Learning Label Structures with Neural Networks for Multi-label Classification

Vom Fachbereich Informatik der Technischen Universität Darmstadt
zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte **Dissertation** von **Jinseok Nam**, M.Sc. aus Tae-baek

Tag der Einreichung: 02. Mai 2018
Tag der Verteidigung: 11. Juni 2018
Darmstadt 2019 – D 17

1. Referent: Prof. Dr. Johannes Fürnkranz
2. Referent: Prof. Dr. Krzysztof Dembczyński

TECHNISCHE
UNIVERSITÄT
DARMSTADT

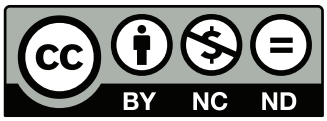
Learning Label Structures with Neural Networks for Multi-label Classification

Genehmigte **Dissertation** von **Jinseok Nam**, M.Sc. aus Tae-baek

1. Referent: Prof. Dr. Johannes Fürnkranz
2. Referent: Prof. Dr. Krzysztof Dembczyński

Tag der Einreichung: 02. Mai 2018
Tag der Verteidigung: 11. Juni 2018

Darmstadt 2019 – D17



Zusammenfassung

Multi-Label-Klassifizierung (MLC) bezeichnet die Aufgabe, eine Menge von Labels für eine gegebene Instanz vorherzusagen. Eine zentrale Herausforderung bei MLC ist die Erfassung der zugrundeliegenden Strukturen im Labelraum. Aufgrund der Komplexität des Lernens aus allen möglichen Labelkombinationen ist es bei großen MLC Datensätzen von entscheidender Bedeutung, sowohl Skalierbarkeit als auch Vorhersagequalität zu berücksichtigen. Ein weiteres Problem, das bei der Erstellung von MLC-Systemen auftritt, ist die Frage nach dem Evaluationsmaß, welches für den Vergleich der Vorhersagequalität herangezogen werden soll. Im Gegensatz zur traditionellen Multi-Klassen-Klassifizierung werden in MLC häufig mehrere Evaluationsmaße gemeinsam eingesetzt, da jedes Maß ein anderes MLC-System präferiert. Mit anderen Worten, es ist entscheidend, die Eigenschaften der verschiedenen MLC Evaluationsmaße zu verstehen, um ein System zu erstellen, das gut in Bezug auf die Maße ist, an denen wir besonders interessiert sind.

In dieser Arbeit entwickeln wir Architekturen von Neuronalen Netzwerken (NN), die Labelstrukturen in großen MLC-Problemen effizient und effektiv bezüglich eines bestimmten Evaluationsmaßes ausnutzen. Obwohl NNs, die aus paarweisen Labelbeziehungen lernen, bereits länger in der Literatur verwendet werden, schlagen wir eine vergleichsweise simple Architektur vor, die eine Verlustfunktion verwendet, die Label-Abhängigkeiten ignoriert. Wir zeigen, dass unser Ansatz besser funktioniert als komplexere neuronale Netze bezüglich des Rank-Loss-Maßes, welches explizit die Anzahl der durch das Verfahren falsch sortierten Labelpaare berücksichtigt.

Ein weiteres Evaluationsmaß, das üblicherweise beachtet wird, ist Subset 0/1-Loss. Der Classifier-Chain-Ansatz (CC) ist ein erfolgreiches, aktuelles Verfahren um dieses Maß zu optimieren. Dies geschieht dadurch, dass das ursprüngliche Problem in ein sequentielles Vorhersageproblem umgewandelt wird, sodass die Aufgabe daraufhin darin besteht, eine Sequenz von Binärwerten für die Labels vorherzusagen. Im Gegensatz zur eben genannten NN-Architektur, die Labelstrukturen ignoriert, setzen wir rekurrente neuronale Netze (RNN) ein, um Sequenzstrukturen in den Labelketten auszunutzen. Die vorgeschlagenen RNNs erweisen sich gegenüber CCs als vorteilhaft bei Problemen mit einer großen Anzahl an Labels wegen Parameter-Sharing-Effekten bei RNNs und bei Problemen mit langen Labelsequenzen.

Zusätzlich zu den NNs, die auf Labelsequenzen gelernt werden, stellen wir zwei weitere neuartige NN-basierte Methoden vor. Diese Methoden projizieren sowohl Instanzen als auch Labels auf eine Art und Weise in einen gemeinsamen niedrig-dimensionalen Raum, welche die Distanz zwischen einer Instanz und ihren relevanten Labels in diesem Raum reduziert. Während das Ziel beider Lernmethoden gleich ist, nämlich das Projizieren von Instanzen und Labels in einen gemeinsamen Raum, verwenden sie unterschiedliche Zusatzinformationen über die Labelräume: Das erste vorgeschlagene Verfahren nutzt hierarchische Strukturen aus und kann insbesondere nützlich sein, wenn solche Stukturen von Experten zur Verfügung gestellt werden. Die zweite Methode nutzt latente Labelräume aus, die von den textuellen Beschreibungen der Labels gelernt werden, sodass wir das Verfahren auf allgemeinere MLC-Probleme anwenden können, für die keine expliziten Labelstrukturen vorhanden

sind. Ungeachtet der Unterschiede ermöglichen uns beide Verfahren, Vorhersagen über Labels zu treffen, die während des Trainings nicht gesehen wurden. Außerdem zeigen wir, dass beide Verfahren in der Lage sind, durch Ausnutzung der Zusatzinformationen insgesamt eine bessere Vorhersagequalität zu erreichen.

Abstract

Multi-label classification (MLC) is the task of predicting a set of labels for a given input instance. A key challenge in MLC is how to capture underlying structures in label spaces. Due to the computational cost of learning from all possible label combinations, it is crucial to take into account scalability as well as predictive performance when we deal with large-scale MLC problems. Another problem that arises when building MLC systems is which evaluation measures need to be used for performance comparison. Unlike traditional multi-class classification, several evaluation measures are often used together in MLC because each measure prefers a different MLC system. In other words, we need to understand the properties of MLC evaluation measures and build a system which performs well in terms of those evaluation measures in which we are particularly interested.

In this thesis, we develop neural network architectures that efficiently and effectively utilize underlying label structures in large-scale MLC problems. In the literature, neural networks (NNs) that learn from pairwise relationships between labels have been used, but they do not scale well on large-scale label spaces. Thus, we propose a comparably simple NN architecture that uses a loss function which ignores label dependencies. We demonstrate that simpler NNs using cross-entropy per label works better than more complex NNs, particularly in terms of rank loss, an evaluation measure that takes into account the number of incorrectly ranked label pairs.

Another commonly considered evaluation measure is subset 0/1 loss. Classifier chains (CCs) have shown state-of-the-art performance in terms of that measure because the joint probability of labels is optimized explicitly. CCs essentially convert the problem of learning the joint probability into a sequential prediction problem. Then, the task is to predict a sequence of binary values for labels. Contrary to the aforementioned NN architecture which ignores label structures, we study recurrent neural networks (RNNs) so as to make use of sequential structures on label chains. The proposed RNNs are advantageous over CC approaches when dealing with a large number of labels due to parameter sharing effects in RNNs and their abilities to learn from long sequences. Our experimental results also confirm that their superior performance on very large label spaces.

In addition to NNs that learn from label sequences, we present two novel NN-based methods that learn a joint space of instances and labels efficiently while exploiting label structures. The proposed joint space learning methods project both instances and labels into a lower dimensional space in a way that minimizes the distance between an instance and its relevant labels in that space. While the goal of both joint space learning methods is same, they use different additional information on label spaces during training: One approach makes use of hierarchical structures of labels and can be useful when such label structures are given by human experts. The other uses latent label spaces learned from textual label descriptions so that we can apply it to more general MLC problems where no explicit label structures are available. Notwithstanding the difference between the two approaches, both approaches allow us to make predictions with respect to labels that have not been seen during training.

Acknowledgements

I would like to express my sincere appreciation to my advisor Professor Dr. Johannes Fürnkranz. He has taught me how to develop solid research questions and encouraged me to pursue interesting research directions that has helped me grow as a researcher. He gave me thought provoking advice when I needed his help and his guidance has been invaluable. I am also grateful for his time, endless support and patience to keep me motivated and make my Ph.D. study more productive.

I would also like to thank Dr. Eneldo Loza Mencïa. As an expert in the field of multi-label learning and also my lab mate, he has spent his time for discussion with me from which I was able to develop a better insight into multi-label problems and solutions. All the discussions I had with him were fruitful and motivational for me.

I would also like to thank my committee members, Professor Dr. Krzysztof Dembczyński, Professor Dr. Iryna Gurevych, Professor Dr. Kristian Kersting and Professor Dr. Arjan Kuijper for serving as my committee members and giving me insightful comments and suggestions.

The members of Knowledge Engineering group have also contributed to my personal and professional time at TU Darmstadt. The events organized by the members for fun were enjoyable and made me feel at home even when I was stuck during my journey toward scientific goals.

I gratefully acknowledge the funding sources. I was funded by the German Institute for International Educational Research under the Knowledge Discovery and Scientific Literature program and the German Research Foundation under the Adaptive Preparation of Information from Heterogeneous Sources program (AIPHES, GRK 1994/1). My scientific contributions would not be made possible without their generous financial support.

Last but not least, I would like to express my deepest gratitude to my wife YoonAh for her faithful support during my Ph.D. study since its beginning.

Contents

List of Abbreviations

1-err	one error 13
ACC	subset accuracy 12, 13, 15, 62, 63, 65, 67
ANN	artificial neural network xi, 22
AvgP	average precision 13
BP-MLL	backpropagation for multi-label learning 3, 18, 41, 43–46
BR	binary relevance 17, 18, 41, 60, 63, 65
C2AE	canonical-correlated autoencoder 21
CBM	conditional Bernoulli mixture 19
CC	classifier chain iii, 55, 56, 60, 101
CCA	canonical correlation analysis 21
CE	cross entropy 45, 46
CML	collective multi-label classifier 19
CMLF	collective multi-label with feature classifier 19
Cov	coverage 13
CPLST	conditional principal label space transformation 20, 21
CRF	conditional random field 19
CS	compressed sensing 20
DAG	directed acyclic graph 57, 61
DFS	depth-first search 57, 61
DNN	deep neural network 2, 3, 21
ebF_1	example-based F_1-measure 13, 62, 63, 65, 67
EncDec	encoder-decoder 37, 38, 58–63, 65–68
FaIE	feature-aware implicit label space encoding 21
FNN	feed-forward neural network 2, 5, 22–24, 27, 30, 31, 55
GPU	graphics processing unit 3
GRU	gated recurrent unit 36–38, 58, 60
HA	Hamming accuracy 12, 15, 62, 63, 65, 67
HMLC	hierarchical multi-label classification 20
KN	Kneser-Ney 29
LEML	low-rank risk minimization for multi-label learning 21
LM	language model 27, 29
LP	label powerset 18, 55, 56, 60, 63, 65, 101
LR	logistic regression 2, 18
LSDR	label space dimensionality reduction 21
LSTM	long short-term memory 32–38
maF_1	macro-averaged F_1 14, 62, 63, 65, 67
miF_1	micro-averaged F_1 14, 62, 63, 65, 67
MLC	multi-label classification iii, xiii, 1–7, 9–14, 17–21, 29, 41, 42, 55–61, 65, 68, 101, 102

MLP	multi-layer perceptron 22, 30
MLTC	multi-label text classification 5, 102
NLL	negative log-likelihood 62
NLM	neural language model 27, 29
NMT	neural machine translation 36, 39
NN	neural network iii, 2–6, 18, 28–30, 41, 42, 44, 45, 47, 48, 57, 58, 60, 63, 65, 101
OOV	out-of-vocabulary 36, 37, 61
PCC	probabilistic classifier chain 5, 19, 55–60, 63, 65, 68
PLST	principal label space transformation 20, 21
PLT	probabilistic label tree 22
PWE	pairwise error function 43–46
ReLU	rectified linear unit 41, 47
REML	robust extreme multi-label learning 21
RL	rank loss 13
RNN	recurrent neural network iii, xiii, 5, 30–34, 36–38, 56–62, 68, 101, 102
Seq2Seq	sequence-to-sequence 58
SGD	stochastic gradient descent 47
SLEEC	sparse local embeddings for extreme multi-label classification 21
SVD	singular value decomposition 20
SVM	support vector machine 18, 41, 42
tf-idf	term frequency-inverse document frequency 41, 48
tSNE	t-distributed stochastic neighbor embedding 78, 79, 97
WARP	weighted approximate-rank pairwise 90, 92, 94
XMLC	extreme multi-label classification 21, 102
ZSL	zero-shot learning xi, 3–5, 87, 88, 99, 100

List of Figures

List of Tables

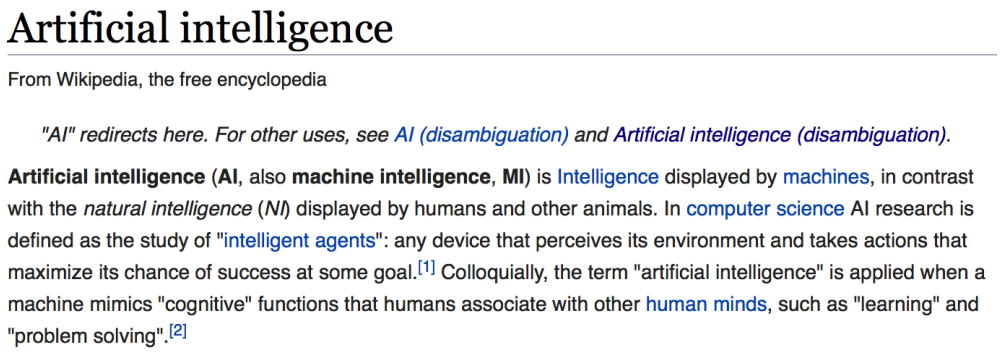
1 Introduction

Classifying instances has been a vital task in machine learning for several decades. In the straightforward setting, the type of responses is binary for classification systems, that is, the expected answer is either *yes* or *no*. A possible extension of the binary classification problem is multi-class classification, where the task is to choose the most probable out of multiple choices. Many multi-class classification problems have been studied widely and extensively across all sub-areas of artificial intelligence such as natural language processing, speech recognition, computer vision, etc. A central goal in learning classification models is to identify relationships between instances and possible responses and then to choose the best mapping function from instances into responses. Although more than two choices are available in multi-class classification, the number of correct answers is always one same as in binary classification. However, in real-world settings, classification systems often need to choose multiple correct answers out of multiple possible options. A wikipedia article as a sample document associated with multiple labels is shown in Figure 1.1. The wikipedia article explains an emerging field of study, which has been a long-standing goal of computer science, namely artificial intelligence, with a few lines of sentences. For the indexing purpose, multiple descriptors assigned to the article are chosen out of tens of thousands of descriptors. Such a small group of descriptors consists of highly related ones and each descriptor explains a certain aspect of the content being discussed.

Multi-label classification (MLC) is the problem of classifying instances into multiple correct responses. Recently, MLC methods have received a great deal of attention in machine learning because the need of predicting multiple class labels per instance arises in many real-world problems. As a group of labels is associated with an instance, it is crucial to exploit label dependencies as well as the relationship between instances and labels in MLC. In most cases, no label dependency information is available explicitly in the problem of interest. One often assumes some underlying label dependency structures, and makes use of the structures during learning. Otherwise, MLC methods work with purely statistical patterns between instances and labels.

Exploiting label dependence. A simple method that ignores the label dependence entirely can also solve MLC problems by taking advantage of traditional classification algorithms that have been studied extensively over the last few past decades, but it leads to a suboptimal solution. Another group of approaches enumerate all possible interactions of labels and exhaustively search for the best label combination for a given instance. Such naïve approaches will work on only very small scale problems. It is worth noting that exploiting label dependence is computationally demanding in general, and its complexity grows in the number of possible choices. In recent years, there has been a rapid growth of interest in large scale MLC problems that have a large number of labels as well as instances.[1] Therefore, the key to developing MLC algorithms is how to exploit label dependence efficiently.

[1] http://manikvarma.org/downloads/XC/XMLRepository.html



Artificial intelligence

From Wikipedia, the free encyclopedia

"AI" redirects here. For other uses, see AI (disambiguation) and Artificial intelligence (disambiguation).

Artificial intelligence (**AI**, also **machine intelligence**, **MI**) is Intelligence displayed by machines, in contrast with the *natural intelligence* (*NI*) displayed by humans and other animals. In computer science AI research is defined as the study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of success at some goal.[1] Colloquially, the term "artificial intelligence" is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving".[2]

(a) Wikipedia article about "Artificial Intelligence"

Categories: Artificial intelligence | Cybernetics | Formal sciences | Technology in society | Computational neuroscience | Emerging technologies | Unsolved problems in computer science | Computational fields of study

(b) Relevant labels

Figure 1.1: A real-world example of multi-label data. From Wikipedia, https://en.wikipedia.org/wiki/Artificial_intelligence, 08 Nov. 2017.

Before going into the details of existing MLC approaches, we need to discuss the importance of evaluation measures taking into consideration when building new MLC algorithms. The most widely used measure in conventional classification problems is *accuracy* which calculates how many times the predictions generated by a learned classifier are correct according to their expected answers. If some labels appear much more frequently than other labels in the training data, another evaluation measure such as F_1 measure would be more appropriate as a measure of interest. In contrast to multi-class classification, we have many more evaluation measures in MLC that show various characteristics of system outputs in different ways, which we discuss in Section 2.2.1. Thus, MLC methods need to set their goal of learning depending on the evaluation measure of interest. Once the learning objective of a classification method is determined, the next step is to choose the best algorithmic architecture for achieving that goal. A most simple, yet effective method is *logistic regression* (LR), which learns the conditional probability distribution of a label space for a given instance and constructs a linear decision function. Note that while LR has been used for conventional classification problems, it is even applicable to MLC as well without any modifications when MLC problems meet some conditions, e.g., a small number of labels. Because of its effectiveness and soundness in modeling the conditional probability, complicated MLC methods often consider LR as a base component to build a more powerful classification function. A *neural network* (NN) is a family of computational learning systems inspired by biological neural circuits in a human brain. NNs without cyclic connections between nodes, namely *feed-forward neural network* (FNN), are used for building non-linear classifiers. Thus, NNs learn more complex decision boundaries than LR models.

Recently, tremendous efforts have been devoted to improving the performance of neural networks with multiple layers of abstraction, which lead to many successful applications in a variety of areas for solving complex problems in artificial intelligence. The first successful examples of *deep neural network* (DNN) have been built by iteratively stacking multiple single layer NNs in a greedy fashion (Bengio et al., 2007; Hinton and Salakhutdinov, 2006).

More complex NNs that exploit spatial or temporal structures in data have been also very successful in learning representations of inputs from data as well as parametric classifiers (Krizhevsky et al., 2012; Sutskever et al., 2014). Massively parallel computing architectures such as *graphics processing units* (GPUs) and many large-scale datasets have proliferated this research field even further over the last years.

Despite their effectiveness in learning complex functions such as classifiers, NNs have received less attention in MLC. A well-known NN-based MLC architecture (Zhang and Zhou, 2006), *backpropagation for multi-label learning* (BP-MLL), is problematic because it is computationally demanding to make use of label dependency patterns on large-scale datasets. Modeling pairwise label dependencies in NNs explicitly makes it difficult for us to build more powerful neural architectures, which learn meaningful intermediate representations from data.

Exploiting additional information on label spaces. In a general workflow, MLC methods learn from only a set of training examples, where the association patterns of instances and labels are available. While training MLC models on the patterns, we expect that they are able to uncover underlying relationships between labels once training is done. The learned information during the training phase helps us make better predictions on unseen instances, where the label space on which MLC models are trained remains the same at prediction time. As an extreme case, one can assume we may receive *unseen instances* associated with *unseen labels* as well as seen labels. In other words, some labels in a set of candidate labels to be predicted have no training instances, so that it is crucial to exploit label dependence if we want to make predictions over label spaces including unseen labels. Figure 1.2 shows the difference between traditional classification and *zero-shot learning* (ZSL). Traditional classification tasks aim at learning mapping functions from training instances to the set of labels in the training set, and then test (unseen) instances are mapped the same set of labels observed during the training time (solid line). In contrast to the conventional setting of classification, we seek a function that maps test instances to a set of unseen labels (dotted line) for ZSL. The underlying assumption in ZSL is that the disjoint label sets, i.e., seen and unseen labels, share some information explicitly or implicitly.

When building MLC systems that have the capability to predict unseen labels, a major issue is the limited information problem of the relationships between seen and unseen labels, which we cannot access in a given training dataset. To bridge the gap between the problem of interest and the information available, one may consider external resources, e.g., knowledge bases, that human experts have organized. The central idea is that label relationships can be extracted if we can find mappings from labels to entities in external resources. This line of research has recently emerged in the machine learning community and is referred to as ZSL. In the literature, there have been several ZSL approaches that take advantage of the recent advancements in learning DNNs. Given side information that describes class labels with human curated annotations, we can exploit it to transfer information between seen and unseen labels (Lampert et al., 2009). This approach assumes that high-level semantic attributes for class labels are predefined by human experts, but it is also possible to utilize textual information on the web for building automatic ZSL systems (Frome et al., 2013).

The key property on which ZSL methods rely is the label dependence. Once learned shared information on seen labels, a ZSL classifier is able to transfer knowledge to predict unseen labels. As noted, it is also important for MLC methods to exploit the label dependence as

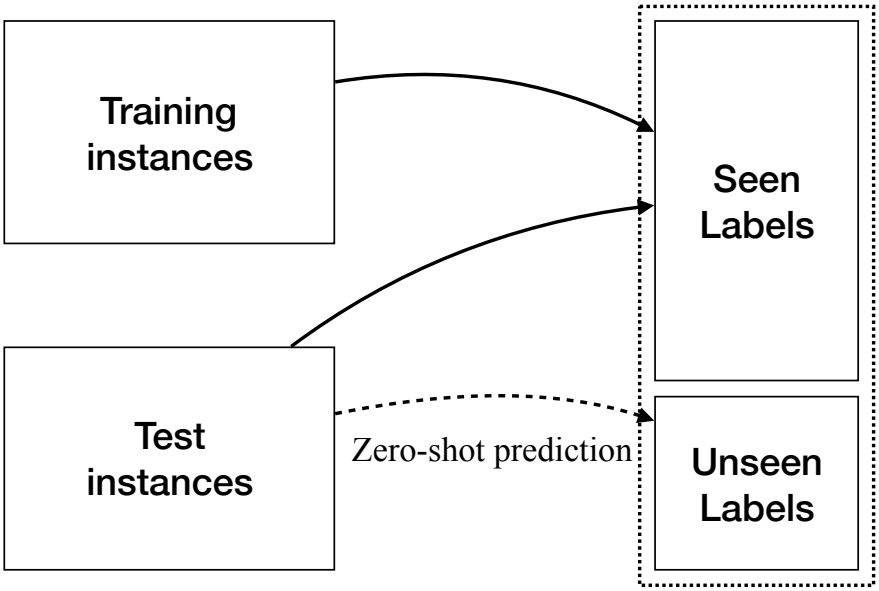


Figure 1.2: Illustration of zero-shot learning.

in ZSL. This implies that the basic ideas for ZSL can be adopted to MLC methods, and it may bring us better predictive MLC systems in practice.

A problem may arise when taking label dependencies even between seen and unseen labels on MLC datasets with many labels into account due to the cost of preparing the information by human experts. Therefore, it is highly desirable to learn shared information between labels in an automatic way as in (Socher et al., 2013) for MLC.

1.1 Summary of Contributions

Given the aforementioned challenges, the goal of this thesis is to present neural network based extensions to existing approaches for the multi-label classification problems in the following:

- New insights on neural network architectures for multi-label classification are provided based on recent theoretical analyses in terms of consistency.
- Efficient neural network models on large-scale text datasets with many labels are presented.
- We propose how to exploit the label dependence with neural networks.
- Towards predicting unseen labels, we present neural networks that make use of additional information on label spaces.

1.2 Thesis Outline

The rest of the thesis is structured in the following way. After providing fundamentals of MLC and NNs in Chapter 2, we present our contributions in the subsequent chapters. Chapter 3 proposes a simple, yet effective NN architecture that minimizes the number of

incorrect pairwise rankings between relevant and irrelevant labels. Subsequently, in Chapter 4 we propose *recurrent neural networks* (RNNs) as a replacement of *probabilistic classifier chain* (PCC) approaches that maximizes the probability of predicting label subsets correctly. We, then, demonstrate the use of label structures represented as a graph for unseen label prediction in Chapter 5. In Chapter 6 we use label descriptions as a source of additional information for ZSL. Our goal is not only to obtain better zero-shot predictions, but also to improve generalization performance to seen labels. Lastly, we conclude our contributions and discuss future work for MLC in Chapter 7.

In the following, we provide an overview of the remaining chapters.

Background. MLC is the generalized problem of classifying instances into multiple classes, where more than one label may need to be assigned to each instance. We build a general framework for MLC in a theoretical point of view, followed by reviews of prior work that have been successfully applied in the literature. Generally, MLC methods are evaluated in terms of several aspects of their performance because different group of approaches often have different objectives to be optimized. Therefore, we discuss multiple evaluation measures commonly used in MLC.

Then, we cover fundamentals of NNs, on which all methods proposed in this thesis will be based.

Efficient neural networks for large-scale multi-label classification. A key objective of MLC is to capture label dependencies so as to make better predictions for unseen instances, and many successful MLC methods rely on computationally expensive operations to achieve the goal. In particular, it is difficult to apply this type of approach on problems which have a larger number of labels because the complexity grows in terms of the number of labels. Although there have been proposed NN architectures that capture label dependencies explicitly on the output layer, we found that FNNs outperform more complex NNs on several text benchmark datasets in terms of ranking measures Nam et al. (2014). We show its theoretical background and provide empirical analysis that suggests effective NN architectures for *multi-label text classification* (MLTC) in Chapter 3.

Estimating joint probabilities of label subsets using label sequences. In contrast to MLC methods that optimize the ranking objectives, where a ranked list of labels generated by MLC systems is compared to a true ranking of labels, another type of methods focus on building a classifier that produces a set of binary predictions. One of the most successful methods in this direction is PCC, which constructs a chain of independent classifiers per label and yields the final predictions. After the success of PCC, many MLC algorithms have been proposed to address its major limitations such as the computational complexity of searching over a label space that grows exponentially in labels.

However, another drawback of PCC is often disregarded: its performance decreases as the chain length gets longer. Using the fact that the average number of labels assigned to instances is much less than the total number of labels in general, we propose a RNN architecture for solving MLC problems as an alternative of PCC (Nam et al., 2017). Our experimental results on three multi-label textual datasets demonstrate that RNNs are effective multi-label classifiers particularly when we have a large number of labels.

Learning from label hierarchies. Identifying relationship among labels from statistical patterns of them available in data is crucial to build MLC systems because the performance of predictive systems depends usually on the availability of training data and its quality. In both of the previous chapters (Chapter 3 and 4), we mainly discuss the way to make use of the label patterns only and the effectiveness of NNs. One can also consider training MLC systems in which the data is scarce or even it is completely unavailable for certain labels. For example, given a database of scientific articles, new articles could be added to the database at a certain time interval, and we want to annotate them with MLC algorithms which is trained on the database previously. In this case, there is a chance that some of new articles may deal with something new which cannot be annotated by any of existing labels used at training time.

In Chapter 5, we propose an algorithm that has the capability to rank labels including ones that do not have training information (Nam et al., 2015). We take advantage of label relationships in a graph structure given as additional information, thereby achieving better label rankings if unseen labels are taken into account at test time.

Discovering latent structures from label descriptions. Label relationships as additional information are useful for a certain type of problem as discussed in Chapter 5. However, it is not always possible to obtain the label relationships as part of the training information. One can make use of another indirect information from which label relationships can be derived implicitly, and a label description in text is a good alternative for this purpose (Nam et al., 2016). Assuming that we are given textual descriptions for unseen labels, it might be possible to predict on *unseen* documents with respect to even *unseen* labels if we can leverage the fact that similar labels often have similar word usage patterns in the descriptions. For example, assuming an organization as a label, we can easily find a textual description of the organization on the web, e.g., Wikipedia (Roth, 2017).

2 Background

In this chapter, we will provide the definition of *multi-label classification* (MLC) as an area of machine learning. Let us start with binary classification to formulate the learning problem in a statistical way and then extend it to multi-label classification.

2.1 Binary Classification and Risk Minimization

Binary classification is the task of classifying instances into two groups. In other words, we seek a function f that returns predicted outputs $\hat{y} \in \{0, 1\}$ of inputs $\boldsymbol{x}$, i.e., $f : \boldsymbol{X} \to \boldsymbol{Y}$ or $\hat{y} = f(\boldsymbol{x})$. Given a loss function $\ell : (y, \hat{y}) \to \mathbb{R}$ which measures the discrepancy between true targets y and predictions $\hat{y}$, let us define the expected loss of a function f over data samples from an underlying probability distribution $P(\boldsymbol{X}, \boldsymbol{Y})$ as *expected* or *true risk* $\mathcal{R}$:

$$\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{XY}} \left[\ell \left(\boldsymbol{Y}, f(\boldsymbol{X}) \right) \right]. \tag{2.1}$$

The quality of a mapping function can be determined by comparing the risk of the function to that of other functions, for example, a function f is better than g if $\mathcal{R}(f) < \mathcal{R}(g)$ (Vapnik, 1999). One can find the best function which achieves the smallest risk over all possible functions. Formally, let us define *minimum risk* or *Bayes risk* as follows

$$\mathcal{R}^* = \inf_{f \in \mathcal{F}_{all}} \mathcal{R}(f) \tag{2.2}$$

where $\mathcal{F}_{all}$ denotes a set of all possible measurable functions mapping inputs $\boldsymbol{X}$ to outputs $\boldsymbol{Y}$. A function that satisfies $\mathcal{R}^* = \mathcal{R}(f)$ is called as a *Bayes classifier*. Assuming that we have *perfect* knowledge on $P(\boldsymbol{X}, \boldsymbol{Y})$, the Bayes classifier for binary classification problems can be defined as follows

$$f^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if } P(\boldsymbol{Y} = 1 | \boldsymbol{X} = \boldsymbol{x}) \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

where $P(\boldsymbol{Y} = 1 | \boldsymbol{X} = \boldsymbol{x})$ is the conditional probability that an instance $\boldsymbol{x}$ is classified as positive. Please note that Eq. (2.3) can be generalized to multi-class classification. As the underlying distribution $P(\boldsymbol{X}, \boldsymbol{Y})$ is unknown in general, it is impossible to calculate the Bayes classifier. Given a function class such that $\mathcal{F} \subseteq \mathcal{F}_{all}$ and a set of N training data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_N, y_N)\}$ sampled from the underlying distribution, one can select a function $f_N \in \mathcal{F}$ and calculate its *empirical risk* given by

$$\mathcal{R}_{\text{emp}}(f_N) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f_N(\boldsymbol{x}_n)). \tag{2.4}$$

The objective of selecting $f_N \in \mathcal{F}$ given the data is to achieve *minimum empirical risk* while keeping the difference between the empirical risk and true risk, e.g., $|\mathcal{R}(f_N) - \mathcal{R}_{\text{emp}}(f_N)|$, small. Please note that the function space $\mathcal{F}$ may or may not include the target function, i.e., the Bayes classifier. If the loss is viewed as a random variable that maps predicted outputs $f_N(\boldsymbol{x})$ to real numbers, i.e., error rates with respect to true targets, the empirical risk converges to the true risk as the number of samples N goes to infinity by the law of large numbers:

$$\mathcal{R}_{\text{emp}}(f_N) \to \mathcal{R}(f_N) \quad \text{when } N \to \infty. \tag{2.5}$$

The next question is a way to measure how close the empirical risk of any (fixed) classifier f to its true risk. Note that for its simplicity we assume that f does not depend on data samples. Hoeffding's inequality provides an upper bound of the probability that the difference between the sample average of random variables and its true expectation is smaller than some arbitrary number. Without loss of generality, suppose that $Z_1 = \ell(y_1, f(\boldsymbol{x}_1)), \cdots, Z_N = \ell(y_N, f(\boldsymbol{x}_N))$ are *i.i.d.* random variables bounded in the range $[0, 1]$, which allows us to rewrite the difference between two risks as follows

$$\mathcal{R}_{\text{emp}}(f) - \mathcal{R}(f) = \frac{1}{N}\sum_{n=1}^{N} Z_n - \mathbb{E}\left[\boldsymbol{Z}\right]. \tag{2.6}$$

Then, for all $\epsilon \geq 0$, we have

$$P\left(\left|\frac{1}{N}\sum_{n=1}^{N} Z_n - \mathbb{E}\left[\boldsymbol{Z}\right]\right| \geq \epsilon\right) \leq 2\exp(-2N\epsilon^2). \tag{2.7}$$

If we denote the r.h.s. of Eq. (2.7) by δ, i.e., $\delta = 2\exp(-2N\epsilon^2)$, it tells us that with probability at least $1 - \delta$ the difference between the empirical risk $\frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$ and the true risk $\mathbb{E}_{\boldsymbol{XY}}\left[\ell(\boldsymbol{Y}, f(\boldsymbol{X}))\right]$ is within a certain constant ϵ. We can interpret δ as a *significance level* or $100 \times (1 - \delta)\%$ as confidence where its confidence interval is $[\mathbb{E}\left[Z\right] - \epsilon, \mathbb{E}\left[Z\right] + \epsilon]$. To be more precise, we can rewrite Eq. (2.7) as follows

$$\left|\frac{1}{N}\sum_{n=1}^{N} Z_n - \mathbb{E}\left[\boldsymbol{Z}\right]\right| \leq \epsilon \tag{2.8}$$

which holds with probability at least $1 - \delta$. We can interpret ϵ as *accuracy* of the empirical risk with respect to the true risk. Furthermore, we can derive an interesting relationship between the number of samples N and the two parameters of Hoeffding's inequality δ, ϵ:

$$N \geq \frac{1}{2\epsilon^2}\log\frac{2}{\delta}. \tag{2.9}$$

As expected, Eq. (2.9) shows that to increase the accuracy ϵ and significance δ, we need more training samples. Rearranging N and ϵ in Eq. (2.9) and plugging it into Eq. (2.8), we have the following relationship between the true risk and empirical risk of any classifier f:

$$\mathcal{R}(f) \leq \mathcal{R}_{\text{emp}}(f) + \sqrt{\frac{1}{2N}\log\frac{2}{\delta}}. \tag{2.10}$$

Note that this bound holds only if we use an arbitrary but fixed classifier f that does not change depending on training samples.

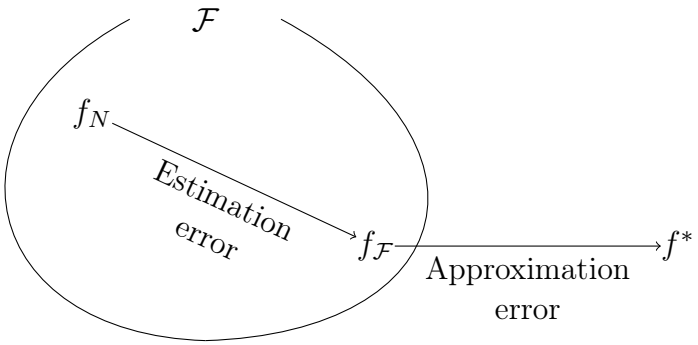


Figure 2.1: Illustration of the regret decomposition.

Consistency We now have an upper bound of the empirical risk. Given a set of training examples $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$ following a *fixed* (but unknown) probability distribution $P(\boldsymbol{X}, \boldsymbol{Y})$, let $\mathcal{A}$ be a learning algorithm that chooses a function (i.e., classifier) f_N from a set of measurable functions $\mathcal{F}$.[1] If the risk of f_N, i.e., $\mathcal{R}(f_N)$, approaches the Bayes risk R^* with *high probability* as the number of training examples gets large, $\mathcal{A}$ is called *Bayes consistent* with respect to the probability distribution $P(\boldsymbol{X}, \boldsymbol{Y})$ and a loss function ℓ used for calculating the risk:

$$P\left(\mathcal{R}(f_N) - \mathcal{R}(f^*) > \epsilon\right) \to 0 \quad \text{as} \quad N \to \infty, \forall \epsilon > 0. \tag{2.11}$$

Note that we can interpret $R(f_N)$ as a random variable that measures f_N estimated by $\mathcal{A}$ on given a finite number of sample data $\mathcal{D}$. The difference between the risk of f_N and the Bayes classifier f^* is often referred to as *excess risk* or *regret*

$$\text{regret}_{\ell,P}(f_N) = \mathcal{R}(f_N) - \mathcal{R}(f^*) \tag{2.12}$$

which can be decomposed into two terms:

$$\mathcal{R}(f_N) - \mathcal{R}(f^*) = \left(\mathcal{R}(f_N) - \inf_{f \in \mathcal{F}} \mathcal{R}(f)\right) + \left(\inf_{f \in \mathcal{F}} \mathcal{R}(f) - \mathcal{R}(f^*)\right). \tag{2.13}$$

The first term is called the *estimation error* and the second one is the *approximation error*. The estimation error measures how much the function f_N is close to the best possible function in the function space $\mathcal{F}$ while the approximation error measures the discrepancy between the optimal error in $\mathcal{F}$ and the Bayes risk. The error decomposition is illustrated in Figure 2.1. When a larger function space is considered for $\mathcal{A}$, the approximation error decreases while the estimation error may increase. On the other hand, a smaller function space $\mathcal{F}$ allows to decrease the estimation error, but this may result in higher approximation error depending on $\mathcal{F}$. There is a tradeoff between both errors, which is also known as the *bias-variance* trade-off. Usually it is hard to estimate the approximation error since it requires knowledge about the target such as $P(\boldsymbol{Y}|\boldsymbol{X})$. In machine learning we make assumptions on the optimal functions $f_{\mathcal{F}}$ so as to minimize the estimation error.

2.2 Multi-label Classification

Multi-label classification (MLC) is the task of learning a function f that maps inputs to subsets of a label set $\mathcal{L} = \{1, 2, \cdots, L\}$. Consider a set of N samples $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$,

[1] Informally, a function is measurable if its outcome is not infinitely sensitive to small changes in input.

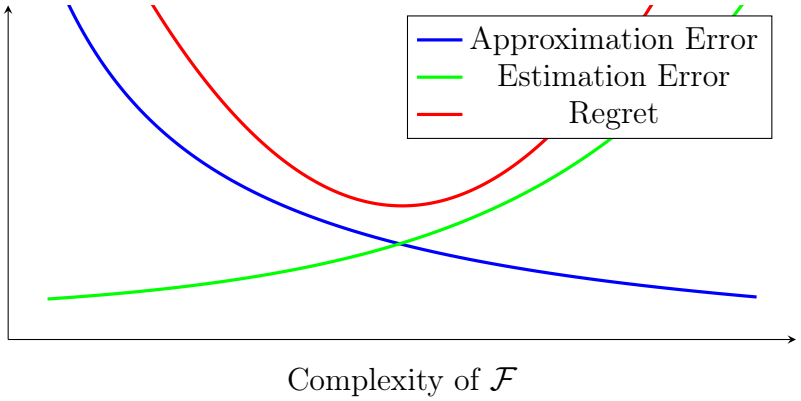


Figure 2.2: The estimation error and approximation error vary according to the complexity of the function space $\mathcal{F}$.

Table 2.1: An example of multi-label instances

	features								labels			
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y_1	y_2	y_3	y_4
Training dataset $\mathcal{D}$	−1.5	0.0	1.8	0.4	0.7	1.0	−1.4	0.4	1	0	1	0
	1.9	0.9	−0.1	0.8	0.8	1.1	−0.9	0.5	0	0	1	1
	−0.8	0.1	0.1	−2.3	0.7	−1.4	0.8	−1.3	1	1	0	1
	0.9	0.0	1.1	0.2	−0.1	−0.4	−1.2	−0.1	0	0	1	0
	−1.3	−1.3	0.2	1.3	−1.4	−1.7	−1.1	−0.3	0	1	0	0
	−0.9	−1.4	0.8	−0.2	0.1	1.3	0.0	−0.3	1	1	1	0
Test dataset $\mathcal{D}^u$	−0.5	−1.4	0.9	−0.9	−0.6	−0.8	−0.2	−2.7	?	?	?	?
	0.3	0.8	−0.3	−0.5	−0.6	0.5	0.2	−0.9	?	?	?	?
	−0.8	−1.4	1.0	−1.1	0.6	−2.0	0.9	0.8	?	?	?	?

each of which consists of an input $\boldsymbol{x}$ and its target $\boldsymbol{y}$. The pairs $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ are assumed to be *i.i.d.* random variables following an unknown distribution $P(\boldsymbol{X}, \boldsymbol{Y})$.We let $T_n = |\boldsymbol{y}_n|$ denote the size of the label set associated to $\boldsymbol{x}_n$ and $C = \frac{1}{N}\sum_{n=1}^{N} T_n$ the cardinality of $\mathcal{D}$, which is usually much smaller than L. Often, it is convenient to view the target $\boldsymbol{y}$ not as a subset of $\mathcal{L}$ but as a binary vector of size L, i.e., $\boldsymbol{y} \in \{0,1\}^L$. Let us denote a set of N_{ts} unseen instances by $\mathcal{D}^u = \{(\boldsymbol{x}^u_{\bar{n}}, \boldsymbol{y}^u_{\bar{n}})\}_{\bar{n}=1}^{N_{ts}}$ following the same unknown distribution $P(\boldsymbol{X}, \boldsymbol{Y})$. Once the function f is learned from $\mathcal{D}$, at test time, we use f to make predictions $\hat{\boldsymbol{y}}^u$ for given $\boldsymbol{x}^u$. The goal of learning f is to minimize the expected loss of instances following the underlying distribution P:

$$\min_f \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim P}\left[\ell\left(\boldsymbol{y}, f(\boldsymbol{x})\right)\right]. \tag{2.14}$$

Table 2.1 shows exemplary training and test instances. We have 6 training instances represented in 8 features and each instance is associated with 4 dimensional binary vector. The label vectors $\boldsymbol{y}$ contain multiple elements of 1. Given the training instances, we want to find the most probable label vectors $\hat{\boldsymbol{y}} \in \{0,1\}^4$ for the given input features of 3 test instances.

In a probabilistic point of view, MLC can be understood as a task of estimating the underlying distribution $P(\boldsymbol{X}, \boldsymbol{Y})$ from the training dataset $\mathcal{D}$. Since in MLC instances $\boldsymbol{x}$

are available, it would be easier to estimate the conditional distribution of labels $\boldsymbol{Y}$ given an instance $\boldsymbol{x}$, i.e., $P(\boldsymbol{Y}|\boldsymbol{X}=\boldsymbol{x})$, than the complete joint distribution $P(\boldsymbol{X},\boldsymbol{Y})$.

Assigning a subset of labels to an instance is equivalent to predicting the most probable label subset out of a label powerset $\mathcal{S}_L = \{\emptyset, \{1\}, \{2\}, \cdots, \{1, 2, \cdots, L\}\}$. To achieve the goal, ideally, we need to calculate the joint probability of labels for a given instance:

$$P\left(\boldsymbol{y}|\boldsymbol{x}\right) = P\left(y_1, y_2, \cdots, y_L|\boldsymbol{x}\right) \tag{2.15}$$

where $y \in \{0, 1\}$. Then, a label subset which yields the maximum value is chosen as an output. As the computational complexity of finding the maximum value of the joint probability of labels grows exponentially in the number of labels, i.e., $|\mathcal{S}_L| = 2^L$, one can come up with a simpler solution by estimating the marginal probability of each label based on the strong assumption that labels are independent conditional on $\boldsymbol{x}$. That is, the joint probability of labels can be factorized as

$$P(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{L} P(y_i = 1|\boldsymbol{x}) \tag{2.16}$$

where $y_i = 1$ denotes label i is associated with a given instance $\boldsymbol{x}$. Although such an assumption allows us to reduce the complexity from $\mathcal{O}(2^L)$ to $\mathcal{O}(L)$, it yields a suboptimal solution to estimating the joint probability of labels.

Let us explain why the joint probability is needed for choosing the most relevant label subset with a concrete example. Let $\boldsymbol{Y} = \{Y_1, Y_2\}$ be a label space and Y_i be binary random variables. Table 2.2 shows an example of the joint probability $P(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{x})$ and the marginal probabilities $P(Y_1 = y_1|\boldsymbol{x})$ and $P(Y_2 = y_2|\boldsymbol{x})$.

Table 2.2: An example of the joint probability of labels $\boldsymbol{Y} = \{Y_1, Y_2\}$ given an instance $\boldsymbol{x}$.

$P(\boldsymbol{Y}\|\boldsymbol{x})$		y_1		$P(Y_2\|\boldsymbol{x})$
		0	1	
y_2	0	0.0	0.4	0.4
	1	0.3	0.3	0.6
$P(Y_1\|\boldsymbol{x})$		0.3	0.7	1

Let us denote by $\boldsymbol{y}^*_{\text{joint}} = \{y^*_{\text{joint},1}, y^*_{\text{joint},2}\}$ the mode of the joint probability of labels and $\boldsymbol{y}^*_{\text{marginal}} = \{y^*_{\text{marginal},1}, y^*_{\text{marginal},2}\}$ the set of the mode of the marginal probabilities. In this example, the mode of the joint probability of labels is $\boldsymbol{y}^*_{\text{joint}} = \{1, 0\}$ where we have the maximum joint probability $P(\boldsymbol{Y} = \boldsymbol{y}^*_{\text{joint}}|\boldsymbol{x}) = 0.4$. By summing $P(Y_1, Y_2|\boldsymbol{x})$ for all possible values of Y_i, we obtain the marginal probabilities $P(Y_1|\boldsymbol{x})$ and $P(Y_2|\boldsymbol{x})$. As can be seen in Table 2.2, we have $\boldsymbol{y}^*_{\text{marginal}} = \{1, 1\}$ and it does not match the mode of the joint probability of labels:

$$\boldsymbol{y}^*_{\text{joint}} \neq \boldsymbol{y}^*_{\text{marginal}}$$

Although there are few exceptions where the relationship between $\boldsymbol{y}^*_{\text{joint}}$ and $\boldsymbol{y}^*_{\text{marginal}}$ does not hold (which shall be discussed shortly), in general, one needs to take the joint probability of labels into account when building a MLC system (Dembczyński et al., 2010). Hence, the

Table 2.3: Example-based evaluation

Target labels				Predicted labels				Subset accuracy
y_1	y_2	y_3	y_4	$\hat{y}_1$	$\hat{y}_2$	$\hat{y}_3$	$\hat{y}_4$	$\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$
1	0	1	0	0	1	1	0	0
0	0	1	1	1	0	1	0	0
1	1	0	1	1	1	0	1	1
0	0	1	0	0	0	1	0	1
0	1	0	0	1	0	0	0	0
1	1	1	0	1	1	0	1	0
								0.33

key issue of MLC is how to design classification systems estimating the joint probability of labels under some constraints related to problem domains, the scale of the problem, the availability of resources, etc.

An obvious way of evaluating MLC systems is to count how many times the systems predict a subset of labels correctly with respect to the target label subset. Depending on the goals of MLC systems, however, we are interested in different aspects of system outputs to get a better understanding of problems and results of our design choices. In the next section, we will discuss several evaluation measures that are widely used in the literature.

2.2.1 Evaluation Measures for Multi-label Classification

MLC algorithms can be evaluated with multiple measures which capture different aspects of the problem. We evaluate all methods in terms of both example-based and label-based measures.

Example-based measures are defined by comparing, for each example, the target vector $\boldsymbol{y} = \{y_1, y_2, \cdots, y_L\}$ to the prediction vector $\hat{\boldsymbol{y}} = \{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_L\}$ for a given example, and averaging the results over all examples.

Subset accuracy (ACC) checks whether a predicted label vector $\boldsymbol{y}$ matches its target *exactly* or not as follows

$$\mathrm{ACC}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = \mathbb{I}\left[\boldsymbol{y} = \hat{\boldsymbol{y}}\right] \tag{2.17}$$

where $\mathbb{I}[\cdot]$ returns 1 if its argument is true otherwise 0. It is very strict to incorrectly predicted labels in that it does not allow any deviation in the predicted label set.

Hamming accuracy (HA) computes how many labels are correctly predicted in $\hat{\boldsymbol{y}}$:

$$\mathrm{HA}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = \frac{1}{L} \sum_{j=1}^{L} \mathbb{I}\left[y_j = \hat{y}_j\right]. \tag{2.18}$$

Both, ACC and HA can be used for datasets with moderate label set sizes L. If the label cardinality of a dataset is higher, entirely correct predictions become increasingly unlikely,

and therefore ACC often approaches 0. In this case, the *example-based* F_1*-measure* (ebF_1) can be considered as a good compromise:

$$\text{eb}F_1\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = \frac{2\sum_{j=1}^{L} y_j \hat{y}_j}{\sum_{j=1}^{L} y_j + \sum_{j=1}^{L} \hat{y}_j}. \tag{2.19}$$

A concrete example of example-based evaluation using ACC is shown in Table 2.3.

MLC also can be viewed as a ranking problem. In order to evaluate the quality of a ranked list, we consider several ranking measures (Schapire and Singer, 2000). Given an instance $\boldsymbol{x}$ and associated label information $\boldsymbol{y}$, consider a multi-label learner $f_\theta(\boldsymbol{x})$ that is able to produce scores for each label. These scores, then, can be sorted in descending order. Let $r(y)$ be the rank of a label y in the sorted list of labels. The most intuitive objective for MLC is to minimize the number of misorderings between a pair of relevant label and irrelevant label. This is called the *rank loss* (RL):

$$\text{RL}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = w(\boldsymbol{y}) \sum_{y_i < y_j} \mathbb{I}\left(\hat{y}_i > \hat{y}_j\right) + \frac{1}{2}\mathbb{I}\left(\hat{y}_i = \hat{y}_j\right) \tag{2.20}$$

where $w(\boldsymbol{y})$ is a normalization factor, $\mathbb{I}\left(\cdot\right)$ is the indicator function.

One error (1-err) evaluates whether the top most ranked label with the highest score is a positive label or not:

$$\text{1-err}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = \mathbb{I}\left(r^{-1}(1) \notin \boldsymbol{y}\right) \tag{2.21}$$

where $r^{-1}(1)$ indicates the index of a label positioning on the first place in the sorted list of predicted labels $\hat{y}$.

Coverage (Cov) measures on average how far one needs to go down the ranked list of labels to achieve recall of 100%:

$$\text{Cov}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = \max_{y_i \in \boldsymbol{y}} r(y_i) - 1 \tag{2.22}$$

Average precision (AvgP) measures the average fraction of labels preceding relevant labels in the ranked list of labels:

$$AvgP\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = \frac{1}{|\boldsymbol{y}|} \sum_{y_i \in \boldsymbol{y}} \frac{\left|\{y_j \in \boldsymbol{y} | r(y_j) \leq r(y_i)\}\right|}{r(y_i)} \tag{2.23}$$

Label-based measures are based on treating each label y_j as a separate two-class prediction problem, and computing the number of *true positives* (tp_j), *false positives* (fp_j) and *false negatives* (fn_j) for this label as follows

$$\begin{aligned} tp_j &= \sum_{n=1}^{N} \mathbb{I}\left[y_{nj} = 1 \wedge \hat{y}_{nj} = 1\right] \\ fp_j &= \sum_{n=1}^{N} \mathbb{I}\left[y_{nj} = 1 \wedge \hat{y}_{nj} = 0\right] \\ fn_j &= \sum_{n=1}^{N} \mathbb{I}\left[y_{nj} = 0 \wedge \hat{y}_{nj} = 1\right] \end{aligned} \tag{2.24}$$

Table 2.4: Label-based evaluation

$\boldsymbol{y}$	Target labels						Predicted labels						Macro-averaged F_1			
													tp	fp	fn	
y_1	1	0	1	0	0	1	0	1	1	0	1	1	2	2	1	0.68
y_2	0	0	1	0	1	1	1	0	1	0	0	1	2	1	1	
y_3	1	1	0	1	0	1	1	1	0	1	0	0	3	0	0	
y_4	0	1	1	0	0	0	0	0	1	0	0	1	1	1	1	

We consider two label-based measures, *micro-averaged* F_1 (miF_1)

$$\text{mi}F_1\left(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\right) = \frac{\sum_{j=1}^{L} 2tp_j}{\sum_{j=1}^{L} 2tp_j + fp_j + fn_j},$$

and *macro-averaged* F_1 (maF_1)

$$\text{ma}F_1\left(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\right) = \frac{1}{L}\sum_{j=1}^{L} \frac{2tp_j}{2tp_j + fp_j + fn_j}$$

where $\boldsymbol{Y}$ is the $N \times L$ matrix where y_{nj} correspond to the true label j of the n-th instance $\boldsymbol{x}_n$, and $\hat{\boldsymbol{Y}}$ is the matrix of predictions $\hat{y}_{nj}$.

While miF_1 favors a system yielding good predictions on majority labels, higher maF_1 scores are usually attributed to superior performance on minority labels. Table 2.4 shows how to calculate maF_1 on the same pairs of the target and predicted label vectors as Table 2.3.

2.2.2 Risk Minimization for Multi-label Classification

We have discussed several evaluation measures commonly used in the context of MLC in the previous section. Although we want to build MLC systems that perform well across multiple measures, it is a very challenging objective to achieve the goal in general. In other words, it is likely that a system yielding good performance in terms of a certain evaluation measure may perform worse in another measure. In this section we will discuss the relationship between different models, each of which is trained to minimize different loss functions.

The goal of MLC is to find an optimal function f^* that minimizes the expected loss on an unknown sample drawn from $P(\boldsymbol{X}, \boldsymbol{Y})$:

$$\begin{aligned} f^* &= \arg\min_f \mathbb{E}_{\boldsymbol{XY}}\left[\ell(\boldsymbol{Y}, f(\boldsymbol{X}))\right] \\ &= \arg\min_f \mathbb{E}_{\boldsymbol{X}}\left[\mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}\left[\ell(\boldsymbol{Y}, f(\boldsymbol{X}))\right]\right]. \end{aligned} \tag{2.25}$$

While the expected risk minimization over $P(\boldsymbol{X}, \boldsymbol{Y})$ is intractable, for a given observation $\boldsymbol{x}$ it can be simplified to

$$\begin{aligned} f^*(\boldsymbol{x}) &= \arg\min_f \mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}\left[\ell\left(\boldsymbol{Y}, f(\boldsymbol{x})\right)\right] \\ &= \arg\min_f \int \ell\left(\boldsymbol{Y}, f(\boldsymbol{x})\right) dP(\boldsymbol{Y}|\boldsymbol{X} = \boldsymbol{x}). \end{aligned} \tag{2.26}$$

Let us consider two evaluation measures: HA and ACC. Whereas HA calculates the prediction accuracy per label independently, ACC favors only prediction results $\hat{\boldsymbol{y}}$ that match their targets $\boldsymbol{y}$ exactly. Since we want to minimize risk, let $\ell_h(\boldsymbol{y}, \hat{\boldsymbol{y}})$ and $\ell_s(\boldsymbol{y}, \hat{\boldsymbol{y}})$ be the Hamming loss and subset $0/1$ loss, respectively, as follows

$$\ell_h(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{L} \sum_{j=1}^{L} \mathbb{I}[y_j \neq \hat{y}_j] \tag{2.27}$$

$$\ell_s(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \mathbb{I}[\boldsymbol{y} \neq \hat{\boldsymbol{y}}] \tag{2.28}$$

where both $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are L-dimensional binary vectors. Using the loss functions, let us denote the optimal functions in terms of the Hamming loss and subset $0/1$ loss given by

$$f_h^*(\boldsymbol{x}) = \arg\min_f \mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}[\ell_h(\boldsymbol{Y}, f(\boldsymbol{x}))] \tag{2.29}$$

$$f_s^*(\boldsymbol{x}) = \arg\min_f \mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}[\ell_s(\boldsymbol{Y}, f(\boldsymbol{x}))] \tag{2.30}$$

where $f_h^*(\boldsymbol{x})$ and $f_h^*(\boldsymbol{x})$ denote Bayes classifiers in terms of the Hamming loss and subset $0/1$ loss, respectively.

Let us begin with calculating the Bayes classifier with respect to the subset $0/1$ loss. Since both targets $\boldsymbol{y}$ and predictions $\boldsymbol{y}$ are defined as binary (discrete) vectors, we can calculate the expected loss of predictions $\hat{\boldsymbol{y}}$ that a function f returns for given $\boldsymbol{x}$ as follows

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}[\ell_s(\boldsymbol{Y}, \hat{\boldsymbol{y}})] &= \sum_{\boldsymbol{y}} \ell_s(\boldsymbol{y}, \hat{\boldsymbol{y}}) P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) \\
&= \sum_{\boldsymbol{y}} (1 - \mathbb{I}[\boldsymbol{y} = \hat{\boldsymbol{y}}]) P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) \\
&= \sum_{\boldsymbol{y}} P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) - \sum_{\boldsymbol{y}} \mathbb{I}[\boldsymbol{y} = \hat{\boldsymbol{y}}] P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}).
\end{aligned} \tag{2.31}$$

In fact, the second term on the r.h.s. of Eq. (2.31) is calculated by a summation over 2^L label configurations. We also know that the function output $\hat{\boldsymbol{y}}$ is fixed given a function, which enables us to factorize the second term into two parts. One is the joint probability of $\boldsymbol{y}$ which is equal to $\hat{\boldsymbol{y}}$. The other is the sum of the joint probabilities of the rest of label combinations, which is equal to zero. More precisely, we can rewrite the second term on the r.h.s. of Eq. (2.31) as follows

$$\begin{aligned}
\sum_{\boldsymbol{y}} \mathbb{I}[\boldsymbol{y} = \hat{\boldsymbol{y}}] P(\boldsymbol{Y} = \hat{\boldsymbol{y}} | \boldsymbol{X} = \boldsymbol{x}) &= P(\boldsymbol{Y} = \hat{\boldsymbol{y}} | \boldsymbol{X} = \boldsymbol{x}) \\
&\quad + \underbrace{\sum_{\boldsymbol{y} \neq \hat{\boldsymbol{y}}} \mathbb{I}[\boldsymbol{y} = \hat{\boldsymbol{y}}] P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x})}_{=0 \text{ because of } \mathbb{I}[\boldsymbol{y}=\hat{\boldsymbol{y}}]=0, \forall \boldsymbol{y} \text{ in the sum}} \qquad (2.32) \\
&= P(\boldsymbol{Y} = \hat{\boldsymbol{y}} | \boldsymbol{X} = \boldsymbol{x}). \qquad (2.33)
\end{aligned}$$

Plugging Eq. (2.33) into Eq. (2.31), we have

$$\mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}[\ell_s(\boldsymbol{Y}, \hat{\boldsymbol{y}})] = 1 - P(\boldsymbol{Y} = \hat{\boldsymbol{y}} | \boldsymbol{X} = \boldsymbol{x}). \tag{2.34}$$

Thus, the expected risk minimization in terms of the subset 0/1 loss is equivalent to finding a mode of the joint probability of labels $\boldsymbol{Y}$ given instances $\boldsymbol{x}$ and the Bayes classifier is given by

$$f_s^*(\boldsymbol{x}) = \arg\max_f P(\boldsymbol{Y} = \hat{\boldsymbol{y}} | \boldsymbol{X} = \boldsymbol{x}). \tag{2.35}$$

Similarly, we can also calculate the *Bayes* classifier in terms of the Hamming loss. Let us rewrite the expected risk in terms of the Hamming loss using definition of the loss function as follows

$$\begin{aligned} \mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}} \left[\ell_h \left(\boldsymbol{Y}, \hat{\boldsymbol{y}} \right) \right] &= \sum_{\boldsymbol{y}} \ell_h \left(\boldsymbol{y}, \hat{\boldsymbol{y}} \right) P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) \\ &= \frac{1}{L} \sum_{\substack{y_1, y_2, \cdots, y_L \\ y_j \in \{0,1\}}} \left(\ell_h^1 \left(y_1, \hat{y}_1 \right) + \cdots + \ell_h^L \left(y_L, \hat{y}_L \right) \right) P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) \end{aligned} \tag{2.36}$$

where $\ell_h^j \left(y_j, \hat{j}_j \right) = \mathbb{I}\left[y_j \neq \hat{j}_j \right]$. As the hamming loss treats each label independently that allows us to assume labels y_j are conditionally independent, we can factorize the summation on the r.h.s. of Eq. (2.36) as follows

$$\begin{aligned} &\sum_{\substack{y_1, y_2, \cdots, y_L \\ y_j \in \{0,1\}}} \left(\ell_h^1 \left(y_1, \hat{y}_1 \right) + \cdots + \ell_h^L \left(y_L, \hat{y}_L \right) \right) P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) \\ &\quad = \sum_{y_1 \in \{0,1\}} \left(1 - \mathbb{I}\left[y_1 = \hat{y}_1 \right] \right) P(Y_1 = y_1 | \boldsymbol{X} = \boldsymbol{x}) \\ &\qquad + \sum_{y_2 \in \{0,1\}} \left(1 - \mathbb{I}\left[y_2 = \hat{y}_2 \right] \right) P(Y_2 = y_2 | \boldsymbol{X} = \boldsymbol{x}) \\ &\qquad + \cdots + \sum_{y_L \in \{0,1\}} \left(1 - \mathbb{I}\left[y_L = \hat{y}_L \right] \right) P(Y_L = y_L | \boldsymbol{X} = \boldsymbol{x}) \\ &\quad = L - \sum_{j=1} P(Y_j = \hat{y}_j | \boldsymbol{X} = \boldsymbol{x}). \end{aligned} \tag{2.37}$$

In turn, substitution of Eq. (2.36) with Eq. (2.37) gives

$$\mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}} \left[\ell_h \left(\boldsymbol{Y}, \hat{\boldsymbol{y}} \right) \right] = 1 - \frac{1}{L} \sum_{j=1}^{L} P(Y_j = \hat{y}_j | \boldsymbol{X} = \boldsymbol{x}). \tag{2.38}$$

The expected risk minimization in terms of the hamming loss is equivalent to finding L marginal modes of Y_j given instances $\boldsymbol{x}$ independently, and the Bayes classifier is given by

$$f_h^*(\boldsymbol{x}) = \{ \arg\max_{f_1} P(Y_1 = \hat{y}_1 | \boldsymbol{X} = \boldsymbol{x}), \cdots, \arg\max_{f_L} P(Y_L = \hat{y}_L | \boldsymbol{X} = \boldsymbol{x}) \}. \tag{2.39}$$

In contrast to that the *Bayes* classifier for the subset 0/1 loss $f_s^*(\boldsymbol{x})$ requires the joint probability distribution of labels, we can obtain the *Bayes* classifier for the hamming loss $f_h^*(\boldsymbol{x})$ only needs marginal distributions of individual labels.

Table 2.5: The *Bayes* classifiers for the Hamming loss and subset 0/1 loss are identical if (a) labels are conditionally independent or (b) the joint mode of labels are greater than or equal to 0.5.

(a)

$P(\boldsymbol{Y}\|\boldsymbol{x})$		y_1		$P(Y_2\|\boldsymbol{x})$
		0	1	
y_2	0	0.12	0.18	0.3
	1	0.28	0.42	0.7
$P(Y_1\|\boldsymbol{x})$		0.4	0.6	1

(b)

$P(\boldsymbol{Y}\|\boldsymbol{x})$		y_1		$P(Y_2\|\boldsymbol{x})$
		0	1	
y_2	0	0.6	0.1	0.7
	1	0.1	0.2	0.3
$P(Y_1\|\boldsymbol{x})$		0.7	0.3	1

As shown already in Table 2.2, the mode of the joint distribution of labels may differ from a set of marginal modes of individual labels except for two conditions where the *Bayes* classifiers for the subset 0/1 loss and hamming loss coincide. Assuming that all labels are *conditionally independent* given instances such that $P(Y_1, Y_2, \cdots, Y_L|\boldsymbol{x}) = \prod_{j=1}^{L} P(Y_j|\boldsymbol{x})$, $f_h^*(\boldsymbol{x})$ and $f_h^*(\boldsymbol{x})$ are same. When a probability assigned to a single label configuration is greater or equal to 0.5, i.e., $P(\boldsymbol{Y} = f_s^*(\boldsymbol{x})|\boldsymbol{X} = \boldsymbol{x}) \geq 0.5$, $f_h^*(\boldsymbol{x})$ and $f_h^*(\boldsymbol{x})$ also return the same function output. Table 2.5 shows two examples of such a probability distribution of labels.

We have discussed that the hamming loss and subset 0/1 loss lead us to different optimal functions. One can be obtained by ignoring label dependence completely, but the other seeks a label configuration that yields the highest probability over the entire label space. Due to the difference, it is unable to find a universal classifier that performs well across all measures. For example, Dembczyński et al. (2012b) have analyzed that the regret in terms of the subset 0/1 loss for the hamming loss is quite high and vice versa.

To be more specific, let us consider the regret of the *Bayes* classifier for the hamming loss in terms of the subset 0/1 loss. In other words, we compare the performance of f_h^* and f_s^* using the subset 0/1 loss. The upper bound of the regret is given by

$$\mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}[\ell_s(\boldsymbol{Y}, f_h^*(\boldsymbol{x}))] - \mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}[\ell_s(\boldsymbol{Y}, f_s^*(\boldsymbol{x}))] < 0.5. \tag{2.40}$$

Please note that the risk of f_h^* and f_s^* in terms of the subset 0/1 loss are identical when $P(\boldsymbol{Y} = f_s^*(\boldsymbol{x})|\boldsymbol{X} = \boldsymbol{x}) \geq 0.5$, so that the risk of f_s^* is greater than 0.5 if f_h^* differs from f_s^*. It is also worth noting that the risk of any classifier f is bounded by $[0, 1]$.

The regret of f_s^* in terms of the hamming loss has the following upper bound for $L > 3$:

$$\mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}[\ell_h(\boldsymbol{Y}, f_s^*(\boldsymbol{x}))] - \mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}[\ell_h(\boldsymbol{Y}, f_h^*(\boldsymbol{x}))] < \frac{L-2}{L+2} \tag{2.41}$$

For more details, please refer to (Dembczyński et al., 2012b).

In this section, we have shown that an optimal function for a certain evaluation measure may perform worse in terms of another measure. Hence, it is crucial to determine which evaluation measure will be optimized and to make sure the objective of a MLC system is consistent with respect to the measure of interest.

2.2.3 Multi-label Learning Algorithms

In this section, we discuss various existing approaches for MLC. The most straightforward way to tackle MLC is *binary relevance* (BR); it constructs L binary classifiers, which are

trained on the L labels independently. Thus, the prediction of the label set is composed of independent predictions for individual labels. Its predictive performance highly depends on a base learner. *Support vector machines* (SVMs), *logistic regression* (LR) and *neural networks* (NNs) are most commonly used in the literature for BR. The major drawback of BR is that the label dependence is ignored, so that we cannot make use of the interesting characteristics in MLC problems, namely that the presence of a specific label may suppress or exhibit the likelihood of other labels.

Learning from pairwise label dependencies. Instead of training L independent classifiers in which label correlations are ignored, several approaches exploit the label dependence directly in a single learning framework. A straightforward extension is to consider pairwise relationships between two labels. Elisseeff and Weston (2001) present a large-margin classifier, RankSVM, that minimizes a ranking loss by penalizing incorrectly ordered pairs of labels. This setting can be used for MLC by assuming that the ranking algorithm has to rank each relevant label before each irrelevant label. In order to make a prediction, the ranking has to be *calibrated* (Fürnkranz et al., 2008), i.e., a threshold has to be found that splits the ranking into relevant and irrelevant labels. Similarly, Zhang and Zhou (2006) introduced a framework that learns pairwise ranking errors in NNs, *backpropagation for multi-label learning* (BP-MLL). Pairwise label dependencies are also used in graphical models for maximizing subset accuracy (Ghamrawi and McCallum, 2005).

The methods based on pairwise comparisons have several limitations although they achieve competitive performance on the standard MLC benchmark datasets. Obviously, the total number of pairwise label dependencies affects the computational complexity, which grows quadratically in L. Thus, pairwise comparison based approaches do not scale on large data sets, specifically in the number of labels. Another limitation is the inability of learning higher-order dependencies which large-scale real-world datasets may contain more frequently than small benchmark datasets.

Subset accuracy maximization. To capture higher-order relationships among labels, there has been a family of approaches that attempt to classify a set of labels correctly instead of individual labels independently as in BR. The simplest approach in this direction, often referred to as subset accuracy maximization, is *label powerset* (LP). It reduces multi-label classification into multi-class classification. In other words, LP assigns a unique class index to each subset of labels. While LP is appealing because most methods well studied in multi-class classification can be used, training LP models becomes intractable for large-scale problems with an increasing number of labels. Even if the number of labels L is small enough, the problem is still prone to suffer from data scarcity because each label subset in LP will in general only have a few training instances. An effective solution to these problems is to build an ensemble of LP models learning from randomly constructed small label subset spaces (Tsoumakas et al., 2011).

An alternative approach is to learn the joint probability of labels, which is again prohibitively expensive due to 2^L label configurations. To address this problem, Dembczyński

et al. (2010) have proposed *probabilistic classifier chain* (PCC) which decomposes the joint probability into L conditional probabilities:

$$P(y_1, y_2, \cdots, y_L|\boldsymbol{x}) = \prod_{i=1}^{L} P(y_i|\boldsymbol{y}_{<i}, \boldsymbol{x}) \tag{2.42}$$

where $\boldsymbol{y}_{<i} = \{y_1, \cdots, y_{i-1}\}$ denotes a set of labels that precede a label y_i in computing conditional probabilities, and $\boldsymbol{y}_{<i} = \emptyset$ if $i = 1$. For training PCCs, L functions need to be learned independently to construct a probability tree with 2^L leaf nodes. In other words, PCCs construct a perfect binary tree of height L in which every node except the root node corresponds to a binary classifier. Therefore, obtaining the exact solution of such a probabilistic tree requires to find an optimal path from the root to a leaf node. A naïve approach for doing so requires 2^L path evaluations in the inference step, and is therefore also intractable. However, several approaches have been proposed to reduce the computational complexity (Dembczyński et al., 2012; Kumar et al., 2013; Mena et al., 2015; Read et al., 2014).

Apart from the computational issue, PCCs have also a few fundamental problems. One of them is a cascadation of errors as the length of a chain gets longer (Senge et al., 2014). During training, the classifiers f_i in the chain are trained to reduce the errors $\mathcal{E}(y_i, \hat{y}_i)$ by enriching the input vectors $\boldsymbol{x}$ with the corresponding previous true targets $\boldsymbol{y}_{<i}$ as additional features. In contrast, at test time, f_i generates samples $\hat{y}_i$ or estimates $P(\hat{y}_i|\boldsymbol{x}, \hat{\boldsymbol{y}}_{<i})$ where $\hat{\boldsymbol{y}}_{<i}$ are obtained from the preceding classifiers $f_1, \cdots, f_{i-1}$.

Another key limitation of PCCs is that the classifiers f_i are trained independently according to a fixed label order, so that each classifier is only able to make predictions with respect to a single label in a chain of labels. Regardless of the order of labels, the product of conditional probabilities in Equation (2.15) represents the joint probability of labels by the chain rule, but in practice the label order in a chain has an impact on estimating the conditional probabilities. This issue was addressed in the past by ensemble averaging (Dembczyński et al., 2010; Read et al., 2011), ensemble pruning (Li and Zhou, 2013) or by a previous analysis of the label dependencies, e.g., by Bayes nets (Sucar et al., 2014), and selecting the ordering accordingly. Similar methods learning a global order over the labels have been proposed by Kumar et al. (2013), who use kernel target alignment to order the chain according to the difficulty of the single-label problems, and by Liu and Tsang (2015), who formulate the problem of finding the globally optimal label order as a dynamic programming problem.

Subset accuracy maximization has been also addressed by graphical models which extend *conditional random field* (CRF) to MLC. Similar to CRFs, Ghamrawi and McCallum (2005) propose *collective multi-label classifier* (CML) and *collective multi-label with feature classifier* (CMLF) that define feature functions over pairs of label and input, and over pairwise label sets. The joint probability of labels given instances is then computed by the sum of all feature functions, followed by normalization. CML and CMLF have also the increasing complexity with respect to the possible number of label pairs. Li et al. (2016) introduce *conditional Bernoulli mixtures* (CBMs), which are an extension of Bernoulli mixtures used for multivariate density estimation in order to learn the joint distribution of labels in MLC. CBM learns a mixture of conditional binary label distributions, each of which represents the probability of a label given an instance. In contrast to CML and CMLF, CBM's complexity grows linearly in L because it does not rely on label pairs to estimate the joint probability.

Exploiting label structures. In *hierarchical multi-label classification* (HMLC) labels are explicitly organized in a tree usually denoting a *is-a* or *composed-of* relation. Several approaches to HMLC have been proposed which replicate this structure with a hierarchy of classifiers which predict the paths to the correct labels (Cesa-Bianchi et al., 2006; Vens et al., 2008; Zimek et al., 2010). Although there is evidence that exploiting the hierarchical structure in this way has advantages over the flat approach (Bi and Kwok, 2011; Cesa-Bianchi et al., 2006; Vens et al., 2008), some authors unexpectedly found that ignoring the hierarchical structure gives better results. For example, in (Zimek et al., 2010) it is claimed that if a strong flat classification algorithm is used the lead vanishes. Similarly, in (Vens et al., 2008) it was found that learning a single decision tree which predicts probability distributions at the leaves outperforms a hierarchy of decision trees. One of the reasons may be that hierarchical relations in the output space are often not in accordance with the input space, as claimed by Fürnkranz and Sima (2010) and Zimek et al. (2010).

Lower dimensional label spaces. One of the most challenging problems in MLC is how to build systems that have the capability to handle a large number of unique labels efficiently. Given that the number of relevant labels per instance on average is much smaller than L, possibly, we need only a few key underlying factors that explain the original label combinations. Dimensionality reduction is a way of reducing the number of variables in an original problem space by ignoring uninformative variance in data. The key objective of dimensionality reduction is to find projections under the assumption that data can be represented by the weighted sum of the projections. As dimensionality reduction enables us to extract key lower dimensional factors from data, prediction models, which project the high dimensional data into the lower dimensional space, often achieve better predictive performance.

In MLC, several approaches have been proposed to tackle the high-dimensional label space problems. Tai and Lin (2012) propose a simple approach, called *principal label space transformation* (PLST), that projects the original label vectors onto the lower dimensional space by using a small number of principal components, say K, obtained by *singular value decomposition* (SVD). The projections are K-dimensional real-valued vectors whereas the original label vectors are L-dimensional binary vectors $\{0,1\}^L$. It solves, in turn, a multidimensional regression problem to learn the relationship between inputs and the projected label vectors. At test time, PLST predicts K responses for a given input, then converts the estimated label projection to the original label space. *Compressed sensing* (CS) by Hsu et al. (2009) aims to reduce the dimensionality of label spaces, but takes a different approach from PLST. CS projects the original label vectors by a random projection matrix unlike PLST, which uses a projection matrix obtained by SVD. CS, then, recovers the original label vectors from the projections which minimize prediction errors while taking the sparsity on the original label space into account.

Both CS and PLST consist of two disjoint components; one transforms the label vectors into a lower dimensional space and the other learns multiple regressors to predict the label projections from inputs, where the two components yield the encoding error and the prediction error, respectively. Another limitation is that only label relationships are considered when they project labels into the lower dimensional space.

Kapoor et al. (2012) propose an extension of CS in a Bayesian framework which allows us to minimize the label compression error and prediction error jointly. Furthermore, it enables to handle missing labels. *Conditional principal label space transformation* (CPLST) by Chen

and Lin (2012) exploits relationships between features and labels as well as between labels in computing label subspaces unlike PLST. Chen and Lin (2012) show also CPLST with kernel regression that significantly outperforms CPLST with linear regression. *Label space dimensionality reduction* (LSDR) approaches usually define a label transformation matrix explicitly, but Lin et al. (2014) propose a LSDR method, called *feature-aware implicit label space encoding* (FaIE), which enables to obtain lower dimensional label representations even without such a transformation matrix.

LSDR methods only project high-dimensional label spaces into lower dimensional subspaces. Although CPLST and FaIE take inputs into consideration when computing lower dimensional label representations, no mapping functions from inputs to such a lower dimensional space are learned.

To leverage input-label relationships more explicitly, one may consider *canonical correlation analysis* (CCA) (Hotelling, 1936) that finds a pair of transformation matrices $\{W_1, W_2\}$ such that input projections by W_1 are maximally correlated with label projections by W_2. In other words, both inputs and label vectors can be projected onto a shared subspace, where an input projection and its corresponding label projection have maximum correlation with each other. Several CCA extensions for MLC including sparse CCA were proposed by Sun et al. (2011), where it is shown that CCA can be formulated as a least square problem. Zhang and Schneider (2011) also use CCA to compute a transformation matrix of label vectors in an error-correcting output code framework for MLC. The directions resulted from vanilla CCA are, in principle, linear transformations which cannot capture complex correlations, i.e., nonlinear relationships, between inputs and labels. *Canonical-correlated autoencoder* (C2AE) (Yeh et al., 2017) extends CCA by using *deep neural networks* (DNNs) as nonlinear functions to project both inputs and labels into a subspace jointly.

Extreme classification. In the literature, some multi-label datasets have more than a million labels. *Extreme multi-label classification* (XMLC) is the problem of learning a multi-label classifier on datasets with extremely many labels. XMLC datasets may have labeling noise or missing labels because of the large number of possible labels. LSDR approaches, in principle, are employed to address XMLC problems, but the aforementioned methods cannot deal with missing labels. A general framework considering missing labels, referred to as *low-rank risk minimization for multi-label learning* (LEML), was proposed by Yu et al. (2014) based on the low-rank assumption on a linear transformation from inputs to labels.

XMLC datasets often have a few number of frequent labels while the majority of labels have a small number of training instances, called infrequent or tail labels. The low-rank approximation violates on such a label matrix, so that tail labels cannot be well approximated. In (Xu et al., 2016), *robust extreme multi-label learning* (REML) focuses on modeling tail labels as well as learning the low-rank matrix, where tail labels are treated as outliers and a label sparsity constraint is used. REML works better than LEML at predicting rare labels. Another approach to the tail label problem is to exploit neighborhood information on a label space. Bhatia et al. (2015) propose a method for learning local label embeddings, referred to as *sparse local embeddings for extreme multi-label classification* (SLEEC), by preserving pairwise distances between labels in a local neighborhood.

For faster learning and inference, one may consider tree-based approaches which allow to build XMLC models with the logarithmic complexity in terms of the number of labels and/or instances. A decision tree based approach, called FastXML (Prabhu and Varma,

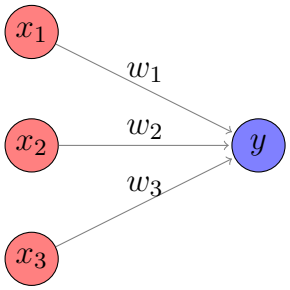


Figure 2.3: Directed graph representing a simple artificial neural network

2014), constructs an ensemble of decision trees, where internal nodes are associated with linear classifiers and trained to maximize top-k ranking of labels. Another tree based approach (Jasinska et al., 2016) makes use *probabilistic label tree* (PLT). Unlike FastXML that learns tree structures during training, PLTs create a fixed full binary tree and all nodes in the tree are associated with a linear classifier which decides whether instances are passed down toward the leaves or not. Leaf nodes in PLT correspond to labels.

2.3 Fundamentals of Neural Networks

An *artificial neural network* (ANN) is a computational model that can be represented by a directed graph in which two nodes are connected by a directed edge. In such a graph, all edges have weights that represent the strength of the connections between nodes. As an example, let us consider a directed graph with 4 nodes where three nodes $\{x_1, x_2, x_3\}$ have directed edges to a node y as shown in Fig. 2.3. This can be interpreted as the information that $\{x_1, x_2, x_3\}$ have flows to y with the corresponding weights $\{w_1, w_2, w_3\}$. The information y has can be calculated by the weighted sum of all incoming information from $\{x_1, x_2, x_3\}$, followed by some activation function f, e.g., sigmoid function $f(x) = \frac{1}{1+\exp(-x)}$, denoted by $y = f(w_1x_1 + w_2x_2 + w_3x_3)$. The output of f is often referred to as *activation*. ANNs have multiple groups of nodes, called *layers*, and inter-connections between layers while no intra-connections between nodes in the same layer.

2.3.1 Feed-forward Neural Networks

One of the simplest ANNs is a *feed-forward neural network* (FNN), or called *multi-layer perceptron* (MLP), which has no cyclic connections. For solving a classification problem with three *mutually exclusive* classes, let us consider a FNN which consists of one input, hidden and output layer with two sets of edges, called *weights* $\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$, illustrated in Figure 2.4. The goal of the problem is to classify instances into the most probable class correctly. The input layer and hidden layer are connected with $\mathbf{W}^{(1)} \in \mathbb{R}^{5\times4}$. Similarly, the other set of weights $\mathbf{W}^{(2)} \in \mathbb{R}^{3\times5}$ connects the hidden layer to the output layer. Please note that we initialize the weights with arbitrary values. The input layer receives 4-dimensional vectors such that $\mathbf{x} = [x_1, x_2, x_3, x_4]^T$. Subsequently, the activations in the hidden layer

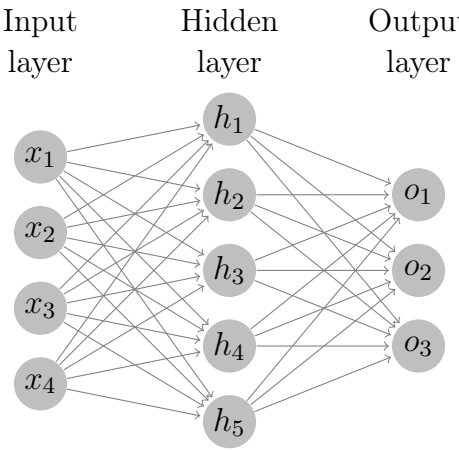


Figure 2.4: A feed-forward neural network with 4 input dimensions, 5 hidden units and 3 outputs.

are calculated by the weighted sum of inputs $\mathbf{x}$, followed by the sigmoid function to obtain non-linear outputs in terms of inputs:

$$z_j = \sum_{d=1}^{D} w_{jd}^{(1)} x_d \tag{2.43}$$

$$h_j = f\left(z_j\right) \tag{2.44}$$

where $w_{jk}^{(1)}$ is an element of the weight matrix $\mathbf{W}^{(1)}$ at row j and column k, and f denotes the sigmoid function. Please note that we ignore *bias* terms throughout this thesis for notational convenience. Given the hidden activations $\mathbf{h}$, the activations in the output layer can be computed in a similar way as follows

$$o_i = \sum_{j=1}^{H} w_{ij}^{(2)} h_j. \tag{2.45}$$

The probability of each class i being predicted given instances is, then, calculated by the *softmax* function as follows

$$p_i = P(y_i = 1|\mathbf{x}) = \frac{\exp(o_i)}{\sum_k \exp(o_k)} \tag{2.46}$$

which satisfies $0 \leq p_i \leq 1$ and $\sum_i p_i = 1$. In turn, the most probable class label $\hat{y}$ is chosen by taking the index of the class that has the highest probability among all possible classes:

$$\hat{y} = \arg\max_i p_i. \tag{2.47}$$

The performance of the network can be evaluated by, for instance, counting how many times the network predictions $\hat{y}$ match correctly the actual targets y on a dataset. Since we initialized $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ with arbitrary values, the predictions $\hat{y}$ will be rather random initially.

The goal of learning the FNN in Fig. 2.4 is to find the optimal set of weights denoted by $\mathbf{W}^{(1)*}$ and $\mathbf{W}^{(2)*}$ that allow to make correct predictions. In other words, our goal is to

minimize misclassification rate by tuning the model parameters, i.e., $\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$, given the training dataset $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ where $\mathbf{x}_n$ is the D-dimensional vector and $\mathbf{y}_n$ is the L-dimensional *one-hot* vector. Let us define the objective function of the FNN in Figure 2.4 at a given training data point $(\mathbf{x}, \mathbf{y})$ by using the cross entropy loss as follows

$$\mathcal{L}\left(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}; \mathbf{x}, \mathbf{y}\right) = \sum_{i=1}^{L} -y_i \log(p_i). \tag{2.48}$$

To find a local minimum of the objective function in Eq. (2.48), one can use *gradient descent* which is a first-order optimization method that uses the first derivatives of a function. Note that $\mathcal{L}\left(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}; \mathbf{x}, \mathbf{y}\right)$ is a function of two variables $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}$, and that the output probability $\mathbf{p} = [p_1, p_2, \cdots, p_L]^T$ results from the composition of multiple functions of $\mathbf{x}$ given the weights $\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$. For the sake of simplicity, we denote the loss function by $\mathcal{L}$ in the following derivations. The update rule of gradient descent for the parameters is given by

$$w_{ij}^{(2)} \leftarrow w_{ij}^{(2)} - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}^{(2)}} \tag{2.49}$$

$$w_{jd}^{(1)} \leftarrow w_{jd}^{(1)} - \eta \frac{\partial \mathcal{L}}{\partial w_{jd}^{(1)}} \tag{2.50}$$

where $\eta \in \mathbb{R}$ denotes a step size in gradient descent, and $w_{jd}^{(1)}$ and $w_{ij}^{(2)}$ are elements of $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$, respectively. By applying the chain rule to Eq. (2.48), the partial derivative of the loss function with respect to $w_{ij}^{(2)}$ is given by

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(2)}} = \frac{\partial \mathcal{L}}{\partial o_i} \frac{\partial o_i}{\partial w_{ij}^{(2)}} \tag{2.51}$$

Let us calculate the second term of the right hand side in Eq. (2.51). The gradient of the network's output at node i in the output layer with respect to output weights $w_{ij}^{(2)}$ can be calculated as

$$\begin{aligned} \frac{\partial o_i}{\partial w_{ij}^{(2)}} &= \frac{\partial}{\partial w_{ij}^{(2)}} \sum_k w_{ik}^{(2)} h_k \\ &= h_j. \end{aligned} \tag{2.52}$$

Given Eqs. (2.46) and (2.48), the next term $\partial \mathcal{L} / \partial o_i$ can be represented as follows

$$\frac{\partial \mathcal{L}}{\partial o_i} = \sum_{k=1}^{L} \frac{\partial \mathcal{L}}{\partial p_k} \frac{\partial p_k}{\partial o_i}. \tag{2.53}$$

As the prediction probabilities p_i are obtained by the *softmax* function in Eq. (2.46) which requires all network outputs o_i, we have the right hand side of Eq. (2.53) as follows

$$\frac{\partial \mathcal{L}}{\partial o_i} = \frac{\partial \mathcal{L}}{\partial p_i} \frac{\partial p_i}{\partial o_i} + \sum_{k \neq i} \frac{\partial \mathcal{L}}{\partial p_k} \frac{\partial p_k}{\partial o_i}. \tag{2.54}$$

Let us take the derivative of the loss function with respect to its inputs, i.e., the prediction probabilities p_i:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial p_i} &= \frac{\partial}{\partial p_i} \sum_{k=1}^{L} -y_k \log(p_k) \\ &= -\frac{y_i}{p_i}.\end{aligned} \tag{2.55}$$

The derivative $\partial p_k / \partial o_i$ of the prediction probability p_k with respect to the network output o_i can be calculated as follows

$$\frac{\partial p_k}{\partial o_i} = \begin{cases} p_i(1 - p_i) & \text{if } k = i \\ -p_k p_i & \text{if } k \neq i \end{cases} \tag{2.56}$$

Replacing Eq. (2.54) with Eq. (2.55) and Eq. (2.56), we have

$$\frac{\partial \mathcal{L}}{\partial o_i} = -\frac{y_i}{p_i} p_i(1 - p_i) + \sum_{k \neq i} -\frac{y_k}{p_k}(-p_k p_i) \tag{2.57}$$

$$= -y_i(1 - p_i) + \sum_{k \neq i} y_k p_i \tag{2.58}$$

$$= -y_i + p_i y_i + p_i \sum_{k \neq i} y_k \tag{2.59}$$

$$= p_i - y_i \tag{2.60}$$

By substituting the right hand side of Eq. (2.51) with Eq. (2.60) and Eq. (2.52), we then have

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(2)}} = \frac{\partial \mathcal{L}}{\partial o_i} \frac{\partial o_i}{\partial w_{ij}^{(2)}} = (p_i - y_i) h_j \tag{2.61}$$

The resulting partial derivative $\partial \mathcal{L} / \partial w_{ij}^{(2)}$ in Eq. (2.61) indicates that weights $w_{ij}^{(2)}$ will be updated in a way to reduce the errors $p_i - y_i$ proportional to hidden activations h_j.

Similarly, the partial derivative of the loss function with respect to $w_{jd}^{(1)}$ is given by

$$\frac{\partial \mathcal{L}}{\partial w_{jd}^{(1)}} = \frac{\partial \mathcal{L}}{\partial h_j} \frac{\partial h_j}{\partial z_j} \frac{\partial z_j}{\partial w_{jd}^{(1)}} \tag{2.62}$$

Let us calculate the first term $\partial \mathcal{L} / \partial h_j$ of the right hand side in Eq. (2.62). Note that the activation of unit j in the hidden layer, namely h_j, contributes to all units in the output layer weighted by $w_{\cdot j}^{(2)}$. Thus, the partial derivative of $\mathcal{L}$ with respect to the hidden unit activation h_j can be written as

$$\frac{\partial \mathcal{L}}{\partial h_j} = \sum_{i=1}^{L} \frac{\partial \mathcal{L}}{\partial o_i} \frac{\partial o_i}{\partial h_j}. \tag{2.63}$$

We have already obtained $\partial\mathcal{L}/\partial o_i$ when calculating $\partial\mathcal{L}/\partial w_{jd}^{(1)}$, so the right term in the summation needs to be calculated.

$$\frac{\partial o_i}{\partial h_j} = \frac{\partial}{\partial h_j}\sum_{l}^{H} w_{il}^{(2)} h_l \tag{2.64}$$

$$= w_{ij}^{(2)} \tag{2.65}$$

Plugging Eq. (2.60) and Eq. (2.65) into Eq. (2.63) results in the following partial derivative of $\mathcal{L}$ with respect to hidden activations h_j:

$$\frac{\partial\mathcal{L}}{\partial h_j} = \sum_{i=1}^{L}(p_i - y_i)w_{ij}^{(2)}. \tag{2.66}$$

The second term of algorithm 1 is the partial derivative of hidden activations h_j with respect to pre-activations z_j. As we use the sigmoid function to apply non-linearity to z_j, its derivative can be calculated simply as follows

$$\frac{\partial h_j}{\partial z_j} = \frac{\partial f(z_j)}{\partial z_j} = \frac{\partial}{\partial z_j}\frac{1}{1+e^{-z_j}} = h_j(1-h_j). \tag{2.67}$$

Then, the derivative of z_j with respect to weights $w_{jd}^{(1)}$ can be easily obtained by

$$\frac{\partial z_j}{\partial w_{jd}^{(1)}} = x_d. \tag{2.68}$$

Now, replacing Eq. (2.62) with Eqs. (2.66) to (2.68) gives the following $\partial\mathcal{L}/\partial w_{jd}^{(1)}$:

$$\frac{\partial\mathcal{L}}{\partial w_{jd}^{(1)}} = \left(\sum_{i=1}^{L}(p_i - y_i)w_{ij}^{(2)}\right)h_j(1-h_j)x_d. \tag{2.69}$$

In summary, the partial derivatives of the loss function with respect to $w_{jd}^{(1)}$ and $w_{ij}^{(2)}$ are:

$$\frac{\partial}{\partial w_{jd}^{(1)}}\mathcal{L}\left(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}; \mathbf{x}, \mathbf{y}\right) = \left(\sum_{i=1}^{L}(p_i - y_i)w_{ij}^{(2)}\right)h_j(1-h_j)x_d \tag{2.70}$$

$$\frac{\partial}{\partial w_{ij}^{(2)}}\mathcal{L}\left(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}; \mathbf{x}, \mathbf{y}\right) = (p_i - y_i)h_j. \tag{2.71}$$

All of the aforementioned operations in the forward and backward passes can be written in the form of matrix-matrix multiplication. For the sake of notational convenience, we define *errors* $\boldsymbol{\delta}_o \in \mathbb{R}^{L\times 1}$ at the output layer given by

$$\boldsymbol{\delta}_o = \mathbf{p} - \mathbf{y} \tag{2.72}$$

where each element $\delta_l \in \boldsymbol{\delta}_o$ corresponds to the difference between a prediction probability and its actual target value for each output unit. We, then, can re-write Eq. (2.71) in a vector form as follows

$$\nabla_{\mathbf{W}^{(2)}} \mathcal{L}\left(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}; \mathbf{x}, \mathbf{y}\right) = \boldsymbol{\delta}_o \mathbf{h}^T \tag{2.73}$$

where $\nabla_{\mathbf{W}^{(2)}}$ denotes the gradient operator with respect to $\mathbf{W}^{(2)}$. In a similar fashion, we have an updated error vector $\boldsymbol{\delta}_h \in \mathbb{R}^{H \times 1}$ for the hidden layer:

$$\boldsymbol{\delta}_h = \left(\mathbf{W}^{(2)T} \boldsymbol{\delta}_o\right) \odot f'(\mathbf{z}) \tag{2.74}$$

where $\odot$ denotes element-wise multiplication. Once we have the error term, we can obtain the gradient of $\mathcal{L}$ with respect to $\mathbf{W}^{(1)}$ using the same rule as Eq. (2.73) given by

$$\nabla_{\mathbf{W}^{(1)}} \mathcal{L}\left(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}; \mathbf{x}, \mathbf{y}\right) = \boldsymbol{\delta}_h \mathbf{x}^T. \tag{2.75}$$

We will use the matrix form of gradient computation in later parts of this chapter.

2.3.2 Neural Language Modeling

FNNs are often used for classification because they have the capability to learn the probability distribution over labels given instances. In natural language processing and speech recognition, A *language model* (LM) estimates a probability distribution over a sequence of words:

$$P(w_1, w_2, w_3, \cdots, w_T) \tag{2.76}$$

where w_i denotes a word at position i in the sequence of length T. The joint probability of words can be approximated by n-gram language models as follows

$$P(w_1, w_2, w_3, \cdots, w_T) \approx \prod_{i=1}^{T} P(w_i \mid w_{i-(n-1)}, w_{i-(n-2)}, \cdots, w_{i-1}) \tag{2.77}$$

where n is the length of truncated word sequences. Traditionally, each conditional probability in Eq. (2.77) has been often modeled by word counts. To overcome the data sparsity problem in count-based LMs, Bengio et al. (2003) have introduced a neural network based language modeling approach, also known as *neural language model* (NLM), which learns continuous vectors representing words. A conditional probability of predicting a word w_i given its context $\boldsymbol{w}_i^c$, namely $\boldsymbol{w}_i^c = \{w_{i-(n-1)}, w_{i-(n-2)}, \cdots, w_{i-1}\}$, can be represented by a FNN that takes vector representations for context words as inputs and predicts the next word in a word vocabulary $\mathcal{V}$. To be more specific, let us define a matrix of word vectors by $\mathbf{U} = \left[\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_{|\mathcal{V}|}\right] \in \mathbb{R}^{D \times |\mathcal{V}|}$ where D is the dimensionality of word vectors. Such word vectors are also referred to as *word embeddings* in the literature so that the terms will be used interchangeably in this thesis. Given the context $\boldsymbol{w}_i^c$ and the word embeddings $\mathbf{U}$, the following input vector is fed into a FNN:

$$\mathbf{x} = \left[\mathbf{u}_{w_{i-(n-1)}}; \mathbf{u}_{w_{i-(n-2)}}; \cdots; \mathbf{u}_{w_{i-1}}\right] \tag{2.78}$$

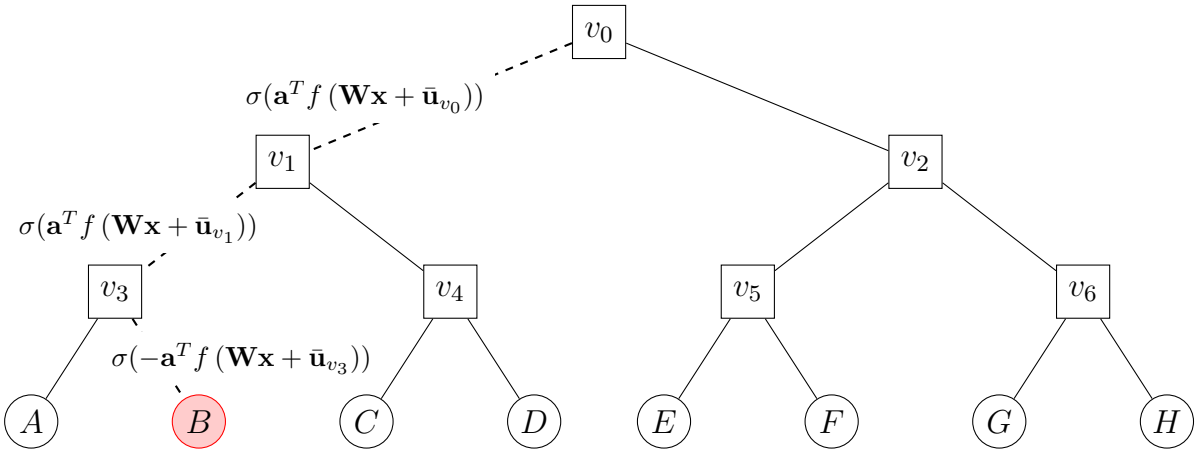


Figure 2.5: Given a target word $w_i = B$ at position i in a sequence of words and a binary tree whose leaf nodes correspond to words $\mathcal{V} = \{A, B, C, D, E, F, G, H\}$, the hierarchical softmax computes the probability of choosing the path from the root v_0 to the target node B, i.e., $\prod_{k=0}^{l(B)-1} P(c_k(B) \mid \boldsymbol{w}_i^c)$, instead of the probabilities over 8-way decisions $P(y = B \mid \boldsymbol{w}_i^c)$.

where $\mathbf{x}$ is a $(n-1)D$ dimensional vector. The target associated with $\mathbf{x}$ is w_i, an index of the n-th word in n-grams. The position of the target word is not necessarily to be the very last word in n-grams depending on an application of interest. Another way of learning word embeddings is to predict a word given its context that surrounds the target word. In other words, the position of a word to be predicted can be also in the middle of n-grams, where the context of w_i is $\boldsymbol{w}_i^c = \{w_{i-\lfloor n/2 \rfloor}, \cdots, w_{i-1}, w_{i+1}, \cdots, w_{i+\lfloor n/2 \rfloor}\}$. Given the type of context, word embeddings encode different information.

The objective of neural language modeling is to maximize the average of the conditional log probabilities

$$\frac{1}{T} \sum_{i=1}^{T} \log P(y = w_i \mid \boldsymbol{w}_i^c). \tag{2.79}$$

This can be also seen as a $|\mathcal{V}|$-way classification problem. As the size of the word vocabulary $\mathcal{V}$ for language modeling is very large in general, the computational cost at the output layer is a bottleneck of learning such an architecture because of the matrix multiplication steps involved in Eqs. (2.73) and (2.74) when L is greater than tens or hundreds of thousands.

As mentioned above, the most expensive computation for training a language model with NNs is to compute probabilities at the output layer that require normalization over all words. Instead of the vanilla softmax, one can consider a tree-based softmax (Morin and Bengio, 2005). This reduces the computational complexity from $\mathcal{O}(L)$ to $\mathcal{O}(\log_2 L)$ if the tree of labels is balanced. The hierarchical softmax builds a tree of labels where leaf nodes correspond to labels, illustrated in Figure 2.5. The probability predicting a target word w_i given its context $\boldsymbol{w}_i^c$ is reformulated as the probability of choosing the path from the root in the tree to the leaf node corresponding to the target word. Non-terminal nodes in the tree have their own parameters denoted by $\overline{\mathbf{U}} = \left[\bar{\mathbf{u}}_{v_0}, \bar{\mathbf{u}}_{v_1}, \cdots, \bar{\mathbf{u}}_{v_{L-1}}\right]$, where each column $\bar{\mathbf{u}}_v$ is a H-dimensional vector. While visiting all non-terminal nodes on a path from the root to a leaf node, we compute multiple binary decision probabilities. To be more specific, we have a context vector $\mathbf{x} \in \mathbb{R}^{D(n-1)\times 1}$ resulting from concatenation of D-dimensional vectors for $n-1$ words in $\boldsymbol{w}_i^c$. A path can be represented by a bit vector $\mathbf{c} = \{0, 1\}^{l(w_i)}$ where

$l(w_i)$ denotes the length of the path to the leaf node corresponding to w_i and 1 in the bit vector stands for moving to the left subtree. The probability of moving to the left subtree at position k over a path of length $l(w_i)$ is computed by

$$P(c_k(w_i) \mid \boldsymbol{w}_i^c) = \sigma\left([\![c_k(w_i) = 0]\!]\mathbf{a}^T f(\mathbf{W}\mathbf{x} + \bar{\mathbf{u}}_{pa(w_i,k)})\right) \tag{2.80}$$

where $c_k(w_i)$ denotes the k-th element in the bit vector for word w_i, $pa(w_i, k)$ denotes the k-th non-terminal node index on the path to w_i, $[\![\cdot]\!]$ is 1 if its argument is true and -1 otherwise, f is a non-linear function, and $\mathbf{W} \in \mathbb{R}^{H \times D(n-1)}$ and $\mathbf{a} \in \mathbb{R}^{H \times 1}$ are parameters. By replacing the conditional probabilities by the regular softmax in Eq. (2.79) with the probabilities by the hierarchical softmax in Eq. (2.80), we have

$$\frac{1}{T}\sum_{i=1}^{T} \log P(y = w_i \mid \boldsymbol{w}_i^c) = \frac{1}{T}\sum_{i=1}^{T}\sum_{k=0}^{l(w_i)} \log P(c_k(w_i) \mid \boldsymbol{w}_i^c). \tag{2.81}$$

The average path length from the root to a leaf node of a binary tree is $\log_2(|\mathcal{V}|)$ if the tree is balanced, so that the number of matrix multiplications required for the hierarchical softmax is much less than ones for the vanilla softmax.

In addition to the use of the hierarchical softmax, Mnih and Hinton (2009) and Mikolov et al. (2013a) have proposed even more efficient neural language modeling approaches. Although the hierarchical softmax greatly reduces the number of operations at the output layer from $\mathcal{O}(|\mathcal{V}|)$ to $\mathcal{O}(\log_2(|\mathcal{V}|))$, computing the hidden activations $f(\mathbf{W}\mathbf{x} + \bar{\mathbf{u}}_v)$ in Eq. (2.80) is still expensive and we need to evaluate them $\log_2(|\mathcal{V}|)$ times per target word. To reduce the computational cost, we can simplify the network architecture. Consider a NN that has a linear hidden layer parameterized by $\mathbf{U} \in \mathbb{R}^{H \times |\mathcal{V}|}$ unlike one used for computing Eq. (2.80). We project the context words by $\mathbf{U}$ and the context vector $\hat{\mathbf{u}}_i$ for a target word w_i is given by

$$\hat{\mathbf{u}}_i = \sum_{w_k \in \boldsymbol{w}_i^c} \mathbf{u}_{w_k}. \tag{2.82}$$

The probability of a binary decision over a path can then be represented by a dot product of two vectors given by

$$P(c_k(w_i) \mid \boldsymbol{w}_i^c) = \sigma\left([\![c_k(w_i) = 0]\!]\hat{\mathbf{u}}_i^T \bar{\mathbf{u}}_{pa(w_i,k)}\right). \tag{2.83}$$

Such a simpler architecture allows us to train neural language models on a large training corpus and to increase the capacity of word embeddings.

NLMs perform better than traditional count-based LMs such as *Kneser-Ney* (KN) smoothing (Kneser and Ney, 1995). In addition to the data-scarcity problem, another major issue of the count-based LMs is the curse of dimensionality that arises when learning from data in high-dimensional spaces, for which the amount of data we need grows exponentially in terms of the dimensionality. For instance, if we have a word vocabulary $\mathcal{V}$ of $100,000$ words and a sequence of 10 words, LMs need to learn a highly complex function over 10^{50} configurations. NLMs are able to avoid the curse of dimensionality problem effectively by learning lower dimensional representations of high dimensional data such as word sequences. We can generalize the central idea to learning representations of some other discrete inputs and outputs. We will discuss the use of lower dimensional representation learning methods for MLC in Chapters 5 and 6.

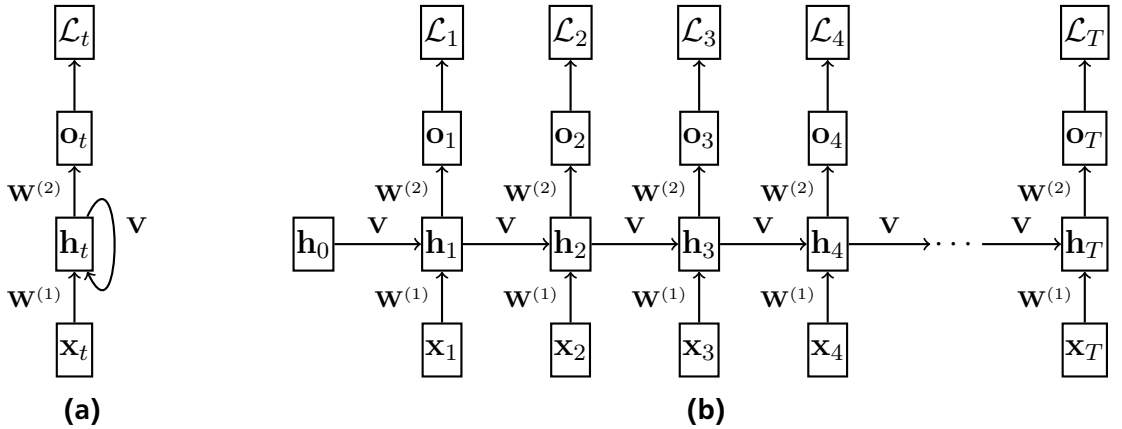


Figure 2.6: (a) A recurrent neural network with a hidden layer in which activations $\mathbf{h}_t$ at time t are fed back into the hidden layer for computing hidden activations $\mathbf{h}_{t+1}$. (b) Another representation of RNNs unfolded over T time steps.

2.3.3 Recurrent Neural Networks

We have shown that language modeling can be addressed by MLPs under the Markovian assumption. As MLP-based approaches to n-grams language modeling use $n-1$ previous words to predict the next word, it is unable to capture dependencies between the target word and words out of the range of $n-1$. To overcome the limitation, one may consider an architecture that does not rely on a fixed number of previous inputs when predicting the future output. Also, it is necessary that the architecture has the capability to learn the long range dependencies between inputs over the entire sequence.

A *recurrent neural network* (RNN) is a special NN architecture that has long been used for learning from sequential data. Let us consider a sequence prediction problem where a pair of an instance and a label vector $(\boldsymbol{x}, \boldsymbol{y})$ is given as training information and both an instance and a label are sequences of T vectors. More specifically, assume that an instance is a sequence of d-dimensional vectors $\boldsymbol{x} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T\}$ and a label is also a sequence of 1-of-L vectors $\boldsymbol{y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_T\}$, where $y_i \in \boldsymbol{y}$ are L dimensional binary vectors and only one element of each vector is 1. In other words, we perform multi-class classification T times for a given instance. Figure 2.6 illustrates a RNN parameterzied by three sets of weights $\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{V}\}$. At step $t \in \{1, 2, \cdots, T\}$, RNN utilizes the previous hidden activation $\mathbf{h}_{t-1}$ to compute the current hidden activation:

$$\mathbf{z}_t = \mathbf{W}^{(1)}\mathbf{x}_t + \mathbf{V}\mathbf{h}_{t-1} \tag{2.84}$$

$$\mathbf{h}_t = f(\mathbf{z}_t) \tag{2.85}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{H \times D}$ denotes a matrix of weights connecting input units to hidden units, $\mathbf{V} \in \mathbb{R}^{H \times H}$ denotes a set of weights from hidden units at step $t-1$ to t, and f is a non-linear transfer function such as tanh. The use of the previous hidden activation in computing the current hidden activation makes RNNs more useful in sequence prediction tasks compared to FNNs. Given hidden activations $\mathbf{h}_t$ at step t, we obtain output activations $\mathbf{o}_t$ at that step as follows

$$\mathbf{o}_t = \mathbf{W}^{(2)}\mathbf{h}_t \tag{2.86}$$

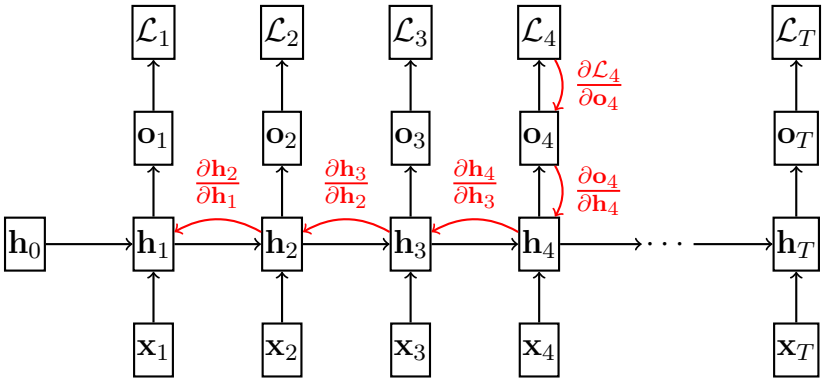


Figure 2.7: The error term $\delta_t = \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$ is propagated backwards while being multiplied by the Jacobian matrices (in red).

where $\mathbf{W}^{(2)} \in \mathbb{R}^{L \times H}$ is the output weights. Then, we have a prediction probability $p_{k,t}$ for output unit k at step t by the softmax function as in Eq. (2.46). In turn, the discrepancy between targets $\mathbf{y}_t$ and predictions $\mathbf{p}_t$ is measured by the cross entropy denoted by $\mathcal{L}_t$. The total error $\mathcal{L}$ of the network in Fig. 2.6 can be defined by a sum of the errors in time:

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t. \tag{2.87}$$

The unrolled RNNs can be thought of as deep feed-forward neural networks in time, so we can apply the backpropagation algorithm to RNNs as well. Since the gradients of the total loss $\mathcal{L}$ with respect to the parameters $\theta = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{V}\}$ are the sum of the gradients of the loss $\mathcal{L}_t$ at step t, we consider only $\partial \mathcal{L}_t / \partial \theta$ (Fig. 2.7).

Similar to Eq. (2.51), we can calculate the gradient of $\mathcal{L}_t$ with respect to the output weights $\mathbf{W}^{(2)}$ in a straightforward way given by

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}^{(2)}}. \tag{2.88}$$

Note that it is possible to parallelize computing $\partial \mathcal{L}_t / \partial \mathbf{W}^{(2)}$ efficiently because this step is time independent. In contrast, we need to pay more attention to computing the gradients of $\mathcal{L}_t$ in terms of $\mathbf{W}^{(1)}$ and $\mathbf{V}$. This is because these weights are shared when calculating the hidden activations in Eq. (2.85) across all steps and used repeatedly.

Training difficulties in RNNs. Let us compute the gradient of $\mathcal{L}_t$ in terms of the recurrent weights $\mathbf{V}$. By applying the chain rule, we have

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{V}} = \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{V}} \tag{2.89}$$

with

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{V}^T \mathrm{diag}(f'(\mathbf{z}_t)) \tag{2.90}$$

where $\text{diag}(f'(\mathbf{z}_t)) \in \mathbb{R}^{H \times H}$ denotes a diagonal matrix with elements of $f'(\mathbf{z}_t) \in \mathbb{R}^{H \times 1}$ on the main diagonal. Likewise, we also obtain the gradient of $\mathcal{L}_t$ in terms of $\mathbf{W}^{(1)}$ as follows

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{W}^{(1)}} = \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}^{(1)}}. \tag{2.91}$$

The vanilla RNNs are prone to two problems while learning: *vanishing* and *exploding* gradients. To obtain the gradients $\partial \mathcal{L}_t / \partial \mathbf{V}$ and $\partial \mathcal{L}_t / \partial \mathbf{W}^{(1)}$, we need to calculate the product of the Jacobian matrices (Eq. (2.90)). According to Eq. (2.90), the 2-norm of the Jacobian matrix $\partial \mathbf{h}_i / \partial \mathbf{h}_{i-1}$ is bounded by two matrices such that

$$\left\| \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \right\| \leq \|\mathbf{V}^T\| \|\text{diag}(f'(\mathbf{z}_t))\| \leq \gamma_{\mathbf{V}} \gamma_{f'} \tag{2.92}$$

where $\gamma_{\mathbf{V}}$ is the largest eigenvalue of $\mathbf{V}$ and $\gamma_{f'}$ denotes the upper bound of the derivative of f. Given Eq. (2.92), we have the upper bound of the 2-norm of temporal contributions $\partial \mathbf{h}_t / \partial \mathbf{h}_k$ as follows

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| \leq \left(\gamma_{\mathbf{V}} \gamma_{f'} \right)^{t-k}. \tag{2.93}$$

This shows that the temporal contributions either *vanish* or *explode* unless $\gamma_{\mathbf{V}} \gamma_{f'} = 1$. In other words, the long-range contributions in time approach zero as $t - k$ goes to infinity, or the magnitude of the gradient increases exponentially in $t - k$ if the spectral norm of $\mathbf{V}$ is large than $\frac{1}{\gamma_{f'}}$.

Solutions to the exploding gradient problem. A simple, yet effective solution to the exploding gradient problem was proposed by Pascanu et al. (2013). Suppose that we have a threshold of the 2-norm of the gradients denoted by γ. If the 2-norm of the gradients of $\mathcal{L}$ in terms of all the parameters θ exceeds the threshold, the gradients are normalized before updating parameters so that we ensure the norm of the gradients is bounded:

$$\nabla_\theta \mathcal{L} \leftarrow \frac{\gamma}{\|\nabla_\theta \mathcal{L}\|} \nabla_\theta \mathcal{L}. \tag{2.94}$$

This method is called *gradient clipping* and it has been used widely in training RNNs to avoid the exploding gradient problem.

Solutions to the vanishing gradient problem. In vanilla RNNs hidden activations $\mathbf{h}_t$ may not exploit effectively the past information, say, hidden activations $\mathbf{h}_k$ if $t \gg k$ because the gradient $\partial \mathbf{h}_t / \partial \mathbf{h}_k$ goes to zero. A simple solution is to add direct connections from the past every l steps, so that RNNs take advantage of shortcuts or skip connections when propagating errors from $\mathbf{h}_t$ to $\mathbf{h}_k$. However, this introduces additional hyperparameters l to be tuned while the problems remain but occur at a lower rate.

A principled way of achieving the goal is *long short-term memory* (LSTM) (Hochreiter and Schmidhuber, 1997) that learns read, write and erase operations when computing hidden activations adaptively from data. The main idea of LSTM is to keep information learned from data and to carry the information through time as long as it is useful.

Let us denote $\mathbf{m}_t \in \mathbb{R}^{H\times 1}$ as the information the model keeps, called *memory* at step t. To compute $\mathbf{m}_t$, LSTMs have three functions called as *gates*: input, forget and output gate. These gates in principle determine the information flow in LSTMs. For simplicity, we will drop the superscript of $\mathbf{W}$ wherever it is clear from the context. The input gate denoted by $\bar{\mathbf{i}}_t \in \mathbb{R}^{H\times 1}$ determines how much information available at step t needs to be *added* to memory $\mathbf{m}_t$:

$$\bar{\mathbf{z}}_t^i = \mathbf{W}^i \mathbf{x}_t + \mathbf{V}^i \mathbf{h}_{t-1} \tag{2.95}$$
$$\bar{\mathbf{i}}_t = \sigma\left(\bar{\mathbf{z}}_t^i\right) \tag{2.96}$$

where $\mathbf{W}^i \in \mathbb{R}^{H\times D}$ and $\mathbf{V}^i \in \mathbb{R}^{H\times H}$ denote weights connecting inputs at step t and previous hidden activations $\mathbf{h}_{t-1}$, respectively, to units in the input gate like hidden activation computation in vanilla RNNs.

It might not be a good way of carrying information by simply adding it to memory every step because the memory capacity is restricted by, say, the dimensionality of the memory vector. Also, some information from the past might not be useful to make future decisions. The forget gate denoted by $\bar{\mathbf{f}}_t \in \mathbb{R}^{H\times 1}$ learns how much information we need to unload or *reset* from memory $\mathbf{m}_{t-1}$:

$$\bar{\mathbf{z}}_t^f = \mathbf{W}^f \mathbf{x}_t + \mathbf{V}^f \mathbf{h}_{t-1} \tag{2.97}$$
$$\bar{\mathbf{f}}_t = \sigma\left(\bar{\mathbf{z}}_t^f\right) \tag{2.98}$$

Given the input and forget gates, we can compute memory states $\mathbf{m}_t$ at step t by using memory states from the previous step $\mathbf{m}_{t-1}$ and (immediate) memory proposals $\widetilde{\mathbf{m}}_t$ as follows

$$\bar{\mathbf{z}}_t^m = \mathbf{W}^m \mathbf{x}_t + \mathbf{V}^m \mathbf{h}_{t-1} \tag{2.99}$$
$$\widetilde{\mathbf{m}}_t = \tanh\left(\bar{\mathbf{z}}_t^m\right) \tag{2.100}$$
$$\mathbf{m}_t = \bar{\mathbf{f}}_t \odot \mathbf{m}_{t-1} + \bar{\mathbf{i}}_t \odot \widetilde{\mathbf{m}}_t \tag{2.101}$$

Once memory states $\mathbf{m}_t$ are updated (Eq. (2.101)), the next step is to calculate hidden activations $\mathbf{h}_t$. Note that memory states $\mathbf{m}_t$ convey information selectively up to t. Like computing input and forget gates, we also put output gates $\bar{\mathbf{o}}_t \in \mathbf{R}^{H\times 1}$ to control which information needs to be retrieved from memory $\mathbf{m}_t$ given by

$$\bar{\mathbf{z}}_t^o = \mathbf{W}^o \mathbf{x}_t + \mathbf{V}^o \mathbf{h}_{t-1} \tag{2.102}$$
$$\bar{\mathbf{o}}_t = \sigma\left(\bar{\mathbf{z}}_t^o\right). \tag{2.103}$$

In turn, LSTM computes hidden activations $\mathbf{h}_t$ as follows

$$\mathbf{h}_t = \bar{\mathbf{o}}_t \odot \tanh\left(\mathbf{m}_t\right). \tag{2.104}$$

The hidden activations calculated in Eq. (2.104) are forwarded subsequently to the output layer as vanilla RNNs. The intermediate steps to compute hidden activations $\mathbf{h}_t$ and memory states $\mathbf{m}_t$ of LSTMs can be represented in a shorthand form as follows

$$\mathbf{h}_t, \mathbf{m}_t = \text{LSTM}\left(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{m}_{t-1}\right) \tag{2.105}$$

where LSTM$(\cdot)$ denotes a parameterized function that returns updated hidden activations and memory states.

Let us compare memory states $\mathbf{m}_t$ in LSTM to hidden states $\mathbf{h}_t$ in a vanilla RNN architecture as an information flow path over time. By expanding the recurrence of LSTM in Eq. (2.101), we have

$$\begin{aligned}\mathbf{m}_t &= \bar{\mathbf{i}}_t \odot \widetilde{\mathbf{m}}_t + \bar{\mathbf{f}}_t \odot \mathbf{m}_{t-1} \\ &= \bar{\mathbf{i}}_t \odot \widetilde{\mathbf{m}}_t + \sum_{k=1}^{t-1} \left(\prod_{j=k+1}^{t} \bar{\mathbf{f}}_j \right) \odot \left(\bar{\mathbf{i}}_k \odot \widetilde{\mathbf{m}}_k \right) + \left(\prod_{k=1}^{t} \bar{\mathbf{f}}_k \right) \odot \mathbf{m}_0 \end{aligned} \tag{2.106}$$

with

$$\prod_{k=1}^{t} \bar{\mathbf{f}}_k = \bar{\mathbf{f}}_1 \odot \bar{\mathbf{f}}_1 \odot \cdots \odot \bar{\mathbf{f}}_t \tag{2.107}$$

where $\mathbf{m}_0$ denotes the initial memory that can be a vector of all zeros or initialized by context depending on applications. Remember that in vanilla RNNs hidden activations $\mathbf{h}_t$ that preserve information from the past can be computed by a product of the linear projections of previous hidden activations $\mathbf{h}_k$, followed by nonlinearity f as follows

$$\mathbf{h}_t = f\left(\mathbf{V} f\left(\mathbf{V} f\left(\cdots f\left(\mathbf{V}\mathbf{h}_0 + \mathbf{W}\mathbf{x}_1\right)\cdots\right) + \mathbf{W}\mathbf{x}_{t-1}\right) + \mathbf{W}\mathbf{x}_t\right). \tag{2.108}$$

In contrast, memory states $\mathbf{m}_t$ at step t in LSTMs are a *weighted* sum of memory proposals $\widetilde{\mathbf{m}}_k$ over time such that $1 \leq k \leq t$. We initialize $\mathbf{m}_0$ with all zeros. Please note that the weighting factors are basically calculated by input and forget gates (Eqs. (2.98) and (2.103)) adaptively based on immediate inputs and previous hidden activations.

As mentioned in Eq. (2.105), one can consider LSTM as a building block that takes three inputs $\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{m}_{t-1}$ and returns two outputs $\mathbf{h}_t, \mathbf{m}_t$. Suppose that we want to calculate the gradients of the loss function $\mathcal{L}_t$ with respect to the LSTM parameters, and that we are given errors δ to be propagated to LSTM from the upper layer and from LSTM at step $t+1$. To be more specific, let $\delta_{\mathbf{h}_t}$ be the contributions of hidden activations to the loss at step t given by

$$\delta_{\mathbf{h}_t} \coloneqq \frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t} = \widetilde{\delta}_{\mathbf{h}_t} + \overleftarrow{\delta}_{\mathbf{h}_t} \tag{2.109}$$

where $\widetilde{\delta}_{\mathbf{h}_t} \in \mathbb{R}^{H \times 1}$ denotes the incoming error term from the upper layer to hidden activations $\mathbf{h}_t$, and $\overleftarrow{\delta}_{\mathbf{h}_t} \in \mathbb{R}^{H \times 1}$ denotes the error contribution of $\mathbf{h}_t$ as an input to LSTM at step $t+1$, which is set to all zeros $\overleftarrow{\delta}_{\mathbf{h}_t} = \mathbf{0}$ if $t = T$. We will derive how to obtain $\overleftarrow{\delta}_{\mathbf{h}_t}$ shortly. Similarly, we also define $\delta_{\mathbf{m}_t}$ as the error term of memory states $\mathbf{m}_t$:

$$\delta_{\mathbf{m}_t} \coloneqq \frac{\partial \mathcal{L}_t}{\partial \mathbf{m}_t} = \widetilde{\delta}_{\mathbf{m}_t} + \overleftarrow{\delta}_{\mathbf{m}_t} \tag{2.110}$$

where $\widetilde{\delta}_{\mathbf{m}_t} \in \mathbb{R}^{H \times 1}$ denotes the error term passed from the upper layer, i.e., hidden activations $\mathbf{h}_t$, and $\overleftarrow{\delta}_{\mathbf{m}_t}$ denotes the error term from memory states but at step $t+1$. If we

rewrite both error terms in Eq. (2.110) with products of the intermediate gradients, the error of memory states at step t can be calculated as

$$\delta_{\mathbf{m}_t} = \frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{m}_t} + \frac{\partial \mathcal{L}_{t+1}}{\partial \mathbf{m}_{t+1}}\frac{\partial \mathbf{m}_{t+1}}{\partial \mathbf{m}_t} \tag{2.111}$$

$$= \delta_{\mathbf{h}_t} \odot \bar{\mathbf{o}}_t \odot \big(1 - \tanh^2(\mathbf{m}_t)\big) + \delta_{\mathbf{m}_{t+1}} \odot \bar{\mathbf{f}}_{t+1} \tag{2.112}$$

where $\delta_{\mathbf{m}_{t+1}}$ denotes the error of memory states at the next step. Given $\delta_{\mathbf{h}_t}$ and $\delta_{\mathbf{m}_t}$ obtained above, error signals for the gates and memory proposal can be derived as follows

$$\delta_{\bar{\mathbf{i}}_t} := \frac{\partial \mathcal{L}_t}{\partial \mathbf{m}_t}\frac{\partial \mathbf{m}_t}{\partial \bar{\mathbf{i}}_t}\frac{\partial \bar{\mathbf{i}}_t}{\partial \bar{\mathbf{z}}_t^i} = \delta_{\mathbf{m}_t} \odot \widetilde{\mathbf{m}}_t \odot \bar{\mathbf{i}}_t \odot (1 - \bar{\mathbf{i}}_t) \tag{2.113}$$

$$\delta_{\bar{\mathbf{f}}_t} := \frac{\partial \mathcal{L}_t}{\partial \mathbf{m}_t}\frac{\partial \mathbf{m}_t}{\partial \bar{\mathbf{f}}_t}\frac{\partial \bar{\mathbf{f}}_t}{\partial \bar{\mathbf{z}}_t^f} = \delta_{\mathbf{m}_t} \odot \mathbf{m}_{t-1} \odot \bar{\mathbf{f}}_t \odot (1 - \bar{\mathbf{f}}_t) \tag{2.114}$$

$$\delta_{\bar{\mathbf{o}}_t} := \frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \bar{\mathbf{o}}_t}\frac{\partial \bar{\mathbf{o}}_t}{\partial \bar{\mathbf{z}}_t^o} = \delta_{\mathbf{h}_t} \odot \tanh(\mathbf{m}_t) \odot \bar{\mathbf{o}}_t \odot (1 - \bar{\mathbf{o}}_t) \tag{2.115}$$

$$\delta_{\bar{\mathbf{m}}_t} := \frac{\partial \mathcal{L}_t}{\partial \mathbf{m}_t}\frac{\partial \mathbf{m}_t}{\partial \widetilde{\mathbf{m}}_t}\frac{\partial \widetilde{\mathbf{m}}_t}{\partial \bar{\mathbf{z}}_t^m} = \delta_{\mathbf{m}_t} \odot \bar{\mathbf{i}}_t \odot (1 - \widetilde{\mathbf{m}}_t^2). \tag{2.116}$$

Once we have all error terms in LSTM, the next step is to calculate the gradients of the parameters. For the sake of notational convenience, let us define the following matrices by concatenating the parameters, and the pre-activations and the error terms of the gates in LSTM:

$$\mathbf{W} := \begin{bmatrix} \mathbf{W}^i \\ \mathbf{W}^f \\ \mathbf{W}^o \\ \mathbf{W}^m \end{bmatrix} \qquad \mathbf{V} := \begin{bmatrix} \mathbf{V}^i \\ \mathbf{V}^f \\ \mathbf{V}^o \\ \mathbf{V}^m \end{bmatrix} \qquad \bar{\mathbf{z}}_t := \begin{bmatrix} \bar{\mathbf{z}}_t^i \\ \bar{\mathbf{z}}_t^f \\ \bar{\mathbf{z}}_t^o \\ \bar{\mathbf{z}}_t^m \end{bmatrix} \qquad \delta_{\bar{\mathbf{z}}_t} := \begin{bmatrix} \delta_{\bar{\mathbf{i}}_t} \\ \delta_{\bar{\mathbf{f}}_t} \\ \delta_{\bar{\mathbf{o}}_t} \\ \delta_{\widetilde{\mathbf{m}}_t} \end{bmatrix} \tag{2.117}$$

where $\mathbf{W}$ is a $(4H \times D)$-dimensional matrix, $\mathbf{V}$ is a $(4H \times H)$-dimensional matrix, and both $\bar{\mathbf{z}}_t$ and $\delta_{\bar{\mathbf{z}}_t}$ are $(4H \times 1)$-dimensional vectors. The gradients of $\mathcal{L}_t$ with respect to the parameters, i.e., $\mathbf{W}$ and $\mathbf{V}$, are given by

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_t}{\partial \bar{\mathbf{z}}_t}\frac{\partial \bar{\mathbf{z}}_t}{\partial \mathbf{W}} = \delta_{\bar{\mathbf{z}}_t}\mathbf{x}_t^T \tag{2.118}$$

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{V}} = \frac{\partial \mathcal{L}_t}{\partial \bar{\mathbf{z}}_t}\frac{\partial \bar{\mathbf{z}}_t}{\partial \mathbf{V}} = \delta_{\bar{\mathbf{z}}_t}\mathbf{h}_{t-1}^T. \tag{2.119}$$

Like $\delta_{\mathbf{m}_t}$, we can also calculate errors of inputs, i.e., $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$, to be propagated backwards further in the network as follows

$$\delta_{\mathbf{x}_t} := \frac{\partial \mathcal{L}_t}{\partial \bar{\mathbf{z}}_t}\frac{\partial \bar{\mathbf{z}}_t}{\partial \mathbf{x}_t} = \mathbf{W}^T\delta_{\bar{\mathbf{z}}_t} \tag{2.120}$$

$$\overleftarrow{\delta}_{\mathbf{h}_{t-1}} := \frac{\partial \mathcal{L}_t}{\partial \bar{\mathbf{z}}_t}\frac{\partial \bar{\mathbf{z}}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{V}^T\delta_{\bar{\mathbf{z}}_t}. \tag{2.121}$$

Note that $\overleftarrow{\delta}_{\mathbf{h}_{t-1}}$ (Eq. (2.121)) can be plugged into Eq. (2.109) when computing $\delta_{\mathbf{h}_{t-1}}$.

We have discussed the vanishing and exploding gradient problem in vanilla RNNs caused by $\partial \mathbf{h}_t / \partial \mathbf{h}_k$, namely the Jacobian terms, when $t \gg k$. In contrast, memory states $\mathbf{m}_t$ in LSTM have a linear recurrence relation which allows us to avoid the computation of powers of $\mathbf{V}$. Thus, LSTMs are more robust than vanilla RNNs with respect to the vanishing gradient problem.

Another recurrent unit using gating functions is the *gated recurrent unit* (GRU) (Cho et al., 2014). Unlike LSTM that has three gates, GRU has only two gates: *update* and *reset*. The *update* gate in GRU plays a role of the input gate and forget gate in LSTM, and the *reset* gate can be considered as the LSTM's output gate. Another difference between GRU and LSTM is that GRU does not maintain an extra internal information flow path such as memory in LSTM. To be more specific, hidden activations $\mathbf{h}_t$ in GRU are calculated as follows

$$\mathbf{z}_t = \sigma\left(\mathbf{W}^z \mathbf{x}_t + \mathbf{V}^z \mathbf{h}_{t-1}\right) \tag{2.122}$$

$$\mathbf{r}_t = \sigma\left(\mathbf{W}^r \mathbf{x}_t + \mathbf{V}^r \mathbf{h}_{t-1}\right) \tag{2.123}$$

$$\widetilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}^h \mathbf{x}_t + \mathbf{V}^h \left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right)\right) \tag{2.124}$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \widetilde{\mathbf{h}}_t. \tag{2.125}$$

We will use GRU as a black-box that returns hidden states $\mathbf{h}_t$ given $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ as follows

$$\mathbf{h}_t = \mathrm{GRU}\left(\mathbf{x}_t, \mathbf{h}_{t-1}\right). \tag{2.126}$$

Although GRU is a light-weight version of LSTM, it works well in practice and has been widely used in various applications.

2.3.4 Neural Machine Translation

Inspired by recent success of methods that learn vector representations of words from large corpora in an unsupervised way (Section 2.3.2), Devlin et al. (2014) have shown that word representations learned by neural language models are also effective as a core component of a complex translation system, which in general requires heavy feature engineering. By contrast, *neural machine translation* (NMT) models are end-to-end learning systems that tune model parameters in a way of generating a desired output sentence for a given input sentence of arbitrary length with minimal processing steps such as tokenization. Most of recently proposed NMT models are based on the encoder-decoder framework (Cho et al., 2014; Sutskever et al., 2014), where a source sentence is mapped into a fixed-size vector that preserves both the syntactic and semantic structure of the source sentence, from which a decoder starts with generating a sequence of words in a target language. Bahdanau et al. (2015) extend the vanilla encoder-decoder NMT framework by *soft-attention*, a small feed-forward neural network which learns which word in the source sentence is relevant for predicting the next word in the target sequence. It has been shown that the performance of soft-attention NMT stays consistent as the sequence length increases.

Although NMT models have been successfully applied to translation tasks, it is still challenging to handle *out-of-vocabulary* (OOV) words because only a small number of words are

considered to reduce computational overhead. All words not in the vocabulary are assigned to a single special token, e.g., UNK. In the literature, the OOV word problem have been addressed by using importance sampling (Jean et al., 2015), smaller linguistic units such as subwords or characters (Chung et al., 2016; Luong and Manning, 2016; Sennrich et al., 2016), and linguistic properties such as compound words (Hirschmann et al., 2016).

We will only explain *encoder-decoder* (EncDec) networks dealing with word sequences because aforementioned approaches addressing the OOV word problem are beyond the scope of this thesis. Consider that we have a pair of a source sentence and its translation in another language, for instance, (*Wie geht es dir?*, *How are you?*) when translating German sentences into English. Suppose that a word is treated as an atomic unit of translation and that we have two word vocabularies $\mathcal{V}_s$ and $\mathcal{V}_t$, let us define a source sentence and target sentence by $\boldsymbol{x} = \{x_1, x_2, \cdots, x_S\}$ and $\boldsymbol{y} = \{y_1, y_2, \cdots, y_T\}$, respectively, where $x_j \in \mathcal{V}_s$ denote source words and $y_i \in \mathcal{V}_t$ denote target words. Note that the lengths S and T for both sentences in each pair $(\boldsymbol{x}, \boldsymbol{y})$ will vary depending on its context, and they are not necessarily to be same.

The goal of training translation systems can be expressed as maximizing the joint probability of target words $\boldsymbol{y}$ conditioned on source words $\boldsymbol{x}$, and then we can factorize it into a product of conditional probabilities of a single target word:

$$p\left(\boldsymbol{y}|\boldsymbol{x}\right) = \prod_{t=1}^{T} p\left(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}\right) \tag{2.127}$$

where $\boldsymbol{y}_{<t} = \{y_1, \cdots, y_{t-1}\}$ denotes a set of words preceding a word at position i in the target sentence. Each conditional probability $p(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x})$ can be computed by

$$p(y_t = k|\boldsymbol{y}_{<t}, \boldsymbol{x}) = \frac{\exp(o_{tk})}{\sum_{v=1}^{|\mathcal{V}_t|} \exp(o_{tv})} \tag{2.128}$$

where $|\mathcal{V}_t|$ denotes the size of the target vocabulary and $o_{tk} \in \mathbb{R}$ is an output score produced by a RNN-based translation system for the k-th word in $\mathcal{V}_t$ being predicted as a word at position t.

For computing Eq. (2.128), we use the EncDec architecture, also known as *sequence-to-sequence* learning (Bahdanau et al., 2015; Cho et al., 2014; Sutskever et al., 2014), where two RNNs are trained jointly to handle such variable-length inputs and outputs. Let us consider an encoder RNN that summarizes source sentences. We use a RNN with GRUs for encoding source sentences instead of a vanilla RNN due to the vanishing gradient problem discussed in the previous section.[2] Let $\mathbf{U}^x \in \mathbb{R}^{d \times |\mathcal{V}_s|}$ be the embeddings for source words. If we denote word embeddings corresponding to words in $\boldsymbol{x}$ by $\{\mathbf{u}_1^x, \mathbf{u}_2^x, \cdots, \mathbf{u}_S^x\}$, hidden states $\mathbf{h}_i^x \in \mathbb{R}^{H \times 1}$ of source words can be calculated as follows

$$\mathbf{h}_i^x = \mathrm{GRU}\left(\mathbf{u}_i^x, \mathbf{h}_{i-1}^x\right) \tag{2.129}$$

where the initial hidden state $\mathbf{h}_0^x$ is set to a vector of zeros. Once hidden states over all of the source words are calculated by the encoder, we set the initial hidden state $\mathbf{h}_0^y \in \mathbb{R}^{H \times 1}$ of another RNN, namely decoder, using the last hidden state $\mathbf{h}_S^x$ of the encoder as follows

$$\mathbf{h}_0^y = \tanh\left(\mathbf{W}_{st}\mathbf{h}_S^x\right) \tag{2.130}$$

[2] LSTMs can be also used instead of GRUs.

where $\mathbf{W}_{st} \in \mathbb{R}^{H \times H}$ denotes weights connecting the encoder's hidden states to the decoder's hidden states.

Like the way to project source words, let $\mathbf{U}^y \in \mathbb{R}^{d \times |\mathcal{V}_t|}$ be the embeddings for target words, and the target words $\boldsymbol{y}$ are converted into $\{\mathbf{u}_1^y, \mathbf{u}_2^y, \cdots, \mathbf{u}_T^y\}$. Given the target word embeddings, one can compute hidden states $\mathbf{h}_t^y$ as in Eq. (2.129) where immediate inputs and previous hidden states are only used. Even though we use gated RNNs including GRU and LSTM to avoid the vanishing gradient problem, it is difficult to train them on long sequence pairs because of the information bottleneck between two RNNs, namely encoder and decoder. Note that the encoder summarizes all source words into a single vector, i.e., $\mathbf{h}_S^x$, that is the only information the decoder exploits from the source side.

Due to the limitation, Bahdanau et al. (2015) have proposed *conditional* GRU that takes a weighted average of hidden states of the encoder RNN as an additional input. In conditional GRUs, we compute hidden states as follows

$$\mathbf{z}_t = \sigma\left(\mathbf{W}^z \mathbf{x}_t + \mathbf{V}^z \mathbf{h}_{t-1} + \mathbf{L}^z \mathbf{c}_t\right) \tag{2.131}$$

$$\mathbf{r}_t = \sigma\left(\mathbf{W}^r \mathbf{x}_t + \mathbf{V}^r \mathbf{h}_{t-1} + \mathbf{L}^r \mathbf{c}_t\right) \tag{2.132}$$

$$\widetilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}^h \mathbf{x}_t + \mathbf{V}^h \left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) + \mathbf{L}^h \mathbf{c}_t\right) \tag{2.133}$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \widetilde{\mathbf{h}}_t. \tag{2.134}$$

where $\mathbf{c} \in \mathbb{R}^{d \times 1}$ is the context vector and $\mathbf{L}^{(\cdot)} \in \mathbb{R}^{H \times d}$ are weight matrices. In contrast to the vanilla GRU, the conditional GRU needs the context vector $\mathbf{c}_t$ as aforementioned, which can be calculated as

$$\mathbf{c}_t = \sum_{i=1}^{S} \alpha_{ti} \mathbf{h}_i^x \tag{2.135}$$

with

$$\alpha_{ti} = \frac{\exp(s_{ti})}{\sum_{k=1}^{S} \exp(s_{tk})} \tag{2.136}$$

$$s_{ti} = \mathbf{v}_{att}^T \tanh\left(\mathbf{W}_{att} \mathbf{h}_{t-1}^y + \mathbf{V}_{att} \mathbf{h}_i^x\right) \tag{2.137}$$

where $\mathbf{v}_{att} \in \mathbb{R}^{d_a \times 1}$, $\mathbf{W}_{att} \in \mathbb{R}^{d_a \times H}$ and $\mathbf{V}_{att} \in \mathbb{R}^{d_a \times H}$ are weights. In other words, the decoder RNN learns which source words are important when generating context vectors while computing hidden activations for each target word. Such a concept is often referred to as *soft-attention* and α_{ti} are attention weights. The decoder RNN computes hidden states $\mathbf{h}_t^y$ that takes into account the encoder's hidden states as follows

$$\mathbf{h}_t^y = \mathrm{GRU}\left(\mathbf{u}_{t-1}^y, \mathbf{h}_{t-1}^y, \mathbf{c}_t\right). \tag{2.138}$$

As the soft-attention component in EncDec allows the decoder to exploit source words while computing hidden states, we make use of information flow path from decoder's hidden states to encoder's hidden states directly.

We calculate output scores $\mathbf{o}_t$ given hidden states $\mathbf{h}_t^y$ as follows

$$\mathbf{o}_t = \mathbf{W}^{(2)} \mathbf{h}_t^y. \tag{2.139}$$

The output scores $\mathbf{o}_t$ are then used to yield the probability of predicting next words (Eq. (2.128)).

NMT approaches are trained in an end-to-end fashion, where all model parameters are learned jointly, and work surprisingly well in practice compared to traditional machine translation systems composed of many sub-components. Furthermore, attention-based NMT methods provide a way to analyze alignments between source words and target words by attention weights $\boldsymbol{\alpha}_t$ learned from data automatically.

3 Efficient Neural Networks for Large-scale Multi-label Classification

3.1 Introduction

As the amount of textual data on the web and in digital libraries is increasing rapidly, the need for augmenting unstructured data with metadata is also increasing. Systematically maintaining a high quality digital library requires extracting a variety types of information from unstructured text, from trivial information such as title and author, to non-trivial information such as descriptive keywords and categories. Time- and cost-wise, a manual extraction of such information from ever-growing document collections is impractical.

In the simplest case, *multi-label classification* (MLC) may be viewed as a set of binary classification tasks that decides for each label independently whether it should be assigned to the document or not. However, this so-called *binary relevance* (BR) approach ignores dependencies between the labels, so that current research in MLC concentrates on the question of how such dependencies can be exploited (Dembczyński et al., 2010; Read et al., 2011). One such approach is *backpropagation for multi-label learning* (BP-MLL) (Zhang and Zhou, 2006), which formulates MLC problems as a neural network with multiple output nodes, one for each label. The output layer is able to model dependencies between the individual labels.

In this work, we directly build upon BP-MLL and show how a simple, single hidden layer *neural network* (NN) may achieve a state-of-the-art performance in large-scale multi-label text classification tasks. The key modifications that we suggest are (i) more efficient and more effective training by replacing BP-MLL's pairwise ranking loss with cross entropy and (ii) the use of recent developments in the area of deep learning such as *rectified linear unit* (ReLU), Dropout, and AdaGrad.

Even though we employ techniques that have been developed in the realm of deep learning, we nevertheless stick to NNs with a single hidden layer. The motivation behind this is two-fold: first, a simple network configuration allows better scalability of the model and is more suitable for large-scale tasks. Second, as it has been shown in the literature (Joachims, 1998), popular feature representation schemes for textual data such as variants of *term frequency-inverse document frequency* (tf-idf) term weighting already incorporate a certain degree of higher dimensional features, and we speculate that even a single-layer NN model can work well with text data.

The most prominent learning method for multi-label text classification is to use a BR approach with strong binary classifiers such as *support vector machines* (SVMs) (Rubin et al., 2012; Yang and Gopal, 2012) despite its simplicity. It is well known that characteristics of high-dimensional and sparse data, such as text data, make decision problems linearly separable (Joachims, 1998), and this characteristic suits the strengths of SVM classifiers well.

Unlike benchmark datasets, real-world text collections consist of a large number of training examples represented in a high-dimensional space with a large amount of labels. To handle such datasets, researchers have derived efficient *linear* SVMs (Fan et al., 2008; Joachims, 2006) that can handle large-scale problems. The training time of these solvers scales linearly with the number of instances, so that they show good performance on standard benchmarks. However, their performance decreases as the number of labels grows and the label frequency distribution becomes skewed (Liu et al., 2005; Rubin et al., 2012). In such cases, it is also intractable to employ methods that minimize ranking errors among labels (Elisseeff and Weston, 2001; Zhang and Zhou, 2006) or that learn joint probability distributions of labels (Dembczyński et al., 2010; Ghamrawi and McCallum, 2005).

This chapter provides an empirical evidence to support that a simple NN model equipped with recent advanced techniques for training NNs performs as well as or even outperforms state-of-the-art approaches on large-scale datasets with diverse characteristics.

3.2 Neural Networks for Multi-label Classification

In this section, we propose a neural network-based multi-label classification framework that is composed of a single hidden layer and operates with recent developments in neural network and optimization techniques, which allow the model to converge into good regions of the error surface in a few steps of parameter updates. Our approach consists of two modules (Figure 3.1): a neural network that produces label scores (Sections 3.2.2–3.2.5), and a label predictor that converts label scores into binary using a thresholding technique (Section 3.2.3).

3.2.1 Rank Loss

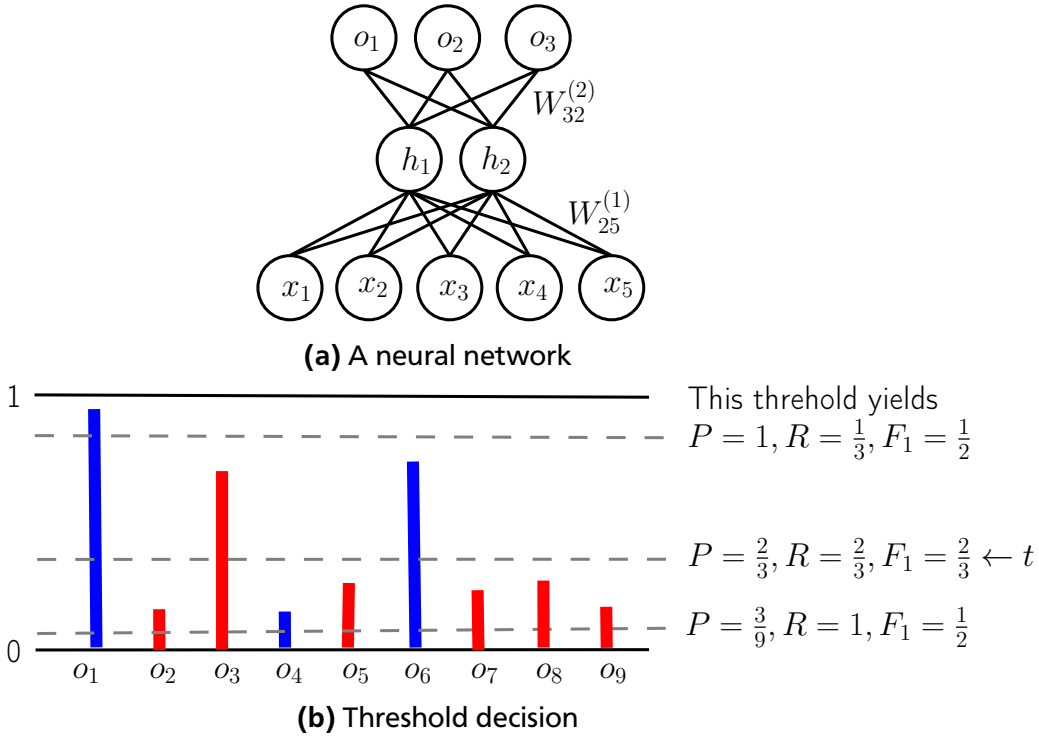
One of the most commonly used objectives for MLC (Section 2.2.1) is to minimize the number of mis-ordering between a pair of relevant label and irrelevant label, which is called *rank loss*:

$$\ell_{\text{rank}}(\mathbf{y}, f(\mathbf{x})) = w(\mathbf{y}) \sum_{y_i < y_j} \mathbb{I}\left(f_i(\mathbf{x}) > f_j(\mathbf{x})\right) + \frac{1}{2}\mathbb{I}\left(f_i = f_j\right) \tag{3.1}$$

where $w(\mathbf{y})$ is a normalization factor, $\mathbb{I}(\cdot)$ is the indicator function, and $f_i(\cdot)$ is a prediction score for a label i. Unfortunately, it is hard to minimize due to non-convex property of the loss function. Therefore, convex surrogate losses have been proposed as alternatives to rank loss (Elisseeff and Weston, 2001; Schapire and Singer, 2000; Zhang and Zhou, 2006).

3.2.2 Pairwise Ranking Loss Minimization in Neural Networks

Let us assume that we would like to make a prediction on L labels from D dimensional input features. Consider the neural network model with a single hidden layer in which H hidden units are defined and input units $\mathbf{x} \in \mathbb{R}^{D\times 1}$ are connected to hidden units $\mathbf{h} \in \mathbb{R}^{H\times 1}$ with weights $\mathbf{W}^{(1)} \in \mathbb{R}^{H\times D}$ and biases $\mathbf{b}^{(1)} \in \mathbb{R}^{H\times 1}$. The hidden units are connected to output units $\mathbf{o} \in \mathbb{R}^{L\times 1}$ through weights $\mathbf{W}^{(2)} \in \mathbb{R}^{L\times H}$ and biases $\mathbf{b}^{(2)} \in \mathbb{R}^{L\times 1}$. The network,



(a) A neural network

(b) Threshold decision

Figure 3.1: (a) A neural network with a single hidden layer of two units and multiple output units, one for each possible label. (b) shows how threshold for a training example is estimated based on prediction output $\mathbf{o}$ of the network. Consider nine possible labels, of which o_1, o_4 and o_6 are relevant labels (blue) and the rest are irrelevant (red). The figure shows three exemplary threshold candidates (dashed lines), of which the middle one is the best choice because it gives the highest F1 score. See Section 3.2.3 for more details.

then, can be written in a matrix-vector form, and we can construct a feed-forward network $f_\Theta : \mathbf{x} \to \mathbf{o}$ as a composite of non-linear functions in the range $[0, 1]$:

$$f_\Theta(\mathbf{x}) = f_o\left(\mathbf{W}^{(2)} f_h\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right) \tag{3.2}$$

where $\Theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}\}$, and f_o and f_h are *element-wise* activation functions in the output layer and the hidden layer, respectively. Specifically, the function $f_\Theta(\mathbf{x})$ can be re-written as follows:

$$\begin{aligned} \mathbf{z}^{(1)} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}, \quad \mathbf{h} = f_h\left(\mathbf{z}^{(1)}\right) \\ \mathbf{z}^{(2)} &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}, \quad \mathbf{o} = f_o\left(\mathbf{z}^{(2)}\right) \end{aligned}$$

where $\mathbf{z}^{(1)}$ and $\mathbf{z}^{(2)}$ denote pre-activations in the hidden and the output layer, respectively.

Our aim is to find a parameter vector Θ that minimizes a cost function $\mathcal{L}(\Theta; \mathbf{x}, \mathbf{y})$. The cost function measures discrepancy between predictions of the network and given targets $\mathbf{y}$.

BP-MLL (Zhang and Zhou, 2006) minimizes errors induced by incorrectly ordered pairs of labels, in order to exploit dependencies among labels. To this end, it introduces a *pairwise error function* (PWE), which is defined as follows:

$$\mathcal{L}_{PWE}(\Theta; \mathbf{x}, \mathbf{y}) = \frac{1}{|\mathbf{y}||\bar{\mathbf{y}}|} \sum_{(p,n)\in\mathbf{y}\times\bar{\mathbf{y}}} \exp(-(o_p - o_n)) \tag{3.3}$$

where p and n are positive and negative label index associated with training example $\mathbf{x}$. $\bar{\mathbf{y}}$ represents a set of negative labels and $|\cdot|$ stands for the cardinality. The PWE is relaxation of the loss function in Equation 3.1 that we want to minimize.

3.2.3 Thresholding

Once training of the neural network is finished, its output may be interpreted as a probability distribution $p(\mathbf{o}|\mathbf{x})$ over the labels for a given document $\mathbf{x}$. The probability distribution can be used to rank labels, but additional measures are needed in order to split the ranking into relevant and irrelevant labels. For transforming the ranked list of labels into a set of binary predictions, we train a multi-label threshold predictor from training data. This sort of thresholding methods are also used in (Elisseeff and Weston, 2001; Zhang and Zhou, 2006)

For each document $\mathbf{x}_m$, labels are sorted by the probabilities in decreasing order. Ideally, if NNs successfully learn a mapping function f_Θ, all correct (positive) labels will be placed on top of the sorted list and there should be large margin between the set of positive labels and the set of negative labels. Using F_1 score as a reference measure, we calculate classification performances at every pair of successive positive labels and choose a threshold value t_m that produces the best performance (Figure 3.1 b).

Afterwards, we train a multi-label thresholding predictor $\hat{\mathbf{t}} = T(\mathbf{x};\theta)$ to learn $\mathbf{t}$ as target values from input pattern $\mathbf{x}$. We use linear regression with $\ell 2$-regularization to learn θ

$$\mathcal{L}(\theta) = \frac{1}{2M}\sum_{m=1}^{M}\left(T(\mathbf{x}_m;\theta) - t_i\right)^2 + \frac{\lambda}{2}\|\theta\|_2^2 \tag{3.4}$$

where $T(\mathbf{x}_m;\theta) = \theta^T\mathbf{x}_m$ and λ is a parameter which controls the magnitude of the $\ell 2$ penalty.

At test time, these learned thresholds are used to predict a binary output $\hat{y}_{kl}$ for label l of a test document $\mathbf{x}_k$ given label probabilities o_{kl}; $\hat{y}_{kl} = 1$ if $o_{kl} > T(\mathbf{x}_k;\theta)$, otherwise 0.

3.2.4 Ranking Loss vs. Cross Entropy

BP-MLL is supposed to perform better in multi-label problems since it takes label correlations into consideration than the standard form of NN that does not. However, we have found that BP-MLL does not perform as expected in our preliminary experiments, particularly, on datasets in textual domain.

Consistency w.r.t Rank Loss. Recently, it has been claimed that none of convex loss functions including BP-MLL's loss function (Equation 3.3) is consistent with respect to *rank loss* which is non-convex and has discontinuity (Calauzènes et al., 2012; Gao and Zhou, 2013). Furthermore, univariate surrogate loss functions such as *log loss* are rather consistent with rank loss (Dembczyński et al., 2012a).

$$\mathcal{L}_{log}(\Theta;\mathbf{x},\mathbf{y}) = w(\mathbf{y})\sum_{l}\log\left(1+e^{-\dot{y}_l z_l}\right)$$

where $w(\mathbf{y})$ is a weighting function that normalizes loss in terms of $\mathbf{y}$ and z_l indicates prediction for label l. Please note that the log loss is often used for logistic regression in

which $\dot{y} \in \{-1, 1\}$ a target and z_l is output of a linear function $z_l = \sum_k W_{lk} x_k + b_l$ where W_{lk} is a weight from input x_k to output z_l and b_l is bias for label l. A typical choice is, for instance, $w(\mathbf{y}) = (|\mathbf{y}||\bar{\mathbf{y}}|)^{-1}$ as in BP-MLL. In this work, we set $w(\mathbf{y}) = 1$, then the log loss above is equivalent to *cross entropy* (CE), which is commonly used to train neural networks for classification tasks if we use *sigmoid* transfer function in the output layer, i.e. $f_o(z) = 1/\left(1 + \exp(-z)\right)$, or simply $f_o(z) = \sigma\left(z\right)$:

$$\mathcal{L}_{CE}(\Theta; \mathbf{x}, \mathbf{y}) = -\sum_l \left(y_l \log o_l\right) + \left(1 - y_l\right) \log(1 - o_l)) \tag{3.5}$$

where o_l and y_l are the prediction and the target for label l, respectively. Let us verify the equivalence between the *log loss* and the CE. Consider the log loss function for only label l.

$$\mathcal{L}_{log}(\Theta; \mathbf{x}, y_l) = \log(1 + e^{-\dot{y}_l z_l}) = -\log\left(\frac{1}{1 + e^{-\dot{y}_l z_l}}\right) \tag{3.6}$$

As noted, $\dot{y}$ in the log loss takes either -1 or 1, which allows us to split the above equation as follows:

$$-\log\left(\frac{1}{1 + e^{-\dot{y}_l z_l}}\right) = \begin{cases} -\log\left(\sigma\left(z_l\right)\right) & \text{if } \dot{y} = 1 \\ -\log\left(\sigma\left(-z_l\right)\right) & \text{if } \dot{y} = -1 \end{cases} \tag{3.7}$$

Then, we have the corresponding CE by using a property of the sigmoid function $\sigma\left(-z\right) = 1 - \sigma\left(z\right)$

$$\mathcal{L}_{CE}\left(\Theta; \mathbf{x}, y_l\right) = -\left(y_l \log o_l + \left(1 - y_l\right) \log\left(1 - o_l\right)\right) \tag{3.8}$$

where $y \in \{0, 1\}$ and $o_l = \sigma\left(z_l\right)$.

Computational Expenses In addition to consistency with rank loss, CE has an advantage in terms of computational efficiency; computational cost for computing gradients of parameters with respect to PWE is getting more expensive as the number of labels grows. The error term $\delta_l^{(2)}$ for label l which is propagated to the hidden layer is defined as

$$\delta_l^{(2)} = \begin{cases} -\frac{1}{|\mathbf{y}||\bar{\mathbf{y}}|} \sum_{n \in \bar{\mathbf{y}}} \exp(-(o_l - o_n)) f_o'(z_l^{(2)}), & \text{if } l \in \mathbf{y} \\ \frac{1}{|\mathbf{y}||\bar{\mathbf{y}}|} \sum_{p \in \mathbf{y}} \exp(-(o_p - o_l)) f_o'(z_l^{(2)}), & \text{if } l \in \bar{\mathbf{y}} \end{cases} \tag{3.9}$$

Whereas the computation of $\delta_l^{(2)} = -y_l/o_l + (1 - y_l)/(1 - o_l) f_o'(z_l^{(2)})$ for the CE can be performed efficiently, obtaining error terms $\delta_l^{(2)}$ for the PWE is L times more expensive than one in ordinary NN utilizing the cross entropy error function. This also shows that BP-MLL scales poorly w.r.t. the number of unique labels.

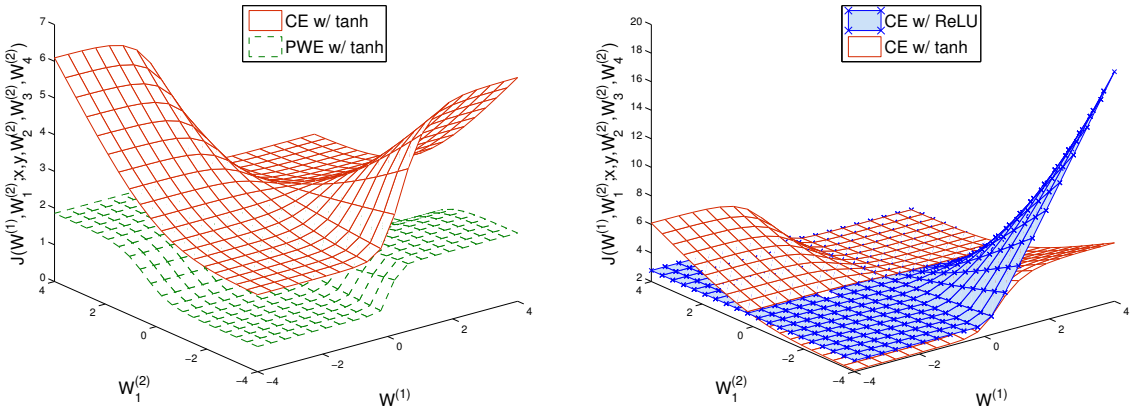


Figure 3.2: Landscape of cost functions and a type of hidden units. $W^{(1)}$ represents a weight connecting an input unit to a hidden unit. Likewise, $W_1^{(2)}$ denotes a weight from the hidden unit to output unit 1. The z-axis stands for a value for the cost function $J(W^{(1)}, W_1^{(2)}; \mathbf{x}, \mathbf{y}, W_2^{(2)}, W_3^{(2)}, W_4^{(2)})$ where instances $\mathbf{x}$, targets $\mathbf{y}$ and weights $W_2^{(2)}, W_3^{(2)}, W_4^{(2)}$ are fixed.

Plateaus To get an idea of how differently both objective functions behave as a function of parameters to be optimized, let us draw graphs containing cost function values. Note that it has been pointed out that the slope of the cost function as a function of the parameters plays an important role in learning parameters of neural networks (Glorot and Bengio, 2010; Solla et al., 1988) which we follow.

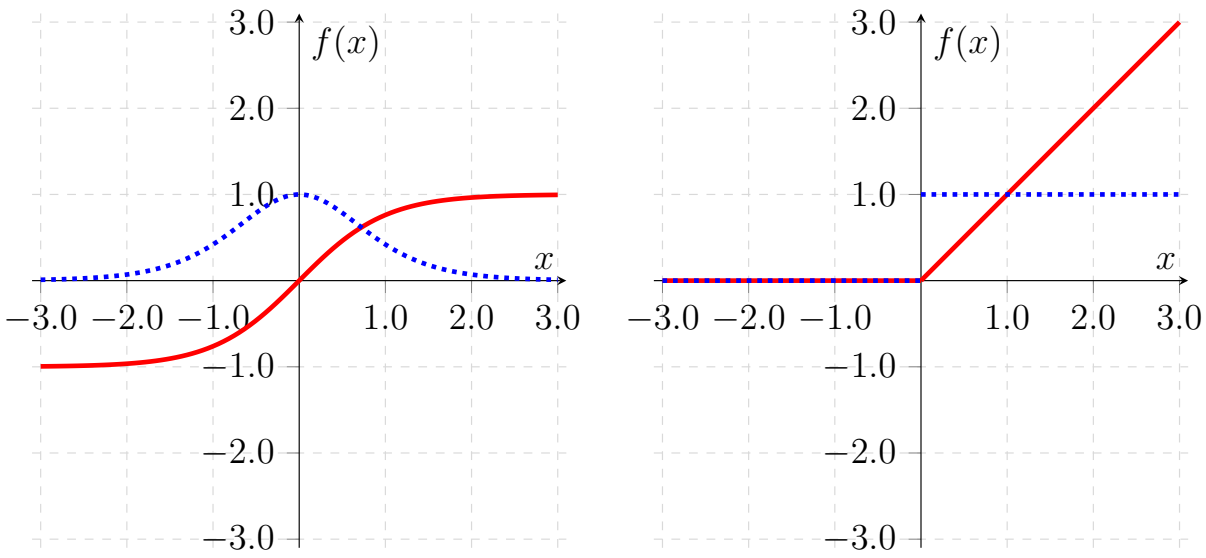
Consider two-layer neural networks consisting of $W^{(1)} \in \mathbb{R}$ for the first layer, $\mathbf{W}^{(2)} \in \mathbb{R}^{4\times 1}$ for the second, output layer. Since we are interested in function values with respect to two parameters $W^{(1)}$ and $W_1^{(2)}$ out of 5 parameters, $\mathbf{W}^{(2)}_{\{2,3,4\}}$ is set to a fixed value c. In this paper we use $c = 0$.[1] Figure 3.2 shows different shapes of the functions and slope steepness. In figure 3.2 both curves have similar shapes, but the curve for PWE has plateaus in which gradient descent can be very slow in comparison with the CE. Figure 3.2 shows that CE with ReLUs, which is explained the next section, has a very steep slope compared to CE with tanh. Such a slope can accelerate convergence speed in learning parameters using gradient descent. We conjecture that these properties might explain why our set-up converges faster than the other configurations, and BP-MLL performs poorly in most cases in our experiments.

3.2.5 Recent Advances in Deep Learning

In recent neural network and deep learning literature, a number of techniques were proposed to overcome the difficulty of learning neural networks efficiently. In particular, we make use of ReLUs, AdaGrad, and Dropout training, which are briefly discussed in the following.

Rectified Linear Units The most commonly used non-linear functions include the sigmoid and hyperbolic tangent (tanh) function. Once these functions are saturated during training,

[1] The shape of the functions is not changed even if we set c to arbitrary value since it is drawn by function values in z-axis with respect to only $W^{(1)}$ and $W_1^{(2)}$.



(a) Hyperbolic tangent function

(b) Rectified linear unit

Figure 3.3: Activation functions (solid and red lines) and their derivative (dotted and blue lines).

for instance, $\tanh(x)$ approaches either -1 or 1, the gradient of $f(x)$ with respect to input x vanishes.

To overcome the vanishing gradient problem, a ReLU has been proposed as a non-linear activation function on the hidden layer and shown to yield better generalization performance (Glorot et al., 2011; Nair and Hinton, 2010; Zeiler et al., 2013). More precisely, a ReLU is defined as

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

ReLUs disable negative activation so that the number of *active* parameters to be learned decreases at each gradient update step. Figure 3.3 compares tanh and ReLU. This sparsity characteristic makes ReLUs advantageous over the traditional activation units such as *sigmoid* and *tanh* in terms of the generalization performance.

Learning Rate Adaptation with AdaGrad *Stochastic gradient descent* (SGD) is a simple but effective technique for minimizing the objective functions of NNs. When SGD is considered as an optimization tool, one of the problems is the choice of the learning rate. A common approach is to estimate the learning rate which gives lower training errors on subsamples of training examples (LeCun et al., 2012) and then decrease it over time. Furhermore, to accelerate learning speed of SGD, one can utilize momentum (Rumelhart et al., 1986).

Instead of a fixed or scheduled learning rate, an *adaptive learning rate* method, namely AdaGrad, was proposed (Duchi et al., 2011). The method determines the learning rate at iteration τ by keeping previous gradients $\Delta_{1:\tau}$ to compute the learning rate for each dimension of parameters

$$\eta_{i,\tau} = \frac{\eta_0}{\sqrt{\sum_{t=1}^{\tau} \Delta_{i,t}^2}}$$

where i stands for an index of each dimension of parameters and η_0 is the initial learning rate and shared by all parameters. For multi-label learning, it is often the case that a few labels occur frequently, whereas the majority only occurs rarely, so that the rare ones need to be updated with larger steps in the direction of the gradient. If we use AdaGrad, the learning rates for the frequent labels decreases because the gradient of the parameter for the frequent labels will get smaller as the updates proceed. On the other hand, the learning rates for rare labels remain comparatively large.

Regularization using Dropout Training In principle, as the number of hidden layers and hidden units in a network increases, its expressive power also increases. If one is given a large number of training examples, training a larger networks will result in better performance than using a smaller one. The problem when training such a large network is that the model is more prone to getting stuck in local minima due to the huge number of parameters to learn. Dropout (Srivastava et al., 2014) is a technique for preventing overfitting in a huge parameter space. Its key idea is to decouple hidden units that activate the same output together, by randomly dropping some hidden units' activations as follows

$$h_j = \begin{cases} f_h\left(\sum_{i=1}^{D} W_{ji}^{(1)} x_i + b_j^{(1)}\right) & \text{if } r_j = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

where r denotes a binary random variable sampled from the Bernoulli distribution with dropout probability $1 - p$. Essentially, this corresponds to training an ensemble of networks with a subset of the parameters shared across all networks, and combining their predictions. However, the individual predictions of all possible hidden layers need not be computed and combined explicitly, but the output of the ensemble can be approximately reconstructed from the full network. Thus, dropout training has a similar regularization effect as ensemble techniques.

3.3 Experimental Setup

We have shown that why the structure of NNs needs to be reconsidered in the previous Sections. In this Section, we describe evaluation measures to show how effectively NNs perform by combining recent development in learning neural networks based on the fact that the *univariate* loss is consistent with respect to rank loss on large-scale textual datasets.

Datasets Our main interest is in large-scale text classification, for which we selected six representative domains, whose characteristics are summarized Table 3.1. For Reuters21578 we used the same training/test split as previous works (Yang and Gopal, 2012). Training and test data were switched for RCV1-v2 (Lewis et al., 2004) which originally consists of 23,149 train and 781,265 test documents. The EUR-Lex, Declicious and Bookmarks datasets were taken from the MULAN repository.[2] Except for Delicious and Bookmarks, all documents are represented with tf-idf features with cosine normalization such that length of the document vector is 1 in order to account for the different document lengths.

In addition to these standard benchmark datasets, we prepared a large-scale dataset from documents of the German Education Index (GEI).[3] The GEI is a database of links to more

[2] http://mulan.sourceforge.net/datasets.html
[3] http://www.dipf.de/en/portals/portals-educational-information/german-education-index

Table 3.1: Number of documents (D), size of vocabulary (D), total number of labels (L) and average number of labels per instance (C) for the six datasets used in our study.

Dataset	M	D	L	C
Reuters-21578	10789	18637	90	1.13
RCV1-v2	804414	47236	103	3.24
EUR-Lex	19348	5000	3993	5.31
Delicious	16105	500	983	19.02
Bookmarks	87856	2150	208	2.03
German Education Index	316061	20000	1000	7.16

than 800,000 scientific articles with metadata, e.g. title, authorship, language of an article and index terms. We consider a subset of the dataset consisting of approximately 300,000 documents which have abstract as well as the metadata. Each document has multiple index terms which are carefully hand-labelled by human experts with respect to the content of articles. We processed plain text by removing stopwords and stemming each token. To avoid the computational bottleneck from a large number of labels, we chose the 1,000 most common labels out of about 50,000. We then randomly split the dataset into 90% for training and 10% for test.

Algorithms Our main goal is to compare our NN-based approach to BP-MLL. NN_A stands for the single hidden layer neural networks which have *ReLUs* for its hidden layer and which are trained with SGD where each parameter of the neural networks has their own learning rate using *AdaGrad*. NN_{AD} additionally employs *Dropout* based on the same settings as NN_A. For both NN and BP-MLL, we used 1000 units in the hidden layer over all datasets. [4] As Dropout works well as a regularizer, no additional regularization to prevent overfitting was incorporated. The base learning rate η_0 was also determined among $[0.001, 0.01, 0.1]$ using validation data.

We also compared the NN-based algorithms to binary relevance (BR) using SVMs (Liblinear) as a base learner, as a representative of the state-of-the-art. The penalty parameter C was optimized in the range of $[10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}]$ based on either average of micro- and macro-average F_1 or rankloss on validation set. BR_B refers to linear SVMs where C is optimized with bipartition measures on the validation dataset. BR models whose penalty parameter is optimized on ranking measures are indicated as BR_R. In addition, we apply the same thresholding technique which we utilize in our NN approach (Section 3.2.3) on a ranked list produced by BR models (BR_R).

3.4 Results

We evaluate our proposed models and other baseline systems on datasets with varying statistics and characteristics. We first show experimental results that confirm that the techniques discussed in Section 3.2.5 actually contribute to an increased performance of NN-based multi-

[4] The optimal number of hidden units of BP-MLL and NN was tested among 20, 50, 100, 500 and 1000 on validation datasets. Usually, the more units are in the hidden layer, the better performance yield. We chose the number of units in terms of computational efficiency.

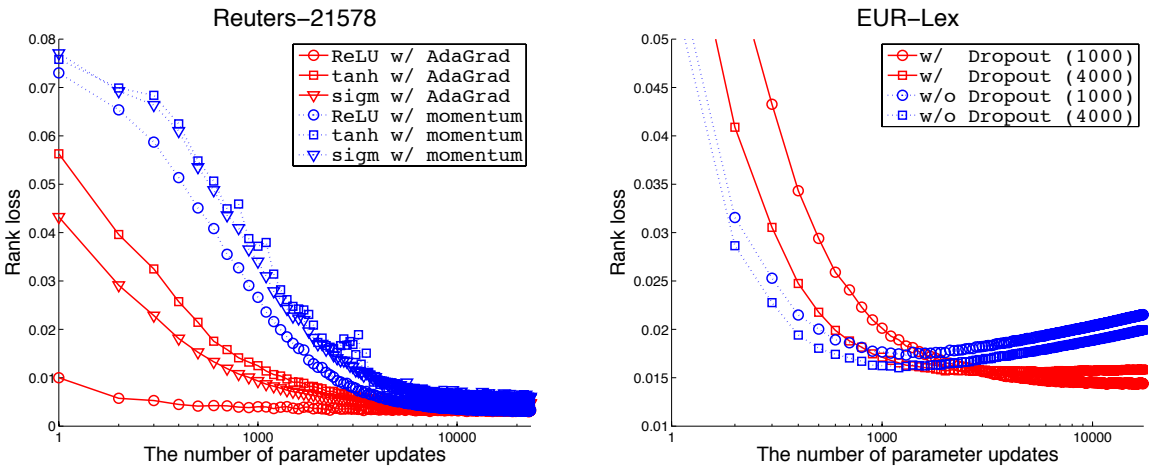


Figure 3.4: (*left*) effects of AdaGrad and momentum on three types of transfer functions in the hidden layers in terms of rank loss on Reuters-21578. (*right*) effects of dropout with two different numbers of hidden units in terms of rank loss on EUR-Lex.

label classification, and then compare all algorithms on the six above-mentioned datasets in order to get an overall impression of their performance.

Better Local Minima and Acceleration of Convergence Speed First we intend to show the effect of ReLUs and AdaGrad in terms of convergence speed and rank loss. The left part of Figure 3.4 shows that all three results of AdaGrad show a lower rank loss than all three versions of momentum. Moreover, within each group, ReLUs outperform the versions using tanh or sigmoid activation functions. That NNs with ReLUs at the hidden layer converge faster into a better weight space has been previously observed for the speech domain (Zeiler et al., 2013).[5] This faster convergence is a major advantage of combining recently proposed learning components such as ReLUs and AdaGrad, which facilitates a quicker learning of the parameters of NNs. This is particularly important for the large-scale text classification problems that are the main focus of this work.

Decorrelating Hidden Units While Output Units Remain Correlated One major goal of multi-label learners is to minimize rank loss by leveraging inherent correlations in a label space. However, we conjecture that these correlations also may cause overfitting because if groups of hidden units specialize in predicting particular label subsets that occur frequently in the training data, it will become harder to predict novel label combinations that only occur in the test set. Dropout effectively fights this by randomly dropping individual hidden units, so that it becomes harder for groups of hidden units to specialize in the prediction of particular output combinations, i.e., they decorrelate the hidden units, whereas the correlation of output units still remains. Particularly, a subset of output activations $\mathbf{o}$ and hidden activations $\mathbf{h}$ would be correlated through $\mathbf{W}^{(2)}$.

We observed overfitting across all datasets except for Reuters-21578 and RCV1-v2 under our experimental settings. The right part of Figure 3.4 shows how well Dropout prevents NNs from overfitting on the test data of EUR-Lex. In particular, we can see that with increasing numbers of parameter updates, the performance of regular NNs eventually got

[5] However, unlike the results of (Zeiler et al., 2013), in our preliminary experiments adding more hidden layers did not further improve generalization performance.

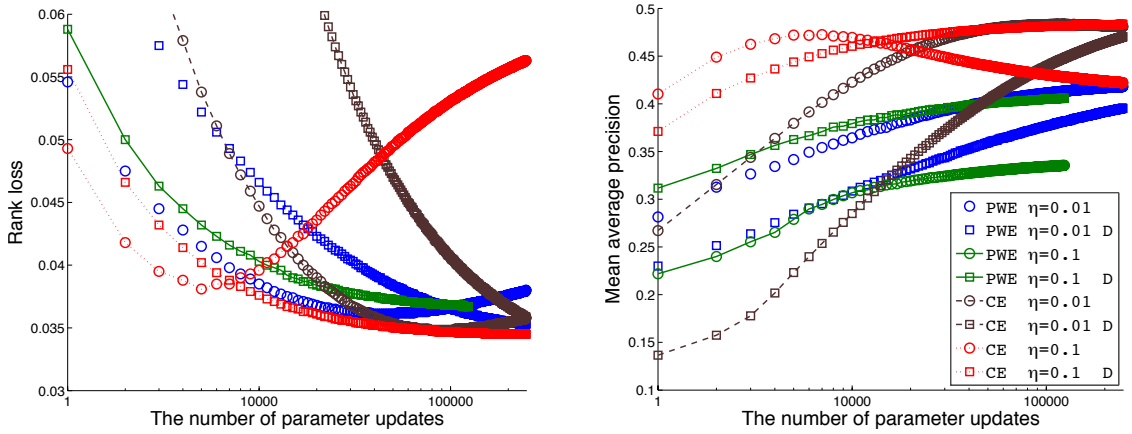


Figure 3.5: Rankloss (left) and mean average precision (right) on the German Education Index test-data for the different cost functions. η denotes the base learning rate and D indicates that Dropout is applied. Note that *x*-axis is in log scale.

worse in terms of rank loss. On the other hand, when dropout is employed, convergence is initially slower, but eventually effectively prevents overfitting.

Limiting Small Learning Rates in BP-MLL The learning rate strongly influences convergence and learning speed. (LeCun et al., 2012). As we have already seen in the Figure 3.2, the slope of PWE is less steep than CE, which implies that smaller learning rates should be used. Specifically, we observed PWE allows only smaller learning rate 0.01 (blue markers) in contrast with CE that works well a relatively larger learning rate 0.1 (red markers) in Figure 3.5. In the case of PWE with the larger learning rate (green markers), interestingly, dropout (rectangle markers in green) makes it converge towards much better local minima, yet it is still worse than the other configurations. It seems that the weights of BP-MLL oscillates in the vicinity of local minima and, indeed, converges *worse* local minima. However, it makes learning procedure of BP-MLL slow compared to NNs with CE making bigger steps for parameter updates.

With respect to dropout, Figure 3.5 also shows that for the same learning rates, networks without dropout converge much faster than ones working with Dropout in terms of both rank loss and MAP. Regardless of the cost functions, overfitting arises over the networks without dropout and it is likely that overfitting is avoided effectively as discussed earlier.

Comparison of Algorithms Table 3.3 shows detailed results of all experiments with all algorithms on all six datasets, except that we could not obtain results of BP-MLL on EUR-Lex within a reasonable time frame. In an attempt to summarize the results, table 3.2 shows the average rank of each algorithm in these six datasets according to all ranking an bipartition measures.

We can see that although BP-MLL focuses on minimizing pairwise ranking errors, thereby capturing label dependencies, the single hidden layer NNs with cross-entropy minimization (i.e., NN_A and NN_{AD}) work much better not only on rank loss but also on other ranking measures. The binary relevance (BR) approaches show acceptable performance on ranking

[5] A trajectory for PWE $\eta = 0.1$ is missing in the figure because it got 0.2 on the rankloss measure which is much worse than the other configurations.

Table 3.2: Average ranks of the algorithms on ranking and bipartition measures.

	Ranking measures			
	rankloss	oneError	Coverage	MAP
NN_A	2.2	2.4	2.6	2.2
NN_{AD}	**1.2**	**1.4**	**1.2**	**1.6**
$BP\text{-}MLL_{TA}$	5.9	6.9	6.2	6.2
$BP\text{-}MLL_{TAD}$	5	5.7	5	5.7
$BP\text{-}MLL_{RA}$	5.2	7	5.4	6.6
$BP\text{-}MLL_{RAD}$	3.1	6.3	3	5.8
BR_B	7.4	3.3	6.9	4.3
BR_R	6	3	5.7	3.6

	Bipartition measures					
	miP	miR	miF	maP	maR	maF
NN_A	**2**	6	2.4	**1.8**	5.6	**2**
NN_{AD}	**2**	5.8	**1.8**	2	5.6	2.2
$BP\text{-}MLL_{TA}$	6.6	3.6	6.6	6.8	3	6.2
$BP\text{-}MLL_{TAD}$	7	3.6	7	7	4	7.2
$BP\text{-}MLL_{RA}$	5.6	**2.8**	5	5	**2.8**	4.2
$BP\text{-}MLL_{RAD}$	6	**2.8**	5.8	5.6	3.6	5.6
BR_B	3.2	6.8	4.6	4.4	6.8	5.6
BR_R	3.6	4.6	2.8	3.4	4.6	3

measures even though label dependency was ignored during the training phase. In addition, NN_A and NN_{AD} perform as good as or better than other methods on bipartition measures as well as on ranking measures.

We did not observe significant improvements by replacing hidden units of BP-MLL from tanh to ReLU. However, if we change the cost function in the previous setup from PWE to CE, significant improvements were obtained. Because $BP\text{-}MLL_{RAD}$ is the same architecture as NN_{AD} except for its cost function,[6] we can say that the differences in the effectiveness of NNs and BP-MLL are due to the use of different cost functions. This also implies that the main source of improvements for NNs against BP-MLL is replacement of the cost function. Again, Figure 3.5 shows the difference between two cost functions more explicitly.

3.5 Conclusion

This chapter presented a multi-label classification framework based on a neural network and a simple threshold label predictor. We found that our approach outperforms the state-of-the-art, BP-MLL, both in predictive performance as well as in computational complexity

[6] For PWE we use *tanh* in the output layer, but *sigmoid* is used for CE because predictions $\mathbf{o}$ for computing CE with targets $\mathbf{y}$ needs to be between 0 and 1.

Table 3.3: Results on ranking and bipartition measures. Results for BP-MLL on EUR-Lex are missing because the runs could not be completed in a reasonably short time.

Eval. measures	Ranking				Bipartition					
	rankloss	oneError	Coverage	MAP	miP	miR	miF	maP	maR	maF
Reuters-21578										
NN_A	0.0037	0.0706	0.7473	0.9484	0.8986	0.8357	0.8660	**0.6439**	0.4424	0.4996
NN_{AD}	**0.0031**	0.0689	**0.6611**	0.9499	0.9042	0.8344	0.8679	0.6150	0.4420	0.4956
$BP\text{-}MLL_{TA}$	0.0054	0.0808	1.0987	0.9431	0.8205	0.8582	0.8389	0.5303	0.4364	0.4624
$BP\text{-}MLL_{TAD}$	0.0063	0.0719	1.2037	0.9476	0.8421	0.8416	0.8418	0.5510	0.4292	0.4629
$BP\text{-}MLL_{RA}$	0.0039	0.0868	0.8238	0.9400	0.7876	0.8616	0.8230	0.5609	**0.4761**	0.4939
$BP\text{-}MLL_{RAD}$	0.0039	0.0808	0.8119	0.9434	0.7945	0.8654	0.8284	0.5459	0.4685	0.4831
BR_B	0.0040	**0.0613**	0.8092	**0.9550**	**0.9300**	0.8096	0.8656	0.6050	0.3806	0.4455
BR_R	0.0040	**0.0613**	0.8092	**0.9550**	0.8982	**0.8603**	**0.8789**	0.6396	0.4744	**0.5213**
RCV1-v2										
NN_A	0.0040	0.0218	3.1564	0.9491	0.9017	0.7836	0.8385	0.7671	0.5760	0.6457
NN_{AD}	**0.0038**	**0.0212**	**3.1108**	**0.9500**	**0.9075**	0.7813	0.8397	**0.7842**	0.5626	0.6404
$BP\text{-}MLL_{TA}$	0.0058	0.0349	3.7570	0.9373	0.6685	0.7695	0.7154	0.4385	0.5803	0.4855
$BP\text{-}MLL_{TAD}$	0.0057	0.0332	3.6917	0.9375	0.6347	0.7497	0.6874	0.3961	0.5676	0.4483
$BP\text{-}MLL_{RA}$	0.0058	0.0393	3.6730	0.9330	0.7712	0.8074	0.7889	0.5741	0.6007	0.5823
$BP\text{-}MLL_{RAD}$	0.0056	0.0378	3.6032	0.9345	0.7612	0.8016	0.7809	0.5755	0.5748	0.5694
BR_B	0.0061	0.0301	3.8073	0.9375	0.8857	0.8232	**0.8533**	0.7654	0.6342	0.6842
BR_R	0.0051	0.0287	3.4998	0.9420	0.8156	**0.8822**	0.8476	0.6961	**0.7112**	**0.6923**
EUR-Lex										
NN_A	0.0195	0.2016	310.6202	0.5975	0.6346	0.4722	0.5415	0.3847	0.3115	0.3256
NN_{AD}	**0.0164**	**0.1681**	**269.4534**	**0.6433**	**0.7124**	0.4823	**0.5752**	**0.4470**	0.3427	0.3687
BR_B	0.0642	0.1918	976.2550	0.6114	0.6124	0.4945	0.5471	0.4260	**0.3643**	**0.3752**
BR_R	0.0204	0.2088	334.6172	0.5922	0.0329	**0.5134**	0.0619	0.2323	0.3063	0.2331
German Education Index										
NN_A	**0.0350**	0.2968	138.5423	**0.4828**	0.4499	0.4200	**0.4345**	0.4110	0.3132	**0.3427**
NN_{AD}	0.0352	**0.2963**	138.3590	0.4797	0.4155	0.4472	0.4308	0.3822	0.3216	0.3305
$BP\text{-}MLL_{TA}$	0.0386	0.8309	150.8065	0.3432	0.1502	**0.6758**	0.2458	0.1507	**0.5562**	0.2229
$BP\text{-}MLL_{TAD}$	0.0371	0.7591	139.1062	0.3281	0.1192	0.5056	0.1930	0.1079	0.4276	0.1632
$BP\text{-}MLL_{RA}$	0.0369	0.4221	143.4541	0.4133	0.2618	0.4909	0.3415	0.3032	0.3425	0.2878
$BP\text{-}MLL_{RAD}$	0.0353	0.4522	**135.1398**	0.3953	0.2400	0.5026	0.3248	0.2793	0.3520	0.2767
BR_B	0.0572	0.3052	221.0968	0.4533	**0.5141**	0.2318	0.3195	0.3913	0.1716	0.2319
BR_R	0.0434	0.3021	176.6349	0.4755	0.4421	0.3997	0.4199	**0.4361**	0.2706	0.3097
Delicious										
NN_A	0.0860	0.3149	396.4659	0.4015	**0.3637**	0.4099	0.3854	0.2488	0.1721	0.1772
NN_{AD}	**0.0836**	**0.3127**	**389.9422**	**0.4075**	0.3617	0.4399	**0.3970**	**0.2821**	0.1777	**0.1824**
$BP\text{-}MLL_{TA}$	0.0953	0.4967	434.8601	0.3288	0.1829	0.5857	0.2787	0.1220	0.2728	0.1572
$BP\text{-}MLL_{TAD}$	0.0898	0.4358	418.3618	0.3359	0.1874	0.5884	0.2806	0.1315	0.2427	0.1518
$BP\text{-}MLL_{RA}$	0.0964	0.6157	427.0468	0.2793	0.2070	**0.5894**	0.3064	0.1479	**0.2609**	0.1699
$BP\text{-}MLL_{RAD}$	0.0894	0.6060	411.5633	0.2854	0.2113	0.5495	0.3052	0.1650	0.2245	0.1567
BR_B	0.1184	0.4355	496.7444	0.3371	0.1752	0.2692	0.2123	0.0749	0.1336	0.0901
BR_R	0.1184	0.4358	496.8180	0.3371	0.2559	0.3561	0.2978	0.1000	0.1485	0.1152
Bookmarks										
NN_A	0.0663	0.4924	22.1183	0.5323	0.3919	0.3907	0.3913	0.3564	0.3069	0.3149
NN_{AD}	**0.0629**	**0.4828**	**20.9938**	**0.5423**	**0.3929**	0.3996	**0.3962**	**0.3664**	0.3149	**0.3222**
$BP\text{-}MLL_{TA}$	0.0684	0.5598	23.0362	0.4922	0.0943	0.5682	0.1617	0.1115	0.4743	0.1677
$BP\text{-}MLL_{TAD}$	0.0647	0.5574	21.7949	0.4911	0.0775	**0.6096**	0.1375	0.0874	**0.5144**	0.1414
$BP\text{-}MLL_{RA}$	0.0707	0.5428	23.6088	0.5049	0.1153	0.5389	0.1899	0.1235	0.4373	0.1808
$BP\text{-}MLL_{RAD}$	0.0638	0.5322	21.5108	0.5131	0.0938	0.5779	0.1615	0.1061	0.4785	0.1631
BR_B	0.0913	0.5318	29.6537	0.4868	0.2821	0.2546	0.2676	0.1950	0.1880	0.1877
BR_R	0.0895	0.5305	28.7233	0.4889	0.2525	0.4049	0.3110	0.2259	0.3126	0.2569

and convergence speed. We have explored why BP-MLL does not perform well as a multi-label text classifier. Our experimental results showed the proposed framework is an effective method for the multi-label text classification task. Also, we have conducted extensive analysis to characterize the effectiveness of combining ReLUs with AdaGrad for fast convergence rate, and utilizing Dropout to prevent overfitting which results in better generalization.

4 Estimating Joint Probabilities of Label Subsets using Label Sequences

4.1 Introduction

In the previous chapter, we have shown that *feed-forward neural networks* (FNNs) with cross-entropy loss (Eq. (3.8)) perform better than those with pairwise loss (Eq. (3.3)) in particular in terms of the rank loss because univariate surrogate loss functions including the cross-entropy loss are rather consistent with the rank loss (Dembczyński et al., 2012a). FNNs with the cross-entropy loss do not explicitly exploit the label dependence since prediction errors are measured by the binary cross-entropy for each label independently. As discussed in Section 2.2, classifiers independently learned per label could yield optimal predictions for each label, but a collection of the optimal predictions does not necessarily equal to the optimal prediction over all possible label combinations.

If we consider *multi-label classification* (MLC) as a problem of assigning a subset of labels out of all possible label subsets to a given instance, MLC can be seen as multi-class classification with extremely large label spaces, which referred to as *label powerset* (LP). In fact, each label subset as a target has a smaller number of associated training instances because we now have an increasing number of labels to take into account while the size of data to train classifiers remains intact. In other words, the data scarcity problem arises much more severely when LP is considered for MLC.

Albeit its poor scalability, it is worth noting that LP is consistent with the subset $0/1$ loss, which is the most strict evaluation measure in MLC. The subset $0/1$ loss favors algorithms that yield predictions exactly, so that predictions that contain just a single mistake are treated equally as entirely incorrect predictions. Also note that the subset $0/1$ loss biases MLC methods towards the mode of the target distribution, as does the $0/1$ loss in binary classification. In our case, the target distribution is the joint probability distribution of labels conditioned on instances.

As aforementioned, such a problem transformation makes it rather difficult to build effective MLC systems. Read et al. (2009) have proposed *classifier chains* (CCs) that creates a chain of L binary classifiers, each of which learns to predict a single label given ground truths for preceding labels in the chain. In fact, CC decomposes the original problem into multiple sub-problems that can be solved in an efficient way. *Probabilistic classifier chain* (PCC) (Dembczyński et al., 2010) is an extension of CC interpreted in a probabilistic point of view that allows us to seek the mode of the estimated target distribution but with higher computational cost. Training multiple binary classifiers independently can be though of as a key advantage of PCCs during training because we can reduce the training time by parallelizing the training process over multiple devices. On the other hand, PCC restricts the classifiers not to share information across them during training and each classifier has its own parameters. As PCC trains independent classifiers mapping instances and a partial label

sequence to each label over an arbitrary sequence of labels, it can be seen as an efficient learning framework for maximizing the joint probability of sequences when the length of sequences is fixed, i.e., L. Given such a setting, one may solve MLC problems by sequence learning methods.

Recently, *recurrent neural networks* (RNNs) have been successfully applied to several sequence learning tasks. Instead of building multiple independent classifiers, RNNs use the same set of parameters to predict all labels through the chain of labels. Parameter sharing across all classifiers allows to better exploit information of previous decisions. In this chapter, we present several RNNs for MLC and discuss key advantages of RNNs compared to traditional MLC approaches for maximizing subset accuracy. Moreover, as both, CCs and RNNs depend on a fixed ordering of the labels, which is typically not part of a multi-label problem specification, we also compare different ways of ordering the label set, and give some recommendations on suitable ordering strategies.

4.2 Learning to Predict Subsets as Sequence Prediction

We have discussed LP and PCC as a means of subset accuracy maximization. More precisely, LP defines a set of all possible label combinations $\mathcal{S}_L = \{\{1\}, \{2\}, \cdots, \{1, 2, \cdots, L\}\}$, from which a new class label is assigned to each label subset consisting of *positive* labels in $\mathcal{D}$. LP, then, addresses MLC as a multi-class classification problem with $\min(N, 2^L)$ possible labels such that

$$P(y_1, y_2, \cdots, y_L|\boldsymbol{x}) \xrightarrow{LP} P(y_{\text{LP}} = k|\boldsymbol{x}) \tag{4.1}$$

where $k = 1, 2, \cdots, \min(N, 2^L)$. In contrast to LP, PCC decomposes the joint probability into L conditional probabilities:

$$P(y_1, y_2, \cdots, y_L|\boldsymbol{x}) = \prod_{i=1}^{L} P(y_i|\boldsymbol{y}_{<i}, \boldsymbol{x}) \tag{4.2}$$

where $\boldsymbol{y}_{<i} = \{y_1, \cdots, y_{i-1}\}$ denotes a set of labels that precede a label y_i in computing conditional probabilities, and $\boldsymbol{y}_{<i} = \emptyset$ if $i = 1$. As discussed in Section 2.2.3, both LP and PCC have drawbacks when L is large.

Note that y_{LP} in Eq. (4.1) denotes a set of positive labels. Instead of solving Eq. (4.1) using a multi-class classifier, one can consider predicting all labels individually in y_{LP}, and interpret this approach as a way of maximizing the joint probability of a label subset given the number of labels T in the subset. Similar to PCC Eq. (4.2), the joint probability can be computed as product of conditional probabilities, but unlike PCC, only $T \ll L$ terms are needed. Therefore, maximizing the joint probability of *positive* labels can be viewed as subset accuracy maximization such as LP in a sequential manner as the way PCC works. To be more precise, $\boldsymbol{y}$ can be represented as a set of 1-of-L vectors such that $\boldsymbol{y} = \{\mathbf{y}_{p_i}\}_{i=1}^{T}$ and $\mathbf{y}_{p_i} \in \mathbb{R}^L$ where T is the number of positive labels associated with an instance $\boldsymbol{x}$. The joint probability of *positive* labels can be written as

$$P(\mathbf{y}_{p_1}, \mathbf{y}_{p_2}, \cdots, \mathbf{y}_{p_T}|\boldsymbol{x}) = \prod_{i=1}^{T} P(\mathbf{y}_{p_i}|\boldsymbol{y}_{<p_i}, \boldsymbol{x}). \tag{4.3}$$

Note that Eq. (4.3) has the same form as Eq. (4.2) except for the number of output variables. While Eq. (4.2) is meant to maximize the joint probability over the entire 2^L configurations, Eq. (4.3) represents the probability of sets of positive labels and ignores negative labels. The subscript p is omitted unless it is needed for clarity. A key advantage of Eq. (4.3) over the traditional multi-label formulation is that the number of conditional probabilities to be estimated is dramatically reduced from L to T, improving scalability. Also note that each estimate itself again depends on the previous estimates. Reducing the length of the chain might be helpful in reducing the cascading errors, which is particularly relevant for labels at the end of the chain. Having said that, computations over the L^T search space of Eq. (4.3) remain infeasible even though our search space is much smaller than the search space of PCC in Eq. (4.2), 2^L, since the label cardinality C is usually very small, i.e., $C \ll L$.

As each instance has a different value for T, we need MLC methods capable of dealing with a different number of output targets across instances. In fact, the idea of predicting positive labels only has been explored for MLC. RNNs have been successful in solving complex output space problems. In particular, Wang et al. (2016) have demonstrated that RNNs provide a competitive solution on MLC image datasets. Doppa et al. (2014) propose *multi-label search* where a heuristic function and cost function are learned to iteratively search for elements to be chosen as positive labels on a binary vector of size L. In this work, we make use of RNNs to compute $\prod_{i=1}^{T} P(\mathbf{y}_{p_i}|\boldsymbol{y}_{<p_i}, \boldsymbol{x})$ for which the order of labels in a label subset $\mathbf{y}_{p_1}, \mathbf{y}_{p_2}, \cdots, \mathbf{y}_{p_T}$ need to be determined a priori, as in PCC. In the following, we explain possible ways of choosing label permutations, and then present three RNN architectures for MLC.

4.2.1 Determining Label Permutations

We hypothesize that some label permutations make it easier to estimate Eqs. (4.2) and (4.3) than others. However, as no ground truth such as relevance scores of each positive label to a training instance is given, we need to make the way to prepare fixed label permutations during training.

The most straightforward approach is to order positive labels by frequency simply either in a descending (from frequent to rare labels) or an ascending (from rare to frequent ones) order. Although this type of label permutation may break down label correlations in a chain, Wang et al. (2016) have shown that the descending label ordering allows to achieve a decent performance on multi-label image datasets. As an alternative, if additional information such as label hierarchies is available about the labels, we can also take advantage of such information to determine label permutations. For example, assuming that labels are organized in a *directed acyclic graph* (DAG) where labels are partially ordered, we can obtain a total order of labels by topological sorting with *depth-first search* (DFS), and given that order, target labels in the training set can be sorted in a way that labels that have same ancestors in the graph are placed next to each other. In fact, this approach also preserves partial label orders in terms of the co-occurrence frequency of a child and its parent label in the graph.

4.2.2 Label Sequence Prediction from Given Label Permutations

A *recurrent neural network* (RNN) is a *neural network* (NN) that is able to capture temporal information. RNNs have shown their superior performance on a wide range of applications

where target outputs form a sequence. In our context, we can expect that MLC will also benefit from the reformulation of PCCs because the estimation of the joint probability of only positive labels as in Eq. (4.3) significantly reduces the length of the chains, thereby reducing the effect of error propagation.

A RNN architecture that learns a sequence of L *binary* targets can be seen as a NN counterpart of PCC because its objective is to maximize Eq. (4.2), just like in PCC. We will refer to this architecture as RNN^b (Fig. 4.1b). One can also come up with a RNN architecture maximizing Eq. (4.3) to take advantage of the smaller label subset size T than L, which shall be referred to as RNN^m (Fig. 4.1c). For learning RNNs, we use *gated recurrent units* (GRUs) which allow to effectively avoid the vanishing gradient problem (Cho et al., 2014). Let $\bar{\mathbf{x}}$ be the fixed input representation computed from an instance $\boldsymbol{x}$. We shall explain how to determine $\bar{\mathbf{x}}$ in Sec. 4.3.2. Given an initial state $\mathbf{h}_0 = f_{\text{init}}(\bar{\mathbf{x}})$, at each step i, both RNN^b and RNN^m compute a hidden state $\mathbf{h}_i$ by taking $\bar{\mathbf{x}}$ and a target (or predicted) label from the previous step as inputs: $\mathbf{h}_i = \text{GRU}\left(\mathbf{h}_{i-1}, \mathbf{V}_{y_{i-1}}, \bar{\mathbf{x}}\right)$ for RNN^b and $\mathbf{h}_i = \text{GRU}\left(\mathbf{h}_{i-1}, \mathbf{V}\mathbf{y}_{p_{i-1}}, \bar{\mathbf{x}}\right)$ for RNN^m where $\mathbf{V}$ is the matrix of d-dimensional label embeddings. In turn, RNN^b computes the conditional probabilities $P_\theta\left(y_i|\boldsymbol{y}_{<i}, \boldsymbol{x}\right)$ in Eq. (4.2) by $f\left(\mathbf{h}_i, \mathbf{V}_{y_{i-1}}, \bar{\mathbf{x}}\right)$ consisting of linear projection, followed by the softmax function. Likewise, we consider $f\left(\mathbf{h}_i, \mathbf{V}\mathbf{y}_{i-1}, \bar{\mathbf{x}}\right)$ for RNN^m. Note that the key difference between RNN^b and RNN^m is whether target labels are binary targets y_i or 1-of-L targets $\mathbf{y}_i$. Under the assumption that the hidden states $\mathbf{h}_i$ preserve the information on all previous labels $\boldsymbol{y}_{<i}$, learning RNN^b and RNN^m can be interpreted as learning classifiers in a chain. Whereas in PCCs an independent classifier is responsible for predicting each label, both proposed types of RNNs maintain a single set of parameters to predict all labels.

The input representations $\bar{\mathbf{x}}$ to both RNN^b and RNN^m are kept fixed after the preprocessing of inputs $\boldsymbol{x}$ is completed. Recently, an *encoder-decoder* (EncDec) framework, also known as *sequence-to-sequence* (Seq2Seq) learning (Cho et al., 2014; Sutskever et al., 2014), has drawn attention to modeling both input and output sequences, and has been applied successfully to various applications in natural language processing and computer vision (Donahue et al., 2015; Kumar et al., 2016). EncDec is composed of two RNNs: an encoder network captures the information in the entire input sequence, which is then passed to a decoder network which decodes this information into a sequence of labels (Fig. 4.1d). In contrast to RNN^b and RNN^m, which only use fixed input representations $\bar{\mathbf{x}}$, EncDec makes use of context-sensitive input vectors from $\boldsymbol{x}$. We describe how EncDec computes Eq. (4.3) in the following.

Encoder. An encoder takes $\boldsymbol{x}$ and produces a *sequence* of D-dimensional vectors $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_E\}$ where E is the number of encoded vectors for a single instance. In this work, we consider *documents* as input data. For encoding documents, we use words as atomic units. Consider a document as a sequence of E words such that $\boldsymbol{x} = \{w_1, w_2, \cdots, w_E\}$ and a vocabulary of $\mathcal{V}$ words. Each word $w_j \in \mathcal{V}$ has its own K-dimensional vector representation $\mathbf{u}_j$. The set of these vectors constitutes a matrix of word embeddings defined as $\mathbf{U} \in \mathbb{R}^{K \times |\mathcal{V}|}$. Given this word embedding matrix $\mathbf{U}$, words in a document are converted to a sequence of K-dimensional vectors $\boldsymbol{u} = \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_E\}$, which is then fed into the RNN to learn the sequential structures in a document

$$\mathbf{x}_j = \text{GRU}(\mathbf{x}_{j-1}, \mathbf{u}_j) \tag{4.4}$$

where $\mathbf{x}_0$ is the zero vector.

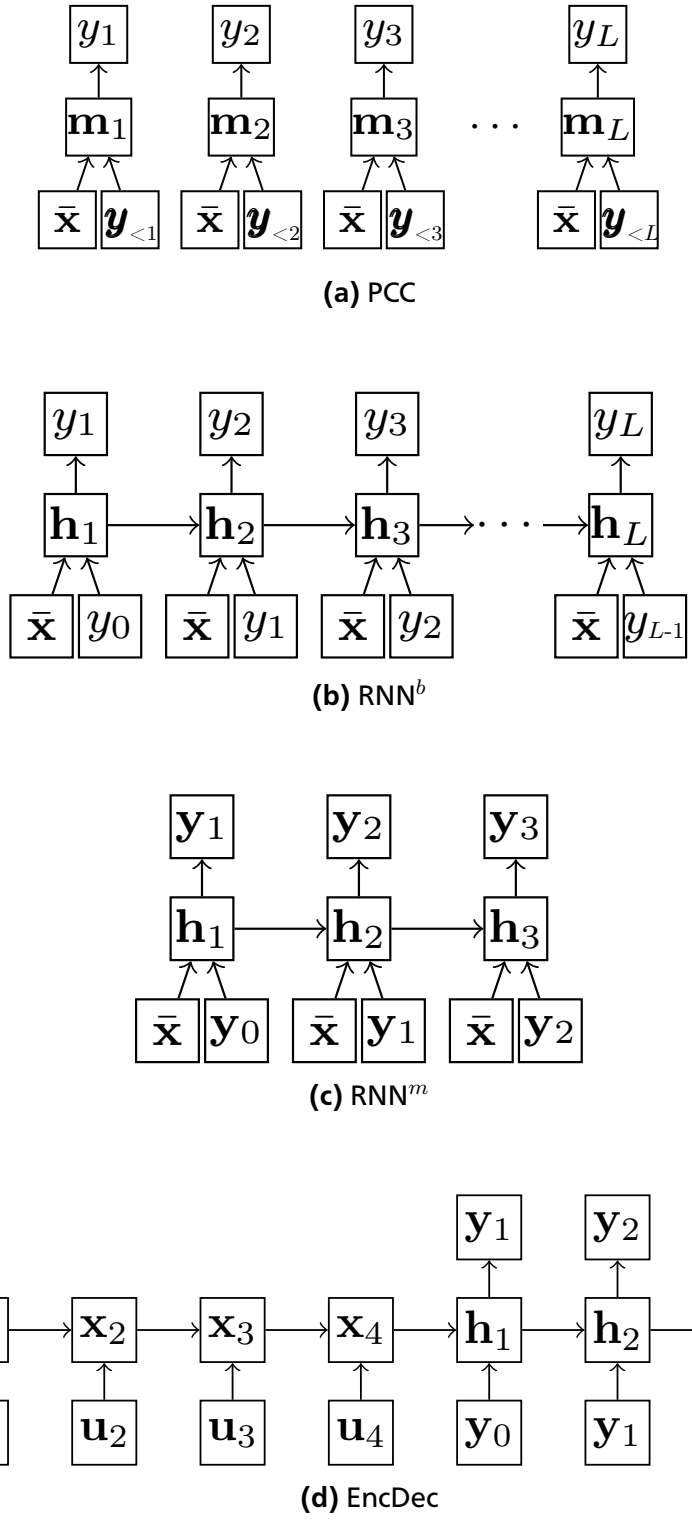


Figure 4.1: Illustration of PCC and RNN architectures for MLC. For the purpose of illustration, we assume $T = 3$ and $\boldsymbol{x}$ consists of 4 elements.

Decoder. After the encoder computes $\mathbf{x}_i$ for all elements in $\boldsymbol{x}$, we set the initial hidden state of the decoder $\mathbf{h}_0 = f_{\text{init}}(\mathbf{x}_E)$, and then compute hidden states $\mathbf{h}_i = \text{GRU}\left(\mathbf{h}_{i-1}, \mathbf{V}\mathbf{y}_{i-1}, \mathbf{c}_i\right)$ where $\mathbf{c}_i = \sum_j \alpha_{ij}\mathbf{x}_j$ is the context vector which is the sum of the encoded input vectors weighted by attention scores $\alpha_{ij} = f_{att}\left(\mathbf{h}_{i-1}, \mathbf{x}_j\right), \alpha_{ij} \in \mathbb{R}$. Then, as shown in Bahdanau

Table 4.1: Comparison of the three RNN architectures for MLC.

	RNN^b	RNN^m	EncDec
hidden states	$\text{GRU}\left(\mathbf{h}_{i-1}, \mathbf{V}_{y_{i-1}}, \bar{\mathbf{x}}\right)$	$\text{GRU}\left(\mathbf{h}_{i-1}, \mathbf{V}\mathbf{y}_{i-1}, \bar{\mathbf{x}}\right)$	$\text{GRU}\left(\mathbf{h}_{i-1}, \mathbf{V}\mathbf{y}_{i-1}, \mathbf{c}_i\right)$
prob. of output labels	$f\left(\mathbf{h}_i, \mathbf{V}_{y_{i-1}}, \bar{\mathbf{x}}\right)$	$f\left(\mathbf{h}_i, \mathbf{V}\mathbf{y}_{i-1}, \bar{\mathbf{x}}\right)$	$f\left(\mathbf{h}_i, \mathbf{V}\mathbf{y}_{i-1}, \mathbf{c}_i\right)$

et al. (2015), the conditional probability $P_\theta(\mathbf{y}_i|\boldsymbol{y}_{<i}, \boldsymbol{x})$ for predicting a label $\mathbf{y}_i$ can be estimated by a function of the hidden state $\mathbf{h}_i$, the previous label $\mathbf{y}_{i-1}$ and the context vector $\mathbf{c}_i$:

$$P_\theta(\mathbf{y}_i|\boldsymbol{y}_{<i}, \boldsymbol{x}) = f(\mathbf{h}_i, \mathbf{V}\mathbf{y}_{i-1}, \mathbf{c}_i). \tag{4.5}$$

Indeed, EncDec is potentially more powerful than RNN^b and RNN^m because each prediction is determined based on the dynamic context of the input $\boldsymbol{x}$ unlike the fixed input representation $\bar{\mathbf{x}}$ used in PCC, RNN^b and RNN^m (cf. Figs. 4.1a to 4.1d). The differences in computing hidden states and conditional probabilities among the three RNNs are summarized in table 4.1.

Unlike in the training phase, where we know the size of positive label set T, this information is not available during prediction. Whereas this is typically solved using a meta learner that predicts a threshold in the ranking of labels, EncDec follows a similar approach as Fürnkranz et al. (2008) and directly predicts a virtual label that indicates the end of the sequence.

4.3 Experimental Setup

In order to see whether solving MLC problems using RNNs can be a good alternative to CC-based approaches, we will compare traditional multi-label learning algorithms such as *binary relevance* (BR) and PCCs with the RNN architectures (Fig. 4.1) on multi-label text classification datasets. For a fair comparison, we will use the same fixed label permutation strategies in all compared approaches if necessary. As it has already been demonstrated in the literature that label permutations may affect the performance of classifier chain approaches (Kumar et al., 2013; Read et al., 2011), we will evaluate a few different strategies.

4.3.1 Baselines and Training Details

We use feed-forward NNs as a base learner of BR, LP and PCC. For PCC, beam search with beam size of 5 is used at inference time (Kumar et al., 2013). As another NN baseline, we also consider a feed-forward NN with binary cross entropy per label (Nam et al., 2014). We compare RNNs to FastXML (Prabhu and Varma, 2014), one of state-of-the-arts in extreme MLC.[1] All NN based approaches are trained by using *Adam* (Kingma and Ba, 2015) and dropout (Srivastava et al., 2014). The dimensionality of hidden states of all the NN baselines as well as the RNNs is set to 1024. The size of label embedding vectors is set to 256. We used the NVIDIA Titan X to train NN models including RNNs and base learners. For FastXML, a machine with 64 cores and 1024GB memory was used.

[1] Note that as FastXML optimizes top-k ranking of labels unlike our approaches and assigns a confidence score for each label. We set a threshold of 0.5 to convert rankings of labels into bipartition predictions.

Table 4.2: Summary of datasets. # training documents (N_{tr}), # test documents (N_{ts}), # labels (L), label cardinality (C), # label combinations (LC), type of label structure (HS).

Dataset	N_{tr}	N_{ts}	L	C	LC	HS
Reuters-21578	7770	3019	90	1.24	468	-
RCV1-v2	781 261	23 149	103	3.21	14 921	Tree
BioASQ	11 431 049	274 675	26 970	12.60	11 673 800	DAG

4.3.2 Datasets and Preprocessing

We use multi-label text classification datasets for which we had access to the full text as it is required for our approach EncDec, namely Reuters-21578,[2] RCV1-v2 (Lewis et al., 2004) and BioASQ,[3] each of which has different properties. Summary statistics of the datasets are given in Table 4.2. For preparing the train and the test set of Reuters-21578 and RCV1-v2, we follow Nam et al. (2014). We split instances in BioASQ by year 2014, so that all documents published in 2014 and 2015 belong to the test set. For tuning hyperparameters, we set aside 10% of the training instances as the validation set for both Reuters-21578 and RCV1-v2, but chose randomly 50 000 documents for BioASQ.

The RCV1-v2 and BioASQ datasets provide label relationships as a graph. Specifically, labels in RCV1-v2 are structured in a tree. The label structure in BioASQ is a directed graph and contains cycles. We removed all edges pointing to nodes which have been already visited while traversing the graph using DFS, which results in a DAG of labels.

Document Representations. For all datasets, we replaced numbers with a special token and then build a word vocabulary for each data set. The sizes of the vocabularies for Reuters-21578, RCV1-v2 and BioASQ are 22 747, 50 000 and 30 000, respectively. *Out-of-vocabulary* (OOV) words were also replaced with a special token and we truncated the documents after 300 words.[4]

We trained *word2vec* (Mikolov et al., 2013a) on an English Wikipedia dump to get 512-dimensional word embeddings $\mathbf{u}$. Given the word embeddings, we created the fixed input representations $\bar{\mathbf{x}}$ to be used for all of the baselines in the following way: Each word in the document except for numbers and OOV words is converted into its corresponding embedding vector, and these word vectors are then averaged, resulting in a document vector $\bar{\mathbf{x}}$. For EncDec, which learns hidden states of word sequences using an encoder RNN, all words are converted to vectors using the pre-trained word embeddings and we feed these vectors as inputs to the encoder. In this case, unlike during the preparation of $\bar{\mathbf{x}}$, we do not ignore OOV words and numbers. Instead, we initialize the vectors for those tokens randomly. For a fair comparison, we do not update word embeddings of the encoder in EncDec.

4.4 Experimental Results

In the following, we show results of various versions of RNNs for MLC on three text datasets which span a wide variety of input and label set sizes. We also evaluate different label

[2] http://www.daviddlewis.com/resources/testcollections/reuters21578/

[3] http://bioasq.org

[4] By the truncation, one may worry about the possibility of missing information related to some specific labels. As the average length of documents in the datasets is below 300, the effect would be negligible.

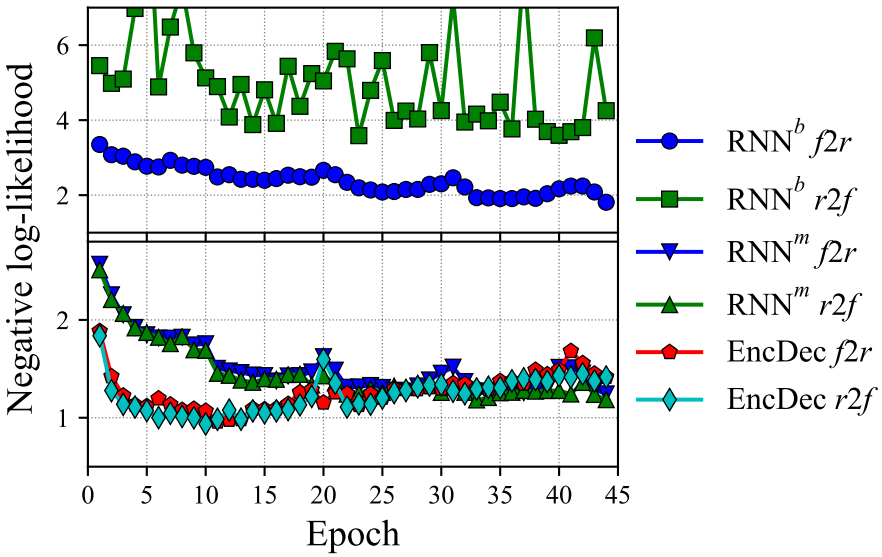


Figure 4.2: Negative log-likelihood of RNNs on the validation set of Reuters-21578.

orderings, such as frequent-to-rare (*f2r*), and rare-to-frequent (*r2f*), as well as a topological sorting (when applicable) in terms of *subset accuracy* (ACC), *Hamming accuracy* (HA), *example-based* F_1*-measure* (ebF_1), *micro-averaged* F_1 (miF_1) and *macro-averaged* F_1 (maF_1).

4.4.1 Experiments on Reuters-21578

Figure 4.2 shows the *negative log-likelihood* (NLL) of Eq. (4.3) on the validation set during the course of training. Note that as RNN^b attempts to predict binary targets, but RNN^m and EncDec make predictions on multinomial targets, the results of RNN^b are plotted separately, with a different scale of the y-axis (top half of the graph). Compared to RNN^m and EncDec, RNN^b converges very slowly. This can be attributed to the length of the label chain and sparse targets in the chain since RNN^b is trained to make correct predictions over all 90 labels, most of them being zero. In other words, the length of target sequences of RNN^b is 90 and fixed regardless of the content of training documents. In particular, RNN^b has trouble with the *r2f* label ordering, where training is unstable. The reason is presumably that the predictions for later labels depend on sequences that are mostly zero when rare labels occur at the beginning. Hence, the model sees only few examples of non-zero targets in a single epoch. On the other hand, both RNN^m and EncDec converge relatively faster than RNN^b and do obviously not suffer from the *r2f* ordering. Moreover, there is not much difference between both strategies since the length of the sequences is often 1 for Reuters-21578 and hence often the same.

Figure 4.3 shows the performance of RNNs in terms of all evaluation measures on the validation set. EncDec performs best for all the measures, followed by RNN^m. There is no clear difference between the same type of models trained on different label permutations, except for RNN^b in terms of NLL (cf. Fig. 4.2). Note that although it takes more time to update the parameters of EncDec than those of RNN^m, EncDec ends up with better results. RNN^b performs poorly especially in terms of maF_1 regardless of the label permutations, sug-

Table 4.3: Performance comparison on Reuters-21578.

	ACC	HA	ebF_1	miF_1	maF_1
No label permutations					
BR(NN)	0.7685	0.9957	0.8515	0.8348	0.4022
LP(NN)	0.7837	0.9941	0.8206	0.7730	0.3505
NN	0.7502	0.9952	0.8396	0.8183	0.3083
Frequent labels first (*f2r*)					
PCC(NN)	0.7844	0.9955	0.8585	0.8305	0.3989
RNN^b	0.6757	0.9931	0.7180	0.7144	0.0897
RNN^m	0.7744	0.9942	0.8396	0.7884	0.2722
EncDec	**0.8281**	0.9961	0.8917	0.8545	**0.4567**
Rare labels first (*r2f*)					
PCC(NN)	0.7864	0.9956	0.8598	0.8338	0.3937
RNN^b	0.0931	0.9835	0.1083	0.1389	0.0102
RNN^m	0.7744	0.9943	0.8409	0.7864	0.2699
EncDec	0.8261	**0.9962**	**0.8944**	**0.8575**	0.4365

gesting that RNN^b would need more parameter updates for predicting rare labels. Notably, the advantage of EncDec is most pronounced for this specific task.

Detailed results of all methods on the test set are shown in table 4.3. Clearly, EncDec perform best across all measures. LP works better than BR and NN in terms of ACC as intended, but performs behind them in terms of other measures. The reason is that LP, by construction, is able to more accurately hit the exact label set, but, on the other hand, produces more false positives and false negatives in our experiments in comparison to BR and NN when missing the correct label combination. As shown in the table, RNN^m performs better than its counterpart, i.e., RNN^b, in terms of ACC, but has clear weaknesses in predicting rare labels (cf. especially maF_1). For PCC, our two permutations of the labels do not affect much ACC due to the low label cardinality.

4.4.2 Experiments on RCV1-v2

In comparison to Reuters-21578, RCV1-v2 consists of a considerably larger number of documents. Though the the number of unique labels (L) is similar (103 vs. 90) in both datasets, RCV1-v2 has a higher C and LC is greatly increased from 468 to 14 921. Moreover, this dataset has the interesting property that all labels from the root to a relevant leaf label in the label tree are also associated to the document. In this case, we can also test a topological ordering of labels, as described in section 4.2.1. As RNN^b takes long to train and did not show good results on the small dataset, we have no longer considered it in these experiments. We instead include FastXML as a baseline.

Table 4.4 shows the performance of the methods with different label permutations. These results demonstrate again the superiority of PCC and RNN^m as well as EncDec against BR and NN in maximizing ACC. Another interesting observation is that LP performs much worse than other methods even in terms of ACC due to the data scarcity problem caused by higher LC. RNN^m and EncDec, which also predict label subsets but in a sequential

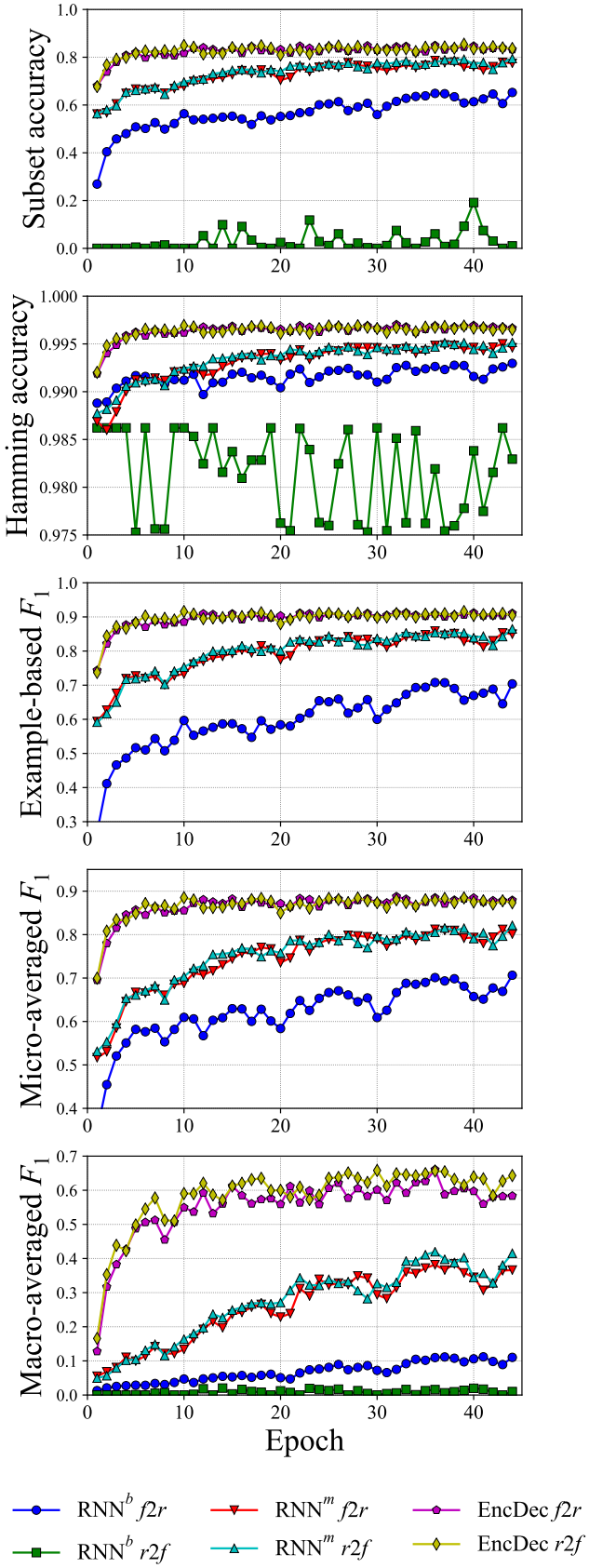


Figure 4.3: Performance of RNN models on the validation set of Reuters-21578 during training. Note that the x-axis denotes # epochs and we use different scales on the y-axis for each measure.

Table 4.4: Performance comparison on RCV1-v2.

	ACC	HA	ebF_1	miF_1	maF_1
No label permutations					
BR(NN)	0.5554	0.9904	0.8376	0.8349	0.6376
LP(NN)	0.5149	0.9767	0.6696	0.6162	0.4154
NN	0.5837	0.9907	0.8441	0.8402	0.6573
FastXML	0.5953	0.9910	0.8409	0.8470	0.5918
Frequent labels first (*f2r*)					
PCC(NN)	0.6211	0.9904	0.8461	0.8324	0.6404
RNN^m	0.6218	0.9903	0.8578	0.8487	0.6798
EncDec	**0.6798**	**0.9925**	0.8895	**0.8838**	0.7381
Rare labels first (*r2f*)					
PCC(NN)	0.6300	0.9906	0.8493	0.8395	0.6376
RNN^m	0.6216	0.9903	0.8556	0.8525	0.6583
EncDec	0.6767	**0.9925**	0.8884	0.8817	**0.7413**
topological sorting					
PCC(NN)	0.6257	0.9904	0.8463	0.8364	0.6486
RNN^m	0.6072	0.9898	0.8525	0.8437	0.6578
EncDec	0.6761	0.9924	0.8888	0.8808	0.7220
reverse topological sorting					
PCC(NN)	0.6267	0.9902	0.8444	0.8346	0.6497
RNN^m	0.6232	0.9904	0.8561	0.8496	0.6535
EncDec	0.6781	**0.9925**	**0.8899**	0.8797	0.7258

manner, do not suffer from the larger number of distinct label combinations. Similar to the previous experiment, we found no meaningful differences between the RNN^m and EncDec models trained on different label permutations on RCV1-v2. FastXML also performs well except for maF_1 which tells us that it focuses more on frequent labels than rare labels. As noted, this is because FastXML is designed to maximize top-k ranking measures such as prec@k for which the performance on frequent labels is important.

4.4.3 Experiments on BioASQ

Compared to Reuters-21578 and RCV1-v2, BioASQ has an extremely large number of instances and labels, where LC is almost close to $N_{tr} + N_{ts}$. In other words, nearly all distinct label combinations appear only once in the dataset and some label subsets can only be found in the test set. Table 4.5 shows the performance of FastXML, RNN^m and EncDec on the test set of BioASQ. EncDec clearly outperforms RNN^m by a large margin. Making predictions over several thousand labels is a particularly difficult task because MLC methods not only learn label dependencies, but also understand the context information in documents allowing us to find word-label dependencies and to improve the generalization performance.

We can observe a consistent benefit from using the reverse label ordering on both approaches. Note that EncDec does show reliable performance on two relatively small bench-

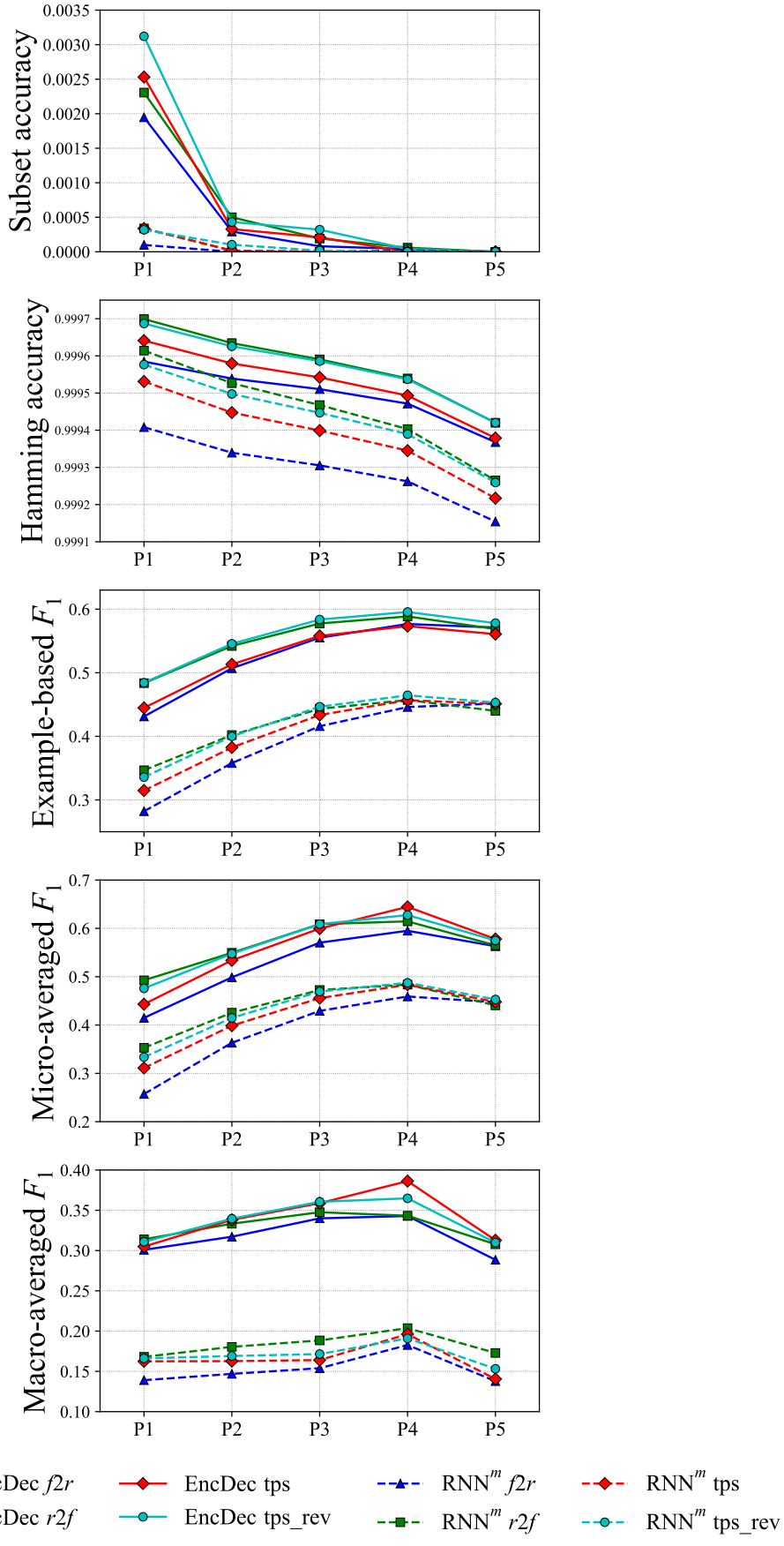


Figure 4.4: Comparison of RNN^m and EncDec wrt. the number of positive labels T of test documents. The test set is divided into 5 partitions according to T. The x-axis denotes partition indices. *tps* and *tps_rev* stand for the label permutation ordered by topological sorting and its reverse.

Table 4.5: Performance comparison on BioASQ.

	ACC	HA	ebF_1	miF_1	maF_1
No label permutations					
FastXML	0.0001	0.9996	0.3585	0.3890	0.0570
Frequent label first (*f2r*)					
RNN^m	0.0001	0.9993	0.3917	0.4088	0.1435
EncDec	0.0004	0.9995	0.5294	0.5634	0.3211
Rare labels first (*r2f*)					
RNN^m	0.0001	0.9995	0.4188	0.4534	0.1801
EncDec	0.0006	**0.9996**	0.5531	0.5943	0.3363
topological sorting					
RNN^m	0.0001	0.9994	0.4087	0.4402	0.1555
EncDec	0.0006	0.9953	0.5311	0.5919	**0.3459**
reverse topological sorting					
RNN^m	0.0001	0.9994	0.4210	0.4508	0.1646
EncDec	**0.0007**	**0.9996**	**0.5585**	**0.5961**	0.3427

marks regardless of the choice of the label permutations. Also, EncDec with reverse topological sorting of labels achieves the best performance, except for maF_1. Note that we observed similar effects with RNN^m in our preliminary experiments on RCV1-v2, but the impact of label permutations disappeared once we tuned RNN^m with dropout. This indicates that label ordering does not affect much the final performance of models if they are trained well enough with proper regularization techniques.

To understand the effectiveness of each model with respect to the size of the positive label set, we split the test set into five almost equally-sized partitions based on the number of target labels in the documents and evaluated the models separately for each of the partition, as shown in Fig. 4.4. The first partition (P1) contains test documents associated with 1 to 9 labels. Similarly, other partitions, P2, P3, P4 and P5, have documents with cardinalities of $10 \sim 12$, $13 \sim 15$, $16 \sim 18$ and more than 19, respectively. As expected, the performance of all models in terms of ACC and HA decreases as the number of positive labels increases. The other measures increase since the classifiers have potentially more possibilities to match positive labels. We can further confirm the observations from table 4.5 w.r.t. to different labelset sizes.

The margin of FastXML to RNN^m and EncDec is further increased. Moreover, its poor performance on rare labels confirms again the focus of FastXML on frequent labels. Regarding computational complexity, we could observe an opposed relation between the used resources: whereas we ran EncDec on a single GPU with 12G of memory for 5 days, FastXML only took 4 hours to complete (on 64 CPU cores), but, on the other hand, required a machine with 1024G of memory.

4.5 Conclusion

We have presented an alternative formulation of learning the joint probability of labels given an instance, which exploits the generally low label cardinality in multi-label classification problems. Instead of having to iterate over each of the labels as in the traditional classifier chains approach, the new formulation allows us to directly focus only on the positive labels. We provided an extension of the formal framework of probabilistic classifier chains, contributing to the understanding of the theoretical background of multi-label classification. Our approach based on recurrent neural networks, especially encoder-decoders, proved to be effective, highly scalable, and robust towards different label orderings on both small and large scale multi-label text classification benchmarks. However, some aspects of the presented work deserve further consideration.

5 Learning from Label Hierarchies

5.1 Introduction

Recent developments in multi-label classification can be roughly divided into two bodies of research. One is to build a classifier in favor of statistical dependencies between labels, and the other is devoted to making use of prior information over the label space. In the former area, many attempts have been made to exploit label patterns (Chekina et al., 2013; Dembczyński et al., 2012b; Read et al., 2011). As the number of possible configurations of labels grows exponentially with respect to the number of labels, it is required for multi-label classifiers to handle many labels efficiently (Bi and Kwok, 2013) or to reduce the dimensionality of a label space by exploiting properties of label structures such as sparsity (Hsu et al., 2009) and co-occurrence patterns (Chen and Lin, 2012). Label space dimensionality reduction (LSDR) methods allow to make use of latent information on a label space as well as to reduce computational cost. Another way of exploiting information on a label space is to use its underlying structures as a prior. Many methods have been developed to use hierarchical output structures in machine learning (Silla Jr. and Freitas, 2011). In particular, several researchers have looked into utilizing the hierarchical structure of the label space for improved predictions in multi-label classification (Rousu et al., 2006; Vens et al., 2008; Zimek et al., 2010).

Although extensive research has been devoted to techniques for utilizing implicitly or explicitly given label structures, there remain the scalability issues of previous approaches in terms of both the number of labels and documents in large feature spaces. Consider a very large collection of scientific documents covering a wide range of research interests. In an emerging research area, it can be expected that the number of publications per year grows rapidly. Moreover, new topics will emerge, so that the set of indexing terms, which has initially been provided by domain experts or authors to describe publications with few words for potential readers, will grow as well.

Interestingly, similar problems have been faced recently in a different domain, namely *representation learning* (Bengio et al., 2013). In language modeling, for instance, a word is traditionally represented by a K-dimensional vector where K is the number of unique words, typically hundreds of thousands or several millions. Clearly, it is desirable to reduce this dimensionality to much smaller values $d \ll K$. This can, e.g., be achieved with a simple log-linear model (Mikolov et al., 2013b), which can efficiently compute a so-called *word embedding*, i.e., a lower-dimensional vector representations for words. Another example for representation learning is a technique for learning a joint embedding space of instances and labels (Weston et al., 2011). This approach maximizes the similarity between vector representations of instances and relevant labels while projecting them into the same space.

Inspired by the log-linear model and the joint space embedding, we address large-scale multi-label classification problems, in which both hierarchical label structures are given *a priori* as well as label patterns occur in the training data. The mapping functions in the joint

space embedding method can be used to rank labels for a given instance, so that relevant labels are placed at the top of the ranking. In other words, the quality of such a ranking depends on the mapping functions. As mentioned, two types of information on label spaces are expected to help us to train better joint embedding spaces, so that the performance on unseen data can be improved. We focus on exploiting such information so as to learn a mapping function projecting labels into the joint space. The vector representations of labels by using this function will be referred to as *label embeddings*. While *label embeddings* are usually initialized randomly, it will be beneficial to learn the joint space embedding method taking label hierarchies into consideration when label structures are known. To this end, we adopt the above-mentioned log-linear model which has been successfully used to learn *word embeddings*.

Learning *word embeddings* relies fundamentally on the use of the context information, that is, a fixed number of words surrounding that word in a sentence or a document. In order to adapt this idea to learning *label embeddings*, we need to define context information in a label space, where, unlike in textual documents, there is no sequence information which can be used to define the context of words. We use, instead, *pairwise* relationships in label hierarchies and in label co-occurrence patterns.

There are two major contributions of this chapter:

1. We build efficient multi-label classifiers which employ label hierarchies so as to predict unseen labels.
2. We provide a novel method to efficiently learn label representations from hierarchical structures over labels as well as their co-occurrence patterns.

5.2 Model Description

5.2.1 Joint space embeddings

Weston et al. (2011) proposed an efficient online method to learn ranking functions in a joint space of instances and labels, namely *Wsabie*. Under the assumption that instances which have similar representation in a feature space tend to be associated with similar label sets, we find joint spaces of both instances and labels where the relevant labels for an instance can be separated from the irrelevant ones with high probability.

Formally, consider an instance $\mathbf{x}$ of dimension D and a set of labels $\boldsymbol{y}$ associated with $\mathbf{x}$. Let $\phi(\mathbf{x}) = \mathbf{W}\mathbf{x}$ denote a linear function which projects the original feature representations of an instance $\mathbf{x}$ to a d-dimensional joint space, where $\mathbf{W} \in \mathbb{R}^{d \times D}$ is a transformation matrix. Similarly, let $\mathbf{U}$ be a $d \times L$ matrix that maps labels into the same joint d-dimensional space. A label $i \in \boldsymbol{y}$ can then be represented as a d-dimensional vector $\mathbf{u}_i$, which is the i-th column vector of $\mathbf{U}$. We will refer to the matrix $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_L]$ as *label embeddings*. The objective function is given by

$$\mathcal{L}\left(\mathbf{\Theta}_F; \mathcal{D}\right) = \sum_{n=1}^{N} \frac{1}{|\boldsymbol{y}_n|} \sum_{i \in \boldsymbol{y}_n} \sum_{j \in \bar{\boldsymbol{y}}_n} h(r_i(\mathbf{x}_n))\, \ell\left(\mathbf{x}_n, y_i, y_j\right) \tag{5.1}$$

with the *pairwise hinge loss* function

$$\ell\left(\mathbf{x}_n, y_i, y_j\right) = \left[m_a - \mathbf{u}_i^T \phi(\mathbf{x}_n) + \mathbf{u}_j^T \phi(\mathbf{x}_n)\right]_+ \tag{5.2}$$

where $r_i(\cdot)$ denotes the rank of label i for a given instance $\mathbf{x}_n$, $h(\cdot)$ is a function that maps this rank to a real number (to be introduced shortly in more detail), $\bar{\boldsymbol{y}}_n$ is the complement of $\boldsymbol{y}_n$, $[x]_+$ is defined as x if $x > 0$ and 0 otherwise, $\boldsymbol{\Theta}_F = \{\mathbf{W}, \mathbf{U}\}$ are model parameters, and m_a is a real-valued parameter, namely the *margin*. The relevance scores $\mathbf{s}(\mathbf{x}) = [s_1(\mathbf{x}), s_2(\mathbf{x}), \cdots, s_L(\mathbf{x})]$ of labels for a given instance $\mathbf{x}$ can be computed as $s_i(\mathbf{x}) = \mathbf{u}_i^T \phi(\mathbf{x}) \in \mathbb{R}$. Then, the rank of label i with respect to an instance $\mathbf{x}$ can be determined based on the relevance scores

$$r_i(\mathbf{x}) = \sum_{j \in \bar{\boldsymbol{y}}, j \neq i} [m_a - s_i(\mathbf{x}) + s_j(\mathbf{x})]_+ . \tag{5.3}$$

It is prohibitively expensive to compute such rankings exactly when L is large.

The rank of label i, i.e., $r_i(\mathbf{x})$, is determined by the number of negative labels $j \in \bar{\mathbf{y}}_n$ that are ranked higher than the positive label i. Let N_k be a random variable that stands for the number trials to choose a negative label j that is placed at a higher rank than a positive label i if there are k such labels. The random variable N_k follows the geometric distribution, which is the probability distribution of the number of trials until the first success with some probability p where $0 < p < 1$. The probability of drawing a *violating* label out of k in this case is defined as

$$p = \frac{k}{L - |\boldsymbol{y}|}.$$

Given the expected number of trials $\mathbb{E}[N_k] = \frac{1}{p}$, we can calculate approximately $r_i(\mathbf{x})$ in Eq. (5.3) as follows

$$r_i(\mathbf{x}) \approx \left\lfloor \frac{L - |\boldsymbol{y}|}{S_i} \right\rfloor$$

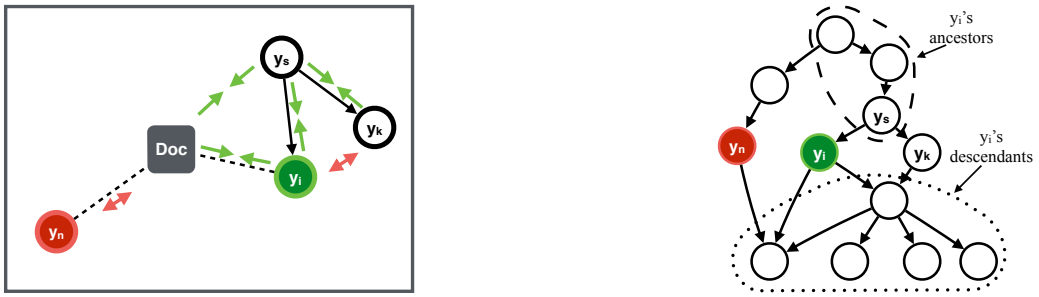
where $\lfloor \cdot \rfloor$ denotes the floor function and S_i, an approximation of $\mathbb{E}[N_k]$, is the number of trials to sample an index j yielding incorrect ranking against label i such that $m_a - s_i(\mathbf{x}) + s_j(\mathbf{x}) > 0$. Having an approximate rank $r_i(\mathbf{x})$, we can obtain a weighted ranking function

$$h(r_i(\mathbf{x})) = \sum_{l=1}^{r_i(\mathbf{x})} \frac{1}{l},$$

which is shown to be an effective way of optimizing precision at the top of rankings.

5.2.2 Learning with hierarchical structures over labels

Wsabie is trained in a way that the margin of similarity scores between positive associations $\mathbf{u}_p^T \phi(\mathbf{x})$ and negative associations $\mathbf{u}_n^T \phi(\mathbf{x})$ is maximized, where $\mathbf{u}_p$ and $\mathbf{u}_n$ denote the embeddings of relevant and irrelevant labels, respectively, for an instance $\mathbf{x}$. In practice, this approach works well if label patterns of test instances appear in training label patterns. If there are few or no training instances for some labels, the model may fail to make predictions accurately on test instances associated with those labels. In such cases, a joint space learning method could benefit from label hierarchies. In this section, we introduce a simple and efficient joint space learning method by adding a regularization term which employs label hierarchies, hereafter referred to as *Wsabie*$_{\mathrm{H}}$.



(a) A joint space of instances and labels **(b)** A label hierarhcy

Figure 5.1: An illustrative example of our proposed method. A label y_i (green circle) indicates a relevant label for a document (rectangle) while y_n (red circle) is one of the irrelevant labels. In the joint space, we learn representations for the relevant label, its ancestor y_s, and the document to be similar whereas the distance between the document and the irrelevant label is maximized. Also, the parent label, y_s, and its children are forced to be similar while sibling labels of y_i, i.e. y_k, are kept away from each other.

Notations. Consider multi-label problems where label hierarchies exist. Label graphs are a natural way to represent such hierarchical structures. Because it is possible for a label to have more than one parent node, we represent a hierarchy of labels in a directed acyclic graph (DAG). Consider a graph $\mathcal{G} = \{V, E\}$ where V denotes a set of nodes and E represent a set of connections between nodes. A node $u \in V$ corresponds to a label. A directed edge from a node u to a node v is denoted as $e_{u,v}$, in which case we say that u is a parent of v and v is a child of u. The set of all parents / children of v is denoted with $\mathcal{S}_{\mathcal{P}}(v)$ / $\mathcal{S}_{\mathcal{C}}(v)$. If there exists a directed path from u to v, u is an ancestor of v and v is a descendant of u, the set of all ancestors / descendants is denoted as $\mathcal{S}_{\mathcal{A}}$ / $\mathcal{S}_{\mathcal{D}}(u)$.

Label structures as regularizers. As an example, let us consider three labels, "computer science" (CS), "artificial intelligence" (AI), and "software engineering" (SE). The label CS can be viewed as a parent label of AI and SE. Given a paper dealing with problems in artificial intelligence and having AI as a label, we wish to learn a joint embedding model in a way that it is also highly probable to predict CS as a relevant label. Following our hypothesis in label spaces, even though we have no paper of software engineering, a label hierarchy allows us to make *reasonable* predictions on such a label by representing label SE close to label CS in a vector space. In order to prevent the model from converging to trivial solutions that representations of all three labels are identical, it is desired that sibling labels such as AI and SE in the hierarchy are well separated from each other in a joint embedding space. For an illustration of our method, see Fig. 5.1.

Formally, we can achieve this by defining a regularization term Ω, which takes into account the hierarchical label structure

$$\begin{aligned}\Omega(\mathbf{\Theta}_H) = &\sum_{n=1}^{N} \frac{1}{\mathcal{Z}_A} \sum_{i \in \boldsymbol{y}_n} \sum_{s \in \mathcal{S}_{\mathcal{A}}(i)} -\log p(y_s | y_i, \mathbf{x}_n) \\ &+ \sum_{l=1}^{L} \frac{1}{|\mathcal{S}_{\mathcal{P}}(l)|} \sum_{q \in \mathcal{S}_{\mathcal{P}}(l)} \sum_{\substack{k \in \mathcal{S}_{\mathcal{C}}(q) \\ k \neq l}} h(r_q(\mathbf{u}_l)) \left[m_b - \mathbf{u}_q^T \mathbf{u}_l + \mathbf{u}_k^T \mathbf{u}_l \right]_+ \end{aligned} \tag{5.4}$$

where m_b is the margin, $\mathcal{Z}_A = |\boldsymbol{y}_n||\mathcal{S}_\mathcal{A}(i)|$, and $p(y_s|y_i, \mathbf{x}_n)$ denotes the probability of predicting an ancestor label s of a label i given i and an instance $\mathbf{x}_n$ for which i is relevant. More specifically, the probability $p(y_s|y_i, \mathbf{x}_n)$ can be defined as

$$p(y_s|y_i, \mathbf{x}_n) = \frac{\exp(\mathbf{u}_s^T \hat{\mathbf{u}}_i^{(n)})}{\sum_{v=1}^{L} \exp(\mathbf{u}_v^T \hat{\mathbf{u}}_i^{(n)})}, \tag{5.5}$$

where $\hat{\mathbf{u}}_i^{(n)} = \frac{1}{2}(\mathbf{u}_i + \phi(\mathbf{x}_n))$ is the averaged-representation of a label i and the n-th instance in a joint space. We use the hierarchical softmax discussed in Section 2.3.2 to compute Eq. (5.5). Intuitively, this regularizer forces labels, which share the same parent label, to have similar vector representations as much as possible while keeping them separated from each other. Moreover, an instance $\mathbf{x}$ has the potential to make good predictions on some labels even though they do not appear in the training set only if their descendants are associated with training instances.

Adding Ω to Eq. 5.1 results in the objective function of $Wsabie_{\text{H}}$

$$\mathcal{L}(\boldsymbol{\Theta}_H; \mathcal{D}) = \sum_{n=1}^{N} \frac{1}{|\boldsymbol{y}_n|} \sum_{i \in \boldsymbol{y}_n} \sum_{j \in \bar{\boldsymbol{y}}_n} h(r_i(\mathbf{x}_n))\, \ell(\mathbf{x}_n, y_i, y_j) + \lambda \Omega(\boldsymbol{\Theta}_H) \tag{5.6}$$

where λ is a control parameter of the regularization term. If we set $\lambda = 0$, then the above objective function is equivalent to the objective function of *Wsabie* in Eq 5.1.

5.2.3 Label ranking to binary predictions

It is often sufficient in practice to just predict a ranking of labels instead of a bipartition of labels, especially in settings where the learning system is comprehended as supportive (Crammer and Singer, 2003). On the other hand, there are several ways to convert ranking results into a bipartition. Basically all of them split the ranking at a certain position depending on a predetermined or predicted threshold or amount of relevant labels.

Instead of experimenting with different threshold techniques, we took a pragmatic stance, and simply assume that there is an oracle which tells us the actual number of relevant labels for an unseen instance. This allows us to evaluate and compare the ranking quality of our approaches independently of the performance of an underlying thresholding technique. The bipartition measures obtained by this method could be interpreted as a (soft) upper bound for any thresholding approach.

5.3 Experimental Setup

Datasets. We benchmark our proposed method on two textual corpora consisting of a large number of documents and with label hierarchies provided.

The RCV1-v2 dataset (Lewis et al., 2004) is a collection of newswire articles. There are 103 labels and they are organized in a tree. Each label belongs to one of four major categories. The original train/test split in the RCV1-v2 dataset consists of 23,149 training documents and 781,265 test documents. In our experiments, we switched the training and the test data,

Table 5.1: Number of instances (M), Size of vocabulary (D), Number of labels (L), Average number of labels per instance (C), and the type of label hierarchy (HS). L subscripted k and u denote the number of *known* and *unseen* labels, respectively.

	Original datasets			Modified datasets				D	HS
	M	L	C	M	L_{k}	L_{u}	C		
RCV1-v2	804 414	103	3.24	700 628	82	21	1.43	20 000	Tree
OHSUMED	233 369	27 483	9.07	100 735	9570	17 913	3.87	25 892	DAG

and selected the top 20,000 words according to the document frequency. We chose randomly 10,000 training documents as the validation set.

The second corpus is the OHSUMED dataset (Hersh et al., 1994) consisting of 348,565 scientific articles from MEDLINE. Each article has multiple index terms known as Medical Subject Headings (MeSH). In this dataset, the training set contains articles from year 1987 while articles from 1988 to 1991 belong to the test set. We map all MeSH terms in the OHSUMED dataset to 2015 MeSH vocabulary[1] in which 27,483 MeSH terms are organized in a DAG hierarchy. Originally, the OHSUMED collection consists of 54,710 training documents and 293,856 test documents. Having removed all MeSH terms that do not appear in the 2015 MeSH vocabulary, we excluded all documents that have no label from the corpus. To represent documents in a vector space, we selected unigram words that occur more than 5 times in the training set. These pre-processing steps left us with 36,883 train documents and 196,486 test documents. Then, 10% of the training documents were randomly set aside for the validation set. Finally, for both datasets, we applied *log tf-idf* term-weighting and then normalized document vectors to unit length.

Preparation of the datasets in zero-shot settings. We hypothesize that label hierarchies provide possibilities of learning representations of unseen labels, thereby improving predictive performance for unseen data. To test our hypothesis, we modified the datasets. For the RCV1-v2 dataset, we removed all labels corresponding to non-terminals in the label hierarchy from training data and validation data while these non-terminal labels remain intact in the test set. In other words, we train models with labels corresponding to the leaves in the label hierarchy, then test them on the modified test set which only contains *unseen* labels.

Since the train and test examples of the OHSUMED dataset was split by year, the training data does not cover all labels in the test set. More specifically, there are 27,483 labels in the label hierarchy (cf. Table 5.1), of which only 9,570 occur in both training and test sets, which will be referred to as the set of *known* labels. Of the 12,568 labels that occur in the test set, 2,998 cannot be found in the *known* labels set, and thus form a set of *unseen* labels together with the 14,915 labels which are only available in the label hierarchy, but not present in the label patterns. In order to test predictive performance on these unseen labels, we omitted all labels in the *known* label set from the test examples. This resulted in some test examples having an empty set of labels, which were ignored for the evaluation. Finally, the above preprocessing steps left us 67,391 test examples.

The statistics of the datasets and the modified ones are summarized in Table 5.1.

[1] http://www.nlm.nih.gov/pubs/techbull/so14/so14_2015_mesh_avail.html

Representing parent-child pairs of MeSH terms in a DAG. As mentioned earlier, we use parent-child pairs of MeSH terms in the 2015 MeSH vocabulary as the label hierarchy for the OHSUMED dataset. If we represent parent-child pairs of labels as a graph, it may contain cycles. Hence, we removed edges resulting in cycles as follows: 1) Pick a node that has no parent as a starting node. 2) Run Depth-First Search (DFS) from the starting node in order to detect edges pointing to nodes visited already, then remove such edges. 3) Repeat the 1 & 2 steps until all nodes having no parents are visited. There are 16 major categories in the MeSH vocabulary. In contrast to RCV1-v2, the MeSH terms are formed in complex structures so that a label can have more than one parent.

Baselines. We compare our algorithm, Wsabie_{H}, which uses hierarchical information for label embeddings, to Wsabie ignoring label hierarchies and several other benchmark algorithms. For *binary relevance* (BR), which decomposes a multi-label problem into L binary problems, we use LIBLINEAR (Fan et al., 2008) as a base learner which is a good compromise between efficiency and effectiveness in multi-label text document classification.

To address the limitations of BR, specifically, when L is large, dimensionality reduction method on label spaces, namely *Principal Label Space Transformation* (PLST) and *Conditional Principal Label Space Transformation* (CPLST), have been proposed (Chen and Lin, 2012; Tai and Lin, 2012) which try to capture label correlations before learning per-label classifiers. Instead of directly predicting labels for given instances, the *LSDR approach* learns d-output linear predictors in a reduced label space. Then, the original label space is reconstructed from the outputs of the linear predictors using the transformation matrix for reducing the label dimensionality. We use ridge regression as a linear predictor.

Pairwise decomposition has been already successfully applied for multi-label text classification (Fürnkranz et al., 2008; Loza Mencía and Fürnkranz, 2008). Here, one classifier is trained for each pair of classes, i.e., a problem with L different classes is decomposed into $\frac{L(L-1)}{2}$ subproblems. At test time, all of the $\frac{L(L-1)}{2}$ base classifiers make a prediction for one of its two corresponding classes, which is interpreted as a full vote (0 or 1) for this label. Adding these up results in a ranking over the labels. To convert the ranking into a multi-label prediction, we use the *calibrated label ranking* (CLR) approach. Though CLR is able to predict cutting points of ranked lists, in this work, in order to allow a fair comparison, it also relies on an oracle to predict the number of relevant labels for a given instance (cf Section 5.2.3). We denote by CLR_{svm} the use of CLR in combination with SVMs.

Training Details. All hyperparameters were empirically chosen based on AvgP on validation sets. The dimensionality of the joint space d was selected in a range of $\{16, 32, 64, 128\}$ for the RCV1-v2 dataset and $\{128, 256, 512\}$ for the OHSUMED dataset. The margins m_a and m_b were chosen ranging from $\{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$. We used Adagrad (Duchi et al., 2011) to optimize parameters $\mathbf{\Theta}$ in Eq. 5.1 and 5.6. Let $\Delta_{i,\tau}$ be the gradient of the objective function in Eq. 5.6 with respect to a parameter $\theta_i \in \mathbf{\Theta}$ at time τ. Then, the update rule for parameters indexed i at time τ is given by $\theta_i^{(\tau+1)} = \theta_i^{(\tau)} - \eta_i^{(\tau)} \Delta_{i,t}$ with an adaptive learning rate per parameter $\eta_i^{(\tau)} = \eta_0 \Big/ \sqrt{\sum_{t=1}^{\tau} \Delta_{i,t}^2}$ where $\eta_0 \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ denotes a base learning rate which decrease by a factor of 0.99 per epoch. We implemented our proposed methods using a lock-free parallel gradient update scheme (Recht et al., 2011), namely *Hogwild!*, in a shared memory system since the number of parameters

Table 5.2: Comparison of *Wsabie*$_{\mathrm{H}}$ to its baselines on the benchmarks. (Best in bold)

	RCV1-v2					
	BR	PLST	CPLST	CLR$_{svm}$	Wsabie	Wsabie$_{\mathrm{H}}$
AvgP	94.20	92.75	92.76	94.76	94.34	94.39
RL	0.46	0.78	0.76	0.40	0.44	0.44

	OHSUMED					
	BR	PLST	CPLST	CLR$_{svm}$	Wsabie	Wsabie$_{\mathrm{H}}$
AvgP	45.00	26.50	-	-	45.72	45.76
RL	4.48	15.06	-	-	4.09	3.72

involved during updates is sparse even though the whole parameter space is large. For BR and CLR$_{svm}$, LIBLINEAR(Fan et al., 2008) was used as a base learner and the regularization parameter $C = \{10^{-2}, 10^{0}, 10^{2}, 10^{4}, 10^{6}\}$ was chosen by validation sets.

5.4 Experimental Results

5.4.1 Learning All Labels Together

Table 5.2 compares our proposed algorithm, *Wsabie*$_{\mathrm{H}}$, with the baselines on the benchmark datasets in terms of two ranking measures. It can be seen that CLR$_{svm}$ outperforms the others including *Wsabie*$_{\mathrm{H}}$ on the RCV1-v2 dataset, but the performance gap across all algorithms in our experiments is not large. Even BR ignoring label relationship works competitively on this dataset. Also, no difference between *Wsabie* and *Wsabie*$_{\mathrm{H}}$ was observed. This is attributed to characteristics of the RCV1-v2 dataset that if a label corresponding to one of the leaf nodes in the label hierarchy is associated with an instance, (almost) all nodes in a path from the root node to that node are also present, so that the hierarchical information is implicitly present in the training data.

Let us now turn to the experimental results on the OHSUMED dataset which are shown at the bottome of Table 5.2. Since the dataset consists of many labels, as an LSDR approach, we include PLST only in this experiment because CPLST is computationally more expensive than PLST, but no significant difference was observed. Similarly, due to the computational cost of CLR$_{svm}$ with respect to the number of labels, we excluded it from the experiment. It can be seen that regardless of the choice of the regularization term, the Wsabie approaches perform better than the other methods. PLST performed poorly under the settings where the density of labels, i.e., C/L in Table 5.1, is very low. Moreover, we projected the original label space L =27,483 into a much smaller dimension $d = 512$ using a small amount of training examples. Although the difference to BR is rather small, the margin is more pronounced that on RCV1.

Table 5.3: The performance of *Wsabie*$_{\mathrm{H}}$ compared to its baseline on the benchmarks in *zero-shot* learning settings.

	RCV1-v2				OHSUMED			
	AvgP	RL	MiF	MaF	AvgP	RL	MiF	MaF
Wsabie	2.31	62.29	0.00	0.00	0.01	56.37	0.00	0.00
Wsabie$_{\mathrm{H}}$	**9.47**	**30.39**	**0.50**	**1.64**	**0.06**	**39.91**	0.00	0.00

5.4.2 Learning to Predict Unseen Labels

Over the last few years, there has been an increasing interest in *zero-shot learning*, which aims to learn a function that maps instances to classes or labels that have not been seen during training. Visual attributes of an image (Lampert et al., 2014) or textual description of labels (Frome et al., 2013; Socher et al., 2013) may serve as additional information for zero-shot learning algorithms. In contrast, in this work, we focus on how to exploit label hierarchies and co-occurrence patterns of labels to make predictions on such *unseen* labels. The reason is that in many cases it is difficult to get additional information for some specific labels from external sources. In particular, while using a semantic space of labels' textual description is a promising way to learn vector representations of labels, sometimes it is not straightforward to find suitable mappings of specialized labels.

Table 5.3 shows the results of *Wsabie* against *Wsabie*$_{\mathrm{H}}$ on the modified datasets which do not contain any known label in the test set (cf. Sec. 5.3). As can be seen, *Wsabie*$_{\mathrm{H}}$ clearly outperforms *Wsabie* on both datasets across all measures except for MiF and MaF on the OHSUMED dataset. Note that the key difference between *Wsabie*$_{\mathrm{H}}$ and *Wsabie* is the use of hierarchical structures over labels during the training phase. Since the labels in the test set do not appear during training, *Wsabie* can basically only make random predictions for the unknown labels. Hence, the comparison shows that taking only the hierarchical relations into account already enables a considerable improvement over the baseline. Unfortunately, the effect is not substantial enough in order to be reflected w.r.t. MiF and MaF on OHSUMED. Note, however, that a relevant, completely unknown label must be ranked approximately as one of the top 4 labels out of 17,913 in order to count for bipartition measures in this particular setting.

In summary, these results show that the regularization of joint embedding methods is an effective way of learning representations for *unseen* labels in a tree-structured hierarchy of a small number of labels . However, if a label hierarchy is defined on more complex structures and while a fewer number of training examples exists per label, it might be difficult for *Wsabie*$_{\mathrm{H}}$ to work well on unseen data.

5.5 Pretrained Label Embeddings as Good Initial Guess

From the previous experiments, we see that the regularization of *Wsabie*$_{\mathrm{H}}$ using the hierarchical structure of labels allows us to obtain better performance for *unseen* labels. The objective function (Eq. 5.6) penalizes parameters of observable labels in the training data by the negative log probability of predicting their ancestors in a hierarchy. If we initialize

label spaces parameterized by $\mathbf{U}$ at random, presumably, the regularizer may rather act as noise at a beginning stage of the training. Especially for OHSUMED, the label hierarchy is complex and positive documents are very few for some labels.

We address this by exploiting both label hierarchies and co-occurrence patterns between labels in the training data. Apart from feature representations of a training instance, it is possible to capture underlying structures of the label space based on the co-occurrence patterns. Hence, we propose a method to learn label embeddings from hierarchical information and *pairwise* label relationships.

The basic idea of pretraining label embeddings is to maximize the probability of predicting an ancestor given a particular label in a hierarchy as well as predicting co-occurring labels with it. Given the labels of N training instances $\mathcal{D}_{\boldsymbol{y}} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_N\}$, the objective function is to maximize the average log probability given by

$$\sum_{n=1}^{N}\Bigg[\frac{(1-\alpha)}{\mathcal{Z}_{\mathcal{A}}}\sum_{i\in\boldsymbol{y}_n}\sum_{j\in\mathcal{S}_{\mathcal{A}}(i)}\log p\left(y_j|y_i\right)+\frac{\alpha}{\mathcal{Z}_{\mathcal{N}}}\sum_{i\in\boldsymbol{y}_n}\sum_{\substack{k\in\boldsymbol{y}_n\\k\neq i}}\log p\left(y_k|y_i\right)\Bigg] \tag{5.7}$$

where α determines the importance of each term ranging from 0 to 1, $\mathcal{Z}_{\mathcal{A}} = |\boldsymbol{y}_n||\mathcal{S}_A(\cdot)|$ and $\mathcal{Z}_{\mathcal{N}} = |\boldsymbol{y}_n|(|\boldsymbol{y}_n| - 1)$. The probability of predicting an ancestor label j of a label i, i.e., $p(y_j|y_i)$, can be computed similarly to Eq. 5.5 by using *softmax* and slight modifications. Thus, the log-probability can be defined by

$$p(y_j|y_i) = \frac{\exp(\mathbf{u'}_j^T\mathbf{u}_i)}{\sum_{v=1}^{L}\exp(\mathbf{u'}_v^T\mathbf{u}_i)} \tag{5.8}$$

where $\mathbf{u}_i$ is the i-th column vector of $\mathbf{U} \in \mathbb{R}^{d\times L}$ and $\mathbf{u'}_j$ is a vector representation for label j and the j-th column vector of $\mathbf{U'} \in \mathbb{R}^{d\times L}$. The softmax function in Eq. 5.8 can be viewed as an objective function of a neural network consisting of a linear activation function in the hidden layer and two weights $\{\mathbf{U}, \mathbf{U'}\}$, where $\mathbf{U}$ connects the input layer to the hidden layer while $\mathbf{U'}$ is used to convey the hidden activations to the output layer. Here, $\mathbf{U}$ and $\mathbf{U'}$ correspond to vector representations for input labels and output labels, respectively. Like Eq. 5.5, we use *hierarchical softmax* instead of Eq. 5.8 to speed up pre-training label embeddings.

The hierarchy of MeSH vocabulary consists of 16 major categories. Each index term belongs to at least one major category. After training the log-linear model on the BioASQ dataset, we selected labels corresponding terminals and belonging to a single major category in order to visualize learned representations. Figure 5.2 shows that using a hierarchy and co-occurrences of labels the model separates reasonably well 16 major categories defined in MeSH vocabulary. Please note that whereas we use tSNE for better visualization in Figure 5.2, *Principal Component Analysis* is used then to project label representations for the rest of figures in this paper. In the following section, we investigate what label representations learn from a hierarchy and co-occurrences.

5.5.1 Encoding Hierarchical Structures

As an illustration of our results, we will focus on a subgraph related to health care, shown in the top of Figure 5.3. Consider the leaf nodes in the figure. According to our objective in Equation 5.7, *Urban Health*, *Suburban Health* and *Rural Health* are trained to have

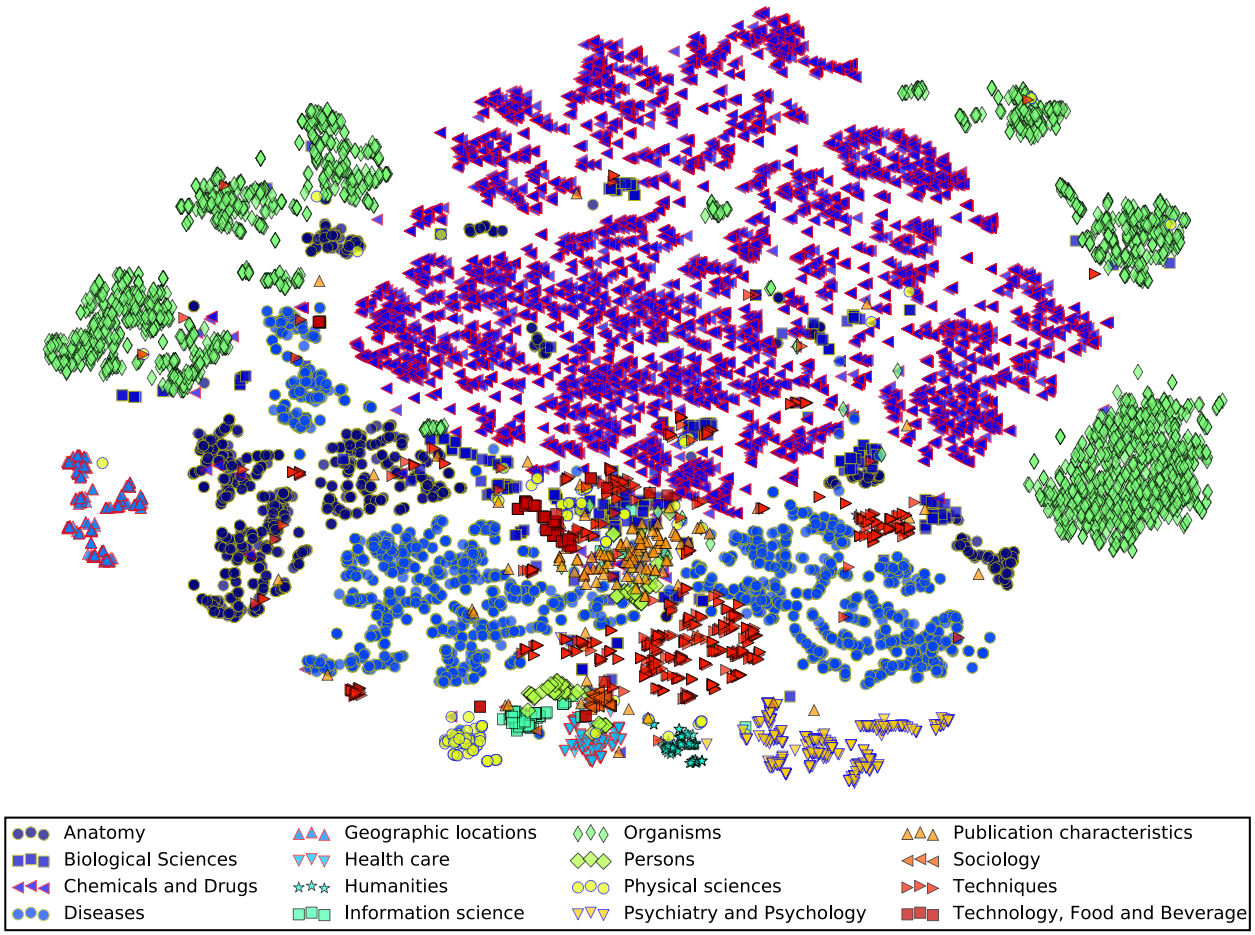


Figure 5.2: Learned representations of 16 major categories in MeSH vocabulary. Projection of label representations into 2D is done by *t-distributed stochastic neighbor embedding* (tSNE) (Van der Maaten and Hinton, 2008).

representations for predicting their common ancestors, i.e., *Health*, *Population Characteristics* and *Health care category*. The child nodes of *Population* are also trained similarly. Although *Urban*, *Suburban* and *Rural Health* are separated from *Urban*, *Suburban* and *Rural Population*, their representations tend to be similar since they share the same ancestors. In other words, *Urban Health* and *Rural Population*, for example, should have somewhat similar representations in part so as to increase a probability of predicting their common ancestors even though they are rarely assigned to the same instance.

We did perform an experiment to see whether learning from only the co-occurrences yields meaningful structures. In this case, our objective function is limited to the right term in Equation 5.7. Co-occurrence information enables to learn internal structures (see Figure 5.3b). If we consider hierarchical relationship between labels as well as co-occurrences of them, a more interesting property of our proposed method can be observed. A major difference between Figure 5.3b and 5.3c are the inter- and intra-group relationships among labels at the leaves. Specifically, all child nodes of *Health* are located in the left side and those of *Population* appear in the opposite side (Figure 5.3c). In addition to such relationship between two groups, relations between labels belonging to *Health* (on the left) resemble those found among the children of *Population* (on the right).

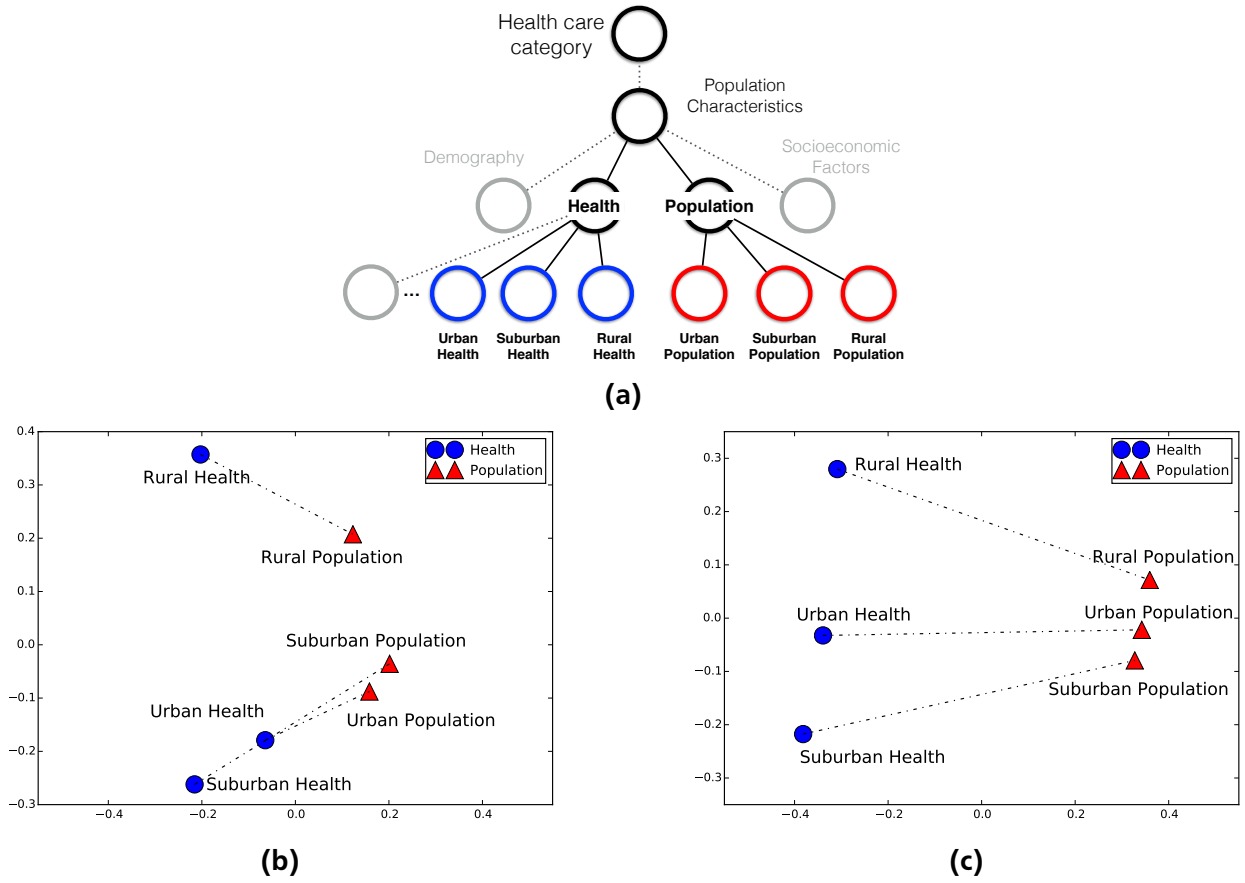


Figure 5.3: (a) A part of the hierarchy related to health care. (b) Learned vector representations for the index terms at leaf **without** the hierarchy (manually rotated). (c) Learned vector representations for the same index terms **with** the hierarchy.

5.5.2 Understanding Label Embeddings

We begin by qualitatively demonstrating label embeddings trained on label co-occurrence patterns from the BioASQ dataset (Balikas et al., 2014), which is one of the largest datasets for multi-label text classification, and label hierarchies in the 2015 MeSH vocabulary. The BioASQ dataset consists of more than 10 millions of documents. Note that we only use its label co-occurrence patterns. Its labels are also defined over the same MeSH vocabulary, so that we can use it for obtaining knowledge about the OHSUMED labels (cf Section 5.5). We trained the label embeddings using Eq. 5.7 by setting the dimensionality of the label embeddings to $d = \{128, 256, 512\}$ with different weighting values $\alpha = \{0, 0.5, 1\}$ for 100 epoch using SGD with a fixed learning rate of 0.1. If we set $d = 128$, training took about 6 hours on a machine with dual Xeon E5-2620 CPUs.

Analysis on learned label representations. Fig. 5.4 shows vector representations of labels related to Disorders/Diseases and their therapy in the 2015 MeSH vocabulary in 2D space.[2] It is likely that label pairs that co-occur frequently are close to each other. Particularly, on the *left* in Fig. 5.4, each therapy is close to a disorder for which the therapy is an effective treatment. If we make use of hierarchical information as well as co-occurrence label patterns

[2] Projection of 128-dim label embeddings into 2D was done by Principal Component Analysis.

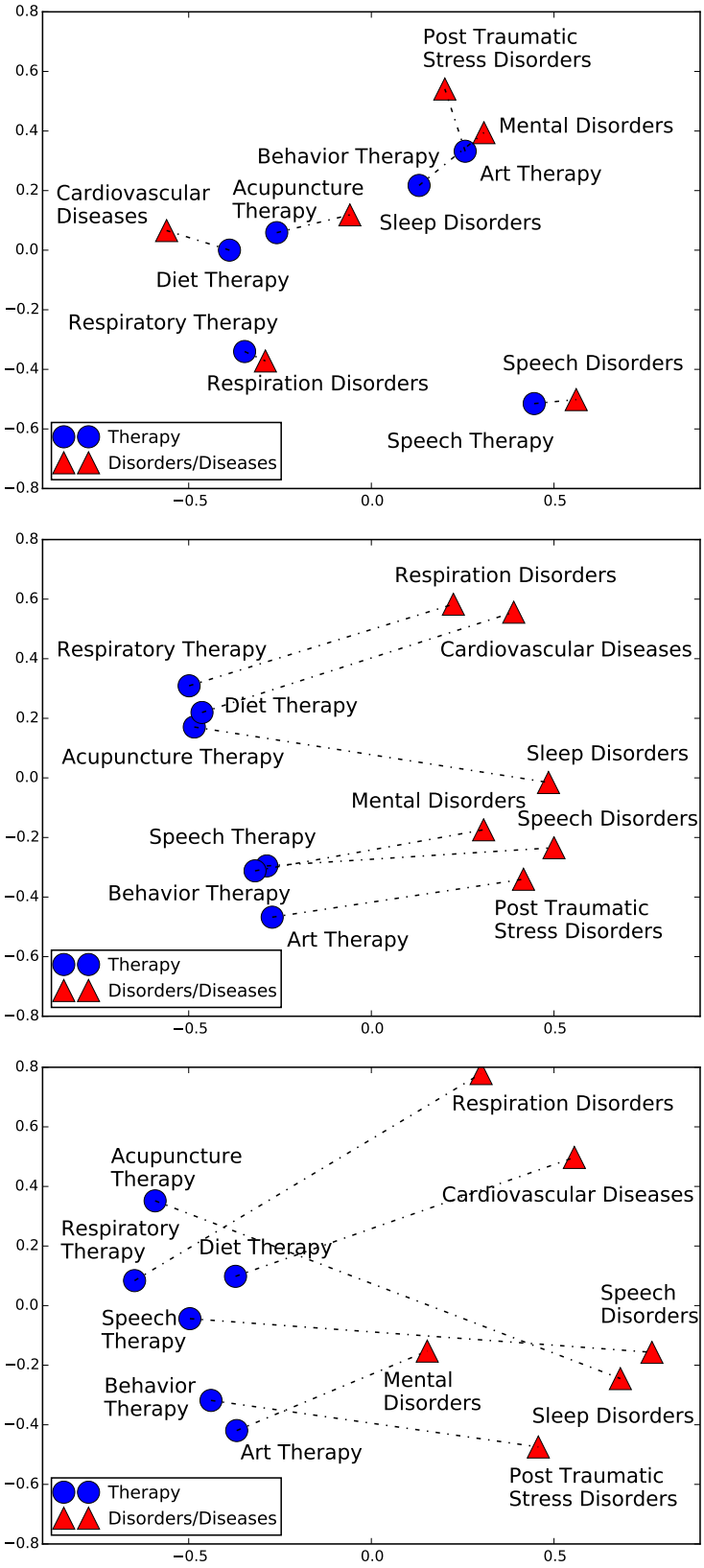


Figure 5.4: Visualization of learned label embeddings by the log-linear model (Eq 5.7). (*top*) using only label co-occurrence patterns $\alpha = 1$ (*middle*) using a hierarchy as well as co-occurrences $\alpha = 0.5$ (*bottom*) using only a hierarchy $\alpha = 0$.

Table 5.4: Analogical reasoning on learned vector representations of MeSH vocabulary

On learned representations *using* the hierarchy	
Analogy questions	Most probable answers
Cardiovascular Diseases : Diet Therapy ≈ Respiration Disorders : ?	Diet Therapy Enteral Nutrition Gastrointestinal Intubation Total Parenteral Nutrition Parenteral Nutrition Respiratory Therapy
Mental Disorders : Behavior Therapy ≈ PTSD : ?	Behavior Therapy Cognitive Therapy Rational-Emotive Psychotherapy Brief Psychotherapy Psychologic Desensitization Implosive Therapy

On learned representations *without using* the hierarchy	
Analogy questions	Most probable answers
Cardiovascular Diseases : Diet Therapy ≈ Respiration Disorders : ?	Respiration Disorders Respiratory Tract Diseases Respiratory Sounds Airway Obstruction Hypoventilation Croup
Mental Disorders : Behavior Therapy ≈ PTSD : ?	Behavior Therapy Psychologic Desensitization Internal-External Control PTSD Phobic Disorders Anger

during training, i.e., $\alpha = 0.5$ in Eq. 5.7, more interesting relationships are revealed which are not observed from the model trained only on co-occurrences ($\alpha = 1$). We can say that the learned vector representations has identified *Therapy-Disorders/Diseases* relationships (on the *middle* in Fig. 5.4). We also present label embeddings trained using only label hierarchies ($\alpha = 0$) on the *right* in Fig. 5.4.

Analogical reasoning in label spaces. One way to evaluate representation quality is analogical reasoning as shown in (Mikolov et al., 2013b). Upon the above observations (on the *middle* in Fig. 5.4), we performed analogical reasoning on both the representations trained with the hierarchy and ones without the hierarchy, specifically, regarding *Therapy-Disorders/Diseases* relationships (Table 5.4). As expected, it seems like the label representations trained with the hierarchy are clearly advantageous to the ones trained without the

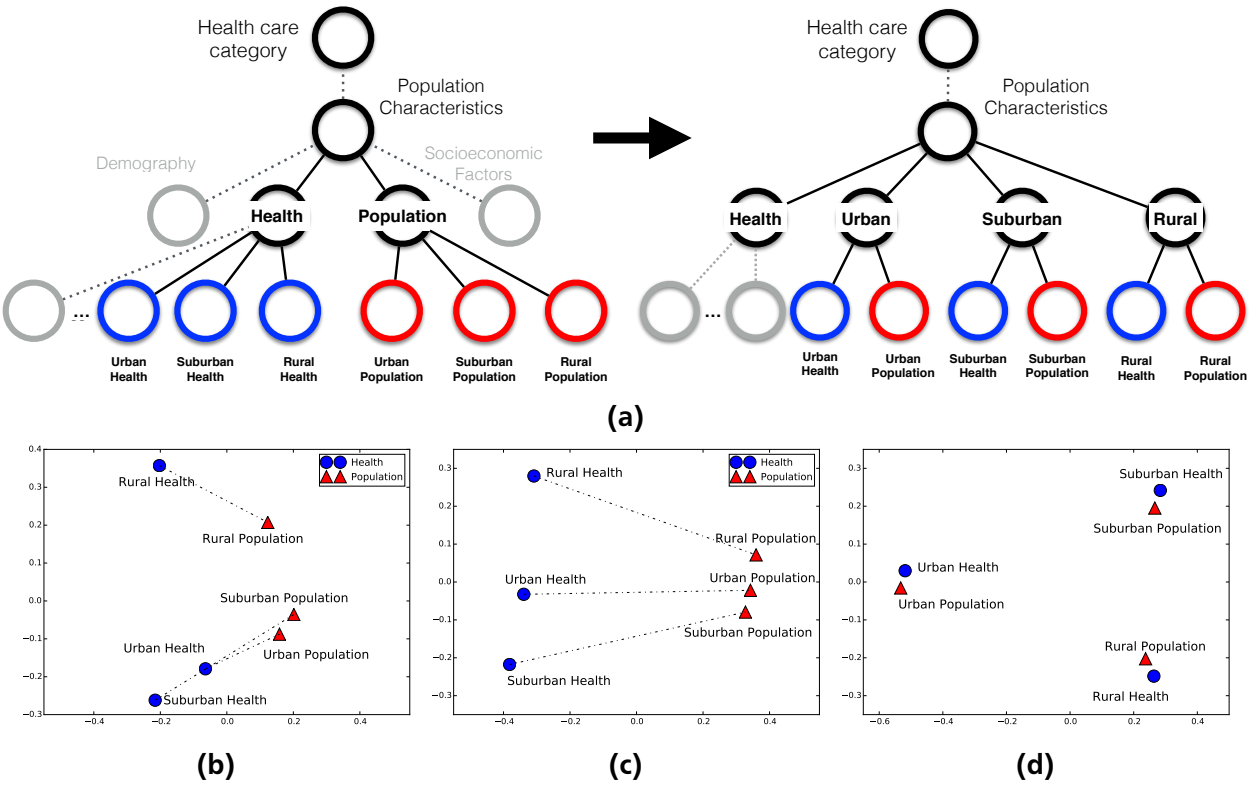


Figure 5.5: (a) A modified hierarchy (*right*) from the original one (*left*) obtained by grouping the same types of developed environments. See text for further explanation. (b) Learned vector representations without using a hierarchy. (c) Learned vector representations using the *original* hierarchy. (d) Learned vector representations using the *modified* hierarchy.

hierarchy on analogical reasoning. To be more specific, consider the first example, where we want to know what kinds of therapies are effective on "Respiration Disorders" as the relationship between "Diet Therapy" and "Cardiovascular Diseases". When we perform such analogical reasoning using learned embeddings with the hierarchy, the most probable answers to this analogy question are therapies that can be used to treat "Respiration Disorders" including nutritional therapies. Unlike the learned embeddings with the hierarchy, the label embeddings without the hierarchy perform poorly. In the bottom-right of Table 5.4, "Phobic Disorders" can be considered as a type of anxiety disorders that occur commonly together with "Post-traumatic Stress Disorders (PTSD)" rather than a treatment of it.

5.5.3 Different Hierarchies, Different Representations

In the previous section, we have shown our proposed method is capable of capturing hierarchical structures over labels and co-occurrences of them. In this case, it can be expected that the learned representations change when some part of the hierarchy is changed while the label co-occurrences remains same. To answer this question, firstly, we modified the original hierarchy (left in Fig. 5.5a) so as to obtain the new hierarchy (right in Fig. 5.5a). Originally, *Population Characteristics* has two child nodes, *Health* and *Population*. Contrary to *Population* that has only three child nodes, *Health* has dozens of child nodes. Instead of removing the other nodes from the hierarchy, we kept them as they are in the hierarchy. However, *Population* was removed from the hierarchy. We then created three new internal

Table 5.5: Initialization of label embeddings on OHSUMED under *zero-shot* settings.

	random label embeddings				**pretrained** label embeddings			
	AvgP	RL	MiF	MaF	AvgP	RL	MiF	MaF
Wsabie	0.01	56.37	0.00	0.00	**1.64**	**2.82**	0.03	0.06
Wsabie_{H}	0.06	39.91	0.00	0.00	1.36	5.33	**0.08**	**0.14**

nodes, *Urban*, *Suburban* and *Rural* representing types of developed environments. Finally, all nodes of interest (in blue and red) were grouped according to the developed types.

Figure 5.5d shows learned vector representations with modified hierarchy and compares the learned representations with the previous results.[3] Unlike the previous result in Fig. 5.5c, where *Health*-related nodes and *Population*-related nodes form clusters, *Urban Health* and *Urban Population* are clustered together since they share the common parent node, *Urban*. We can also observe similar patterns for *Suburban* and *Rural*. Besides, relative distance between each *Population* and *Health* within a cluster is identical, and the same direction vector from *Health* to *Population* or the other way around can be defined. Please note that, in this case, learning the model only from co-occurrences yields always similar label representations since we only modified the hierarchy (Fig. 5.5b).

5.5.4 Results

The results on the modified zero-shot learning datasets in Table 5.5 show that we can obtain substantial improvements by the pretrained label embeddings. Please note that the scores obtained by using *random* label embeddings on the left in Table 5.5 are the same as those of *Wsabie* and $\textit{Wsabie}_{\text{H}}$ in Table 5.3. In this experiment, we used very small base learning rates (i.e., $\eta_0 = 10^{-4}$ chosen by validation) for updating label embeddings in Eq. 5.6 after being initialized by the pretrained ones. This means that our proposed method is trained in a way that maps a document into the some point of label embeddings while the label embeddings hardly change. In fact, the pretrained label embeddings have interesting properties shown in Section 5.5.2, so that *Wsabie* starts learning at good initial parameter spaces. Interestingly, it was observed that some of the unseen labels are placed at the top of rankings for test instances, so that relatively higher scores of bipartition measures are obtained even for Wsabie. We also performed an experiment on the *full* OHSUMED dataset. The experimental results are given in Table 5.6. $\textit{Wsabie}_{\text{HP}}$ combining pretrained label embeddings with hierarchical label structures is able to further improve, outperforming both extensions by its own across all measures.

5.6 Conclusions

We have presented a method that learns a joint space of instances and labels taking hierarchical structures of labels into account. This method is able to learn representations of

[3] For the sake of readability, we used repeated results, the left graph in a, b and c taken from Figure 5.3, in order to clearly show the difference between representations learned from the original hierarchy and from the modified one as well as learned from only co-occurrences.

Table 5.6: Evaluation on the *full* test data of the OHSUMED dataset. Numbers in parentheses are standard deviation over 5 runs. Subscript P denotes the use of pretrained label embeddings.

	Wsabie		Wsaibe_{H}		Wsabie_{P}		$\text{Wsabie}_{\text{HP}}$	
AvgP	45.72	(0.04)	45.76	(0.06)	45.88	(0.09)	45.92	(0.02)
RL	4.09	(0.18)	3.72	(0.11)	3.44	(0.13)	3.11	(0.10)
MiF	46.32	(0.04)	46.34	(0.04)	46.45	(0.07)	46.50	(0.01)
MaF	13.93	(0.03)	13.96	(0.07)	14.19	(0.05)	14.25	(0.02)

labels, which are not presented during the training phase, by leveraging label hierarchies. We have also proposed a way of pretraining label embeddings from huge amounts of label patterns and hierarchical structures of labels.

We demonstrated the joint space learning method on two multi-label text corpora that have different types of label hierarchies. The empirical results showed that our approach can be used to place relevant unseen labels on the top of the ranked list of labels. In addition to the quantitative evaluation, we also analyzed label representations qualitatively via a 2D-visualization of label representations. This analysis showed that using hierarchical structures of labels allows us to assess vector representations of labels by analogical reasoning.

6 Discovering Latent Structures from Label Descriptions

6.1 Introduction

Classification is a classical task in machine learning whose goal is to assign class labels to instances based on instances' properties. This can be seen as a learning process to identify common properties in instances and to aggregate instances, which are characterized by similar properties, in the same class. That is, classes represent commonality among instances in an abstract level. Thus, we evaluate how well the classifiers generalize to *unseen instances*. In a similar sense, evaluation can also be extended to the performance of the classifiers on *unseen labels*. For the latter, however, classification algorithms cannot work well if they exploit association patterns only between instances and labels given in the training set. This is because, in classification problems, a label is often represented by one of a fixed number of discrete values. In other words, there is no way to know how unseen labels are related to seen labels. This sort of problem is often referred to as *zero-shot learning* (ZSL) where a subset of labels is associated with none of training examples, but only appears among the target labels at test time (Farhadi et al., 2009; Palatucci et al., 2009). Hence, the main question in ZSL is how we can define more meaningful labels in order to improve performance of classifiers even on unseen labels.

Recently, several approaches have been proposed to address ZSL problems by making use of additional information such as attributes of labels (Lampert et al., 2014) and their textual information such as the labels' name (Akata et al., 2015; Frome et al., 2013; Sappadla et al., 2016; Socher et al., 2013). Such information allows for classifiers to make reasonable predictions on unseen instances associated with unseen labels, without losing generalization performance. As an example, assume that we are given a classifier trained on a collection of documents about "dogs" and "cats." What if documents about "wolves" and "lions" arrive at test time? Given the fixed label set, i.e., "dogs" and "cats," the classifier may predict the label of documents about wolves as "dogs" because it is likely that the documents about "wolves" shares more terms with ones about dogs than cats. Similarly, the documents about lions will be predicted as "cats." Let us consider a slightly different scenario that "wolves" and "lions" are also used as labels to be predicted at test time even though we did not train the classifier for such labels. Defining $A \prec B$ which means A comes before B in a ranked list, we want the classifier to yield the following ranked lists of labels for the documents about wolves: "dogs" $\prec$ "cats" $\prec$ "wolves" $\prec$ "lions," "dogs" $\prec$ "wolves" $\prec$ "cats" $\prec$ "lions," or, ideally, "wolves" $\prec$ "dogs" $\prec$ "cats" $\prec$ "lions" based on the fact that "dogs" and "wolves" belong to the same family, and under the assumption that the classifier also knows such fact learned from external resources. In other words, for the documents about wolves it is reasonable that "wolves" always precedes "lions" in label ranking based on the relationship between "dogs" and "wolves."

One way that allows classifiers to learn relationships between labels and to exploit the information for making predictions for unseen labels has been introduced in (Frome et al., 2013). This approach first represents words as d-dimensional vectors. These word embeddings are learned from large textual corpora such as Wikipedia whose vocabulary includes textual descriptions for labels such as "dogs" and "cats". In turn, representations of words corresponding to label names are used instead when labels need to be considered. As the embedding space has the interesting property that words used in similar contexts have similar representations, one is able to make reasonable predictions for unseen labels even when no prior information on them is available.

Although it sheds light on an interesting direction of ZSL, it is still problematic when we consider this method on problems where textual information of labels is quite complex to be converted into words by looking up in the dictionary. To circumvent this problem, one can make the assumption that each label has its own description in textual format. Then, such descriptions can be represented by *tf-idf* as in (Elhoseiny et al., 2013). For example, "dog" in Wikipedia is described as follows:

> *The domestic dog (Canis lupus familiaris or Canis familiaris) is a domesticated canid which has been selectively bred for millennia for various behaviors, sensory capabilities, and physical attributes. ...*

Furthermore, it is worth noting that learning word representations is independent of the training data in (Frome et al., 2013). If instances are also in textual format, we may further exploit word embeddings by finding a joint space of all available information such as word sequence patterns in both instances and label descriptions, and association patterns between instances and labels.

Hence, in this chapter, we aim at learning document, label, and word representations from such textual information where labels descriptions and documents share the same word vocabulary, as well as association patterns between documents and labels. This joint learning scheme allows us to infer representations for unseen labels and to obtain better classification systems in terms of generalization performance on both unseen instances and labels.

6.2 Problem Statement

In the following we will define a set of notations which will be used throughout this chapter. Assume that we are given a vocabulary of V words $\mathcal{W} = \{1, 2, \cdots, V\}$, a set of L labels $\mathcal{C}_s = \{1, 2, \cdots, L\}$, and a set of N training examples $\mathcal{D} = \{(\mathcal{T}_n^{(x)}, \boldsymbol{y}_n)_{n=1}^N\}$ where $\mathcal{T}_n^{(x)} = \{w_1^{(x)}, w_2^{(x)}, \cdots, w_{M_n}^{(x)}\}$ denotes a sequence of M_n words $w \in \mathcal{W}$, and $\boldsymbol{y}_n = \{y_1, y_2, \cdots, y_{Q_n}\}$ a set of Q_n relevant labels $y \in \mathcal{C}_s$ for the n-th training example. Each label $y_l \in \mathcal{C}_s$ has its own description $\mathcal{T}_l^{(y)} = \{w_1^{(y)}, w_2^{(y)}, \cdots, w_{M_l}^{(y)}\}$ consisting of M_l words. Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \in \mathbb{R}^{k \times N}$, $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_L\} \in \mathbb{R}^{k \times L}$ and $\mathbf{U} = \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_V\} \in \mathbb{R}^{k \times V}$ be document, label, and word representations, respectively. For example, $\mathbf{x}_1$ corresponds to the k-dimensional vector for the document indexed 1 in $\mathcal{D}$, i.e., $\mathcal{T}_1^{(x)}$.

In this chapter, we examine our hypothesis on a multi-label text classification dataset where $|\boldsymbol{y}_n| \geq 1$ for all n. Given multiple labels per document, our task is to learn a ranking function which yields higher similarity scores between a document and its relevant labels

than ones between a document and irrelevant labels. More formally, the objective is to learn a ranking function $f : (\mathbf{x}, \mathbf{y}) \rightarrow \mathbb{R}$ such that $f\left(\mathbf{x}, \mathbf{y}_{y_i}\right) > f\left(\mathbf{x}, \mathbf{y}_{y_j}\right)$ where $y_i \in \boldsymbol{y}$ and $y_j \in \bar{\boldsymbol{y}}$.

At test time we have a set of *unseen* labels $\mathcal{C}_u = \{L+1, L+2, \cdots, L+L_u\}$ and each unseen label $y_l^* \in \mathcal{C}_u$ also has its description $\mathcal{T}_l^{(y^*)}$.

6.3 Method

In this section, we describe how to learn representations of both documents and labels jointly from their textual description in a way that a document and its relevant labels yield higher similarity scores in the joint embedding space.

6.3.1 Documents and Labels as Word Sequences

As for documents, i.e., instances represented by sequences of words, we can also deal with labels as instances of word sequences, provided they have textual descriptions. Based on the assumption that a representation of such an instance should contain *global* information on its description, one can learn fixed-size vector representations for documents and labels while learning a *local* predictor of a word given its context in the textual description (Le and Mikolov, 2014).

Given the training document set $\mathcal{K}_X = \{\mathcal{T}_n^{(x)} | 1 \leq n \leq N\}$, firstly, we show how to learn representations for a *document* and individual *words*, respectively. For convenience, we will drop n from both $\mathcal{T}_n^{(x)}$ and $\mathbf{x}_n$ when it is not confusing. Note that the document representation $\mathbf{x}$ is a set of learnable parameters as well as the word representations. The objective function is to maximize the probability of predicting a word at position t in $\mathcal{T}^{(x)}$ given its $c-1$ surrounding words and the document representation $\mathbf{x}$:

$$p(w_t | \boldsymbol{w}_{-t}, \mathbf{x}) = \frac{\exp(\mathbf{u'}_{w_t}^T \hat{\mathbf{u}}_{w_t})}{\sum_{v=1}^{V} \exp(\mathbf{u'}_v^T \hat{\mathbf{u}}_{w_t})} \tag{6.1}$$

where $\mathbf{u}'_{w_t}$ is the ck-dimensional vector for an output word w_t, and $\hat{\mathbf{u}}_{w_t}$ denotes the *context* representation of the output word, which is a concatenation of representations for the context words $\boldsymbol{w}_{-t} = \{w_{t-(c-1)/2}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+(c-1)/2}\}$ and the document representation $\mathbf{x}$ defined as

$$\hat{\mathbf{u}}_{w_t} = \left[\mathbf{x}, \mathbf{u}_{w_{t-(c-1)/2}}, \cdots, \mathbf{u}_{w_{t+(c-1)/2}}\right] \in \mathbb{R}^{ck}. \tag{6.2}$$

Here, $\hat{\mathbf{u}}_{w_t}$ can be interpreted as a combination of *global* (i.e., $\mathbf{x}$) and *local* (i.e., $\left[\mathbf{u}_{w_{t-(c-1)/2}}, \cdots, \mathbf{u}_{w_{t+(c-1)/2}}\right]$) context information of a word w_t in $\mathcal{T}^{(x)}$. Instead of using the softmax in Eq. 6.1 directly, we use its approximation, namely *negative sampling* (Mikolov et al., 2013b):

$$\begin{aligned} &\log p\left(w_t | \boldsymbol{w}_{-t}, \mathbf{x}\right) \\ &\approx \log \sigma(\mathbf{u'}_{w_t}^T \hat{\mathbf{u}}_{w_t}) + \sum_{i=1}^{\kappa} \mathbb{E}_{P_n(w)} \left[\log \sigma(\mathbf{u'}_{w_i}^T \hat{\mathbf{u}}_{w_t})\right] \end{aligned} \tag{6.3}$$

where $\sigma(x)$ is the sigmoid function, κ is the number of negative samples, and $P_n(w)$ is the unigram distribution raised to the power of $3/4$.

Then, we optimize both $\mathbf{X}$ and $\mathbf{U}$ in a way of maximizing the average log probability over all words in documents $\mathcal{K}_X$ as follows

$$\mathcal{L}_X\left(\Theta_X; \mathcal{K}_X\right) = \sum_{n=1}^{N} \frac{1}{|\mathcal{T}_n^{(x)}|} \sum_{t=1}^{|\mathcal{T}_n^{(x)}|} -\log p(w_{t,n}|\boldsymbol{w}_{-t,n}, \mathbf{x}_n) \tag{6.4}$$

where $\Theta_X = \{\mathbf{X}, \mathbf{U}, \mathbf{U}'\}$. Similarly, one can learn $\mathbf{Y}$ for the label descriptions $\mathcal{K}_Y = \{\mathcal{T}_l^{(y)} | 1 \leq l \leq L\}$ and $\mathbf{U}$:

$$\mathcal{L}_Y\left(\Theta_Y; \mathcal{K}_Y\right) = \sum_{l=1}^{L} \frac{1}{|\mathcal{T}_l^{(y)}|} \sum_{t=1}^{|\mathcal{T}_l^{(y)}|} -\log p(w_{t,l}|\boldsymbol{w}_{-t,l}, \mathbf{y}_l) \tag{6.5}$$

where $\Theta_Y = \{\mathbf{Y}, \mathbf{U}, \mathbf{U}'\}$.

6.3.2 Joint Embeddings

So far we have discussed how to learn document, label and word representations jointly from textual description of documents and labels. Once we learn the document representations $\mathbf{X}$ and the label representations $\mathbf{Y}$, they are assumed to be *global* representations for their textual description. In that case, modeling the relationship between documents and labels is disregarded. However, since our goal in multi-label classification tasks is to make relevant labels distinguishable from irrelevant labels for a given instance, we learn a ranking function to place relevant labels at the top of a ranking of labels by similarity scores w.r.t. a given instance.

Defining the $k \times k$ matrix $\mathbf{W}$, the bilinear function $f(\mathbf{x}, \mathbf{y})$ is written as

$$f(\mathbf{x}, \mathbf{y}) = \mathbf{y}^T \mathbf{W} \mathbf{x}. \tag{6.6}$$

By using the bilinear function $f(\mathbf{x}, \mathbf{y})$, we can compute the rank of $y_i \in \boldsymbol{y}$ with respect to $\mathbf{x}$ as sum of the number of incorrectly ranked pairs as follows

$$\ell\left(\mathbf{x}, y_i\right) = \sum_{y_j \in \bar{\boldsymbol{y}}} \mathbb{I}\left[f\left(\mathbf{x}, \mathbf{y}_{y_i}\right) \leq f\left(\mathbf{x}, \mathbf{y}_{y_j}\right)\right] \tag{6.7}$$

where $\mathbb{I}[\cdot]$ takes 1 if its argument is true otherwise 0.

As discussed in Section 5.2.1, one can use the *weighted approximate-rank pairwise* (WARP) loss (Weston et al., 2011), which uses an *approximation* of Eq. (6.7) given by

$$\ell_{\text{WARP}}\left(\mathbf{x}, y_i\right) = \sum_{y_v \in \mathcal{V}_{y_i}} h(r_i(\mathbf{x}))\left[m - f(\mathbf{x}, \mathbf{y}_{y_i}) + f(\mathbf{x}, \mathbf{y}_{y_v})\right]_+ \tag{6.8}$$

where $h(r_i(\mathbf{x}))$ denotes a function of the rank of positive labels y_i, $[x]_+$ returns x if $x > 0$ otherwise 0, $m \in \mathbb{R}$ denotes a margin, and $\mathcal{V}_{y_i}$ is the set of labels defined by

$$\mathcal{V}_{y_i} = \{y_n | \left(m + f(\mathbf{x}, \mathbf{y}_{y_j})\right) \geq f(\mathbf{x}, \mathbf{y}_{y_i}), \forall y_j \in \bar{\boldsymbol{y}}\}. \tag{6.9}$$

Algorithm 1: Training *AiTextML*

input : $\mathcal{D} = \{(\mathcal{T}_n^{(x)}, \boldsymbol{y}_n)_{n=1}^N\}$, $\mathcal{K}_Y = \{\mathcal{T}_l^{(y)} | 1 \leq l \leq L\}$
output: $\Theta = \{\mathbf{U}, \mathbf{U}', \mathbf{X}, \mathbf{Y}, \mathbf{W}\}$

do
  for $n = 1$ **to** N **do**
    $\mathcal{V}^* \leftarrow \emptyset$ `// violation labels set`
    foreach $y_i \in \boldsymbol{y}_n$ **do**
      $S \leftarrow 0$
      $pos \leftarrow f(\mathbf{x}_n, \mathbf{y}_{y_i})$
      do
        $S \leftarrow S + 1$
        pick y_j from $\{1, \cdots, L\} \setminus \boldsymbol{y}_n$ at random
        $neg \leftarrow f(\mathbf{x}_n, \mathbf{y}_{y_j})$
        if $m + neg \geq pos$ **then**
          $\mathcal{V}^* \leftarrow \mathcal{V}^* \cup y_n$
          update Θ_J using Eq. 6.11
          break
      while $m + neg \leq pos$ *and* $S < L - |\boldsymbol{y}_n|$
    foreach $w_t \in |\mathcal{T}_n^{(x)}|$ **do**
      update Θ_X using Eq. 6.4
    foreach $l \in \{\boldsymbol{y}_n \cup \mathcal{V}^*\}$ **do**
      foreach $w_t \in |\mathcal{T}_l^{(y)}|$ **do**
        update Θ_Y using Eq. 6.5
while *until termination conditions are met*

The rank of the positive label y_i can be approximated by

$$h(r_i(\mathbf{x})) \approx \left\lfloor \frac{L - |\boldsymbol{y}|}{S} \right\rfloor \tag{6.10}$$

where S is the number of samples drawn uniformly from $\bar{\boldsymbol{y}}$ until a label $y_v \in \mathcal{V}_{y_i}$ is sampled. Thus, the objective to learn embeddings in a joint space is given by

$$\mathcal{L}_w\left(\Theta_J; \mathcal{D}\right) = \sum_{n=1}^{N} \frac{1}{|\boldsymbol{y}_n|} \sum_{y_i \in \boldsymbol{y}_n} \left\lfloor \frac{L - |\boldsymbol{y}_n|}{S} \right\rfloor \left[m - f(\mathbf{x}, \mathbf{y}_{y_i}) + f(\mathbf{x}, \mathbf{y}_{y_v})\right]_+ . \tag{6.11}$$

6.3.3 Putting It All Together

Our goal is to learn representations for documents, labels, and words, which are all in textual format, jointly to improve the generalization performance of our proposed method to unseen labels as well as to seen ones on multi-label text classification datasets. We call this method All-in Text Multi-label Learner (*AiTextML*). The goal is achieved by combining the losses

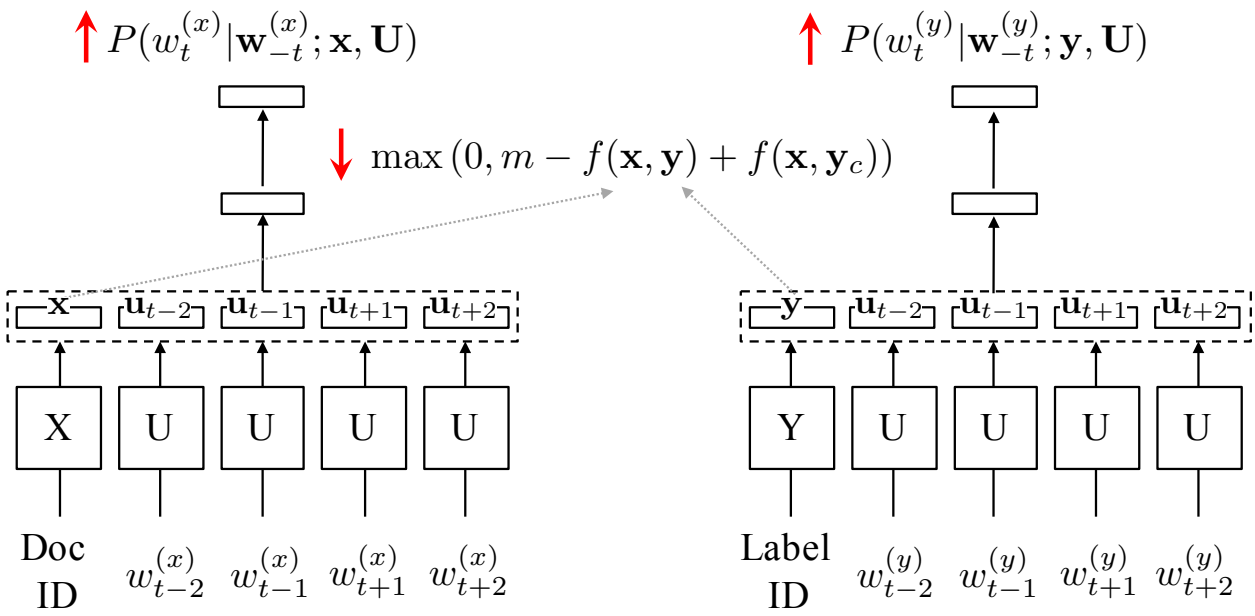


Figure 6.1: Illustration of *AiTextML*. Given a pair of a document and its relevant label, the objective of *AiTextML* is to minimize the pairwise hinge loss $max(0, m - f(\mathbf{x},\mathbf{y}) + f(\mathbf{x},\mathbf{y}_c))$, where $\mathbf{y}$ denotes a relevant label's embedding and $\mathbf{y}_c$ is a irrelevant label's embedding, while maximizing the probability of predicting a next word w_t given its context $\boldsymbol{w}_{-t}$ from documents $\mathcal{T}^{(x)}$ and label descriptions $\mathcal{T}^{(y)}$.

regarding document and label representations from word sequences in Eqs. 6.4 and 6.5, and the WARP loss, i.e., Eq. 6.11. Thus, the objective is

$$\begin{aligned} \mathcal{L}\left(\Theta; \mathcal{D}, \mathcal{K}_Y\right) &= \alpha\mathcal{L}_w + \beta\mathcal{L}_X + \gamma\mathcal{L}_Y \\ &\text{s.t. } \alpha + \beta + \gamma = 1 \end{aligned} \tag{6.12}$$

where $\Theta = \{\mathbf{U}, \mathbf{U}', \mathbf{X}, \mathbf{Y}, \mathbf{W}\}$ denotes the set of parameters which are randomly initialized, and the control parameters α, β, γ determine the impact of the WARP loss $\mathcal{L}_w$ and the representation learning losses $\mathcal{L}_X$ and $\mathcal{L}_Y$ to the total loss $\mathcal{L}$. Figure 6.1 illustrates *AiTextML*. We use stochastic gradient descent (SGD) with a fixed learning rate η for all time steps τ to update the parameters Θ given a training example indexed n at a time. The pseudo-code of our proposed method is shown in Algorithm 1.

6.3.4 Inference on Unseen Documents and Labels

As shown in the previous sections, our proposed method needs document and label representations to be estimated as parameters from word sequences. The same holds for unseen data points at test time. Consider that we are given a test set $\mathcal{D}^* = \{(\mathcal{T}_{x^*}^{(n)}, \mathcal{Y}^{*(n)})\}_{n=1}^{N_t}$, and that some of labels do not appear in the training set such that $y_{(\cdot)}^* \in \{L+1, L+2, \cdots, L+L_u\}$ where L_u is the number of unseen labels. To make predictions on unseen documents w.r.t. unseen labels as well, we initialize $\mathbf{X}^* = \{\mathbf{x}_1^*, \mathbf{x}_2^*, \cdots, \mathbf{x}_{N_t}^*\}$ and $\mathbf{Y}^* = \{\mathbf{y}_1^*, \mathbf{y}_2^*, \cdots, \mathbf{y}_{L_u}^*\}$ randomly for unseen documents and labels, respectively. In turn, we define only $\mathbf{X}^*$ and $\mathbf{Y}^*$ as trainable parameters for the *AiTextML* model on the test set $\mathcal{D}^*$ while all the other parameters $\{\mathbf{U}, \mathbf{U}', \mathbf{X}, \mathbf{Y}, \mathbf{W}\}$ are kept fixed as shown in Fig. 6.2. At inference time, we use

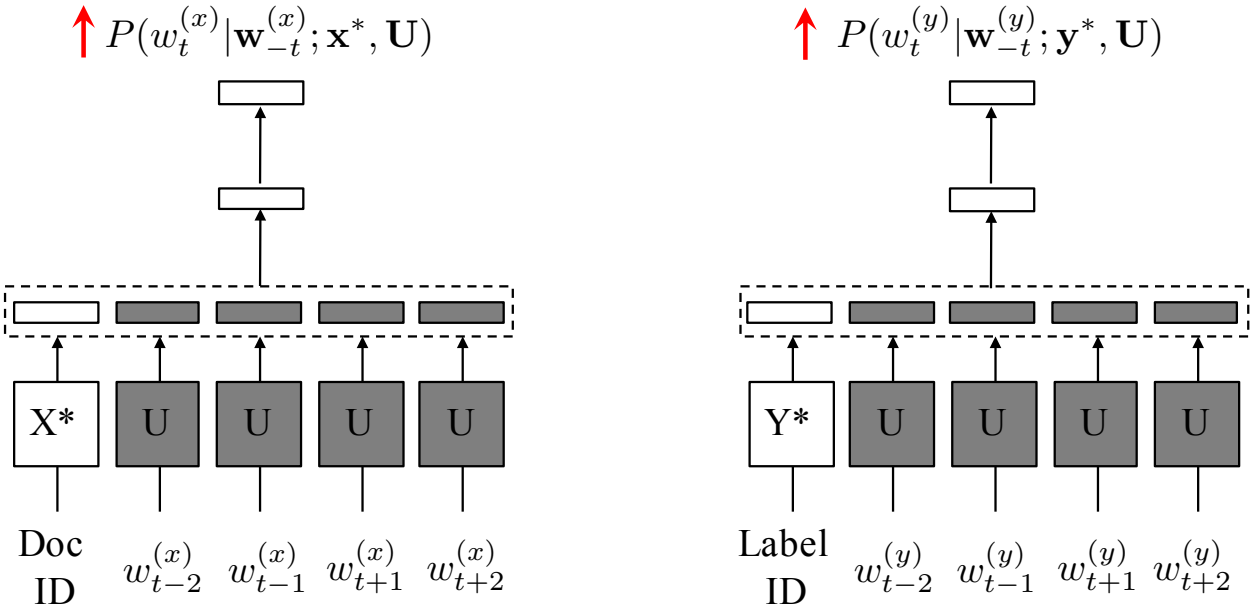


Figure 6.2: During the inference stage, embeddings for test documents $\mathbf{X}^*$ and unseen labels $\mathbf{Y}^*$ are randomly initialized and then tuned in a way of maximizing the probability of a next word given its context words whereas the rest of the parameters (in gray) are fixed.

Table 6.1: Statistics of the BioASQ dataset

# training examples (N)	6,692,815
# validation examples (N_v)	100,000
# test examples (N_t)	4,912,719
# words (V)	528,156
# *seen* labels (L)	23,669
# *unseen* labels (L_u)	2,435
Avg. # of relevant seen labels per training example	10.83
# test examples that have unseen labels	432,703
Avg. ratio of relevant unseen labels in the test set	10.31%

the same control parameters β, γ and number of parameter updates used in the training phase. To prevent learning $\mathbf{X}^*$ and $\mathbf{Y}^*$ from document-label association patterns in $\mathcal{D}^*$, we set α to 0.

Note that as unseen document representations $\mathbf{x}^*$ and unseen label representations $\mathbf{y}^*$ are independent of each other, we can easily parallelize this inference stage.

6.4 Experimental Setup

6.4.1 Dataset

We use the BioASQ Task 3a dataset, a collection of scientific publications in biomedical research, to examine our proposed method.[1] It contains about 12 million publications, each of which is associated with around 11 descriptors on average out of 27,455, which come from the Medical Subject Headings (MeSH) hierarchy.[2] We removed 1003 descriptors from the MeSH hierarchy because they do not have textual descriptions as well as 348 descriptors not appearing in the BioASQ Task 3a dataset. We split the dataset by year so that the training set includes all papers by 2004 and the rest of papers published between 2005 and 2015 belongs to the test set. Thus, descriptors introduced to the MeSH hierarchy after 2004 can be considered as unseen labels. 100,000 papers before 2005 were randomly sampled and set aside as the validation set for tuning hyperparameters. Since we split the dataset by year, 2,435 labels in the test set do not appear in the training set. About 10% of test examples contain such unseen labels in their target label set. The ratio of unseen labels in the target label set of the test data is 10.31%.

We applied minimal preprocessing to documents and label descriptions; tokenization and replacement of numbers and rare words to special tokens, e.g., *NUM* and *UNK*. The word vocabularies were built according to the word frequency in the training documents, for which words occurring more than 10 times were chosen. The statistics on the dataset used are summarized in Table 6.1.

6.4.2 Baseline

Since no work has been reported yet in this line of research to our best knowledge, we compare *AiTextML* with the same model using fixed $\gamma = 0$ in Eq. 6.12. That is, our baseline also optimizes the WARP loss. However, our baseline considers learning representation of documents and words simultaneously, whereas Wsabie in (Weston et al., 2011) uses fixed feature representations for instances. Hence, our baseline is also able to learn feature representations and can be seen as an extension of Wsabie. Unlike conventional multi-label learning algorithms, Wsabie scales well on large-scale datasets in terms of both the number of training examples and labels, and performs comparably even in standard benchmark datasets for multi-label text classification (Nam et al., 2015).

6.5 Experiments

We used the validation set to set our hyperparameters as follows: the number of negative samples $\kappa = 5$, the dimensionality of all representations 100, the size of the context window $c = 5$, learning rate $\eta = 0.025$, margin $m = 0.1$, and the control variables $\alpha = 1/3, \beta = 1/3, \gamma = 1/3$. For the baseline, different control parameters $\alpha = 1/3, \beta = 2/3, \gamma = 0$ were used, but the rest of the hyperparameters were same with our proposed method. Unless we specify otherwise, the hyperparameter settings are used throughout all experiments. In order to prevent overfitting, we impose constraints on norm of document, label and word vectors such that $\|\mathbf{u}_i\|_2 \leq 1, i \in \{1, \cdots, V\}$, $\|\mathbf{x}_d\|_2 \leq 1, d \in \{1, \cdots, N\}$, and

[1] http://www.bioasq.org/participate/data
[2] https://www.nlm.nih.gov/mesh/introduction.html

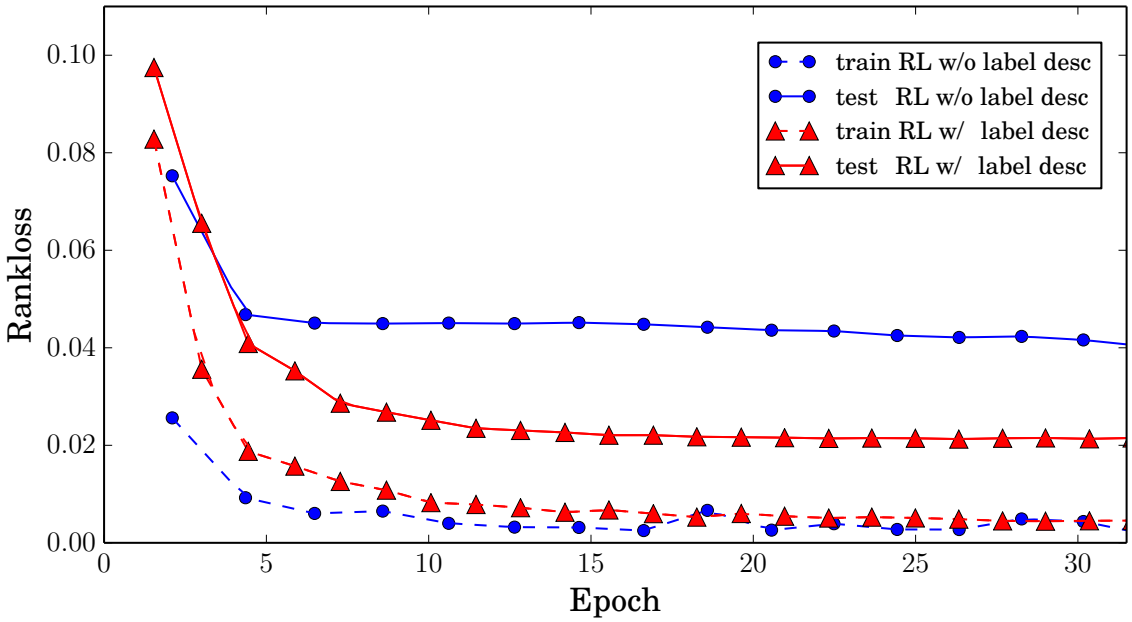


Figure 6.3: Effect of learning from label descriptions in terms of rank loss on the BioASQ dataset w.r.t. the *seen* labels. The rank loss was estimated on randomly sampled 10,000 training examples and on a fixed subset of 10,000 test examples every 60 mins in the course of training, indicated by markers.

Table 6.2: Comparison of *AiTextML* to the baseline w.r.t. *seen* labels. The *AiTextML* model was trained for the same amount of time (24 hrs) as the baseline. The numbers in the parentheses following the methods correspond to the control parameters (α, β, γ) in Eq. 6.12.

	RL	AvgPr	OneErr
Baseline $\left(\frac{1}{3}, \frac{2}{3}, 0\right)$	0.05217	**0.36645**	0.41728
AiTextML $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$	**0.03544**	0.32786	**0.25992**

$\|\mathbf{y}_l\|_2 \leq 1, l \in \{1, \cdots, L\}$. We performed all experiments on a machine with two Intel Xeon E5-2670 CPUs and 32GB of memory.

6.5.1 Effect of Label Descriptions

We carried out experiments to compare the models which learns purely from the association patterns and the other which learn from label descriptions as well as the association patterns. As can be seen in Fig. 6.3, learning from label descriptions improves the generalization performance of our method. Indeed, rank loss on the training set of the model without learning from label descriptions is even lower than that of the model trained on label descriptions. In contrast to the baseline, *AiTextML* achieves better rank loss scores on the test set. This shows that label descriptions help *AiTextML* prevent from overfitting. Since *AiTextML* learns label representations not only from the association patterns, but also

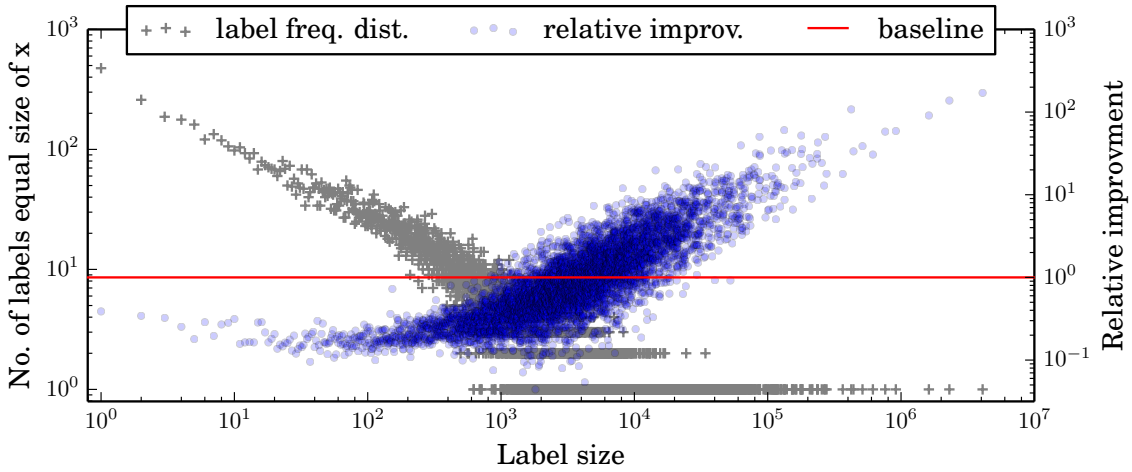


Figure 6.4: Label frequency distribution and relative improvement over the baseline with respect to label size.

textual description of labels, it takes more time for a single iteration indeed under the same hyperparameter settings.

Once having trained *AiTextML* and the baseline for 50 epochs, we evaluated two models on the full set of test examples. We observed that *AiTextML* outperforms substantially the baseline in terms of rank loss and one-error, which tells us learning from label descriptions plays an important role for the improvements. However, AvgPr of our proposed method rather decreases compared to the baseline. Note that our objective measure in the optimization corresponds to ranking. The results are shown in Table 6.2. It is often the case that label frequency distribution in real world multi-label text datasets follows a *power law* as shown in Fig. 6.4, which means, informally, there are few frequent labels, but many infrequent ones. This property makes it difficult for classifiers to generalize well to unseen instances if they have rare labels in their target labels since a classifiers tend to overfit rare labels.

In order to take a closer look at the source of improvements, we comapared both the baseline and our proposed method in terms of AvgRank. Fig. 6.4 shows that *AiTextML* performs significantly better than the baseline for frequent labels, whereas its performance on rare labels is worse than the baseline. Our model learns more often from descriptions of frequent labels in a way that their representations are effective in predicting a next word given its context and maximizing similarity scores to the documents that they belong to as well. Due to the fact that *AiTextML* focuses more on frequent labels, average ranks rare labels are rather ignored which results in lower average precision.

6.5.2 Unseen Label Representations

We demonstrate the quality of unseen label representations by listing nearest neighbors in both *seen* and *unseen* label spaces to selected unseen labels, shown in Table 6.3. For example, given a query "Tundra," we have "Genetic Speciation," "Biological Extinction," and "Wetlands" as similar labels from the seen label set, which are somehow related to environmental danger in the tundra. "Grassland" from the unseen label set is another type of biomes which is often used to contrast different characteristics of "Tundra." "Permafrost" and "Ponds" are also related labels to "Tundra" when a paper discusses climate changes and

Table 6.3: Nearest neighbors for given *unseen* labels in seen and unseen label representations.

Query label	Seen labels	Unseen labels
Tundra	Genetic Speciation	Grassland
	Arcidae	Permafrost
	Secernentea	Click Chemistry
	Biological Extinction	Ponds
	Wetlands	Cambium
Night Vision	Halorhodopsins	Retinal Photoreceptor Cell Outer Segment
	Fluorophotometry	Mesopic Vision
	Arthropod Compound Eye	Plant Photoreceptors
	Retinoscopes	Rod-Cone Interaction
	Color Vision	Bleaching Agents
Hope	Adult Children	Time-to-Treatment
	World War II	Anatomists
	Healthy Volunteers	Pragmatic Clinical Trials as Topic
	World War I	Secondary Care
	Health Status Disparities	Historically Controlled Study

their effects in the tundra. Such relationships can be also found for the unseen label "Night Vision."

In contrast, there is no clear relationship between the unseen query label "Hope" and both seen and unseen labels. This is because such a label has a very short description and unclear terms are used in the description. For example, "Hope" is described as "Belief in a positive outcome."

To understand label embeddings, we projected all embeddings for both seen and unseen labels onto 2D space by using *t-distributed stochastic neighbor embedding* (tSNE) (Van der Maaten and Hinton, 2008). A sub-region of the space is shown in Fig. 6.5. The label "Health_Level_Seven" is an unseen label and such a label embedding can be obtained at the inference stage (Section 6.3.4). During the inference stage, the following description is used:

> *an american national standards UNK organization working on specifications to support development and advancement of clinical and administrative standards for health care .*

Given the above description of "Health_Level_Seven", one may see connections between the unseen label and other seen labels.

6.5.3 Zero-Shot Prediction

One of the promising aspects of our proposed method is the capability of learning unseen label representations from their descriptions. About 400,000 test examples have 1∼2 unseen labels in their target label sets on average as shown in Table 6.1. Without using the inference step and the joint space embedding, a reasonably straightforward solution to obtain unseen label representations is averaging embeddings of words which occur in textual description

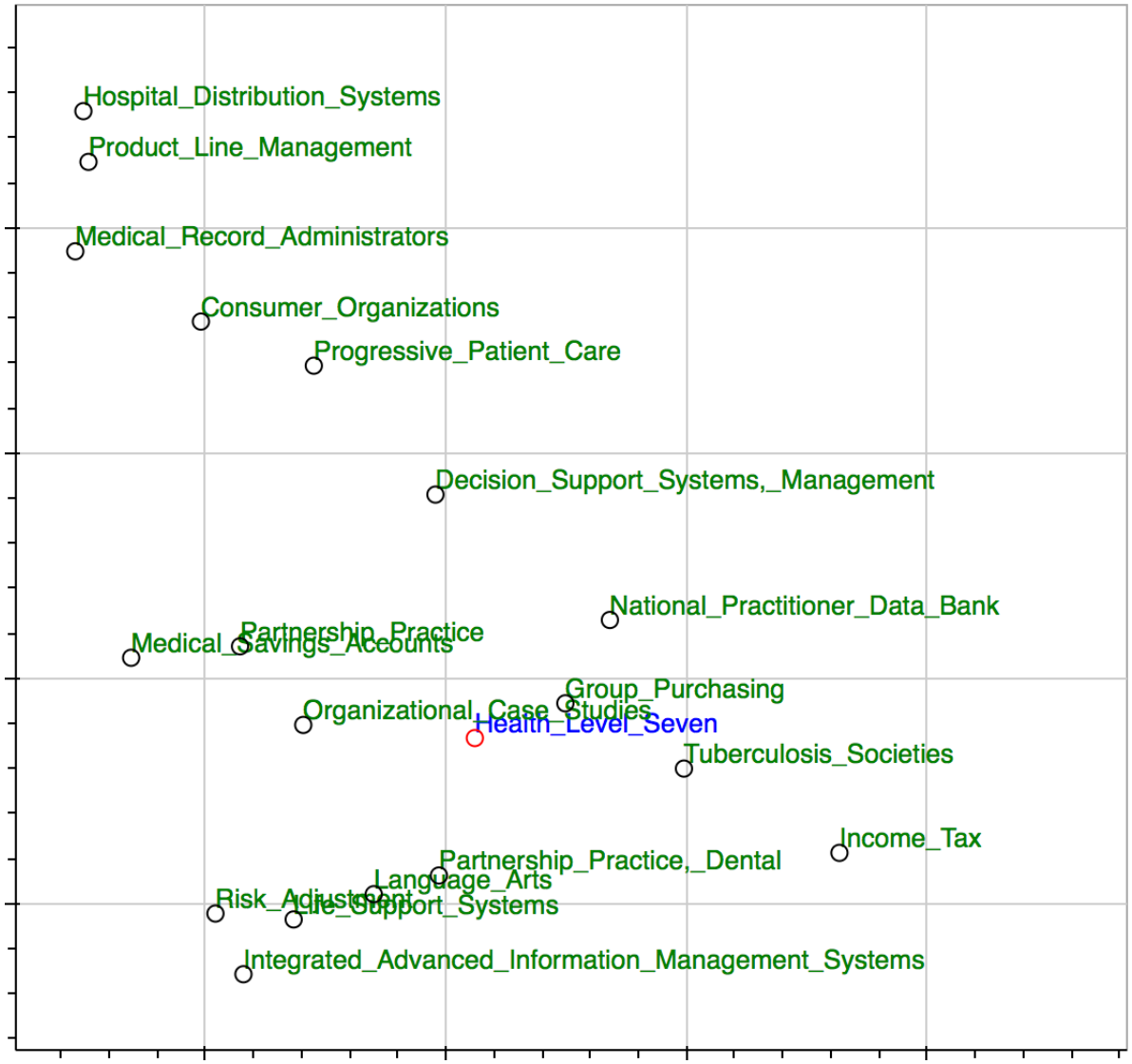


Figure 6.5: "Health_Level_Seven" (in blue) is a unseen label while all other labels (in green) are seen labels.

of labels including their name. For label names, we applied the same preprocessing pipeline used for the documents. For example, if we have an unseen label "1918-1919 Influenza Pandemic," it is replaced with "NUM-NUM influenza pandemic" and then its representation is determined by the averaged representations of three words "NUM-NUM," "influenza," and "pandemic." We use a special token "UNK" when a word cannot be found in the vocabulary. Also, the norm of unseen label representations is scaled to 1. Instead of learning such word embeddings independently of our task, we used word embeddings of the baseline and *AiTextML* in Sec. 6.5.1. Note that our baseline has the same architecture and number of parameters for *AiTextML*, but does not learn from label descriptions.

We compare the proposed method with four possible combinations of two word embeddings from the baseline and *AiTextML*, and two textual information sources to be used for representing unseen labels, i.e., names and descriptions. As can be seen in Table 6.4, *AiTextML*, which *infers* unseen label representations from textual descriptions, outperforms the baseline models for estimating unseen label representations by averaging over representa-

Table 6.4: Comparison of *AiTextML*, which represents unseen labels by the inference step, to averaging of embeddings for words in label names or descriptions on the zero-shot task. For averaging words in the textual information, we use the word embeddings from the baseline and the *AiTextML* model.

	RL	AvgPr	OneErr
Baseline avg (names)	0.50225	0.00317	0.99969
Baseline avg (desc.)	0.48812	0.00375	0.99946
AiTextML avg (names)	0.52335	0.00290	0.99979
AiTextML avg (desc.)	0.52890	0.00388	0.99941
AiTextML inf (desc.)	**0.21622**	**0.02665**	**0.98608**

tions for words appearing in either label names or descriptions. Moreover, using the averaged word embeddings from label descriptions does not achieve relevant improvements over using only the label names. In other words, when we consider the word embeddings to obtain unseen label representations, using label descriptions seems to be a better choice than label names. However, the gain is not comparable to what our proposed method achieves. This shows that the inference step for unseen label representations in our proposed method plays an important role for yielding more useful information than given by the average of word embeddings in this task.

6.6 Discussion

We have presented a framework for learning document, label, and word representations jointly to leverage shared information available in textual format. This allows not only to make better predictions w.r.t. seen labels, but also produces better representations for unseen labels in a zero-shot learning setting. In particular, we could show that our methods outperforms a baseline approach which simply averages representations of all words in either the label names or the label descriptions.

Our objective in this work is to jointly learn document, label and word representations to exploit shared information, and we demonstrated *AiTextML* only on textual data. However, we note that the label representation learning part can be also applied to other domains such as object classification in images under the ZSL setting instead of defining attributes for unknown labels. A major limitation when considering our proposed method in learning label representations is the availability of label descriptions. If a dataset does not have such label descriptions, one can make use of external knowledge resources such as Wikipedia to construct the label description set. For example, the first sentence or paragraph in Wikipedia articles contain very general terms for describing facts of interest.

Finally, we would like to highlight the key differences between our proposed method and the approaches where label names are used to obtain unseen label representations. The principle of *AiTextML* is more general because we can easily and efficiently add representations for unseen labels to the model by the inference step under the assumption that label descriptions consist of general terms. If words in label names are out of the vocabulary, we need to handle them more carefully because label names are rather short in general and such information

loss occur frequently, which often leads to inaccurate unseen label representations in the ZSL task. Furthermore, whereas label representations by using their names provide only a good starting point for label embeddings, the proposed method allows us to obtain improved label rankings on test instances as well by learning all representations jointly in conjunction with label descriptions in the whole training process (Loza Mencía et al., 2016).

7 Conclusions and Future Work

Multi-label classification (MLC) is the problem of assigning multiple labels to a single instance, so the goal of MLC includes learning not only a mapping function from instances to individual labels but also how labels are related.

Among several ways of evaluating the multi-label classification performance, the rank loss has been used widely because it measures the quality of label rankings by pairwise comparisons. For MLC, neural networks with a surrogate objective function of the rank loss have also been used although the computational costs at the output layer increase quadratically with the number of labels. In addition to their computational complexity, it has been shown in prior work that pairwise comparisons are unnecessary for rank loss optimization. Thus, we have presented a rather simpler neural network architecture whose complexity at the output layer grows only linearly in the number of labels, which means that we can achieve better performance in terms of the rank loss on large-scale MLC problems when ignoring label dependence.

That being said, it is important that MLC methods make use of dependencies in the label space so as to obtain better predictive performance in terms of evaluation measures that consider label dependence. In other words, the rank loss is an evaluation measure that prefers MLC methods that train classifiers independently per label over more complex ones which take label dependence into consideration. For Hamming loss optimization, we can utilize the same classifiers used to optimize the rank loss.

In contrast to the rank loss and Hamming loss, the subset 0/1 loss measures how well MLC methods predict multiple labels as a set, so that modeling dependencies among labels in a label subset is the ultimate goal of training MLC methods in this regard. As the possible number of label subsets grows exponentially with respect to the number of unique labels, we need methods that learn efficiently from a large number of label subsets. A recurrent neural network has been used in this thesis as a means of minimizing the subset 0/1 loss. We have empirically shown that recurrent neural network-based MLC methods perform better than label powerset and classifier chain approaches on multiple multi-label text datasets in terms of several evaluation measures including the subset 0/1 loss. In fact, recurrent neural networks predict a set of relevant labels one by one in a sequential manner, so that we have presented several label ordering strategies. We have also demonstrated in our experiments that label ordering affects the performance of our recurrent neural networks.

Many MLC methods mainly aim to learn only from statistical patterns on a label space given input instances. However, in some cases we have additional information on label spaces such as pairwise relationships between two labels that can be represented in a graph. Such additional information enables to make predictions with respect to unseen labels that have no training information as well as to improve the generalization performance with respect to labels that have been observed during the training phase, i.e., unseen labels.

We have proposed a method learning a joint space of instances and labels where label hierarchies are respected as well as label cooccurrence patterns. If the method learns the joint space from only label association patterns, an instance on that space would be placed

near its relevant labels that have training information. The use of label structures given a priori allows to infer unseen labels' representations at test time, so that we were able to have MLC systems that make reasonable predictions with respect to unseen labels without losing overall performance on the entire label space. To better understand the effects of label structures we have also analyzed label spaces learned from both label structures and label subsets.

We have shown that label structures are useful to build MLC systems that have the capability to make predictions with respect to unseen labels. In other words, it is difficult to use the above idea unless we are given the structure. For MLC problems with many labels, it might be very expensive to maintain such label structures. Also the performance of the joint space learning method may depend on the quality of the structures. Thus, we need another type of information on label spaces that is more easy to acquire than label structures built by human experts.

In this work we have presented another joint space learning method that exploits textual description of labels instead of label structures. Under the assumption that documents and labels share a vocabulary of words on multi-label text classification datasets, we learned document and label representations from words as well as relationships between documents and their relevant labels. Similar to the joint space learning method that uses label structures, we were also able to make predictions with respect to unseen labels. We have observed that unseen label representations estimated from their descriptions are close to seen label representations on the joint space. Furthermore, we have achieved much better performance in terms of frequent labels than the baseline method which does not take label descriptions into consideration.

Recall that we need to determine how to sort labels to train recurrent neural networks for subset $0/1$ loss minimization. One interesting future direction is to learn label ordering strategies from data (Nam et al., 2019).

In recent years, the size of MLC problems has grown rapidly, so that there are several public benchmark datasets that consist of several millions instances and hundreds of thousands of labels. As traditional MLC approaches have less focused on scalability problems, in particular on the aforementioned datasets, it is highly desirable that MLC methods handle problems with even millions of labels under time and resource constraints.

When considering MLC problems with extremely large numbers of labels, a problem often referred to as extreme multi-label classification, label-based F_1-measure maximization is often preferred to subset accuracy maximization because it is less susceptible to the very large number of label combinations and imbalanced label distributions. Label frequencies, in fact, are highly skewed in extreme multi-label classification so that we may have lots of rare labels though the proportion of rare labels is quite low. If we want to have a MLC system that performs well with respect to rare labels, for instance, then label-based macro F_1-measure needs to be taken into account. Though the recurrent neural networks that optimize the subset $0/1$ loss per instance work also well in terms of that measure, their performance might be suboptimal because the macro F_1-measure is a non-decomposable evaluation measure in terms of instances.

Bibliography

Akata, Z., Reed, S., Walter, D., Lee, H., and Schiele, B. (2015). Evaluation of output embeddings for fine-grained image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2927–2936.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*.

Balikas, G., Partalas, I., Ngomo, A. N., Krithara, A., and Paliouras, G. (2014). Results of the BioASQ track of the question answering lab at CLEF 2014. In *Working Notes for CLEF 2014 Conference*, pages 1181–1193.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, pages 153–160.

Bhatia, K., Jain, H., Kar, P., Varma, M., and Jain, P. (2015). Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pages 730–738.

Bi, W. and Kwok, J. T. (2011). Multilabel classification on tree- and DAG-structured hierarchies. In *Proceedings of the International Conference on Machine Learning*, pages 17–24.

Bi, W. and Kwok, J. T. (2013). Efficient multi-label classification with many labels. In *Proceedings of the International Conference on Machine Learning*, pages 405–413.

Calauzènes, C., Usunier, N., and Gallinari, P. (2012). On the (non-)existence of convex, calibrated surrogate losses for ranking. In *Advances in Neural Information Processing Systems*, pages 197–205.

Cesa-Bianchi, N., Gentile, C., and Zaniboni, L. (2006). Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research*, 7:31–54.

Chekina, L., Gutfreund, D., Kontorovich, A., Rokach, L., and Shapira, B. (2013). Exploiting label dependencies for improved sample complexity. *Machine Learning*, 91(1):1–42.

Chen, Y.-N. and Lin, H.-T. (2012). Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems*, pages 1529–1537.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734.

Chung, J., Cho, K., and Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1693–1703.

Crammer, K. and Singer, Y. (2003). A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*, 3:1025–1058.

Dembczyński, K., Cheng, W., and Hüllermeier, E. (2010). Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the International Conference on Machine Learning*, pages 279–286.

Dembczyński, K., Kotłowski, W., and Hüllermeier, E. (2012a). Consistent multilabel ranking through univariate losses. In *Proceedings of the International Conference on Machine Learning*, pages 1319–1326.

Dembczyński, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2012b). On label dependence and loss minimization in multi-label classification. *Machine Learning*, 88(1-2):5–45.

Dembczyński, K., Waegeman, W., and Hüllermeier, E. (2012). An analysis of chaining in multi-label classification. In *Proceedings of the European Conference on Artificial Intelligence*, pages 294–299.

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R., and Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1370–1380, Baltimore, Maryland.

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634.

Doppa, J. R., Yu, J., Ma, C., Fern, A., and Tadepalli, P. (2014). HC-Search for multi-label prediction: An empirical study. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Elhoseiny, M., Saleh, B., and Elgammal, A. (2013). Write a classifier: Zero shot learning using purely textual descriptions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2584–2591.

Elisseeff, A. and Weston, J. (2001). A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems*, pages 681–687.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Farhadi, A., Endres, I., Hoiem, D., and Forsyth, D. (2009). Describing objects by their attributes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1785.

Frome, A., Corrado, G. S., Shlens, J., Bengio, S., Dean, J., Ranzato, M., and Mikolov, T. (2013). Devise: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, pages 2121–2129.

Fürnkranz, J., Hüllermeier, E., Loza Mencía, E., and Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153.

Fürnkranz, J. and Sima, J. F. (2010). On exploiting hierarchical label structure with pairwise classifiers. *SIGKDD Explorations*, 12(2):21–25.

Gao, W. and Zhou, Z.-H. (2013). On the consistency of multi-label learning. *Artificial Intelligence*, 199–200:22–44.

Ghamrawi, N. and McCallum, A. (2005). Collective multi-label classification. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 195–200.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics, JMLR W&CP*, pages 315–323.

Hersh, W., Buckley, C., Leone, T. J., and Hickam, D. (1994). Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th Annual International ACM SIGIR Conference*, pages 192–201.

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

Hirschmann, F., Nam, J., and Fürnkranz, J. (2016). What makes word-level neural machine translation hard: A case study on english-german translation. In *Proceedings of the International Conference on Computational Linguistics*, pages 3199–3208.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hotelling, H. (1936). Relations between two sets of variates. *Biometrika*, 28(3/4):321–377.

Hsu, D., Kakade, S., Langford, J., and Zhang, T. (2009). Multi-label prediction via compressed sensing. In *Advances in Neural Information Processing Systems*, volume 22, pages 772–780.

Jasinska, K., Dembczynski, K., Busa-Fekete, R., Pfannschmidt, K., Klerx, T., and Hüllermeier, E. (2016). Extreme F-measure maximization using sparse probability estimates. In *Proceedings of the International Conference on Machine Learning*, pages 1435–1444.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (2015). On using very large target vocabulary for neural machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, pages 1–10.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*, pages 137–142.

Joachims, T. (2006). Training linear svms in linear time. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–226.

Kapoor, A., Viswanathan, R., and Jain, P. (2012). Multilabel classification using bayesian compressed sensing. In *Advances in Neural Information Processing Systems*, pages 2645–2653.

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 181–184.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., and Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of the International Conference on Machine Learning*, pages 1378–1387.

Kumar, A., Vembu, S., Menon, A. K., and Elkan, C. (2013). Beam search algorithms for multilabel learning. *Machine Learning*, 92(1):65–89.

Lampert, C., Nickisch, H., and Harmeling, S. (2009). Learning to detect unseen object classes by between-class attribute transfer. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 951–958.

Lampert, C. H., Nickisch, H., and Harmeling, S. (2014). Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the International Conference on Machine Learning*, pages 1188–1196.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, pages 9–48. Springer, Berlin, Heidelberg.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397.

Li, C., Wang, B., Pavlu, V., and Aslam, J. A. (2016). Conditional Bernoulli mixtures for multi-label classification. In *Proceedings of the International Conference on Machine Learning*, pages 2482–2491.

Li, N. and Zhou, Z.-H. (2013). Selective ensemble of classifier chains. In Zhou, Z.-H., Roli, F., and Kittler, J., editors, *Multiple Classifier Systems*, volume 7872, pages 146–156. Springer Berlin Heidelberg.

Lin, Z., Ding, G., Hu, M., and Wang, J. (2014). Multi-label classification via feature-aware implicit label space encoding. In *International Conference on Machine Learning*, pages 325–333.

Liu, T.-Y., Yang, Y., Wan, H., Zeng, H.-J., Chen, Z., and Ma, W.-Y. (2005). Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explorations*, 7(1):36–43.

Liu, W. and Tsang, I. (2015). On the optimality of classifier chain for multi-label classification. In *Advances in Neural Information Processing Systems*, pages 712–720.

Loza Mencía, E., de Melo, G., and Nam, J. (2016). Medical concept embeddings via labeled background corpora. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 4629–4636.

Loza Mencía, E. and Fürnkranz, J. (2008). Pairwise learning of multilabel classifications with perceptrons. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2899–2906.

Luong, M.-T. and Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1054–1063.

Mena, D., Montañés, E., Quevedo, J. R., and Del Coz, J. J. (2015). Using A* for inference in probabilistic classifier chains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 3707–3713.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Mnih, A. and Hinton, G. E. (2009). A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems*, pages 1081–1088.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, pages 246–252.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning*, pages 807–814.

Nam, J., Kim, J., Loza Mencía, E., Gurevych, I., and Fürnkranz, J. (2014). Large-scale multi-label text classification–revisiting neural networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 437–452.

Nam, J., Kim, Y.-B., Loza Mencía, E., Park, S., Sarikaya, R., and Fürnkranz, J. (2019). Learning context-dependeent label permutations for multi-label classification. In *Proceedings of the International Conference on Machine Learning*, pages 4733–4742.

Nam, J., Loza Mencía, E., and Fürnkranz, J. (2016). All-in text: Learning document, label, and word representations jointly. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1948–1954.

Nam, J., Loza Mencía, E., Kim, H. J., and Fürnkranz, J. (2015). Predicting unseen labels using label hierarchies in large-scale multi-label learning. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 102–118.

Nam, J., Loza Mencía, E., Kim, H. J., and Fürnkranz, J. (2017). Maximizing subset accuracy with recurrent neural networks in multi-label classification. In *Advances in neural information processing systems*, pages 5413–5423.

Palatucci, M., Pomerleau, D., Hinton, G. E., and Mitchell, T. M. (2009). Zero-shot learning with semantic output codes. In *Advances in Neural Information Processing Systems*, pages 1410–1418.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 1310–1318.

Prabhu, Y. and Varma, M. (2014). Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 263–272.

Read, J., Martino, L., and Luengo, D. (2014). Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, 47(3):1535 – 1546.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2009). Classifier chains for multi-label classification. In Buntine, W., Grobelnik, M., Mladenić, D., and Shawe-Taylor, J., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 254–269, Berlin, Heidelberg. Springer Berlin Heidelberg.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359.

Recht, B., Re, C., Wright, S., and Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701.

Roth, D. (2017). Incidental supervision: Moving beyond supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4885–4890.

Rousu, J., Saunders, C., Szedmák, S., and Shawe-Taylor, J. (2006). Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, 7:1601–1626.

Rubin, T. N., Chambers, A., Smyth, P., and Steyvers, M. (2012). Statistical topic models for multi-label document classification. *Machine Learning*, 88(1-2):157–208.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Sappadla, P. V., Nam, J., Loza Mencía, E., and Fürnkranz, J. (2016). Using semantic similarity for multi-label zero-shot classification of text documents. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 423–428.

Schapire, R. E. and Singer, Y. (2000). BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168.

Senge, R., Del Coz, J. J., and Hüllermeier, E. (2014). On the problem of error propagation in classifier chains for multi-label classification. In *Data Analysis, Machine Learning and Knowledge Discovery*, pages 163–170. Springer.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725.

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

Socher, R., Ganjoo, M., Manning, C. D., and Ng, A. (2013). Zero-shot learning through cross-modal transfer. In *Advances in Neural Information Processing Systems*, pages 935–943.

Solla, S. A., Levin, E., and Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Systems*, 2(6):625–640.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Sucar, L. E., Bielza, C., Morales, E. F., Hernandez-Leal, P., Zaragoza, J. H., and Larrañaga, P. (2014). Multi-label classification with bayesian network-based chain classifiers. *Pattern Recognition Letters*, 41:14 – 22.

Sun, L., Ji, S., and Ye, J. (2011). Canonical correlation analysis for multilabel classification: A least-squares formulation, extensions, and analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):194–200.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Tai, F. and Lin, H.-T. (2012). Multilabel classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2011). Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(7):1079–1089.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85.

Vapnik, V. N. (1999). *The Nature of Statistical Learning Theory*. Springer-Verlag New York.

Vens, C., Struyf, J., Schietgat, L., Džeroski, S., and Blockeel, H. (2008). Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214.

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., and Xu, W. (2016). CNN-RNN: A unified framework for multi-label image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2285–2294.

Weston, J., Bengio, S., and Usunier, N. (2011). Wsabie: Scaling up to large vocabulary image annotation. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 2764–2770.

Xu, C., Tao, D., and Xu, C. (2016). Robust extreme multi-label learning. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1275–1284.

Yang, Y. and Gopal, S. (2012). Multilabel classification with meta-level features in a learning-to-rank framework. *Machine Learning*, 88(1-2):47–68.

Yeh, C.-K., Wu, W.-C., Ko, W.-J., and Wang, Y.-C. F. (2017). Learning deep latent space for multi-label classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2838–2844.

Yu, H.-F., Jain, P., Kar, P., and Dhillon, I. (2014). Large-scale multi-label learning with missing labels. In *Proceedings of the International Conference on Machine Learning*, pages 593–601.

Zeiler, M. D., Ranzato, M., Monga, R., Mao, M. Z., Yang, K., Le, Q. V., Nguyen, P., Senior, A., Vanhoucke, V., Dean, J., and Hinton, G. E. (2013). On rectified linear units for speech processing. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521.

Zhang, M.-L. and Zhou, Z.-H. (2006). Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1338–1351.

Zhang, Y. and Schneider, J. (2011). Multi-label output codes using canonical correlation analysis. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 873–882.

Zimek, A., Buchwald, F., Frank, E., and Kramer, S. (2010). A study of hierarchical and flat classification of proteins. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7:563–571.

Author's Own Publications

Hirschmann, F., Nam, J., and Fürnkranz, J. (2016). What makes word-level neural machine translation hard: A case study on english-german translation. In *Proceedings of the International Conference on Computational Linguistics*, pages 3199–3208.

Loza Mencía, E., de Melo, G., and Nam, J. (2016). Medical concept embeddings via labeled background corpora. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 4629–4636.

Nam, J., Kim, J., Loza Mencía, E., Gurevych, I., and Fürnkranz, J. (2014). Large-scale multi-label text classification–revisiting neural networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 437–452.

Nam, J., Kim, Y.-B., Loza Mencía, E., Park, S., Sarikaya, R., and Fürnkranz, J. (2019). Learning context-dependeent label permutations for multi-label classification. In *Proceedings of the International Conference on Machine Learning*, pages 4733–4742.

Nam, J., Loza Mencía, E., and Fürnkranz, J. (2016). All-in text: Learning document, label, and word representations jointly. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1948–1954.

Nam, J., Loza Mencía, E., Kim, H. J., and Fürnkranz, J. (2015). Predicting unseen labels using label hierarchies in large-scale multi-label learning. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 102–118.

Nam, J., Loza Mencía, E., Kim, H. J., and Fürnkranz, J. (2017). Maximizing subset accuracy with recurrent neural networks in multi-label classification. In *Advances in neural information processing systems*, pages 5413–5423.

Sappadla, P. V., Nam, J., Loza Mencía, E., and Fürnkranz, J. (2016). Using semantic similarity for multi-label zero-shot classification of text documents. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 423–428.

Erklärung zur Dissertation[1]

Hiermit versichere ich, die vorliegende Dissertation selbständig nur mit den genannten Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 02. Mai 2018

Jinseok Nam

[1] Gemäß §9 Absatz 1 der Promotionsordnung der Technischen Universität Darmstadt.