Hierarchical text classification of large-scale topics

a Neural approach

Koustuv Sinha

260721248

School of Computer Science

McGill University, Montreal

November 16, 2018

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Master of Science.

Acknowledgements

I am deeply grateful to my supervisors Derek Ruths and Joelle Pineau for their precious insights and encouragement. I am also highly indebted to my collaborator Yue Dong for helping me shaping up the paper before EMNLP submission and lots of encouragement and feedback. Special thanks to Nicolas Angelard-Gontier, Edward Newell and Sumana Basu for their persistence in constructively challenging my ideas.

I would also like to thank my parents and sister for their constant support and encouragement from miles away, having their faith in me to pursue research.

Finally, I would like to take this space to thank my sponsors, Pierre Arbour Foundation and Fonds Nature et Technologies Quebec (FRQNT) for supporting my research throughout my Masters studies. For an international student, I am extremely grateful to my sponsors and my supervisors to have faith in my research and fund my studies in this critical juncture of my career.

Abstract

Topic modelling and classification of documents is a well studied problem in Natural Language understanding. Deep neural networks have displayed superior performance over the traditional supervised classifiers in text classification. They learn to extract useful features automatically when sufficient amount of data is presented. However, along with the growth in the number of documents comes the increase in the number of categories, which often results in poor performance of the multiclass classifiers. In this work, we use external knowledge of topic category taxonomy to aide the classification by introducing a deep hierarchical neural attention-based classifier. Our model performs better than or comparable to state-of-the-art hierarchical models with significantly fewer computational resources while maintaining high interpretability.

Résumé

La modélisation de sujet et la classification de documents est un problème bien étudié dans la compréhension du langage naturel. Les réseaux de neurones profonds affichent des performances supérieures à celles des classificateurs supervisés traditionnels dans la classification de textes. Ils apprennent à extraire automatiquement des attributs utils lorsqu'une quantité suffisante de données est présentée. Cependant, avec la croissance du nombre de documents vient l'augmentation du nombre de catégories, ce qui entraîne souvent une mauvaise performance des classificateurs à nombreuses catégories. Dans ce travail, nous utilisons l'information donné par la taxonomie de catégorie de sujet pour aider la classification en introduisant un classificateur hiérarchique basé sur l'apprentissage profond et les systèmes d'attention. Notre modèle est comparable et peut mieux performer que des modèles hiérarchiques de pointe tout en utilisant moins de ressources de calcul tout en conservant une haute interprétabilité.

Contents

List of Figures

1

Introduction

Text classification has been a core supervised and unsupervised learning problem in Natural Language Processing (NLP) that has many real-world applications from sentiment analysis to biomedical text mining (Allahyari et al. 2017). With the sheer number of documents that are being generated all over the world every day, automatic text classification has become an essential tool for searching, retrieving, and managing text. We also face an ever-increasing demand for categorizing documents into a large number of classes, which poses a unique challenge to classification systems. Modern neural-based approaches, while handling a large number of documents efficiently by performing automatic feature extraction, also suffer from the same issues. In this thesis, we look back into traditional methods of dealing with large classes and hierarchical classification. In light of modern neural-based approaches, we present an alternative model which is trained to iteratively read and classify a document with respect to varying class hierarchy.

1.1 Overview of text classification systems

Traditionally text classification is performed with the use of probabilistic classifiers such as Naive Bayes (McCallum, Nigam, et al. 1998b), Nearest neighbors (Han,

Karypis, and Kumar 2001), Decision Trees (Apté, Damerau, and Weiss 1994a), Support Vector Machines (Lodhi et al. 2002) etc. However, these classical systems often need extensive hand-held feature extraction methods which tend to not generalize well on large amounts of data.

With the advent of Neural networks, there has been an increasing trend in developing data-driven neural text classifiers (Collobert, Weston, et al. 2011; Lai et al. 2015; X. Zhang, Zhao, and LeCun 2015; Yogatama et al. 2017; Conneau, Schwenk, et al. 2017), due to their ability to handle large-scale corpora and their robustness in automatic feature extraction. The scalability and flexibility of neural networks allow these data-driven methods to handle large datasets in the scale of millions easily. In addition, the robust automatic feature extraction provided by deep neural networks have been proven to be highly generalizable when trained on large corpora (Conneau, Schwenk, et al. 2017).

Neural text classifiers use either standard Recurrent Neural Networks (RNN) or Convolution Neural Networks (CNN) to generate a document representation, which is then fed to a Multi-layer perceptron (MLP) (Bengio, Ducharme, et al. 2003). With the increase in individual document sizes, RNN's tend to lose information as they encounter more words, and the last hidden state of the RNN fails to combine the information of the entire document. Hence, attention mechanisms (Bahdanau, Cho, and Bengio 2014; Z. Lin et al. 2017; Vaswani et al. 2017) have been proposed to counter this problem which constructs a probability vector over the words in the document and combine the intermediate hidden representation of the words according to the probability. These approaches have been proven to be superior to non-neural classical text classification methods. We discuss these technical preliminaries in detail in Chapter 3.

1.2 The challenge of large classes

Text classification has become increasingly challenging as the number of categories grows, which is an inevitable consequence of the continually expanding corpora. In neural text classification, the standard approach is to use a softmax function that outputs a probability vector over the entire set of classes. This computation becomes very expensive with the increase in class size, so different workarounds to have been suggested such as Hierarchical Softmax (Goodman 2001) based on Huffman encoding Tree (Mikolov, K. Chen, et al. 2013). Even so, the problem remains mostly due to the significant semantic overlap among the classes which makes the decision boundary of the classifiers hard to determine.

To alleviate this problem, external knowledge is often used to supplement classification, such as language models (Howard and Ruder 2018b). One form of the external knowledge – *class taxonomy* – has been introduced to aid the classification in a hierarchical fashion (Koller and Sahami 1997). The intuition is that since the classes have semantic overlap, we bin the classes according to a semantic taxonomy which enables the classifier to break down the classification into multiple levels. Various methodologies have been proposed to perform classification hierarchically.

1.3 Hierarchical classification

In general, hierarchical classifiers can be categorized into two broad approaches: *local* and *global* (Silla and A. A. Freitas 2011). The local approaches create a unique classifier for each node in the taxonomy which has a subtree (S. Liu et al. 2001; Quinn and Laier 2006; Vens et al. 2008; Kowsari, D. E. Brown, Heidarysafa, Meimandi, et al. 2017). They can be further categorized into *top-down* and *bottom-up* approaches. Each classifier depends on the classification of its previous level, and hence is termed as *hierarchical*. They differ in the direction of classifier creation: whether to classify

starting from from top level nodes of the taxonomy or from leaf level nodes. In contrast, global approaches create a single classifier for the entire taxonomy (Silla Jr and A. A. Freitas 2009). They are also termed as *big-bang* classifiers. We discuss the hierarchical classification literature in detail in Chapter 2.

The local models, however, suffer an inherited disadvantage: the number of sub-models required grows equally with respect to the number of sub-trees. In addition, the top-down based models have no ability to correct mistakes of classification on the parent level: if the parent class is wrongly predicted, the child level classification doesn't even have the right candidates to choose from.

Hierarchical approaches are also used by neural models, such as in image classification (Salakhutdinov, Tenenbaum, and Torralba 2013), where the top level task is to classify between an animal and a vehicle, and the next level the task is to classify among different types of animals or vehicles. Recently this approach has also been used in neural text models. Kowsari, D. E. Brown, Heidarysafa, Meimandi, et al. 2017 proposed the HDLTex model which displays superior performance over traditional non-neural based models. HDLTex creates an RNN for each parent level node and thereafter proceeds to create more models per level and per subtree of the taxonomy.

However, this model suffers the same inherited disadvantage of the *top-down* approach where the number of sub-models grows exponentially with respect to the number of sub-trees when encountered with a larger corpora. This is especially problematic in HDLTex as it uses deep networks with a large number of parameters for the sub-models and the combined model itself grows exponentially with the depth of taxonomy. In contrast, we propose a unified *global* deep neural-based classifier that overcomes the problem of exploding models.

1.4 Summary of Contributions

We propose a *global* deep neural-based hierarchical classifier which uses an external taxonomy to predict a large number of classes. In this task, we predict all the levels of the taxonomy using a single classifier, at the same time. That is, we predict the parent class, its subclass(es), and the leaf for a given document. The backbone of our approach is an encoder-decoder structure that sequentially predicts the class label of the next level, conditioned on a dynamic document representation obtained from a variant of the traditional attention (Bahdanau, Cho, and Bengio 2015). We explain the details of our model in Chapter 4. This attention variant constructs the dynamic document representation based on the predicted class label rather than the hidden state from the previous layer. Our novel attention mechanism is conditioned on the predicted label from the previous layer classification to construct a dynamic document representation that takes into account each level of granularity. Concretely, our contributions in this work are as follows:

1. We propose an end-to-end *global* neural attention-based model for hierarchical classification, which performs better than or comparable to the state-of-the-art hierarchical classifier with significantly less computational resource.

2. We empirically show that the use of hierarchical taxonomy improves robustness in classifying large number of classes, by comparing with state-of-the-art flat classifiers (Chapter 7).

3. We present a new dataset and a methodology to create such hierarchical dataset with community curated taxonomy (Chapter 5).

2

Literature Review

In the previous chapter, we presented an overview of the problem and our proposed approach. In this chapter, we look at existing works in text classification and hierarchical classification. In general, text classification can be categorized into two tasks: *flat* classification where we classify documents into a number of classes without any parent-child relationship; and Hierarchical text classification, where hierarchical information of parent-child relationship is typically incorporated in the classification process (*local* and *global* approaches).

2.1 Text classification literature review

2.1.1 Introduction

Traditional text classification methods (*flat*) focus on selecting a good set of features to represent the documents, usually converted into numeric vectors, termed as Vector Space Models (VSM) (Turney and Pantel 2010). Here, each word is represented by a numerical value indicating the importance or weight of the word in the document. These numeric representations are usually learned from different weight models: 1) *Boolean model* where a constant weight is assigned to any word appearing in the doc-

ument, 2) *Bag-of-words* model where the words appearing in the document is assigned a numeric value based on its frequency of occurrence, and 3) *Term Frequency-inverse document frequency* (TF-IDF) (Salton and Buckley 1988) where relative weights with respect to term frequencies are normalized by their frequency across the document collection. The latter model makes low frequency words more distinctive and hence provides a powerful way to represent a document.

2.1.2 Classical algorithms

These traditional approaches employ linear to non-linear classifiers such as Naive Bayes (McCallum, Nigam, et al. 1998a; S.-B. Kim et al. 2006), Support Vector Machines (SVM) (Dumais et al. 1998; Joachims 1999; Tong and Koller 2001) or Decision trees (Apté, Damerau, and Weiss 1994b) for text classification. The *Naive Bayes* classifier models the distribution of documents in each class into a probabilistic model. Concretely, this model finds the posterior probability of a class conditioned on the distribution of the words in the document. Support Vector Machines (SVM) are usually used when the document vectors of the classes in consideration are not linearly separable, with the use of kernel functions which transforms the input representation into a higher dimension where the hyperplane becomes linearly separable. Decision Trees, on the other hand, recursively partition the training dataset into smaller subdivisions based on a decision criterion, thereby converting one linear separable plane into multiple nested linearly separable planes which provides a non-linear decision boundary over the text documents.

2.1.3 Neural algorithms

The core of this thesis relies on the recent advancements in Neural Text classification models. Deep Neural networks merge feature extraction and classification into one single process, where the parameters can be jointly learned through *back-propagation*

(Xue et al. 2008a; Lai et al. 2015; X. Zhang, Zhao, and LeCun 2015). Instead of building a numeric vector representation from document statistics, Mikolov, Sutskever, et al. 2013's seminal work on *word2vec* paved the road ahead of neural classification models by building a fixed-length distributed vector representation for words. This fixed-length vector representation can then be fed into a Multi-layer perceptron (MLP) to generate a probability vector over the classification classes (Collobert, Weston, et al. 2011; Iyyer et al. 2014). These recent approaches of distributed sentence representation again fall into two categories: *sequence-based models* and *tree-structured models*. Sequence-based models construct sentence representations from word sequences by taking into account the relationship between the successive words (R. Johnson and T. Zhang 2014). On the other hand, tree-structured models use a syntactic parse tree and learn sentence representations from the leaves to the root in a recursive fashion (Socher et al. 2013).

2.1.3.1 Convolution Neural Network Approaches

For both of these types of models, Convolution Neural Networks (CNNs) (LeCun et al. 1998) and Recurrent Neural Networks (RNNs) are the main architectures used widely by researchers in NLP. CNNs have recently achieved top performance in text classification by extracting n-gram features at different positions of a sentence with the use of convolution filters, and can learn short and long-range relations through pooling operations (Denil et al. 2014; Kalchbrenner, Grefenstette, and Blunsom 2014; X. Zhang and LeCun 2015; Mou et al. 2015). CNN with pooling show promising results on a wide array of tasks, such as document classification (R. Johnson and T. Zhang 2014), text categorization (M. Wang and Manning 2013), sentiment classification (Kalchbrenner, Grefenstette, and Blunsom 2014; Y. Kim 2014), event detecion (Y. Chen et al. 2015), paraphrase identification (Yin and Schütze 2015) , and question answering (Dong et al. 2015) .

2.1.3.2 Recurrent Neural Network Approaches

The other popular architecture used in neural text classification is Recurrent Neural Networks (RNN's) (Elman 1990). They can handle sequences of any length and capture long-term dependencies. To avoid the problem of gradient explosion / vanishing in standard RNN, Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber 1997) and Gated Recurrent Units (GRU) (Cho et al. 2014) are common variants used in text classification to gain a much superior performance than non-neural models (Tang, Qin, and T. Liu 2015; Tai, Socher, and Manning 2015; C. Zhou et al. 2015). In addition, recursive architectures (Goller and Kuchler 1996) allow for working with tree structures while preserving structural information. RNNs have also shown very strong results for language modelling (Adel, Vu, and Schultz 2013), sequence tagging (Irsoy and Cardie 2014), machine translation (Sutskever, Martens, and G. E. Hinton 2011), parsing (Dyer et al. 2015), dialog state tracking (Mrkšić et al. 2015) and response generation (Sordoni et al. 2015).

2.1.3.3 Pooling & Attention

A common theme in these convolutional neural networks (CNN)-based or recurrent neural network (RNN)-based approaches are to create a document representation from either the last hidden state of the RNN or some sort of pooling mechanism. The most common methods are to form a fixed-size vector representation of the document either by selecting the maximum value over each dimension of the hidden units of the word representations, termed as *max pooling* (Collobert and Weston 2008); or by considering the average of the word representations (Conneau, Kiela, et al. 2017). A pictorial representation of a Bidirectional LSTM with max-pooling is given in Figure 2.1.

However, the last hidden state of an RNN might forget most of the discriminative information about a document and mean/max pooling might not be the best ap-

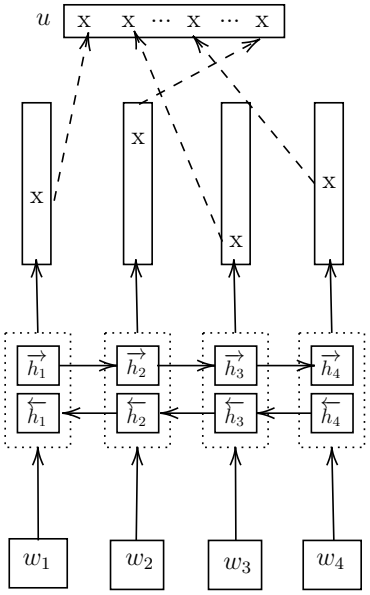


Figure 2.1: BiLSTM maxpooling network, figure adapted from Conneau, Kiela, et al. 2017

proach to combine the features in the document (Sutskever, Vinyals, and Le 2014a). Furthermore, attention mechanisms (Bahdanau, Cho, and Bengio 2015; Sutskever, Vinyals, and Le 2014a) have become a common practice to be used on top of the CNN/RNN structures to enrich the power of sentence/document representation in sequential generation tasks. These approaches have been adapted for text classification too to generate richer document representation than mean/ max pooling (Yang et al. 2016; Conneau, Kiela, et al. 2017; Z. Lin et al. 2017). Since attention computes the individual *weight* of the words to form a document representation, they provide high interpretability and allow us to inspect which parts of the text are discriminative for a particular classifier. More detailed review of existing attention architectures is given in Chapter 3.

2.2 Hierarchical classification : A review

2.2.1 Introduction

Hierarchical text classification has been studied extensively in the natural language processing literature (Silla and A. A. Freitas 2011). Hierarchical classification is a supervised classification problem where the output of the classification is defined over a class *taxonomy*. A class taxonomy can be defined as a tree structured hierarchy of knowledge concepts. It is defined over a partially order set $(C, <)$, where C is a finite set that enumerates all class concepts in the application domain, and the relation $<$ represents a *"is-a"* relationship (Wu, J. Zhang, and Honavar 2005). This taxonomy can be organized into a tree or a Directed Acyclic Graph (DAG) (A. Freitas and Carvalho 2007) (Figure 2.2), in this thesis we consider the case of the former. This approach has a large number of applications from text categorization (Chakrabarti et al. 1998; Koller and Sahami 1997), protein function prediction (Clare and King 2003; Kriegel et al. 2004; Otero, A. A. Freitas, and C. G. Johnson 2009), music genre classification (Burred and Lerch 2003; DeCoro, Barutcuoglu, and Fiebrink 2007) to image classification (Dimitrovski et al. 2011; Binder, Kawanabe, and Brefeld 2009; Salakhutdinov, Tenenbaum, and Torralba 2013).

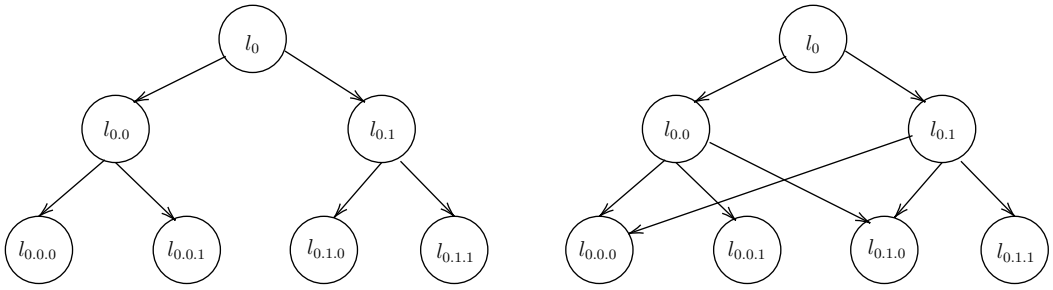


Figure 2.2: Tree Structure (*left*) and DAG structure (*right*), figure adapted from Silla and A. A. Freitas 2011

2.2.2 Categorization

Hierarchical classification can also be categorized into how deep the classification in the hierarchy is performed. For hierarchical classification, we typically use a fixed taxonomy of concepts, and each document in the training class is annotated by the tree to which it belongs in this fixed taxonomy. If the output of the classifier is at the leaf node of this tree, then it is termed as mandatory leaf node (A. Freitas and Carvalho 2007) or virtual category node prediction (Sun and Lim 2001). If the classifier can output the class at any level of the taxonomy tree, then it is referred to as non-mandatory leaf node (A. Freitas and Carvalho 2007) or category tree prediction (Sun and Lim 2001). Finally, hierarchical classification is categorized by the type of algorithm chosen. Broadly, the algorithms can be classified into *local* and *global* approaches. Local approaches typically construct a local classifier at each parent node of the tree, (Koller and Sahami 1997), and it is also known as *top-down* approach.

2.2.2.1 Top-down approach

In *top-down approach*, a classifier is trained with the documents associated with each node level from the fixed hierarchy. For a given new document, it is first classified into top-level categories under the root, and then progressively classified in the next levels, until a stopping condition is met (Koller and Sahami 1997; Bennett and Nguyen 2009; Cai and Hofmann 2004; T.-Y. Liu et al. 2005; Sun and Lim 2001). However, this approach has two major disadvantages. Firstly, an error at a certain class level is propagated downwards in the hierarchy. Several alternatives have been suggested to counter this issue (Bennett and Nguyen 2009; T.-Y. Liu et al. 2005) by performing bottom-up training and utilizing cross-validation and meta-features. Secondly, the top-down approach results in an explosion of models when trained on larger taxonomies, creating huge computation overhead. To counter some of these issues, *narrow-down approach* is used by first cutting down the search space of the

entire hierarchy and building a classifier for a small number of resulting categories as in Xue et al. 2008b. Given an input document, the method uses a search engine to search for candidate categories in relevant category levels, which results in significant performance improvement specifically in deeper levels. Narrow-down approaches are further enhanced by the use of global, local (Oh, Choi, and Myaeng 2010) and path context information derived from the target hierarchy (Oh, Choi, and Myaeng 2011; Oh and Jung 2014; Oh and Myaeng 2014; Oh and Jung n.d.). Here, they use label information to learn a language model which augments the narrow down search by observing the label terms that are not occurring as frequently as expected.

2.2.2.2 Global approach

Finally, *global* approaches or *big-bang* approaches learn a single global model for all classes in the tree of the hierarchy. This significantly reduces the computation overhead but suffers from scaling up to large number of concepts (Koller and Sahami 1997; Costa et al. 2007). Various classification techniques have been used using traditional Support Vector Machines (Cai and Hofmann 2004; Labrou and Finin 1999; Sasaki and Kita 1998), Naive Bayes (Silla Jr and A. A. Freitas 2009), Predictive clustering trees (Blockeel et al. 2006), C4.5 (Clare and King 2003) and rule based classifiers (K. Wang, S. Zhou, and He 2001).

2.2.3 Neural approaches

While many classical linear or kernel-based expressive models have been explored in the context of hierarchical text classification, in this thesis we focus on using neural models to learn to classify documents using hierarchical information. Recently, Kowsari, D. E. Brown, Heidarysafa, Jafari Meimandi, et al. 2017 have formulated the hierarchical classification problem as a *top-down* neural classification approach, where they use *Web of Science*[1] data and the taxonomy to classify hierarchical con-

[1]We provide a brief review on available hierarchical text classification datasets in Chapter 5

cepts. The model is a local top-down classifier where a new Deep Neural Network based classifier (DNN) is used in each parent node, and the models at each level are separately trained. They test a combination of CNN's and RNN's as the individual model and report RNN having the best classification accuracy on each individual level of the taxonomy. However their model also suffers the inherent problems of local classifiers: the increasing number of models present an explosion in parameter space, rendering such a model on corpora with large taxonomies computationally infeasible.

In this thesis, we will thus formulate a *global* neural model which imbibes the class taxonomies in classification and achieves superior performance than prior research.

3

Technical Preliminaries

In the previous chapter we went over related work in text classification and hierarchical classification. In this chapter, we go over some technical preliminaries required to understand our model.

3.1 Feedforward Neural Networks

3.1.1 Introduction

Feed-forward neural networks, also called multi-layer perceptrons (MLP) are the fundamental building blocks of neural models used in deep learning, and they are used for a wide range of supervised and unsupervised learning problems. The goal of this network is to approximate a function $f^*(x) = y$ with a learned non-linear function $f(x;\theta)$, where θ are the parameters of the MLP, $x \in \mathbb{R}^k$ are the inputs and $y \in \mathbb{R}^l$ are the labels of the dataset. These models are called feed-forward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to output y. [1]

[1] http://www.deeplearningbook.org/contents/mlp.html

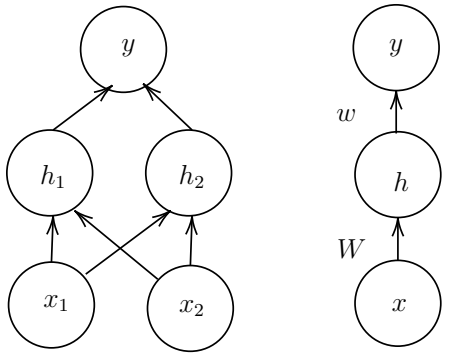


Figure 3.1: A fully connected Multi-layer perceptron, with internal composition of a node. Figure adapted from Ian Goodfellow and Courville 2016

3.1.2 Technical details

An MLP has multiple layers such that it is formed from the composition of multiple functions, i.e $f(x) = f^{(n)}(f^{(n-1)}(...f^{(1)}(x))...))$ for a network with n layers, where the final layer $f^{(n)}$ is often called the output layer, while the layers $f^{(1)}, ..., f^{(n-1)}$ are called hidden layers as there is no target output for these layers. MLP can be represented as a directed acyclic graph, which is shown in Figure 3.4.

In practice, a standard formulation for MLP is used for each layer $f^{(i)}$ in the network:

$$f^{(i)} = g(W_i x + b_i) \tag{3.1}$$

where W_i is a matrix of real-valued parameters, b_i is a vector-valued bias, and W_i and b_i are learned separately for each layer. $g(.)$ is a non-linear activation function, which is usually chosen as a fixed function for the entire network. Some popular choices of $g(.)$ include the hyperbolic tangent function $tanh = \frac{e^{2x}-1}{e^{2x}+1}$, sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, and the rectified linear unit (ReLU) $ReLU(x) = max(0, x)$ (Glorot, Bordes, and Bengio 2011)

The output layer of the neural network depends on the task. For regression tasks, where the goal is to predict a real-valued outcome, the final layer is often linear,

omitting $g(.)$:

$$f^{(n}(x) = W.x + b \tag{3.2}$$

For classification tasks where each output neuron represents a class, a softmax distribution is often used for the non-linearity:

$$softmax(x) = \frac{e^{x_j}}{\sum_j e^{x_j}} \tag{3.3}$$

where the sum in the denominator is taken over all of the neurons in the output layer (i.e over all classes). Thus the output at each neuron represents the probability that the input x belongs to a certain class. In our application of topic classification, we will see how this exact same softmax is calculated over a set of classes for each levels of the taxonomy.

3.1.3 Training

Feedforward neural network weights are usually trained with *backpropagation*. Often termed as backprop, it is an algorithm that describes how to update θ of an MLP to minimize the loss function L with gradient descent. Essentially, it is the application of chain rule: for each computation layer $f^{(i)}(x) = g(W_i x + b_i)$ that composes the neural network, the gradients are calculated by taking first the gradient of the output layer $f^{(t)}$ in the network with respect to the loss, and then taking the gradient of the output layer with respect to the parameters $\theta_i = \{W_i, b_i\}$:

$$\nabla_{\theta_i} L = \frac{\delta L}{\delta f^{(i)}} \nabla_{\theta_i} f^{(i)} \tag{3.4}$$

The parameters θ_i are incrementally updated towards the direction of negative gradient. Also, a step size α, termed as learning rate, determines how strong the

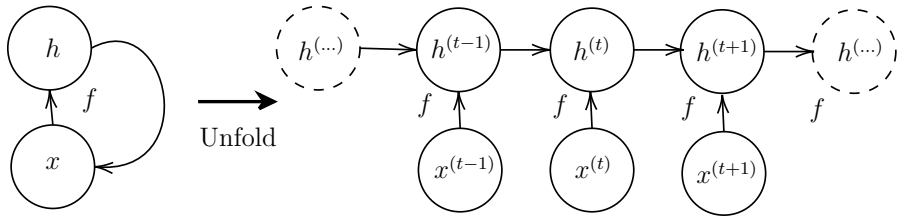


Figure 3.2: An unfolded view of a recurrent neural network. Figure adapted from Ian Goodfellow and Courville 2016

update is towards the direction.

$$\theta_i \leftarrow \theta_i - \alpha \nabla_{\theta_i} L \tag{3.5}$$

3.2 Recurrent Models

3.2.1 Introduction

Recurrent Neural Networks (RNN) are feedforward neural networks with feedback loops. Most often, these are self loops from hidden neurons to themselves and to other neurons in the same layer. To model the data that does not correspond to the independence assumption we use the recurrent neural networks, which models the data that is correlated in time. This can take the form of sequences of input data $x_1, ..., x_m$, which is the exact case for natural language, where each token x_i represents a word in some language. An RNN can be represented as a directed cyclic graph, shown in Figure 3.2.

3.2.2 Technical details

The addition of the feedback loops to the hidden units of an MLP can be thought of forming a hidden state of the network $h(x)$, corresponding to the vector of activations (value after the non-linearity) of the hidden neurons, that evolves over time as we

present inputs to the network. The hidden state is updated at each time step according to some function f:

$$h_t = f(h_{t-1}, x_t) \tag{3.6}$$

Here, the crucial point is that the parameters of $f(.)$ are the same for all time steps. This form of parameter sharing forms the critical foundations of RNNs, because if we had different parameters for each step, not only would the model be much more prone to overfitting, but it would not be able to generalize to sequence lengths unseen during training.

More specifically, an RNN is governed by the following update equations at time t:

$$h_t = g(Wh_{t-1} + Ux_t + b), o_t = softmax(Vh_t + c), \tag{3.7}$$

where W is the hidden-to-hidden matrix of parameters, U is the input matrix, V is the output matrix, b is the hidden state bias and c is the output bias. The most common non-linearity $g(.)$ used in RNN is the $tanh$ function. The initial hidden state h_0 can be either set in advance (e.g. a vector of all 0's), or can be learned during optimization.

3.2.3 Training

The loss is calculated for a given sequence x paired with corresponding y values. It can thus be calculated as the sum of loss over all time steps, usually calculated as a negative log likelihood:

$$L = -\sum_t \log p_{model}(y^{(t)}|x^{(1)}, ...x^{(t)}) \tag{3.8}$$

To update the weights, the gradient calculation is usually performed as a forward propagation pass over the unrolled graph and a subsequent backward propagation pass. This backpropagation applied over the unrolled graph is coined as *backpropagation through time (BPTT)*. Here, the gradient calculation is started at the last time step output, and recursively propagated backwards. For the final timestep T, the gradient is calculated as:

$$\nabla_{h^{(T)}} L = V^\top \nabla_{o^{(T)}} L \tag{3.9}$$

Thus when we have an intermediate hidden state h_t which has o_t and the next hidden state h_{t+1} as descendents in the graph, the gradient is calculated recursively as:

$$\nabla_{h^{(t)}} L = (\frac{\delta h^{(t+1)}}{\delta h^{(t)}})^\top (\nabla_{h^{(t+1)}} L) + (\frac{\delta o^{(t+1)}}{\delta h^{(t)}})^\top (\nabla_{o^{(t)}} L) \tag{3.10}$$

3.2.4 Bidirectional RNN

Recurrent networks however may want to depend on the entire input sequence, for example in speech recognition the current sound as a phoneme may depend on the subsequent phonemes. In text, the linguistic dependencies among words may not always depend on nearby words but depend on words in the future. For these use cases, typically in Natural language processing we use Bidirectional recurrent neural networks (Schuster and Paliwal 1997). These networks combine an RNN which moves *forward* in time from start to end of a sequence with another RNN which moves *backwards* in time from end to start of a sequence. The resulting hidden state $h(t)$ contains both the forward view representation and backward view representation. This enables the model to look at any timestep t the information from both the directions and hence dependencies allow for long-range lateral interactions (Visin et al. 2015).

3.2.5 Challenge of Long Term dependencies

One of the key problems alluding recurrent neural networks is the vanishing or exploding gradients (Bengio, Frasconi, and Simard 1993; Hochreiter and Schmidhuber 1997; Bengio, Simard, and Frasconi 1994; Pascanu, Mikolov, and Bengio 2013) for long sequences, making long-term dependencies difficult to train. Here, the error gradients accumulate during an update result in very large gradients, or if the gradients are small then the accumulated gradient becomes very small by power multiplication. At these extremes, numerical instability such as overflow and underflow occurs resulting in NaN values. In order to account for long term dependencies in a recurrent neural network, the gradients almost always reach the parameter space of vanishing gradients (Bengio, Frasconi, and Simard 1993), making the training very long and unstable. There are various workarounds suggested to tackle this issue, such as norm clipping, gradient regularization (Pascanu, Mikolov, and Bengio 2013). Another way to deal with long-term dependencies is to design a model that operates at multiple timescales. Approaches such as adding skip connections (T. Lin et al. 1998) through time and leaky units for different timescales (Mozer 1992) have been proposed to alleviate this issue. This idea of organizing the RNN into multiple time scales also arises in the use case of removing connections to force the units to operate on a longer time scale (El Hihi and Bengio 1996; Pascanu, Mikolov, and Bengio 2013). The most practical solution to long-term dependencies is the use of *gated* RNN's, one of which we use in our model: *long short-term memory*.

3.2.6 Long Short Term Memory (LSTM)

Hochreiter and Schmidhuber 1997 introduced the seminal work on *long short-term memory* (LSTM) which contains self-loops to produce paths where the gradient can flow for long durations. This self-loop is controlled or gated, thus introducing dynamicity to the time scale integration. LSTM has been found to be extremely successful

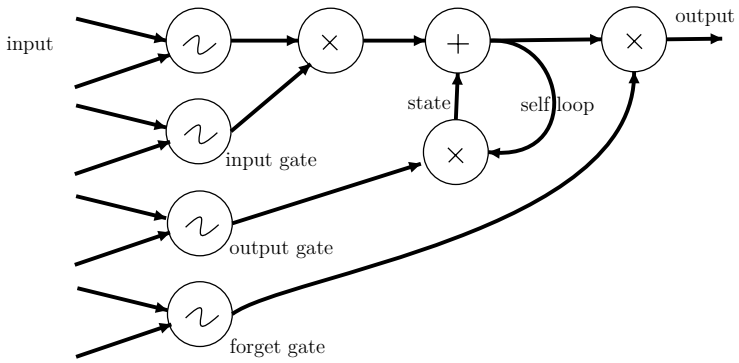


Figure 3.3: Block diagram of LSTM cell. Figure adapted from Ian Goodfellow and Courville 2016

on a variety of tasks such as speech recognition, handwriting recognition, machine translation, image captioning and parsing.

An LSTM cell (Figure 3.3) consists of a cell state which is controlled by several gates to incorporate and forget the information via self-loops. Concretely, an LSTM block consists of an inner self-loop, which is controlled by a *forget gate* $f_i^{(t)}$, which sets the gate on or off via a sigmoid unit.

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}) \tag{3.11}$$

where, $x^{(t)}$ is the current input vector and $h^{(t)}$ is the hidden layer vector. (Ian Goodfellow and Courville 2016). The LSTM cell internal state is updated with a conditional self loop weight $f_i^{(t)}$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}) \tag{3.12}$$

The external input gate unit $g_i^{(t)}$ is computed similarly to the forget gate but having its own parameters.

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}) \tag{3.13}$$

The output of the LSTM $h_i^{(t)}$ can be calculated with the use of another output gate $q_i^{(t)}$ as follows:

$$h_i^{(t)} = tanh(s_i^{(t)}) q_i^{(t)} \tag{3.14}$$

$$q_i^{(t)} = \sigma(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}) \tag{3.15}$$

LSTM architectures are therefore used to tackle long-term dependency problem as they have proven to learn better memory capacity than traditional RNN's (Graves n.d.; Graves, Mohamed, and G. Hinton 2013; Sutskever, Vinyals, and Le 2014b).

3.3 Optimization & Regularization

3.3.1 Optimization

As we discussed in the previous sections (Section 3.1.3, Section 3.2.3), the objective of any deep learning model is to get the optimal weights θ by minimizing the loss function L with *gradient descent*. For example, the negative conditional log likelihood loss can be rewritten as:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta) \tag{3.16}$$

where, L is the per example negative log likelihood loss : $L(x, y, \theta) = -logp(y|x; \theta)$, which explains the probability of observing the class y given training data x, which we want to maximize, therefore we minimize the log of the value to be easily differentiable to get the cost gradient:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta) \tag{3.17}$$

The parameters θ are incrementally updated towards this negative gradient. This method is known as *gradient descent* and we usually use a derivative *stochastic gradient descent* (SGD) (Bottou 1998) to update the parameters θ over a minibatch of m examples. We use a learning rate α which we gradually diminish over time to reduce oscillations and noise in the learned parameters.

Algorithm 1 The Adam Algorithm

Require: Step size α, default 0.001
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2, default 0.9 and 0.999
Require: Small constant δ used for numerical stabilization, default 10^{-8}
Require: Initial parameters θ
Initialize 1st and 2nd moment variables $s = 0, r = 0$
Initialize time step $t = 0$
1: **while** stopping criterion not met **do**
Sample a minibatch of m examples from the training set $x^{(1)}, \ldots, x^{(m)}$ with corresponding targets $y^{(i)}$
Compute gradient: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
$t \leftarrow t + 1$
Update biased first moment estimate: $s \leftarrow \rho_1 s + (1 - \rho_1) g$
Update biased second moment estimate $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
Correct bias in first moment: $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$
Correct bias in second moment: $\hat{r} \leftarrow \frac{r}{1 - \rho_2^r}$
Compute update: $\Delta\theta = -\alpha \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$
Apply update: $\theta \leftarrow \theta + \Delta\theta$
2: **end**

This is the simplest optimization algorithm which is very powerful but often regarded as slow to arrive to the optimal parameter values, as it heavily depends on the initial choice of the learning rate and the batch size. We therefore use an extension of stochastic gradient update training aided by *momentum* (Polyak 1964) which accumulates the decaying moving averages of the past gradients and continues to move in the direction of global minima, therefore enabling faster convergence. Also, since the learning rate is one of the most difficult hyperparameters to tune in a neural network model, a common trend in optimization is to use *adaptive learning rates* which makes use of separate learning rate for each parameters and automatically adapt the rates

along the course of training.

In this thesis, we heavily use such an adaptive learning rate optimization algorithm know as Adam (Kingma and Ba 2014), which is presented in algorithm 1. Adam maintains per-parameter learning rate that improves performance on problems with sparse gradients such as natural language task. Also, Adam maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradient's, to observe how quickly the weight is changing. This helps Adam to adapt well on non-stationary and online problems.

3.3.2 Regularization

To restrict a neural network to *overfit* on the training data we usually employ specific regularization strategies. Overfitting is a common problem in Machine Learning algorithms where the model performs very well on the training data but fails to generalize on the testing data. The common approach in regularization is to limit the capacity of the neural models so that they cannot rote-learn the training data. This is done by adding a parameter norm penalty $\omega(\theta)$ to the objective function J (Section 3.16)

$$\hat{J}(\theta; X, y) = J(\theta; X, y) + \alpha\omega(\theta) \tag{3.18}$$

where, $\alpha \in [0, \inf)$ is a weight hyperparameter which controls the contribution of the norm penalty. For neural networks we typically only penalize the weight of the affine transformation at each layer keeping the biases unregularized. Now, the choice of norm can either be an L^2 norm or an L^1 norm. L^2 norm is most common norm penalty used which is also termed as *weight decay*. Here, the regularization term is $\omega(\theta) = \frac{1}{2}\|w\|_2^2$, which is also called *ridge regression* in the literature. Concretely, the total objective function can be rewritten as:

$$\hat{J}(w; X, y) = \frac{\alpha}{2} w^\top w + J(w; X, y) \tag{3.19}$$

which has the gradient update in the following equation:

$$\nabla_w \hat{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y) \tag{3.20}$$

which can be updated by regular backpropagation. We use L^2 regularization heavily in our work.

Another popular computationally inexpensive but powerful option is *Dropout* (Srivastava et al. 2014). Dropout specifically removes certain units from a base neural network and thereby creates an ensemble of bagged neural networks. This is a powerful regularization method as it creates multiple *weak* networks by iteratively removing units or connections from a neural network, and then performs an ensemble classification with those units, where the individual weak networks focus on different aspects of the task. Unlike bagged ensemble classifiers, Dropout allows weight sharing among the weak models, which makes it possible to represent an exponential number of models with tractable amount of memory. Concretely, Dropout uses a mask vector μ which applies a binary mask to all input and hidden units in a neural network, and $J(\theta, \mu)$ is the cost of the model defined by the parameters θ and the mask. Dropout training then consists of minimizing $E_\mu J(\theta, \mu)$.

3.4 Neural representations for text

3.4.1 Overview

For neural models to understand text we need to look beyond discrete word representations such as TF-IDF and bag of words used for classical text classification systems (Chapter 2). Instead, for neural models we use *distributional hypothesis* of language (Firth 1957) which states the meaning of a particular word depends on the context in which it is used. These can be broadly classified into *clustering-based* methods (P. F. Brown et al. 1992) and *embedding-based* (Collobert and Weston 2008; Mikolov,

Sutskever, et al. 2013) methods. Clustering-based methods assignes similar words to the same cluster and they represent the words by its cluster ID. Embedding-based methods, which are more popular with current neural systems, represent each word as vector such that similar words have similar vectors in cosine space.

3.4.2 Embedding based methods

For embedding-based methods, first all unique words from the document are extracted and a dictionary is formed, which we term as *word2id* dictionary, which assignes an unique identifier digit to each word. If we invert this dictionary then we get the identifier mapping in *id2word* dictionary. Then the document is transformed into a series of identifier digits. Then, we use a lookup table to assign an unique word vector $\vec{w}_i$ to all the words in the document, using the identifier. This is usually done in an *embedding* layer. The word vectors can be randomly initialized, such as the Word2vec implementation of Mikolov, Sutskever, et al. 2013, where we uniformly sample random numbers in the range $[-\frac{1}{2d}, \frac{1}{2d}]$. There are other initialization options as well, such as normal initialization, Xavier initialization (Glorot and Bengio 2010), etc. These word vector representations are learned via backpropagation with the end task, such as classification for our case.

3.4.2.1 Pretrained word embeddings

We can also use pre-trained word embeddings to initialize the word vectors. The *word2vec* family of algorithms use a language modelling setup to predict the next word given all its neighbouring words in a document, thereby learning the distributed representation of the words in an unsupervised fashion. Since these are learned over a large corpora, the word representations which are learned are termed as global representations and thus can be re-used in any classification systems. Another advantage is that it provides vector representations for words that do not appear in the training

set. Neural models such as RNN's therefore feed in this word vector representation and learn dependencies among the words which are used down the pipeline for classification or language modelling.

However, one needs to be careful while using pre-trained word embeddings as they incorporate inherent gender/race/religious/occupational bias (Bolukbasi et al. 2016; Henderson et al. 2017). One way to reduce bias (although not totally eliminate it) is to just use the pretrained embeddings as an initialization and letting the model further tune the embeddings with the downstream task.

3.5 Attention

3.5.1 Introduction

From our previous section we learned the tools needed to form a document representation. One can therefore construct the document vector representation as being the hidden state of the last step of recurrence. However, we also discussed the issues presented with long term dependencies, and thus the last hidden state may not contain the necessary information about a document, especially if the document is long. A common approach thus in many recurrent methods to represent a document vector is to create a max or average pooling (Collobert and Weston 2008) from the RNN hidden states. However these methods also suffer from the fact that these aggregation methods are inherently biased and they fail to represent the entire document representation in a fixed size vector.

In essence, the importance of a document lies within a certain set of words for classification, and for that we introduce the concept of *attention*, which is a weighted sum over the hidden states of the recurrent model over all the words in the document. The weights of this sum are referred to as attention scores and allow the network to focus on different parts of the input sequence as it generates the output sequences.

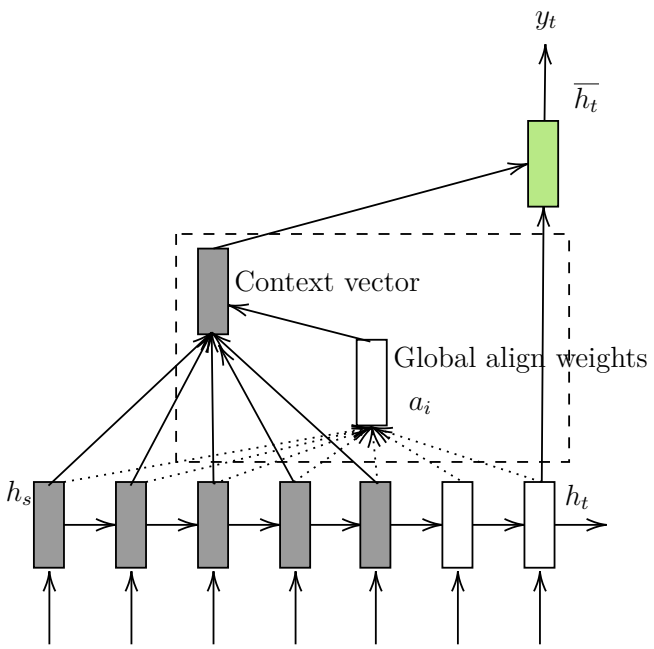


Figure 3.4: Global attention model. Figure adapted from Luong, Pham, and Manning 2015

Introduced by Bahdanau, Cho, and Bengio 2014 for neural machine translation, where one recurrent network (typically known as *encoder*) represents a document into a set of document vectors, and another recurrent network (typically known as *decoder*) predicts the next word given previous predicted words and the encoder representation. During each decoding step, the decoder uses attention weights to calculate a weighted sum over the encoder states. Recently, attention has gained popularity for document classification tasks (Z. Lin et al. 2017), and attention framework is perceived as a more general one-in-all framework for neural machine translation tasks (Vaswani et al. 2017). In our work too, we heavily rely on attention mechanisms to generate the document representation.

3.5.2 Terminologies

Typically to calculate Attention we need three different vector representations, such as a *query* vector $\vec{q}$, *key* vector k and *value* vector $\vec{v}$. $\vec{q}$ generally consists of the decoder states, $\vec{k}$ as being all encoder states, and $\vec{v}$ also being encoder states. For

each query-key vector, we calculate the weight using a *scoring function* a. After normalization using a softmax function we get the attention weights α_i over a set of words. Finally, these attention weights are multiplied by V to get the weighted sum of the word vectors. Various attention-based models are also typically classified into two broad categories, *global* and *local*, (Luong, Pham, and Manning 2015) which differ in terms of whether the attention is placed on all source positions or on only few source positions based on a context window. Concretely, the document or sentence representation D is calculated as a weighted sum of the values V:

$$D(Q, K, V) = \frac{e^{a(\vec{q},\vec{k})}}{\sum_w e^{a(\vec{q},\vec{k})}} V \tag{3.21}$$

3.5.3 Attention Score Functions

Subsequently, there has been a wealth of research into developing efficient attention score functions. Bahdanau, Cho, and Bengio 2015 introduced a multi-layer perceptron to calculate the attention weights given a query $\vec{q}$ and key $\vec{k}$:

$$a(\vec{q}, \vec{k}) = w_2^\top tanh(W_1[\vec{q}; \vec{k}]) \tag{3.22}$$

where, $[\vec{q}; \vec{k}]$ depicts the concatenation of query and key vectors. Bahdanau attention is usually often very good with large amounts of data.

Luong, Pham, and Manning 2015 introduced two new variants of global attention models, Bilinear and Dot product attention. With Bilinear attention, we can look into the full set of encoded context vectors instead of the last step context vectors in the case of Bahdanau, Cho, and Bengio 2014's model.

$$a(\vec{q}, \vec{k}) = \vec{q}^\top W \vec{k} \tag{3.23}$$

Dot product attention is however simpler as it doesn't have any parameters to learn, although requires the sizes of key and query to be same.

$$a(\vec{q}, \vec{k}) = \vec{q}^{\top}\vec{k} \tag{3.24}$$

The problem of dot product attention is that the product increases as the dimensions gets larger, while pushing the softmax function into regions with extremely small gradients. More recently, Vaswani et al. 2017 proposed the use of scaling factor with the dot product attention to mitigate this issue, where the scaling is done by the size of the key vector.

$$a(\vec{q}, \vec{k}) = \frac{\vec{q}^{\top}\vec{k}}{\sqrt{|\vec{k}|}} \tag{3.25}$$

3.5.4 Review of Attention architectures

Various improvements to attention mechanism has been proposed for different tasks. Gu et al. 2016 introduce a copying mechanism where a separate attention layer generates a softmax over whether to copy a particular word or not, and combined with Bahdanau, Cho, and Bengio 2014 attention this model works well for out of vocabulary words. Arthur, Neubig, and Nakamura 2016 uses lexicon-based probabilities to enhance Neural Machine Translation, where another attention layer is used to focus on the lexicons to generate the translation. Attention has also been used to focus on previously generated words (Merity et al. 2016) or previous hidden input or output states (Vaswani et al. 2017). Yang et al. 2016 propose a hierarchical attention structure, where first a word level attention is used to calculate sentence representation, and then a sentence level attention to generate document representation for classification. Sentence representations are also formed using an internal attention over its own hidden states, where each element in the sentence attends to other elements to form a context sensitive embedding representation (Cheng, Dong, and Lapata 2016;

Parikh et al. 2016; Y. Liu et al. 2016). Z. Lin et al. 2017 present similar intra attention with multiple reads over the sentence, essentially converting the attention vector to an attention matrix, which is also proven powerful for Neural Machine Translation by Vaswani et al. 2017, where it is termed as Multi-Head attention. The idea here is instead of using a single attention over the sentence, multiple attention vectors are trained to look at different aspects of the document or sentence using a penalty to ensure that each attention spans are mutually exclusive.

In this thesis we also heavily exploit attention to classify documents hierarchically. Concretely, we define a parent context-sensitive attention which formulates different attention spans over the document based on the depth of the taxonomy, which we detail in Chapter 4.

4

Model overview

In the previous chapter, we looked into the technical details required to understand our model. In this chapter, we dive deep into the proposed deep neural hierarchical classifier model **H**ierachical **M**ulti-read **A**ttentive **C**lassifier (HMAC).

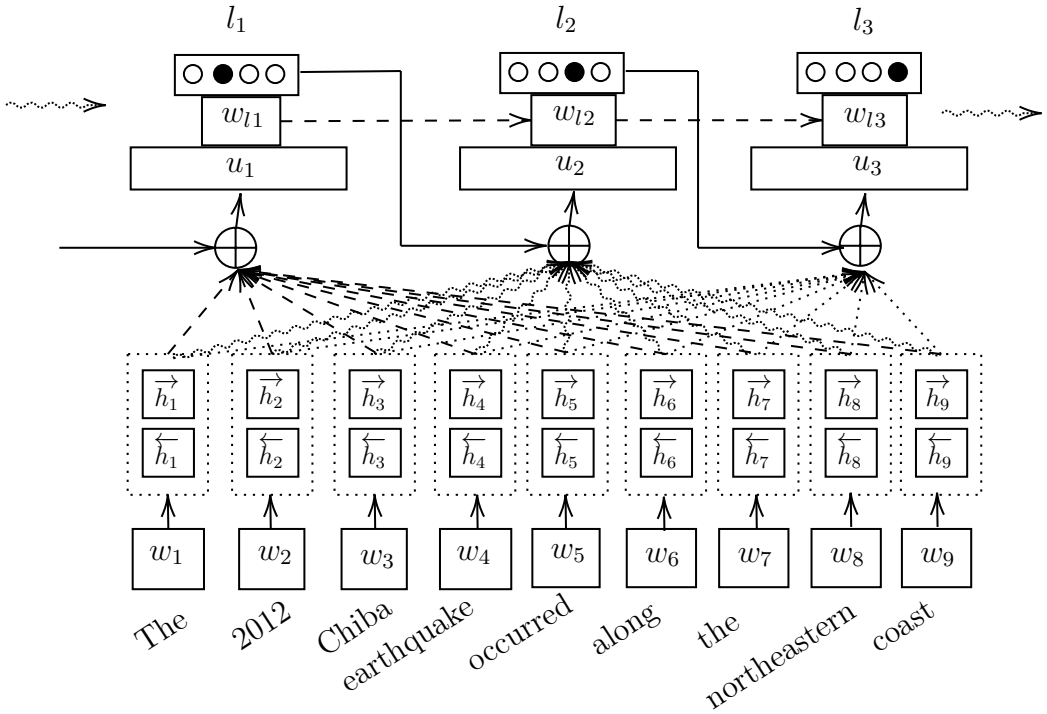


Figure 4.1: Proposed model architecture

4.1 Introduction

The proposed model (Figure 4.1) consists three parts: 1) A bidirectional LSTM encoder (Hochreiter and Schmidhuber 1997) that transforms each word into vector representations based on their context. 2) An attention module that aids generation of dynamic document representations across different level of classification. 3) A multilayer perceptron (MLP) classifier that makes the prediction based on this dynamic document representation and the level masking.

Our hierarchical classification model can be viewed as a sequence-to-sequence autoregressive model, where a sequence of word embeddings are used to generate a sequence of hierarchical class labels. In addition, we employ a modified attention module from the traditional attention mechanism used in the sequential generation tasks (Bahdanau, Cho, and Bengio 2015; Sutskever, Vinyals, and Le 2014a) : instead of computing attention weights conditioned on the hidden state of the decoder at time step i, we condition on the parent category embedding c_{k-1}. This is intuitive in our setting as the document representation should depend on the parent class the model predicted.

4.2 Technical details

4.2.1 Document encoding

Formally, suppose we are given a document with n tokens $D = (w_1, w_2, ..., w_n)$ and its category labels of m levels $C = (c_1, \ldots, c_m), c_k \in \{c_1^{l_k}, \ldots, c_{s_k}^{l_k}\}$ where l_k indicates the k-th level of the class taxonomy and s_k represents the number of classes in level. We suppose w_i and c_i are word embeddings and class embedding[1] respectively, of size u and v. A bidirectional LSTM is first used to capture the dependencies between

[1]Each class c_i is fed through a trainable lookup table to retrieve an unique embedding vector.

adjacent words:

$$\begin{aligned} \overrightarrow{h_t} &= \overrightarrow{LSTM}(w_t, \overrightarrow{h_{t-1}}), \\ \overleftarrow{h_t} &= \overleftarrow{LSTM}(w_t, \overleftarrow{h_{t+1}}). \end{aligned} \tag{4.1}$$

where $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ are the forward and backward pass representation of size $n \times u$. The encoder's hidden states $H = (\mathbf{h_1}, \ldots, \mathbf{h_n})$ are constructed by the concatenation of $(\overrightarrow{h_t})$ and $(\overleftarrow{h_t})$ as $\mathbf{h_i} = [\overrightarrow{h_t}, \overleftarrow{h_t}]$. Thus, the size of $\mathbf{h_i}$ is $n \times 2u$.

When classifying the class label at level k, we first form the contextual word features $\bar{H}$ by concatenating the previously predicted category embedding c_{k-1} (parent) with each of the encoder's outputs $H = (\mathbf{h_1}, \ldots, \mathbf{h_n})$:

$$\bar{H}_k = H \oplus c_{k-1}. \tag{4.2}$$

where, the size of $\bar{H}_k$ is $n \times (2u + v)$.

4.2.2 Pooling with Attention

To form a fixed vector document representation D we have the choice of either using max pooling over the dimensions, or using an attention module. For attention we have a choice between two recent types of attention architectures, Self attention (Z. Lin et al. 2017) and Scaled dot product attention (Vaswani et al. 2017), and we employ the former due to its better empirical results.

For attention, we devise a contextualized self attention which is conditioned over the parent category embedding c_{k-1}. After we get the document encoding, we transform these n vectors in $\bar{H}_k$ into n attention scores (scalars) through a series of linear and non-linear transformation:

$$\mathbf{a}_k = \text{softmax}(\mathbf{w_{s_2}} \tanh(W_{s_1} \bar{H}_k^T)). \tag{4.3}$$

where, W_{s_1} is a weight matrix of size $(2u + v) \times d_a$, $\mathbf{w_{s_2}}$ is a weight vector of size d_a, and $\mathbf{a_k}$ is the attention vector for one hop of size n. d_a is a hyperparameter chosen as explained in Chapter 7.

As one single attention distribution might only focus on a specific component of the semantics in the document, we follow Z. Lin et al. 2017; Vaswani et al. 2017's work to perform m hops of attention and form the multi-head attention matrix A_k $(m \times n)$.

The document representation for level k is obtained by the matrix multiplication of the multi-head attention matrix and the contextual word features:

$$D_k = W_{s_3} A_k \bar{H}_k. \tag{4.4}$$

where, W_{s_3} is the weight matrix of size $m(2u + v) \times d_{hidden}$, where d_{hidden} is the size of document representation D_k.

4.2.3 Classification

Finally, a multi-layer perceptron (MLP) is employed to classify the category at level k. We concatenate the current document represention D_k with previous level document representation d_{k-1}.

$$\begin{aligned} d_k &= \tanh(W_D[D_k, d_{k-1}]), \\ y_k &= \text{softmax}(W_k d_k) \end{aligned} \tag{4.5}$$

where, W_D is the weight matrix of size $2d_{hidden} \times d_{hidden}$, d_{hidden} is the size of the first layer of the MLP responsible to create the intermediate document representation d_k. W_k is the weight matrix for the second layer of MLP of size $d_k \times j$, where j is the sum of all classes across all levels.

Usually, the softmax in equation 4.5 is computed over the all the class labels. For our scenario, we would have to compute the softmax over the entire classes in the serialized taxonomy. This is not desirable when the taxonomy is deep and the number of classes is large. We solve this by employing a *level masking* technique where we mask out all the classes that are not in the current classification level k. We can either have one single end classifier MLP to classify all category levels 0 to k, or we

can have unique classifier layers for each level in order to learn unique parameters per level k.

4.3 Training

Loss is calculated as the joint cross entropy loss among all the levels of the taxonomy:

$$l = \sum_{i=1}^{m} l_i \tag{4.6}$$

Since we are using multi-hop attention, to encourage the diversity over the multiple hops of the attention distributions, we employ Frobenius norm penalty (Z. Lin et al. 2017) to force the attention hops focus on different aspects of the semantics.

$$P = \left\| A_k A_k^\top - I \right\|_F^2 \tag{4.7}$$

which is added to the loss function. We use the standard Adam optimizer (Kingma and Ba 2014) to perform gradient descent with momentum to converge to the optimal parameter space. The details of hyperparameters used in our model is given in Chapter 7.

5

Data collection

In our previous chapter, we looked into the inner workings of our model. In this chapter, we investigate hierarchical datasets to test our model on. The challenge to create such a hierarchical dataset is that we need a knowledge-base taxonomy, which can either be a tree or directed acyclic graph, and we extract the text from the leaf nodes and try to classify them hierarchically.

5.1 Existing Datasets

However, most of the existing datasets are either not available for research or not suitable for deep learning models. Open Directory Project (ODP) [1] was used by researchers for most of the hierarchical classification experiments. This was built as a meta-information aggregator for the entire web and contains hyperlinks of the pages under a taxonomy. This dataset has a hierarchy of 70,000 categories and 4.5M documents associated with the nodes. On the top-level, the dataset has 17 categories such as *Adult*, *Arts*, *Business*, *Computer*, *Games*, *Health*, *Home*, *Kids and Teens*, *News*, *Recreation*, *Reference*, *Regional*, *Science*, *Shopping*, *Society*, *Sports* and *World*. Oh and Myaeng 2014 used this taxonomy to create a smaller dataset of 65,564 categories and 607,944 web pages or documents after pruning incoherent nodes. However, the authors do not make the dataset public. Also, the ODP project was discontinued as

[1] http://www.dmoz.org

of March 2017 [2], so we have no way to extract the information and taxonomies to recreate the dataset.

Another source of hierarchical classification data is the Large Scale Hierarchical Text Classification challenge (Partalas et al. 2015). This challenge was setup in 2015, and the winning team used Multinomial naive Bayes [3] without any hierarchical information, and the contest had best Macro F-score of 0.3391 [4]. The authors used the ODP taxonomy and DBPedia taxonomy to collect a large scale text classification dataset, which consists of 32,056 categories, 2,365,436 training instances, 452,167 test instances. However, the data is presented in a sparse-vector format where the categories and words of the documents are encoded as integers, and the authors did not provide any option to recover the original text back from the encoded representation [5]. This makes the dataset unusable for deep learning models and they are not interpretable.

We therefore used the *Web of Science* dataset originally curated by Kowsari, D. E. Brown, Heidarysafa, Meimandi, et al. 2017. *Web of Science* (Reuters 2012) (WOS) is a hierarchical two-level taxonomy dataset [6] that contains 46,985 documents and 7 top level categories and 134 leaf level categories. The top-level categories include topics such as *Biochemistry, Civil Engineering, Computer Science, Electrical Engineering, Medical Sciences, Mechanical Engineering* and *Psychology*, which contain 9,11,17,16,53,9, and 19 children respectively. This is a tree based taxonomy as it does not have multiple parents for a sub-children. We compare this dataset with our curated data from DBPedia which we describe in the next section.

[2] https://www.ergoseo.com/blog/dmoz-is-shutting-down/

[3] https://www.kaggle.com/c/lshtc/discussion/7980

[4] https://www.kaggle.com/c/lshtc/leaderboard

[5] We contacted the authors separately and they acknowledged the issue, but they couldn't provide the raw text information as the challenge is over 3 years old.

[6] http://dx.doi.org/10.17632/9rw3vkcfy4.6

5.2 Curating Hierarchical dataset

As deep learning models usually contain a large number of parameters that need to be learned, the requirement of large dataset arises and becomes necessary to prevent over-fitting (Lawrence, Giles, and Tsoi 1997; Srivastava et al. 2014). In order to collect a text dataset for hierarchical classification, one first needs to use a taxonomy of categories. We already investigated two such categories before, ODP and LSHTC, but they are not readily available to extract leaf node information. In the next subsection, we investigate few other taxonomies one could potentially use to curate the data.

5.2.1 Leveraging topical hierarchies

To extract topical taxonomies, one needs to go no further than the largest resource of topics and documents, Wikipedia. Wikipedia articles are themselves tagged with meta information about its category and related articles, which can be mined to extract a taxonomy. There are two such existing taxonomies extracted from Wikipedia, Wibi Taxonomy and DBpedia Taxonomy.

5.2.1.1 Wibi Taxonomy

Wibi Taxonomy, or MultiWibi (Flati et al. 2016) is a multilingual Wikipedia bitaxonomy. This is an automatic creation of integreated bitaxonomy over Wikipedia for multiple languages. It is termed as *bitaxonomy* as it consists of a pair of taxonomies, one for Wikipedia pages and one for Wikipedia categories. It uses a novel method to create the taxonomy by parsing the textual definitions of each pages and extracting and disambiguating the hypernym lemma(s), and then linking those hypernyms back to the corresponding categories. The taxonomy can be readily downloadable from their website [7] and which also includes a visual medium to inspect the taxonomies.

[7] http://wibitaxonomy.org/

For example, the node *The Da Vinci Code (film)* presents a hierarchy as depicted in Figure 5.1 which is directly extracted from the website.

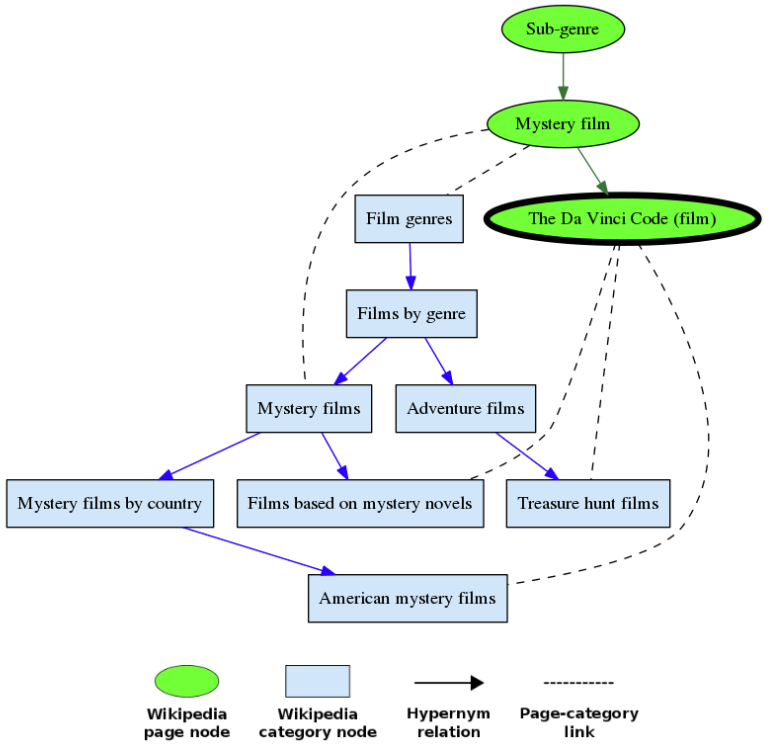


Figure 5.1: Wibi Taxonomy hierarchy for the node *The Da Vinci Code (film)*, as depicted in the website http://wibitaxonomy.org/display.jsp?item=The_Da_Vinci_Code_(film)

Although, we found this taxonomy to be difficult to use to extract a hierarchical information owing to its directed acyclic graph (DAG) structure. The taxonomy contains cycles and most of the nodes link back to the universal parent nodes *Being*, *Concept* and *Idea*. We provide one such sample in Figure 5.2. To make use of this taxonomy one have to carefully prune it to get a semantically related tree structure.

5.2.1.2 DBPedia Taxonomy

DBPedia [8] is a meta information aggregator service over Wikipedia. It provides a category taxonomy which is community curated and thus is manually created covering 2,630,717 Wikipedia pages. Since this is manually created, the size of the

[8] https://wiki.dbpedia.org/

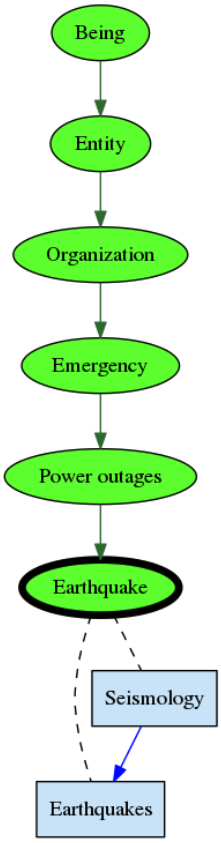


Figure 5.2: Wibi Taxonomy hierarchy for the node *Earthquake*, as depicted in the website http://wibitaxonomy.org/display.jsp?item=Earthquake&lang=EN&type=page&pageH=5&categH=3

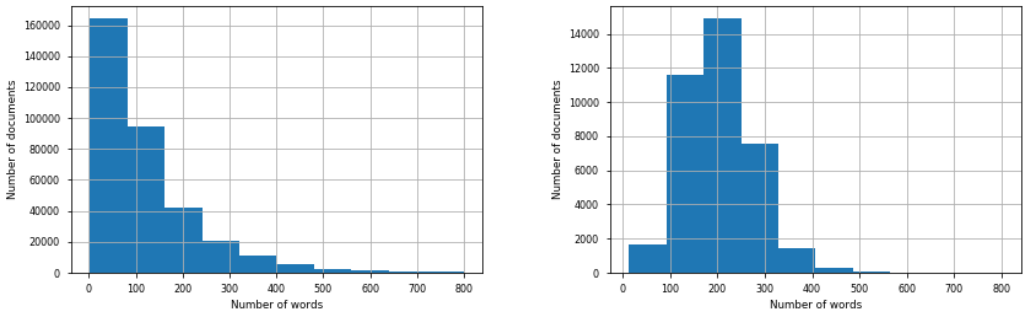
taxonomies is small but it is tree structured and hence is devoid of any self loops and missing tags. The extracted ontology has 24 top level categories spanning the entire Wikipedia, such as *Event*, *SportsSeason*, *Biomolecule*, *Work*, *TopicalConcept*, *AnatomicalStructure*, *GeneLocation*, *MeanOfTransportation*, *Food*, *Species*, *Place*, *Activity*, *Device*, *TimePeriod*, *UnitOfWork*, *ChemicalSubstance*, *SportCompetitionResult*, *Agent*, *PersonFunction*, *Relationship*, *Language*, *Name*, *Award*, and *List*. Along with the ontology DBpedia also provides the long abstracts of the corresponding leaf nodes of these categories.

	DBpedia	WOS
Level 1 Categories	9	7
Level 2 Categories	70	134
Level 3 Categories	219	NA
Number of documents	381,025	46,985
Mean document length	106.9	200.7

Table 5.1: Dataset Comparison

5.2.2 Extracting documents from DBPedia taxonomy

Thus, we curated a bigger dataset with hierarchical labels from DBpedia owing to its relative simplicity, ease of use, and absence of cycles. Compared to Web of Sicence, our DBpedia dataset is larger in two aspects: the number of data instances and the number of hierarchical levels (Table 5.1). DBpedia ontology was first used in X. Zhang and LeCun 2015 for flat text classification. We instead use the DBpedia ontology to construct a dataset with three-level taxonomy of the classes.



((a)) DBpedia ((b)) Web of Science

Figure 5.3: Dataset length statistics

We preprocess the taxonomy to only extract the trees which have more than three levels. Then, we collapse the leaf levels up so that we only end up with three levels. Concretely, if we encounter a tree such as $l_0 \rightarrow l_1 \rightarrow ... \rightarrow l_n$, where $n > 3$, then we shorten the tree such that we keep the first top level and keep the last two levels to form the tree $l_0 \rightarrow l_{n-1} \rightarrow l_n$. We end up with 486 unique trajectories from the roots to the leaf categories. Then, we collect the long abstracts from the leaf nodes of these

trajectories. In order to ensure enough documents are presented per-class, we only extract leaf-classes with more than 200 documents. We also limit an upper bound of 3,000 documents per category to balance the number of leaf-level categories. We keep documents having word length less than 1000 in our dataset. This results in 381,025 documents in total, which we split into 90% for training (from which 10% kept aside for validation) and 10% on testing, on which we report our classification metrics.

Although, since we collect only the long abstracts, the mean length of the documents is only 100 words, which is about half that of *Web of Science* (Figure 5.3), which makes the classification task relatively easier for the *flat* classifiers, as we see in Chapter 7. One can therefore choose to include the entire document rather than the long abstract to make the classification task more challenging.

6

Baselines

In the previous chapter we explained our choice of the datasets and how we curated them. In this section we will revisit some of the baseline models we would pitch our model against.

To select a baseline, we have two-fold objective. One, we want to explore whether using an external knowledge like hierarchical taxonomy helps in classification of large number of documents. And two, we explore whether our method is better or comparable to other hierarchical classification approaches. For the former, state-of-the-art *flat* classifiers such as FastText (Joulin et al. 2017), Bidirectional LSTM with max/mean pooling (Collobert and Weston 2008; Lee and Dernoncourt 2016) and Self-attentive classifier (Z. Lin et al. 2017) are used for the comparison. For the latter, we use the only available neural hierarchical classifier HDLTex (Kowsari, D. E. Brown, Heidarysafa, Meimandi, et al. 2017) to compare against our approach.

6.1 Flat classifier baselines

6.1.1 FastText

FastText (Joulin et al. 2017) is a simple yet powerful baseline for text classification task. Concretely, the model is a one layer multi-layer perceptron which takes averaged word representations from the document and uses a linear classifier to predict the

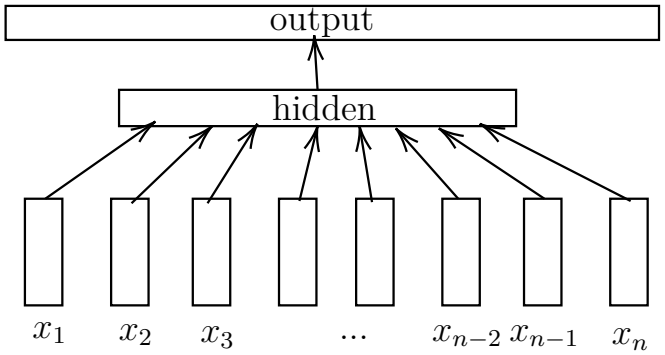


Figure 6.1: Fasttext model. Figure adapted from Joulin et al. 2017

class. The word representations are typically n-gram features which are represented as a word embedding similar to Mikolov, Sutskever, et al. 2013. Then, a softmax function is used to compute probability distribution over predefined classes. When the number of classes is large, it uses hierarchical softmax (Goodman 2001) based on Huffman coding tree (Mikolov, Sutskever, et al. 2013). This simple architecture has been reported to outperform state-of-the-art CNN architectures on various flat classification tasks.

6.1.2 BiLSTM with Pooling

BiLSTM with mean or max pooling has been used successfully by Collobert and Weston 2008 to classify documents. Specifically, if we have a sequence of m words, a bidirectional LSTM computes a set of hidden representations h_t for each of those words in *both* the directions as depicted in Equation 4.1. Then, for each word we concatenate the forward $\overrightarrow{h_t}$ and backward $\overleftarrow{h_t}$ to form $h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$. To get a fixed size vector, either we can select the maximum value over each hidden units, which is termed as max pooling (Figure 2.1). Or, we can consider the average over all units for all dimensions, which is termed as mean pooling. This representation is then fed to a multi-layer perceptron (MLP) to predict the softmax over the classes. There is another popular choice of pooling mechanism named *concat pooling*, which is a concatenation of max pooled and mean pooled representation: $[max, mean]$.

6.1.3 Self-attentive classifier

The self-attentive classifier (Z. Lin et al. 2017) uses an attention mechanism over a BiLSTM hidden states instead of pooling. Concretely, given the concatenated hidden representations over n words in $H = (h_1, h_2, ...h_n)$, the self attention mechanism computes an attention matrix by calculating the following over multiple hops r:

$$A = softmax(W_{s_2} tanh(W_{s_1} H^T)) \tag{6.1}$$

where W_{s_1} and W_{s_2} are weight matrix. The model also uses Frobenius norm or L^2 penalty to make sure the attention hops are diverse among each other. Since we heavily use this architecture for our model it makes sense to compare with this baseline.

6.2 Hierarchical classifier baseline

6.2.1 HDLTex

To compare our model with a hierarchical neural classification model we use HDLTex (Kowsari, D. E. Brown, Heidarysafa, Meimandi, et al. 2017). HDLTex uses a *top-down* classification approach where it creates a neural classifier, either a Convolutional Neural Network (CNN) or a Recurrent Neural Network (RNN) at each parent node in the taxonomy, as depicted in Figure 6.2. Thus, each classifier is tasked to predict within the children of the particular parent node, and the classifiers in combination predict the leaf nodes. While the parent level Deep neural network (DNN) is trained with all the documents, the subsequent levels DNN's are trained only with the documents of the specified domain. We compare our results with this model in Chapter 7.

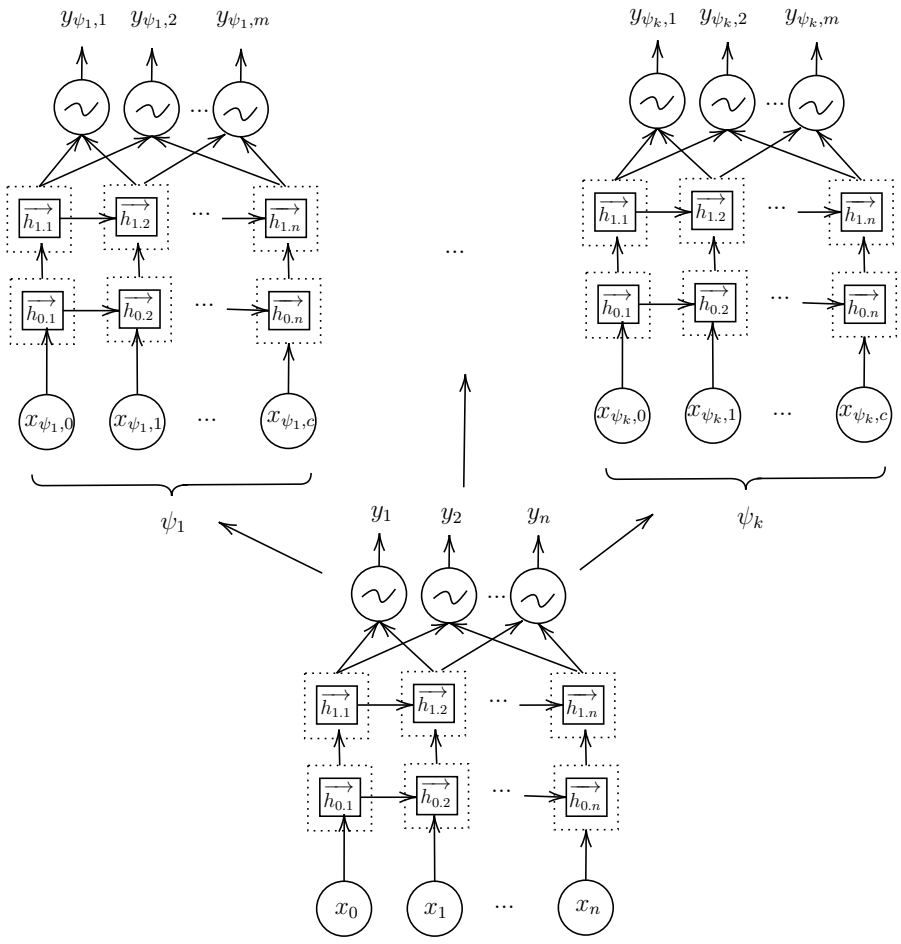


Figure 6.2: HDLTex model. Figure adapted from Kowsari, D. E. Brown, Heidarysafa, Meimandi, et al. 2017

7

Empirical Evaluation

In the previous chapter, we looked into the choice of baselines we want to compare our model against. In this chapter, we present the empirical results of our model against the baselines, and we discuss and analyse our results.

7.1 Experimental Setup

7.1.1 Hyperparameters

In our experiments, we use 300-dimensional word embeddings which are randomly initialized and fine-tuned during training. Two-layer Bidirectional LSTM with 300 hidden units in each layer are employed. In multi-head attention mechanism, we use 15 heads (hops) with Frobenius norm penalty as it gives the best validation performance. The final fully-connected MLP layer has 300 hidden units, and in our best performing model we use individual MLP layer k for each level in the taxonomy (i.e 2 layers for WOS and 3 layers for DBPedia). In addition, we add 0.2 dropout on BiLSTM layers and the MLP layers to prevent over-fitting.

For the optimization, we use standard Adam optimizer (Kingma and Ba 2014) with weight decay of 10^{-4} and 10^{-6} for WOS and DBpedia datasets respectively. The gradients are clipped to 0.5 in order to prevent the gradient from explosion. All the results are obtained after 20 epochs of training. In addition, weighted loss function is

utilized to balance the performance on under-represented classes. For learning rate, we use the Slanted Triangular Learning Rate (SLTR) from Howard and Ruder 2018a; Smith 2017 which is suitable for quickly converging to a region of parameter space in the beginning of the training and then refining the parameters. Concretely, the SLTR uses a maximum learning rate of 0.01 and after a short increase it decays iteratively to the learning rate of 0.0001.

Detailed ablation study is provided in Section 7.4.

7.1.2 Preprocessing

Preprocessing is very important for any text classification system. We use similar preprocessing pipeline as Kowsari, D. E. Brown, Heidarysafa, Meimandi, et al. 2017 to ensure fairness. Concretely, we prune the documents which has more than 1000 words out of the dataset. Then, we remove special characters such as delimiters, parenthesis, and mathematical operators. We also remove consecutive spaces, hyperlinks and trailing / starting whitespace. Finally, we lowercase all the text documents.

7.2 Results & Discussion

	DBpedia				WOS		
Flat Baselines				Overall			Overall
FastText				86.2			61.3
BiLSTM + MLP + Maxpool				94.20			77.69
BiLSTM + MLP + Meanpool				94.68			73.08
Structured Self Attention				94.04			77.40
Hierarchical Models	l_1	l_2	l_3	Overall	l_1	l_2	Overall
HDLTex (5B params [1])	99.26	97.18	95.5	92.10	90.45	84.66	76.58
Our model (34M params)	99.21	97.12	95.32	93.72	89.32	82.42	77.46

Table 7.1: Test accuracy results on WOS and DBpedia datasets

[1] 5B params on the model is for the case of DBpedia dataset

7.2.1 Comparative results

Table 7.1 shows the results from our experiments. The flat baseline models are trained without the hierarchical taxonomy of classes and therefore only have results on the leaf-node classification. Both hierarchical classifiers (ours and HDLTex) perform comparably or slightly better than state-of-the-art flat classifiers, which indicates that the external knowledge on class taxonomy has the potential to improve the classification performance marginally.

Interestingly, the class taxonomy seems to be more beneficial in boosting the performance of hierarchical classifiers on DBpedia than WOS. We observe this behaviour due to the dataset of DBpedia being shorter in average length making it easier to classify for *flat* classifiers, hence hierarchical classifiers overfit on the training data. Compared to HDLTex, our model has a significant performance improvement on the DBpedia dataset, and comparable performance on the WOS dataset.

7.2.2 Classifier complexity

Moreover, our model takes significantly less time and resources to train, especially when the dataset is large in terms of the number of taxonomy classes other than the leaf-node. As HDLTex needs to build one sub-classifier for each parent nodes, the number of sub-classifiers grows quickly. For example, there are 80 parent nodes in the taxonomy of the DBpedia dataset and HDLTex needs to build 80 RNNs, where each sub-classifier contains around 67 million parameters. As a consequence, we can barely fit the whole model of HDLTex on our CPU [1] because it requires 60 GB RAM to build these 80 deep neural networks.

[1]It is not possible to fit the entire model in one GPU as our best GPU has the RAM capacity of 12GB, one needs to have multiple GPU's and parallel execution for this task.

7.2.3 Error Analysis

However, we also analyze the error of the predicted classes and notice a significant advantage of using a hierarchical approach over flat classification approach. We perform error analysis in two stages. First, we analyze the hierarchical nature of the errors in general to qualitatively show how robust our model is. Secondly, we perform human evaluation over the errors generated by our model and the baseline to assess the complexity of the test samples from human perspective and compare the performance with our model and flat classification baseline.

7.2.3.1 Qualitative analysis

Qualitatively, our hierarchical approach outperforms the best performing flat classifier in terms of getting the correct sub-tree class (Table 7.2). Concretely, since we know the taxonomy beforehand, we can calculate whether predicted class c_k in l_k lies in the same subtree of its parent l_{k-1}. We calculate this in two scenarios: When we know the correct parent l_{k-1} class and when the parent class is predicted by our model. In both the scenarios, our hierarchical approach, while being comparable to a flat classification approach, is significantly better at its failure cases where it gets the correct subtree in more number of occasions. While we use the hierarchical taxonomy, since we use the *level masking* technique (refer Section 4.2.3), we are not explicitly informing the classifier of the correct subtree, and its a fair comparison. This is especially important for production classification systems as they are more sensitive to error handling.

Classifier	Correct parent	Predicted parent
Flat classifier - BiLSTM Max Pooling	90.74	85.56
Hierarchical approach - Our model	98.33	88.57

Table 7.2: Effect of taxonomy in error analysis on Web of Science dataset among the two approaches. We analyze the error as number of times the predicted class is within the same sub-tree as the parent.

7.2.3.2 Human Evaluation of Errors

Classifier	Approximately correct
Flat classifier - BiLSTM Max Pooling	39.0
Hierarchical approach - Our model	49.25

Table 7.3: Analysis of errors generated by the models according to human evaluation.

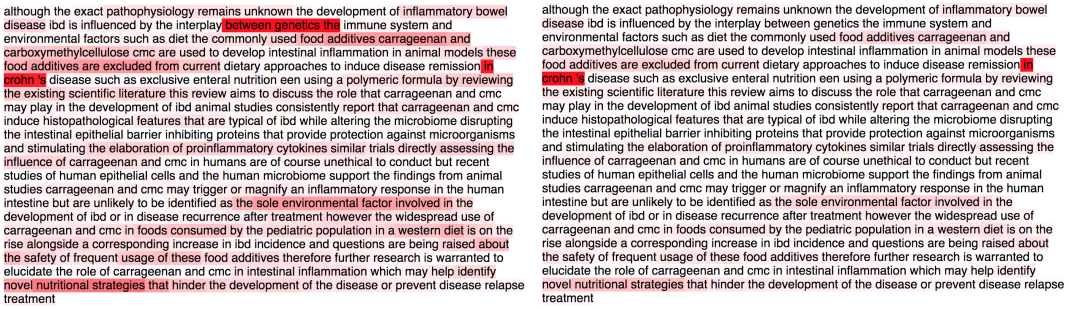
Metrics such as *accuracy*, *recall* and *F-score* offer a qualitative evaluation of model performance. However, these do not give insight into the underlying complexity of the data. For example, in the *Web of Science* dataset, a particular topic about *Image processing* presented here, which is mis-labeled as *Machine Learning* by our classifier.

> The development of automated morphological classification schemes can successfully distinguish between morphological types of galaxies and can be used for studies of the formation and subsequent evolution of galaxies in our universe. In this paper, we present a new automated machine supervised learning astronomical classification scheme based on the Nonnegative Matrix Factorization algorithm. This scheme is making distinctions between all types roughly corresponding to Hubble types such as elliptical, lenticulars, spiral, and irregular galaxies. The proposed algorithm is performed on two examples with different number of image (small dataset contains 110 image and large dataset contains 700 images). The experimental results show that galaxy images from EFIGI catalog can be classified automatically with an accuracy of similar to 93% for small and similar to 92% for large number. These results are in good agreement when compared with the visual classifications.

When a human reader reads through this example, it is clear that the misclassification is not exactly wrong, as the abstract also falls in *Machine Learning* subclass. Therefore, to address this data complexity, we perform human evaluation of incorrect model predictions. We use human annotators to label each prediction from our models into either of two categories : *approximately correct* and *wrong*. We take a sample of 200 rows from the *Web of Science* dataset to do this labelling. We find that 39% of

the incorrect predictions of the flat classifier are tagged as *approximately correct* by our human evaluators, while for our hierarchical classifier the score is 49.25%. This analysis shows the need to closely examine classifier error : the classifiers often provide reasonable results even when not perfectly matching test data. Further more, we find that even in a human evaluation our model achieves a significant advantage over a flat classifier. We attribute this to it's hierarchical nature. Even if it fails to correctly classify an article it still have a higher chance to get the parent class correct because of the k-step classification. This will lead the classifier to choose the next semantically best class which lies under the correct parent class.

7.3 Analysis of Attention



((a)) Level 1 - correct class : Medical ((b)) Level 2 - correct class : Crohn's disease

Figure 7.1: WOS dataset attention rereading per level.

7.3.1 Effect of multi-level attention

The intuition behind building dynamic document representations, using multiple attentions across different hierarchical levels, is to have a re-reading effect over the taxonomy. When we first encounter an article as humans, we tend to read it carefully, but on subsequent reads, we can easily identify the key aspects of the article. We find in our exploratory experiments the attention vectors behave exactly the same. For the first level, the attention values are more spread out to help our classifier to pick

up more important aspects of the article, but on the subsequent levels the attention is more focused towards specific keywords for that subclass, as the example from WOS shows [2] in Figure 7.1.

7.3.2 Qualitative Analysis

To qualitatively analyze the focus of attention, we show the difference in mean attentions from parent level to child level using Euclidean distance (L^2 norm), and we observe that trees in the taxonomy having more number of children has higher difference in attention spread (Figure 7.2). To measure the decrease in spread in the next level, we turn to statistical metrics like Kurtosis (Mardia 1970) which measures the tailedness of a distribution, and we show the increase in tailedness of the attentions in l_2 w.r.t l_1, quantifying the narrowing of focus throughout the test dataset.

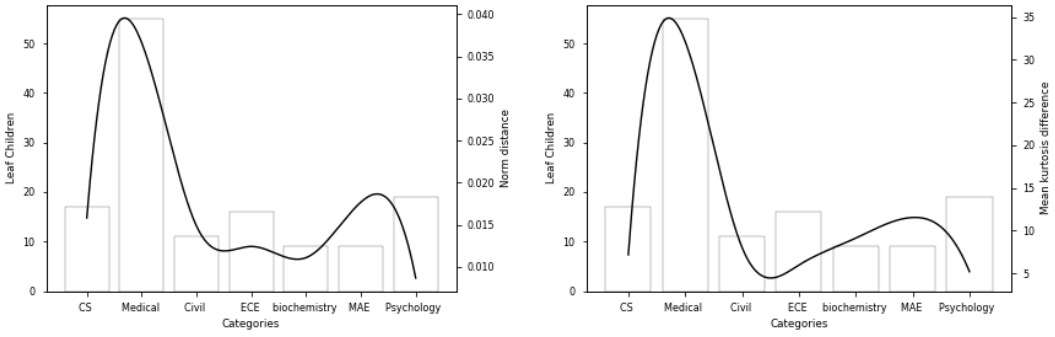


Figure 7.2: Difference in attentions among levels ($l_2 - l_1$) using Euclidean distance & Kurtosis. The bar chart represents number of children within that parent category.

7.3.3 Semantic analysis

Finally, we analyze the words focused by our attention mechanism. In Table 7.4, we show the top 10 highly attended words per category in *Web of Science* dataset, which shows that the model is clearly able to pick meaningful discriminative words. We also further analyze the discriminative words picked up by the next level classification as

[2] We use the same visualization script as of Z. Lin et al. 2017.

CS	Civil	ECE	MAE	Medical	Psychology	Biochemistry
algorithm	ambient	control	aided	allergies	antisocial	cell
computer	cablestayed	digital	computeraided	and	borderline	creatinine
data	construction	electric	fluid	angioedema	eating	enzymology
distributed	geotextile	electrical	hydraulic	birth	false	genetics
network	green	lorentz	industrial	digestive	gender	human
operating	rainwater	microcontroller	machine	fungal	nonverbal	molecular
parallel	smart	operational	manufacturing	menopause	prejudice	northern
relational	stealth	satellite	materials	senior	prenatal	should
software	suspension	space	strength	skin	problemsolving	southern
symbolic	water	statespace	surface	weight	thirdperson	the

Table 7.4: Top 10 words in each domain in WOS having highest attention

Image Processing	Relational databases	Algorithm design	Symbolic computation	Parallel computing
algorithm	algebra	algorithm	darboux	06
camera	campaign	architecture	for	computer
edges	data	at	inhomogeneous	digital
image	database	attention	lie	image
imageprocessing	distributed	can	machine	machine
images	modelling	competitiveness	maple	molecular
malignant	nonrelational	into	model	multicore
many	problem	multistage	physical	parallel
message	reaction	stable	rational	partition
the	relational	very	symbolic	rectangular

Table 7.5: Top 10 words in five randomly sampled sub-domain in Computer Science, having the highest attention

we see a sample of such words in Table 7.5. While being comparatively generic, the specific level classifier attention module still picks up relevant semantic words for the subdomain in question.

7.4 Ablation Study

We also perform extensive ablation studies on our model to see the effect of various components of our architecture. Ablation study provides an unique way to see which parts of our model could have contributed to enhancing performance of the model, and we discuss on which structures worked and which did not. The results are given in Table 7.6. We perform all our ablation studies on Web of Science dataset because it is proven to be more difficult to classify (Section 7.2).

We test our model selection with various modifications. We experimented with a one-hot parent encoding which represents the parent class in a vector of size k where

Architecture		WOS		
		l_1	l_2	Overall
Our Model		89.32	82.42	77.46
Attention	Without previous layer encoding	88.82	79.21	75.97
	Without BiLSTM encoder - pure attention	86.56	79.60	72.09
	With single final classifier	86.69	76.78	71.83
	With parent encoding	88.57	82.66	76.83
	With low attention hops - 2	89.15	78.80	74.99
	With high attention hops - 15	88.71	78.62	74.65
	With attention penalty	82.09	50.43	45.38
Pooling	Without attention - max pooling	88.37	77.39	77.39
	Without attention - mean pooling	87.69	73.59	73.59
	Without attention - concat pooling	88.63	80.92	77.28
	Without BiLSTM encoder - pure concat pooling	85.59	73.01	73.01

Table 7.6: Ablation Study with various architecture changes on WOS dataset

k is the total number of parent classes. We also experimented with increasing and decreasing attention hops. Lastly we experimented with alternative pooling mechanisms such as max pooling, mean pooling and concat pooling (Collobert and Weston 2008). Refer to Chapter 6 for a review these pooling mechanisms.

We observe that for *Web of Science* dataset, we only get a marginal gain by using attention with respect to concat pooling. This can be attributed to the inherent mechanism of pooling. Attention mechanisms focus on certain words by either increasing or decreasing the vector representation of the words, while a pooling mechanism like *concat pooling* achieves comparable performance by identifying the discriminative *dimension* of the word representation across all words. Thus, a pooling mechanism has an advantage of using the raw representations over all the words to identify discriminative signals. Although, we do acknowledge the relatively small size of *Web of Science* dataset, which can be a deciding factor why attention mechanisms perform at most comparably.

8

Conclusion

After presenting our model, experiments and discussing about the results, in this chapter we conclude our thesis with a short summary of contributions, limitation and future work.

8.1 Summary of Contributions

In this work, we propose a light-weight neural-based hierarchical classifier that performs better than or comparable to the state-of-the-art hierarchical model with a significantly fewer computational resources. Our model employs an adapted version of attention to representing document dynamically throughout the hierarchy, which provides the additional interpretability with the semantic granularity of the dynamic document representations. In addition, we explore the possibility of improving flat text classification using an external knowledge such as a hierarchical taxonomy.

While the hierarchical approach provides limited advantage with respect to a flat classifier for Web of Science and DBpedia datasets, we empirically show that the error handling of our hierarchical approach is significantly better as even when the classifier misses the correct class the prediction is within the subtree of the predicted as well as the actual parent classes. This shows that our approach is more robust.

8.2 Limitations

One of the major limitations of our model is that it is still not perfectly *global*, as we had to employ a per level unique classifier MLP layer. This still makes the model grow in the number of parameters as we work on deeper and deeper levels, although magnitudes lesser compared to the state-of-the-art hierarchical models.

8.3 Future Work

A natural future direction would be to advance our model to automatically construct the hierarchical taxonomy in order to improve text classification with a large number of classes. This would require building a taxonomy graph from scratch and then perform classification. Recent advancements in neural graph representation (Hamilton, Ying, and Leskovec 2017), construction (D. D. Johnson 2016) and decoding (Xu et al. 2018) can be leveraged to construct an automatic end-to-end hierarchical classifier.

Another future direction would be to explicitly leverage the taxonomy to create a novel hierarchical loss function. The current approach still uses the negative log-likelihood loss to increase the softmax probability of the correct class while decreasing the probability of *all* classes uniformly. However, this poses a problem when the number of classes is large and the softmax probabilities are more *flat* or less peaky. In order to use the hierarchical information, one possible way would be to explicitly decrease the probability of the out of subtree classes by rewriting the equation of log-likelihood loss with an inspiration from maximum margin loss. Alternatively, we can perform a similar operation by using a non-differentiable auxiliary loss function. We could use REINFORCE (Williams 1992) to train using temporal difference loss similar to the approaches commonly used in Reinforcement Learning (Sutton and Barto 1998).

Bibliography

Adel, Heike, Ngoc Thang Vu, and Tanja Schultz (2013). "Combination of recurrent neural networks and factored language models for code-switching language modeling". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vol. 2, pp. 206–211.

Allahyari, Mehdi et al. (2017). "A brief survey of text mining: Classification, clustering and extraction techniques". In: *arXiv preprint arXiv:1707.02919*.

Apté, Chidanand, Fred Damerau, and Sholom M Weiss (1994a). "Automated learning of decision rules for text categorization". In: *ACM Transactions on Information Systems (TOIS)* 12.3, pp. 233–251.

— (1994b). "Automated learning of decision rules for text categorization". In: *ACM Transactions on Information Systems (TOIS)* 12.3, pp. 233–251.

Arthur, P., G. Neubig, and S. Nakamura (2016). "Incorporating Discrete Translation Lexicons into Neural Machine Translation". In: *ArXiv e-prints*.

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

— (2015). "Neural machine translation by jointly learning to align and translate". In:

Bengio, Yoshua, Réjean Ducharme, et al. (2003). "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb, pp. 1137–1155.

Bengio, Yoshua, Paolo Frasconi, and Patrice Simard (1993). "The problem of learning long-term dependencies in recurrent networks". In: *Neural Networks, 1993., IEEE International Conference on*. IEEE, pp. 1183–1188.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

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

Binder, Alexander, Motoaki Kawanabe, and Ulf Brefeld (2009). "Efficient classification of images with taxonomies". In: *Asian Conference on Computer Vision*. Springer, pp. 351–362.

Blockeel, Hendrik et al. (2006). "Decision trees for hierarchical multilabel classification: A case study in functional genomics". In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, pp. 18–29.

Bolukbasi, Tolga et al. (2016). "Man is to computer programmer as woman is to homemaker? debiasing word embeddings". In: *Advances in Neural Information Processing Systems*, pp. 4349–4357.

Bottou, Léon (1998). "Online Algorithms and Stochastic Approximations". In: *Online Learning and Neural Networks*. Ed. by David Saad. revised, oct 2012. Cambridge, UK: Cambridge University Press.

Brown, Peter F et al. (1992). "Class-based n-gram models of natural language". In: *Computational linguistics* 18.4, pp. 467–479.

Burred, Juan José and Alexander Lerch (2003). "A hierarchical approach to automatic musical genre classification". In: *Proceedings of the 6th international conference on digital audio effects*. Citeseer, pp. 8–11.

Cai, Lijuan and Thomas Hofmann (2004). "Hierarchical document categorization with support vector machines". In: *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. ACM, pp. 78–87.

Chakrabarti, Soumen et al. (1998). "Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies". In: *The VLDB journal* 7.3, pp. 163–178.

Chen, Yubo et al. (2015). "Event extraction via dynamic multi-pooling convolutional neural networks". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Vol. 1, pp. 167–176.

Cheng, Jianpeng, Li Dong, and Mirella Lapata (2016). "Long short-term memory-networks for machine reading". In: *arXiv preprint arXiv:1601.06733*.

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

Clare, Amanda and Ross D King (2003). "Predicting gene function in Saccharomyces cerevisiae". In: *Bioinformatics* 19.suppl_2, pp. ii42–ii49.

Collobert, Ronan and Jason Weston (2008). "A unified architecture for natural language processing: Deep neural networks with multitask learning". In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 160–167.

Collobert, Ronan, Jason Weston, et al. (2011). "Natural language processing (almost) from scratch". In: *Journal of Machine Learning Research* 12.Aug, pp. 2493–2537.

Conneau, Alexis, Douwe Kiela, et al. (2017). "Supervised learning of universal sentence representations from natural language inference data". In: *arXiv preprint arXiv:1705.02364*.

Conneau, Alexis, Holger Schwenk, et al. (2017). "Very deep convolutional networks for text classification". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Vol. 1, pp. 1107–1116.

Costa, Eduardo P et al. (2007). "Comparing several approaches for hierarchical classification of proteins with decision trees". In: *Brazilian Symposium on Bioinformatics*. Springer, pp. 126–137.

DeCoro, Christopher, Zafer Barutcuoglu, and Rebecca Fiebrink (2007). "Bayesian Aggregation for Hierarchical Genre Classification." In: *ISMIR*. Vienna, pp. 77–80.

Denil, Misha et al. (2014). "Modelling, visualising and summarising documents with a single convolutional neural network". In: *arXiv preprint arXiv:1406.3830*.

Dimitrovski, Ivica et al. (2011). "Hierarchical annotation of medical images". In: *Pattern Recognition* 44.10-11, pp. 2436–2449.

Dong, Li et al. (2015). "Question answering over freebase with multi-column convolutional neural networks". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Vol. 1, pp. 260–269.

Dumais, Susan et al. (1998). "Inductive learning algorithms and representations for text categorization". In: *Proceedings of the seventh international conference on Information and knowledge management*. ACM, pp. 148–155.

Dyer, Chris et al. (2015). "Transition-based dependency parsing with stack long short-term memory". In: *arXiv preprint arXiv:1505.08075*.

El Hihi, Salah and Yoshua Bengio (1996). "Hierarchical recurrent neural networks for long-term dependencies". In: *Advances in neural information processing systems*, pp. 493–499.

Elman, Jeffrey L (1990). "Finding structure in time". In: *Cognitive science* 14.2, pp. 179–211.

Firth, John R (1957). "A synopsis of linguistic theory, 1930-1955". In: *Studies in linguistic analysis*.

Flati, Tiziano et al. (2016). "Multiwibi: The multilingual wikipedia bitaxonomy project". In: *Artificial Intelligence* 241, pp. 66–102.

Freitas, Alex and André Carvalho (2007). "A tutorial on hierarchical classification with applications in bioinformatics". In: *Research and trends in data mining technologies and applications*. IGI Global, pp. 175–208.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep sparse rectifier neural networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323.

Goller, Christoph and Andreas Kuchler (1996). "Learning task-dependent distributed representations by backpropagation through structure". In: *Neural Networks, 1996., IEEE International Conference on*. Vol. 1. IEEE, pp. 347–352.

Goodman, Joshua (2001). "Classes for fast maximum entropy training". In: *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*. Vol. 1. IEEE, pp. 561–564.

Graves, Alex (n.d.). "Supervised sequence labelling with recurrent neural networks. 2012". In: *ISBN 9783642212703. URL http://books. google. com/books*.

Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton (2013). "Speech recognition with deep recurrent neural networks". In: *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, pp. 6645–6649.

Gu, J. et al. (2016). "Incorporating Copying Mechanism in Sequence-to-Sequence Learning". In: *ArXiv e-prints*.

Hamilton, William L, Rex Ying, and Jure Leskovec (2017). "Inductive Representation Learning on Large Graphs". In:

Han, Eui-Hong Sam, George Karypis, and Vipin Kumar (2001). "Text categorization using weight adjusted k-nearest neighbor classification". In: *Pacific-asia conference on knowledge discovery and data mining*. Springer, pp. 53–65.

Henderson, Peter et al. (2017). "Ethical Challenges in Data-Driven Dialogue Systems". In: *arXiv preprint arXiv:1711.09050*.

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

Howard, Jeremy and Sebastian Ruder (2018a). "Fine-tuned Language Models for Text Classification". In:

— (2018b). "Fine-tuned Language Models for Text Classification". In: *CoRR* abs/1801.06146.

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016). "Deep Learning". Book in preparation for MIT Press.

Irsoy, Ozan and Claire Cardie (2014). "Opinion mining with deep recurrent neural networks". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 720–728.

Iyyer, Mohit et al. (2014). "A neural network for factoid question answering over paragraphs". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 633–644.

Joachims, Thorsten (1999). "Transductive inference for text classification using support vector machines". In: *ICML*. Vol. 99, pp. 200–209.

Johnson, Daniel D (2016). "Learning Graphical State Transitions".

Johnson, Rie and Tong Zhang (2014). "Effective use of word order for text categorization with convolutional neural networks". In: *arXiv preprint arXiv:1412.1058*.

Joulin, Armand et al. (2017). "Bag of Tricks for Efficient Text Classification". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, pp. 427–431.

Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom (2014). "A convolutional neural network for modelling sentences". In: *arXiv preprint arXiv:1404.2188*.

Kim, Sang-Bum et al. (2006). "Some effective techniques for naive bayes text classification". In: *IEEE transactions on knowledge and data engineering* 18.11, pp. 1457–1466.

Kim, Yoon (2014). "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882*.

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

Koller, Daphne and Mehran Sahami (1997). *Hierarchically classifying documents using very few words*. Tech. rep. Stanford InfoLab.

Kowsari, K., D. E. Brown, M. Heidarysafa, K. Jafari Meimandi, et al. (2017). "HDLTex: Hierarchical Deep Learning for Text Classification". In: *ArXiv e-prints*.

Kowsari, Kamran, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, et al. (2017). "HDLTex: Hierarchical Deep Learning for Text Classification". In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 364–371.

Kriegel, Hans-Peter et al. (2004). "Using support vector machines for classifying large sets of multi-represented objects". In: *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, pp. 102–113.

Labrou, Yannis and Tim Finin (1999). "Yahoo! as an ontology: using Yahoo! categories to describe documents". In: *Proceedings of the eighth international conference on Information and knowledge management*. ACM, pp. 180–187.

Lai, Siwei et al. (2015). "Recurrent Convolutional Neural Networks for Text Classification." In: *AAAI*. Vol. 333, pp. 2267–2273.

Lawrence, Steve, C Lee Giles, and Ah Chung Tsoi (1997). "Lessons in neural network training: Overfitting may be harder than expected". In: *AAAI/IAAI*. Citeseer, pp. 540–545.

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

Lee, Ji Young and Franck Dernoncourt (2016). "Sequential short-text classification with recurrent and convolutional neural networks". In: *arXiv preprint arXiv:1603.03827*.

Lin, Tsungnan et al. (1998). *Learning long-term dependencies is not as difficult with NARX recurrent neural networks*. Tech. rep.

Lin, Zhouhan et al. (2017). "A structured self-attentive sentence embedding". In: *arXiv preprint arXiv:1703.03130*.

Liu, Shaohui et al. (2001). "An approach of multi-hierarchy text classification". In: *Info-tech and Info-net, 2001. Proceedings. ICII 2001-Beijing. 2001 International Conferences on*. Vol. 3. IEEE, pp. 95–100.

Liu, Tie-Yan et al. (2005). "Support vector machines classification with a very large-scale taxonomy". In: *Acm Sigkdd Explorations Newsletter* 7.1, pp. 36–43.

Liu, Yang et al. (2016). "Learning natural language inference using bidirectional LSTM model and inner-attention". In: *arXiv preprint arXiv:1605.09090*.

Lodhi, Huma et al. (2002). "Text classification using string kernels". In: *Journal of Machine Learning Research* 2.Feb, pp. 419–444.

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

Mardia, Kanti V (1970). "Measures of multivariate skewness and kurtosis with applications". In: *Biometrika* 57.3, pp. 519–530.

McCallum, Andrew, Kamal Nigam, et al. (1998a). "A comparison of event models for naive bayes text classification". In: *AAAI-98 workshop on learning for text categorization*. Citeseer.

— (1998b). "A comparison of event models for naive bayes text classification". In: *AAAI-98 workshop on learning for text categorization*. Vol. 752. 1. Citeseer, pp. 41–48.

Merity, Stephen et al. (2016). "Pointer sentinel mixture models". In: *arXiv preprint arXiv:1609.07843*.

Mikolov, Tomas, Kai Chen, et al. (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Mikolov, Tomas, Ilya Sutskever, et al. (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*, pp. 3111–3119.

Mou, Lili et al. (2015). "Discriminative neural sentence modeling by tree-based convolution". In: *arXiv preprint arXiv:1504.01106*.

Mozer, Michael C (1992). "Induction of multiscale temporal structure". In: *Advances in neural information processing systems*, pp. 275–282.

Mrkšić, Nikola et al. (2015). "Multi-domain dialog state tracking using recurrent neural networks". In: *arXiv preprint arXiv:1506.07190*.

Oh, Heung-Seon, Yoonjung Choi, and Sung-Hyon Myaeng (2010). "Combining global and local information for enhanced deep classification". In: *Proceedings of the 2010 ACM symposium on applied computing*. ACM, pp. 1760–1767.

— (2011). "Text classification for a large-scale taxonomy using dynamically mixed local and global models for a node". In: *European Conference on Information Retrieval*. Springer, pp. 7–18.

Oh, Heung-Seon and Yuchul Jung (n.d.). "Enhancing the Narrow-down Approach to Large-scale Hierarchical Text Classification with Category Path Information". In:

— (2014). "External methods to address limitations of using global information on the narrow-down approach for hierarchical text classification". In: *Journal of Information Science* 40.5, pp. 688–708.

Oh, Heung-Seon and Sung-Hyon Myaeng (2014). "Utilizing global and path information with language modelling for hierarchical text classification". In: *Journal of Information Science* 40.2, pp. 127–145.

Otero, Fernando EB, Alex A Freitas, and Colin G Johnson (2009). "A hierarchical classification ant colony algorithm for predicting gene ontology terms". In: *European*

Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Springer, pp. 68–79.

Parikh, Ankur P et al. (2016). "A decomposable attention model for natural language inference". In: *arXiv preprint arXiv:1606.01933*.

Partalas, Ioannis et al. (2015). "LSHTC: A benchmark for large-scale text classification". In: *arXiv preprint arXiv:1503.08581*.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). "On the difficulty of training recurrent neural networks". In: *International Conference on Machine Learning*, pp. 1310–1318.

Polyak, Boris T (1964). "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5, pp. 1–17.

Quinn, Michael J and Mary L Laier (2006). *Method and apparatus for fast lookup of related classification entities in a tree-ordered classification hierarchy*. US Patent 7,032,072.

Reuters, Thomson (2012). "Web of Science." In:

Salakhutdinov, Ruslan, Joshua B Tenenbaum, and Antonio Torralba (2013). "Learning with hierarchical-deep models". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8, pp. 1958–1971.

Salton, Gerard and Christopher Buckley (1988). "Term-weighting approaches in automatic text retrieval". In: *Information processing & management* 24.5, pp. 513–523.

Sasaki, Minoru and Kenji Kita (1998). "Rule-based text categorization using hierarchical categories". In: *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*. Vol. 3. IEEE, pp. 2827–2830.

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

Silla Jr, Carlos N and Alex A Freitas (2009). "A global-model naive bayes approach to the hierarchical prediction of protein functions". In: *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, pp. 992–997.

Silla, Carlos N and Alex A Freitas (2011). "A survey of hierarchical classification across different application domains". In: *Data Mining and Knowledge Discovery* 22.1-2, pp. 31–72.

Smith, L N (2017). "Cyclical Learning Rates for Training Neural Networks". In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472.

Socher, Richard et al. (2013). "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642.

Sordoni, Alessandro et al. (2015). "A neural network approach to context-sensitive generation of conversational responses". In: *arXiv preprint arXiv:1506.06714*.

Srivastava, Nitish et al. (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Sun, Aixin and Ee-Peng Lim (2001). "Hierarchical text classification and evaluation". In: *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE, pp. 521–528.

Sutskever, Ilya, James Martens, and Geoffrey E Hinton (2011). "Generating text with recurrent neural networks". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024.

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

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

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

Tai, Kai Sheng, Richard Socher, and Christopher D Manning (2015). "Improved semantic representations from tree-structured long short-term memory networks". In: *arXiv preprint arXiv:1503.00075*.

Tang, Duyu, Bing Qin, and Ting Liu (2015). "Document modeling with gated recurrent neural network for sentiment classification". In: *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1422–1432.

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

Turney, Peter D and Patrick Pantel (2010). "From frequency to meaning: Vector space models of semantics". In: *Journal of artificial intelligence research* 37, pp. 141–188.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems*, pp. 6000–6010.

Vens, Celine et al. (2008). "Decision trees for hierarchical multi-label classification". In: *Machine Learning* 73.2, p. 185.

Visin, Francesco et al. (2015). "Renet: A recurrent neural network based alternative to convolutional networks". In: *arXiv preprint arXiv:1505.00393*.

Wang, Ke, Senqiang Zhou, and Yu He (2001). "Hierarchical classification of real life documents". In: *Proceedings of the 2001 SIAM International Conference on Data Mining*. SIAM, pp. 1–16.

Wang, Mengqiu and Christopher D Manning (2013). "Effect of non-linear deep architecture in sequence labeling". In: *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pp. 1285–1291.

Williams, Ronald J (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Reinforcement Learning*. Ed. by Richard S Sutton. Boston, MA: Springer US, pp. 5–32.

Wu, Feihong, Jun Zhang, and Vasant Honavar (2005). "Learning classifiers using hierarchically structured class taxonomies". In: *International Symposium on Abstraction, Reformulation, and Approximation*. Springer, pp. 313–320.

Xu, Kun et al. (2018). "Graph2Seq: Graph to Sequence Learning with Attention-based Neural Networks". In:

Xue, Gui-Rong et al. (2008a). "Deep classification in large-scale text hierarchies". In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 619–626.

— (2008b). "Deep classification in large-scale text hierarchies". In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 619–626.

Yang, Zichao et al. (2016). "Hierarchical attention networks for document classification". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1480–1489.

Yin, Wenpeng and Hinrich Schütze (2015). "Convolutional neural network for paraphrase identification". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 901–911.

Yogatama, Dani et al. (2017). "Generative and discriminative text classification with recurrent neural networks". In: *arXiv preprint arXiv:1703.01898*.

Zhang, Xiang and Yann LeCun (2015). "Text understanding from scratch". In: *arXiv preprint arXiv:1502.01710*.

Zhang, Xiang, Junbo Zhao, and Yann LeCun (2015). "Character-level convolutional networks for text classification". In: *Advances in neural information processing systems*, pp. 649–657.

Zhou, Chunting et al. (2015). "A C-LSTM neural network for text classification". In: *arXiv preprint arXiv:1511.08630*.