits* with regards of the choice of the aggregation function $f_{U,V}$ with 3 acquisitions steps.

FIGURE 4.6

We summarize the results by focusing on 5 sparsity levels: 90%,75%,50%,25% and 10% of features used. Best accuracy for each sparsity levels is highlighted in the Table. One can see that our approach leads to competitive performance, with the best accuracy on a majority of datasets and acquisition level (42 best accuracies for 60 couples dataset-sparsity). Interestingly, some dataset need really few features to obtain interesting level of accuracy, for example "page-blocks" (10 features total) where our approach, as well as the decision tree baseline, manage to obtain 93% of accuracy with an average sparsity of 10% (i.e 1 feature acquired)[8]. However, the three baselines provide results that are less stable among datasets and levels of acquisition, performing well on some and poorly on other. More particularly, GreedyMiser is competitive on almost all dataset, except for high levels of sparsity (e.g "cardiotocography" or "statlog", where compared to our performance, GreedyMiser drops significantly when having few features acquired). DecisionTree, while seeming a quite naive baseline for the problem, actually gives reasonable results on several datasets and even perform best on some (e.g "letter"). However, it is interesting to note that some results of this model are similar for all levels of acquisition (for example in "Statlog","mnist" or "musk"), as it was difficult to force the algorithm to pick more features. As we use a validation protocol relying on selecting the Pareto front, it is also possible that the models achieving higher acquisition cost were generalizing poorly on the validation set, thus weren't selected in the Pareto-set.

Regarding the choice of the aggregation function in our architecture, the impact of choosing a Gated Recurrent Unit or a more simple cell without gate seems to indicate a slight advantage for the GRU. For example, we plot in Figure 4.6b the curves results for models with 3 acquisitions steps with GRU *vs* RNN. It illustrates that

[8] Yet, note that this dataset is quite unbalanced, with one class representing 89% of examples.

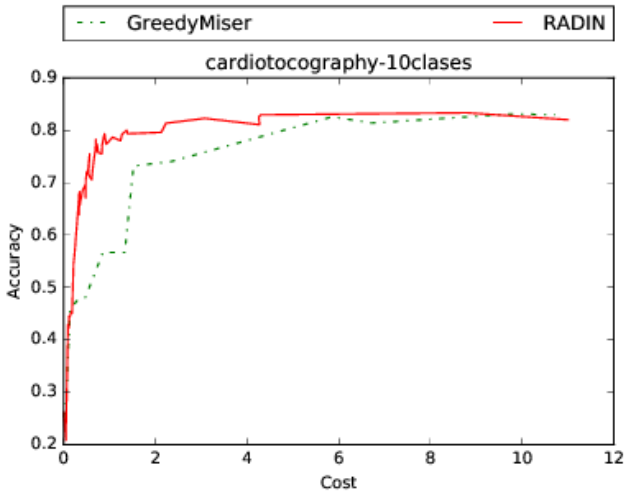


FIGURE 4.7: Cost-sensitive setting on *cardio* dataset, where model RADIN has 3 acquisition steps.

both models perform almost equally well, except between 20% and 50% of features acquired, where GRU manages to gain a few percent in accuracy.

Results on larger datasets (5000 to 6224 features) are provided in Table 4.2, with 4 levels of acquisition: 25%,10%,5% and 1% of features used. Our approach performs well on these datasets more specifically when the percentage of acquired features drops substantially, e.g to 1 or 5 % (e.g on dataset r8, where our approach still manage to obtain 95.9 % of accuracy using only 62 features out of 6224, while the second best accuracy is given by GreedyMiser which achieves 93.9 %). It is interesting to note however that simple (naive) method such as SVM L_1 still manage to perform well and competitively on these tasks, particularly if the inputs contains mainly noisy features and a few relevant features to acquire "no matter what" (typically the dataset "gisette" is built in such a way, where about 90% of features are to be discarded as useless and noisy).

Cost-sensitive setting

We propose now to study the ability of our approach in a cost-sensitive setting, by testing it on datasets with artificial costs. We choose to define the cost of feature i as $c_i = \frac{i}{n}$, which makes no assumption with regard to the potential real meaning or usefulness of the features. Figure 4.7 shows the performance obtained by RADIN, with a number of acquisition steps of $T = 3$, and GreedyMiser on such artificial cost-sensitive version of the dataset "cardiotocography". One can observe that our model yields better accuracy results than Greedy Miser for all the cost range, which indicates its ability to not only acquire the relevant features but also to integrate into the process their different costs c_i.

Corpus Name	Nb. Train Ex	Nb. Feat	Nb. Cat	Model	Amount of features used (%)				
					90%	75%	50%	25%	10%
Abalone	1396	8	3	SVM L_1	64.3	63.2	60.0	56.7	50.5
				DecisionTree	60.5	60.5	60.5	61.3	56.1
				GreedyMiser	63.8	63.6	60.3	59.9	54.9
				RADIN	63.9	64.0	63.8	63.5	53.6
Page-Blocks	1754	10	5	SVM L_1	94.2	92.3	90.6	89.3	88.9
				DecisionTree	95.8	95.8	96.4	95.8	93.3
				GreedyMiser	95.9	94.7	94.2	92.0	90.1
				RADIN	96.7	96.6	96.4	96.0	93.6
Magic	6315	10	2	SVM L_1	79.6	79.6	79.3	78.9	73.4
				DecisionTree	84.5	84.5	83.1	79.2	72.9
				GreedyMiser	85.6	85.7	82.6	82.1	73.5
				RADIN	86.9	86.9	86.4	82.4	73.5
White wine	1635	11	7	SVM L_1	53.7	53.3	52.8	52.3	44.8
				DecisionTree	51.8	51.8	52.7	52.0	45.2
				GreedyMiser	54.6	54.3	53.2	49.7	46.3
				RADIN	53.1	53.8	53.6	53.4	51.2
Red Wine	541	11	6	SVM L_1	55.9	55.5	54.3	53.7	53.7
				DecisionTree	58.1	58.1	58.1	58.1	54.0
				GreedyMiser	58.1	55.8	53.9	52.1	46.5
				RADIN	57.2	57.2	58.7	57.0	55.3
Adult	10708	14	2	SVM L_1	84.4	84.3	81.2	77.8	76.3
				DecisionTree	85.3	85.3	85.3	84.6	79.0
				GreedyMiser	86.0	86.0	85.9	84.8	78.0
				RADIN	85.3	85.3	84.9	84.4	82.1
Letter	6661	16	26	SVM L_1	48.3	33.0	23.6	14.2	08.6
				DecisionTree	82.3	82.3	82.3	48.4	10.2
				GreedyMiser	74.9	40.1	27.5	15.6	08.5
				RADIN	68.5	67.7	62.7	47.8	17.7
Pendigits	2460	16	10	SVM L_1	79.5	55.5	32.7	24.5	20.2
				DecisionTree	94.4	94.4	94.4	79.6	32.0
				GreedyMiser	85.8	67.8	64.9	37.5	21.1
				RADIN	98.6	97.6	95.1	80.7	43.0
Cardiotocography	685	21	10	SVM L_1	68.3	58.0	49.6	33.8	25.9
				DecisionTree	77.5	77.5	77.5	77.1	64.3
				GreedyMiser	82.7	81.8	75.1	48.0	34.3
				RADIN	80.2	80.2	80.2	79.6	66.2
Statlog	1444	36	3	SVM L_1	77.5	74.1	70.3	63.0	58.7
				DecisionTree	82.3	82.3	82.3	82.3	82.1
				GreedyMiser	85.1	84.6	83.1	76.5	60.5
				RADIN	85.9	85.8	85.8	85.2	83.3
Musk	2175	166	2	SVM L_1	95.0	95.0	94.2	92.1	86.5
				DecisionTree	94.2	94.2	94.2	94.2	94.2
				GreedyMiser	95.0	95.0	95.1	95.2	94.9
				RADIN	97.4	97.1	96.7	96.8	94.4
MNIST	9000	780	10	SVM L_1	89.7	89.7	88.2	70.4	57.7
				DecisionTree	80.8	80.8	80.8	80.8	80.8
				GreedyMiser	92.0	92.0	90.3	84.6	77.6
				RADIN	95.0	94.8	92.6	92.0	85.9

TABLE 4.1: Accuracy at different *cost* levels, here directly the amount (%) of features used. The accuracy is obtained through a linear interpolation on accuracy/cost curves. Highlighted results corresponds to the best performance obtained at each cost level. The same subset of train/validation/test data have been used for all models for each dataset.

Corpus Name	Nb. Ex	Nb. Feat	Model	Amount of features used (%)			
				25%	10%	5%	1%
gisette	6000	5000	SVM L_1	97.0	96.8	96.3	91.0
			DecisionTree	91.9	91.9	91.9	91.9
			GreedyMiser	88.4	88.4	86.7	78.5
			RADIN	95.7	95.7	95.7	94.7
r8	7674	6224	SVM L_1	96.9	96.8	95.1	91.3
			DecisionTree	90.1	90.1	90.1	90.1
			GreedyMiser	94.8	94.7	94.5	93.9
			RADIN	96.2	96.1	96.1	95.9
webkb	4162	5388	SVM L_1	89.1	88.7	85.9	71.7
			DecisionTree	79.3	79.3	79.3	79.3
			GreedyMiser	86.1	86.4	85.7	82.8
			RADIN	96.2	96.1	86.5	83.1

TABLE 4.2: Accuracy at different *cost* levels on datasets with a large number of features, specified as an amount (%) of features used. The accuracy is obtained through a linear interpolation on accuracy/cost curves. Highlighted results corresponds to the best performance obtained for each cost level.

4.4 Closing remarks

In this chapter, we proposed to study the more general problem of feature acquisition without making assumptions on the nature of the data, and focus on adaptive process to enhance the performance on the goal task at hand. We presented a model that encapsulates such adaptive acquisition mechanism in the prediction process. The model can be implemented as a recurrent neural network and allows different variations in implementation. One of its main new aspects w.r.t state of the arts is its ability to perform acquisition per block, and its differentiable nature. We showed that while the optimized loss is not the exact optimal loss one wants to achieve, the method performs well in various settings, on datasets of different nature, and is able to scale well to larger tasks.

However, it seems interesting to study now how to build a model that provides real binary values for acquisition, in order to get closer to the real optimal loss. Indeed, we optimized our approach following a loss that is not exactly corresponding to the real problem at hand. While our choices seemed to provide the same global effect regarding the budget acquisition constraint, we want to go further. We propose to study this in the next Chapter, where we propose to use a stochastic method relying on policy gradient technique to obtain binary weights.

Chapter 5

Adaptive cost-sensitive feature acquisition - Stochastic Approach

Abstract: We presented in the previous chapter a model for adaptive feature acquisition, relying on a recurrent neural network architecture. However, the loss we defined for our instantiation is a continuous relaxation of the "true" loss of the problem. We propose to study in this chapter how to design a model that fit closer the actual adaptive feature acquisition loss we defined in the first place. We present a stochastic framework, relying on policy gradient inspired techniques, which allows introducing real binary values for the acquisition weights. The performance of the method is illustrated on a variety of experiments, and a comparative study with the previous model is provided. This chapter is partially based on the work presented in [Contardo, Denoyer, and Artières, 2016b].

5.1 Introduction

In the previous chapter, we proposed a recurrent neural network-based approach to solve the problem of adaptive feature acquisition. We proposed a formulation of the generic loss in Equation 4.3 for the problem of sequential acquisition, that we rewrite here as a reminder:

$$\mathcal{L}(\theta, \beta_1, \ldots, \beta_T) = \Delta(d_\theta(x[\alpha_{1:T}]), y) + \lambda \sum_{i=1}^{n} |\sum_{t=1}^{T} \alpha_{t,i}| c_i \tag{5.1}$$

However, the instantiated model optimizes a continuous relaxation of the budget, where the cost of an acquired feature was counted each time the feature was acquired during the process:

$$\mathcal{L}(\theta, \beta_1, \ldots, \beta_T) = \Delta(d_\theta(x[\alpha_{1:T}]), y) + \lambda \sum_{t=1}^{T} \alpha_t^\intercal . c$$

As explain in Section 4.2.3, this corresponds to the true loss in the "ideal" case, if the system provides binary values for α_t, and if a feature can not be acquired more than

once (or if one considers that the cost should be counted as many times a feature is acquired during the process)[1]. However, our approach predicts continuous values and has no constraint on re-acquisition of features. We expected that the loss would have an overall similar effect on the acquisition constraint, and we indeed observed satisfying results on experiments. But we now want to go further on this task and study how to get closer to an "ideal" system, and if such system would provide better results.

We propose in this Chapter to consider the first steps to close this gap, which resides in removing the continuous relaxation on the α_t weights, thus extending our system to provide binary values for the acquisition process. It is actually not straightforward, as we also want to keep the properties we defined for the previous framework, in Section 4.2.2. Indeed, one of the aspects of our approach is to propose a completely differentiable method, allowing the use of efficient gradient descent optimization algorithms for learning. However, predicting binary values is not possible using differentiable functions.

To tackle this problem, we first propose to redefine the acquisition process as a stochastic sequential process. In the previous chapter, we had deterministic continuous weights, outputted by the acquisition function. We now consider that the acquisition function outputs probability distributions, which are used to sample the actual acquisition binary weights. With this change of formulation, the adaptive acquisition of features can be approached as a reinforcement learning problem, where the choice of features to acquire is made by a policy, and the final performance (quality of prediction) is the reward. From this observation, we present an adaptation of the previous architecture to a stochastic one, which can be learned using reinforcement learning algorithms, inspired from policy-gradient techniques, adapted to our loss.

We fist present a brief summary of the reinforcement learning problem and policy gradients in Section 5.2. Then we present the stochastic formulation of the adaptive acquisition and we define our stochastic framework in Section 5.3. Specific instantiations are given in Section 5.4 Experimental results, with a comparison between the previous approach and this one, are shown in Section 5.5.

5.2 Related work

5.2.1 Reinforcement learning

In the specific reinforcement learning domain, the learning process is defined with a different point of view than classical supervised learning. Let us first define the overall setting. We consider an *agent*, the system which has to solve a particular task.

[1] However this can be discussed depending on the application one considers.

This agent evolves and acts in an *environment*, where it can make different *actions*, which will change its environment (therefore the observation the system has of it). The choices of taken actions are driven by a *policy*, which will guide which action to take depending on the current *state* of the environment. Each action the agent takes also results in a potential change in the environment, therefore in an observation of this new state. The agent also receives after each action a *reward*, depending on the state, action and the overall task to solve. The reward can potentially happen only when the task is solved, thus actions can have a null reward but not be necessarily bad. For example, a mouse in a maze (see Figure 5.1) can choose different paths (left/right/forward), and its goal is to find the cheese (the reward), similarly to a robot aiming at a certain point on the map. But the environment could also be a video game, where at each step and action taken, the agent receives a positive feedback if it has destroyed an opponent. The general goal is thus to find a policy of actions that best solves the task, i.e that maximizes the total reward received. When the underlying rules of operation of the environments are known (i.e the transition probabilities between the states depending on the action taken, namely the Markov Decision Process formulation), the problem of finding the optimal policy of action can be formulated as an optimization or a planning problem. Reinforcement learning approaches are however used when these transition probabilities are unknown.

Various methods have been proposed, with different characteristics (see for instance [Sutton and Barto, 1998; Kaelbling, Littman, and Moore, 1996] for surveys). There is a first distinction, between *model-based* approaches (e.g [Kuvayev and Sutton, 1997; Littman and Szepesvári, 1996]), which aim at finding the underlying markov decision process before solving the optimal policy, vs *model-free* (e.g [Strehl et al., 2006]), which learns solely the policy. Methods are also usually differentiated between *value-based*, where the goal is to associate a value to each action, then implicitly deduce a policy, vs *policy-based*, where only a policy is learned, vs *actor-critic*, where both policy and action-value function are learned simultaneously.

We do not describe in further details all approaches here and propose to focus instead on policy-based methods, more specifically policy-gradient ones, which have nice convergence properties and will fit in our framework.

5.2.2 Policy Gradient

Policy gradient techniques are model-free approaches that have been first presented in [Williams, 1992; Gullapalli, 1992], and have been later extended to better handle RL tasks with continuous states, continuous or high dimensional actions, and non-Markovian environments (Wierstra et al., 2007). The general principle is to represent the policy as a parametric function and to update its parameters directly with an estimated gradient in the direction of the higher reward. We rely on [Wierstra et al., 2007] to describe in this section the formalism of policy gradient, and how it can be

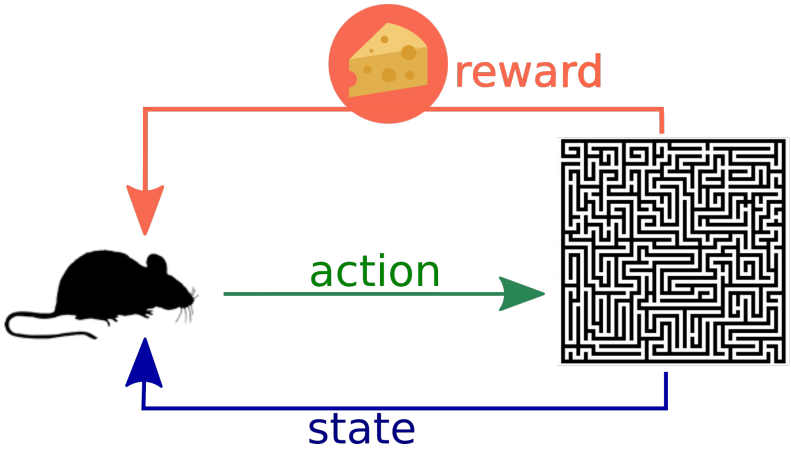


FIGURE 5.1: An example of a reinforcement learning environment. The agent (the mouse) takes actions (left, right, etc.) in an environment, here a labyrinth. Its goal is to find the cheese, the reward. Each time the mouse makes a decision, it perceives a change in the environment : a new state (or observation).

combined with a recurrent neural network as the parametric policy function, where each "step" of the RNN is a decision, an action taken by the agent. This will allow us in the next section to formalize our acquisition problem as a stochastic decision process sampled from our model output, and to use similar techniques as below to compute the gradient and learn.

Formalization of the problem

We first provide some notations and a more formal definition of the problem. We consider a set of states $\mathcal{S}$, a set of possible actions $\mathcal{A}$, a transition distribution $P(s_{t+1}|a_t)$ (unknown to the model). We note r_t the reward received by the agent at step t, which is dependent of the state s_t. x_t is the observation the agent gets of the environment at step t, associated to the state s_t. R_t is the *return* at step t, $R_t = \sum_{k=t}^{\infty} r_k \gamma^{t-k-1}$, where γ is a discount factor between 0 and 1. Finally, π is the policy defined by a probability distribution on the actions w.r.t. the observations.
The measure of quality of the policy is defined as the return's expectation at step 0, $J(\pi) = E_\pi[R_0] = E[\sum_{t=0}^{T-1} \gamma^t r_t]$. The goal of RL is to find an optimal policy w.r.t J.
As the agent interacts with the environment, it builds *episodes* : sequences of observations and actions, as well as reward. The *observed history* h_t is defined as $h_t =< x_0, a_0, x_1, a_1, \ldots, a_{t-1}, x_t >$, the sequence of observations and actions from the beginning of the episode up to step t. A stochastic policy π thus makes its decision of action w.r.t. the observed history h_t, where $\pi(a|h_t)$ outputs a probability distribution over the possible actions, from which a_t is sampled ($a_t \sim \pi(a|h_t)$).

Computing the gradient

Let us now define how the gradient for these parameters θ can be estimated, in order to maximize the expected reward.
$R(h)$ denotes a measure of the total reward gained during a trajectory. $P(h|\theta)$ is the probability of this trajectory to happen given the policy parameters θ (as the history depends on the actions taken by the agent). With this in mind, one can rewrite the measure of quality of the policies as the reward over all possible trajectories, weighted by the probability of the trajectory under π :

$$J = \int_h P(h|\theta)R(h)dh$$

Its gradient w.r.t θ can be written as follow using the likelihood ratio trick:

$$\nabla_\theta J = \int \nabla_\theta P(h)R(h)dh = \int \frac{P(h)}{P(h)}\nabla_\theta P(h)R(h)dh = \int p(h)\nabla_\theta \log P(h)R(h)dh$$

As, for a single fixed h, $\nabla_\theta R(h) = 0$, because the reward for a given trajectory do not depend on the policy parameters. Using Monte Carlo approximation of this expectation on M trajectories leads to:

$$\nabla_\theta J = E_h[\nabla_\theta \log P(h)R(h)] \approx \frac{1}{M}\sum_{n=1}^{M} \nabla_\theta \log P(h^n)R(h^n)$$

This expectation is still dependent on distributions unknown to the system within $P(h)$. However, the actions probabilities are known by the agent and one can decompose the probability of an history as :

$$P(h_T) = P(x_0)\Pi_{t=1}^{T} P(x_t|h_{t-1}, a_{t-1})\pi(a_{t-1}|h_{t-1})$$

i.e the product of all actions and observations probabilities given the observed history h. The log-derivative of $P(h_T)$ results in a sum where the only part dependent of θ is the actions probabilities. Thus, we can rewrite the above approximation as:

$$\nabla_\theta J \approx \frac{1}{M}\sum_{n=1}^{M}\sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|h_t^n)R_t^n$$

Where R_t^n is the return, defined above, as future actions do not depend on past rewards.

Using policy gradient with RNN

The authors of [Wierstra et al., 2007] motivate their use of RNN (specifically LSTM) with policy gradient algorithms by the *memory capacity* these models have. Indeed,

using these architectures would allow the system (i.e agent) to map *histories* to action probabilities through the internal state and recurrence of the network, instead of making a decision based on the current observation only.
More precisely, the RNN integrates each new observation by updating its internal state. Each output of the *Recurrent Policy Gradients* are interpreted as a probability distribution for the actions. At each step, the network receives an observation x_t and a reward r_{t-1}, updates its internal state (which could be interpreted as a "sufficient statistic" of the history $T(h_t)$), and outputs a distribution $P(a_t|T(h_t);\theta)$, where θ are the RNN parameters. Similarly as before, a_t is sampled following this distribution. The authors propose to use *eligibility back-propagation through time* (Williams, 1992) to learn the network. We inspire from this approach to build our network, however, we do not have reward but a differentiable loss.

5.3 Definition of the stochastic model

5.3.1 Cost-sensitive Learning problem with stochastic acquisition method

We consider as in the previous chapter, a sequence of acquisition vectors $(\alpha_1, \ldots, \alpha_T)$, for an acquisition process of T steps. Here the vectors α_t are binary. Let us denote this sequence α for simplicity of writing. We define $max(\alpha) \in \{0,1\}^n$ as the vector such that $max(\alpha)_i = \sum_{t=1}^{T} |\alpha_{t,i}|$, i.e the binary vector that corresponds to all features acquired through the entire process.

We propose now to adopt a stochastic probabilistic formalization of the process. Thus, we consider that each binary vectors α_t have been sampled from a **probability distribution**. We introduce π an **acquisition policy**, where $\pi(\alpha_t|\alpha_{t-1}, \ldots, \alpha_1, x)$ corresponds to this probability distribution, i.e the probability of acquiring the features, depending on all the previously acquired features observed on input x.

If we consider a generic function d that predicts an output $\hat{y}$ from a sequence of acquisition vectors α and an input x (which we denote for sake of clarity $d(x[\alpha])$ as in the previous chapter), the loss of this stochastic process can be rewritten as the expectation on α w.r.t π the acquisition policy of the (weighted) sum of the prediction error (Δ) and the acquisition cost (similarly as before):

$$\mathcal{J}(d,\pi) = \mathbb{E}_{P(x,y)}\left[\mathbb{E}_{\alpha\sim\pi(\alpha|x)}\left[\Delta(d(x[\alpha]),y) + \lambda max(\alpha)^{\intercal}.c\right]\right] \tag{5.2}$$

Where $\mathbb{E}_{\alpha\sim\pi(\alpha/x)}[.]$ stands for the expectation on the sequence of acquisition α given a particular input sample x and the acquisition policy induced by π. $max(\alpha)^{\intercal}.c$ is the overall cost of feature extraction and λ controls the trade-off between prediction quality and feature acquisition cost. $P(x,y)$ is the unknown underlying data distribution. Given a training set of ℓ samples $(x^1,y^1),...,(x^\ell,y^\ell)$, the loss function can be

approximated by the empirical loss $\mathcal{J}^{emp}(d,\pi)$ defined as:

$$\mathcal{J}^{emp}(d,\pi) = \frac{1}{\ell}\sum_{k=1}^{\ell} \mathbb{E}_{\alpha\sim\pi(\alpha|x^k)}\left[\Delta(d(x^k[\alpha]),y^k) + \lambda max(\alpha)^\intercal.c\right] \tag{5.3}$$

5.3.2 Gradient computation

Let us consider first generic parametric functions for both the prediction function and the acquisition policy, respectively denoted d_W and π_A. We consider for d_W any functions that take as input the sequence of acquired features on x w.r.t the given sequence of acquisition α. Typically, representation-learning based functions can be used, but other could also be proposed. We also consider any parametric function that predicts a sequence of probability distribution vectors of size n, n being the number of features in the input[2].

We explain below how the gradient of the empirical loss of Equation 5.3 can be estimated. Let us rewrite the empirical loss associated with a single training example (x,y), $\mathcal{J}^{emp}(x,y,A,W)$:

$$\mathcal{J}^{emp}(x,y,A,W) = \mathbb{E}_{\alpha\sim\pi_A(\alpha|x)}\left[\Delta(d_W(x,\alpha),y) + \lambda max(\alpha)^\intercal.c\right] \tag{5.4}$$

Even with differentiable functions Δ, π_A and d_W, the right term of the following loss is not derivable due to the max term, so we first propose to upper bound this loss, and to perform the gradient descent over this bound.

As the α_t are binary vector, the computation of the acquisition cost component of the loss can be bound by dropping the L_1 norm on the sum of α_t for all features i. The bound thus corresponds to a loss where the cost of acquisition is counted each time a feature i is acquired during the process, even if it has already been acquired before. We denote this loss $\hat{\mathcal{J}}$.

$$\begin{aligned}
\mathcal{J}^{emp}(x,y,A,W) &= \mathbb{E}_{\alpha\sim\pi_A(\alpha|x)}\left[\Delta(d_W(x,\alpha),y) + \lambda max(\alpha)^\intercal.c\right] \\
&= \mathbb{E}_{\alpha\sim\pi_A(\alpha|x)}\left[\Delta(d_W(x,\alpha),y) + \lambda\sum_{i=1}^{n}|\sum_{t=1}^{T}\alpha_{t,i}|.c_i\right] \\
&\leq \mathbb{E}_{\alpha\sim\pi_A(\alpha|x)}\left[\Delta(d_W(x,\alpha),y)\right] + \lambda\mathbb{E}_{\alpha\sim\pi_A(\alpha|x)}\left[\sum_{t=1}^{T}\alpha_t^\intercal.c\right]
\end{aligned} \tag{5.5}$$

The expected value of the sum of the binary vector α_t for all steps, w.r.t the acquisition policy $\pi_A(\alpha|x)$ can be expressed directly as the sum of the probabilities of sampling outputted by this policy π for all acquisition steps. We therefore rewrite

[2]We do not make any hypothesis regarding the sampling, we propose two approaches in this regard in Section 5.4.1

the above bounded loss as follow:

$$\begin{aligned}\hat{\mathcal{J}}(x,y,A,W) &= \mathbb{E}_{\alpha\sim\pi_A(\alpha|x)}\left[\Delta(d_W(x,\alpha),y)\right] + \lambda\sum_{t=1}^{T}(\mathbb{E}_{\alpha\sim\pi_A(\alpha|x)}\left[\alpha_t\right])^\intercal.c \\ &= \mathbb{E}_{\alpha\sim\pi_A(\alpha|x)}\left[\Delta(d_W(x,\alpha),y)\right] + \lambda\sum_{t=1}^{T}\sum_{i=1}^{n}\pi_A(\alpha_{t,i}=1|\alpha_1,\ldots,\alpha_{t-1},x).c_i\end{aligned} \tag{5.6}$$

where $\pi_A(a_{t,i}=1|x)$ is the probability of acquiring features i at time-step t.
We propose to learn by optimizing this bound $\hat{\mathcal{J}}$. We first decomposed its gradient $\nabla_{A,W}\hat{\mathcal{J}}(x,y,A,W)$ as follow:

$$\nabla_{A,W}\hat{\mathcal{J}}(x,y,A,W) = \nabla_{A,W}\underbrace{\mathbb{E}_{\alpha\sim\pi_A(\alpha|x)}\Delta(d_W(x,\alpha),y)}_{\mathcal{Q}(x,y,A,W)} + \lambda\nabla_{A,W}\underbrace{\sum_{t=1}^{T}\sum_{i=1}^{n}\pi_A(\alpha_{t,i}=1|x).c_i}_{\mathcal{C}(x,y,A,W)} \tag{5.7}$$

where $\mathcal{Q}(x,y,A,W)$ is the **prediction quality** component of the loss while $\mathcal{C}(x,y,A,W)$ is the cost of the **acquisition policy** component. Let us now explain how the gradient of these two terms are computed.

Computing $\nabla_{A,W}\mathcal{Q}(x,y,A,W)$: The gradient of the prediction quality term may be computed using policy-gradient inspired techniques (Wierstra et al., 2007; Mnih, Heess, and Graves, 2014) as presented in Section 5.2.2. However, here our "reward" is the prediction quality computed through Δ. In our case, it is dependent of the parameters of the policy. Using the likelihood ratio trick similarly as before, we can write the gradient as:

$$\begin{aligned}\nabla_{A,W}\mathcal{Q}(x,y,A,W) &= \int\nabla_{A,W}(\pi_A(\alpha|x))\Delta(d_W(x,\alpha),y)d\alpha + \int\pi_A(\alpha|x)\nabla_{A,W}\Delta(d_W(x,\alpha),y)d\alpha \\ &= \int\frac{\pi_A(\alpha|x)}{\pi_A(\alpha|x)}\nabla_{A,W}(\pi_A(\alpha|x))\Delta(d_W(x,\alpha),y)d\alpha + \int\pi_A(\alpha|x)\nabla_{A,W}\Delta(d_W(x,\alpha),y)d\alpha \\ &= \int\pi_A(\alpha|x)\nabla_{A,W}\log\pi_A(\alpha|x)\Delta(d_W(x,\alpha),y)d\alpha + \int\pi_A(\alpha|x)\nabla_{A,W}\Delta(d_W(x,\alpha),y)d\alpha\end{aligned} \tag{5.8}$$

Although computing the exact expectation above is not tractable, one can approximate it through Monte-Carlo sampling on M trajectories over α, yielding:

$$\begin{aligned}\nabla_{A,W}\mathcal{Q}(x,y,A,W) \approx \frac{1}{M}\sum_{m=1}^{M}[&\Delta(d_W(x[\alpha]),y)\nabla_{A,W}\log\pi_A(\alpha|x) \\ &+ \nabla_{A,W}(\Delta(d_W(x[\alpha]),y)]\end{aligned} \tag{5.9}$$

We detail now the sequential acquisition process and the shape of the policy $\pi_A(\alpha|x)$. Since our model performs a series of acquisition steps, the policy $\pi_A(a|x)$, which is

the probability of the series of acquisitions a on input x, may be decomposed as:

$$\pi_A(a|x) = \pi_A(\alpha_1, ..., \alpha_T|x) = \prod_{t=1}^{T} \pi_A(\alpha_t|\alpha_{t-1}, ...\alpha_1, x) \tag{5.10}$$

The probability distribution $\pi_A(\alpha_t|\alpha_{t-1}, ...\alpha_1, x)$ corresponds to the probability of acquiring a particular set of features at step t, given the input and all the previously acquired features. We may now rewrite the quantity $\mathcal{Q}(x, y, A, W)$ w.r.t these probabilities. Note that from Equation 5.10, the log-probability of α can be decomposed over the different time-steps in the following manner:

$$\log \pi_A(\alpha|x) = \sum_{t=1}^{T} \log \pi_A(\alpha_t|\alpha_1, ..., \alpha_{t-1}, x) \tag{5.11}$$

allowing us to rewrite $\nabla_{A,W}\mathcal{Q}(x, y, A, W)$ such that:

$$\begin{aligned}\nabla_{A,W}\mathcal{Q}(x, y, A, W) \approx \frac{1}{M}\sum_{m=1}^{M} [\Delta(d_W(x[\alpha]), y) \sum_{t=1}^{T} \nabla_{A,W} \log \pi_A(\alpha_t|\alpha_1, ..., \alpha_{t-1}, x) \\ + \nabla_{A,W}\Delta(d_W(x[\alpha]), y) \ ]\end{aligned} \tag{5.12}$$

Computation of $\nabla_{A,W}\mathcal{C}(x, y, A, W)$: The term $\mathcal{C}(x, y, A, W)$ can also be evaluated by using Monte-Carlo sampling:

$$\begin{aligned}\nabla_{A,W}\mathcal{C}(x, y, A, W) =& \nabla_{A,W} \sum_{t=1}^{T}\sum_{i=1}^{n} \pi_A(\alpha_{t,i} = 1|x).c_i \\ =& \nabla_{A,W} \sum_{t=1}^{T}\sum_{i=1}^{n} c_i \sum_{\alpha_1,...,\alpha_{t-1}} [\pi_A(\alpha_{t,i} = 1|\alpha_1, ..., \alpha_{t-1}, x)P(\alpha_1, ..., \alpha_{t-1})] \\ \approx& \frac{1}{M}\sum_{m=1}^{M}\sum_{t=1}^{T}\sum_{i=1}^{n} c_i \nabla_{A,W}\pi_A(\alpha_{t,i} = 1|\alpha_1, ..., \alpha_{t-1}, x)\end{aligned} \tag{5.13}$$

where α is sampled M times w.r.t $\pi_A(\alpha|x)$

Final loss gradient: Putting all together, the final gradient is computed as:

$$\begin{aligned}\nabla_{A,W}\hat{\mathcal{J}}(x, y, A, W) = \frac{1}{M}\sum_{m=1}^{M} \Bigg[\Delta(d_W(x[\alpha]), y) \sum_{t=1}^{T} \nabla_{A,W} \log \pi_A(\alpha_t|\alpha_1, ..., \alpha_{t-1}, x) \\ + \nabla_{A,W}(\Delta(d_W(x[\alpha]), y) + \lambda \sum_{t=1}^{T}\sum_{i=1}^{n} c_i \nabla_{A,W}\pi_A(\alpha_{t,i} = 1|\alpha_1, ..., \alpha_{t-1}, x)\Bigg]\end{aligned} \tag{5.14}$$

Algorithm 4 Inference algorithm for the REAMs models.

procedure INFERENCE(x, A, W, U, V, T) ▷ input and parameters
  Initialize z_0
  for $t = 1, T$ **do**
    α_t sampled following $h_A(z_{t-1})$.
    Information acquisition: Acquire each features x_i such that $\alpha_{t,i} = 1$.
    $z_t \leftarrow f_{U,V}(z_{t-1}, x[\alpha_t])$
  end for
  return $d_W(z_T)$
end procedure

Variance reduction : The estimate of the gradient that is computed above can have a high variance. Following Wierstra et al., 2007, we replace the loss $\Delta(d_W(x[\alpha]), y)$ by $\Delta - b$, where $b = \mathbb{E}_{x,\alpha,y}[\Delta(d_W(x[\alpha]), y)]$.

5.4 REpresentation-based Acquisition Models (REAMs)

We propose, in a similar fashion as the previous model, to use parametric functions for both the acquisition policy and the decision process. The goal of the representations here again will be to aggregate the acquired information through the acquisition steps and to guide the acquisition process and the final prediction. We thus define a parametric function π_A, which takes as input a representation z_t, and outputs the probability distribution $\pi_A(\alpha_{t+1}|z_t)$, which corresponds to the probability distribution $\pi(\alpha_{t+1}|\alpha_1, \ldots, \alpha_t, x)$ as z_t is expected to contain the necessary information of the previously acquired features. The acquisition policy is then implemented with a parameterized function $h_A : \mathbb{R}^N \rightarrow [0; 1]^n$:

$$\pi_A(\alpha_t|\alpha_1, ..., \alpha_{t-1}, x) = h_A(z_{t-1}) \tag{5.15}$$

where z_t can be computed through an aggregation function similar as the previous model, such as LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Cho et al., 2014b). The final representation z_T will be taken as input of the decision function d_W.

Using above choices, the gradient of the loss function is now:

$$\begin{aligned}\nabla_{A,W,U,V}\hat{\mathcal{J}}(x, y, A, W, U, V) = \frac{1}{M}\sum_{m=1}^{M}\Bigg[&\Delta(d_W(z_T), y)\sum_{t=1}^{T}\nabla_{A,W,U,V}\log f_A(z_t) \\ &+\nabla_{A,W,U,V}(\Delta(d_W(z_T), y) + \lambda\sum_{t=0}^{T-1}\sum_{i=1}^{n}\nabla_{A,W,U,V}h_{A,i}(z_t).c_i\Bigg] \\ &\text{with } z_t = f_{U,V}(z_{t-1}, x[\alpha_t]) \text{ and } \alpha_t \text{ sampled w.r.t } h_A(z_{t-1})\end{aligned} \tag{5.16}$$

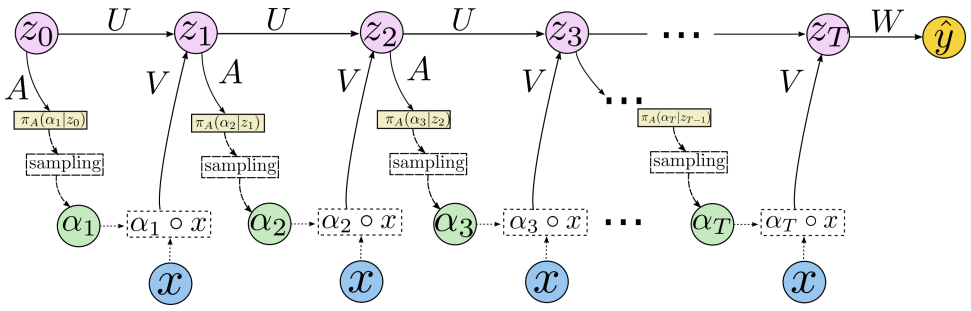


FIGURE 5.2: Representation-based Acquisition Model - Generic architecture of the framework using information aggregation through a latent representation, where $\circ$ is the Hadamard product. The acquisition process is done by sampling a binary vector α_t following distribution outputted by $\pi_A(\alpha_t|z_{t-1})$.

where $h_{A,i}$ is the i-th component of h_A. Note that one can learn a unique function h_A or one can learn a distinct function h_A for every step (i.e. with its own set of parameters A^t), which is what we did in our experiments, following the architecture shown in Figure 5.2 .
The optimization can be done by using (stochastic) gradient descent methods. More precisely, the different gradient terms can be computed using back-propagation techniques[3] as it is done with recurrent or deep neural networks.

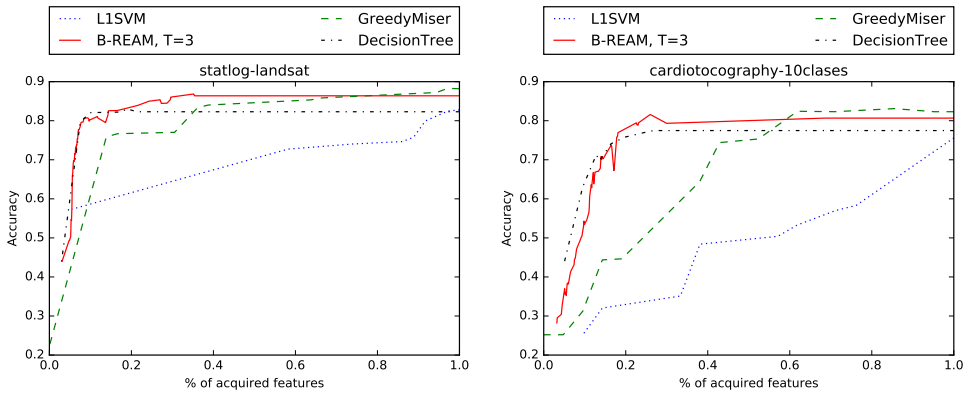
5.4.1 Instances of REAM

Different instances of the proposed framework can be described, depending on the choices of the h_A, $f_{U,V}$ and d_W functions. We propose to use non-linear functions for $f_{U,V}$, and linear functions for d_W(similar settings as the previous Chapter, see Section 4.3). Regarding the acquisition policy, we present below two expressions of the function h_A that will directly impact the way the acquisition is sampled and the features are acquired, thus resulting in three distinct models of acquisition:

1. **Multinomial Sampling Model (M-REAM):** In this model, α_t is sampled by following a multinomial distribution $h_A(z_t)$ such that $h_{A,i}(z_t)$ is the probability of sampling x_i and only x_i. It corresponds to a model that selects only one feature at each time-step. This model is not efficient when facing a large number of features but is close to Dulac-Arnold et al., 2012 and will serve as a reference model. This probability is typically obtained by using a softmax activation function in h_A.

2. **Bernoulli-based sampling model (B-REAM):** In this model, α_t is sampled by following a bernoulli distribution, i.e each component i of h_A corresponds to the probability of sampling feature x_i. In this model, multiple features can be

[3]Details on the back-propagation computations are not given here.



(A) Accuracy/Cost curves on *statlog*. (B) Accuracy/Cost curves on *cardio*.

FIGURE 5.3: Accuracy/Cost curves on two UCI datasets, comparing L1SVM, GreedyMiser and B-REAM with 3 acquisition steps. Abscissa is the percentage of acquired feature from 0% to 100% (complete inputs). Ordinate is the accuracy obtained.

sampled at each time-step. This probability is typically obtained by using a sigmoid activation function in h_A.

5.5 Experimental results

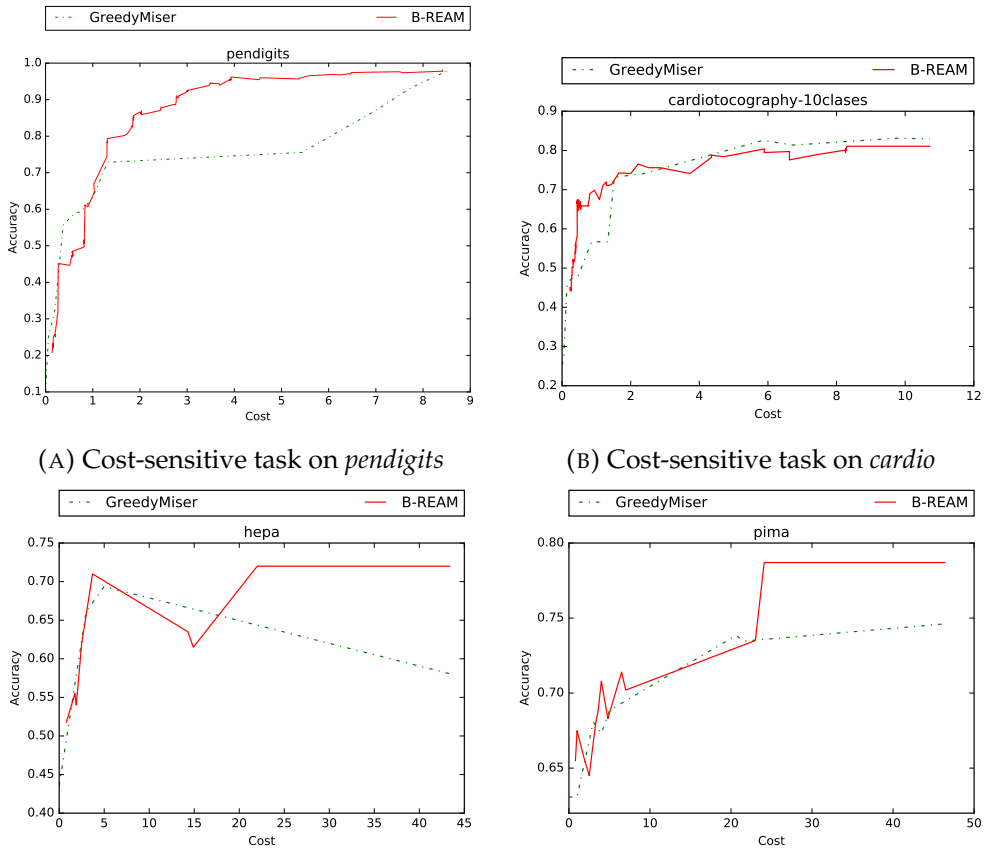
We present in this section a series of experiments on feature-selection problems and on a cost-sensitive setting, conducted on a variety of datasets on the mono-label classification problem.

Experimental protocol: To evaluate our stochastic Bernouilli-based acquisition model **B-REAM**, we propose a similar experimental protocol used in the previous chapter (see Section 4.3.1 for more details). The choice of functions for each component is the same as well as the loss used (RMSE).

5.5.1 Feature Selection Problem

In this setting, we consider that all the features have the same cost, i.e $\forall i, c_i = 1$. Therefore, we express the cost directly as the percentage of feature gathered regarding the total number of features. It thus corresponds to a problem of adaptive sparse classification.

The results obtained on different UCI datasets are summarized in Table 5.1 for various percentages amount of acquisition. Conjointly, Figure 5.3 presents the associated accuracy/cost curves on two of these datasets for better illustration. For example, on dataset *cardio* (Figure 5.3b), the model B-REAM learned with 3 steps of acquisition obtains an accuracy of approximately 70% for a cost of 0.2 (i.e acquiring 20% of the features on average), while GreedyMiser reaches 45% accuracy for the



(A) Cost-sensitive task on *pendigits*

(B) Cost-sensitive task on *cardio*

(C) Cost-sensitive task on *hepatitis*

(D) Cost-sensitive task on *pima*

FIGURE 5.4: Accuracy/Cost in the cost-sensitive setting. Top: Results on two UCI datasets, in Fig. 5.4a, 5.4b, artificially made cost-sensitive by defining the cost of a feature i as $c_i = \frac{i}{n}$, where n is the total number of features. Bottom: Results on two medical datasets, with real costs as given in Turney, 1995 for Fig. 5.4c, 5.4d.

same amount of features.

Overall, the results provided in Table 5.1 illustrate the competitiveness of our approaches in regard to the baselines. On average, B-REAM exhibits a high ability to adaptively select the "good" features and to simultaneously use the gathered information for prediction. Best results for each level of acquisition are highlighted in the Table, and second best results are in bold. One can see that B-REAM often have the best or second-best performance. It is however often outperformed by our continuous non-stochastic method RADIN. While B-REAM provides the higher accuracy 16 times, and second best 21 times, RADIN has respectively 31 best results and 16 second higher results, for a total of 5 levels of acquisitions on 12 datasets[4]. Comparatively, the closest baseline is GreedyMiser, with 10 best results and 13 second best. With this analysis, it seems that both of our approaches are more robust for a variety of problems in terms of the number of features, categories or examples.

Corpus Name	Nb. Ex	Nb. Feat	Nb. Cat	Model	Amount of features used (%)				
					90%	75%	50%	25%	10%
Abalone	1396	8	3	SVM L_1	64.3	63.2	60.0	56.7	50.5
				C4.5	60.5	60.5	60.5	61.3	56.1
				GreedyMiser	63.8	**63.6**	60.3	59.9	**54.9**
				B-REAM	62.9	62.7	**62.2**	**61.4**	51.9
				RADIN	**63.9**	64.0	63.8	63.5	53.6
Page-Blocks	1754	10	5	SVM L_1	94.2	92.3	90.6	89.3	88.9
				C4.5	95.8	95.8	96.4	**95.8**	**93.3**
				GreedyMiser	**95.9**	94.7	**94.2**	92.0	90.1
				B-REAM	96.7	96.7	96.4	95.6	91.6
				RADIN	96.7	**96.6**	96.4	96.0	93.6
Magic	6315	10	2	SVM L_1	79.6	79.6	79.3	78.9	**73.4**
				C4.5	84.5	84.5	83.1	79.2	72.9
				GreedyMiser	**85.6**	**85.7**	82.6	**82.1**	73.5
				B-REAM	86.9	86.9	86.5	81.8	**73.4**
				RADIN	86.9	86.9	**86.4**	82.4	73.5
White wine	1635	11	7	SVM L_1	53.7	53.3	52.8	52.3	44.8
				C4.5	51.8	51.8	52.7	52.0	45.2
				GreedyMiser	54.6	54.3	53.2	49.7	46.3
				B-REAM	**54.1**	**54.1**	54.1	**52.8**	**49.6**
				RADIN	53.1	53.8	**53.6**	53.4	51.2
Red Wine	541	11	6	SVM L_1	55.9	55.5	54.3	53.7	53.7
				C4.5	58.1	58.1	**58.1**	58.1	**54.0**
				GreedyMiser	58.1	55.8	53.9	52.1	46.5
				B-REAM	**57.6**	**57.4**	56.9	54.7	52.3
				RADIN	57.2	57.2	58.7	**57.0**	55.3
Adult	10708	14	2	SVM L_1	84.4	84.3	81.2	77.8	76.3
				C4.5	**85.3**	**85.3**	**85.3**	**84.6**	79.0
				GreedyMiser	86.0	86.0	85.9	84.8	78.0
				B-REAM	84.9	84.9	84.6	84.3	**81.6**
				RADIN	**85.3**	**85.3**	84.9	84.4	82.1
Letter	6661	16	26	SVM L_1	48.3	33.0	23.6	14.2	08.6
				C4.5	82.3	82.3	82.3	48.4	10.2
				GreedyMiser	**74.9**	40.1	27.5	15.6	08.5
				B-REAM	73.8	**69.5**	**66.0**	44.1	23.4
				RADIN	68.5	67.7	62.7	**47.8**	**17.7**
Pendigits	2460	16	10	SVM L_1	79.5	55.5	32.7	24.5	20.2
				C4.5	94.4	94.4	94.4	**79.6**	32.0
				GreedyMiser	85.8	67.8	64.9	37.5	21.1
				B-REAM	**97.5**	**96.3**	**94.8**	78.2	43.2
				RADIN	98.6	97.6	95.1	80.7	**43.0**
Cardiotocography	685	21	10	SVM L_1	68.3	58.0	49.6	33.8	25.9
				C4.5	77.5	77.5	77.5	77.1	**64.3**
				GreedyMiser	82.7	**81.8**	75.1	48.0	34.3
				B-REAM	**81.9**	81.9	80.7	80.9	64.1
				RADIN	80.2	80.2	**80.2**	**79.6**	66.2
Statlog	1105	60	3	SVM L_1	77.5	74.1	70.3	63.0	58.7
				C4.5	82.3	82.3	82.3	82.3	82.1
				GreedyMiser	85.1	84.6	83.1	76.5	60.5
				B-REAM	86.0	85.9	86.0	**83.9**	**82.9**
				RADIN	**85.9**	**85.8**	**85.8**	85.2	83.3
Musk	2175	166	2	SVM L_1	95.0	95.0	94.2	92.1	86.5
				C4.5	94.2	94.2	94.2	94.2	94.2
				GreedyMiser	95.0	95.0	95.1	95.2	94.9
				B-REAM	**96.8**	**96.9**	97.0	**96.3**	93.6
				RADIN	97.4	97.1	**96.7**	96.8	**94.4**
MNIST	62000	780	10	SVM L_1	89.7	89.7	88.2	70.4	57.7
				C4.5	80.8	80.8	80.8	80.8	**80.8**
				GreedyMiser	**92.0**	**92.0**	**90.3**	**84.6**	77.6
				B-REAM	86.4	83.8	82.8	81.1	78.7
				RADIN	95.0	94.8	92.6	92.0	85.9

TABLE 5.1: Accuracy at different *cost* levels, here the amount (%) of features used. The accuracy is obtained through a linear interpolation on accuracy/cost curves. Highlighted results corresponds to the best performance obtained at each cost level, bold results to the second best performance. The same subset of train/validation/test data have been used for all models for each dataset. Acquiring 25% of the features is equivalent for these datasets to using from 2 features (on *abalone*) to 195 features (on *MNIST*).

5.5.2 Cost-sensitive setting

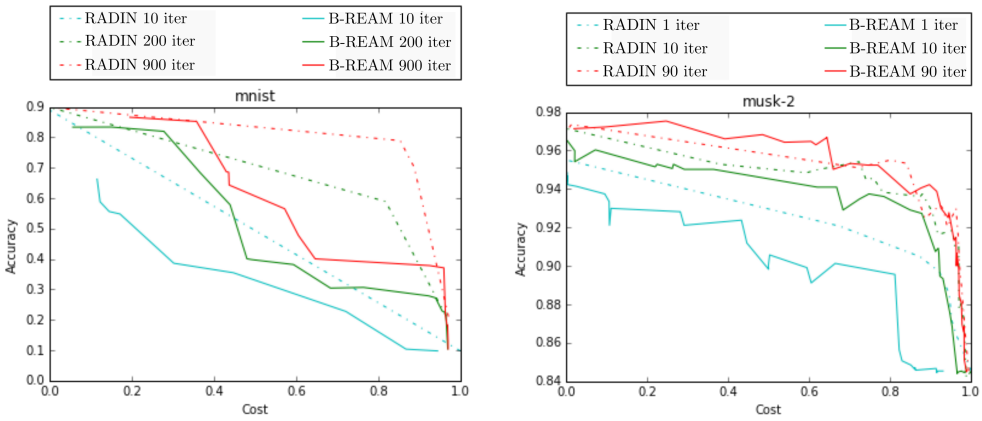
This section focuses on the *cost-sensitive* setting, where each feature is associated with a particular cost. We propose to study the ability of our approach to tackle such problems on two artificially generated cost-sensitive datasets (from UCI) and on two cost-sensitive datasets of the literature Turney, 1995. Figure 5.4 illustrates the performance on these 4 different datasets. The X-axis corresponds to the acquisition cost which is the sum of the costs of the acquired features during inference on the test set. On the 4 datasets, one can see that our B-REAM approach obtain similar results or outperforms GreedyMiser (to which we compare our work since it has been designed for cost-sensitive feature acquisition as well). We can observe an interesting behaviour on the two real medical datasets: there exist cost thresholds to reach a given level of accuracy (e.g Figure 5.4d, when $cost \approx 23$, or Figure 5.4c when $cost \approx 14$). This phenomenon is due to the presence of expensive features that clearly bring relevant information. A similar behavior is observed with GreedyMiser and with B-REAM, but the latter seems agiler and able to better benefit from relevant expensive features[5]. We suppose that this is due to the use of reinforcement-learning inspired learning techniques which are able to optimize a long-term target i.e the cumulative sum of the costs over an acquisition trajectory.

5.5.3 Comparison of the learning complexity

The major difference between B-REAM and RADIN is the stochasticity, which implies a sampling phase for B-REAM. This can yield slower learning since it requires many more iterations to converge. We illustrate complexity issues in Figure 5.5, where we compare different accuracy-cost curves resulting in experiments with a varying number of iterations setting through convergence. One can see that RADIN obtains better performance with fewer iterations, indicating a faster convergence rate. Note that the time spent in one iteration of RADIN and B-REAM is almost the same, the only difference being that B-REAM as an additional sampling step over the Bernoulli distribution. While B-REAM optimizes a loss closer to our real problem, this observation coupled with the results shown earlier indicate a strong advantage towards the relaxed continuous version RADIN, more particularly for the uniform cost case.

[4]Note that there are ties in the results thus not summing necessarily at 60

[5]Due to the small size of the real-world datasets (*hepa* and *pima*) the performance curve is not monotonous. Actually, the difference between the Pareto front on the validation set and the resulting performance on the test set suffers from a "high" variance. Moreover, this variance cannot be reduced by averaging over different runs because resulting accuracy/cost curves are composed of points at different cost/accuracy levels and cannot be matched easily. Yet these curves show significant trends in our opinion.



(A) Comparison of convergence rate on dataset *MNIST*.

(B) Comparison of convergence rate on dataset *musk-2*

FIGURE 5.5: Accuracy/cost curves for RADIN and B-REAM at different learning stages (i.e number of iterations). Both models have 3 acquisitions teps. The cost is uniform thus represents the average quantity of features used.

5.6 Closing remarks

We proposed in this chapter a model to get closer to the loss we define for the feature acquisition problem. We presented a stochastic adaptation of the previous architecture, which relies on policy gradient techniques in order to learn its weights. We illustrated on the same datasets that this model performs well, however it seems less robust than RADIN. This can be explained by the fact that B-REAM is more "expensive" to train, due to the stochasticity and the supervision, which require sampling a lot of trajectories in order to learn properly. However, B-REAM should be more resilient for problems with non-uniform costs as it is integrated more efficiently, compared to RADIN.

We have studied in these two chapters how to sequentially acquire features on a static input. At each step of acquisition, a single feature remains the same, during all the process. However, the problem of features acquisition can also be defined on sequential inputs, for example videos (i.e streams of images). One can think of the task of tracking objects in videos. In such case, the system has to acquire a subpart of the features (pixels) where the object is, at each step. The value of a feature can vary through time (as the object moves for instance), and the system has to adapt its acquisition. Our model could be easily used for such problem with a fully recurrent architecture.

We now focus on a completely different setting to consider the cost of labels acquisition during inference.

Chapter 6

Meta Active Learning

Abstract: This chapter presents our recent work on **meta-active learning**. We consider the problem of costly labels, usually tackled in the literature using active-learning techniques. These approaches provide strategies to choose the examples to label before training. We design a model in order to *learn an active-learning strategy*, based on a meta-learning setting. We propose different instantiations of the method. Experiments on artificial and real datasets are provided and show encouraging results.

6.1 Introduction

We have focused on the previous chapters on approaches for specific problems of budgeted learning: static and adaptive feature acquisition, to reduce the cost of features actually used for prediction.
As mentioned in Chapter 2, the field of *budgeted learning* covers different aspects, be it where the cost appears (features, labels) or when (test-time, training). We now propose to focus on the labels' cost, in a particular setting we call *meta-active learning* that we present now.

Machine learning, and more specifically deep learning techniques, are now recognized for their ability to give very good results on a variety of problems, from image recognition to natural language processing. However, most of the tasks tackled for now are supervised and need a lot of labeled data to be learned properly. These labeled examples are often expensive to get (e.g manual annotation), and not always available in large quantity. Moreover, one can observe that humans, on the other hand, seem able to learn and generalize well from only a few examples (e.g children can recognize rapidly any depiction of a car or some animals -drawing, photo, real life- after having been shown only a few pictures with explicit "supervision"). These arguments motivate the need for methods able to learn from a very small supervised training set.

This problem has been studied in the literature as *one-shot* (or few-shots) learning. This setting was first proposed in [Yip and Sussman, 1997], and it knows a renewal of interest under slightly different flavors. Recently, several methods have been presented, relying on different techniques such as Siamese networks (Koch, 2015) or memory-networks (Santoro et al., 2016) (see Section 2.2.3 for details). These methods do not learn a single "one-shot problem" solely based on the few labeled examples, but instead, they use additional data. They rely on supervised or unsupervised data of similar nature (e.g all data are images) but on categories that are different than the final problem. For instance, the model has access to images of cats, dogs, planes and houses during training, but is evaluated on elephants and bicycles problems. The goal is to use the available information of the supplementary data to build models that can "transfer" this knowledge when confronted with a one-shot task. Some approaches rely on unsupervised learning, but other methods design a training protocol that "mimics" one-shot learning problems. The key idea is to learn from problems that are close to the final task. In this original setting, a training point is not a single input example anymore, but a whole (small) dataset, i.e a problem. After learning, the system is evaluated on similar problems but on categories unseen before.

In parallel, the field of *active learning* focuses on approaches that allow a model to ask an oracle for the label of some training examples, to improve its learning. In this case, different settings can be defined, regarding the nature of the unsupervised examples set (a finite dataset completely observable or a stream of inputs), and the nature of the acquisition process (single step or sequential). We provide more details on this regard in Chapter 2.2.1. The goal in this setting is to define how to select the best examples to label, in order to obtain good performances. For instance, one can define some criterion, e.g information gain, computed for each example, and choose the more "useful" ones based on this criterion. Note that in this setting, the labels cost is considered during training: one aims at limiting the number of labeled examples for a single training problem. Therefore, the strategy to choose which example to label is not learned in classical active approaches.

We propose to study a problem at the crossroad of one-shot learning and active learning. We consider that developing methods able to learn from a small amount of labeled information is crucial. We present a method that not only learns to classify examples (of unseen categories) using small supervision but additionally learns an acquisition strategy of the examples to label. Our underlying goal is thus to study how an acquisition strategy can impact the performance of the system.

In Section 6.2, we define more formally the problem, and the different possible settings (similar to active learning), which need a specific training strategy as in one-shot learning. We then describe our approach in Section 6.3, which considers

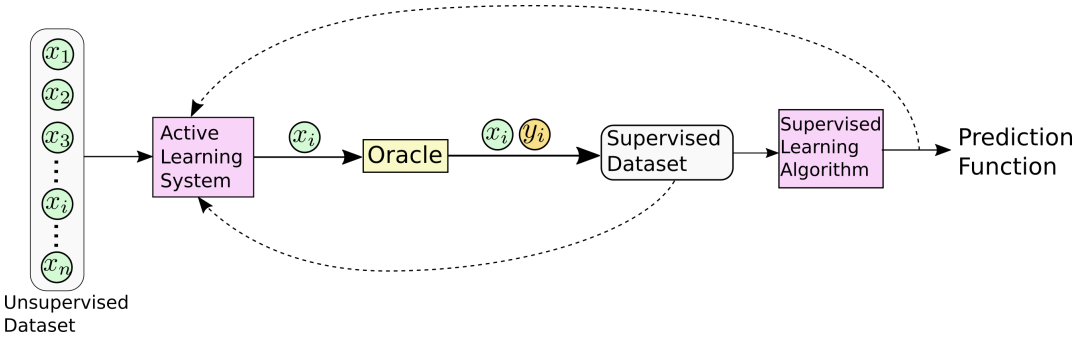


FIGURE 6.1: Graphic representation of the active learning process as illustrated in [Collet, 2016], from unsupervised data, acquisition of labels, learning of model and final prediction model.

the unsupervised dataset as a whole (non-myopic method) and has a static acquisition strategy, i.e one subset of examples is selected to be labeled in a single step decision. Section 6.4 provides preliminary experimental results. Finally, we present in Section 6.5 insights for a possible extension of this work for future investigation, with a "hybrid" model to go towards adaptiveness in a "non-myopic" setting.

6.2 Definition of setting

We define in this section the setting we propose, **meta-active learning**, in order to tackle active-learning coupled with few-shot learning, in a meta-training fashion. We first describe briefly the generic scheme of an active-learning system, and its two main settings. Then, we provide further details on the meta-training strategy, used recently for one-shot learning, that we will rely on to learn a "meta-active" system.

Active learning setting

The global process of an active-learning system is illustrated in Figure 6.1. In the most generic form, the system has access to data-points (i.e examples), and to an oracle. It can choose to ask the oracle the corresponding labels of some data-points. The system then uses this supervised sub-dataset to train its prediction model. This general formulation can encapsulate various situations. As mentioned in Chapter 2.2.1, the literature distinguishes two cases for the "nature" of the unsupervised dataset: (i) **pool-based** dataset, where the system receives a dataset of N examples at the beginning of the process, and where all examples are observable. (ii) **stream-based** dataset, where the system receives an input, one after the other, and has to decide at each step if its label is necessary. This leads to approaches of different natures, as the stream-based setting requires intrinsically sequential models, while pool-based methods can make their decision in a single shot. Another major difference is the available information. A pool-based method has access to all unlabeled

FIGURE 6.2: Examples of a complete dataset for a meta-active learning strategy, with a set of *training problems* $\mathcal{S}$, with P categories per problem, on a total of $|\mathcal{P}_{train}|$ classes, and a set of *testing problems* on distinct categories. Each problem is composed of a set of N examples that can be labeled and used for prediction, and a set of M examples to classify.

data, which provide a "global" view of the dataset, thus the problem at hand. On the other hand, a stream-based model has a "myopic" view as the future examples are unseen, but the problem to solve (asking or not for the label of a single example) seems easier. Additionally, this type of model can potentially rely on the feedback obtained through the oracle on all the previous steps. It can also choose to update the current prediction model and use its performances to drive the acquisition (e.g using uncertainty measures on classification on examples).

Meta-learning strategy inspired from one-shot learning

We propose to use a protocol similar to what has been recently presented for one-shot learning problems, e.g in [Santoro et al., 2016]. It aims at extending the basic principle of training in ML, where a model is trained on data-points from a similar distribution to the data-points observed during inference. For one-shot learning, it

resumes as designing data-points as one-shot problems, on dataset of similar nature (e.g all inputs are images). The protocol therefore replicates the final task during training and aims at *learning to learn from few examples*. We propose to use a similar protocol to learn an active-learning strategy in a meta-learning fashion.
We denote $\mathcal{S}$ a set of supervised problems (i.e datasets), where each problem $\mathcal{S}_i$ is composed of a set of inputs $x_j^i \in \mathbb{R}^K, , j \in \{1, N\}$, where N is the number of supervised examples for the problem[1]. Their respective labels y_j^i, are in a set of categories $\mathcal{P}_i$ of size P. For each training problem i, there is an additional set of examples that the model has to classify, denoted $\mathcal{S}_i^{test}$. The total set of categories in $\mathcal{S}$ is denoted $\mathcal{P}_{train} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \ldots \mathcal{P}_{|\mathcal{S}|}$. Similarly, we define the testing dataset $\mathcal{Q}$ as a set of problems, where each problem has N examples that can be labeled and M examples that has to be classified. All categories in the testing problems $\mathcal{P}_{test}$ are unseen in $\mathcal{P}_{train}$ (i.e $\mathcal{P}_{test} \cap \mathcal{P}_{train} = \emptyset$). An example of such dataset is illustrated in Figure 6.2 with image classification. All problems are binary classification (i.e P=2). The training problems contains categories such as *cats*, *dogs*, *houses* and *bicycles*, and the testing problems contains categories such as *elephants*, *cars*, *cakes* and *planes*.
We resume the generic learning scheme in pseudo-code in Algorithm 5. We consider that the system has an active-learning component, which controls the acquisition of labels, and a prediction component, that classifies inputs based on few labeled examples. Note however that these two components can be tied together. During training, the process iteratively picks a random problem i. The acquisition model receives only the examples in $\mathcal{S}_i$ (without labels)[2] and predicts which examples should be labeled. Using these labeled inputs, it provides predictions on the examples of the prediction dataset, $\mathcal{S}_i^{test}$, using the prediction module. It evaluates its performance (in a supervised fashion) and updates accordingly. This algorithm is broad and still encapsulate the different settings described above, i.e pool-based and stream-based, as well as static and sequential methods for label acquisition.
During testing, the process is similar : the system receives a dataset of unsupervised examples $\mathcal{Q}_i$, and acquire labels for some of them. Based on this small supervised dataset, it provides predictions on examples of $\mathcal{Q}_i^{test}$.

6.3 One-step acquisition model

We present in this section a strategy to design systems that **learn how to actively learn**, on a **pool-based** dataset. When a new problem is observed by the system, we assume one has access to all the (unsupervised) examples of the dataset. In this preliminary study, we propose to follow a **static acquisition** strategy for the labeling phase : the system will ask for the labels of a single subset of examples, in a unique step. Different possible instanciations are provided as well as experiments

[1] Note that each problem could have a different number of examples.
[2] As our goal is to simulate an active-learning process with no preliminary labeling.

Algorithm 5 Learning algorithm for meta-active learning algorithms.

Require: $\mathcal{S}$: Set of supervised problems, where $S_i = \{(x_1^i, y_1^i), \ldots, (x_N^i, y_N^i)\}$, $S_i^{test} = \{(x_1^{i,test}, y_1^{i,test}), \ldots, (x_M^{i,test}, y_M^{i,test})\}$, and $y_j^i, y_{j'}^{i,test} \in \mathcal{P}_i, \forall j \in \{1, N\}, j' \in \{1, M\}$
Require: A = Active-learning model
Require: M = Prediction model
1: **repeat**
2: Pick a random problem i
3: A predicts $\mathcal{D}_l^i$ the subset of examples to label in $\mathcal{S}_i$
4: Feed M with labeled examples $\mathcal{D}_l^i$.
5: Evaluate error of M on predictions of all $x_j \in \mathcal{S}_i^{test}$
6: Update A and M accordingly.
7: **until** stopping criterion

that show the relevance of this first "proof of concept" model. We present in Section 6.5 insights to extend this approach into an adaptive acquisition process.
We first define in a generic fashion the different components needed in a model to tackle this setting. We then present specific architectures that can be learned with classical ML techniques.

Components

Generically speaking, any model apprehending static dataset and one-step acquisition needs the following components:

- **Labels acquisition component**: this module should take as input the whole dataset of the current problem at hand, thus a set of examples. Its output is a binary vector of the size of the set, which guides the acquisition : if the j-th value of the output is 1, the label of the example x_j is asked to an oracle.
- **Prediction component**: this module takes a (new) example as input and outputs a prediction (e.g a category) based on the labeled examples obtained through the labels acquisition component. Note that this model is not necessarily parametric, nor needs retraining. In this first study, we propose to use a nearest neighbours based technique, using a similarity measure on top of representation learning.

As mentioned, we introduce an additional component useful in our case, a **representation component**. This module takes as input an example in $\mathbb{R}^K$ and output its representation in a latent space $\mathbb{R}^L$. The above modules can each potentially work on the representation visualization of the example, or the original raw input. We illustrate this in Figure 6.3, where the generic flow of the framework during inference is depicted, including the representation module. Note that this module can be used before acquisition, during prediction, or both.

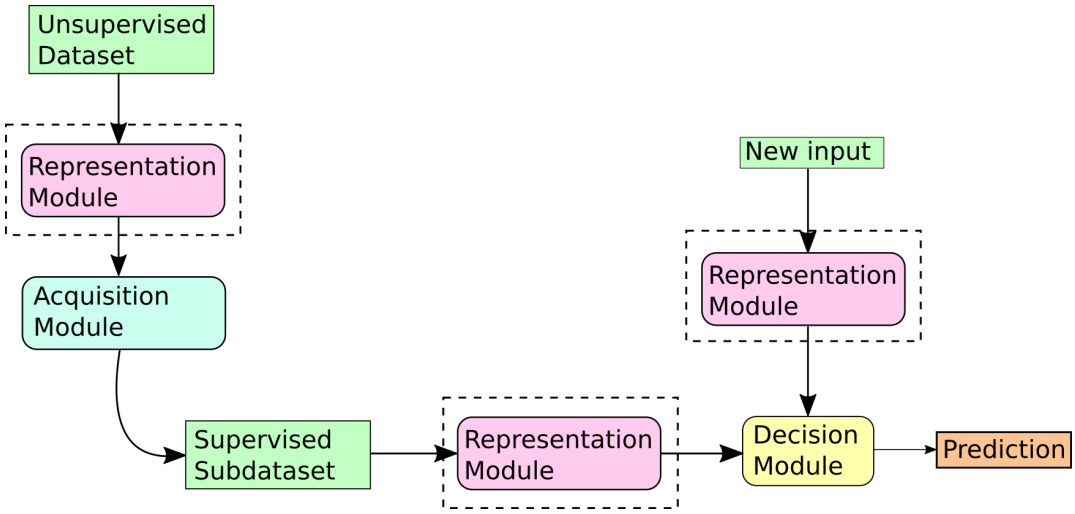


FIGURE 6.3: Generic architecture of the framework, with three components : acquisition module that receives an unsupervised dataset, in its original space or in the representation space, through the representation module, and decision module which predicts on a new input, in its original space or in the representation space, based on a labeled sub-part of the dataset.

Optimization criterion

The loss can be expressed as a trade-off between the performance of the model (the sum of all error Δ for all examples to classify, where Δ measures the error between the model prediction and the expected output) and the labeling cost that led to such performance, i.e the size of the labeled subset. We note here f_a the acquisition component, the function predicting the subset of examples to label ($\mathcal{D}_i^l$) from the unsupervised training set $\mathcal{S}_i$. The prediction component is noted as a function d which takes an input x_i^j, and the labeled dataset, to output a label $\hat{y}_i^j$. The optimization criterion resumes to:

$$\begin{aligned} \mathcal{D}_i^l &= f_a(\mathcal{S}_i) \\ \mathcal{L} &= \sum_{i \in \mathcal{S}} [\sum_{j \in \mathcal{S}_i^{test}} [\Delta(d(x_j^i, \mathcal{D}_i^l), y_j^i)] + \lambda |\mathcal{D}_i^l|] \end{aligned} \tag{6.1}$$

Choice of architecture and implementations of components

Let us first focus on the **label acquisition component**. As mentioned above, this module takes a set of vectors (raw examples or representations) as input, and outputs an acquisition vector. Ideally, the module should rely on the fact that the input is a set. We propose to use recurrent neural networks, which were initially proposed to consider sequences of inputs. More specifically, we propose in this work to use **bi-directional GRU**, which ensure that the output i of the network is computed with regards to all inputs examples (see Chapter 2.3.2 for more details). It would also be relevant to use attentional-LSTM, presented in [Vinyals, Bengio, and Kudlur, 2015] (described in Chapter 2.3.2), as it provides an order-invariant network, but this has not been tested yet in our experiments. The labels acquisition module

is also expected to return a binary vector of acquisition. This type of outputs is not trivial to produce in neural networks as it is non differentiable. We propose two solutions :

- **Policy gradient** (Wierstra et al., 2007, see Chapter 5.2.2 for details): we propose to obtain the binary acquisition vector through sampling from a continuous probability distribution outputted e.g by a bidirectional RNN. By using policy gradient techniques and Monte Carlo sampling, such approaches can be used efficiently in neural networks, as they become differentiable.
- **Continuous relaxation** It is possible to use activation functions that provide true null outputs and positive values of any range (e.g ReLU). The outputted vector of size N (where N is the number of unsupervised examples in the dataset i) can be considered as the weights to apply on each examples for prediction, as was done in Chapter 4.

The **prediction component** could be any prediction algorithm, parametric or not, which requires learning or not. In our case, the component should be able to back-propagate some gradients of errors to drive the overall learning. As we mentionned before, we propose to use similarity based prediction, and to provide more prediction ability the system conjointly learns a representation model for the inputs. The similarities are computed on these representations. We test two similarity measures, a normalized cosine similarity and an euclidean-based similarity. Additionally, computing the predicted label for a new input is done as follow : (i) each similarity with the supervised examples is computed. (ii) This vector of similarities is then converted into a probability distribution, using a softmax with temperature. (iii) The predicted label is computed as the sum of one-hot-vector labels of supervised examples weighted by this distribution. Note that when the temperature is high enough, this distribution is a one-hot vector, which is similar to a 1-nearest neighbour technique.

6.4 Experiments

We first describe our experimental protocol and the baselines we used. Then we study how our model behave on artificial gaussian datasets. Finally, we sum up results on different datasets that illustrate the relevance of our approach and show encouraging results.

Experimental Protocol

We propose to work on datasets with a sufficient number of classes, which allow to have a common domain for all inputs and to create different sub-problems. Before creating our dataset, we set P, the number of categories of each problem (common

for all problems[3]), N the number of examples in the dataset that can be labeled and M the number of examples to classify on (N and M are also common to all problems). The generation of the complete dataset is done as follow:

- **training dataset**: we select some "training classes" (e.g 50% of all classes) and their corresponding examples. We generate a large amount of sub-problems, by randomly selecting P categories in the "training classes" for each problem, and randomly selecting N examples in these P categories, for the set ox examples "that can be labeled", and additionnal M examples to evaluate the predictions.
- **validation and testing datasets** are generated similarily, on distincts "validation classes" and "testing classes", unobserved in the complete training dataset.

Baselines

As this problem has not been studied before to the best of our knowledge, we propose for this preliminary study two baselines. These baselines follow the same global scheme, but with a different acquisition component:

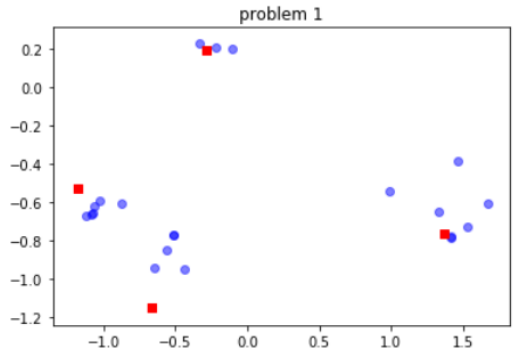
- **Random** acquisition: the examples to label are chosen randomly in the dataset.
- **K-medoids** acquisition: the examples to label are selected following a k-medoid clustering technique, where we label each example if it is a centroid of a cluster.

Note that both acquisition methods do not learn during the process, only the representation component (if one is used) is learned.
Note also that classical active-learning for pool-based static acquisition should be tested and will be studied in future works. We however expect the k-medoids baseline to be a reasonable and efficient baseline in such setting.

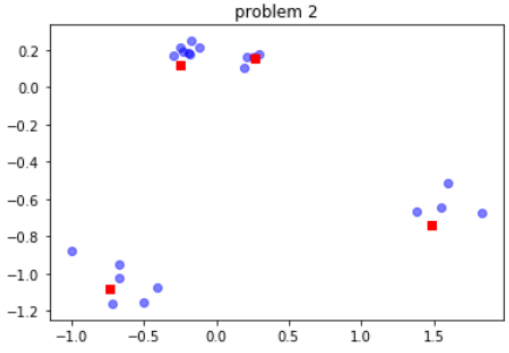
Results on artificial gaussian datasets

We propose to study first the behaviour of our method on artificial datasets. We generated a dataset of 50 categories, where examples of each category are sampled on a gaussian with specific mean and variance. We generated 300 examples per class. We selected 13 categories for training, 6 for validation, and the remaining for testing. From these 13 classes (resp. 6 and 31) we generated 3000 problems (1000 for testing and validation), where we randomly pick 4 categories , and 25 examples (total) for these categories. As we mentioned, we have for each problem an additional dataset of 25 examples to classify. The goal of these experiments is to provide an insight about the relevance of our approach, on 2D datasets to visualize easily

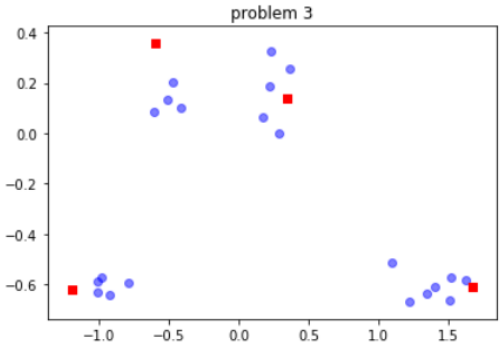
[3]Note that future works should include a study on the robustness of the approaches regarding the change of the number of categories in the set of sub-problems, e.g the ability of detecting if a problem has 3, 5 or 10 classes.



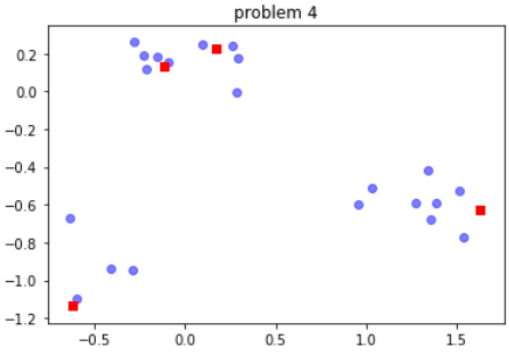
(A) Problem 1



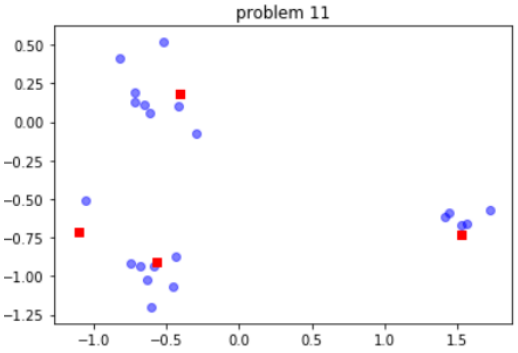
(B) Problem 2



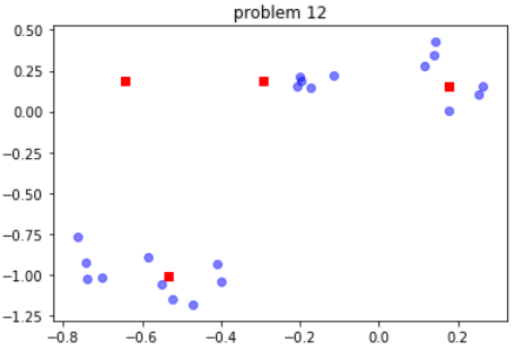
(C) Problem 3



(D) Problem 4



(E) Problem 11

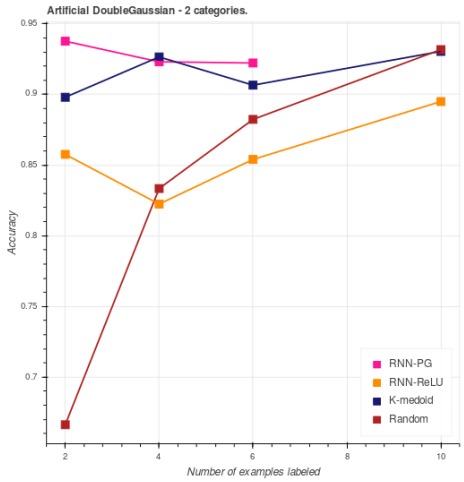


(F) Problem 12

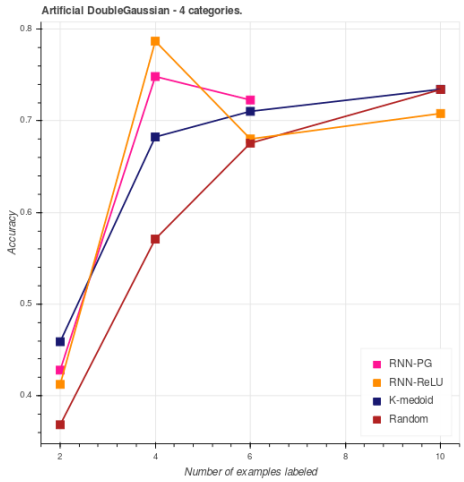
FIGURE 6.4: Results on artificial datasets with 4 classes sampled from 4 distinct gaussian distributions. Each figure plots the data-points of a problem (in blue). The examples plotter in red squares are the examples selected by our model at the end of the learning process. Note that these are test-problems.

the labeled examples. We provide a qualitative study of the acquisition behaviour of our method in Figure 6.4. The model uses a soft-similarity based on euclidean distance, a policy-gradient based acquisition, and a representation module with a linear layer and an hyperbolic tangent as activation function, with a representation space of size 5. These plots show the examples of 6 problems with 4 classes, during testing (not seen during training). One can see that the number of examples per categories can vary a lot, as well as the "separation" between categories from one problem to another, which makes some problems more "difficult". For each plot, the examples asked to label by the model after training are depicted in red squares. The "budget" is 4 examples, which matches the number of categories for each problem. Before learning, the acquisition is mostly random, thus it does not find any structure in the data and miss some categories, which prevent good classification results. After learning, it appears clearly that the model can detect each "cluster" of points, and manages to acquire one label in each cluster, even when the cluster has few examples (e.g Figure 6.4e with a category with only 2 examples). This indicates the ability of our approach to find an underlying structure in the datasets that should guide its acquisition process.

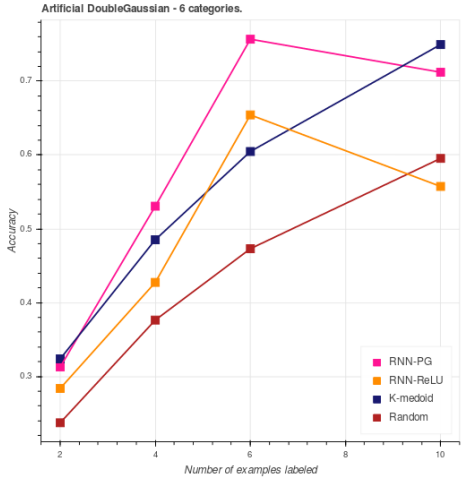
We designed a more complex dataset, which is generated in a similar fashion, however each point of a category can be drawn from 2 distinct gaussian. We created datasets with different P values (number of categories in each problem), where P can be equal to 2, 4 or 6, specified for each experiment below. Figure 6.5 resumes the results obtained by our models and the two baselines for the three types of problems. For each "budget" (i.e number of examples labeled), we select for each model the best results on validation problems and plot the corresponding performance on test problems (results plotted in solid square). Baselines have the same grid-search parameters as our models w.r.t. the representation module (architecture and size), the decision module (similarities measures, temperature parameter, type of similarity), and the learning rate. The additional hyper-parameter for our models is the size of the internal representations of the bi-directional RNN.
Results of the model using policy-gradient technique for acquisition are in pink (**RNN-PG**), in yellow for the continuous relaxation using Rectified Linear Unit (**RNN-ReLU**), in blue for k-medoid, and in red for random acquisition. We can see that our RNN methods perform well, especially RNN-PG, for all types of problems and budget. More particularly, for each budget that matches the number of categories, RNN-PG performs better than k-medoid : about 5% better for binary and 4-classes problems (with budget 2 and 4), 15% better for 6-classes problem with budget 6. However, RNN-ReLU does not provide consistent performances on the three settings. We propose to focus on the next section on the performance of RNN-PG only.



(A) Results on artificial DoubleGaussian Dataset with 2-categories classification problems.

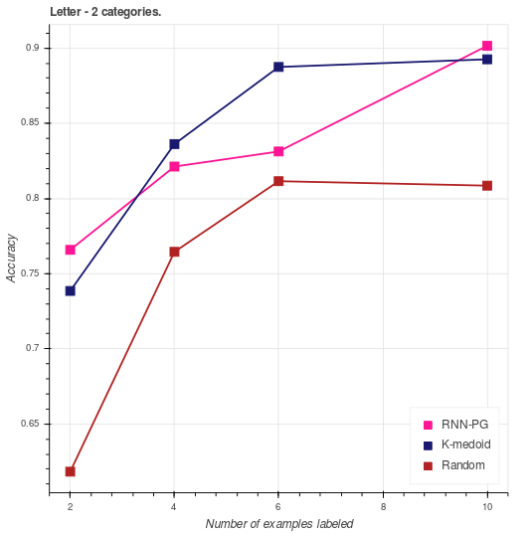


(B) Results on artificial DoubleGaussian Dataset with 4-categories classification problems.

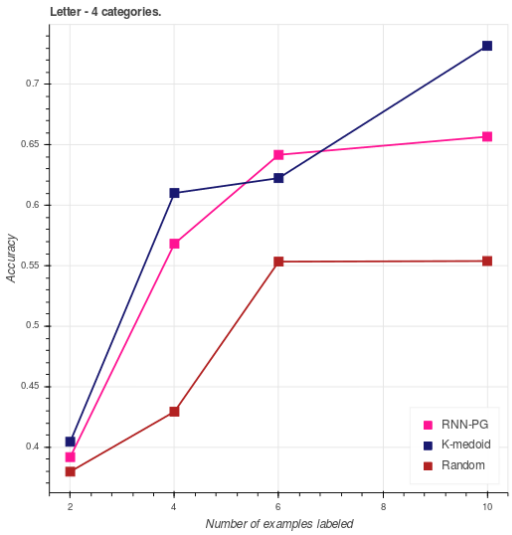


(C) Results on artificial DoubleGaussian Dataset with 6-categories classification problems.

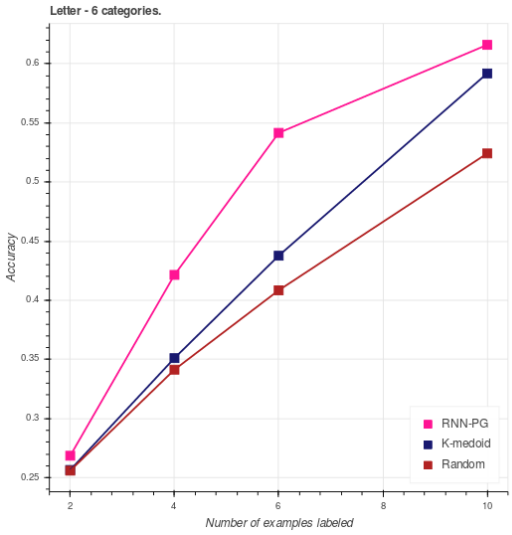
FIGURE 6.5: Plots of results on three artificial datasets, where inputs are sampled from a double-gaussian distribution, with 2,4 or 6 categories per problem. K-medoids acquisition strategy is depicted in blue, random acquisition strategy in red. Our model using Policy-Gradient is in pink, and using the deterministic ReLU instantiation in yellow. Abscissa is the number of examples selected for labeling, ordinate is the average accuracy obtained on all test-problems. For each model, we select the best results on validation problems for each budget, and plot the corresponding performance on test problems (square points).



(A) Results on dataset *letter* with 2-categories classification problems.



(B) Results on dataset *letter* with 4-categories classification problems.



(C) Results on dataset *letter* with 6-categories classification problems.

FIGURE 6.6: Plots of results on uci-dataset *letter*, with 2,4 or 6 categories per problem. K-medoids acquisition strategy is depicted in blue, random acquisition strategy in red. Our model using Policy-Gradient is in pink. Abscissa is the number of examples selected for labeling, ordinate is the average accuracy obtained on all test-problems. For each model, we select the best results on validation problems for each budget, and plot the corresponding performance on test problems (square points).

Results on real datasets

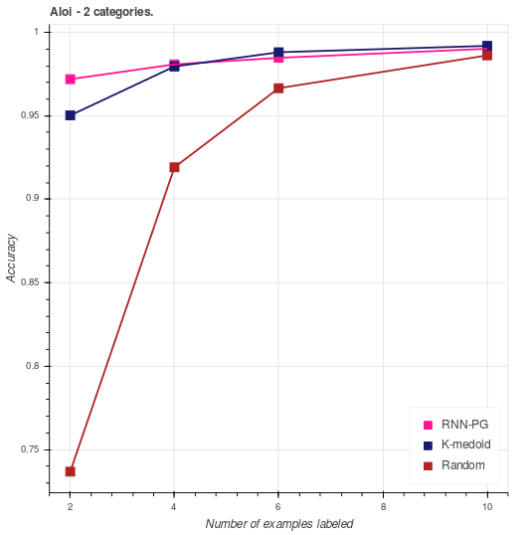
We propose to work on the UCI dataset *letter*, which has 26 categories and 16 features. We took 10 categories for training, 7 for validation and 9 for testing. We generated 2000 problems in training and 500 problems for validation and testing. The size of a dataset (examples that can be labeled) is 25, and the number of examples to classify per problem is 40. Here again we study 3 types of problems, binary, 4-classes and 6-classes with various budget levels. The results are plotted in Figure 6.6. We observe mixed results. Our model performs better than a k-medoid acquisition strategy for a budget of 2 on binary-classification problems, but k-medoid leads to a better accuracy for higher budgets. It is also better for all budgets except 6 on 4-categories problems. For 6-categories problems, our model beats the two baselines for all budgets. This difference of performance can be explained by the small amount of different categories in the training dataset; with 10 categories and binary problems (45 possible combinations), our model can observe the same problem a large number of times, which could lead to overfitting. This seems the case, as it performs better on 6-classes problems (210 possible combinations). We propose thus to study now a dataset with a larger number of categories.

On the dataset *aloi*, a dataset of 1000 small objects, with around a hundred images per object, we created 4000 training problems on 350 categories, and 500 validation and testing problems on respectively 300 and 350 categories. The number of examples that can be labeled is 25, and the number of examples to classify per problem is 40. The results are shown in Figure 6.7, for the 3 types of problems (2-classes, 4-classes and 6-classes). We see that our method performs better than k-medoid for all budgets and all types of problems, except on binary-classification with budget 6, where k-medoid performs slightly better (0.5%). On this bigger dataset, our approach is less prone to overfit, and thus manages to generalize well its acquisition strategy to novel problems on unseen categories.

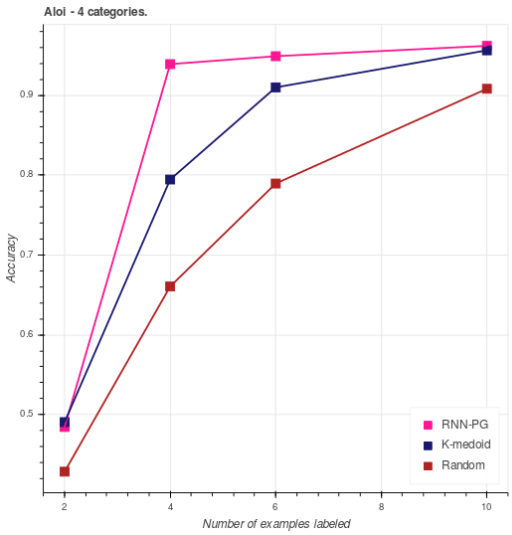
6.5 Perspective : Hybrid Model

We have presented an approach for *static* acquisition of labels in a single step. The model seems to perform well on the datasets we studied, which illustrates its ability to find a smart strategy (as efficient or better than a k-medoid strategy) for choosing which examples to label with a single glance at the data. Yet, an adaptive method may be more effective. In our approach, the vector for acquisition is computed w.r.t. the entire unsupervised dataset, but each value has no explicit information about the probabilities of acquiring the other examples[4]. Furthermore, as acquisition is done in a single step, the system do not use any feedback regarding the acquired labels. The results could be improved if the decision was made sequentially, one example after another, waiting for the category of the selected example.

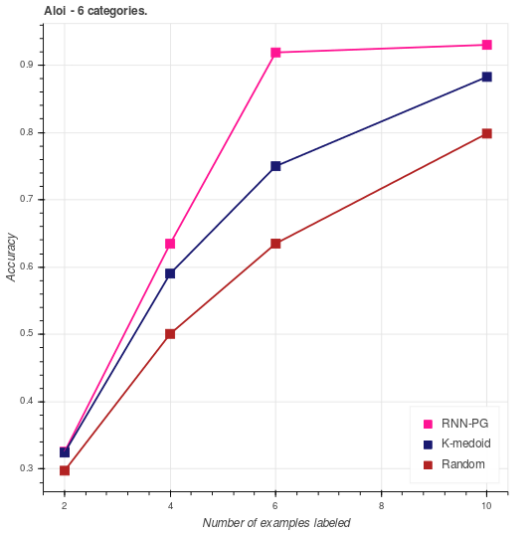
[4]The information is implicit from the internal states of the bi-directional RNN



(A) Results on dataset *aloi* with 2-categories classification problems.



(B) Results on dataset *aloi* with 4-categories classification problems.



(C) Results on dataset *aloi* with 6-categories classification problems.

FIGURE 6.7: Plots of results on uci-dataset *aloi*, with 2,4 or 6 categories per problem. K-medoids acquisition strategy is depicted in blue, random acquisition strategy in red. Our model using Policy-Gradient is in green. Abscissa is the number of examples selected for labeling, ordinate is the average accuracy obtained on all test-problems. For each model, we select the best results on validation problems for each budget, and plot the corresponding performance on test problems (square points).

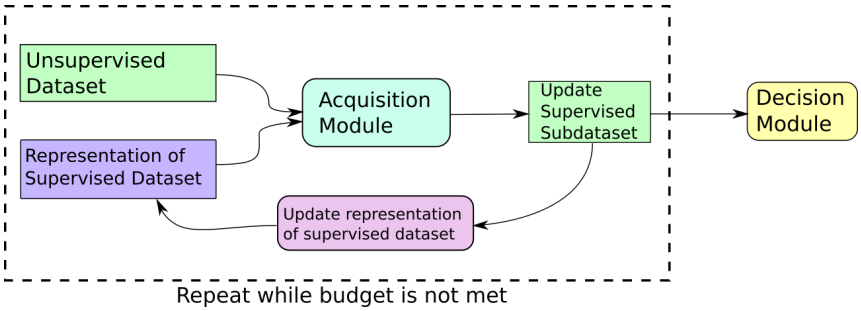


FIGURE 6.8: Architecture of an adaptive framework relying on a representation of the currently supervised dataset. The acquisition process is repeated T times (e.g if T is the number of examples to label and one example is labeled per step). After acquisition, i.e. receiving the labels of the selected inputs, the representation of the supervised dataset is updated with this new information.

It is possible to design such method as an extension of our approach. The key idea is to use an additional input in the acquisition component: a representation vector of the currently supervised dataset. The goal is to have a representation that encapsulates the inputs that are already selected, with their labels, to guide the acquisition on the remaining examples. It relies on a similar idea as [Vinyals et al., 2016], where the representation of each example is dependant of the dataset. In our case, the acquisition decision should be made w.r.t. the labeled examples so far. Such representation can be built using recurrent networks architectures, e.g. order invariant RNN, or a memory network. If the representation model for the supervised dataset is fully differentiable, it can be integrated easily into our framework. The process can be sequentialized by stacking the network acquisition as many times as there are acquisition steps. A schematic illustration of such model is provided in Figure 6.8.

Closing remarks The problem and the framework presented in this chapter open various leads for future research. It seems crucial to go further on this path to design more complex models that get closer to a "general" M.L. method. We defined here a possible step to develop systems capable of considering different problems, as a whole, instead of a unique task. Yet, we want to highlight several limitations, which encourage more thorough studies. First, we presented here a framework able to predict on various problems, however, there is a strong assumption w.r.t. the common nature of all inputs. This is quite restrictive and requires further investigation. We note on the other hand that for robotic multi-tasks problems, it could be a reasonable hypothesis (e.g. a robot has always access to the same nature of feedback, like images or radar measures). Secondly, the instantiations we presented have a fixed budget (i.e always acquire the same number of examples), and the datasets were composed of similar problems (e.g all problems are binary classification). This encourages us to (i) study the capacity of our approach to predict on distinct types of problems, (ii) design methods that have a *soft* budget for each problem. Third,

we focused here on prediction methods based on similarities and on classification problems. More generic models should be investigated for other types of tasks.

Chapter 7

Conclusion and Closing Remarks

The main contribution of this thesis is a general framework for adaptive sequential acquisition of features under cost-sensitive constraints, which allows designing differentiable and scalable models for this task. We presented several instantiations that can acquire several features at each step, without suffering from combinatorial limitations and without making assumptions about the nature of the input.

In Chapter 3, we presented a model for static feature selection. We focused on the specific problem of user cold start recommendation, where a new user arrives in the system with no information about him or her. Unless previous approaches that focus solely on designing interview processes for the user, our method proposes to learn in a tied fashion the interview process as well as the decision process, based on representation learning techniques. We chose to rely on inductive representations for the users, at the opposite of matrix factorization based techniques. The key idea is that a representation can be easily updated when a new information arrives (e.g a rating), as a translation in the representation space, instead of having to re-optimize the whole system. This choice enables us to design a system that is not only efficient during cold-start but can be used all through the life of the recommender system. We showed the relevance of our model on various settings on different recommendation datasets, from strict cold-start to more classical "warm" set-up.

While static feature selection is relevant for the specific problem of user cold-start, adaptive approaches are more appropriate for the general problem of costly features. Adaptiveness should allow a better trade-off between cost and performance: for a similar "budget", the system can choose to acquire more relevant features by adapting its behavior w.r.t. what has been currently observed. We presented in Chapters 4 and 5 a framework for such adaptive methods, based on recurrent neural networks. One key idea is, similarly to the previous method, to rely on representation learning. With adaptive acquisition processes, the goal is to acquire different features from one input to another, and potentially fewer features for some inputs that are maybe "easier" to classify. Thus, a common representation space allows expressing all partially observed inputs w.r.t. their currently gathered features. In our framework, this representation also drives the acquisition process as well as the final prediction process. This framework can be implemented using recurrent neural

networks, which intrinsically provides sequential abilities, and builds an internal representation state of the process. We propose a first instantiation in Chapter 4, based on a relaxation of the loss we defined for the problem. The model is fully differentiable and thus can be learned efficiently with classical or more recent algorithms for gradient descent. It can also gather several features at each acquisition steps. This is a novel aspect compared to the methods in literature which can prove really useful. We illustrate the performance of this model on several experiments, with two cost settings (uniform and cost-sensitive). In Chapter 5, a stochastic instantiation of the framework is presented. The goal of having such stochasticity is to get closer to the real loss we defined for the adaptive feature acquisition problem. We relied on policy gradient technique and Monte-Carlo sampling to learn a stochastic version of our recurrent neural networks. More specifically, we showed that we can define various ways of sampling the features to acquire, and we proposed to use Bernoulli-sampling, which keeps the ability of the network to acquire several features at each step. While the results are still competitive with the baselines we proposed, it was slightly less accurate than our continuous relaxation from the previous chapter. It can be explained by the sampling phase in the stochastic method, which induces a longer training phase. Mainly, the stochastic approach is slower to converge than the continuous one.

In Chapter 6, we proposed to focus on the cost of labels. We presented a novel framework to design meta-active learning methods. Unlike classical active-learning algorithms, the key idea here is to learn an active-learning strategy, i.e learn which examples should be labeled to better predict afterward. We proposed a protocol inspired by recent work on one-shot learning, but instead of receiving each example of a problem sequentially, we chose to consider the dataset as a pool of examples, fully observed at the beginning of the process. We instantiated a model composed of specific recurrent neural networks like bidirectional-LSTM to predict which labels to acquire, and we show on preliminary results the relevance of such systems compared to more naive acquisition techniques. Yet, this work opens several leads for research that we briefly describe now.

7.1 Future directions

Our preliminary works at the crossroad of meta-learning and active-learning lead to several interesting paths for future research. This research seems particularly crucial on two main aspects. First, meta-leaning is a key problem in artificial intelligence. Going further with models able to **rapidly learn new tasks**, without forgetting, instead of having a model learned for a single task, is a huge step for A.I. It has been a trending topic recently, for example with challenges on gradual learning, or focuses

on life-long learning. Secondly, **budgeted and constrained systems** are not only interesting for real-life applications with costs, but also for a more general goal: to design models that go towards more **semi-supervised learning**, closer to human-like learning. Unsupervised and semi-supervised learning are considered the next big steps for artificial intelligence, and several recent works show encouraging results in this regard. If one considers "human-like" learning as an acceptable referential, we can observe that we learn from a variety of information, that are sometimes "labeled", and most of the time not. We also have several constraints regarding the information we can access when we learn. Integrating budget or cost in a system is therefore not only necessary to deal with several applications but also to force the system to be "smarter", instead of solely process a large amount of data. Mimicking these constraints seems to be one way to move towards this goal.
These motivations encourage us to go further on designing **interacting meta-learning systems**, where the model not only has to learn to learn, but also has to learn how to use and interact with the available information and more generally with the new problem it is facing. Firstly, several extensions of the model presented in Chapter 6 are straightforward. Specifically, improvements could be done by using recent order-invariant RNN model to process the dataset of examples. Going further with the hybrid model, to provide adaptive acquisition and integration of labels feedback is the next logical step. On the long term, developing models that can acquire information at various levels (labels and features) with a meta-learning strategy is a key target. One interesting aspect would be to focus on strategies of acquisitions as well as strategies of "storage" (or "remembering"), relying for example on the recent memory networks. The underlying problem that we want to highlight here is both how to use at best the information the environment can provide, but also how to use what the system has seen and learn before, and how to store it to not forget the more important knowledge. Additionally, we focused on prediction methods that rely only on similarities, as they do not have to be learned per say for each new problem. Unfortunately, this limits the type of tasks one can tackle, and prevents to deal with a large heterogeneity of problems.

Bibliography

Aha, David W (1992). "Generalizing from case studies: A case study". In: *Proc. of the 9th International Conference on Machine Learning*, pp. 1–10.

Ahn, Hyung Jun (2008). "A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem". In: *Information Sciences* 178.1, pp. 37–51.

Almuallim, Hussein and Thomas G Dietterich (1991). "Learning with Many Irrelevant Features." In: *AAAI*. Vol. 91, pp. 547–552.

Andrychowicz, Marcin et al. (2016). "Learning to learn by gradient descent by gradient descent". In: *Advances in Neural Information Processing Systems*, pp. 3981–3989.

Ba, Jimmy, Volodymyr Mnih, and Koray Kavukcuoglu (2014). "Multiple Object Recognition with Visual Attention". In: *arXiv preprint arXiv:1412.7755*.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473*.

Basilico, Justin and Thomas Hofmann (2004). "Unifying collaborative and content-based filtering". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM, p. 9.

Bekkerman, Ron et al. (2003). "Distributional word clusters vs. words for text categorization". In: *Journal of Machine Learning Research* 3.Mar, pp. 1183–1208.

Bellet, Aurélien, Amaury Habrard, and Marc Sebban (2013). "A survey on metric learning for feature vectors and structured data". In: *arXiv preprint arXiv:1306.6709*.

Benbouzid, Djalel, Róbert Busa-Fekete, and Balázs Kégl (2012). "Fast classification using sparse decision DAGs". In: *ICML*.

Bennett, James, Stan Lanning, et al. (2007). "The netflix prize". In: *Proceedings of KDD cup and workshop*. Vol. 2007. New York, NY, USA, p. 35.

Berko, Jean (1958). "The child's learning of English morphology". In: *Word* 14.2-3, pp. 150–177.

Bertinetto, Luca et al. (2016). "Learning feed-forward one-shot learners". In: *Advances in Neural Information Processing Systems*, pp. 523–531.

Bi, Jinbo et al. (2003). "Dimensionality reduction via sparse support vector machines". In: *JMLR* 3, pp. 1229–1243.

Bilgic, Mustafa and Lise Getoor (2007). "Voila: Efficient feature-value acquisition for classification". In: *Proceedings of AAAI*. Vol. 22. 2, p. 1225.

Bobadilla, JesúS et al. (2012). "A collaborative filtering approach to mitigate the new user cold start problem". In: *Knowledge-Based Systems* 26, pp. 225–238.

Breiman, Leo et al. (1984). *Classification and regression trees*. CRC press.

Carpenter, Bob (2008). "Lazy sparse stochastic gradient descent for regularized multinomial logistic regression". In: *Alias-i, Inc., Tech. Rep*, pp. 1–20.

Chai, Xiaoyong et al. (2004). "Test-cost sensitive naive bayes classification". In: *Data Mining,ICDM'04*.

Chen, Minmin et al. (2012). "Classifier cascade for minimizing feature evaluation cost". In: *AISTATS*, pp. 218–226.

Cho, Kyunghyun et al. (2014a). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*.

Cho, Kyunghyun et al. (2014b). "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259*.

Collet, Timothe (2016). "Methodes optimistes d'apprentissage actif pour la classification". PhD thesis. Universite de Lorraine.

Collet, Timothé and Olivier Pietquin (2014). "Optimistic Active Learning for Classification". In: *ECML/PKDD 2014*, p. 11.

Contardo, Gabriella, Ludovic Denoyer, and Thierry Artieres (2015). "Representation Learning for cold-start recommendation". In: *arXiv preprint arXiv:1412.7156*.

Contardo, Gabriella, Ludovic Denoyer, and Thierry Artières (2016a). "Recurrent neural networks for adaptive feature acquisition". In: *International Conference on Neural Information Processing*. Springer, pp. 591–599.

– (2016b). "Sequential Cost-Sensitive Feature Acquisition". In: *International Symposium on Intelligent Data Analysis*. Springer, pp. 284–294.

Dhillon, Inderjit S, Subramanyam Mallela, and Rahul Kumar (2003). "A divisive information-theoretic feature clustering algorithm for text classification". In: *Journal of machine learning research* 3.Mar, pp. 1265–1287.

Duda, Richard O, Peter E Hart, and David G Stork (2001). "Pattern classification. 2nd". In: *Edition. New York*, p. 55.

Dulac-Arnold, Gabriel et al. (2012). "Sequential approaches for learning datum-wise sparse representations". In: *Machine learning*.

Dulac-Arnold, Gabriel et al. (2013). "Sequentially generated instance-dependent image representations for classification". In: *arXiv preprint arXiv:1312.6594*.

Edwards, Harrison and Amos Storkey (2016). "Towards a Neural Statistician". In: *arXiv preprint arXiv:1606.02185*.

Fe-Fei, Li et al. (2003). "A Bayesian approach to unsupervised one-shot learning of object categories". In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE, pp. 1134–1141.

Fei-Fei, Li, Rob Fergus, and Pietro Perona (2006). "One-shot learning of object categories". In: *IEEE transactions on pattern analysis and machine intelligence* 28.4, pp. 594–611.

Fink, Michael (2004). "Object Classification from a Single Example Utilizing Class Relevance Metrics." In: *NIPS*. Vol. 1. 2, p. 8.

Forman, George (2003). "An extensive empirical study of feature selection metrics for text classification". In: *Journal of machine learning research* 3.Mar, pp. 1289–1305.

Freund, Yoav et al. (1997). "Selective sampling using the query by committee algorithm". In: *Machine learning* 28.2-3, pp. 133–168.

Fukushima, Kunihiko and Sei Miyake (1982). "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, pp. 267–285.

Gao, Tianshi and Daphne Koller (2011). "Active classification based on value of classifier". In: *NIPS*.

Golbandi, Nadav, Yehuda Koren, and Ronny Lempel (2010). "On bootstrapping recommender systems". In: *Proceedings of the 19th ACM CIKM*. ACM, pp. 1805–1808.

– (2011). "Adaptive bootstrapping of recommender systems using decision trees". In: *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, pp. 595–604.

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). "Neural turing machines". In: *arXiv preprint arXiv:1410.5401*.

Graves, Alex et al. (2009). "A novel connectionist system for unconstrained handwriting recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 31.5, pp. 855–868.

Greff, Klaus et al. (2016). "LSTM: A search space odyssey". In: *IEEE transactions on neural networks and learning systems*.

Gregor, Karol et al. (2015). "DRAW: A recurrent neural network for image generation". In: *arXiv preprint arXiv:1502.04623*.

Gu, Quanquan et al. (2012). "Selective labeling via error bound minimization". In: *Advances in neural information processing systems*, pp. 323–331.

Guillory, Andrew and Jeff A Bilmes (2009). "Label selection on graphs". In: *Advances in Neural Information Processing Systems*, pp. 691–699.

Gullapalli, Vijaykumar (1992). "Reinforcement learning and its application to control". PhD thesis. Citeseer.

Guo, Yuhong and Dale Schuurmans (2008). "Discriminative batch mode active learning". In: *Advances in neural information processing systems*, pp. 593–600.

Guyon, Isabelle and André Elisseeff (2003). "An introduction to variable and feature selection". In: *JMLR*.

Harpale, Abhay S and Yiming Yang (2008). "Personalized active learning for collaborative filtering". In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 91–98.

He, He, Hal Daumé III, and Jason Eisner (2012). "Cost-sensitive dynamic feature selection". In: *ICML Workshop: Interactions between Inference and Learning, Edinburgh*.

Herlocker, Jonathan L et al. (1999). "An algorithmic framework for performing collaborative filtering". In: *Proceedings of the 22nd annual international ACM SIGIR*. ACM, pp. 230–237.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hochreiter, Sepp, A Steven Younger, and Peter R Conwell (2001). "Learning to learn using gradient descent". In: *International Conference on Artificial Neural Networks*. Springer, pp. 87–94.

Hoi, Steven CH et al. (2006). "Batch mode active learning and its application to medical image classification". In: *Proceedings of the 23rd international conference on Machine learning*. ACM, pp. 417–424.

– (2009). "Semisupervised SVM batch mode active learning with applications to image retrieval". In: *ACM Transactions on Information Systems (TOIS)* 27.3, p. 16.

Ji, Shihao and Lawrence Carin (2007). "Cost-sensitive feature acquisition and classification". In: *Pattern Recognition* 40.5, pp. 1474–1485.

Jin, Rong and Luo Si (2004). "A bayesian approach toward active learning for collaborative filtering". In: *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, pp. 278–285.

Kaelbling, Leslie Pack, Michael L Littman, and Andrew W Moore (1996). "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4, pp. 237–285.

Kaiser, Łukasz and Samy Bengio (2016). "Can Active Memory Replace Attention?" In: *Advances in Neural Information Processing Systems*, pp. 3774–3782.

Kaiser, Łukasz and Ilya Sutskever (2015). "Neural gpus learn algorithms". In: *arXiv preprint arXiv:1511.08228*.

Kim, Hyunsoo and Haesun Park (2007). "Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis". In: *Bioinformatics* 23.12, pp. 1495–1502.

Kim, Jingu and Haesun Park (2008). *Sparse nonnegative matrix factorization for clustering*. Tech. rep. Georgia Institute of Technology.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Koch, Gregory (2015). "Siamese neural networks for one-shot image recognition". PhD thesis. University of Toronto.

Kohavi, Ron and George H John (1997). "Wrappers for feature subset selection". In: *Artificial intelligence* 97.1, pp. 273–324.

Koren, Yehuda (2010). "Factor in the neighbors: Scalable and accurate collaborative filtering". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4.1, p. 1.

Koren, Yehuda and Robert Bell (2015). "Advances in collaborative filtering". In: *Recommender systems handbook*. Springer, pp. 77–118.

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix factorization techniques for recommender systems". In: *Computer* 42.8, pp. 30–37.

Krishnapuram, Balaji et al. (2005). "Active learning of features and labels". In: *Workshop on learning with multiple views at the 22nd International Conference on Machine Learning (ICML-05)*, pp. 43–50.

Kulis, Brian et al. (2013). "Metric learning: A survey". In: *Foundations and Trends® in Machine Learning* 5.4, pp. 287–364.

Kuvayev, D and Richard S Sutton (1997). *Model-based reinforcement learning*. Tech. rep. Citeseer.

Lake, Brenden M, Ruslan R Salakhutdinov, and Josh Tenenbaum (2013). "One-shot learning by inverting a compositional causal process". In: *Advances in neural information processing systems*, pp. 2526–2534.

Lake, Brenden M et al. (2011). "One shot learning of simple visual concepts." In: *CogSci*. Vol. 172, p. 2.

Lake, Brenden M et al. (2014). "One-shot learning of generative speech concepts." In: *CogSci*.

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lee, Daniel D and H Sebastian Seung (1999). "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755, pp. 788–791.

Li, Ke and Jitendra Malik (2016). "Learning to optimize". In: *ICLR*.

Littman, Michael L and Csaba Szepesvári (1996). "A generalized reinforcement-learning model: Convergence and applications". In: *ICML*, pp. 310–318.

Lomasky, Rachel et al. (2007). "Active class selection". In: *European Conference on Machine Learning*. Springer, pp. 640–647.

Luo, Xin et al. (2014). "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems". In: *IEEE Transactions on Industrial Informatics* 10.2, pp. 1273–1284.

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning (2015). "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025*.

Luong, Minh-Thang et al. (2014). "Addressing the rare word problem in neural machine translation". In: *arXiv preprint arXiv:1410.8206*.

Melville, Prem et al. (2005). "An expected utility approach to active feature-value acquisition". In: *Data Mining, Fifth IEEE International Conference on*. IEEE, 4–pp.

Miller, George A (1956). "The magical number seven, plus or minus two: some limits on our capacity for processing information." In: *Psychological review* 63.2, p. 81.

Mnih, Volodymyr, Nicolas Heess, Alex Graves, et al. (2014). "Recurrent models of visual attention". In: *NIPS*.

Munkhdalai, Tsendsuren and Hong Yu (2017). "Meta Networks". In: *arXiv preprint arXiv:1703.00837*.

Nan, Feng, Joseph Wang, and Venkatesh Saligrama (2015). "Feature-budgeted random forest". In: *arXiv preprint arXiv:1502.05925*.

Paatero, Pentti and Unto Tapper (1994). "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values". In: *Environmetrics* 5.2, pp. 111–126.

Pan, Sinno Jialin and Qiang Yang (2010). "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10, pp. 1345–1359.

Pazzani, Michael J and Daniel Billsus (2007). "Content-based recommendation systems". In: *The adaptive web*. Springer, pp. 325–341.

Pratt, L and S Thrun (1997). *Second special issue on inductive transfer*.

Pratt, Lorien and Barbara Jennings (1996). "A survey of connectionist network reuse through transfer". In: *Learning to learn*. Springer, pp. 19–43.

Provost, Foster, Prem Melville, and Maytal Saar-Tsechansky (2007). "Data acquisition and cost-effective predictive modeling: targeting offers for electronic commerce". In: *Proceedings of the ninth international conference on Electronic commerce*. ACM, pp. 389–398.

Raghavan, Hema, Omid Madani, and Rosie Jones (2006). "Active learning with feedback on features and instances". In: *The Journal of Machine Learning Research* 7, pp. 1655–1686.

Rashid, Al Mamunur, George Karypis, and John Riedl (2008). "Learning preferences of new users in recommender systems: an information theoretic approach". In: *ACM SIGKDD Explorations Newsletter* 10.2, pp. 90–100.

Rashid, Al Mamunur et al. (2002). "Getting to know you: learning new user preferences in recommender systems". In: *Proceedings of the 7th IUI*. ACM, pp. 127–134.

Ravi, Sachin and Hugo Larochelle (2017). "Optimization as a model for few-shot learning". In: *ICLR*.

Raykar, Vikas C, Balaji Krishnapuram, and Shipeng Yu (2010). "Designing efficient cascaded classifiers: tradeoff between accuracy and cost". In: *16th ACM SIGKDD*.

Rendell, Larry, Raj Seshu, and David Tcheng (1987). "More robust concept learning using dynamically-variable bias". In: *Proceedings of the Fourth International Workshop on Machine Learning*, pp. 66–78.

Rendell, Larry A, Raj Sheshu, and David K Tcheng (1987). "Layered Concept-Learning and Dynamically Variable Bias Management." In: *IJCAI*, pp. 308–314.

Rendle, Steffen and Lars Schmidt-Thieme (2008). "Online-updating regularized kernel matrix factorization models for large-scale recommender systems". In: *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, pp. 251–258.

Resnick, Paul et al. (1994). "GroupLens: an open architecture for collaborative filtering of netnews". In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, pp. 175–186.

Rezende, Danilo Jimenez et al. (2016). "One-shot generalization in deep generative models". In: *arXiv preprint arXiv:1603.05106*.

Rosenblatt, Frank (1958). "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Rubens, Neil, Dain Kaplan, and Masashi Sugiyama (2011). "Active Learning in Recommender Systems". In: *Recommender Systems Handbook*. Ed. by P.B. Kantor et al. Springer, pp. 735–767. DOI: 10.1007/978-0-387-85820-3_23.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. DTIC Document.

Sak, Hasim, Andrew W Senior, and Françoise Beaufays (2014). "Long short-term memory recurrent neural network architectures for large scale acoustic modeling." In: *Interspeech*, pp. 338–342.

Salakhutdinov, Ruslan, Andriy Mnih, and Geoffrey Hinton (2007). "Restricted Boltzmann machines for collaborative filtering". In: *Proceedings of the 24th international conference on Machine learning*. ACM, pp. 791–798.

Salakhutdinov, Ruslan, Joshua B Tenenbaum, and Antonio Torralba (2012). "One-Shot Learning with a Hierarchical Nonparametric Bayesian Model." In: *ICML Unsupervised and Transfer Learning*, pp. 195–206.

Santoro, Adam et al. (2016). "Meta-learning with memory-augmented neural networks". In: *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1842–1850.

Sarwar, Badrul et al. (2001). "Item-based collaborative filtering recommendation algorithms". In: *Proceedings of WWW*. ACM, pp. 285–295.

Schmidhuber, Juergen (1995). "On learning how to learn learning strategies". In:

Schmidhuber, Jürgen (2004). "Optimal ordered problem solver". In: *Machine Learning* 54.3, pp. 211–254.

Schmidhuber, Jurgen, Jieyu Zhao, and Marco Wiering (1996). "Simple principles of metalearning". In:

Schuster, Mike and Kuldip K Paliwal (1997). "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681.

Sermanet, Pierre, Andrea Frome, and Esteban Real (2014). "Attention for Fine-Grained Categorization". In: *arXiv preprint arXiv:1412.7054*.

Settles, Burr (2010). "Active learning literature survey". In: *University of Wisconsin, Madison* 52.55-66, p. 11.

Shi, Yue, Martha Larson, and Alan Hanjalic (2014). "Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges". In: *ACM Computing Surveys (CSUR)* 47.1, p. 3.

Strehl, Alexander L et al. (2006). "PAC model-free reinforcement learning". In: *Proceedings of the 23rd international conference on Machine learning*. ACM, pp. 881–888.

Sun, Mingxuan et al. (2013). "Learning multiple-question decision trees for cold-start recommendation". In: *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, pp. 445–454.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*, pp. 3104–3112.

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge.

Thrun, Sebastian and Lorien Pratt (1998). "Learning to learn: Introduction and overview". In: *Learning to learn*. Springer, pp. 3–17.

Tibshirani, Robert (1994). *Regression selection and shrinkage via the lasso*. Tech. rep. Technical report, Stanford University, Palo Alto, CA.

Tibshirani, Robert (1996). "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288.

Tong, Simon and Daphne Koller (2001). "Support vector machine active learning with applications to text classification". In: *Journal of machine learning research* 2.Nov, pp. 45–66.

Torkkola, Kari (2003). "Feature extraction by non-parametric mutual information maximization". In: *Journal of machine learning research* 3.Mar, pp. 1415–1438.

Trapeznikov, Kirill and Venkatesh Saligrama (2013). "Supervised sequential classification under budget constraints". In: *AISTATS*.

Trapeznikov, Kirill, Venkatesh Saligrama, and David Castañón (2013). "Multi-stage classifier design". In: *Machine learning* 92.2-3, pp. 479–502.

Turney, Peter D. (1995). "Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm". In: *Journal of artificial intelligence research*.

Utgoff, Paul E (1986). "Shift of bias for inductive concept learning". In: *Machine learning: An artificial intelligence approach* 2, pp. 107–148.

Verikas, Antanas and Marija Bacauskiene (2002). "Feature selection with neural networks". In: *Pattern Recognition Letters* 23.11, pp. 1323–1335.

Vilalta, Ricardo and Youssef Drissi (2002). "A perspective view and survey of meta-learning". In: *Artificial Intelligence Review* 18.2, pp. 77–95.

Vinyals, Oriol, Samy Bengio, and Manjunath Kudlur (2015). "Order matters: Sequence to sequence for sets". In: *arXiv preprint arXiv:1511.06391*.

Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly (2015). "Pointer networks". In: *Advances in Neural Information Processing Systems*, pp. 2692–2700.

Vinyals, Oriol et al. (2016). "Matching networks for one shot learning". In: *Advances in Neural Information Processing Systems*, pp. 3630–3638.

Viola, Paul and Michael Jones (2001). "Robust real-time object detection". In: *International Journal of Computer Vision* 4, pp. 51–52.

Wang, Xuezhi, Tzu-Kuo Huang, and Jeff G Schneider (2014). "Active Transfer Learning under Model Shift." In: *ICML*, pp. 1305–1313.

Weiss, David J and Ben Taskar (2013). "Learning adaptive value of information for structured prediction". In: *NIPS*.

Werbos, Paul (1974). "Beyond regression: New tools for prediction and analysis in the behavioral sciences". In:

Weston, Jason, Sumit Chopra, and Antoine Bordes (2014). "Memory networks". In: *arXiv preprint arXiv:1410.3916*.

Weston, Jason et al. (2000). "Feature selection for SVMs". In: *NIPS*.

Weston, Jason et al. (2003). "Use of the zero norm with linear models and kernel methods". In: *JMLR*.

Wierstra, Daan et al. (2007). "Solving deep memory POMDPs with recurrent policy gradients". In: *ICANN*.

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4, pp. 229–256.

Wolpert, David H (1992). "Stacked generalization". In: *Neural networks* 5.2, pp. 241–259.

Woodward, Mark and Chelsea Finn (2017). "Active One-shot Learning". In:

Wu, Dongrui, Brent J Lance, and Thomas D Parsons (2013). "Collaborative filtering for brain-computer interaction using transfer learning and active class selection". In: *PloS one* 8.2, e56624.

Wu, Dongrui and Thomas D Parsons (2011). "Active class selection for arousal classification". In: *Affective Computing and Intelligent Interaction*. Springer, pp. 132–141.

Xu, Kelvin et al. (2015). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention." In: *ICML*. Vol. 14, pp. 77–81.

Xu, Zhixiang, Kilian Weinberger, and Olivier Chapelle (2012). "The greedy miser: Learning under test-time budgets". In: *arXiv preprint arXiv:1206.6451*.

Xu, Zhixiang et al. (2014a). "Classifier cascades and trees for minimizing feature evaluation cost". In: *JMLR*.

Xu, Zhixiang et al. (2014b). "Gradient boosted feature selection". In: *ACM SIGKDD*.

Yan, Yan et al. (2012). "Active transfer learning for multi-view head-pose classification". In: *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, pp. 1168–1171.

Yip, Kenneth and Gerald Jay Sussman (1997). "Sparse representations for fast, one-shot learning". In:

Yu, Kai, Jinbo Bi, and Volker Tresp (2006). "Active learning via transductive experimental design". In: *Proceedings of the 23rd international conference on Machine learning*. ACM, pp. 1081–1088.

Yuan, M. and Y. Lin (2005). "Efficient Empirical Bayes Variable Selection and Estimation in Linear Models". In: *Journal of the American Statistical Association*.

Zhang, Tong and F Oles (2000). "The value of unlabeled data for classification problems". In: *Proceedings of the Seventeenth International Conference on Machine Learning,(Langley, P., ed.)* Citeseer, pp. 1191–1198.

Zhao, Lili et al. (2013). "Active Transfer Learning for Cross-System Recommendation." In: *AAAI*. Citeseer.

Zhou, Ke, Shuang-Hong Yang, and Hongyuan Zha (2011). "Functional matrix factorizations for cold-start recommendation". In: *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, pp. 315–324.