National University of Singapore

Master Thesis : Statistics and Applied Probabilities

Efficient Extreme Classification with Label Taxonomy based Neural Network

Author :
Nicolas Maurice

Supervisors at UPMC-LIP6/team Machine Learning and Artificial Intelligence :
Patrick Gallinari
Thierry Artières

Supervisor at NUS - Department of Statistics :
Tao Yu

Contents

List of Tables

List of Figures

Acknoledgements

Thanks to Patrick Gallinari and Thierry Artières my supervisors at UPMC (University Pierre & Marie Curie in Paris) where I followed a 6 months internship in the Machine Learning and Artificial Intelligence team and has been one of the most enriching experience in my career

I also would like to thanks Dr Yu Tao in the deparment of Statistics at NUS (National Univerity of Singapore) who accepted to be my supervisor

Chapter 1

Introduction

A fundamental characteristic of the Human mind is the ability to put information into categories. For example, we naturaly extract semantic concepts from visual information or text data, that allow us to organise knowledge. One of the main purpose of artificial intelligence is to endow the machine with the same abilities. The past decades have known a large research effort in the Machine Learning commmunity to automate daily classification tasks. Among many others, Handwritten Digit Recognition (Lecun et al., 1998 [17]) and spam filtering (Tretyakov, 2004 [26]) are succesful industrial applications of classification tools.

Despite this success stories, we have to highlight the gap that still exist, in terms of scale and complexity, between the problems that Humans can naturally tackle and the one a machine can solve. Almost every classification problems solved by computers count at most a few hundred of classes while Humans are able to discriminate between several thousands of categories. Obviously, this is due to the basic fact of high level human inteligence, Humans describe the world thanks to an extremely large semantic space. As an example psychologists consider that we are able to make the difference between thirty thousand visual categories (Biederman, 1987 [7]) and we have a potentially unbounded number of semantic concepts to explain wich are the relevant topics for a given textual document.

Thanks to social and collaborative websites such as Wikipedia, Facebook, Instagram, etc. The past few years have also been the witnesses of an exponential growth of the amount of data uploaded every day on the internet. For example, between 2002 and the end of 2014, the number of articles on the english version of Wikipedia has jumped from 20,000 to 4.7 millions and during the year 2014, Instagram has known an average of 40 millions pictures

shared daily. The relevant point is that all this datas (textual or visual) in most cases come with a system of labels that describe it. For example, every article on Wikipedia is tagged with at least one topic of the roughly 1 million labels reported in the Wikipedia hierarchy. In order to deal with this kind of very large datasets and create user friendly applications such as search engines, it is critical to first create learning machines that can efficiently deal with a large number of categories.

This field of research called *Extreme Classification* has remained unexplored until recently and most of the existing classic machine learning algorithms are not well suited to problems of this size because of their large time and space computational complexity. In most cases, the computational ressources needed to solve a problem grow much faster than the volume of data. This thesis proposes a new approach that efficiently solves classification problems with a large number of categories.

1.1 Challenges in Extreme Classification

As we said earlier, Extreme Classification is an emerging field of research which goal is to handle classification tasks that involve a large number of categories (that we will also call labels or classes). Thanks to several workshops, it has become more popular the past few years, for example the ECML-PKDD Large Scale Hierarchical Text Classification (LSHTC) workshop series 2010-2013, the NIPS 2013 Extreme Classification workshop and more recenty the WSDM 2014 Web-Scale classification workshop. During the same period some contests have been organized such has the Pascal Large Scale Hierarchical Text Classification in conjunction with the workshop, the Imagenet challenge or the BioAsQ challenge.

We can distinguish two main problems in Extreme Classification, single label classification which means that an instance belongs to one and only one category, and multilabel classification where an instance can belong to several categories. In the following we will raise the speficities involved by problems of large scale.

1.1.1 Class Imbalance/Data scarsity

A first issue encountered when dealing with Extreme Classification problems is the categories average size. Indeed the larger the number of categories, the smaller the average category size, this has been observed on sev-

eral datasets, see on figure 1.1, the same conclusion does not hold for small size dataset such as CLEF and RCV. An example of dataset used for regular classification is the MNIST database of handwritten digits (Lecun et al, 1998 [17]), that counts 70,000 examples divided into ten classes of balanced sizes. On the other hand, in Extreme Classification problems, when the number of labels gets large, the average size of a category decreases. For example, the DMOZ database (Directory Mozilla) is an open-content directory of World Wide Web links, as of April 2013 it listed 5 millions links grouped into more than one million categories.

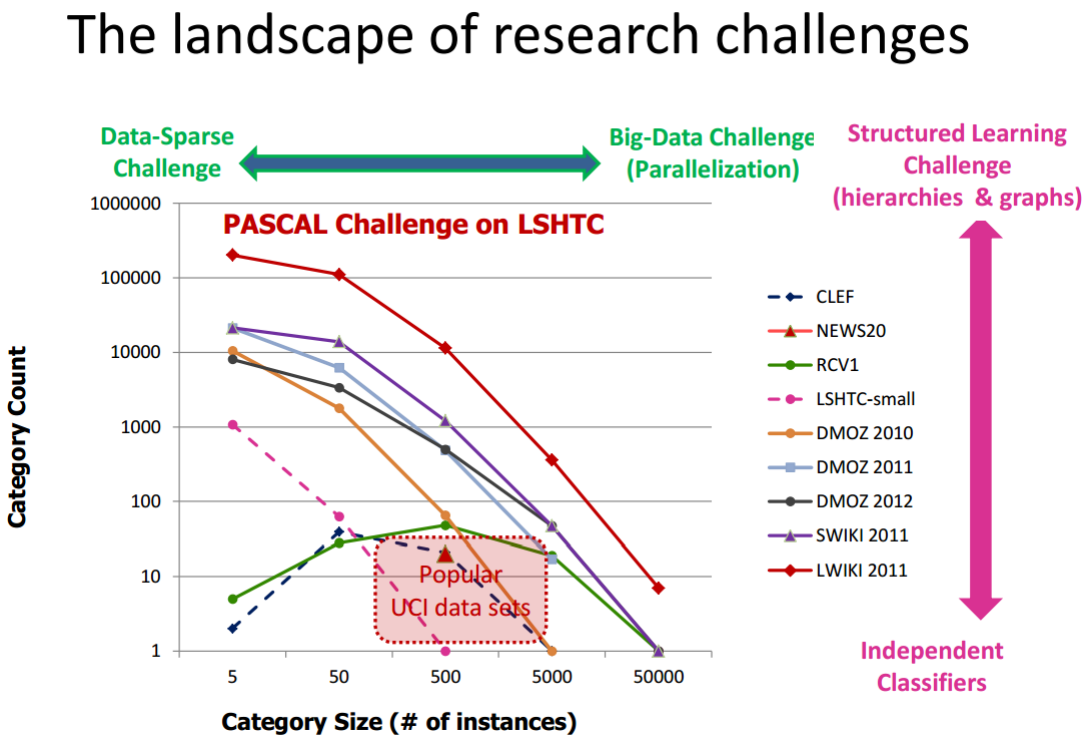


Figure 1.1: Landscape of research challenges in Extreme Classification (Yimin Yang talk at WSC workshop WSDM 2014)

Coupled with this scarsity of datas, a second issue comes from the fact that the size of the different categories is often very imbalanced. As depicted on Figure 1.1, for most of the Extreme Classification datasets, the size of the categories are power law distributed. Even though solutions to the class imbalance have already been proposed (Japkowitcz and Stephen, 2002 [15]),

the scale of the considered problems was much smaller.

1.1.2 High dimensionality/Large sample size

In Extreme Classification problems we often deal with an input space of very large dimensionality. In the large Wikipedia's documents database, that has roughly 325,000 categories, an instance is represented by more than one million features. Indeed for textual data such as DMOZ or Wikipedia, some words of the vocabulary are very specific to some catagories, it comes up that to manage to discriminate between more categories, we need a larger vocabulary.

Since each category is represented by at least one instance, a second fact that is naturally related to the large number of labels is the very large size of the datasets. As quoted before, the DMOZ database counts roughly 5 millions examples.

1.1.3 Structure and Label dependence exploitation

Databases concerned by Extreme Classification, because of their large size often comes with a structure that organizes the categories. For example, the Wikipedia database comes with a graph that structures the labels and with DMOZ, labels are organized in a hierarchy. This kind of structures carry semantic information about the relationship between the labels and this information can be leveraged in order to improve classification performances.

Another kind of structure information is the co-occurence between labels in extreme multilabels problems, some classes are positevely correlated while many others never appear together.

1.1.4 Training/Inference Complexity reduction

The different issues quoted previously : sample size, dimensionality of the input space... result in both memory and time computational burden for classical machine learning algorithm. For example using the one versus rest classifier, that needs to learn one binary classifier for each category (Rifkin and Klautau, 2004b [23]), on the Wikipedia dataset (325,000 labels, dimension of the input space : 1 million), more than 1000 GB would be necessary to store all the parameters. An issue even more problematical is the necessity to evaluate all the 325K classifiers to recover the relevant

categories of a tested instance, it makes the inference's computational time far too long.

The same negative effects due to the large number of labels occur with several other methods such as single large margin classifiers (Weston and Watkins, 1998 [32]) and deep neural networks (Bengio, 2009 [3]). It is therefore critical to come up with new approaches having sublinear training and inference complexity in the number of categories.

Chapter 2

Extreme Single Label Classification

2.1 Introduction

Contrary to *multilabel classification* where an instance can belong to multiple categories, *single label classification* consists in classifying instances that can belong to only one category. It is a well studied problem in machine learning with several solution widely applied in the industry. However from now on with the very fast expansion of the volume of data uploaded every day on the internet with website such as Wikipedia or Instagram, most of the tradional approaches became obsolete because they do not scale to extreme classification. Single label classification has been initially tackled with *flat methods*. Recently there has been an increased interest in *hierarchical techniques*. In this chapter we propose to present the main contributions in the litterature from this two families of approaches that can be applied to solve extreme single label classification.

2.2 Flat Approaches

Hierarchical and flat approaches are opposed in the sense that hierarchical methods rely on a hierarchy of labels to reduce their inference complexity, we will see later how. We can distinguish two groups of flat methods, first machine learning reduction(Allwein et al., 2001 [1], Dietterich and Bakiri, 1995 [12]), second single machine classifier (Weinderberg and Chapelle, 2008, [30], Weston et Watkins, 1999 [33]). The first consists in learning several binary classifiers and then to combine the prediction of all these classifiers to create a classifier over all the labels, for example the one-vs-rest method that

will be described later. On the other hand we have single machine methods that are either embeddings method or extension of binary classifiers.

We will not focus here on extension of binary classifier. Support vector machines and logistic regression have been extended to multiclass (M-SVM)(Weston et Watkins, 1999 [33]). Softmax regression or more recently deep neural network (Bengio, 2009 [3]) are solutions to multiclass classification tasks that have proven their performances however they don't scale to extreme classification, so we will not describe these methods here.

2.2.1 Machine learning reduction

Machine learning reduction consists in creating a model of classification with multiple possible categories that relies on several binary classifiers. In other words, the main goal is to combine the prediction of several binary classifiers in order to make a prediction on an instance that can belong to several categories. Many different methods already exist that only differ on how the binary classifiers are combined to create the multiclass classifier, for example Error Correcting Output Codes, One-versus-All, One-versus-One and Filter Trees among others (Allwein, 2001 [1]).

For example with One-vs-One classifier, for each pair of labels a binary classifier that discriminates between the two labels is trained, at inference an instance is put through all these classifiers and the label that has received the largest number of votes is predicted. Such method has proven good accuracy but at inference it involves to test a classifier for each pair of labels making the complexity quadratic in the number of labels $O(L^2)$, which is way to large for extreme classification.

Binary classification

Let us suppose we are given an input space $\mathcal{X}$ and $\mathcal{Y}$ a set of categories an instance of the input space can belong to. A classification problem is a supervised learning task that consists in, based on a training set $\mathcal{S} = \{(x_i, y_i)\}_{1 \leq i \leq n} \in (\mathcal{X}, \mathcal{Y})^n$ of correctly identified observations (in which each instance x_i belongs to the category y_i), to create a classification rule that allows to predict which category in $\mathcal{Y}$ an unknown instance belongs to.

Binary classification is the simplest case of classification where there are only two possible categories $\mathcal{Y} = \{-1, 1\}$. This is one of the most studied machine learning problem because of its obvious practical interest on his own

and because as we have seen earlier we can use it as building block of more sophisticated multiclass machine learning system.

If we choose an *empirical risk minimisation* point of view, basically binary classification consists in finding the best hypothesis $h : \mathcal{X} \to \{-1, 1\}$ from a set of hypothesis $\mathcal{H}$ that minimizes the average number of disagreements between the prediction $h(x_i)$ and the actual label y_i over the training set $\mathcal{S}$. For $h \in \mathcal{H}$ and $\mathcal{L}$ a loss function that measures the penalty of measuring $h(x_i)$ when the actual label is y_i, we can define the *empirical risk* as,

$$R_{\mathcal{S}}(h) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(h(x_i), y_i)$$

The empirical risk is an approximation of the classifier's expected risk on the distribution P from which the data was drawn. To prevent overfitting we often add a regularization term (Vapnik, 2000 [27]),

$$R_{\mathcal{S}}(h) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(h(x_i), y_i) + \lambda \|h\|$$

where $\lambda > 0$ is a fixed hyper parameter that weigths the importance given to this regularisation term. The goal of the optimisation is to find the hypothesis $h^\star$ in $\mathcal{H}$ that minimizes the *empiracl risk*,

$$h^\star = \arg\min_{h \in \mathcal{H}} R_{\mathcal{S}}(h)$$

Depending on the problem we can use various classes of hypothesis such as linear models, kernel machines or neural networks. In the case of a linear hypothesis, an hyphotesis h is parametrized by a vector w and $h_w(x) = w^T x + b$ where b is a bias term. In this case looking for best hyphotesis $h^\star$ is equivalent to find the vector $w^\star$ that minimizes $R_{\mathcal{S}}(h_w)$ according to the loss function $\mathcal{L}$.

The natural loss function we would like to consider is the *zero-one loss*, that is equal to 0 if $h(x_i) = y_i$ (correct prediction) and 1 otherwise : $\mathcal{L}_{0/1}(h(x_i), y_i) = \mathbf{1}_{\{h(x_i) \neq y_i\}}$. The *zero-one loss* is natural in the sense that it counts the number of missed predictions but it also involves a disadvantage it is not convex (wrt $h(x)$). In order to handle this issue we usually use surrogates of the *zero-one loss* such as *Hinge loss* or *Logistic loss*.

$$\begin{cases} \mathcal{L}_{Hinge}(h(x), y) = \max(1, 1 - y.h(x)) \\ \mathcal{L}_{Logistic}(h(x), y) = \log(1 + \exp(y.h(x))) \end{cases}$$

This two functions are upper-bounds of the *zero-one loss*, it involves that by minimizing this two surrogated losses we can also bound the *zero-one loss*. When the linear hyphotesis is chosen, the Hinge loss and the logistic loss stand for respectively *Support Vector Machines* (SVM) and *Logistic Regression* (LR). To optimize these problems various solutions exist such as stochastic gradient and dual coordinate descent algorithm. These two methods SVM and LR have proven state of the art performances on several datasets, and are widely used either to solve binary problems or to build multiclass classifiers as we will see in the following.

One-versus-Rest Classifier

The One-versus-Rest Classifier also called One-versus-All classifier is one of the simplest approach of multiclass reduction to binary. It consists in, for each category, training one binary classifier that discriminates if an instance belongs to the selected category or to any other one. At inference, all the classifiers are tested and the category whose binary classifier has the largest score (in other word the most confident binary classifier) is predicted. A widely cited paper dealing with this method is (Rifkin and Klautau, 2004b [23]), in which it is empirically shown that One-vs-All method outperform other approaches such as ulti class support vector machines (M-SVM), One-vs-One classifier (one binary classifier for each pair of categories) and Error Correcting Output Codes (ECOC). An other aspect that we can benefit from One-vs-All is that binary classifiers can be trained and tested independently so OVA can be parallelized easily.

Error Coding Output Codes

Error Correcting Output Codes (ECOC) is a method to classify with multiple categories, it has been introduced in (Dietterich and Bakiri, 1995 [12]). The methods consist in associating to each one of the L classes a binary code of lenght d_e. These binary codes or codewords are usually represented as the rows of a coding matrix $M \in \{-1, +1\}^{L \times d_e}$. For each column of the matrix we train a binary classifier to answer the binary classification problem induced by the column. For a training set $\mathcal{S} = \{(x_i, y_i), i \leq n\}$, a given column l induces two subset of $\mathcal{S}$, the subset of postive examples

$\mathcal{S}^+ = \{(x_i, y_i), M_{i,l} = +1\}$ that corresponds to the examples that belongs to either one of the labels that have been coded by a $+1$ in column l. Similarly we have the negative examples $\mathcal{S}^- = \{(x_i, y_i), M_{i,l} = -1\}$ that have been coded by -1. These d_e binary classifiers h_i (also called dichotomizers) are then used to make the final prediction over the L categories. A given test instance x has a predicted codeword $h(x) = (h_1(x), \cdots, h_{d_e}(x))$ and the predicted label for x is the label whose codeword in the coding matrix is the closest $D(x) = \arg\min_l d(h(x), M_{l,.})$. D is called the decoding procedure, it generally uses the hamming distance d_H that counts the number of different bits between two codewords.

2.2.2 Embedding approaches

Nearest Neighbor (NN) approaches are powerful non parametric methods that can split an input space into multiple classification regions with complex non linear boundaries (Bishop, 2006 [8]). They have achieved state of the art performances when the distance metric used is learned, as for large margin nearest neighbor (Winderberg and Saul, 2009 [31]). However, Nearest Neighbor does not scale well to extreme classification because of its linear computational complexity in the number of examples in the training set. Some solutions have been suggested to speed up the nearest neighbor search such as *k-d trees* (Bentley, 1975 [5]) that reduces the search complexity to $O(d \log n)$ where d is the dimension of the input space and n is the size of the training set, at the cost of a small drop of performance. This method has been improved by at the same time learning the metric distance and projecting the data into a space of lower dimension d_e ($d_e << d$). For example, Large margin Component Analysis (Toressani and Chih Lee, 2006 [25]), results in a complexity of $O(d_e(d + n))$ (or $O(d_e(d + \log n))$ if *k-d trees* are used).

Another lead to speed up nearest neighbor is *nearest centroid classifier* (NC). It consists in computing the centroids of each class j :

$$u_j = \frac{1}{\#\{y_i = j\}} \sum_{y_i = j} x_i$$

,

the average vector of all the elements of the training set within a given class. At inference an instance x is assigned to the class wich centroid is the closest, $y = \arg\min_j \|x - u_j\|$. This method also called *Rocchio classifier*

is widely adopted in the text classification community, with a complexity in $O(d\,L)$ where L is the number of classes instead of $O(d\,n)$, this methods scales well to problems with very large data set. Despite this improved complexity, it remains very costly for extreme classification where both d and L are large. Moreover, the high dimensionality of the input space generally leads to poor performances when euclidian distance for example is used for nearest mean search.

The combination of distance metric learning and dimensionality reduction has proven effectivity to reduce the inference complexity of Nearest Neighbor while maintaining a state of the art performance. On the other hand Nearest Centroids allows a complexity reduction up to linear dependance with respect to the number of categories rather than the size of the training set. *Embedding methods* try to take advantage of both previous method. To do so they project (or map) the data and the labels into the same space of low dimension d_e (with $d_e << d$ and $d_e << n$) and then try to locate the projected data as close as possible to the projection of the correct label. The final complexity is then $O(d_e(d + L))$ and can be reduce to $O(d_e(d + \log L))$ using *k-d trees*.

Embedding procedure consists in learning two projection matrices, first $W \in M_{d_e,d}$ that projects the examples and second $V \in M_{d_e,L}$ that projects the labels, the common space of projection is called the latent space. A label y is represented in the original label space by a vector of size L made of zeros except a one at y^{th} position : $\phi(y) = (0, \cdots, 1, \cdots, 0)$. In the latent space an instance x is represented by Wx and a label y by $V\phi(y)$. The main goal of *embeddings* is to learn the two matrices W and V such that in the latent space an example is close to the right label. At inference, the prediction on an instance x is then naturally the label whose representation in the latent space is the closest to the projection of x :

$$f_{embed}(x) = \arg\min_{y} d(Wx, V\phi(y))$$

where d is a similarity measure in the latent space generally the euclidean distance or the inner product. The two matrices can be learned either separatly or jointly. The label embedding matrix V can also be learned using prior information such that the similarity between labels. In (Bengio et Al, 2010 [2]) such similarity information is obtained via the confusion matrix of a previously trained *One-vs-Rest* classifier whereas in (Weinberger and Chapelle, 2009 [30]) the similarity between labels is calculated as the distance between the labels in an existing hierarchy.

Sequence of convex model

In (Weinberger and Chapelle, 2009 [30]) a two steps procedure is suggested to learn the embedding matrices. The first step uses prior information (supposed to be available) to calculate V. Based on a dissimilarity matrix $C \in M_L(\mathbf{R}^+)$ in which C_{ij} corresponds to the dissimilarity between label i and label j. A metric multi-dimensional scaling is solved in order to project similar labels in neighboring regions of the latent space. This problem corresponds to minimize :

$$\sum_{i,j}(\|V_i - V_j\|^2 - C_{ij})^2 \text{ (with } V_i \text{ the } i^{th} \text{ column of } V)$$

It can be easily done, if C defines squared euclidean distances which is the case if the costs are defined as the length of the shortest path between nodes in a hierarchy of labels and then squared.

Once V is found we need to find a proper W that maps each input as close as possible to its label representation, that can be done by setting :

$$W^* = \arg\min_W \sum_i \|V\phi(y_i) - Wx_i\|^2 + \lambda\|W\|_F^2$$

The parameter λ is the weight of the regularization of W, which is necessary to prevent potential overfitting due to high number of parameters in W. W^* is an instance of a linear ridge regression and can be found very accurately by solving with linear conjuguate gradient for each row of W.

At inference, prediction on an example x is naturally done by finding the label whose representation in the latent space is the closest to the projection of x :

$$h(x) = \arg\min_{i \leq L} \|Wx - V\phi(i)\|$$

Joint non convex model

A second possible approach is to find W and V at the same time without using prior information. Even though such procedures poses an "egg and chicken" problem as pointed out by (Weinberger and Chapelle, 2009 [30]). A joint optimazation procedure has been suggested in (Bengio et al., 2010, [2]) :

$$\text{minimize} : \gamma\|W\|_{FRO} + \frac{1}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \begin{cases} (Wx_i)^T V\phi(y_i) \geq (Wx_i)^T V\phi(y_j) + 1 - \xi_i, \forall i \neq j \\ \|V_k\| \leq 1, \forall k \leq L \\ \xi_i \geq 0, \forall i \leq n \end{cases}$$

In this case no prior information is used and the scheme to minimize in non-convex because of the first constraint. Training can be efficiently performed by using stochastic gradient descent with randomly initialized weights. At inference a distinct similarity measure can be used : $d(a, b) = a^T b$ which is the dot product in the latent space. Despite the apparent difficulty of this problem, it has shown better performances than the previous convex one in real world extreme classification problems (Bengio et al., 2010, [2]).

2.3 Hierarchical Approaches

2.3.1 Introduction

Given $x \in \mathcal{X}$ an instance of an input space, classification tasks aim to assign x to its relevant caterogy among several possible categories $\mathcal{Y}$. In order to decrease the complexity w.r.t the number of possible labels a natural lead is to apply *divide and conquer* strategies. Hierarchical classifiers (Liu et al., 2005 [19]; Silla and Freitas, 2011, [24]) are one of the main *divide and conquer* approaches adopted when dealing with a large number of categories. This popularity is due to both accuracy and effeciency reasons. An other particularity that makes hierarchical methods such popular is that most of the real world extreme clasification problems come with a taxonomy that structures the labels. For example DMOZ which is a comprehensive directory of the Web that comes with a strong hierarchical backbone organization. We can also quote the MESH directory that organizes medical subjects in a hierarchy and that is then used by PubMed to index more than 24 millions citations in biomedical litterature.

First, exploiting hierarchical information can lead to important perfomance improvements in classification (Bennett and Nguyen, 2009 [4]; Koller and Sahami, 1997 [16]; Weigend et al., 1999 [29]). Moreover, flat approaches are most of the time computationnaly prohibitive for real world applications

because they don't scale to problems with a large number of labels. On the other hand, hierarchical methods can easily decrease the time complexity up to logarithmic dependance w.r.t the number of classes for example by using balanced trees (Beygelzimer et al., 2009b [6]; Deng et al., 2011 [11]). It is then easy to understand why these kind of approaches are getting very popular in a context of *extreme classification*.

A label tree is a tree $T = (\mathbf{N}, \mathbf{E}, \mathbf{F}, \mathbf{L})$ with $N + 1$ nodes indexed $\mathbf{N} = \{0, \cdots, n\}$, a set of edges $\mathbf{E} = \{(p_1, c_1), \cdots, (p_{|E|}, c_{|E|})\}$ which are ordered pairs of parent and child node indices, a set of node predictors $\mathbf{F} = \{f_1, \cdots, f_N\}$ and labels sets $\mathbf{L} = \{l_0, \cdots, l_N\}$ where l_k indicates the set of labels to which an instance should belong if it arrives at the node k and progress from generic to specific along the tree. The root set contains all the labels $|l_0| = L$ and each child label set is a subset of its parent label set with $l_p = \cup_{(p,c)\in E}\, l_c$. Generally they are two kinds of structures, *regular trees* in which each node has a single parent, or *Directed Acyclic Graphs (DAG)* where many parents are allowed for a single node.

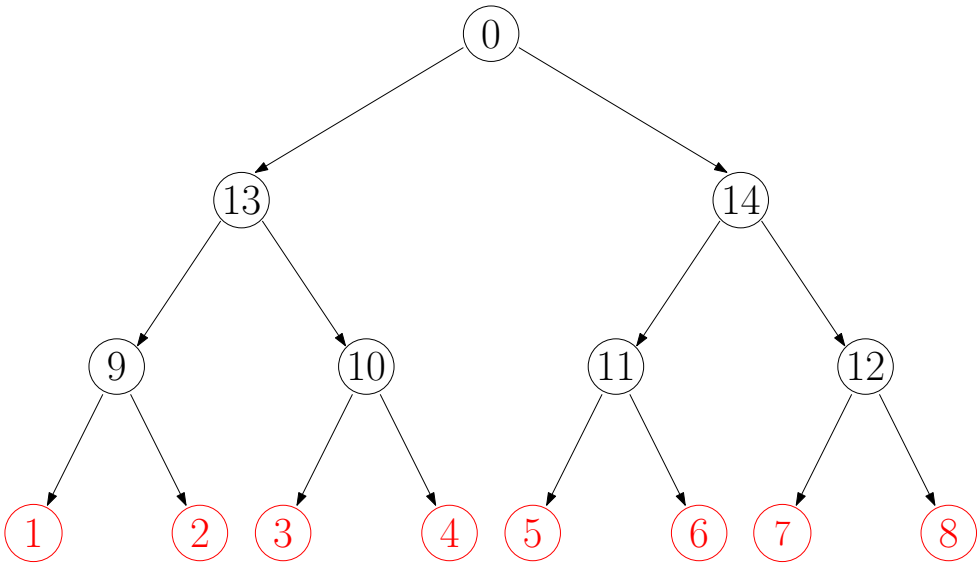


Figure 2.1: Example of hierarchical classifier (binary). Each node (besides the root) is associated with a classifier f_i. They are eight possible labels $l_0 = \{1, \cdots, 8\}$ and for example $l_{14} = \{5, 6, 7, 8\}$ and $l_9 = \{1, 2\}$.

Each node predictor f_c is trained in order to predict whether or not the example x belongs to the set of labels l_c. At inference, the prediction is performed by applying algorithm 1. The process is a *depth first search (DFS)*

based on the scores of the local classifiers. It starts at the root node and at each round selects among the current node's children the one whose associated classifier has the largest score (the most confident child). The process is repeated until a leaf is reached and the label corresponding to this leaf is predicted.

Algorithm 1: Hierarchical Prediction Algorithm

Input: Test example x
Output: The predicted label
Let $p = 0$;
repeat
  $p \leftarrow \text{argmax}_{c,\ (p,c)\in E}\ f_c(x)$;
until $|l_p| = 1$;
return l_p

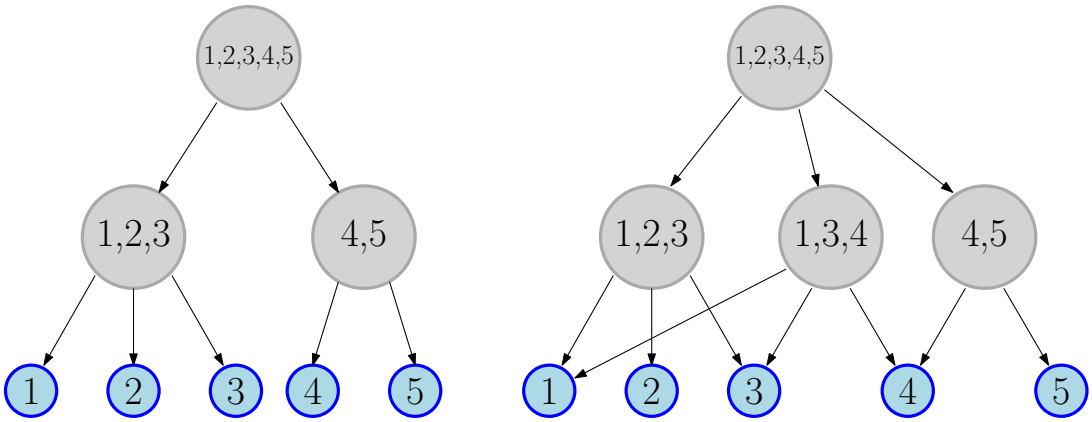
In the following we will describe the main methods concerning hierarchical classification suggested in the litterature. These approaches can be split into three groups. First find algorithms to learn descriminative hierarchies from the data regarless of the internal node's classifiers. The second line of work is interested in learning performing classifiers based on a given hierarchy. The last family of methods consists in learning jointly the structure of the hierarchy and the node's classifiers.

2.3.2 Hierarchical Structure Learning

Most of the classification models used when dealing with several possible categories are flat models such as M-SVM (Weston and Watkins, 1998 [32]) or One-vs-Rest (Rifkin and Klautau, 2004 [23]) to name a few, that have linear to quadratic complexity w.r.t the number of labels. Such complexities are restrictive for *Extreme Classification* where problems can have up to hundred of thousands categories. In order to apply *divide and conquer* strategies, authors have started to create discriminatives hierarchies (Chen et al., 2004 [10]; Liu et al, 2005 [19]; Vural and Dy, 2004 [28]; Zhang et al., 2010 [36]). Methods that have been proposed ever since can be very different, for example (Marszalek and Schmid, 2007 [20]) learn a class hierarchy by exploiting the semantics of the classes and some additional knowledge about interclass relationships such as Wordnet. Another approach has been introduced in (Griffin and Perona, 2008 [14]) it relies on a *recursive top-down partioning* of the set of classes to build hierarchies conversely in (Zhigang et al., 2005 [37])

a *bottom up agglomerative clustering* is applied. Another lead has been suggested in (Liu et al., 2005 [18]) to use a method based on K-means clustering whereas in (Zhang et al., 2010 [36]) the structure of the classes hierarchy is randomly sampled and then evaluated by cross-validation.

Among these different approaches, (Bengio et al., 2010 [2]; Griffin and Perona, 2008 [14]) have empiracally proven the superiority of *top-down* approaches over the other ones. Therefore, the discussion concerning *Hierchical structure learning* is not really how the structure is learned but what kind of structure allows the best performing method in terms of generalization performances and inference speed. Two main groups of structures are put in competition, *tree structured class hierarchies* in which each child has one and only one parent and *Directed Acyclic Graphs (DAG)* in which a child can have several parents, see figure 2.2



(a) Tree structure (b) Directed Acyclic Graph

Figure 2.2: Hierarchical structures

Spectral Clustering

The goal of clustering algorithm is to split a group of items into multiple subgroups or clusters such that items within the same subgroup are similar to each other. The similarity between items is defined in terms of distance measure. For example K-means algorithm aims to optimize the within cluster sum of square distances. Spectral clustering algorithms are powerful alternatives to K-means. Spectral clustering consits in first representing the set of items as a graph in which the vertices N stand for the items and the edges E are associated to a weight that represents the similarity between

the items. The similarity matrix is denoted $W = (w_{ij})_{(i,j)\in N}$. The similarity matrix can be calculated in different ways and will result in different type of graph. A first option is the k-nearest neighboor approach where each node of the graph is connected to only his k-nearest neighboors (acording to a given ditance measure) with weight 1 and all the other weights set to 0, W is then a more or less sparse matrix made of zeros and ones, $(w_{ij} = 1, \forall (i,j) \in E)$, not necessarily symetric with k ones per column. On the other hand we can obtain a dense graph by using a gaussian kernel : $w_{ij} = \exp(\|u_i - u_j\|^2/2\sigma^2)$, where u_i stands for item associated to node i.

Once the graph is defined, the next step consists in spliting the items into different clusters such that the sum of the weigts edges linking vertices belonging to different clusters is minimized. For two clusters A and B, this quantity is called the *cut* : $cut(A,B) = \sum_{i\in A, j\in B} w_{ij}$. (Ng et al., 2002 [22]) suggest to minimize instead the normalized cut which leads to more balanced clusters. The corresponding objective function is :

$$J_N = \frac{cut(A,B)}{Vol(A)} + \frac{cut(A,B)}{Vol(B)}$$

where $Vol(A) = \sum_{i\in A, j\in N} w_{ij}$ is the volume of the cluster A. This problem can equivalently be formulated as an eigenvalue problem : $(D-W)y = \lambda Dy$ where D is the diagonal matrix with $d_{ii} = \sum_j w_{ij}$ and y is the indicator vector of vertices belonging to clusters A ($y = 1$) and B ($y = -1$). This problem is actually NP-hard and in order to solve it we need to relax the binary condition to a continuous condition : $y \in [-1,1]^{|N|}$. The solution is then obtained by the second eigenvector of the normalized laplacian : $L = D^{-\frac{1}{2}}(D-W)D^{-\frac{1}{2}}$.

Finally we need to choose to which cluster associate each vertice from the solution of the previous problem. If the second eigenvector y^* is considered, the clustering is obtained thanks to a simple thresholding rule : $A = \{i : y_i^* > 0\}$ and $B = \{i : y_i^* < 0\}$.

Learning Class Hierarchies

So far the most promising approaches to learn hierarchical structures as suggested in the litterature are based on recursive partitioning of the set of classes using spectral clustering (Bengio et al., 2010 [2]; Chen et al., 2004 [10]; Griffin and Perona, 2008 [14], Marszalek and Schmid, 2008 [21]). The

research effort is now focused on finding the best form of hierarchical structures.

Tree structured class hierarchies The goal is to partition the categories into separable clusters. In order to apply spectral clustering methods where the categories stand for the items (nodes of the graph), the similariaty matrix used is the symmetrized confusion matrix between the classes (Godbole et al., 2002 [13]). The underlying idea is that to evaluate how much two categories are similar, a lead is to count how many times an instance belonging to one of the two categories is misclassified into the other category. For example the confusion matrix can be obtained from a previously trained surrogate classifier such as One-vs-Rest. Most of the time the confusion matrix is obtained by averaging different confusion matrices and using k-fold cross validation to guarantee more stability. For a given depth and branching factor for the hierarchy, (Bengio et al., 2010 [2]) recursively solve graph cut problems (with a previously built affinity matrix) until the desired shape is obtained. Another approach proposed by (Griffin and Perona, 2008 [14]) who, instead of fixing the branching factor, use *selt-tunning spectral clustering* (Zelnik-Manor and Perona, 2004 [35]) to automatically find the number of clusters at each step.

DAG structured class hierarchies To create DAG structures, conversely to trees, spectral clustering is directly applied on the instances of the training set (Marszalek and Schmid, 2008 [21]). At each step the examples $S = \{(x_i, y_i), i \leq n\}$ are partitioned into two clusters R and L. The labels are then distributed into two groups $\mathcal{R}$ and $\mathcal{L}$, following a simple rule, whenever a class c has one of its instance belonging to a cluster (cluster R for example), then c is associated to the corresponding group of classes ($\mathcal{R}$ in the same example) : $\mathcal{R} = \{y : \exists(x, y) \in R\}$ (respectively $\mathcal{L} = \{y : \exists(x, y) \in L\}$). Obviously defined this way $\mathcal{R}$ and $\mathcal{L}$ are very likely to overlap. In practice, (Marszalek and Schmid, 2008 [21]) suggest to relax the division rule in order to obtain shallower hierarchies, by considering that a class is assigned to a cluster only if a large enough proportion of the instances within the class belong to the corresponding cluster.

2.3.3 Discriminative Models Learning

Independant Optimization of Models : Pachinko Machines

Pachinko Machines (Liu et al., 2005 [19]; Yang et al., 2003 [34]) is the most simple case of hierarchical classification and corresponds to the early

work in the field. It consists in learning independently classifiers at each node of a given hierarchy. It is simple in the sense that the hierarchy is only used to partition the training data in order to learn local classifiers. At each interior node $\eta \in \mathbf{N}$, f_η is trained considering the positive data as the subset of training examples that belong to one of the leaf descending from η : $\mathcal{S}^+ = \{x_i : y_i \in l_\eta\}$ and the negative values the examples that belong to any label descending from a sibling of η : $\mathcal{S}^- = \{x_i : y_i \in \bigcup_{\substack{(p_\eta,c)\in E \\ c\neq\eta}} l_c\}$ where p_η is the father node of η.

(Bengio et al., 2010 [2]) suggested to use a *Support Vector Machine* form of objective function to make the optimization. Denoting $b_j(x)$ the index of the best node in the hierarchy at depth j and $C_j(y) = 1$ if $y \in l_j$ and -1 otherwise , we have :

$$R_{emp}(h) = \frac{1}{n}\sum_{i=1}^{n} \max_{j\in B(x_i)} \mathbf{1}(y_i \notin l_j) \leq G_h = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{D(x_i)} \max\left(1, 1 - C_j(y_i).f_j(x_i)\right)$$

where $B(x) = \{b_1(x), \cdots, d_{D(x)}(x)\}$ and $D(x)$ is the depth of the final prediction of x. This obective function ends up in the followin optimization problem :

$$\sum_{j=1}^{N}\left(\gamma\|w_j\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_{ij}\right) \text{ s.t } \forall(i,j), \begin{cases} C_j(y_i)f_j(x_i) \geq 1 - \xi_{ij} \\ \xi_{ij} \geq 0 \end{cases}$$

Since the parameters of the local classifiers w_j do not interact in the objective, the optimization problem can be decomposed, therefore the local classifiers can be trained in parallel. This property is very desirable and explains the success of these models in the early stages of hierarchical classification.

Joint Optimization of Models

On the opposite side from *Pachinko Machines* some authors have recently proposed to learn the hierarchical classifier as a unique classifier and learn all its parameters jointly. The main idea underlying this approach is to make individual errors (at a given node) impact the updates of all the parameters of each local classifier. When learning a hierachical problem, the set of parameters $w = (w_i)_{i\leq N}$ of the model is the solution of the generic problem :

$$\text{Minimize : } \arg\min_{w} \mathcal{R}(w) + C \times \mathcal{L}_{emp}(w)$$

where $\mathcal{L}_{emp}$ is the empirical loss on the training set and $\mathcal{R}(w)$ is a regularization term and C is a constant hyper-parameter controlling the trade-off between the two terms. For the *Pachinko Machine*, both the regularization term and the empirical loss are fully decomposable and no interaction exists between the parameters of the local classifier, however inter-dependence can be introduced.

A first framework has been introduced by (Cai and Hofmann, 2004 [9]) as a generalization of the classical multiclass support vector machines :

$$\text{Minimize : } \gamma \sum_{j=1}^{N} \|w_j\|^2 + \frac{1}{N} \sum_{i=1}^{n} \xi_i$$

$$\text{s.t } \forall i, \begin{cases} F(x_i, y_i; w) - F(x_i, c; w) \geq 1 - \xi_i \ (\forall c \neq y_i) \\ \xi_i \geq 0 \end{cases}$$

where $F(x, y; w)$ is the linear discriminant function corresponding to the class y and is expressed as :

$$F(x, y; w) = \langle w, \Phi(x, y) \rangle = \sum_{\eta=1}^{N} \lambda_\eta(y) \langle w_\eta, x \rangle$$

In this formula, $\Phi(x, y) = \Lambda(y) \otimes x$ ($\otimes$ stands for the tensor dot product) where $\Lambda(y) = (\lambda_\eta(y))_{\eta \leq N}$ is a vector made of zeros and ones such as $\lambda_\eta(y) = 1$ if $y \in l_\eta$ (label y belongs to the descendance of node η) and $\lambda_\eta(y) = 0$ otherwise, see on figure 2.3. In other words, classes are represented by binary attribute vectors of size the number of nodes in the hierarchy (root not counted). Similarly, if the structure considered is a DAG then class c attribute vector is made of ones for every node that has c in its descendance.

The main idea behind this approach is that the attribute vectors capture semantic relation between the classes of the hierarchy. For example, on figure 2.3, labels 1 and 3 have closer attribute vectors than 1 and 7 that are far away from each other in the hierarchy.

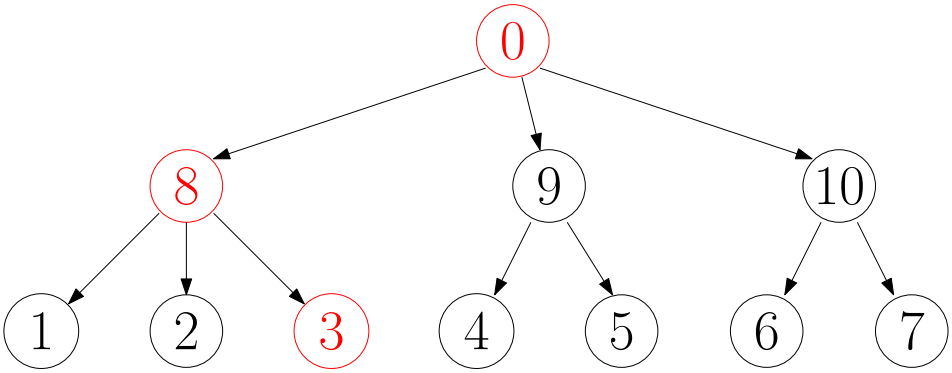


Figure 2.3: Illustrating the use of class attributes reflecting the hierarchical structure. Class attribute vectors have the same dimension as the number of nodes in the hierarchy (root not counted). For label 3 the components corresponding to nodes 8 and 3 are set to one since they are on the path from the root to node 3 : $\Lambda(3) = (0, 0, 1, 0, 0, 0, 0, 1, 0, 0)$.

Sequential Learning of Models

As seen earlier, *Pachinko Machines* are appealing because training is easily parallelizable since the local classifiers are totally independent. However, not taking into consideration the semantic relashionships between classes the classes decrease the capacity of a hierarchical classifier to reach state of the art performances. On the other hand, jointly learned models have achieved record breaking performances in many studies (Bengio et al., 2010 [2]; Cai and Hofmann, 2004 [9]) but training them efficiently is a challenging task moreover in Extreme Classification because of all the interdependencies between the parameters that make the parallelisation of the learning process almost impossible. *Sequential Learning* is an intermediate framework of research that leverage the available hierarchical information by training local classifiers sequentially. This approach allows to parallelize the training and make it more feasible.

Chapter 3

Label Taxonomy based Neural Network

3.1 Principle

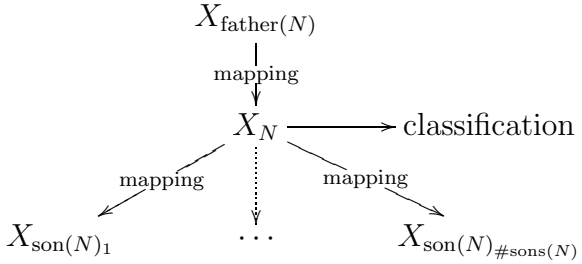
Our goal is to build a method that allows, fast inference and high accuracy. To do so we propose a method called Label Taxonomy based Neural Network. This approach is a hierarchical approach based on an existing taxonomy of labels. In our case we are given a data set of examples that belongs to one of several possible categories, this data set also provides a hierarchy (or taxonomy) that organizes the labels. The taxonomy presents a particularity, labels are the leaves of the hierarchy. We can benefit from this particularity to create a hierarchical classifier over all the possible labels by creating a classifier at each interior node of the taxonomy that classify an example into the right son's branch. The classification over all the labels is then naturally completed by following the sequence of answers given by the interior node's classifiers until we reach a leaf, the final prediction made is the label corresponding to the leaf we reached.

Contrary to a regular hierarchical model in which the classifiers at each interior node are trained over the input space that has a very large dimension (up to one milion features), our method proposes to first map the input space into low dimensional spaces and then train the different node's classifiers over these mapped spaces. The benefits of such a method are obvious, first using a hierarchical method drastically decreases the inference's complexity wrt the number of categories since the number of classifiers we need to test to make a prediction is upper bounded by the maximal depth of the taxonomy. In comparison the One-vs-All classifier need to test as many classifiers as the

number of categories to make a prediction. On the other hand mapping the input space into low dimensional spaces avoids to train and test the interior node's classifier over the input space that has a very large dimension, it follows a huge economy of computational ressources at training and inference.

Our approach proposes to successevely remap the input space from node to node by following the hierarchy from top to bottom. Basically, the input space of dimension d is first mapped into a space of dimension d_e with $d_e << d$, this mapping will be used by the root's classifier at training and inference. This first mapping is then remap into new spaces of dimension d_e or lower, at each root's son so their classifier can be trained on their own mapped space. This new mappings are then remaped into new spaces of low dimension following the hierarchy and so on.

For example, if N is a node in the hierarchy and X is an instance of the input space that has been mapped into $X_{\text{father}(N)}$ at node father(N) then here is the scheme of mapping at node N :



This approach presents several assets. First, the input space is only mapped once at the root, the next mappings at regular nodes are done from a low dimensionsional space to a low dimensional space, so we need to deal with a space of large dimension only once, it ensues a large saving of computational ressources. Second, since we have a chain of successive mappings, the highest nodes are concerned by their own classification problem but also by all their descendants ones. So the highest nodes' mappings will take out all the information they need to answer their own classification problem but also take out the information their descendants need. Third, by introducing non linearity properties in the model (we will see later how), the fact of remapping gives additional liberties to the model, it results more flexibility to fit the problem.

3.2 Model

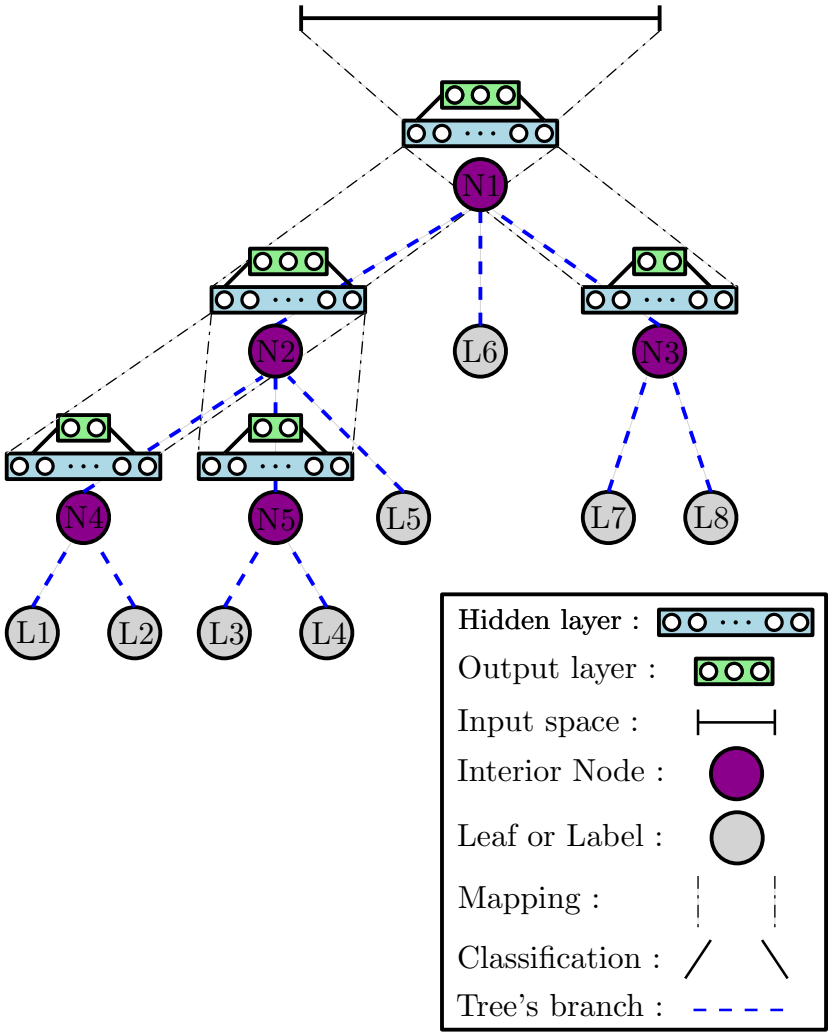
To formalize our approach, an interresting lead is to use Neural Networks, indeed the successive mappings reminds the successive layers of a Neural Network. Moreover a competitive advantage of using Neural Networks is to introduce some non linearity in the model. By putting our mappings through a sigmoid or an hyperbolic tangent function, we give more flexibility and a better chance to the model to fit the problem.

In terms of Neural Network, our approach consists in successive hidden layers of low dimension, one hidden layer at each interior node of the hierarchy, it corresponds to the successive mapings. The leaves of the taxonomy don't need an hidden layer since there is no classification to complete at these nodes. Each node's hidden layer is fully connected to the hidden layer of the node's father, and the root's hidden layer is fully connected to the input space. The number of units in each hidden layer (very low in comparison with the dimension of the input space) is arbitrarly fixed when creating the architecture.

The architecture we just described is not complete and only corresponds to the mapping part of the approach, each mapping is represented by an hidden layer. To handle the classification, we add an output layer at each interior node with as many units as the number of sons of the corresponding node. This output layer is fully connected to the node's hidden layer (see on figure 3.1) and it will behave like a One-vs-All classifier that discriminates between the node's son, except that it will be trained over the space of the corresponding hidden layer instead of the input space.

More precisely the output layers are made of several units, one for each son of the corresponding node, these units correspond to the activity of each son and take values between 0 and 1. Basically when we focus on an interior node the different sons define non overlappings subsets of labels (the ones descendind from a given son), the activities will measure the confidence the node's classifiers has to classify an instance into each one of this subset of labels. When we put an instance from the input space through the neural network, all the units in the output layers activate. Our goal is, for each node, to maximize the activity of the correct son (the branch to follow to reach the correct label) and minimize the activity of the others sons. At inference, the final classification is performed by following the branchs with the largest activities until we reach a leaf that is the label predicted, as we

Figure 3.1: Label Taxonomy based Neural Network



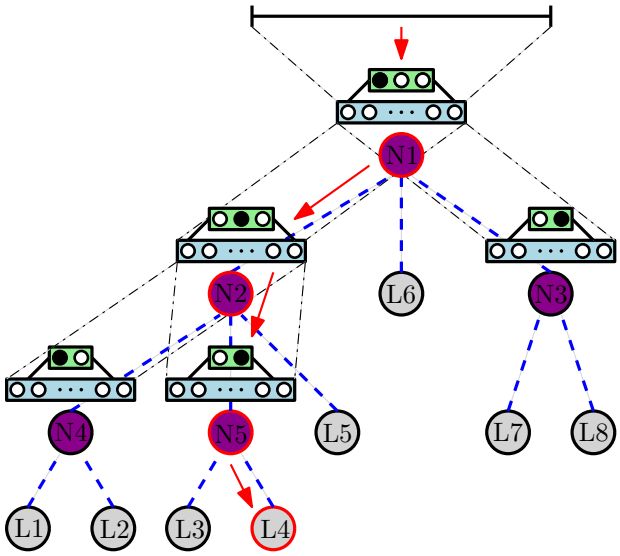
can see on figure 3.2.

3.3 Optimization

First the notations, if X is an instance in the input space :

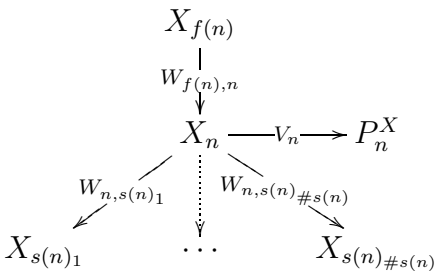
- $\mathcal{N} = \{n_1, \cdots, n_N\}$ correspond to the set of the N interior nodes (not leaf) in the hierarchy. They corresponds to the nodes where a classifier is needed. We also suppose that n_1 refers to the root of the taxonomy.
- For $n \in \mathcal{N}$, we denote $f(n)$ the father of n.

Figure 3.2: Example of Inference. Black units correspond to the unit with the largest value in the output layer. Here L4 is predicted



- For $n \in \mathcal{N}$, we denote $s(n)$ the sons' list of n. $\#s(n)$ refers to the number of sons of node n and $s(n)_i$ refers ot the i^{th} son of n ($i \leq \#s(n)$).
- X_n corresponds to the mapping (hidden layer) of X at node n, and we consider that $X_{f(n_1)} = X$ the input instance.
- For all n and m such as n is father of m, we denote $W_{n,m}$ the mapping matrice from hidden layer n to m.
- At node n, we denote P_n^X the output layer of dimension $\#s(n)$ that corresponds to the activities of each branch wrt the instance X.
- V_n corresponds to the classification matrix at each node, connecting the hidden layer to the output layer.

Here is the scheme at node n :



And we have,

$$\begin{cases} X_n = \tanh(W_{f(n),n} X_{f(n)}) \\ P_n^X = \sigma(V_n X_n) \end{cases} \tag{3.1}$$

where σ corresponds to a sigmoid function :

$$\sigma : \left| \begin{array}{ccc} \mathbf{R} & \to & [0\,,1] \\ x & \longmapsto & \frac{1}{1+e^{-x}} \end{array} \right.$$

and tanh is the usual hyperbolic tangent function. Note that we commited a misuse of language in equation (3.1), it has to be understood as the function applied to every coordinates of the vector.

The choice of using an hyperbolic tangent function and a sigmoid is to add non linearity properties to the model and make it more flexible. More precisely to keep mappings that take positive and negative values we have chosen an hyperbolic tangent to map and a sigmoid to calculate the activity units since we want the activities to be positive values.

We have several parameters (or weights) to optimize, at each interior node we look for the best W and the best V. Given a training set $\mathcal{S}$, we propose to minimize the next problem :

$$\mathcal{L}_{W_{.,.},V_.} = \frac{1}{2} \sum_{(X,Y)\in\mathcal{S}} \sum_{n\in\mathcal{N}(Y)} \lambda_n \|P_n^X - n(Y)\|_F^2 \text{ wrt } W_{.,.} \text{ and } V_.$$

where, for a label Y :

- $\mathcal{N}(Y)$ corresponds to the family of nodes in the taxonomy that have Y in their descendants.

- for $n \in \mathcal{N}(Y)$, $n(Y)$ is a vector made of 0 except a 1 at the position corresponding to the branch of Y. $n(Y)$ has dimension $\#s(n)$
- λ_n are hyper-parameters that we fix before optimisation to give more or less weight to some nodes in the loss calculation. For example, we can give more importance to the loss due to the root by choosing λ_{n_1} large.

Easily speaking, for an example X that belongs to label Y, at each node concerned by Y (node that has Y in his descendance), we try to maximize the activity of the branch that we have to follow to reach Y and to minimize the activity of the other branchs.

The optimisation is completed by stochastic gradient descent :

Algorithm 2: Stochastic gradient descent algorithm

Data: Training set $\mathcal{S}$
Result: $W_{.,.}$ and $V_.$ that minimize $\mathcal{L}_{W_{.,.},V_.}$

Randomly initiate $W_{.,.}$ and $V_.$;
while *Convergence criterion is not met* **do**
  Randomly pick an example $(X, Y) \in \mathcal{S}$;
  `/* Calculate the gradient step with`
  $\mathcal{L}^{X,Y}_{W_{.,.},V_.} = \sum_{n \in \mathcal{N}(Y)} \lambda_n \|P^X_n - n(Y)\|^2$ `*/`
  $step_W \longleftarrow \partial \mathcal{L}^{X,Y}_{W_{.,.},V_.} / \partial W_{.,.}$;
  $step_V \longleftarrow \partial \mathcal{L}^{X,Y}_{W_{.,.},V_.} / \partial V_.$;
  `/* Update` $W_{.,.}$ `and` $V_.$ `with learning rate` η `*/`
  $W_{.,.} \longleftarrow W_{.,.} - \eta.step_W$;
  $V_. \longleftarrow V_. - \eta.step_V$;
return $W_{.,.}$ *and* $V_.$

3.4 Backpropagation algorithm

Introduction

The main point of the stochastic gradient algorithm is that we need to calculate $\partial \mathcal{L}^{X,Y}_{W_{.,.},V_.} / \partial W$ which is the gradient of the loss function for a given

example. Moreover, this calculation has to be done at each step of the descent, and the descent can have up to millions of steps (depending on the size of the training set). A time effective method widely used when dealing with neural networks is the backpropagation algorithm. It consists in finding a recursive formula that links the gradient's coordinates by taking advantage of the structure of successive layers. Then to use this formula to successively calculate all the coordinates of the gradient.

A particularity of our model is that contrary to regular neural networks in which there is only one output layer at the last layer, in our model we have outputs at each layers of the neural network so we can not use the regular backpropagation algorithm. In our case, a first solution to calculate the gradient would be for each output layer to consider the neural network that has the same architecture as our basic network but with all the output layers removed except the selected one. Then compute the gradient for each one of these networks, since they have only one output layer we can use a regular backpropagation algorithm to make the calculation. Finally by summing all these intermediate gradients we obtain the global gradient.

The main issue with this naïve solution is that we make extra matrices multiplications. For example, to calculate each one of the intermediate gradients we need one multiplication by the matrix that makes the mapping from the input space to the first hidden layer, this matrix is huge (the input space as a very large dimension) and each multiplication is time consuming. In this section, we propose to find a backpropagation algorithm that suits our problem and avoid to make extra matrices multiplications.

Notations

To make the next proof clearer we will simplify the notations. For $(X, Y) \in \mathcal{S}$, we denote $\mathcal{N}(Y) = \{1, \cdots, n_Y\}$ the ordered sequence of nodes to cross in the hierarchy to reach the label Y, 1 corresponding to the root of the taxonomy. We will also denote $W_k = W_{k-1,k}$, $\forall k \in \mathcal{N}(Y)$.

The corresponding mapping sequence can be then written :

$$\begin{cases} X_0 = X \\ X_n = \tanh(W_n X_{n-1}), \forall n \in \{1, \cdots, n_Y\} \end{cases}$$

and,

$$\mathcal{L}^{X,Y}_{W_{.,.},V_.} = \frac{1}{2}\sum_{n=1}^{n_Y} \lambda_n \|\sigma(V_n X_n) - v_n^Y\|_F^2$$

Calculation of $\partial \mathcal{L}^{X,Y}_{W_{.,.},V_.}/\partial W_n$

In the following, for any function f considered $\mathbf{d}f|_x$ relates to the matrix of the differential of f evaluated in x.

Step 1 With $W = (W_1, \cdots, W_{n_Y})$ and $H = (H_1, \cdots, H_{n_Y})$, the Taylor formula applied to X_n at the first order is,

$$X_n(W+H) = X_n(W) + \mathbf{d}\tanh|_{X_n}\left[H_n X_{n-1}(W) + W_n\, \mathbf{d}X_{n-1}|_W\, H\right] + o(H)$$

And we obtain the next recursive formula on $\mathbf{d}X_n|_W$,

$$\mathbf{d}X_n|_W\, H = \mathbf{d}\tanh|_{X_n} H_n X_{n-1} + \mathbf{d}\tanh|_{X_n} W_n\, \mathbf{d}X_{n-1}|_W\, H,\ \forall H$$

Using the recursivity, it is then easy to show that,

$$\mathbf{d}X_n|_W\, H = \sum_{k=1}^{n} B_k^n H_k X_{k-1} \tag{3.2}$$

where,

$$\begin{cases} B_n^n = \mathbf{d}\tanh|_{X_n},\ \forall n \in \{1, \cdots, n_Y\} \\ B_k^n = B_n^n W_n B_{n-1}^{n-1} \cdots B_{k+1}^{k+1} W_k B_k^k,\ \forall k < n \end{cases}$$

Step 2 If $l_n = \frac{\lambda_n}{2}\|\sigma(V_n X_n) - v_n^Y\|_F^2$, it follows from step 1 that,

$$\begin{aligned} \mathbf{d}l_n|_W\, H &= \lambda_n(\sigma(V_n X_n) - v_n^Y)^T\, \mathbf{d}\sigma|_{V_n X_n} V_n\, \mathbf{d}X_n|_W\, H \\ &= \sum_{k=1}^{n} A_n B_k^n H_k X_{k-1} \text{ by (3.2)} \\ &= \mathrm{Tr}\left(\sum_{k=1}^{n} X_{k-1} A_n B_k^n H_k\right) \text{ (trace applied to a scalar and } \mathrm{Tr}(AB) = \mathrm{Tr}(BA)) \end{aligned} \tag{3.3}$$

with $A_n = \lambda_n(\sigma(V_n X_n) - v_n^Y)^T \, \mathbf{d}\sigma|_{V_n X_n} V_n$ where $.^T$ is the transposition operator for matrices.

We can finally conclude,

$$\begin{aligned} \mathbf{d}\mathcal{L}^{X,Y}_{W_{.,.},V_.}\Big|_W H &= \sum_{n=1}^{n_Y} \mathbf{d}l_n|_W \, H \\ &= \mathrm{Tr}\left(\sum_{n=1}^{n_Y}\sum_{k=1}^{n} X_{k-1} A_n B_k^n H_k\right) \\ &= \mathrm{Tr}\left(\sum_{k=1}^{n_Y} X_{k-1} T_k H_k\right) \end{aligned} \tag{3.4}$$

where we denote $T_k = \sum_{n=k}^{n_Y} A_n B_k^n$

and by definiton of the gradient,

$$\partial \mathcal{L}^{X,Y}_{W_{.,.},V_.} / \partial W_k(W) = (X_{k-1} T_k)^T, \, \forall k \in \{1, \cdots, n_Y\}$$

Step 3 Our point here is that from now on the calculation of $\partial \mathcal{L}^{X,Y}_{W_{.,.},V_.} / \partial W_k$ relies on the calculation of T_k and as we will see, the calculation of T_k can be done recursively.

If $k < n_Y$, we can write :

$$T_k = \sum_{n=k}^{n_Y} A_n B_k^n = A_k B_k^k + \sum_{n=k+1}^{n_Y} A_n B_k^n$$

and, since we have $B_k^n = B_{k+1}^n W_k B_k^k$, it comes,

$$T_k = A_k B_k^k + \sum_{n=k+1}^{n_Y} A_n B_{k+1}^n W_k B_k^k = (A_k + (\sum_{n=k+1}^{n_Y} A_n B_{k+1}^n) W_k) B_k^k$$

We can finally conclude on our recursive formula,

$$\begin{cases} T_{n_Y} = A_{n_Y} B_{n_Y}^{n_Y} \\ T_k = (A_k + T_{k+1} W_k) B_k^k, \, \forall k < n_Y \end{cases}$$

Conclusion To summarize,

$$\partial \mathcal{L}^{X,Y}_{W_{.,.},V_.} / \partial W_k = (X_{k-1} T_k)^T, \, \forall k \in \{1, \cdots, n_Y\}$$

with,

$$\begin{cases} T_{n_Y} = A_{n_Y} B^{n_Y}_{n_Y} \\ T_k = (A_k + T_{k+1} W_k) B^k_k, \, \forall k < n_Y \end{cases}$$

where, $\forall n \in \{1, \cdots, n_Y\}$,

$$\begin{cases} B^n_n = \mathbf{d}\tanh|_{X_n} \\ A_n = \lambda_n (\sigma(V_n X_n) - v^Y_n)^T \, \mathbf{d}\sigma|_{V_n X_n} V_n \end{cases}$$

It is called a backpropagation algorithm because as we can see the recursive formula is actually a backward recursive formula. If we think in the Neural Network based on the taxonomy, it means that we calculate the coordinates of the gradient node by node starting from the node that is the closest from the leaves and going backward finishing by the root.

To conclude on our basic goal, we can see that by using this algorithm, we only multiply once by each matrix W_k. The same occurs with all the A_k and B^k_k that we need to calculate many times with the naive method presented in introduction. Reducing the number of matricial operations drasticly speed up the calculation of the gradient making each step of the stochastic descent up to ten times faster than the naive method.

Algorithm 3: Backpropagation algorithm

Result: Calculation of $T = [T_1, \cdots, T_{n_Y}]$

`/* Initialization */`

$A \longleftarrow \lambda_{n_Y}(\sigma(V_{n_Y} X_{n_Y}) - v_{n_Y}^Y)^T \, \mathbf{d}\sigma|_{V_{n_Y} X_{n_Y}} \, V_{n_Y}$;

$B \longleftarrow \mathbf{d}\tanh|_{X_{n_Y}}$;

$T[n_Y] \longleftarrow AB$;

`/* Backpropagation */`

for k *from* $(n_Y - 1)$ *to 1* **do**

  `/* Compute` A_k `and` B_k^k `*/`

  $A \longleftarrow \lambda_n(\sigma(V_k X_k) - v_k^Y)^T \, \mathbf{d}\sigma|_{V_k X_k} \, V_k$;

  $B \longleftarrow \mathbf{d}\tanh|_{X_k}$;

  `/* Compute` $T[k]$ `thanks to` $T[k+1]$ `*/`

  $T[k] \longleftarrow (A + T[k+1]W_k)B$;

return T

Chapter 4

Experiments and results

We performed experiments on a large scale multi-class single label datasets. The proposed method *label taxonomy base Neural Netwok* is compared to standard hierarchical classification, and to two machine learning reduction methods : ECOC and One-vs-All.

4.1 Dataset

We used dataset with 150 classes. The dataset was created by randomly selecting the corresponding classes from a large scale dataset released for the first PASCAL large scale hierarchical text classificaiton challenge. This dataset was extracted from the open Mozilla directory DMOZ (www.dmoz.org). The classes are organized in a tree hierarchy, classes being at the leaves of the hierarchy and internal nodes being not instantiated classes. Hierarchy is of depth 5.

The documents were provided as word counts over a vocubalary of size 347,255. Since the features space dimensionality is the number of words in the vocabulary, it was impossible to work on the full feature space. For a matter of computational resources we had to remove words from the vocabulary. The option retained has been to remove every words that appears less than a certain number of times over the whole documents corpus. This option is criticable since non common words are expected to be very discriminative between documents. We tested the solution using OVA and ECOC for various cutting threshold to evaluate the consequences involded by a reduction of vocabulary. We finnaly decided to keep the words that appear a least ten times over the whole documents corpus. Decreasing the dimention of the features space from 347,255 to roughly 20,000. Statistics of the final dataset

are datailed in table 4.1

Set	Space dimension	#categories	#examples
Training	16479	150	14302
Test			3514

Table 4.1: Description of training and test sets

4.2 Weights initialization

4.2.1 Reminder : the label taxonomy based Neural Network's architecture

They are 150 classes that are the leaves of a hierarchy of depth 5. A first weights matrix map the input space to the first hidden layer, it corresponds to the root node's mapping. This matrix is denoted $W_{0,1}$ has dimension $d \times d_{hidden_0}$, where d is the dimension of the input space (in our case 16470) and d_{hidden_0} is the dimension of the first hidden layer. Next step, for each node n of the hierarchy at depth 1, a matrix $W_{1,n}$ maps the first hidden layer to a new hidden layer belonging to the second level of hidden layers. They are as many hidden layers in the second level as the number of nodes of depth 1, the corresponding mapping matrices have dimension $d_{hidden_0} \times d_{hidden_1}$. Next step, matrices of dimension $d_{hidden_1} \times d_{hidden_2}$ maps the hidden layers of the second level to the hidden layers of the third level and so on following the hierarchy.

This way there is an hidden layer at each interior node of the hierarchy. Finally each one of these hidden layers are mapped to an output layer with as many units as the number of sons of the corresponding node. For a node n, we denote V_n the matrice that projects the hidden layer of node n to its output layer, V_n has dimension $d_{hidden_{depth(n)}} \times \#s(n)$ where $\#s(n)$ is the number of sons of note n.

4.2.2 About the weights initialization

The coordinates of the matrice $W_{.,.}$ and $V_.$ are the pool of weights we need to optimize in order to minimize the loss function. Before starting the gradient descent algorithm, the matrices $W_{.,.}$ and $V_.$ are randomly initialized.

- Matrices $W_{\cdot,\cdot}$'s coordinates are randomly picked in $[-0.25; 0.25]$ following a uniform probability distribution.
- Matrices $V_{\cdot}$'s coordinates are initialized either at $\frac{1}{16}$ or $-\frac{1}{16}$.

These choices of weight initialization are experimental. They have been made because they led to the best convergence behavior during the first steps of the gradient descent.

4.3 Experimental setups

4.3.1 The loss weighting

We recall the form of the loss function we aim to minimize :

$$\mathcal{L}^{X,Y}_{W_{\cdot,\cdot},V_{\cdot}} = \frac{1}{2}\sum_{n=1}^{n_Y} \lambda_n \|\sigma(V_n X_n) - v_n^Y\|_F^2$$

The parameters $\lambda_{\cdot}$ can be set in order to attribute more or less importance to the error commited by some layers. For example, we could attribute more weight to the nodes in the uppest layers (closest to the root) because we want them to be as accurate as possible. Thus using a hierarchical inference implies that when one of the uppest classifiers misclassify an example, it is then impossible to recover the example in its correct category. We tried various sets of lambdas :

$$\begin{cases} \lambda_n = 1, \forall n \\ \lambda_n = \dfrac{1}{(1+\text{depth}(n))^k}, k \in \{1, 2\} \end{cases}$$

4.3.2 The learning rate η

They are various hyper parameters that can be set in order to increase the performance of the optimization algorithm.

The principle of the gradient descent algorithm is a step by step procedure consisting in slightly modifying the sets of weight following the gradient of the loss function, in order to make the set of weights converge to a minimizer of the loss function. By following a rule like :

$$W_{new} \leftarrow W_{old} - \eta . \frac{\partial L}{\partial W}$$

where η is called the learning rate of the procedure.

η influences the size of the steps of the procedure. Choosing a large η will accelerate the learning procedures since the step will be larger. On the other hand choosing η too large may lead to bad convergence behavior because when W gets close to the minimizer of the loss function, the last steps can be too large and instead of converging to the minimizer the algorithm will step over this point. Usually η is chosen betwenn 0.001 and 0.01. We tested $\eta = 0.01$ and $\eta = 0.1$

4.3.3 The dimension of the hidden layers

The dimension of the hidden layers is the hyperparameter that will most influence the memory and time complexity of our model. Since the main goal of our approach is to decrease the inference complexity to $0(L)$ wrt the number of labels. We arbitrary decided to choose $d_e \simeq 10 \times \log(L)$ wich is the recommended dimension for random ECOC. In our case we have 150 labels so we set all the hidden layers' dimension to 50.

Unfortunately, we did not have enough time to try all the possibilities of hidden layer's dimensions. A very promising idea was to decrease the dimension of the hidden layers as the layers were deep in the hierarchy. The main idea behind this choice was that the deeper a node is in the hierarchy the less information the node and its descendance need to perform their classification task because they are less categories in the subtree. It is very likely that decreasing the number of units in the deepest layers would not decrease the final performance of the model but it would have led to a huge drop of complexity.

4.4 Convergence issues encountered

When we first tested the optimization we met with a critical issue, the uppest layer corresponding to the root and the nodes of depth 1 and 2 showed great performances but the hidden layers corresponding to the deepest node (at depth 3 and 4) were unable to learn correctly, leading to global very poor performances.

When we tried to explain why we observed this effect, we focused on the fact that during the optimization procedure the weights in the uppest layers are updated way more often than the weights in the deepest levels. For example et us look at a node n at depth 4 with two sons corresponding to label l_1 and l_2. The weights corresponding to the mapping at hidden layer n are updated only when an example from category l_1 or l_2 is picked to calculate the gradient step. On the other hand the weights mapping at the root are updated at every steps. Since they are 150 categories, statistically it means that weigths at root node are updated almost 75 times when weights of node n are updated once (if we consider that the number of examples per category is balanced).

This was an hypothetic example (very likely to happend) but it illustrates the fact that the deepest weights may not be able to follow the updating intensity of the uppest weights during the optimization. To fix this issue we decided to use a technic similar to the *dropout* used in deep learning to avoid overfitting by randomly turning off some neurons at each step of the gradient descent. In our case we decided to sequentially update the weights starting from the deepest weights while keeping the uppest weights untouched.

As we have seen earlier at each step of the descent, in order to calculate the gradient, we randomly pick an example from the training set that has not already been picked until all the possibles examples have been used. Going through all the training set is called an *epoch*. Usually models need several epochs to make the weights converge to the minimizer of the loss function.

Epoch	1	2	3	4	5	6	7	8	9	10	11
Depth 0	×	×	×	×	×	×	×	×	×	×	×
Depth 1											×
Depth 2					×					×	×
Depth 3		×		×	×		×		×	×	×
Depth 4	×	×	×	×	×	×	×	×	×	×	×

Table 4.2: Sequence of weights to update at each epoch (when a cross mark the weights at the corresponding are updated when blank the weights keep untouched during the full epoch). This sequence is followed twice (so during 22 epochs) then we repeat epochs on full network until we consider that the model has converged

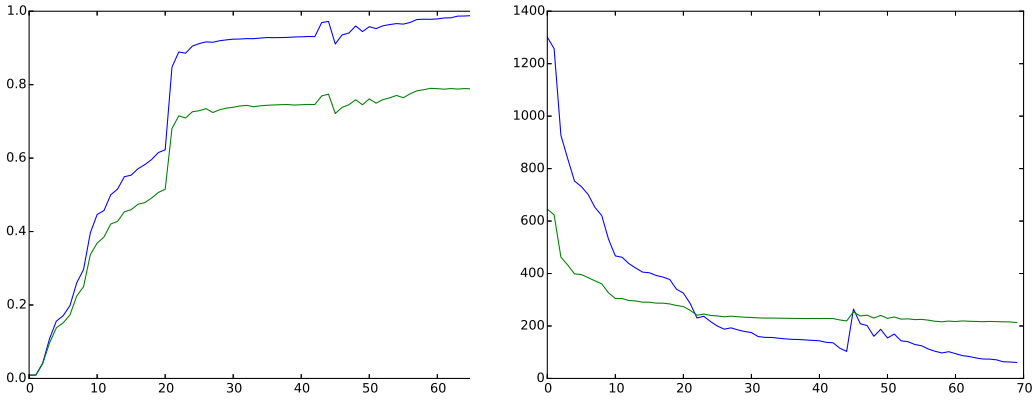
In order to fix the issue of convergence of the deepest layers, at each epoch we decided to update weights only at certains depth following the sequence described in table 4.2

The choice of the sequence has been made to ensure that the deepest weights are well optimized before updating upper weights. The weights at depth 0 corresponding to the first mapping matrix (from the input space) are always updated because they are the one selecting information in the input space.

4.5 Results

Computation has been done using Python, I used the library *numpy* to store the weights matrices, calculate the gradient steps and perform all the matrices calculation. I also used the library *SciPy* to store training and test set as sparse matrices in order to prevent to overload the RAM during experiments and to accelerate the optimization algorithm.

4.5.1 Performances and loss evolution during optimization



(a) Performance evolution

(b) Loss evolution

Figure 4.1: Evolution of performances and Loss function during optimization (blue curves correspond to train set and green curves to test set. Hyperparemeters chosen : $\eta = 0.1$ and $\lambda : -1$

4.5.2 Final performances

In table 4.3, we gather the performances obtained with each combination of hyper parameters. We also refer the performances obtained by the comparison baseline on the same dataset : standard hierarchical approach, ECOC and One-vs-All.

About the comparison baseline

One-vs-All Regular OVA classifier as described in chapter 2. This model has an inference time complexity linear wrt L the number of labels, it makes it a very bad candidate for Extreme Classification but is known to lead to high performances and is useful to measure the highest performances that can be reached.

Random ECOC We generated 500 random ECOC matrices, optimized 500 ECOC based on each one of this coding matrices and finally kept the performances of the one that led to the best results. With an inference time complexity in $O(L)$ wrt the number of labels, random ECOC is a good candidate for Extreme Classification but is known to be disapointing when the number of labels gets to large.

Standard hierarchical approach We trained a One-vs-All classifier at each interior node of the hierarchy that discriminates between the node's sons. As we have seen in chapter 2 hierarchical approaches are very good solutions to decrease inference complexity by using a *divide and conquer* trick.

Table of performances

Model	Label Taxonomy based NN						Hier.	ECOC	OVA
	$\eta = 0.1$			$\eta = 0.01$					
	$\lambda : 0$	$\lambda : -1$	$\lambda : -2$	$\lambda : 0)$	$\lambda : -1$	$\lambda : -2$			
Training	99.04	**99.13**	92.86	87.69	88.48	80.65	99.32	99.45	**99.56**
Test	78.51	**79.65**	73.36	78.40	69.46	64.23	80.04	80.92	**81.57**

Table 4.3: Table of performances in percentage of example correctly categorized by the model at the end of the optimization

The performance with $\eta = 0.01$ are lower beacause the optimization is actually not finished after 30 epochs because the learning rate is too small. The evolution of the performances with $\eta = 0.1$ and $\lambda(-2)$ during optimization showed a strange patern : they increased until reaching a maximum and then drop down, we could not find an explanation to this phenomenom.

Finally we notice that as we expected we found the best performances of our model for a set of λs decreasing wrt the depth of the layer, attributing more importance to the error commited by the uppest layers leads to better performances.

4.6 Conclusion

As we can see in table 4.3 the model that gives the best performances is the OVA but it is also the most costly in computional ressources with a time complexity linear wrt the number of labels and make it a very bad candidate for extreme classification. Random ECOC gives performances slightly better than the hierarchical approach but it is very likely that if we increase the number of labels the hierarchical will scale way better than ECOC.

In the end Label Taxonomy based Neural Network gives slightly lower performances than the classical hierarchical approach. This result is very promisive since our model uses 10 times less weights than the classical hierarchical approach and could use up to 100 times less weights if we decrease the number of units in the hidden layers as they are deep in the network.

Personnaly this work of research at UPMC (University Pierre et Marie Curie in Paris) as a research intern in the *Machine Learning and Artificial Intelligence team* has been a great experience and an excellent dive into the machine learning. I learned a lot about various classification models, learned how to optimize complex statistical models.

Bibliography

[1] Erin L Allwein, Robert E Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *The Journal of Machine Learning Research*, 1:113–141, 2001.

[2] Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*, pages 163–171, 2010.

[3] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

[4] Paul N Bennett and Nam Nguyen. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 11–18. ACM, 2009.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[6] Alina Beygelzimer, John Langford, and Pradeep Ravikumar. Error-correcting tournaments. In *Algorithmic Learning Theory*, pages 247–262. Springer, 2009.

[7] Irving Biederman. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115, 1987.

[8] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[9] Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 78–87. ACM, 2004.

[10] Yangchi Chen, Melba M Crawford, and Joydeep Ghosh. Integrating support vector machines in a hierarchical output space decomposition framework. In *Geoscience and Remote Sensing Symposium, 2004. IGARSS'04. Proceedings. 2004 IEEE International*, volume 2, pages 949–952. IEEE, 2004.

[11] Jia Deng, Sanjeev Satheesh, Alexander C Berg, and Fei Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In *Advances in Neural Information Processing Systems*, pages 567–575, 2011.

[12] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *arXiv preprint cs/9501101*, 1995.

[13] Shantanu Godbole, Sunita Sarawagi, and Soumen Chakrabarti. Scaling multi-class support vector machines using inter-class confusion. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 513–518. ACM, 2002.

[14] Gregory Griffin and Pietro Perona. Learning and using taxonomies for fast visual categorization. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[15] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[16] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. 1997.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] Song Liu, Haoran Yi, Liang-Tien Chia, and Deepu Rajan. Adaptive hierarchical multi-class svm classifier for texture-based image classification. In *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, pages 4–pp. IEEE, 2005.

[19] Tie-Yan Liu, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. Support vector machines classification with a very large-scale taxonomy. *ACM SIGKDD Explorations Newsletter*, 7(1):36–43, 2005.

[20] Marcin Marszalek and Cordelia Schmid. Semantic hierarchies for visual object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE, 2007.

[21] Marcin Marszałek and Cordelia Schmid. Constructing category hierarchies for visual recognition. In *Computer Vision–ECCV 2008*, pages 479–491. Springer, 2008.

[22] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[23] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.

[24] Carlos N Silla Jr and Alex A Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.

[25] Lorenzo Torresani and Kuang-chih Lee. Large margin component analysis. In *Advances in neural information processing systems*, pages 1385–1392, 2006.

[26] Konstantin Tretyakov. Machine learning techniques in spam filtering. In *Data Mining Problem-oriented Seminar*, volume 3, pages 60–79, 2004.

[27] Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2000.

[28] Volkan Vural and Jennifer G Dy. A hierarchical method for multi-class support vector machines. In *Proceedings of the twenty-first international conference on Machine learning*, page 105. ACM, 2004.

[29] Andreas S Weigend, Erik D Wiener, and Jan O Pedersen. Exploiting hierarchy in text categorization. *Information Retrieval*, 1(3):193–216, 1999.

[30] Kilian Q Weinberger and Olivier Chapelle. Large margin taxonomy embedding for document categorization. In *Advances in Neural Information Processing Systems*, pages 1737–1744, 2009.

[31] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research*, 10:207–244, 2009.

[32] Jason Weston and Chris Watkins. Multi-class support vector machines. Technical report, Citeseer, 1998.

[33] Jason Weston, Chris Watkins, et al. Support vector machines for multi-class pattern recognition. In *ESANN*, volume 99, pages 219–224, 1999.

[34] Yiming Yang, Jian Zhang, and Bryan Kisiel. A scalability analysis of classifiers in text categorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 96–103. ACM, 2003.

[35] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *Advances in neural information processing systems*, pages 1601–1608, 2004.

[36] Ning Zhang, Ling-Yu Duan, Qingming Huang, Lingfang Li, Wen Gao, and Ling Guan. Automatic video genre categorization and event detection techniques on large-scale sports data. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, pages 283–297. IBM Corp., 2010.

[37] Liu Zhigang, Shi Wenzhong, Qin Qianqing, Li Xiaowen, and Xie Donghui. Hierarchical support vector machines. In *Geoscience and Remote Sensing Symposium, 2005. IGARSS'05. Proceedings. 2005 IEEE International*, volume 1, pages 4–pp. IEEE, 2005.