UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Learning to Rank with Deep Neural Networks

Dissertation presented by
Goeric HUYBRECHTS

for obtaining the Master's degree in
Computer Science and Engineering
Options:
Artificial Intelligence
Computing and Applied Mathematics

Supervisor
Pierre DUPONT

Readers
Aymen CHERIF, Roberto D'AMBROSIO, Salim JOUILI

Academic year 2015-2016

Declaration of Authorship

I, Goeric HUYBRECHTS, declare that this thesis titled, "Learning to Rank with Deep Neural Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a Master's degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 06/06/16

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

Abstract

Ecole Polytechnique de Louvain

Master in Computer Science and Engineering

Learning to Rank
with Deep Neural Networks

by Goeric HUYBRECHTS

Due to the growing amount of available information, learning to rank has become an important research topic in machine learning. In this thesis, we address the issue of learning to rank in the document retrieval area. Many algorithms have been devised to tackle this problem. However, none of the existing algorithms make use of deep neural networks. Previous research indicates that deep learning makes significant improvements on a wide variety of applications. Deep architectures are able to discover abstractions, with features from higher levels formed by the composition of lower level features. The thesis starts with the analysis of shallow networks, by paying special attention to the network architecture. In a second phase, the number of hidden layers is increased. By applying strategies that are particularly suited for deep learning, the results are improved significantly. These strategies include regularization, smarter initialization schemes and more suited activation functions. Experimental results on the TD2003 dataset of the LETOR benchmark show that these well-trained deep neural networks outperform the state-of-the-art algorithms.

Keywords - Learning to rank, document retrieval, neural networks, deep learning, pairwise approach, LETOR benchmark.

Acknowledgements

Foremost, I would like to express my profound gratitude to my supervisor Prof. Pierre Dupont for his constructive feedback and for the guidance of my Master's thesis.

I would also like to acknowledge the other members of my thesis committee, Dr. Aymen Cherif, Dr. Roberto D'Ambrosio and Dr. Salim Jouili for their encouragement and insightful comments.

My sincere thanks also goes to Euranova for offering me the chance to work on this project. I would like to thank Dr. Aymen Cherif and Dr. Salim Jouili for our weekly meetings during which they gave me very useful advice. The discussions, ideas and feedback have been absolutely invaluable for writing this thesis. Furthermore, the presentation of my work to the staff of Euranova was a unique opportunity to share my results and to discuss some findings.

Last but not the least, I would like to express my special appreciation to my parents for their love and continued support.

Contents

Declaration of Authorship **iii**

Abstract **v**

Acknowledgements **vii**

Introduction **1**

1 Overview **3**
1.1 Learning to Rank . . . 3
1.1.1 Problem formulation . . . 3
1.1.2 Framework . . . 3
1.1.3 Formalization . . . 4
1.1.4 Notation . . . 5
1.1.5 Approaches . . . 6
1.1.6 Evaluation measures . . . 7
1.1.7 Benchmarks . . . 8
1.1.8 Related works . . . 9
1.2 Deep neural networks . . . 13

2 Experimental setting **15**
2.1 Dataset . . . 15
2.2 Basic setting . . . 16
2.2.1 Pre-processing . . . 16
2.2.2 Optimizer . . . 16
2.3 Experimental protocol . . . 17
2.4 Statistical test . . . 18
2.5 Software tools . . . 18

3 Shallow network **19**
3.1 CmpNN architecture . . . 19
3.2 Loss functions . . . 22
3.3 Learning set configuration . . . 22

4 Deep network **25**
4.1 Multiple hidden layers . . . 25
4.2 Regularization . . . 28
4.2.1 Dropout . . . 28
4.2.2 l2-regularization . . . 31
4.3 Initialization and Activation . . . 33
4.3.1 Initialization . . . 33
4.3.2 Activation . . . 36
4.3.3 Initialization/Activation results . . . 38
4.4 Summarization results . . . 38

4.5 Benchmark 40

5 Advanced approaches **43**
5.1 Extensive learning set 43
5.2 Multiple nested ranker 44
5.3 Rank aggregation 45
5.4 Results 46

Conclusion **49**

Bibliography **51**

A LETOR 2.0 - TD2003 Dataset **53**

B SortNet - Incremental learning procedure **55**

List of Figures

1.1 Learning-to-rank framework 4

2.1 TD2003 dataset - Percentage of relevant documents per query 16

3.1 CmpNN architecture 20
3.2 Performances in function of loss functions 23
3.3 Performances in function of learning set configuration 24

4.1 Deep CmpNN architecture 26
4.2 Performances in function of number of hidden layers 28
4.3 Performances in function of number of hidden layers with dropout regularization 29
4.4 MAP score of a 3-hidden layered CmpNN in function of the dropout probability p 29
4.5 Average activation value of hidden neurons without and with dropout regularization 30
4.6 Average and standard deviation of the neuron's activation values with l2-regularization 31
4.7 MAP score of a 3-hidden layered CmpNN in function of the l2-regularization strength λ 32
4.8 Average activation value of hidden neurons without and with l2-regularization 33
4.9 Stacked auto-encoders 37
4.10 Benchmark TD2003 - MAP 41

5.1 Multiple nested ranker - Training phase 45
5.2 Multiple nested ranker - Test phase 45
5.3 Rank aggregation 46

List of Tables

1.1 Notations and explanations 5
1.2 Learning-to-rank approaches 6
1.3 Benchmark datasets 9

3.1 Comparison of performances between dense NN and CmpNN 21

4.1 Configuration deep CmpNN used for experiments 27
4.2 Comparison of performances between 1-hidden layered CmpNN without regularization and 3-hidden layered CmpNN with dropout regularization . 30
4.3 Comparison of performances between 3-hidden layered CmpNN without and with smart initialization schemes and activation functions 38
4.4 Comparison of performances summarization 39
4.5 Benchmark TD2003 - MAP 41
4.6 Benchmark TD2003 - P@n 41
4.7 Benchmark TD2003 - NDCG@n 41

5.1 Comparison of performances between small and extensive learning set . . . 44
5.2 Comparison of performances between different advanced approaches . . . 47

A.1 LETOR 2.0 - TD2003 Dataset; Description of features 53

Introduction

Due to the growing amount of available information, learning to rank has become an important research topic in machine learning. In this thesis, we address the issue of learning to rank in the document retrieval area. Many algorithms have been devised to tackle this problem. However, none of the existing algorithms make use of deep neural networks. Previous research indicates that deep learning makes significant improvements on a wide variety of applications. Deep networks are able to discover abstractions, with features from higher levels formed by the composition of lower level features. The main objective of this thesis is to study whether deep architectures are able to improve the state-of-the-art performances in the learning-to-rank field.

Some of the state-of-the-art algorithms already use neural networks. However, none of them are deep. For instance, the RankNet algorithm, the LambdaRank algorithm, as well as the SortNet algorithm use shallow networks with only one hidden layer. As deep neural networks have been a hot topic since the breakthrough in 2006, a lot of research has been done to improve their performances. New regularization techniques, weight initialization schemes and activation functions have been devised over the years. By applying these strategies, we try to build well-working deep neural networks, hoping to outperform the state-of-the-art algorithms.

This thesis is organized in 6 parts:

Chapter 1 gives a broad overview of the learning-to-rank problem. Furthermore, it presents briefly deep neural networks and the reason why they are so popular nowadays.

Chapter 2 details the experimental setting which will be applied throughout the thesis.

In chapter 3, we conduct several experiments using shallow networks consisting of one hidden layer. Special attention is paid to the network architecture.

In chapter 4, we study deep neural networks by adding more hidden layers. By applying strategies which are known to help the optimization of deep architectures, we investigate whether it is possible to improve the performances. We compare our results with state-of-the-art algorithms.

In chapter 5, we compare various approaches which use larger training sets in the learning process, trying to improve the performances even further.

Finally, in the last chapter some conclusions are drawn and future directions are given.

Chapter 1

Overview

In this first chapter, we start by giving a broad overview of the learning-to-rank problem (section 1.1). Afterwards, we present briefly deep neural networks and the reason why they are so popular nowadays (section 1.2).

1.1 Learning to Rank

1.1.1 Problem formulation

Learning to rank refers to the application of supervised machine learning techniques to construct ranking models for information retrieval systems. It is employed in a wide variety of areas such as document retrieval[1], collaborative filtering and tag ranking. Most supervised learning tasks can be solved by classification or regression. However, some applications require other properties than regular classes or scores. The learning-to-rank problem is one of these applications. The ranking model's purpose is to sort as accurately as possible a list of items with respect to a certain query. Learning to rank doesn't care about the exact classes or scores of each item. Instead, it cares about the relative order between the items. The objective is to rank items with a high degree of relevancy higher than items with a low degree of relevancy.

1.1.2 Framework

Figure 1.1 shows the typical setting for the learning-to-rank application. Training data consists of queries q_i and their associated documents $\mathbf{x}_j^{(i)}$. Furthermore, each query-document pair contains a label/score $y_j^{(i)}$. Using this training data, a specific machine learning algorithm is employed to learn a model h. This model is then applied to sort the test data, for which the ranking is not known. The objective is to predict the ground truth label of that test set as accurately as possible.

The query-document pairs are characterized by three types of features. The first type of features are the query-independent or static features which only depend on the documents. The second type of features are the query-dependent or dynamic features which depend both on the queries and the documents. Finally, the query features are the features which only depend on the queries. Besides these features, each query-document pair contains a relevance score. This score may have been given manually by humans or may have been derived automatically.

[1]The document retrieval application is the one we consider in this thesis.

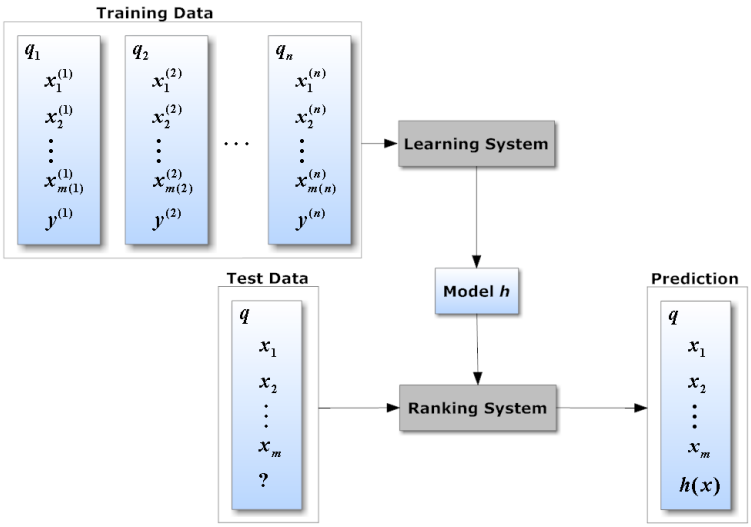


FIGURE 1.1: Learning-to-rank framework - Image taken from [1].

Three strategies have been used in the literature to determine the relevance judgement of a query-document pair:

Relevance degree - For each document associated to a query, a degree of relevancy (with respect to the query) has been specified. This judgement may be binary (e.g. "Relevant" or "Irrelevant") or may contain multiple ordered categories (e.g. "Perfect", "Excellent", "Good", "Fair" or "Bad").

Pairwise preference - A human or automated assessor specifies whether a specific document is more relevant than another one (with respect to a query).

Total order - A complete ordering of the list of documents (with respect to a query) has been established.

Among the three aforementioned types of relevance judgements, the first strategy is the most popular one due to its simplicity to obtain. Therefore, we only consider this strategy in this thesis.

1.1.3 Formalization

In this subsection, we give a more rigourous formalization of the learning-to-rank problem.

Let's denote the set of queries by $\mathcal{Q}$ and the set of documents by $\mathcal{D}$. The query set consists of m queries. Each of these queries q_i is associated with a list of retrieved documents $\mathbf{d_i} = \{d_{i,1}, d_{i,2}, ..., d_{i,n_i}\}$. $d_{i,j}$ represents the j-th document associated to the query q_i and n_i represents the total number of documents retrieved for the query q_i.

All documents have a certain label with respect to their corresponding query. We denote the set of possible labels by $\mathcal{L} = \{l_1, l_2, ..., l_k\}$. These labels take a finite number k of relevance judgements. There exists an order between the relevance levels: $l_1 \prec l_2 \prec ... \prec l_k$, where $\prec$ symbolizes the preference relationship. $\mathbf{y_i} = \{y_{i,1}, y_{i,2}, ..., y_{i,n_i}\}$ denotes the labels of all documents for a certain query q_i with $y_{i,j} \in \mathcal{L}$.

Each query-document pair is characterized by a feature vector $x_{i,j} = \psi_{(q_i,d_{i,j})}$, where ψ is a function extracting useful features from the specific pair. A typical feature is the *PageRank*-feature, which counts the number and quality of links to a page to make a guess about the importance of a website.

The training set is denoted as $S = \{(q_i, \mathbf{d_i}, \mathbf{y_i})\}_{i=1}^{m_{Train}}$, whereas the test set is denoted as $T = \{(q_i, \mathbf{d_i}, \mathbf{y_i})\}_{i=1}^{m_{Test}}$. This latter contains new, unseen queries with their corresponding documents. The goal of the learning-to-rank application is to find the optimal ranking model f, i.e. the one sorting the test set as accurately as possible. This ranking model f assigns to each query-document pair a relevance score. The final ordering of documents is obtained by sorting the documents by decreasing score (with respect to each query).

Each list of retrieved documents $\mathbf{d_i} = \{d_{i,1}, d_{i,2}, ..., d_{i,n_i}\}$ for a certain query q_i can be identified by a list of integers $\{1, 2, ..., n_i\}$. Ranking the documents comes down to building a permutation of this list: $\pi(q_i, \mathbf{d_i}, f)$. The function π is defined as a bijection from $\{1, 2, ...n_{q_i}\}$ to itself. The objective is to obtain the permutation list with the highest performance measure score $E(\pi(q_i, \mathbf{d_i}, f), \mathbf{y_i})$. The most typical performance measures are detailed in subsection 1.1.6.

It is important to be aware that instead of taking each query-document separately as instance of the model, the ranking model could take other types of input. The model could take two query-document pairs as input and tell which one is more relevant. Based on this pairwise comparison, a global ranking can then easily be obtained with a classical sorting algorithm. Another technique is to take the entire set of query-documents pairs as input. The existence of these different approaches are detailed in subsection 1.1.5.

1.1.4 Notation

The notations[2] used in subsection 1.1.3 are summarized in table 1.1 and will be used throughout the report.

Notations	Explanations
q_i	i^{th} query
$\mathbf{d}_i = \{d_{i,1}, d_{i,2}, ..., d_{i,n_{q_i}}\}$	List of documents for q_i
$y_{i,j} \in \{l_1, l_2, ..., l_k\}$	Label of $d_{i,j}$ w.r.t. q_i
$\prec, \succ$	Preference relationships
$l_1 \prec l_2 \prec ... \prec l_k$	Label order
$\mathbf{y}_i = \{y_{i,1}, y_{i,2}, ..., y_{i,n_{q_i}}\}$	List of labels for $\mathbf{d}_i$
n_{q_i}	Size of lists $\mathbf{d}_i$ and $\mathbf{y}_i$
m	Number of queries
$S = \{(q_i, \mathbf{d}_i, \mathbf{y}_i)\}_{i=1}^{m_{Train}}$	Training set
$T = \{(q_i, \mathbf{d}_i, \mathbf{y}_i)\}_{i=1}^{m_{Test}}$	Test set
$x_{i,j} = \psi(q_i, d_{i,j})$	Feature vector for $(q_i, d_{i,j})$
$f(x_{i,j})$ or $h(x_{i,j})$	Ranking model
$\pi(q_i, \mathbf{d}_i, f)$	Permutation for $q_i, \mathbf{d}_i$ and f
$E(\pi(q_i, \mathbf{d}_i, f), \mathbf{y}_i) \in [0, 1]$	Performance measure function

TABLE 1.1: Notations and explanations

[2]The notation is based on the one used in [2] and [3].

1.1.5 Approaches

Tie-Yan Liu, lead researcher of Microsoft Research Asia, has stated in [1] that the existing algorithms for learning-to-rank problems can be categorized into three groups by their input representation and loss function.

Approach	Input	Output
Pointwise	Single documents	Class or scores labels
Pairwise	Document pairs	Partial order preference
Listwise	Document collections	Ranked document list

TABLE 1.2: Learning-to-rank approaches

Pointwise approach - Each query-document pair in the training set has a numerical or ordinal score. The learning-to-rank problem can be seen as a regression or classification problem, where each item is scored independently of the others. An ordering of the items is then obtained by sorting on the scores.

Advantages:

- Existing methodologies and theories on regression or classification are directly supported.

Drawbacks:

- The relative order between the documents isn't taken into account, as each document is considered separately.
- The pointwise approach overlooks the fact that some documents belong to the same queries. This results in a greater impact of the queries with a large number of associated documents when the distribution of documents per queries varies considerably.
- The actual position of each document in the list is not taken into consideration in the loss function.

Examples state-of-the-art algoritms: PRank [4], McRank [5]

Pairwise approach - The model works as a binary classifier, predicting which document in a given pair is the most relevant with respect to a certain query. The learned preference function can then be embedded as the comparator into a classical sorting algorithm to provide a global ranking of the set of items.

Advantages:

- Like in the pointwise approach, existing methodologies and theories on regression or classification are in large parts supported. Furthermore, the pairwise approach is able to model the relative order between documents.

Drawbacks:

- Most algorithms don't take the multiple levels of relevance degrees into consideration. A document pair consisting of two documents labeled "Perfect" and "Bad" is treated in the same way as a document pair consisting of two documents labeled "Good" and "Fair". This results in a loss of information.

- The unbalanced distribution between documents is a greater issue than in the pointwise approach, as the number of pairs is of quadratic order with respect to the number of documents. The model will therefore be biased towards the queries with a higher number of associated documents.
- The pairwise approach is more sensible to noisy data. A single mis-labeled document can result in a huge number of mis-labeled document pairs.
- The actual position of each document in the list is not taken into account in the loss function. However, compared to the pointwise approach the relative order between pairs of documents is considered.

Examples state-of-the-art algoritms: RankSVM [6], RankBoost [7], RankNet [8], LambdaRank [9], SortNet [10]

Listwise approach - The model takes lists as instances and tries to directly optimize the value of one of the evaluation measures[3].

Advantages:

- The listwise approach addresses the learning-to-rank problem in the most natural way, by taking document lists as instances in the learning process.

Drawbacks:

- The resulting optimization problem is more challenging than those in the pointwise or pairwise approach, due to the non-continuous and non-differentiable loss function. This often results in higher time complexities.

Examples state-of-the-art algoritms: AdaRank [3], ListNet [11]

1.1.6 Evaluation measures

There exist several evaluation measures which are commonly used to judge how well an algorithm is performing. These measures allow to compare the performances of different learning-to-rank algorithms. The ranking quality measures we consider are the Precision at position n (P@n), the Mean Average Precision (MAP) and the Normalized Discounted Cumulative Gain (NDCG@n).

Precision at position n (P@n) - P@n measures the fraction of relevant documents in the n top positions.

$$P@n = \frac{\text{\# Relevant docs in top } n \text{ results}}{n}$$

Mean Average Precision (MAP) - For a single query q_i, the Average Precision (AP) is defined as the average of the P@n values for all relevant documents.

$$AP = \frac{\sum_{n=1}^{N} P@n.rel(n)}{\text{\# Relevant docs associated to query } q_i},$$

where N is the number of retrieved documents (for the specific query) and rel(n) is a binary function on the relevance of the n-th document. MAP is obtained by averaging the AP values corresponding to all queries.

[3]The evaluation measures are specified in the next subsection.

Normalized Discounted Cumulative Gain (NDCG@n) - The Discounted Cumulative Gain (DCG@n) measures the gain of a document based on its relevance judgement and position in the list. The higher a document is ranked, the more its relevance judgement contributes to the overall performance. The DCG@n measure can be further normalized such that it takes values from 0 to 1, resulting in the Normalized Discounted Cumulative Gain (NDCG@n). This is done by dividing it by the Ideal DCG (IDCG), i.e. the maximum performance attained when all documents are ranked in an optimal way.

$$NDCG@n = \frac{DCG@n}{IDCG@n}$$

$$DCG@n = \sum_{i=1}^{n} \frac{2^{rel(i)} - 1}{log_2(i+1)},$$

where rel(i) is the multi-label, graded relevance of the document at position i.

It is interesting to point out that the evaluation measures P@n and MAP can only handle cases with binary judgements ("Relevant" or "Irrelevant"), whereas NDCG can handle multiple levels of relevance.

Algorithms based on the pointwise or pairwise approach are not capable of directly optimizing these evaluation measures as they don't take the positions of the documents in the list into account. One may thus think that directly optimizing these evaluation measures with the listwise approach is undeniably the best way of proceeding. Unfortunately, it isn't as simple as that. As all evaluation measures depend on the positions of the documents, the measures are non-continuous and non-differentiable. This makes the optimization complicated and slow. Therefore, most algorithms adopting the listwise approach approximate or bound the actual evaluation measures.

1.1.7 Benchmarks

To boost the research in the learning-to-rank field, some helpful datasets have been released. The most famous benchmark datasets are the LETOR benchmark, the benchmark of Yandex, the benchmark of Yahoo! and the benchmark of Microsoft. The details of these benchmarks are summarized in the table 1.3.

LETOR - LETOR [12] is a benchmark collection released by Microsoft Research Asia. Official baselines (P@n, MAP and NDCG@n) are provided for various state-of-the-art algorithms. LETOR is the most widely used benchmark in the learning-to-rank field.

Yandex - The Russian search engine Yandex has released one of their internal datasets for a competition. Unfortunately, no official baselines are provided. The results of the contenders could be used as baselines.

Yahoo! - Yahoo! has publicly released two datasets used internally at Yahoo! during the Yahoo! Learning to Rank Challenge that took place in spring 2010 [13][14]. As it is the case for the Yandex benchmark, the results of the contenders can be used as baselines.

Microsoft - Besides the LETOR benchmark, Microsoft has released two larger scale datasets. No official baselines are provided.

Benchmark	Name dataset	#Queries	#Doc.	#Rel. lev.	#Feat.	Year
LETOR 2.0	TD2003	50	≈ 50k	2	44	2007
	TD2004	75	≈ 75k	2	44	2007
	OSHUMED	106	≈ 15k	3	25	2007
LETOR 3.0	TD2003	50	≈ 150k	2	64	2008
	TD2004	75	≈ 75k	2	64	2008
	NP2003	150	≈ 150k	2	64	2008
	NP2004	75	≈ 75k	2	64	2008
	HP2003	150	≈ 50k	2	64	2008
	HP2004	75	≈ 75k	2	64	2008
	OSHUMED	106	≈ 15k	3	45	2008
LETOR 4.0	MQ2007	1692	≈ 70k	3	46	2009
	MQ2008	784	≈ 15k	3	46	2009
Yandex	/	20267	≈ 200k	5	245	2009
Yahoo!	/	36251	≈ 875k	5	700	2010
Microsoft	MSLR-WEB30k	31531	≈ 3775k	5	136	2010
	MSLR-WEB10k	10000	≈ 1200k	5	136	2010

TABLE 1.3: Characteristics of well-known document retrieval benchmark datasets.

1.1.8 Related works

Several algorithms tackling the learning-to-rank problem have emerged in the past decades. As stated previously, the existing algorithms can be categorized into three approaches. We present here briefly the most famous state-of-the-art algorithms of each learning-to-rank approach. The algorithms are detailed by approach and by date.

1. Pointwise approach

PRank (2002) [4] - The PRank algorithm considers instances $\mathbf{x}$ whose ranks are an element from a finite set $y = \{1, 2, ..., k\}$, defined by the total order relation $\prec$. A ranking rule $\mathcal{H}$: $\mathbb{R}^n \rightarrow y$ maps each instance to a specific rank. The rule uses a vector $\mathbf{w} \in \mathbb{R}^n$ and a set of k thresholds $b_1 \leqslant ... \leqslant b_{k-1} \leqslant b_k = \infty$. In order to compute the rank of an instance $\mathbf{x}$, the algorithm computes the dot product between $\mathbf{w}$ and $\mathbf{x}$. The rank is then defined as the index of the smallest threshold b_r for which $\mathbf{w} \cdot \mathbf{x} < b_r$. The PRank algorithm is an iterative procedure. At each round, it takes an instance $\mathbf{x}$ as input and predict its output $\hat{y} = min_r\{r : \mathbf{w} \cdot \mathbf{x} - b_r < 0\}$. Taking the true value of $\mathbf{x}$ into account, the ranking rule is adapted by modifying $\mathbf{w}$ and $\mathbf{b}$ if a classification mistake is made. The update rule is motivated by the perceptron algorithm. Basically, the values of $\mathbf{w} \cdot \mathbf{x}$ and b_r are pushed towards each other if the label has been incorrectly predicted. The objective of the algorithm is to minimize the ranking-loss, characterized by the number of thresholds between the true ranks and the predicted ranks.

McRank (2007) [5] - The ranking problem is seen as a multiple (ordinal) classification issue, in which the DCG evaluation measure is considered. Classifying instances perfectly leads to a perfect DCG score. In such case, instances with the same class labels may be ranked arbitrarily (within their same class), without consequences on the score. However, obtaining a perfect classification is unfeasible. A mechanism is therefore required to convert the classification results into ranking scores. First, a soft classification algorithm is

used to learn the class probabilities $p_{i,k} = Pr(y_i = k)$, with i being a specific instance and k being a specific class label out of the K possible labels. Then, each instance is scored according to the expected relevance $S_i = \sum_{k=0}^{K-1} \hat{p}_{i,k} T(k)$, where $T(k)$ equals k. Once all scores S_i have been computed, the items are sorted by descending score with respect to each query. The learning process of the class probabilities $p_{i,k}$ is done by a boosting tree algorithm. The remaining issue is the fact that the DCG measure is hard to optimize, since it is non-continuous and non-smooth. Therefore, motivated by the fact that DCG error can be bounded by classification errors, a well-known surrogate loss function is used instead: $\sum_{i=1}^{N} \sum_{k=0}^{K-1} -log(p_{i,k}) \mathbb{1}_{y_i=k}$. Slightly better results are obtained when the natural order among the class labels is kept. In that case, cumulative probabilities $Pr(y_i \leqslant k)$ are learned instead of the regular class probabilities.

2. Pairwise approach

RankSVM (2000) [6] - Generally, a ranking model f isn't able to produce the optimal ranking r^*. Instead, the performance of the model is evaluated by measuring how well the predicted ordering r_f approximates r^*. If a document d_i is ranked higher than another document d_j for an ordering r, then $(d_i, d_j) \in r$, otherwise $(d_i, d_j) \notin r$. The optimization is performed over a set of m training queries, where $\Phi(q, d)$ is a function mapping each query-document pair to an appropriate feature space describing the similarities between query and document. RankSVM computes the weight vector $\mathbf{w}$ such that as many as possible of following inequalities are fulfilled:

$$\forall (d_i, d_j) \in r_1^* : \mathbf{w} \cdot \Phi(q_1, d_i) \geq \mathbf{w} \cdot \Phi(q_1, d_j)$$

...

$$\forall (d_i, d_j) \in r_m^* : \mathbf{w} \cdot \Phi(q_m, d_i) \geq \mathbf{w} \cdot \Phi(q_m, d_j)$$

Unfortunately, this problem is NP-hard. Hence, the solution is approximated by introducing slack variables $\xi_{i,j,k}$ and minimizing the upper bound $\sum \xi_{i,j,k}$. The optimization problem reduces to following problem, where regularization has been added in order to maximize the margin:

$$minimize : V(\mathbf{w}, \xi) = \tfrac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum \xi_{i,j,k}$$

$$subject\ to :$$

$$\forall (d_i, d_j) \in r_1^* : \mathbf{w} \cdot \Phi(q_1, d_i) \geq \mathbf{w} \cdot \Phi(q_1, d_j) + 1 - \xi_{i,j,1}$$

...

$$\forall (d_i, d_j) \in r_m^* : \mathbf{w} \cdot \Phi(q_m, d_i) \geq \mathbf{w} \cdot \Phi(q_m, d_j) + 1 - \xi_{i,j,n}$$

$$\forall i \forall j \forall k : \xi_{i,j,k} \geq 0$$

By rearranging the constraints as $\mathbf{w} \cdot (\Phi(q_k, d_i) - \Phi(q_k, d_j)) \geq 1 - \xi_{i,j,k}$, the problem reduces to a classification SVM on pairwise difference vectors $\Phi(q_k, d_i) - \Phi(q_k, d_j)$.

RankBoost (2003) [7] - RankBoost is largely inspired by AdaBoost [15]. The major difference between these two algorithms is the fact that RankBoost is defined on document pairs, whereas AdaBoost is defined on individual documents. Like all boosting algorithms, RankBoost is an iterative procedure working in T rounds. A weight distribution D_t over the document pairs is maintained and updated at each iteration. In each round, a weak ranker $h_t(x)$ is built in order to minimize $\sum_{x_0,x_1} D_t(x_0, x_1) exp(\alpha_t(h_t(x_0) - h_t(x_1)))$, where x_1 has a higher rank than x_0 and $D_t(x_0, x_1)$ is the weight associated to that specific pair. Each weak

ranker $h_t(x)$ is given a certain weight, characterized by the coefficient α_t. The weight distribution of the document pairs is then updated with regards to the weak learner $h_t(x)$ and its corresponding weight α_t. The weights of incorrectly classified pairs $(h_t(x_1) < h_t(x_0))$ are increased, while the weights of correctly classified pairs $(h_t(x_1) > h_t(x_0))$ are decreased. The final ranking function is obtained by linearly combining the weak rankers, based on the coefficients α_t: $\mathcal{H}(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$.

RankNet (2005) [8] - RankNet uses neural networks to model the ranking function. The learned model f assigns real values to samples estimating their rank. If $f(x_i) > f(x_j)$, then x_i is considered more relevant than x_j. The authors propose the cross entropy as a probabilistic cost function on pairs of examples: $C_{ij} = -\bar{P}_{ij} log(P_{ij}) - (1 - \bar{P}_{ij}) log(1 - P_{ij})$, where P_{ij} denotes the modeled posterior that x_i is ranked higher than x_j. To map the outputs to probabilities, a logistic function is used: $P_{ij} = \frac{e^{o_{ij}}}{1+e^{o_{ij}}}$, where $o_{ij} = f(x_i) - f(x_j)$. Training the neural network is done by modifying the standard back-propagation algorithm. The neural network takes only one instance as input and has a single output node. Hence, the cross entropy depends on the difference of the outputs of two consecutive training samples. The first training sample introduced in the neural network should have a higher or equal rank than the second training sample. Consequently, the forward propagation consists of two steps. First, a forward propagation is executed for the first training sample. The activations and gradients of every node are stored. Afterwards, a second propagation is executed for the second training sample. Again, all activations and gradients are stored. Finally, all weights of the network are adapted with respect to the gradients of the aforementioned cost function.

LambdaRank (2006) [9] - LambdaRank improves RankNet with an upgraded loss function. It is faster and achieves better performances. Optimizing the evaluation measures directly is difficult, as they depend on the scores only through the returned ranked list of documents. The authors of the paper found out that the training process of RankNet didn't require the actual value of the loss function, but only the gradients of the loss function with respect to the scores. Let's suppose a ranking algorithm is being trained on a set of training samples consisting of only two relevant documents, x_1 and x_2, with NDCG@1 as evaluation measure. After one iteration, x_1 (with score s_1) is ranked near the top of the list, whereas x_2 (with score $s_2 < s_1$) is ranked near the bottom of the list. Moving x_1 to the top position demands clearly less effort than moving x_2 to the top position. This leads to the idea to define an implicit optimization cost function C which has the property: $|\frac{\delta C}{\delta s_{j_1}}| \gg |\frac{\delta C}{\delta s_{j_2}}|$, whenever $j_2 \gg j_1$ (j_1 and j_2 being the rank indices of x_1 and x_2).

FRank (2007) [16] - The cross-entropy loss function of the RankNet algorithm contains some flaws. Therefore, the authors of the FRank algorithm have proposed a new loss function, which they call the fidelity loss. This loss function is inspired by the concept of fidelity, which is used in the field of physics to measure the difference between two states of a quantum. The fidelity of two probability distributions p_x and q_x is defined by $F(p_x, q_x) = \sum_x \sqrt{p_x q_x}$. The authors have adapted this distance metric to measure the loss of a pair of instances as follows: $F_{ij} = 1 - \sqrt{P^*_{ij} P_{ij}} + \sqrt{(1 - P_{ij})^* (1 - P_{ij})}$, where P^*_{ij} is the given target value for the posterior probability and P_{ij} is the modeled probability. This novel loss function has some desired advantages over the cross-entropy loss. This latter has a non-zero minimum, meaning there will always be some loss (even for a perfect ranking). Furthermore, the cross-entropy loss isn't bounded, which makes it very sensible to pairs that are difficult to classify. The fidelity loss doesn't have these drawbacks, as it has a zero minimum loss and it is bounded between 0 and 1. However, a deficiency is the fact that the loss isn't convex, making it more difficult to optimize.

SortNet (2008) [10] - The SortNet algorithm uses neural networks to construct a model able to compare pairs of objects. More precisely, it uses a feed-forward neural network, consisting of one hidden layer and two output nodes. The neural network takes two objects, $\mathbf{x_1}$ and $\mathbf{x_2}$, as input and computes the evidence of the relationships $\mathbf{x_1} \succ \mathbf{x_2}$ and $\mathbf{x_2} \succ \mathbf{x_1}$, to tell which object out of the two is the most relevant one. The particularity about the neural network is its constrained weight architecture, which allows to model the ranking aspect between two objects more accurately. The architecture ensures that the reflexivity property as well as the equivalence property between $\succ$ and $\prec$ are satisfied. A global ranking can then easily be obtained by embedding the preference model into a classical sorting algorithm. Furthermore, an incremental learning procedure has been devised. The SortNet algorithm is an iterative procedure, where at each iteration the learning set is extended with mis-classified pairs. This allows the algorithm to concentrate on the most informative patterns of the input domain. The algorithm finishes when the maximum number of iterations has been reached or when the learning set remains unchanged from one iteration to the other.

3. Listwise approach

AdaRank (2007) [3] - All aforementioned state-of-the-art algorithms minimize loss functions which are only loosely related to the actual evaluation measures. AdaRank solves this flaw by optimizing a loss function that is directly defined on the evaluation measures. AdaRank repeatedly constructs weak rankers over several rounds. At each round, a weight distribution over the queries is kept. At first, each query is given the same weight. Then as the procedure advances, the weights associated to queries which are wrongly ranked are increased. This allows the model to focus on the so-called hard queries. The final ranking model is obtained by linearly combining the weak rankers. Each weak ranker is weighted by a coefficient α_t measuring its importance. Ideally, the algorithm should maximize following performance measure: $max_{f \in F} \sum_{i=1}^{m} E(\pi(q_i, \mathbf{d}_i, f), \mathbf{y}_i)$, where F is the set of possible ranking functions. This comes down to minimizing the equivalent loss function: $min_{f \in F} \sum_{i=1}^{m} (1 - E(\pi(q_i, \mathbf{d}_i, f), \mathbf{y}_i))$. However, as this loss function is non-continuous, optimizing it directly is rather complicated. An upperbound of the measure is minimized instead: $min_{f \in F} \sum_{i=1}^{m} exp(-E(\pi(q_i, \mathbf{d}_i, f), \mathbf{y}_i))$.

ListNet (2007) [11] - As in AdaRank, the paper postulates that the learning-to-rank problem should be solved by concentrating on the listwise approach. Let $\mathbf{x}^{(i)} = (x_1^{(i)}, ..., x_n^{(i)})$ be the list of documents associated to the i-th query and $\mathbf{y}^{(i)} = (y_1^{(i)}, ..., y_n^{(i)})$ be the ground truth of that list. A ranking function estimates the scores of the documents in the list $\mathbf{x}^{(i)}$, denoted by $\mathbf{z}^{(i)} = (z_1^{(i)}, ..., z_n^{(i)})$. Sorting the documents is then achieved by ranking these by decreasing score. The loss function that ListNet tries to optimize is $\sum_{i=1}^{m} L(\mathbf{y}^{(i)}, \mathbf{z}^{(i)})$, where L is a listwise loss function. To define this listwise loss function, the authors propose to use a probability distribution on permutations, as ranking documents comes down to permute the initial positions of these documents. The probability of a permutation π, given the list of scores $\mathbf{s}$, is defined as $P_s(\pi) = \Pi_{j=1}^{n} \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^{n} \phi(s_{\pi(k)})}$, where ϕ is an increasing and strictly positive function. Given two lists of scores, $\mathbf{y}$ and $\mathbf{z}$, we can compute the associated permutation probability distributions. Any metric (e.g. cross entropy) between these two distributions can been used to serve as loss function. The main issue of this approach is the high time complexity. To tackle this problem, a second probability model has been introduced, in which the top k probability is considered. This reduces the exponential complexity time to a polynomial complexity time in order of the number of queries.

1.2 Deep neural networks

Artificial neural networks are a family of models inspired by biological nervous systems such as the brain. They are powerful computational models able to approximate highly complex functions. A neural network is composed of a large number of highly interconnected processing elements, called neurons. Like the neural networks present in our human brains, they learn through training. The knowledge acquired through training is stored within the adaptable connection weights between the neurons.

The difference between deep learning algorithms [18] [19] and shallow learning algorithms is the number of (non-linear) transformations a signal encounters as it propagates from the input layer to the output layer. For instance, a deep neural network contains multiple hidden layers between the input and output layers. The key of deep learning is to discover abstractions, with features from higher levels formed by the composition of lower level features. The learning of these multiple levels of abstraction is done by extensive training. Deep learning allows the system to learn complex functions, which may be too difficult to represent with shallow architectures.

A breakthrough in the domain of deep neural networks occurred in 2006. Researchers have managed to devise new strategies able to boost significantly the training process of these deep neural networks. Current neural networks outperform many of the state-of-the-art algorithms on various tasks. They are now being deployed on a large scale by companies such as Google, Microsoft and Facebook. However, training deep neural networks remains (extremely) complicated. Two common issues are overfitting and the high computation cost.

Chapter 2

Experimental setting

Ranking is a central part of many information retrieval applications, such as document retrieval, collaborative filtering and tag ranking. In this thesis, we tackle the learning-to-rank problem by focusing on document retrieval. This is the application which has gained the most interest in recent years.

First, we introduce the dataset that we have used to conduct our experiments (section 2.1). Afterwards, we explain the basic setting of our neural network implementation (section 2.2). Next, we present the experimental protocol used for all our experiments (section 2.3). Then, we detail the statistical test we have applied to compare the models obtained with different settings (section 2.4). Finally, we state the software tools which have been used to conduct our experiments (section 2.5).

2.1 Dataset

The Text REtrieval Conference (TREC[1]) is an on-going series of workshops focusing on a list of different information retrieval research areas or tracks. Its purpose is to encourage research within the information retrieval by providing helpful dataset collections.

One of their tracks is the TREC 2003 web track, whose corresponding dataset is denoted as TD2003. It is part of the LETOR 2.0 benchmark [12]. The task of this track is topic distillation, which aims to find a list of entry points for good websites principally devoted to the topic. Instead of returning pages containing pertinent information themselves, the goal is to return entry pages of good websites, since these pages give a better overview of the topic. For each webpage, a binary judgement has been made by humans to tell whether the page is relevant or not with respect to a certain query.

The TD2003 dataset consists of 2 relevance levels, 50 queries and 44 features for each query-document pair. The features have been extracted in such a way to cover all the standard features in information retrieval. The description of each feature is given in Appendix A.

As stated in chapter 1, one of the issues of the pairwise approach[2] is the imbalanced distribution across queries. Since the number of pairs can be as large as in the quadratic order of the number of documents, this can be quite problematic. A quick analysis of the TD2003 dataset tells us that each query has between 1,000 and 1,045 associated documents, except for two queries which have respectively 525 and 533 associated documents. Therefore, when selecting a training set of query-document pairs, we choose these such that each query is represented equally. Figure 2.1 shows the percentage of relevant documents per

[1] http://trec.nist.gov

[2] The pairwise approach is the approach we consider in this thesis, as will be made clear in the next chapter.

query. From the figure, we conclude that the number of irrelevant documents is much higher than the the number of relevant documents. Incorporating pairs randomly in the training set may therefore not be the best way of proceeding. This issue is analysed in the next chapter.

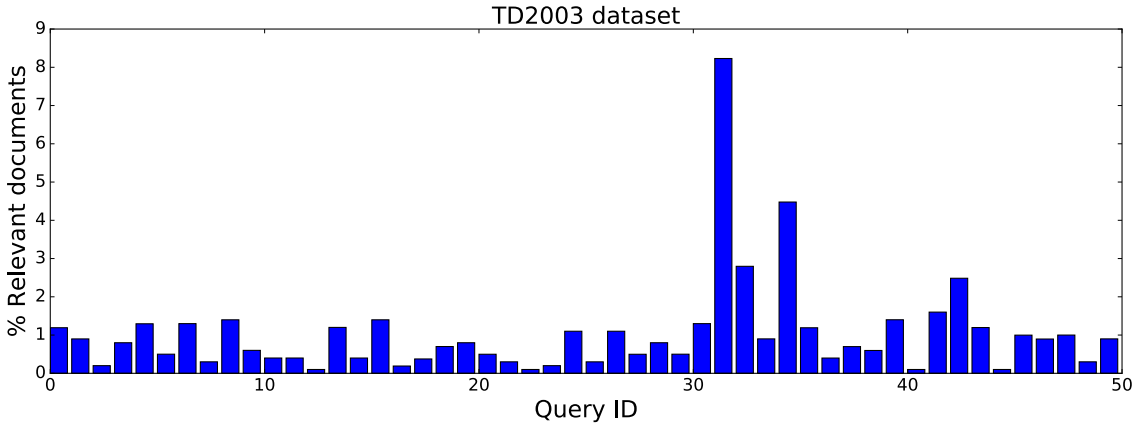


FIGURE 2.1: TD2003 dataset - Percentage of relevant documents per query

2.2 Basic setting

2.2.1 Pre-processing

The type of pre-processing that has been applied on the data is the following:

$$\hat{x}^i_{j,r} = \frac{x^i_{j,r} - \mu^i_r}{max_{s=1,\ldots,n_{q_i}} |x^i_{s,r}|},$$

where n_{q_i} is the number of documents associated to the i-th query, μ^i_r is the average value of the r-th feature of all documents corresponding to the i-th query, $x^i_{j,r}$ denotes the r-th feature of the j-th document associated to the i-th query and $\hat{x}^i_{j,r}$ denotes its normalized version. This type of feature scaling constrains the features into the range [-1, 1]. It appears to achieve better results than the standard normalization procedure.

2.2.2 Optimizer

To train the neural networks, the standard backpropagation method with gradient descent is applied. Batch training is preferred to online training, as this speeds up the learning process significantly. Instead of a constant learning rate, we use the same learning rate as in [10]. Basically, the learning rate is increased by a multiplicative factor when the classification error decreases from one epoch to the other. If the classification error increases, the learning rate is decreased and the weights of the neural network at the previous iteration are restored. The exact procedure is shown below.

Algorithm 1 Adaptive learning rate

$\eta_{increment-ratio} = 1,\ \ \eta_{increment} = 1.05,\ \ \eta_{max} = 10^3$
$\eta_{decrement-ratio} = 1.05,\ \ \eta_{decrement} = 0.30,\ \ \eta_{min} = 10^{-6}$
$repeat_{max} = 10$

$R = \frac{error_{i-1}}{error_i}$
if $R < \eta_{increment-ratio}$ **then**
    if $\eta < \eta_{max}$ **then**
        $\eta = min(\eta_{max}, \eta * \eta_{increment})$
    end if
else if $R > \eta_{decrement-ratio}$ **then**
    if $\eta > \eta_{min}$ **then**
        if $repeat < repeat_{max}$ **then**
            Restore old weights
            $repeat++$
            $\eta = max(\eta_{min}, \eta * \eta_{decrement})$
            return
        end if
    end if
end if
$repeat = 0$

2.3 Experimental protocol

The experiments of subsequent chapters are conducted using a 5-fold cross-validation, given that the folds are already provided by the LETOR benchmark. Each fold is partitioned into one training set, one validation set and one test set.

Training neural networks and more especially deep neural networks demands expensive computations. Experiments including all possible training and validation samples in the learning process take several days (to weeks) to execute. For these reasons, the experiments are performed using a limited number of samples. In the last chapter, we will consider larger learning sets.

All performed experiments are repeated 30 times. In each experiment, the neural network is trained on 10,000 training and validated on 5,000 validation samples. The training process of the neural networks is done over 1,000 epochs, a value which has been chosen heuristically. Once the neural network has been trained, all documents of the entire training, validation and test set are ranked according to the model. Then, the scores of the evaluation measures (P@n, NDCG@n and MAP) of the sorted document lists are computed. Additionally, we provide the number of mis-classified pairs based on 10,000 randomly selected test pair samples.

In chapter 3, we study shallow neural networks. Then, we conduct experiments on deep neural network architectures in chapter 4. Finally, in chapter 5, we compare different approaches which take larger learning sets of documents into consideration during the training process.

2.4 Statistical test

To certify if a certain model m_1 outperforms another model m_2, a statistical test is applied. The statistical test which we will use, is the Welch's t-test. The Welch's t-test is an adaptation of the regular t-test. It is more reliable when the two samples have unequal variances (and unequal sample sizes). The assumption of normality is maintained. We fix α, the probability of wrongly rejecting the null hypothesis while it is actually true, to 0.05.

The Welch's t-test defines the statistic T by following formula:

$$T = \frac{\bar{p}_1 - \bar{p}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}},$$

where $\bar{p}$, s^2 and N are respectively the sample mean, sample variance and sample size.

The degrees of freedom ν are approximated as follows:

$$\nu \approx \frac{(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2})^2}{\frac{s_1^4}{N_1^2\nu_1} + \frac{s_2^4}{N_2^2\nu_2}},$$

where $\nu_1 = N_1 - 1$ is the degrees of freedom associated with the first variance estimate and $\nu_2 = N_2 - 1$ is the degrees of freedom associated with the second variance estimate.

The exact procedure of our statistical test is given hereafter:

1. Null hypothesis $H_0 : p = p_2 - p_1 = 0\%$
2. Alternative hypothesis $H_a : p = p_2 - p_1 \neq 0\%$
3. Computation test statistic T
4. Significant difference?
   - Is $T < -t_{\alpha/2,\nu}$ or $T > t_{\alpha/2,\nu}$ with $Pr(T \notin [-t_{\alpha/2,\nu}, t_{\alpha/2;\nu}]) = \alpha$?
   - P-value < α?
5. If the sample contradicts the theory, the current theory is rejected. Otherwise, we don't know whether the theory is true or not.

2.5 Software tools

In order to implement deep neural networks and conduct our experiments, we have used following libraries:

Theano[3] - Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.

Keras[4] - Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano.

[3]Reference: http://deeplearning.net/software/theano/
[4]Reference: http://keras.io

Chapter 3

Shallow network

Tackling the learning-to-rank problem using neural networks can be done in multiple ways. In this thesis, we take a closer look at the neural network architecture suggested by L. Rigutini et al. as vital element of the SortNet algorithm [10], whose performances outperform many of the state-of-the-art algorithms. L. Rigutini et al. present an interesting preference learning method to approximate the comparison function for a pair of objects. The neural network adopts a particular weight architecture designed to implement the symmetries naturally present in a preference function.

In this chapter, we study shallow networks consisting of one hidden layer. First, we analyse the constrained weight architecture suggested in [10] (section 3.1). Afterwards, we examine several loss functions to see which one outperforms the others (section 3.2). Finally, we investigate the influence of the type of document pairs on which to train the neural network (section 3.3).

3.1 CmpNN architecture

The neural network used to model the preference function in [10] is called the comparative neural network (CmpNN). It is a feed-forward neural network consisting of one hidden layer and two output nodes, whose weights have been initialised randomly and whose activation functions are sigmoids. The input layer takes the concatenation ($[\mathbf{x}, \mathbf{y}]$) of two documents, $\mathbf{x}$ and $\mathbf{y}$, as input. The values of the two output nodes, which are denoted by $N_{\prec}([\mathbf{x}, \mathbf{y}])$ and $N_{\prec}([\mathbf{x}, \mathbf{y}])$, estimate respectively the evidence of $\mathbf{x} \succ \mathbf{y}$ and the evidence of $\mathbf{x} \prec \mathbf{y}$.

The idea is to train the comparator by examples. For each pair of training inputs $[\mathbf{x}, \mathbf{y}]$, the assigned target is:

$$\mathbf{t} = [t_1, t_2] = \begin{cases} [1,0] & if \ \ \mathbf{x} \succ \mathbf{y} \\ [0,1] & if \ \ \mathbf{x} \prec \mathbf{y} \end{cases}$$

The loss function is the mean-squared error, where the error of each document pair is computed as follows:

$$E([\mathbf{x}, \mathbf{y}], \mathbf{t}) = \frac{1}{2}\Big((t_1 - N_{\succ}([\mathbf{x}, \mathbf{y}]))^2 + (t_2 - N_{\prec}([\mathbf{x}, \mathbf{y}]))^2\Big)$$

After the learning process, the comparator can be used to compare any pair of documents. If $N_{\succ}([\mathbf{x}, \mathbf{y}])$ is superior to $N_{\prec}([\mathbf{x}, \mathbf{y}])$, $\mathbf{x}$ is considered to be more relevant. Otherwise, $\mathbf{y}$ is considered to be more relevant.

The particularity about the CmpNN architecture is its constrained weight architecture, especially suited to model a preference function.

A total ordering of objects is satisfied if the four following properties hold:

1. Reflexivity: both x $\succ$ x and x $\prec$ x hold.
2. Equivalence between $\prec$ and $\succ$: if x $\succ$ y, then y $\prec$ x and, vice versa, if y $\succ$ x, then x $\prec$ y.
3. Anti–symmetry: if x $\succ$ y and x $\prec$ y, then x=y.
4. Transitivity: if x $\succ$ y and y $\succ$ z, then x $\succ$ z (similarly for the $\prec$ relation).

The architecture of the CmpNN adopts a weight sharing technique which ensures that the reflexivity property and the equivalence property between $\prec$ and $\succ$ are satisfied[1]. This results in $N_{\succ}([\mathbf{x}, \mathbf{y}])$ being equal to $N_{\prec}([\mathbf{y}, \mathbf{x}])$. The anti-symmetry property fails only if the two output nodes have the same value. Given that these values are real numbers, the probability that this occurs is almost zero. Only the transitivity property is not guaranteed to be respected by the comparator. However, the results in [10] show that in practice the model is respecting this property in most cases.

We now explain the constraints enforced on the weight architecture to ensure that the reflexivity and the equivalence properties hold. We do so by detailing the illustration given in [10] and shown in figure 3.1. Let $v_{x_k,i}/v_{y_k,i}$ denote the weight of the link from the k-th feature of document $\mathbf{x}/\mathbf{y}$ to the i-th hidden node h_i and let $w_{i,\succ}/w_{i,\prec}$ denote the weight of the link from the i-th hidden node h_i to the output node $N_{\succ}/N_{\prec}$. Let b_i be the bias of the hidden node h_i and $b_{\succ}/b_{\prec}$ be the bias of the output node $N_{\succ}/N_{\prec}$. For each hidden neuron h_i, there exists a dual neuron $h_{i'}$ whose weights are shared with h_i according to following schema:

1. $v_{x_k,i'} = v_{y_k,i}$ and $v_{y_k,i'} = v_{x_k,i}$ hold, i.e., the weights from x_k, y_k to h_i are swapped in the connections to $h_{i'}$.
2. $w_{i',\succ} = w_{i,\prec}$ and $w_{i',\prec} = w_{i,\succ}$ hold, i.e., the weights of the connections from the hidden node h_i to the outputs $N_{\succ}, N_{\prec}$ are swapped in the connections leaving from $h_{i'}$.
3. $b_i = b_{i'}$ and $b_{\succ} = b_{\prec}$ hold, i.e., the biases are shared between the dual hidden neurons h_i and $h_{i'}$ and between the outputs $N_{\succ}$ and $N_{\prec}$.

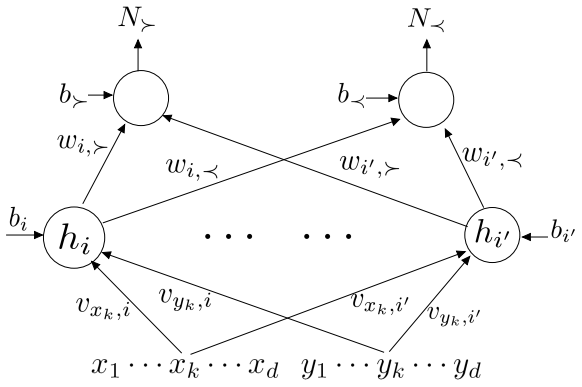


FIGURE 3.1: CmpNN architecture - Figure taken from [10]

[1] See [10] for proof.

The CmpNN we train, learns a preference function to tell which document among a pair is the most relevant one. In order to sort all documents (with respect to each query), the learned preference function must be embedded into a classical sorting algorithm. Given that the transitive property can't be guaranteed, the sorting algorithm may produce different orderings for different initial shufflings of the document set. Experiments in [10] show that the difference in ranking results in function of the initial shuffling is negligible. As a consequence, the choice of the sorting algorithm has almost no impact on the performances. In this thesis, we simply use the default sorting algorithm of Python[2].

Although L. Rigutini et al. highlight the properties of the CmpNN architecture, they haven't studied whether this type of architecture outperforms a standard dense architecture (incapable of ensuring the reflexivity and equivalence properties) with concrete figures. Therefore, we examine if the CmpNN architecture indeed makes a significant improvement over a standard dense architecture or if it is only advantageous from a theoretic point of view.

Table 3.1 summarizes the performance scores for both the standard dense architecture and the CmpNN architecture. We observe that for both the validation and the test set, all scores are higher when using the CmpNN architecture. To validate the results, we apply the Welch's t-test. The 1.5% increase (from 0.244 to 0.259) of the MAP score on the test set corresponds to a p-value of 3.10^{-4}. The 4.0% increase (from 0.40 to 0.44) of the P@1/NDCG@1 score on the test set corresponds to a p-value of 1.10^{-3}. This confirms that ensuring that the reflexivity and equivalence properties hold with the use of a CmpNN improves the performances significantly.

Validation set

DENSE NN					**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
Pair error	Avg	0.137	**Precision**	Avg	0.40	0.37	0.34	0.30	0.28	0.26	0.24	0.23	0.22	0.21
	Std	0.004		Std	0.03	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01
MAP	Avg	0.247	**NDCG**	Avg	0.40	0.38	0.36	0.35	0.34	0.34	0.34	0.34	0.34	0.34
	Std	0.010		Std	0.03	0.03	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01

CMPNN					**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
Pair error	Avg	**0.127**	**Precision**	Avg	**0.44**	**0.40**	**0.36**	**0.33**	**0.30**	**0.28**	**0.27**	**0.25**	**0.23**	**0.22**
	Std	0.005		Std	0.04	0.03	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01
MAP	Avg	**0.271**	**NDCG**	Avg	**0.44**	**0.41**	**0.40**	**0.38**	**0.38**	**0.37**	**0.37**	**0.37**	**0.37**	**0.36**
	Std	0.016		Std	0.04	0.03	0.03	0.02	0.02	0.02	0.02	0.02	0.02	0.02

Test set

DENSE NN					**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
Pair error	Avg	0.137	**Precision**	Avg	0.40	0.35	0.32	0.30	0.28	0.26	0.25	0.23	0.22	0.21
	Std	0.004		Std	0.04	0.03	0.03	0.03	0.02	0.02	0.02	0.01	0.01	0.01
MAP	Avg	0.244	**NDCG**	Avg	0.40	0.35	0.35	0.34	0.34	0.34	0.34	0.34	0.34	0.34
	Std	0.017		Std	0.04	0.04	0.03	0.03	0.03	0.03	0.02	0.02	0.02	0.02

CMPNN					**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
Pair error	Avg	**0.127**	**Precision**	Avg	**0.44**	**0.38**	**0.34**	**0.32**	**0.30**	**0.28**	**0.26**	**0.25**	**0.24**	**0.22**
	Std	0.005		Std	0.05	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01
MAP	Avg	**0.259**	**NDCG**	Avg	**0.44**	**0.39**	**0.38**	**0.37**	**0.36**	**0.36**	**0.36**	**0.36**	**0.36**	**0.36**
	Std	0.013		Std	0.05	0.03	0.03	0.02	0.02	0.02	0.02	0.02	0.02	0.02

TABLE 3.1: Comparison of performances (MAP, P@n, NDCG@n) on validation and test set between a standard dense neural network and the CmpNN.[3]

[2]The default sorting algorithm of Python is the *TimSort* algorithm.

[3]The hidden layer of both networks consists of 10 hidden neurons. Experiments have shown that the number of hidden neurons has almost no impact on the results.

3.2 Loss functions

L. Rigutini et al. have suggested that future research should include the investigation of the performances of the CmpNN using different loss functions. In this section, we therefore compare several loss functions. The loss function which has been used so far is the mean squared error (MSE). We analyse two other classical loss functions, namely the mean absolute error (MAE) and the cross-entropy. Moreover, we test the fidelity loss, the loss function proposed in [16]. Compared to the cross-entropy, the fidelity loss has a zero minimum and is bounded between 0 and 1. According to [16], this should boost the performances. The error for each document pair in terms of each loss function is precised hereafter:

1. Mean squared error (MSE):

$$E([\mathbf{x},\mathbf{y}],\mathbf{t}) = \frac{1}{2}\Big((t_1 - N_{\succ}([\mathbf{x},\mathbf{y}]))^2 + (t_2 - N_{\prec}([\mathbf{x},\mathbf{y}]))^2\Big)$$

2. Mean absolute error (MAE):

$$E([\mathbf{x},\mathbf{y}],\mathbf{t}) = \frac{1}{2}\Big(|t_1 - N_{\succ}([\mathbf{x},\mathbf{y}])| + |t_2 - N_{\prec}([\mathbf{x},\mathbf{y}])|\Big)$$

3. Cross-entropy (Log loss):

$$E([\mathbf{x},\mathbf{y}],\mathbf{t}) = \frac{1}{2}\Big(t_1 log(N_{\succ}([\mathbf{x},\mathbf{y}])) + (1-t_1)log(1-N_{\succ}([\mathbf{x},\mathbf{y}])) \; +$$

$$t_2 log(N_{\prec}([\mathbf{x},\mathbf{y}])) + (1-t_2)log(1-N_{\prec}([\mathbf{x},\mathbf{y}]))\Big)$$

4. Fidelity loss:

$$E([\mathbf{x},\mathbf{y}],\mathbf{t}) = \frac{1}{2}\Big(1 - \sqrt{t_1 N_{\succ}([\mathbf{x},\mathbf{y}])} - \sqrt{(1-t_1)(1-N_{\succ}([\mathbf{x},\mathbf{y}]))} \; +$$

$$1 - \sqrt{t_2 N_{\prec}([\mathbf{x},\mathbf{y}])} - \sqrt{(1-t_2)(1-N_{\prec}([\mathbf{x},\mathbf{y}]))}\Big)$$

Looking at figure 3.2, we see that none of the novel loss functions improves the performances. Both the cross-entropy and the fidelity loss function achieve similar performances as the MSE. The MAE loss function has the worst results for all evaluation measures. The likely reason is the fact that this loss function is not differentiable at the origin. This causes problems for the gradient descent optimization. As no improvements are observed, we continue our further experiments with the MSE loss function.

3.3 Learning set configuration

The webpages in the TD2003 dataset are either relevant or irrelevant with respect to a certain query. Obviously, this has some repercussions on the configuration of the learning set. In all previous experiments, only document pairs consisting of documents with a different relevance degree have been used in the learning process. This has been done for a reason, as the experiments hereafter will make clear.

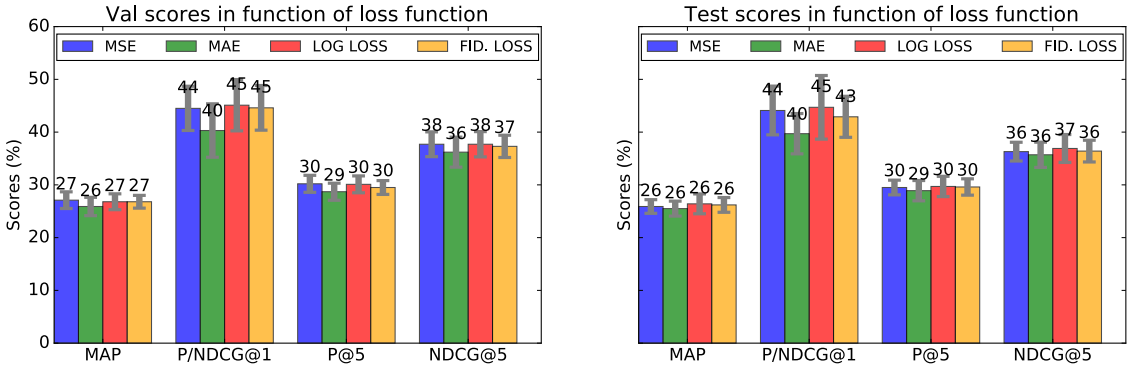


FIGURE 3.2: Comparison of performances (MAP, P@n, NDCG@n) on validation and test set in function of different loss functions: MSE, MAE, Log loss and Fidelity loss.

Three types of document pairs can be built. A pair consisting of two relevant documents (denoted as a R-R pair), a pair consisting of two irrelevant documents (denoted as a I-I pair) and a pair consisting of a relevant and a irrelevant document (denoted as R-I pair). Training the model on all possible pairs is clearly a bad idea. First of all, it is computationally too expensive since the number of pairs is quadratic in the order of the numbers of documents. Moreover, the number of irrelevant documents is much higher than the number of relevant documents. This unbalanced distribution has been shown in chapter 2 in figure 2.1 and may be harmful for the performances if not taken into account correctly.

Let's analyse the performances in terms of the types of document pairs used in the learning process. We examine five possible configurations:

(A) The three aforementioned types of document pairs are used in the learning process, with the unbalanced distribution between the relevant and irrelevant documents not taken into account.

(B) The three types of document pairs are used in the learning process, with the unbalanced distribution between the relevant and irrelevant documents taken into account. An equal number of these three types of pairs are sampled.

(C) Only pairs consisting of one relevant and one irrelevant document are used.

(D) Pairs consisting of one relevant and one irrelevant document, as well as pairs consisting of two relevant documents are used. An equal number of these two types of pairs are sampled.

(E) Pairs consisting of one relevant and one irrelevant document, as well as pairs consisting of two irrelevant documents are used. An equal number of these two types of pairs are sampled.

Figure 3.3 shows the performances in function of the types of document pairs used in the training process. As expected, we see that randomly selecting pairs out of all possible pairs without taking the unbalanced distribution between the relevant and irrelevant documents achieves the worst results. The best results are obtained when the learning process only considers pairs of documents with a different relevance level. Intuitively, one could think that adding some pairs with the same relevance degree above the pairs of documents with different relevance degrees could be beneficial. When ranking a list of documents, the learned preference function must tell which one of two documents is the most relevant one,

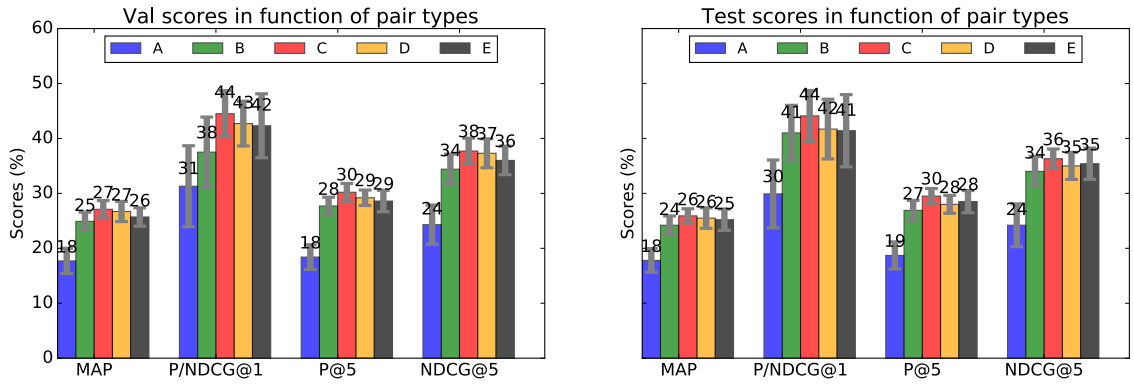


FIGURE 3.3: Comparison of performances (MAP, P@n, NDCG@n) on validation and test set using different document pair types in the learning process:
(A) R-I, R-R and I-I pairs are sampled, with the unbalanced distribution between the relevant and irrelevant documents not taken into account.
(B) Equal number of R-I, R-R and I-I pairs are sampled.
(C) Only R-I pairs are sampled.
(D) Equal number of R-I and R-R pairs are sampled.
(E) Equal number of R-I and I-I pairs are sampled.
R-I pair: Training pair consisting of one relevant and one irrelevant document; R-R pair: Training pair consisting of two relevant documents; I-I pair: Training pair consisting of two irrelevant documents.

even though in most cases they have the same relevance degree. Therefore, adding such pairs to the training process may seem helpful. The experiments show us that this isn't the case. The model works best when it only focuses on the difference between relevant and irrelevant documents.

Besides the aforementioned settings (A), (B), (C), (D) and (E), the settings (B), (D) and (E) have been investigated where the number of R-I pairs were in a majority compared to the other types of pairs (66% - 33% and 75% - 25%). Once more, the configuration consisting solely of pairs with the same relevance degrees achieved the best results.

A question which one may ask, is whether these findings are generalizable or if they simply apply to our particular setting. It appears that no concrete research has been done on the configuration of the learning set for algorithms adopting the pairwise approach. As this isn't the primary objective of this thesis, we won't investigate this any further. However, this may be an interesting future direction to explore, especially if the documents are characterized by more than two levels of relevance degrees.

Chapter 4

Deep network

After analysing thoroughly shallow neural networks, we now study whether deep architectures are able to improve the ranking performances by adding more hidden layers. Training deep neural networks is known to be a difficult task. Therefore, we apply some strategies that are the best suited for deep learning, according to researchers.

First, we simply increase the number of hidden layers (section 4.1). Then, we add some regularization (section 4.2). We apply the dropout technique as well as more classical regularization techniques, such as the l2-regularization. Afterwards, we try out new initialization schemes and activation functions (section 4.3). Finally, we summarize briefly all our results obtained so far (section 4.4) and compare them with state-of-the-art algorithms (section 4.5).

4.1 Multiple hidden layers

In this section, we simply add some hidden layers to our 1-hidden layered CmpNN and analyse the resulting performances. When adding these hidden layers, we ensure that we maintain the reflexivity and the equivalence properties that are naturally present in a preference function.

Enforcing the constraints on the weight architecture of a deep neural network to ensure that the two aforementioned properties are satisfied, is done in a similar way as in the regular CmpNN consisting of one hidden layer (suggested in [10]). We explain how to achieve this by detailing the illustration, shown in figure 4.1, of a deep CmpNN consisting of multiple hidden layers.

Let's consider a deep neural network of p hidden layers, where each hidden layer $m \in [1, 2, ..., p]$ has $2n_m$ hidden neurons. Let $v^1_{x_k,i}/v^1_{y_k,i}$ denote the weight of the link from the k-th feature of document $\mathbf{x}/\mathbf{y}$ to the i-th hidden node h^1_i lying in the first hidden layer and let $w_{i,\succ}/w_{i,\prec}$ denote the weight of the link from the i-th hidden node h^p_i lying in the last hidden layer p to the output node $N_\succ/N_\prec$. Let $v^m_{i,j}/v^m_{i',j}$ denote the weight of the link from the hidden node $h^{m-1}_i/h^{m-1}_{i'}$ lying in the hidden layer $m-1$ to the j-th hidden node h^m_j lying in the hidden layer m. Let b^m_i be the bias of the i-th hidden node h^m_i lying in the hidden layer m and $b_\succ/b_\prec$ be the bias of the output node $N_\succ/N_\prec$. For each hidden neuron h^m_i, there exist a dual neuron $h^m_{i'}$ whose weights are shared with h^m_i according to following schema:

1. For the weights between the input layer and the first hidden layer, we have: $v^1_{x_k,i'} = v^1_{y_k,i}$ and $v^1_{y_k,i'} = v^1_{x_k,i}$ hold, i.e., the weights from x_k, y_k to h^1_i are swapped in the connections to $h^1_{i'}$.

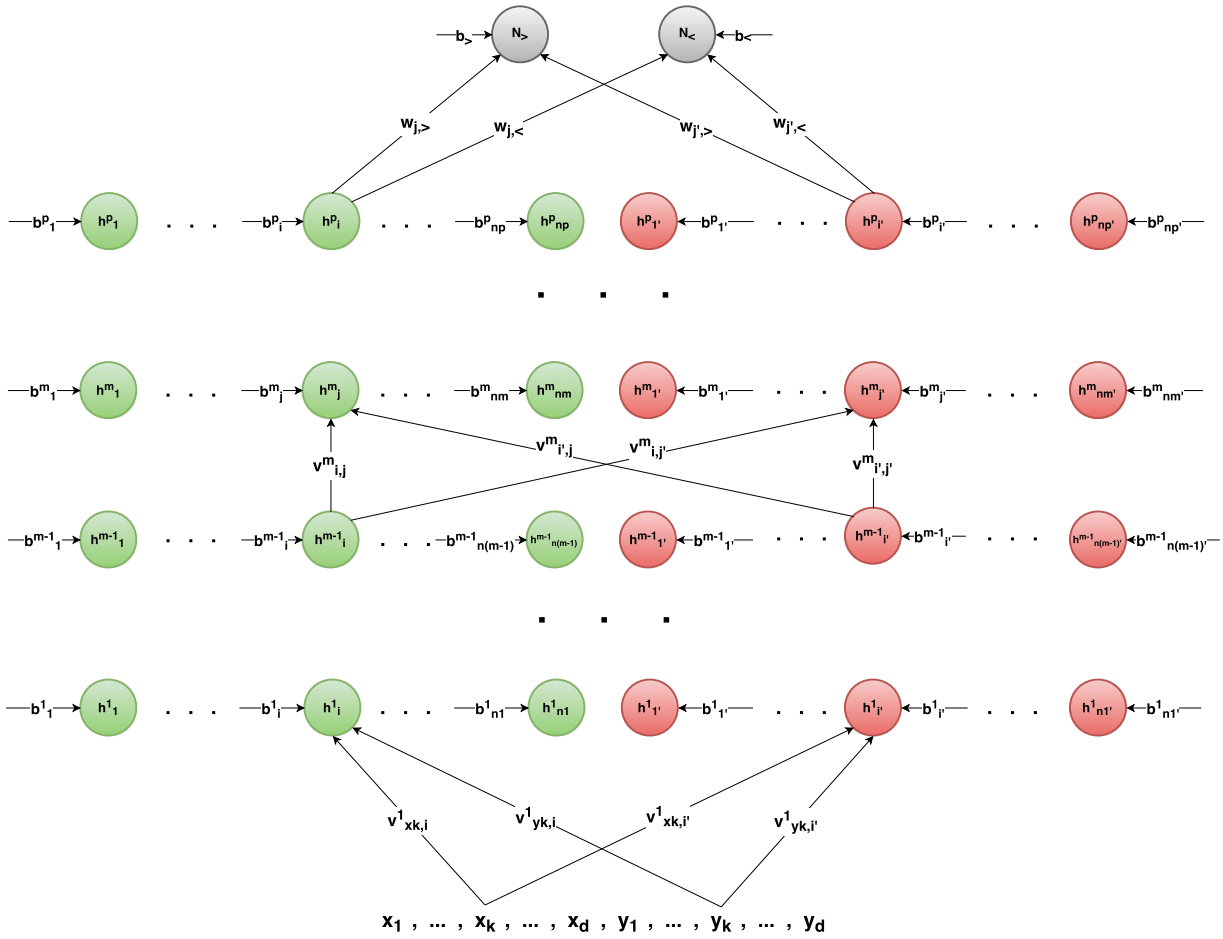


FIGURE 4.1: Deep CmpNN architecture

2. For the weights between the last hidden layer and the output layer, we have: $w_{j',\succ} = w_{j,\prec}$ and $w_{j',\prec} = w_{j,\succ}$ hold, i.e., the weights of the connections from the hidden node h_j^p to the outputs $N_\succ, N_\prec$ are swapped in the connections leaving from $h_{j'}^p$.
3. For the weights between the hidden layers $m-1$ and m, we have: $v_{i,j'}^m = v_{i',j}^m$ and $v_{i',j'}^m = v_{i,j}^m$ hold, i.e., the weights from $h_i^{m-1}, h_{i'}^{m-1}$ to h_j^m are swapped in the connections to $h_{j'}^m$.
4. For each pair of hidden neurons h_i^m and $h_{i'}^m$ in each hidden layer m, we have: $b_i^m = b_{i'}^m$ hold, i.e., the biases are shared between the dual hidden neurons h_i^m and $h_{i'}^m$.
5. For the two output nodes $N_\succ$ and $N_\prec$, we have: $b_\succ = b_\prec$ hold, i.e., the biases are shared between the outputs $N_\succ$ and $N_\prec$.

Let's prove that the reflexivity property and equivalence property between $\succ$ and $\prec$ are satisfied for this deep CmpNN. To do so, we further elaborate the proof detailed in [10], which is a proof for a CmpNN consisting of one hidden layer.

In the proof, $h_i^m([\mathbf{x}, \mathbf{y}])$ denotes the output of the i-th hidden node lying in the m-th hidden layer when $[\mathbf{x}, \mathbf{y}]$ is introduced in the network, whereas σ denotes the used activation function.

Applying the weight-sharing rules of point 1 and 4, we obtain:

$$\begin{aligned} h_i^1([\mathbf{x},\mathbf{y}]) &= \sigma\Big(\sum_k (v^1_{x_{k,i}} x_k + v^1_{y_{k,i}} y_k) + b^1_i\Big) \\ &= \sigma\Big(\sum_k (v^1_{x_{k,i'}} y_k + v^1_{y_{k,i'}} x_k) + b^1_{i'}\Big) \\ &= h_{i'}^1([\mathbf{y},\mathbf{x}]) \end{aligned}$$

Having proved that $h_i^1([\mathbf{x},\mathbf{y}]) = h_{i'}^1([\mathbf{y},\mathbf{x}])$ makes it possible to prove $h_i^2([\mathbf{x},\mathbf{y}]) = h_{i'}^2([\mathbf{y},\mathbf{x}])$, which at its turn makes it possible to prove $h_i^3([\mathbf{x},\mathbf{y}]) = h_{i'}^3([\mathbf{y},\mathbf{x}])$. This way of proceeding can be repeated for all hidden layers. As a matter of fact, the relation $h_i^m([\mathbf{x},\mathbf{y}]) = h_{i'}^m([\mathbf{y},\mathbf{x}])$ (of the hidden layer m) can be obtained from the relation $h_i^{m-1}([\mathbf{x},\mathbf{y}]) = h_{i'}^{m-1}([\mathbf{y},\mathbf{x}])$ (of the previous hidden layer $m-1$) while applying the weight-sharing rules of point 3 and 4:

$$\begin{aligned} h_j^m([\mathbf{x},\mathbf{y}]) &= \sigma\Big(\sum_{i,i'} (v^m_{i,j} h_i^{m-1}([\mathbf{x},\mathbf{y}]) + v^m_{i',j} h_{i'}^{m-1}([\mathbf{x},\mathbf{y}])) + b^m_j\Big) \\ &= \sigma\Big(\sum_{i,i'} (v^m_{i,j'} h_i^{m-1}([\mathbf{y},\mathbf{x}]) + v^m_{i',j'} h_{i'}^{m-1}([\mathbf{y},\mathbf{x}])) + b^m_{j'}\Big) \\ &= h_{j'}^m([\mathbf{y},\mathbf{x}]) \end{aligned}$$

Applying the weight-sharing rules of point 2 and 5 and given that $h_i^p([\mathbf{x},\mathbf{y}]) = h_{i'}^p([\mathbf{y},\mathbf{x}])$, we obtain:

$$\begin{aligned} N_{\succ}([\mathbf{x},\mathbf{y}]) &= \sigma\Big(\sum_{i,i'} (w_{i,\succ} h_i^p([\mathbf{x},\mathbf{y}]) + w_{i',\succ} h_{i'}^p([\mathbf{x},\mathbf{y}])) + b_{\succ}\Big) \\ &= \sigma\Big(\sum_{i,i'} (w_{i,\prec} h_i^p([\mathbf{y},\mathbf{x}]) + w_{i',\prec} h_{i'}^p([\mathbf{y},\mathbf{x}])) + b_{\prec}\Big) \\ &= N_{\prec}([\mathbf{y},\mathbf{x}]) \end{aligned}$$

We have thus proved that $N_{\succ}([\mathbf{x},\mathbf{y}]) = N_{\prec}([\mathbf{y},\mathbf{x}])$. This implies that the equivalence property between $\succ$ and $\prec$ is satisfied. As a consequence, the reflexivity property is guaranteed as well given that $N_{\succ}([\mathbf{x},\mathbf{x}]) = N_{\prec}([\mathbf{x},\mathbf{x}])$.

Let's now analyse the performances of a deep CmpNN consisting of multiple hidden layers. The number of possible network architecture configurations is infinite. Besides the number of hidden layers, the number of hidden neurons is a hyper-parameter to tune as well. This makes it impossible to test every possible configuration. Furthermore, adding hidden layers and increasing the number of hidden neurons require very expensive computations. Experiments now take several days to be executed. In this thesis, we consider neural networks consisting of 1, 2, 3 and 4 hidden layers. Table 4.1 summarizes the structure of the four different sized neural networks, whose performances are analysed in subsequent experiments, by specifying the number of hidden neurons in each layer.

#Hidden layers	#Input neurons	#Hidden neurons in each successive hidden layer	#Output neurons
1	88	6	2
2	88	$12 \rightarrow 6$	2
3	88	$24 \rightarrow 12 \rightarrow 6$	2
4	88	$48 \rightarrow 24 \rightarrow 12 \rightarrow 6$	2

TABLE 4.1: Configuration of the different deep CmpNN, consisting of 1, 2, 3 and 4 hidden layers, used for the subsequent experiments.

A few other network architecture configurations with a different number of neurons in each hidden layer have been analysed as well. Experiments have shown that there was almost no impact on the performances. These experiments have been carried out using neural networks with a decreasing number of hidden neurons from one layer to the other, as this is the most standard procedure. Future directions may include the investigation of deep neural network configurations where this constraint is not necessarily respected.

Figure 4.2 summarizes the performance scores in function of the number of hidden layers. As can be observed, no improvements are made by adding more hidden layers. The results are even slightly worse. We try to solve this issue by adding regularization in the next section.

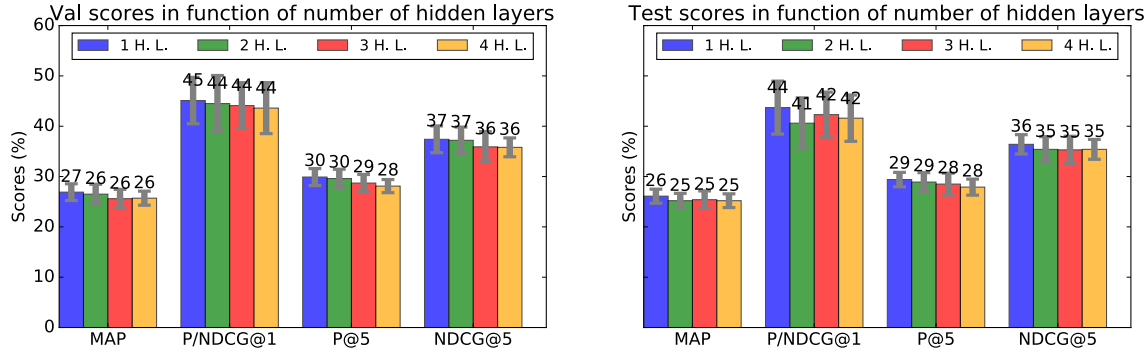


FIGURE 4.2: Comparison of performances (MAP, P@n, NDCG@n) on the validation and test set between CmpNN with 1, 2, 3 and 4 hidden layers.

4.2 Regularization

4.2.1 Dropout

Dropout [24] is a regularization technique principally applied to reduce overfitting of deep neural networks. Moreover, it speeds the learning process as the resulting network becomes smaller. Dropout means literally dropping neurons (temporarily) during the learning process. Input and hidden neurons are removed from the network architecture, along with their incoming and outgoing connections. The removal of these neurons is done randomly. Each neuron has a probability p of being kept.

Dropout results in a thinned network. Each neuron that withstood the dropout process, must learn to work with a randomly selected set of other neurons. This process makes the neurons robust. Each neuron becomes less dependent on other neurons and tries to make more decisions on its own. It prevents complex co-adaptations on training data.

We have applied dropout regularization to the four different sized neural networks presented in the previous section (see table 4.1). The hyper-parameter p, for which the values $[0, 0.1, 0.2, ..., 0.8]$) have been considered, has been tuned through cross-validation for each network configuration. The results are shown in figure 4.3. The first thing we conclude from the figure, is that applying dropout regularization improves the results of all network configurations, regardless of the number of hidden layers. For instance, without the dropout regularization, we had MAP scores around 0.26 on both the validation and test set. The MAP scores now lie around 0.29. Furthermore, we observe that the best results are obtained for the neural networks consisting of 3 and 4 hidden layers, showing the gain

of deep architectures. As there doesn't seem to be a significant performance difference between these two neural networks of different size, we will continue our experiments with the 3-hidden layered CmpNN for computation time reasons.

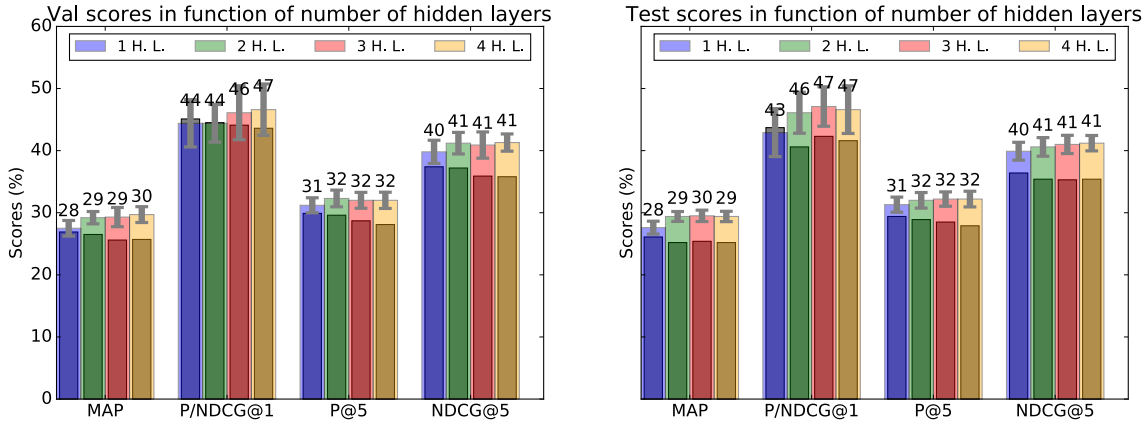


FIGURE 4.3: Comparison of performances (MAP, P@n, NDCG@n) on the validation and test set between CmpNN with 1, 2, 3 and 4 hidden layers, applying dropout regularization. The previously obtained results without the dropout regularization are displayed by the thinner, superimposed bar-charts.

Figure 4.4 shows the evolution of the MAP score in function of the dropout probability p for the 3-hidden layered CmpNN. A peak is observed between 0.1 and 0.4. Tuning the parameter even further on a more refined range between 0.1 and 0.4, gives an optimal value p of 0.25.

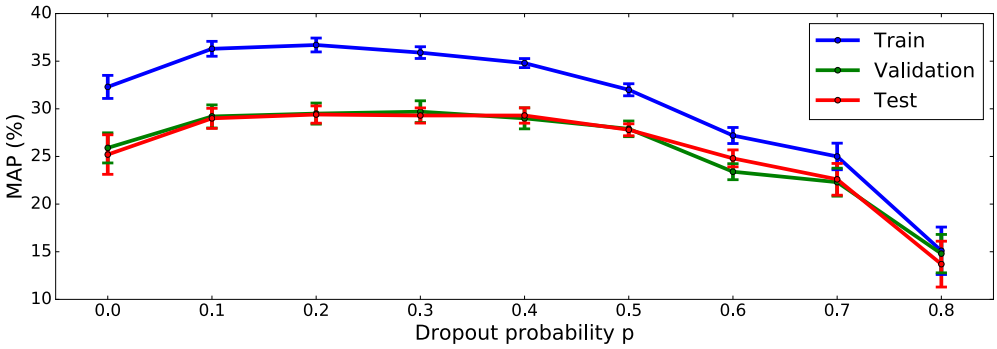
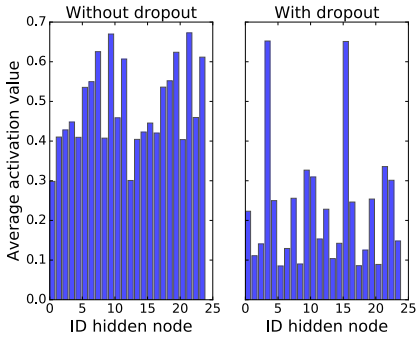
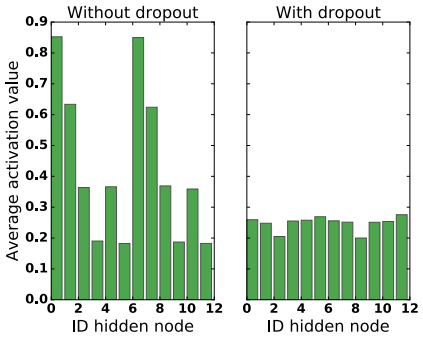


FIGURE 4.4: MAP score on the training, validation and test set of a 3-hidden layered CmpNN in function of the dropout probability p.

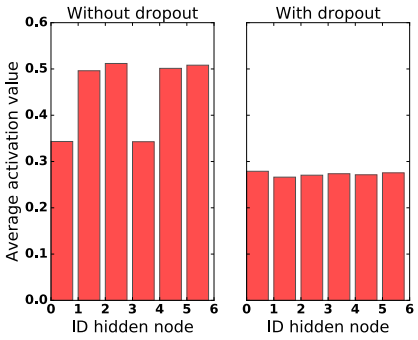
We now try to explain why dropout regularization improves the performances in our case. Figures 4.5a, 4.5b and 4.5c show respectively the average activation value of each neuron in the hidden layers 1, 2 and 3, without and with dropout regularization. We deduce from the figures that when using the dropout technique, the average activation values of the hidden neurons have approximately the same value. This is especially true for the second and third hidden layers. On the other hand, we see that this isn't the case at all for the setting without the dropout regularization. The average activation values fluctuate a lot. This observation indicates that each neuron becomes robust thanks to the regularization. Consequently, every neuron is able to find useful features on its own. Each neuron contributes to the overall performance in a more or less equal way, leading to better results.



(A) Hidden layer 1



(B) Hiden layer 2



(C) Hidden layer 3

FIGURE 4.5: Average activation value of each neuron in all hidden layers, considering a 3-hidden layered CmpNN without and with dropout regularization.

Table 4.5 summarizes the performance scores comparing our initial 1-layered CmpNN consisting of 10 hidden neurons for which no regularization has been applied (results obtained at the end of chapter 3) with our new 3-hidden layered CmpNN for which we have applied the dropout technique. We see an important improvement of all performance scores. To validate the results, we apply the Welch's t-test. The 3.9% increase (from 0.259 to 0.298) of the MAP score corresponds to a p-value of 1.10^{-17}. The 3% increase (from 0.44 to 0.47) of the P@1/NDCG@1 score corresponds to a p-value of 7.10^{-3}. This proves that the new results are significantly better.

SETTING 1 - TEST SET					**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
Pair error	Avg	0.127	**Precision**	Avg	0.44	0.38	0.34	0.32	0.30	0.28	0.26	0.25	0.24	0.22
	Std	0.005		Std	0.05	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01
MAP	Avg	0.259	**NDCG**	Avg	0.44	0.39	0.38	0.37	0.36	0.36	0.36	0.36	0.36	0.36
	Std	0.013		Std	0.05	0.03	0.03	0.02	0.02	0.02	0.02	0.02	0.02	0.02

SETTING 2 - TEST SET					**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
Pair error	Avg	**0.118**	**Precision**	Avg	**0.47**	**0.43**	**0.39**	**0.35**	**0.32**	**0.30**	**0.29**	**0.27**	**0.26**	**0.24**
	Std	0.002		Std	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01
MAP	Avg	**0.298**	**NDCG**	Avg	**0.47**	**0.46**	**0.43**	**0.42**	**0.41**	**0.41**	**0.41**	**0.41**	**0.41**	**0.40**
	Std	0.005		Std	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01

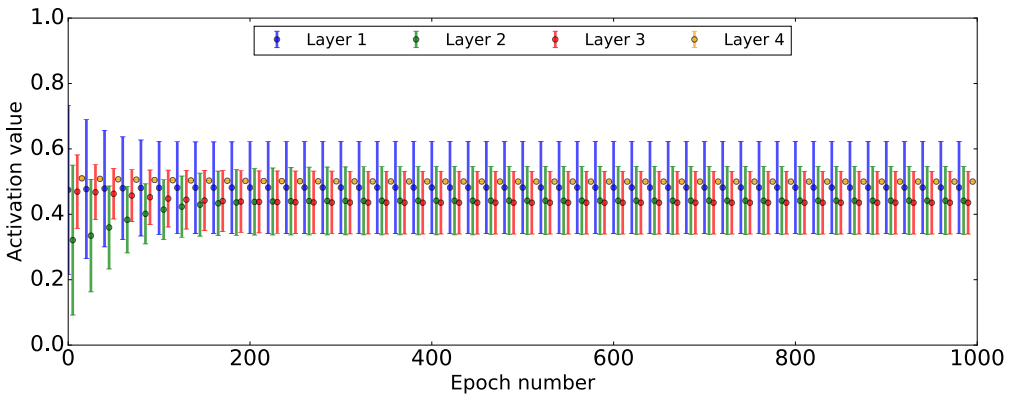
TABLE 4.2: Comparison of performances (MAP, P@n, NDCG@n)
Setting 1 - Results obtained at the end of chapter 3: 1 hidden-layered CmpNN, uniform initialization, sigmoid activation, no regularization.
Setting 2 - New results: 3-hidden layered CmpNN, uniform initialization, sigmoid activation, dropout $p = 0.25$.

4.2.2 l2-regularization

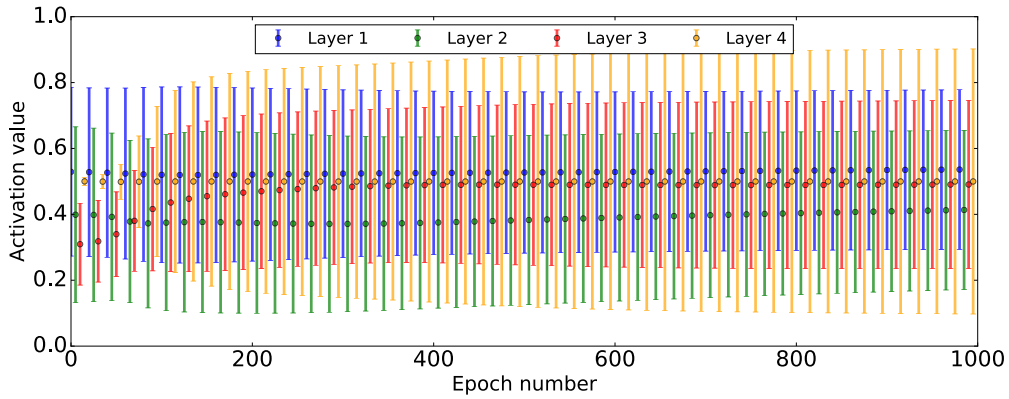
Dropout is perhaps the biggest invention in the field of (deep) neural networks. However, other regularization techniques exist. It is worthwhile to take a closer look at them.

In this subsection, we focus on the l2-regularization and study whether it can match or even outperform the performances of the dropout technique. l2-regularization prevents overfitting by penalizing the squared magnitude of the weight parameters in the loss function. A term $\frac{1}{2}\lambda w_i^2$ is added to the original loss function for each weight w_i present in the neural network architecture. The idea behind the l2-regularization is to penalize peaky weight values.

A common default value for the l2-regularization strength parameter λ is 0.01. This value is also the default value suggested by Keras. However, using this default value results in catastrophic performances for the four different sized networks presented earlier (see table 4.1). We obtain for instance a MAP score of approximately 0.08 for the network consisting of 3 hidden layers. Trying to figure out the issue, we have plotted the average and standard deviation of the activation values for the different layers of that particular network. The results are shown in figure 4.6a. Looking at the plot, it becomes clear that the output layer saturates. It isn't able to learn anything useful. Whatever the input, it is always predicting 0.5 for both outputs. Smaller l2-regularization values must therefore be examined.



(A) $\lambda = 0.01$



(B) $\lambda = 0.0001$

FIGURE 4.6: Average and standard deviation of the neuron's activation values for the different layers, considering a 3-hidden layered CmpNN with the l2-regularization hyper-parameter λ set to 0.01 and 0.0001.

Figure 4.7 shows the MAP score of the training, validation and test set in function of the l2-regularization strength hyper-parameter λ for the 3-hidden layered CmpNN. The best performances are obtained for λ equal to 10^{-4}. A new plot of the average and standard deviation of the activation values for the different layers of the network, shown in figure 4.6b, indicates that there is no saturation anymore. As with the dropout technique, the l2-regularization improves the performances. However, the current performances are somewhat lower. The MAP scores on the validation and test set are respectively equal to 0.284 and 0.281, whereas these were equal to 0.297 and 0.298 when applying dropout regularization. Additional experiments are required to confirm whether the dropout regularization is indeed more efficient.

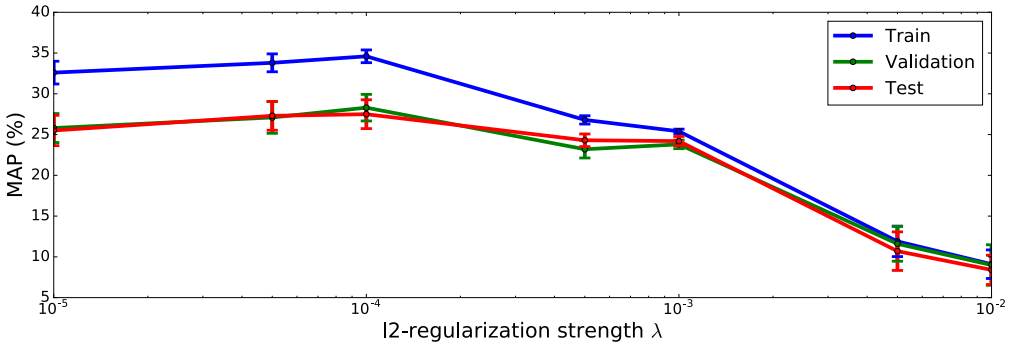
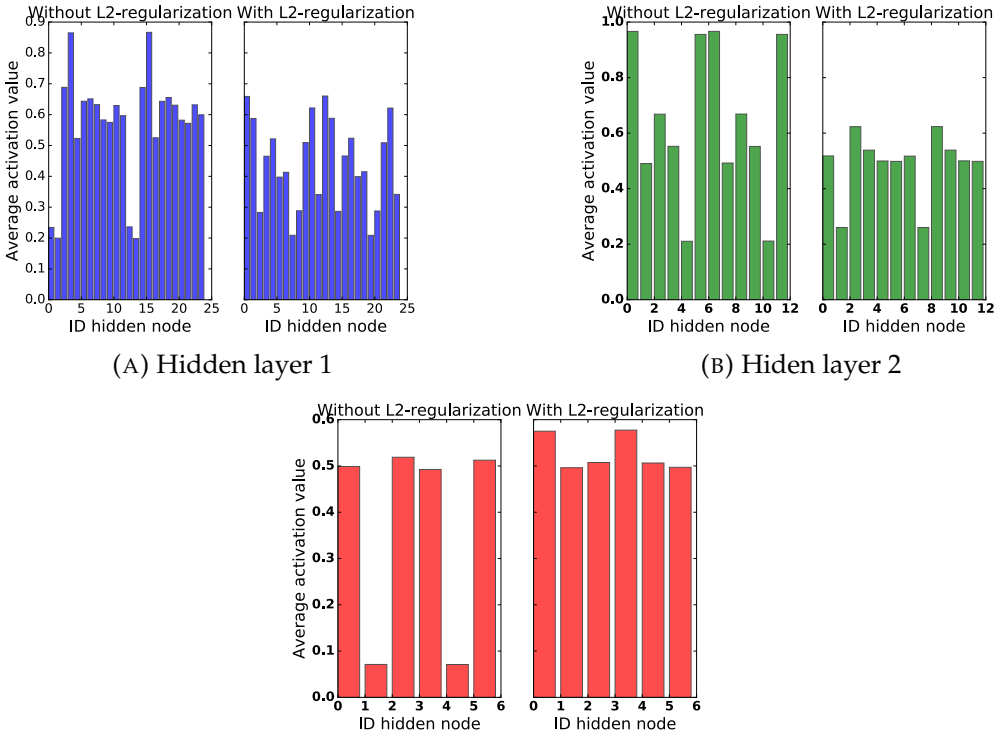


FIGURE 4.7: MAP score on the training, validation and test set of a 3-hidden layered CmpNN in function of the l2-regularization strength λ.

A more accurate fine-tuning of the hyper-parameter λ is therefore performed on the range $[5.10^{-5}, 5.10^{-4}]$. However, the obtained performances are still smaller than the ones obtained with the dropout regularization on both the validation set and the test set. Similar results are observed for the other neural networks of different size.

Finding the exact explanation why dropout works better in our case than the more classical l2-regularization is not trivial. Figure 4.8 may provide a likely reason. As in figure 4.5, the average activation value of each hidden neuron for the 3-hidden layered CmpNN has been displayed. Comparing both figures, we notice that the activation values with the dropout technique are more equally scattered. This is especially the case for the second and third hidden layer. The activation values obtained with the l2-regularization contain a larger number of peaks. The fact that the activation values are more equally diffused with the dropout technique shows that each neuron is more robust. The output doesn't depend on the value of a single dominating neuron. Dropout prevents the neurons from co-adapting too much. Every neuron captures useful information, resulting in better performances. These results are a bit surprising, given that penalizing peaky weight values is exactly what the l2-regularization is about.

Other forms of regularization such as the l1-regularization and the Elastic net regularization have been analysed as well. However, none of them were capable of outperforming the dropout technique. We therefore continue our subsequent experiments with a 3-hidden layered CmpNN, which we regularize with the dropout technique by setting the dropout probability p to 0.25.



(A) Hidden layer 1

(B) Hiden layer 2

(C) Hidden layer 3

FIGURE 4.8: Average activation value of each neuron in all hidden layers, considering a 3-hidden layered CmpNN without and with l2-regularization.

4.3 Initialization and Activation

The performances of deep neural networks can be further improved with smarter initialization schemes and more suited activation functions. In the two next subsections 4.3.1 and 4.3.2, we describe some of these initialization schemes and activation functions. Using cross-validation, we determine which setting achieves the best results on the validation set and present the results on the test set in subsection 4.3.3.

4.3.1 Initialization

A wide variety of weight initialization schemes for deep neural network exists. In this thesis, we analyse four different ways to initialize the weights:

1. Uniform: $Weights \sim Unif\left[-1, 1\right]$
2. *Glorot* uniform: $Weights \sim Unif\left[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right]$,

   where n_{in}/n_{out} is the number of neurons in the input/output layer with respect to the weights.
3. *He* uniform: $Weights \sim Unif\left[-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right]$,

   where n_{in} is the number of neurons in the input layer with respect to the weights.
4. Unsupervised pre-training through the use of stacked auto-encoders.

Uniform

This is the most standard way to initialize the weights of a neural network. All weights are uniformly assigned a value between -1 and 1.

Glorot Uniform

In [25], a new initialization scheme is proposed, which is especially suited for deep neural networks. The issue of a standard uniform initialization is the saturation of the top hidden layers. The *Glorot* uniform initialization tries to solve the issue by initializing the weights in such a way to maintain the activation variances and back-propagated gradient variances as one moves up or down the network. Basically, it helps the signals to reach deep into the network. If the weights are too small, the signal gradually shrinks from one layer to the other with the risk of vanishing before reaching the output layer. On the other hand, if the weights are too big, the signal gradually grows from one layer to the other with the risk of being too gigantic to be helpful. The *Glorot* uniform scheme prevents this from happening. We explain the reasons, inspired by the explanations given in Andy's blog[1].

Let $\mathbf{x} = [x_1, x_2, ..., x_n]$ be the input of a certain layer L^{in} and let y be the output of a certain neuron lying in the subsequent layer L^{out}. The output y is obtained by computing the dot product between $\mathbf{x}$ and the weights $\mathbf{w} = [w_1, w_2, ..., w_n]$ connecting $\mathbf{x}$ to y:

$$y = \mathbf{w} \cdot \mathbf{x} = w_1 x_1 + w_2 x_2 + ... + w_n x_n$$

We know that the variance of a product of two independent random variables w_i and x_i can be computed as follows:

$$Var(w_i x_i) = E(x_i)^2 Var(w_i) + E(w_i)^2 Var(x_i) + Var(w_i) Var(x_i)$$

This formula can then be further simplified if the inputs and weights have been set such that their mean is equal to 0 (i.e. $E(x_i) = E(w_i) = 0$):

$$Var(w_i x_i) = Var(w_i) Var(x_i)$$

By making the assumption that x_i and w_i are all independent and identically distributed, we obtain that the variance of y is equal to:

$$Var(y) = Var(w_1 x_1 + w_2 x_2 + ... + w_n x_n) = n Var(w_i) Var(x_i)$$

To maintain the variance of the input equal to the variance of the output, $nVar(w_i)$ must be equal to 1. Therefore, the variance of each weight component should be equal to:

$$Var(w_i) = \frac{1}{n} = \frac{1}{n_{in}},$$

where n_{in} is the number of neurons present in the input layer L^{in}.

[1] http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

Applying the same reasoning to the backpropagated signal, we find that:

$$Var(w_i) = \frac{1}{n_{out}},$$

where n_{out} is the number of neurons present in the output layer L^{out}.

Satisfying these two constraints can only be guaranteed if n_{in} is equal to n_{out}. As a compromise, we therefore take the average of the two, leading to the following constraint:

$$Var(w_i) = \frac{2}{n_{in} + n_{out}}$$

Given this constraint and knowing that the variance of a uniform distribution $Unif[a, b]$ is equal to $\frac{1}{12}(b-a)^2$, we are able to deduce the *Glorot* uniform initialization scheme:

$$Weights \sim Unif\left[-\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right]$$

He uniform

The authors of [26] suggest a new initialization scheme, largely inspired by the *Glorot* uniform initialization scheme. The main difference between the two types of initializations is the fact that the first one addresses the non-linearities caused by ReLU activations[2], whereas the latter one only considers linear activations. Furthermore, the new initialization scheme only tries to preserve the variance of the forward propagated signal and not the variance of the back-propagated one. This results in the constraint:

$$Var(w_i) = \frac{2}{n_{in}}$$

The factor 2 present instead of the factor 1 is the consequence of the non-linearities which are taken into consideration. A ReLU activation has the property that about half of the neurons are not active. Therefore, the variance should be doubled to maintain its value throughout the network. Given this constraint and knowing that the variance of a uniform distribution $Unif[a, b]$ is equal to $\frac{1}{12}(b-a)^2$, we are able to deduce the *He* uniform initialization scheme:

$$Weights \sim Unif\left[-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right]$$

Stacked auto-encoders

As the name implies, a stacked auto-encoder is a neural network consisting of multiple layers of auto-encoders. A stacked auto-encoder is used for pre-training [27] a deep neural network. After this process, the network must still be fine-tuned in a supervised fashion with respect to the original loss function. The process of pre-training the different layers is done by a greedy layer-wise unsupervised approach. It is one of the strategies explored

[2]See next subsection.

in [28] and [29]. One layer/auto-encoder is trained at the time, proceeding from the input layer towards the output layer. The parameters of each layer are trained separately while freezing the parameters of the remaining layers. Each auto-encoder consists of two parts, namely the encoder and the decoder. The encoder tries to learn an representative encoding of the input data, which is then decoded by the decoder to replicate this input data by minimizing the reconstruction error. Once the unsupervised pre-training took place, the fine-tuning of the parameters is performed.

A good illustration detailing the way of working of stacked auto-encoders on our particular CmpNN architecture is shown on figures 4.9a, 4.9b and 4.9c. Figure 4.9a shows the overall procedure. One auto-encoder is trained at the time before the fine-tuning takes place. Figure 4.9b shows the training of the first auto-encoder. By introducing the input data $[\mathbf{x}, \mathbf{y}]$ in the network, the auto-encoder learns the primary features $\mathbf{h}^1$, trying to replicate $[\mathbf{x}, \mathbf{y}]$. Then, these primary features $\mathbf{h}^1$ are used at their turn as new inputs to learn the secondary features $\mathbf{h}^2$, as shown on figure 4.9c. This procedure is repeated for all hidden layers.

Stacked auto-encoders are used for two reasons. First of all, it results in a better initialization of the weight components, driving the optimization towards better local optima. Secondly, it performs some implicit regularization. By pre-training the weights before the actual fine-tuning of the parameters, we constrain the weights during the actual fine-tuning to be maintained in a certain range defined by the unsupervised pre-training.

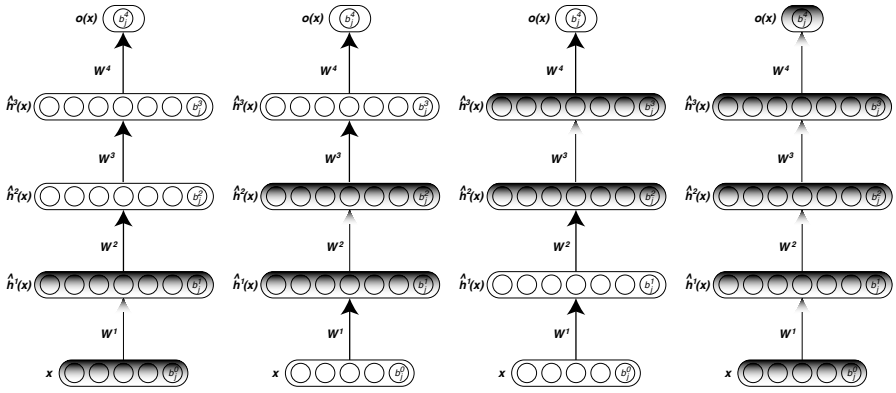
4.3.2 Activation

As it is the case for the initialization schemes of neural network architectures, a wide variety of activation functions exists. The four activation functions whose performances are analysed in this thesis are the sigmoid, the tanh, the ReLU and the softplus functions:

1. Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
2. Tanh: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
3. ReLU: $f(x) = max(0, x)$
4. Softplus: $f(x) = ln(1 + e^{x})$

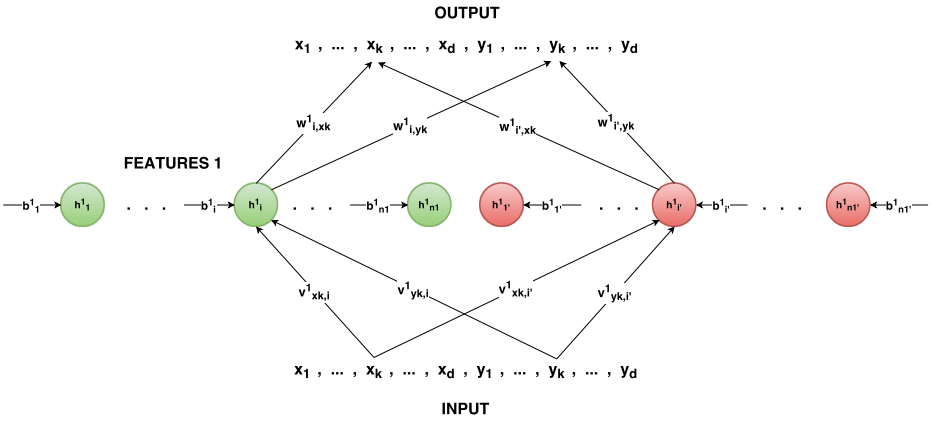
The two most common activation functions are the sigmoid and tanh functions. However, research has shown that these two activation functions are not suited for training deep neural networks. One of the most popular activation function for deep neural networks is the rectified linear unit (ReLU) activation. [31] and [32] show that using this type of activation boosts the performances significantly when using deep networks.

ReLU has two major advantages compared to the sigmoid and tanh activation functions. The first advantage is the reduced probability of a vanishing gradient, arising when the input value x is larger than 0. The gradient of a sigmoid or tanh activation function vanishes as x increases or decreases too much. The gradient of the ReLU activation function is a constant. It is either 0 when x < 0 or 1 whenever x > 0. The second advantage is the sparsity, arising when the input value is smaller than 0. Sparsity is a desirable property in many machine learning applications. The drawback of ReLU is that it isn't differentiable at the origin. The softplus activation is a smoothed approximation of ReLU, which has been devised to avoid this issue.

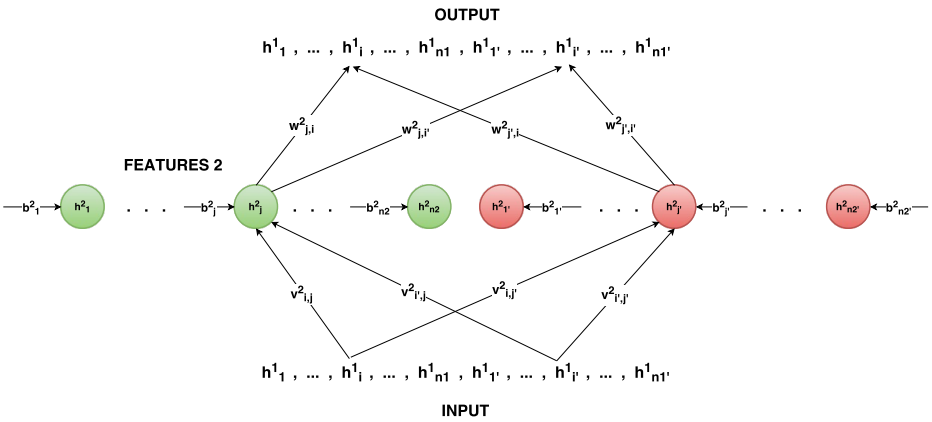


(a) First hidden layer pre-training (b) Second hidden layer pre-training (c) Third hidden layer pre-training (d) Fine-tuning of whole network

(A) Overall unsupervised pre-training and fine-tuning procedure - Image taken from [28]



(B) Pre-training of the first auto-encoder on the input data $[\mathbf{x}, \mathbf{y}]$ in order to learn the primary features $\mathbf{h}^1$.



(C) Pre-training of the second auto-encoder on the primary features $\mathbf{h}^1$ in order to learn the secondary features $\mathbf{h}^2$.

FIGURE 4.9: Unsupervised pre-training and fine-tuning procedure through the use of stacked auto-encoders.

4.3.3 Initialization/Activation results

Performing a hyper-parameter optimization on the aforementioned initialization schemes and activation functions, teaches us that the optimal performances (on the validation set) are obtained by pre-training the network with stacked auto-encoders and using ReLU activations. Table 4.3 compares the performances on the test test obtained in section 4.2 with a 3-hidden layered CmpNN regularized with the dropout technique, with the new performances obtained with a 3-hidden layered CmpNN regularized with the dropout technique, pre-trained with stacked auto-encoders and using ReLU activations. Looking at the results presented in the table, we clearly see that the new setting is outperforming the previous one. To validate the results, we apply the Welch's t-test. The 1.0% increase (from 0.298 to 0.308) of the MAP score corresponds to a p-value of 1.10^{-4}. The 6.0% increase (from 0.47 to 0.53) of the P@1/NDCG@1 score corresponds to a p-value of 2.10^{-10}. This confirms that using smarter initialization schemes and activation functions significantly improves the standard setting consisting of an uniform initialization scheme and sigmoid activation functions.

SETTING 1 - TEST SET					**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
Pair error	Avg	**0.118**	**Precision**	Avg	0.47	0.43	0.39	0.35	0.32	**0.30**	**0.29**	0.27	**0.26**	**0.24**
	Std	0.002		Std	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01
MAP	Avg	0.298	**NDCG**	Avg	0.47	0.46	0.43	0.42	0.41	0.41	0.41	**0.41**	**0.41**	0.40
	Std	0.005		Std	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01

SETTING 2 - TEST SET					**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
Pair error	Avg	0.123	**Precision**	Avg	**0.53**	**0.47**	**0.40**	**0.36**	**0.33**	**0.30**	0.28	**0.26**	0.25	**0.24**
	Std	0.004		Std	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01
MAP	Avg	**0.308**	**NDCG**	Avg	**0.53**	**0.49**	**0.46**	**0.44**	**0.43**	**0.42**	**0.42**	**0.41**	**0.41**	**0.41**
	Std	0.012		Std	0.03	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01	0.01

TABLE 4.3: Comparison of performances (MAP, P@n, NDCG@n)
Setting 1 - Results obtained at section 4.2: 3-hidden layered CmpNN, uniform initialization, sigmoid activation, dropout p = 0.25.
Setting 2 - New results: 3-hidden layered CmpNN, pre-trained with stacked auto-encoders, ReLU activation, dropout p = 0.25.

4.4 Summarization results

We now summarize the most relevant results obtained so far. We compare the performances of three settings in table 4.4. Setting 1 (of chapter 3) corresponds to the 1-hidden layered CmpNN. Setting 2 (of section 4.2) corresponds to the 3-hidden layered CmpNN regularized with the dropout technique. Setting 3 (of section 4.3) corresponds to the 3-hidden layered CmpNN regularized with the dropout technique, pre-trained with stacked auto-encoders and using ReLU activations.

The results tell us that using a well-trained deep neural network drastically improves the performances of a shallow neural network. The scores of all evaluation measures got better. For instance, the MAP value has been improved by approximately 5% compared to the initial setting, whereas the P@1/NDCG@1 value has been improved by nearly 10%. This is exactly what we were hoping for, writing this thesis. It seems that the added hidden layers are able to discover new features, boosting the overall performances significantly.

Evaluation measure	1-hidden layered CmpNN • 10 hidden neurons • Uniform initialization • Sigmoid activation • No regularization	3-hidden layered CmpNN • 24/12/6 hidden neurons in hidden layer 1/2/3 • Uniform initialization • Sigmoid activation • Dropout p = 0.25	3-hidden layered CmpNN • 24/12/6 hidden neurons in hidden layer 1/2/3 • Stacked auto-encoders • ReLU activation • Dropout p = 0.25
MAP			
MAP	0.259 ± 0.013	0.298 ± 0.005	0.308 ± 0.012
P@n			
P@1	0.44 ± 0.05	0.47 ± 0.03	0.53 ± 0.03
P@2	0.38 ± 0.03	0.43 ± 0.02	0.47 ± 0.02
P@3	0.34 ± 0.02	0.39 ± 0.02	0.40 ± 0.02
P@4	0.32 ± 0.02	0.35 ± 0.01	0.36 ± 0.01
P@5	0.30 ± 0.01	0.32 ± 0.01	0.33 ± 0.01
P@6	0.28 ± 0.01	0.30 ± 0.01	0.30 ± 0.01
P@7	0.26 ± 0.01	0.29 ± 0.01	0.28 ± 0.01
P@8	0.25 ± 0.01	0.27 ± 0.01	0.26 ± 0.01
P@9	0.24 ± 0.01	0.26 ± 0.01	0.25 ± 0.01
P@10	0.22 ± 0.01	0.24 ± 0.01	0.24 ± 0.01
NDCG@n			
NDCG@1	0.44 ± 0.05	0.47 ± 0.03	0.53 ± 0.03
NDCG@2	0.39 ± 0.03	0.46 ± 0.02	0.49 ± 0.02
NDCG@3	0.38 ± 0.03	0.43 ± 0.02	0.46 ± 0.02
NDCG@4	0.37 ± 0.02	0.42 ± 0.01	0.44 ± 0.02
NDCG@5	0.36 ± 0.02	0.41 ± 0.01	0.43 ± 0.02
NDCG@6	0.36 ± 0.02	0.41 ± 0.01	0.42 ± 0.01
NDCG@7	0.36 ± 0.02	0.41 ± 0.01	0.42 ± 0.01
NDCG@8	0.36 ± 0.02	0.41 ± 0.01	0.41 ± 0.01
NDCG@9	0.36 ± 0.02	0.41 ± 0.01	0.41 ± 0.01
NDCG@10	0.36 ± 0.02	0.40 ± 0.01	0.41 ± 0.01

TABLE 4.4: Comparison of performances (MAP, P@n, NDCG@n)
Setting 1 - 1-hidden layered CmpNN, uniform initialization, sigmoid activation, no regularization.
Setting 2 - 3-hidden layered CmpNN, uniform initialization, sigmoid activation, dropout p = 0.25.
Setting 3 - 3-hidden layered CmpNN, pre-trained with stacked auto-encoders, ReLU activation, dropout p = 0.25.

4.5 Benchmark

In this last section, we compare our results with those obtained by state-of-the-art algorithms. It is important to insist on the fact that our models are currently trained and validated on respectively 10,000 training and 5,000 validation pairs. Only in the next chapter, we will analyse approaches taking a larger part of the dataset into consideration.

The LETOR 2.0 benchmark provides the performances of several baseline algorithms for the TD2003 dataset. These baseline algorithms are RankSVM [6], RankBoost [7], FRank [16], AdaRank [3] and ListNet [11]. All these state-of-the-art algorithms have been explained in chapter 1.

As our neural network is based on the comparative neural network (CmpNN) of the SortNet algorithm [10], it is interesting to compare it with SortNet as well. However, SortNet further improves the CmpNN with an incremental learning procedure[3]. Basically, it is an iterative procedure where at each iteration the learning set is extended with misclassified pairs. Therefore, we will compare our model, which we call DeepNet, with both the CmpNN of SortNet without and with the incremental learning procedure. We summarize here briefly the different versions for clarity:

1. SortNet - CmpNN (A): The 1-hidden layered comparative neural network of [10]. This model has been trained and validated taking all training and validation document pairs into consideration. (Only the MAP score is provided in the paper for comparison.)
2. SortNet - CmpNN (B)[4]: Same network as "SortNet - CmpNN (A)", but obtained with our own implementation and trained and validated on respectively 10,000 training and 5,000 validation document pairs.
3. SortNet MAP/P@10/NDCG@10: Complete algorithm implemented in [10]. The difference with "SortNet - CmpNN (A)" is the additional use of the incremental learning procedure. MAP/P@10/NDCG@10 refers to the evaluation measure used as criterion during the iterative procedure.
4. DeepNet: Our tuned 3-hidden layered CmpNN. The network is regularized with the dropout technique, pre-trained with stacked auto-encoders, uses ReLU activation functions and is trained and validated on respectively 10,000 training and 5,000 validation pairs.

Figure 4.10 compares the MAP score obtained with SortNet - CmpNN (A), SortNet - CmpNN (B) and DeepNet. The barchart clearly shows that using well-trained deep neural networks significantly improves the results of shallow neural networks. An improvement of no less than 7% and 5% can be observed compared to the two shallow neural networks.

Tables 4.5, 4.6 and 4.7 compare the aforementioned state-of-the-art algorithms for respectively the MAP, the P@n and the NDCG@n evaluation measures. DeepNet outperforms all the considered state-of-the-art algorithms. It even outperforms SortNet for almost all evaluation measures, although it doesn't make use of the incremental learning procedure and is only considering a small fraction of the learning set. These results show once more that our deep neural network achieves remarkable results. We can already conclude that the main objective of the thesis has been met.

[3] See Appendix B for the exact explanation of the procedure.
[4] This is the model created in chapter 3.

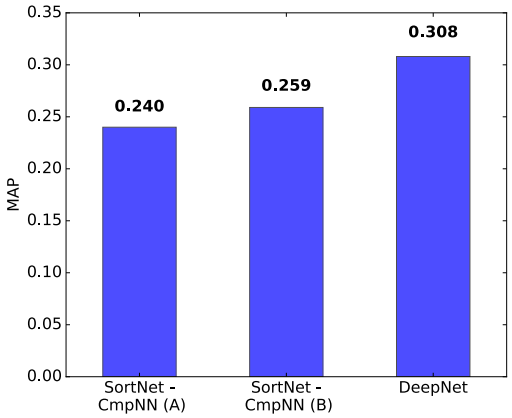


FIGURE 4.10: Benchmark TD2003 - Comparison of the MAP score between SortNet-CmpNN (A), SortNet-CmpNN (B) and DeepNet.

	MAP
RankBoost	0.212
RankSVM	0.256
FRank	0.245
ListNet	0.273
AdaRank MAP	0.137
AdaRank NDCG	0.185
SortNet MAP	0.307
SortNet P@10	0.256
SortNet NDCG@10	0.297
SortNet - CmpNN (A)	0.240
SortNet - CmpNN (B)	0.259
DeepNet	0.308

TABLE 4.5: Benchmark TD2003 - Comparison of the MAP score between the different state-of-the-art algorithms.

P@n	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
RankBoost	0.26	0.27	0.24	0.23	0.22	0.21	0.21	0.19	0.18	0.18
RankSVM	0.42	0.35	0.34	0.30	0.26	0.24	0.23	0.23	0.22	0.21
FRank	0.44	0.37	0.32	0.26	0.23	0.22	0.21	0.21	0.20	0.19
ListNet	0.46	0.42	0.36	0.31	0.29	0.28	0.26	0.24	0.23	0.22
AdaRank MAP	0.42	0.31	0.27	0.23	0.19	0.16	0.14	0.13	0.11	0.10
AdaRank NDCG	0.52	0.40	0.35	0.31	0.27	0.24	0.21	0.19	0.17	0.16
SortNet MAP	0.58	0.45	0.39	0.33	0.31	0.29	0.28	0.27	0.26	0.24
SortNet P@10	0.44	0.40	0.33	0.30	0.29	0.27	0.26	0.25	0.23	0.22
SortNet NDCG@10	0.50	0.41	0.38	0.34	0.31	0.30	0.29	0.27	0.26	0.24
SortNet - CmpNN (B)	0.44	0.38	0.34	0.32	0.30	0.28	0.26	0.25	0.24	0.22
DeepNet	0.53	0.47	0.40	0.36	0.33	0.30	0.28	0.26	0.25	0.24

TABLE 4.6: Benchmark TD2003 - Comparison of the P@n scores between the different state-of-the-art algorithms.

NDCG@n	**n=1**	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**	**n=10**
RankBoost	0.26	0.28	0.27	0.27	0.28	0.28	0.29	0.28	0.28	0.29
RankSVM	0.42	0.37	0.38	0.36	0.35	0.34	0.34	0.34	0.34	0.34
FRank	0.44	0.39	0.37	0.34	0.33	0.33	0.33	0.33	0.34	0.34
ListNet	0.46	0.43	0.41	0.39	0.38	0.39	0.38	0.37	0.38	0.37
AdaRank MAP	0.42	0.32	0.29	0.27	0.24	0.23	0.22	0.21	0.20	0.19
AdaRank NDCG	0.52	0.41	0.37	0.35	0.33	0.31	0.30	0.29	0.28	0.27
SortNet MAP	0.58	0.46	0.43	0.40	0.40	0.39	0.40	0.39	0.39	0.39
SortNet P@10	0.44	0.42	0.38	0.37	0.37	0.37	0.37	0.37	0.36	0.36
SortNet NDCG@10	0.50	0.42	0.41	0.39	0.39	0.39	0.39	0.39	0.38	0.38
SortNet - CmpNN (B)	0.44	0.39	0.38	0.37	0.36	0.36	0.36	0.36	0.36	0.36
DeepNet	0.53	0.49	0.46	0.44	0.43	0.42	0.42	0.41	0.41	0.41

TABLE 4.7: Benchmark TD2003 - Comparison of the NDCG@n scores between the different state-of-the-art algorithms.

Chapter 5

Advanced approaches

The main objective of this thesis has already been met. We have shown that by using well-trained deep neural networks, we were able to obtain excellent results in the document retrieval area. We have significantly improved the performances of the state-of-the-art algorithms, despite the fact that our models have been trained on a small fraction of the dataset.

In this final chapter, we conduct a few additional experiments. Basically, the idea is to use larger training sets in the learning process and study whether this improves the performances even further. First, we simply increase the size of the learning set (section 5.1). Then, we present two advanced learning-to-rank approaches. The first one is the multiple nested ranker approach (section 5.2) and the second one is an approach based on rank aggregation (section 5.3). After detailing these two approaches, we study their results on the TD2003 dataset (section 5.4). The experiments are conducted using the deep CmpNN of the previous chapter[1].

5.1 Extensive learning set

The models in previous chapters have been trained and validated on respectively 10,000 training and 5,000 validation samples. In this section, we simply increase the size of the learning set by 10 to 100,000 training and 50,000 validation samples. Given that the average number of possible training document pairs (consisting of documents of different relevance degree) in each fold is approximately equal to 300,000, our new training dataset now consists of 33% of the total number of possible pairs.

Table 5.1 summarizes the performance scores between our previous model trained on a small fraction of the learning set and our new model trained on a learning set ten times as big. We observe that training our model on the larger training set doesn't improve the ranking results. Globally, the performances between the two settings are similar. This is rather surprising. A possible explanation is that increasing the size of the learning set simply doesn't provide any new helpful information to the deep learning model in order to find or create new useful features.

[1] A 3-hidden layered CmpNN, regularized with the dropout technique, pre-trained with stacked auto-encoders and using ReLU activation functions.

SETTING 1 - TEST SET					n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
Pair error	Avg	**0.123**	**Precision**	Avg	0.53	**0.47**	**0.40**	**0.36**	**0.33**	0.30	0.28	0.26	0.25	**0.24**
	Std	0.004		Std	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01
MAP	Avg	**0.308**	**NDCG**	Avg	0.53	**0.49**	**0.46**	**0.44**	**0.43**	**0.42**	**0.42**	**0.41**	**0.41**	**0.41**
	Std	0.012		Std	0.03	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01	0.01

SETTING 2 - TEST SET					n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
Pair error	Avg	0.125	**Precision**	Avg	**0.54**	0.46	**0.40**	**0.36**	**0.33**	**0.31**	**0.29**	**0.27**	**0.26**	**0.24**
	Std	0.003		Std	0.04	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
MAP	Avg	0.307	**NDCG**	Avg	**0.54**	0.48	0.45	0.43	0.42	**0.42**	**0.42**	**0.41**	**0.41**	**0.41**
	Std	0.011		Std	0.04	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01	0.01

TABLE 5.1: Comparison of performances (MAP, P@n, NDCG@n) of a 3-hidden layered CmpNN regularized with the dropout technique, pre-trained with stacked auto-encoders and using ReLU activation functions.
Setting 1 - 10,000 training and 5,000 validation samples.
Setting 2 - 100,000 training and 50,000 validation samples.

5.2 Multiple nested ranker

The authors of [34] present a multiple nested ranker approach that improves the accuracy at the top ranks by iteratively re-ranking the top scoring documents. Given that this algorithm has only be applied on the RankNet algorithm, it is interesting to see whether it is also able to improve the performances of our deep CmpNN.

The learning process is done in stages, as illustrated in figure 5.1. At the first stage, a model (Net_1) is trained on the entire set of the top N_1 documents (with respect to each query). Then, at the next stage, a new model (Net_2) is built on a reduced training set. This training set is formed by only including the documents which are ranked in the N_2 top positions of the ordering of training documents (with respect to each query) produced by model Net_1. The remaining documents at the lower positions of the ordering aren't taken into consideration anymore in the learning process. This procedure is repeated for all stages.

The same idea is applied in the test phase, as illustrated in figure 5.2. The re-ranking is done using the same number of stages as during the training phase. At the first stage, the model Net_1 is used to rank the entire set of N_1 documents (with respect to each query). In the next stage, the model Net_2 is used to rank the top N_2 documents (with respect to each query) that received the highest scores according to model Net_1. The positions of the remaining documents at the lower positions of the ordering are fixed and remain unchanged in the subsequent stages. This procedure is repeated for all stages.

In our experiments, 8 iterations are performed. The number of documents per query kept at each iteration is: $N_1 = all\ documents$, $N_2 = 1000$, $N_3 = 750$, $N_4 = 500$, $N_5 = 200$, $N_6 = 100$, $N_7 = 50$ and $N_8 = 10$. We limit the number of possible training/validation document pairs at each iteration to 100,000/50,000 for computation time reasons.

The multiple nested ranker has as great advantage compared to boosting algorithms such as RankBoost by splitting the problem into smaller and easier subtasks. RankBoost create weak rankers by maintaining a weight distribution over the entire set of documents pairs, which is updated at each iteration. This is computationally very expensive as the number of document pairs is quadratic in the order of the number of documents. The multiple nested ranker algorithm doesn't face this problem, as the training set is pruned at each iteration.

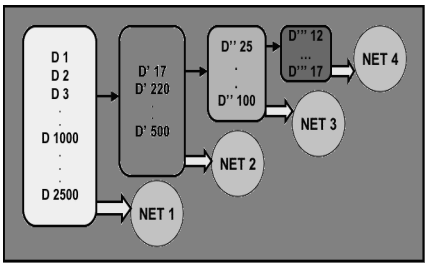


FIGURE 5.1: Illustration of the training procedure for the multiple nested ranker. NET 1 is trained on the sets of 2500 documents D per query, NET 2 is trained on the sets of the top 1000 documents D' per query, NET 3 is trained on the top 100 documents D'', NET 4 is trained on the top 10 documents D'''. - Image taken from [34].

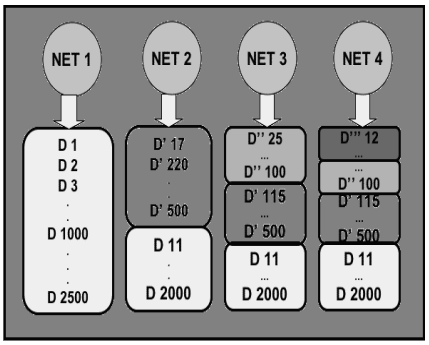


FIGURE 5.2: Illustration of the re-ranking procedure for the multiple nested ranker. NET 1 is applied to the sets of 2500 documents D per query, NET 2 is applied to the sets of the top 1000 documents D' per query, NET 3 is applied to the top 100 documents D'', NET 4 is applied to the top 10 documents D'''. - Image taken from [34].

5.3 Rank aggregation

Rank aggregation consists in combining several rank orderings on the same set of items in order to obtain a better ordering. Many rank aggregation techniques exist for a wide variety of applications. We study here one particular rank aggregation approach, which is originally used for feature selection algorithms. It is a pretty straightforward technique, renown for its simplicity.

The rank aggregation algorithm works as follows. We train a certain number of models on different learning sets. Once the learning process is done, each model m_i ranks separately the documents of the test set. The ranking of documents yielded by model m_i is denoted as R_i. In each ranking list, every document is assigned a rank. The most relevant document (the one at the top of the list) is assigned rank 1, the second most relevant document rank 2, etc. We then sum up the ranks given to each document in all orderings. The final ranking list R^* is built by ordering the documents in increasing order by the total ranks. Figure 5.3 illustrates the procedure where three different models produce three different orderings in order to form the final ranking list R^*. In our experiments, we train 15 different models and limit the number of training/validation examples to 100,000/50,000 for computation time reasons.

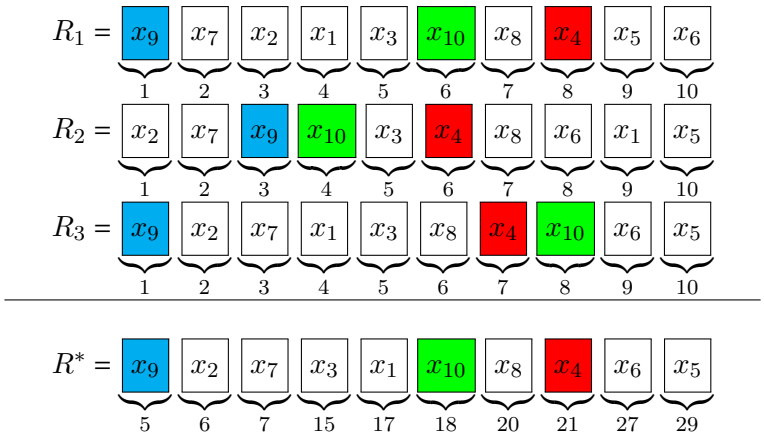


FIGURE 5.3: Rank aggregation illustration

5.4 Results

Table 5.2 summarizes the performances of the two aforementioned approaches (detailed in sections 5.2 and 5.3), comparing those with the performances of the basic setting (detailed in section 5.1). Both the multiple nested ranker approach and the rank aggregation approach improve the scores of all evaluation measures. The only exception is the P/NDCG@1 score for the multiple nested ranker approach that decreases by 2%. This is a bit surprising as the central idea of the multiple nested ranker approach is to improve the accuracy at the top ranks. A higher precision at the top rank was therefore expected.

The rank aggregation technique has slightly better results for all evaluation measures than the multiple nested ranker approach. Compared to the basic setting, the rank aggregate approach seems to improve the performances significantly. To validate the results, we apply the Welch's t-test. The 1.3% increase (from 0.307 to 0.320) of the MAP score corresponds to a p-value of 7.10^{-7}. The 2.0% increase (from 0.54 to 0.56) of the P@1/NDCG@1 score on the test test corresponds to a p-value of 0.02. These results confirm our assumption.

Overall, the results of our experiments are very satisfactory. By applying strategies that are suited for deep neural networks, we have been able to significantly improve the performances of the state-of-the-art algorithms, as it has been shown in the previous chapter. The main goal of the thesis has therefore been achieved. Furthermore, we have shown in this last chapter that the performances could be improved a bit further using more advanced approaches, such as the multiple nested ranker approach and a particular rank aggregation technique.

Evaluation measure	Extensive learning set • 24/12/6 hidden neurons in hidden layer 1/2/3 • Stacked auto-encoders • ReLU activation • Dropout p = 0.25	Multiple nested ranker • 24/12/6 hidden neurons in hidden layer 1/2/3 • Stacked auto-encoders • ReLU activation • Dropout p = 0.25	Rank aggregation • 24/12/6 hidden neurons in hidden layer 1/2/3 • Stacked auto-encoders • ReLU activation • Dropout p = 0.25
MAP			
MAP	0.307 ± 0.011	0.316 ± 0.012	0.320 ± 0.005
P@n			
P@1	0.54 ± 0.04	0.52 ± 0.03	0.56 ± 0.02
P@2	0.46 ± 0.02	0.46 ± 0.02	0.46 ± 0.01
P@3	0.40 ± 0.01	0.42 ± 0.01	0.42 ± 0.01
P@4	0.36 ± 0.01	0.37 ± 0.01	0.38 ± 0.01
P@5	0.33 ± 0.01	0.34 ± 0.01	0.34 ± 0.01
P@6	0.31 ± 0.01	0.31 ± 0.01	0.32 ± 0.01
P@7	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01
P@8	0.27 ± 0.01	0.27 ± 0.01	0.28 ± 0.01
P@9	0.26 ± 0.01	0.26 ± 0.01	0.26 ± 0.01
P@10	0.24 ± 0.01	0.25 ± 0.01	0.25 ± 0.01
NDCG@n			
NDCG@1	0.54 ± 0.04	0.52 ± 0.03	0.56 ± 0.02
NDCG@2	0.48 ± 0.02	0.48 ± 0.02	0.48 ± 0.01
NDCG@3	0.45 ± 0.02	0.46 ± 0.02	0.46 ± 0.01
NDCG@4	0.43 ± 0.02	0.44 ± 0.01	0.45 ± 0.01
NDCG@5	0.42 ± 0.02	0.43 ± 0.01	0.44 ± 0.01
NDCG@6	0.42 ± 0.01	0.42 ± 0.01	0.43 ± 0.01
NDCG@7	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01
NDCG@8	0.41 ± 0.01	0.42 ± 0.01	0.42 ± 0.01
NDCG@9	0.41 ± 0.01	0.42 ± 0.01	0.42 ± 0.01
NDCG@10	0.41 ± 0.01	0.42 ± 0.01	0.42 ± 0.01

TABLE 5.2: Comparison of performances (MAP, P@n, NDCG@n) of a 3-hidden layered CmpNN regularized with the dropout technique, pre-trained with stacked auto-encoders and using ReLU activation functions.
Setting 1 - Extensive learning set (section 5.1).
Setting 2 - Multiple nested ranker (section 5.2).
Setting 3 - Rank aggregation (section 5.3).

Conclusion

As stated in the title of the thesis, our objective was to make a contribution to the learning-to-rank problem using deep neural networks. None of the existing state-of-the-art algorithms tackling the document retrieval application use such networks. Therefore, the main goal was to study whether deep neural networks are able to improve the baseline performances.

The results of our experiments show that the objective of this thesis has been met. The first step consisted in the analysis of the so-called comparative neural network with a single hidden layer. This network adopts a particular architecture designed to implement the symmetries naturally present in a preference function. Although its desirable properties have been highlighted from a theoretic point of view, no study has demonstrated that this type of architecture outperforms a standard dense architecture with concrete figures. Our experiments show that this is the case. In a second phase, we have added multiple hidden layers to the network, while maintaining these properties. By applying strategies that are suited for deep neural networks, we have been able to significantly improve the performances of the state-of-the-art algorithms. These strategies include regularization, smarter initialization schemes, as well as more suited activation functions.

Matter of future research includes the validation of our results on other datasets. Experiments could be conducted on datasets considering more than two relevance degrees for instance. Furthermore, it may be interesting to extend this research to other areas than document retrieval.

Bibliography

[1] Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2011, pp. I–XVII, 1–285. ISBN: 978-3-642-14266-6.

[2] Hang Li. "A Short Introduction to Learning to Rank". In: *IEICE Transactions* 94-D.10 (2011), pp. 1854–1862.

[3] Jun Xu 0001 and Hang Li. "AdaRank: a boosting algorithm for information retrieval". In: *SIGIR*. Ed. by Wessel Kraaij et al. ACM, 2007, pp. 391–398. ISBN: 978-1-59593-597-7.

[4] Koby Crammer and Yoram Singer. "Pranking with Ranking". In: *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*. 2001, pp. 641–647.

[5] Ping Li 0001, Christopher J. C. Burges, and Qiang Wu. "McRank: Learning to Rank Using Multiple Classification and Gradient Boosting". In: *NIPS*. Ed. by John C. Platt et al. Curran Associates, Inc, 2007, pp. 897–904.

[6] Thorsten Joachims. "Optimizing search engines using clickthrough data". In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*. 2002, pp. 133–142.

[7] Yoav Freund et al. "An Efficient Boosting Algorithm for Combining Preferences". In: *Journal of Machine Learning Research* 4 (2003), pp. 933–969.

[8] Christopher J. C. Burges et al. "Learning to Rank using Gradient Descent". In: *ICML'05*. Bonn, Germany, Aug. 2005.

[9] Christopher J. C. Burges, Robert Ragno, and Quoc Viet Le. "Learning to Rank with Nonsmooth Cost Functions". In: *NIPS*. Ed. by Bernhard Schölkopf, John C. Platt, and Thomas Hofmann. MIT Press, 2006, pp. 193–200. ISBN: 0-262-19568-2.

[10] Leonardo Rigutini et al. "SortNet: Learning to Rank by a Neural Preference Function". In: *IEEE Transactions on Neural Networks* 22.9 (2011), pp. 1368–1380.

[11] Zhe Cao et al. "Learning to rank: from pairwise approach to listwise approach". In: *ICML*. Ed. by Zoubin Ghahramani. Vol. 227. ACM International Conference Proceeding Series. ACM, 2007, pp. 129–136. ISBN: 978-1-59593-793-3.

[12] Tao Qin et al. "LETOR: A benchmark collection for research on learning to rank for information retrieval". In: *Inf. Retr* 13.4 (2010), pp. 346–374.

[13] Olivier Chapelle and Yi Chang. "Yahoo! Learning to Rank Challenge Overview". In: *Yahoo! Learning to Rank Challenge*. Ed. by Olivier Chapelle, Yi Chang, and Tie-Yan Liu. Vol. 14. JMLR Proceedings. JMLR.org, 2011, pp. 1–24.

[14] Olivier Chapelle, Yi Chang, and Tie-Yan Liu. "Future directions in learning to rank". In: *Yahoo! Learning to Rank Challenge*. Ed. by Olivier Chapelle, Yi Chang, and Tie-Yan Liu. Vol. 14. JMLR Proceedings. JMLR.org, 2011, pp. 91–100.

[15] Y. Freund and R. Schapire. "A short introduction to boosting". In: *Journal of Japanese Society for Artificial Intelligence* 14.5 (1999), pp. 771–780.

[16] Ming-Feng Tsai et al. "FRank: a ranking method with fidelity loss". In: *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007*. 2007, pp. 383–390.

[17] Qiang Wu et al. “Adapting boosting for information retrieval measures”. In: *Inf. Retr* 13.3 (2010), pp. 254–270.

[18] Yoshua Bengio. “Learning Deep Architectures for AI”. In: *Foundations and Trends in Machine Learning* 2.1 (2009), pp. 1–127.

[19] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural Networks* 61 (2015), pp. 85–117.

[20] Christopher M Bishop. *Pattern Recognition and Machine Learning*. eng. 2006.

[21] Vitor R. Carvalho et al. *A Meta-Learning Approach for Robust Rank Learning*. en. 2009.

[22] Gerald Tesauro. “Connectionist Learning of Expert Preferences by Comparison Training”. In: 1989, pp. 99–106.

[23] Leonardo Rigutini et al. “A Neural Network Approach for Learning Object Ranking”. In: *ICANN (2)*. Ed. by Vera Kurková, Roman Neruda, and Jan Koutník. Vol. 5164. Lecture Notes in Computer Science. Springer, 2008, pp. 899–908. ISBN: 978-3-540-87558-1.

[24] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[25] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *AISTATS*. Ed. by Yee Whye Teh and D. Mike Titterington. Vol. 9. JMLR Proceedings. JMLR.org, 2010, pp. 249–256.

[26] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *ICCV*. IEEE, 2015, pp. 1026–1034. ISBN: 978-1-4673-8391-2.

[27] Dumitru Erhan et al. “Why Does Unsupervised Pre-training Help Deep Learning?” In: *Journal of Machine Learning Research* 11 (2010), pp. 625–660.

[28] Hugo Larochelle et al. “Exploring Strategies for Training Deep Neural Networks”. In: *Journal of Machine Learning Research* 10 (2009), pp. 1–40.

[29] Pierre Baldi. “Autoencoders, Unsupervised Learning, and Deep Architectures”. In: *ICML Unsupervised and Transfer Learning*. Ed. by Isabelle Guyon et al. Vol. 27. JMLR Proceedings. JMLR.org, 2012, pp. 37–50.

[30] Pascal Vincent et al. “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion”. In: *Journal of Machine Learning Research* 11 (2010), pp. 3371–3408.

[31] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Networks”. In: (2011).

[32] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. “Improving deep neural networks for LVCSR using rectified linear units and dropout”. In: *ICASSP*. IEEE, 2013, pp. 8609–8613.

[33] Hao Zheng et al. “Improving deep neural networks using softplus units”. In: *IJCNN*. IEEE, 2015, pp. 1–4. ISBN: 978-1-4799-1960-4.

[34] Irina Matveeva et al. “High accuracy retrieval with multiple nested ranker”. In: *SIGIR*. Ed. by Efthimis N. Efthimiadis et al. ACM, 2006, pp. 437–444. ISBN: 1-59593-369-7.

Appendix A

LETOR 2.0 - TD2003 Dataset

Feature ID	Description
1	BM25
2	dl of body
3	dl of anchor
4	dl of title
5	dl of URL
6	HITS authority
7	HITS hub
8	HostRank
9	idf of body
10	idf of anchor
11	idf of title
12	idf of URL
13	Sitemap based feature propagation
14	PageRank
15	LMIR.ABS of anchor
16	BM25 of anchor
17	LMIR.DIR of anchor
18	LMIR.JM of anchor
19	LMIR.ABS of extracted title
20	BM25 of extracted title
21	LMIR.DIR of extracted title
22	LMIR.JM of extracted title
23	LMIR.ABS of title
24	BM25 of title
25	LMIR.DIR of title
26	LMIR.JM of title
27	Sitemap based feature propagation
28	tf of body
29	tf of anchor
30	tf of title
31	tf of URL
32	tfidf of body
33	tfidf of anchor
34	tfidf of title
35	tfidf of URL
36	Topical PageRank
37	Topical HITS authority
38	Topical HITS hub
39	Hyperlink base score propagation: weighted in-link
40	Hyperlink base score propagation: weighted out-link
41	Hyperlink base score propagation: uniform out-link
42	Hyperlink base feature propagation: weighted in-link
43	Hyperlink base feature propagation: weighted out-link
44	Hyperlink base feature propagation: uniform out-link

TABLE A.1: LETOR 2.0 - TD2003 Dataset; Description of features

Appendix B

SortNet - Incremental learning procedure

Besides a constrained neural network architecture approximating the pairwise comparison of two documents more adequately, the authors of the SortNet algorithm [10] have devised a novel incremental learning procedure. Basically, the SortNet algorithm is an iterative procedure, where at each iteration the learning set is extended with mis-classified document pairs. This allows the algorithm to concentrate on the most informative patterns of the input domain.

The exact algorithm of SortNet is shown below. T and V are respectively the set of training and validation documents, while P_T and P_V are respectively the set of training and validation document pairs on which a comparative neural network (CmpNN) is trained and validated. At each iteration i, a new CmpNN C^i is trained and validated on the growing learning sets, P_T and P_V. Initially, P_T and P_V are empty and the weights of the network are assigned a random value. Once a CmpNN is trained, it is used to sort both the training set T and the validation set V, producing respectively the orderings R^i_T and R^i_V. While sorting the documents, mis-classified pairs are monitored. The learning set P_T is extended at each iteration with the mis-classified training pairs E^i_T, whereas P_V is extended with the mis-classified validation pairs E^i_V. Duplicates are removed in the process. The quality of each model is measured by computing the score of one of the evaluation measures (i.e. MAP, P@10 or NDCG@10) on the validation set. The final model C^* is the one with the highest quality/score. The procedure is repeated 30 times or stopped when both the training set P_T and validation set P_V remain unchanged from one iteration to the other.

SortNet - Incremental learning procedure

$T \leftarrow Set\ of\ training\ objects$
$V \leftarrow Set\ of\ validation\ objects$
$P_T^0 \leftarrow \emptyset$
$P_V^0 \leftarrow \emptyset$
$C^0 \leftarrow RandomWeightNetwork()$
for $i = 0$ to max_{iter} **do**
    if $i \geq 1$ **then**
        $C^i \leftarrow TrainAndValidate(P_T^i, P_V^i)$
    end if
    $[E_T^i, R_T^i] \leftarrow Sort(C^i, T)$
    $[E_V^i, R_V^i] \leftarrow Sort(C^i, V)$
    $score \leftarrow RankQuality(R_V^i)$
    if $score > best_{score}$ **then**
        $best_{score} \leftarrow score$
        $C^* \leftarrow C^i$
    end if
    $P_T^{i+1} \leftarrow P_T^i \cup E_T^i$
    $P_V^{i+1} \leftarrow P_V^i \cup E_V^i$
    if $P_T^{i+1} = P_T^i$ and $P_V^{i+1} = P_V^i$ **then**
        $return\ C^*$
    end if
end for
$return\ C^*$

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**