Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

A ROBUST DEEP CONVOLUTIONAL NEURAL NETWORK MODEL FOR TEXT CATEGORIZATION

Edgard de Freitas Júnior

VOLUME I

Belo Horizonte
Março de 2016

A ROBUST DEEP CONVOLUTIONAL NEURAL NETWORK MODEL FOR TEXT CATEGORIZATION

EDGARD DE FREITAS JÚNIOR

A ROBUST DEEP CONVOLUTIONAL NEURAL NETWORK MODEL FOR TEXT CATEGORIZATION

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

ADVISOR: ADRIANO ALONSO VELOSO

Belo Horizonte

Março de 2016

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Freitas Júnior, Edgard de.

F866r A robust deep convolutional neural network model for text categorization. / Edgard de Freitas Júnior. Belo Horizonte, 2016.
xxv, 157f.: il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Adriano Alonso Veloso

1. Computação – Teses. 2. Aprendizado do computador – Teses. 3. Processamento da linguagem natural (Computação) – Teses. I. Orientador. II. Título.

CDU 519.6*82 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

A robust deep convolutional neural network model for text categorization

EDGARD DE FREITAS JUNIOR

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]

PROF. ADRIANO ALONSO VELOSO - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]

PROF. MARCO ANTÔNIO PINHEIRO DE CRISTO
Departamento de Ciência da Computação - UFAM

[illegible]

PROF. NIVIO ZIVIANI
Departamento de Ciência da Computação - UFMG

[illegible]

PROF. RENATO ANTÔNIO CELSO FERREIRA
Departamento de Ciência da Computação - UFMG

[illegible]

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 30 de março de 2016.

Acknowledgments

I would like to thank my sister Anna Lee for her support and exemplary life trajectory that inspired my decision on taking this challenge.

This work would not have been possible without the patience and support of my girlfriend Denise and her family.

I would also like to thank my friends Andréia Martinez, Salomão Fraga and Paulo Fernando for their encouragement and support that kept me going along the way.

I would like to express my gratitude to Prof. Alfredo Loureiro and Prof. Marcos André Gonçalves for their recommendation letters and confidence.

I specially thank my longtime friend Prof. Wagner Meira for his support and advices.

My sincere thanks to Prof. Renato Ferreira and NVIDIA Corporation for the donation of the K40 GPU, without which would not been possible to run all the experiments.

Thanks to Prof. Marco Cristo for the opportunity to participate on a deep learning course taught by LISA researches at UFAM.

I wish to specially thank my advisor, Prof. Adriano Veloso, for his patience, optimism and availability to participate in long discussions about deep learning and philosophy matters.

I would also like to thank all the students from the machine learning group at UFMG for the interesting discussions and research they have been done.

“Colorless green ideas sleep furiously.”
(Avram Noam Chomsky)

Resumo

Categorização de textos é uma das tarefas mais importantes nas aplicações do domínio do Processamento de Linguagem Natural (PLN), a qual consiste em associar automaticamente categorias pré-definidas a documentos escritos em linguagem natural. Técnicas tradicionais de aprendizado de máquina utilizam características elaboradas manualmente para a construção dos modelos, tais como, n-gramas, palavras de negação, sinais de pontuação, símbolos representando emoções, palavras alongadas e dicionários léxicos. Esta abordagem, chamada de engenharia de características, além de requerer um trabalho árduo, resulta geralmente em modelos que apresentam uma performance ruim em tarefas para as quais não foram especificamente criados.

Neste trabalho, propomos um modelo robusto baseado em uma Rede Neural de Convolução (RNC) profunda para aprendizado chamado de PLN profundo. Nosso modelo utiliza uma abordagem composicional, na qual o projeto da arquitetura da RNC profunda induz a criação de uma representação hierárquica para o texto através da descoberta de representações intermediárias para as palavras e sentenças do texto. As representações iniciais para as palavras, chamadas de incorporação de palavras, são obtidas de um modelo de linguagem neural treinado previamente de forma não supervisionada, as quais são ajustadas para o contexto da tarefa para a qual o modelo está sendo treinado.

O nosso modelo foi avaliado em tarefas de categorização de textos comparando sua acurácia com os resultados publicados para alguns modelos tradicionais e de aprendizado profundo utilizando seis conjuntos de dados de larga escala. Os resultados mostram que nosso modelo é robusto no sentido de que, mesmo quando nós utilizamos os mesmos parâmetros globais, ele supera a acurácia dos modelos considerados estados da arte em diferentes tarefas de categorização de textos. Os resultados também mostram que a utilização de um dicionário de sinônimos semânticos juntamente com as representações iniciais de palavras ajuda na generalização das representações aprendidas pelo modelo, aumentando sua acurácia.

Palavras-chave: Aprendizado Profundo, PLN, RNC, Categorização de Textos.

Abstract

Text categorization is the task of automatically assigning pre-defined categories to documents written in natural languages and it is one of the most important tasks in Natural Language Processing (NLP) domain applications. Traditional machine learning techniques rely on handcrafted features such as ngrams, negation words, punctuation, emoticons, stop words, elongated words and lexicons to build their models. This approach, called feature engineering, in addition to being labor intensive, results in models that, in general, present poor performance on tasks for what they have not been specifically tailored.

In this work, we propose a robust deep learning Convolutional Neural Network (CNN) model named Deep NLP. Our model adopts a compositional approach, in which the design of the deep CNN architecture induces the creation of a hierarchical representation for the text, through the extraction of intermediate representations for the words and sentences of the text. The initial word representations, called word embeddings, are obtained from a pre-trained unsupervised neural language model and they are adjusted for the context of the task that the model is being trained.

We evaluated our model comparing its accuracy against the results reported by some traditional and deep learning models in text categorization tasks using six large-scale data sets. The results show that our model is robust in the sense that, even when we use the same hyperparameters, it surpasses the accuracy of the state-of-the-art models in different text categorization tasks. The results also show that the use of a semantic synonyms dictionary together with the word embeddings helps to generalize the representations learned by the model increasing its accuracy.

Keywords: Deep Learning, NLP, CNN, Text Categorization.

List of Figures

List of Tables

List of Acronyms

1D	One-dimensional
2D	Two-dimensional
ANN	Artificial Neural Network
BoW	Bag of Words
CBOW	Continuous Bag-of-Words
CNN	Convolutional Neural Network
DBN	Deep Belief Network
DCNN	Dynamic Convolutional Neural Network
GPU	Graphical Processing Unit
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
LSTM	Long Short Term Memory
NLL	Negative Log-Likelihood
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NN	Neural Network
POS	Poverty of the Stimulus
RecNN	Recursive Neural Network
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
TDNN	Time-Delay Neural Network

Contents

Chapter 1

Introduction

Text categorization is one of the most important tasks in Natural Language Processing (NLP). Text categorization is the task of automatically assigning pre-defined categories to documents written in natural languages. Text categorization can be used for, among others, classifying a document in a set of topics, rating a product review written by a costumer or associating a sentiment with a text posted by a user [Manning & Schütze, 1999] [de Oliveira Jr., et al., 2014] [Veloso, Jr., Cristo, Gonçalves, & Zaki, 2006].

Traditional machine learning techniques used to build models for text categorization rely on handcrafted features to succeed. Features such as ngrams, negation words, stop words, punctuation, emoticons, elongated words and lexicons are carefully chosen by a domain specialist for a specific task. This approach leads to models tailored for a specific context and seldom achieve good performance in different tasks. This approach is called feature engineering [Bottou, et al., 2011].

1.1 Deep Learning Models

The deep learning paradigm adopts a different approach to find the model features. The models built using the deep learning approach learn not only the parameters of the features but also the features themselves for a given task. This approach, called feature learning, leads to more general models that achieve good performance in different tasks and domains . In this scenario, the model specialist has to fine-tune the model hyperparameters for the given task [Goodfellow, Bengio, & Courville, 2016].

Despite the good performance achieved by some traditional machine learning NLP techniques as Bag of Words (BoW) in text categorization tasks such as topic identification, these models present poor performance in tasks where the semantic of the text is sensitive to the word positions. For example, a BoW model will give the same value for the expressions “know a little bit about everything” and “know everything about a little bit”. Both expressions have exactly the same words, but have different meanings. The first one designates a generalist and the second one designates a specialist. To be able to capture the semantic of a sentence, a model has to take into account the word positions in the sentence.

There are different deep learning models suited for specific application domains. The Recursive Neural Network (RecNN) model is claimed to be well suited for NLP applications because of its hierarchical structure. The weakness of this type of model is its dependency on an external syntactic parse tree. This restriction limits the learning of semantic relations between words to syntactically dictated phrases. Extended models based on RecNN achieved the state of the art in some NLP tasks using specific data sets [Socher, et al., 2013].

The Recurrent Neural Network (RNN) model is a special case of RecNN that is suited for modeling sequential data. Despite its power in representing sequential structures, it is seldom being used for NLP tasks such as text categorization because of its difficulty in learning long-term dependencies. This limitation is due to the exploding and vanishing gradients problems that occur in the training phase [Pascanu, Mikolov, & Bengio, 2013].

To overcome the exploding and vanishing gradients problems, a type of RNN architecture called Long Short Term Memory (LSTM) has been used with success in some application domains [Zaremba, Sutskever, & Vinyals, 2014].

1.2 Convolutional Neural Network Models

Two-dimensional (2D) Convolutional Neural Network (CNN) models have been successfully applied in computer vision domain problems for some time [LeCun, Bottou, Bengio, & Haffner, 1998]. More recently, the remarkable results achieved by deep CNN on image classification challenges got the state of the art to a new level and promoted the renascence of the deep learning paradigm [Krizhevsky, Sutskever, & Hinton, 2012].

It has been showed that deep CNN models are able not only to discover the features of the data, but also they are able to learn a hierarchical representation for the data through the discovered features. The revealed features have desirable properties such as compositionality, increasing invariance and class discrimination as they ascend the network layers [Zeiler & Fergus, 2013].

One-dimensional (1D) CNN models like Time-Delay Neural Networks (TDNN) have been successfully used for some time in speech recognition applications such as phoneme recognition [Waibel, Hanazawa, Hinton, Shikano, & Lang, 1989]. More recently, deep 1D-CNN models have been used in NLP tasks like language modeling [Bottou, et al., 2011].

1.3 Data Representation

A central problem present in all NLP applications is how to represent the input text. Some models view the input text as a stream of characters [Zhang, Zhao, & LeCun, 2015]. Other models deal with the input text as a sequence of phonemes [dos Santos & Gatti, 2014]. Most models make the natural choice of viewing the input text as a sequence of words. In these models, the problem is how to represent a word. The simplest way is to associate with each vocable present in the text a unique id and use a lookup table to encode the text's words. Each id is represented as a vector that has the size of the vocabulary and only the bit that identifies the vocable is set to one. That is why this type of word representation is called one-hot.

The problem with the one-hot representation is the dimensionality of the word vectors. For example, in a dataset with 30K vocables, each word in the text will be represented by a vector of size 30K. Despite of this limitation, some models using one-hot representation have achieved remarkable results in text categorization tasks [Johnson & Zhang, 2015].

A more sophisticated way of word representation is called word embedding. This type of representation tries to create a mapping from the symbolic representation of a word into a lower dimensional vector space. In addition to effectively dealing with the problem of the dimensionality, it has been shown that word embeddings are able to capture many semantic relationships between the words they represent [Mikolov, Chen, Corrado, & Dean, 2013].

Intuitively, in the context of deep learning models, the use of word embeddings makes sense. Like humans do, the model learns a semantic representation of a word in some context and it adjusts this representation for the specific context that it is being trained. The models that make use of word embeddings are considered semi-supervised models because the initial word representations are learned in an unsupervised way and they fit these representations for a specific task through a supervised training.

1.4 Compositionality

Another central problem present in all NLP tasks is how much syntax is needed to extract semantics. As we have already mentioned, the RecNN models depend on an external syntactic parse tree to extract semantic from a text. The performance of these models degrades on NLP tasks where the input text is written using an informal language style that does not strictly follow syntactic rules.

Some models try to learn a representation in an unsupervised way not only for the words but also for the whole paragraph. These models are suited for NLP tasks where there is not data sets with enough labeled data. [Le & Mikolov, 2014]

Most of the deep CNN models used in NLP tasks convolves a set of filters over the sequence of text words. They do not explicitly take into account the syntactic information of text sentence units. They consider the whole text as a syntactic unit. These models are suited for NLP tasks where the input text holds in one sentence. An example of this type of task is sentiment classification of texts from a Twitter dataset [Kalchbrenner, Grefenstette, & Blunsom, 2014].

In some NLP tasks, it seems to make sense to apply a compositional approach. These models explicitly consider a text made up by sentence units that, in turn, are compounded by words [Denil, Demiraj, Kalchbrenner, Blunsom, & de Freitas, 2014].

Analogously to an image made up by different classes of objects in the computer vision domain, a text is made up by sentence units that can have different semantics. It was showed, in the computer vision domain, that forcing information to pass through carefully chosen bottlenecks makes it possible to control the types of intermediate representations learned by the model [Hinton, Krizhevsky, & Wang, 2011]. This strategy helps on the generalization of the representations learned by the model [Gülçehre & Bengio, 2013].

1.5 Depth

Another central question in the design of deep learning models is how deep a network must be. There is a consensus that shallow models are not able to extract complex features of the data but there is not a rule of thumb to determine how many layers suffice to extract the required features for a specific task. In the computer vison domain, a deep CNN model that achieved the state of the art in image classification tasks was implemented using 22 layers [Szegedy, et al., 2014].

Another critical issue on the design of deep CNN models is the relationship between the number of parameters and the depth of the model. In the specific case of NLP models, the dimensionality of the data representation has a huge impact on the number of parameters of the model.

1.6 Objectives of this Work

The objective of this work is to propose a robust deep CNN model for text categorization tasks, named Deep NLP. The design of this model aims to overcome the limitations of other models reported in the literature and to achieve a robustness in the sense that the model can be used in different text categorization tasks without the need of tuning the model hyperparameters. To achieve these objectives, we employ a series of deep learning concepts and techniques. We adopt a semi-supervised approach where the initial vocable representations are obtained from a pre-trained unsupervised neural language model publicly available[1] (Word2Vec). The vocable representations are adjusted to a specific task context during the training phase. The model implements a compositional approach explicitly creating intermediate representations for the sentences. The size of the input text is not limited by the model. The model is made up of seven layers to extract complex features of

[1] https://code.google.com/archive/p/word2vec/

the input text. We make use of the WordNet corpus to find semantic synonyms for the vocables not found in Word2Vec and for the text words not found in the generated vocabulary [Miller, 1995].

1.7 Contributions of this Work

We evaluated our model comparing its accuracy against the results reported by deep learning models in text categorization tasks [Zhang, Zhao, & LeCun, 2015]. The model was trained without and with the use of the WordNet synonyms. We also made experiments to measure how the data set size affects the accuracy and training time of our model.

The results show that our model is robust in the sense that, even when using the same model hyperparameters, it can beat the state of the art models' accuracy in different text categorization tasks. The results also show that the use of the WordNet semantic synonyms helps to generalize the representation learned by the model, thus increasing its accuracy. The experiments made with the data set size show that our model beat the accuracy of the state of the art model using only one third of the data set size.

Another contribution resultant from the design of the proposed architecture is that our model do not impose any limit on the size of the input text. The implementation of our model makes an efficient use of the massively parallel processing power of the GPU, which makes it possible to train huge data sets in a shorter processing time.

1.8 Organization

The remaining part of this work is organized as follows. In Chapter 2, we introduce some underlying concepts used in artificial neural networks and deep learning models. In Chapter 3, we present the related work that apply or develop similar concepts used in this dissertation. In Chapter 4, we provide an in-depth description of the architecture of our model. In Chapter 5, we discuss the implementation details of our model. In Chapter 6, we describe the data sets and experiments used to evaluate our model. In Chapter 7, we report and analyze the results of the experiments. In Chapter 8, we discuss the main contributions of this work. In Chapter 9, we address some future work.

Chapter 2

Background

In this chapter, we present the underlying concepts necessaries for the understanding of this work. We introduce some Artificial Neural Networks basic concepts, then we present some principles discovered by the neural science that inspired the development of the neurocomputing algorithms. Finally, we present an overview of the deep learning paradigm.

2.1 Neural Networks

The basic concepts used in the deep learning paradigm are inherited from the Artificial Neural Networks (ANNs) models or Neural Networks (NNs) for short. The aim of the NNs paradigm is to develop computer programs capable of solving abstract problems that are hard to be described using formal rules, but are easily solved by human beings.

The development of the NNs paradigm started in the 1950s [McCulloch & Pitts, 1943] [Rosenblatt, 1962]. The NNs models were inspired by the concepts and principles of the way the human brain works, which was discovered by the neural science.

The human cortex can be viewed as a complex network whose nodes are neurons. Each neuron receives input signals from other neurons through its dendrites. The neurons connections are established through the synapses. The strength of the input signals is determined by the stimulus received. The input signals are combined inside the neuron to create an output signal. The output signal is transmitted to other neurons through the axon if its amplitude is greater than a pre-determined value called action potential. The output signal is called a spike. It is estimated that the human cortex has 10 billion of neurons and 60 trillion of synapses [Kandel, Schwartz, & Jessel, 2000].

At the cell level, the human behavior adaptability or learning mechanism can be explained by the plasticity hypothesis. The stimulus received from the environment and the output produced by the network cause permanent changes on the neurons connections. Figure 2.1 shows a node of a feedforward NN model projected over a schematic view of a typical neuron.

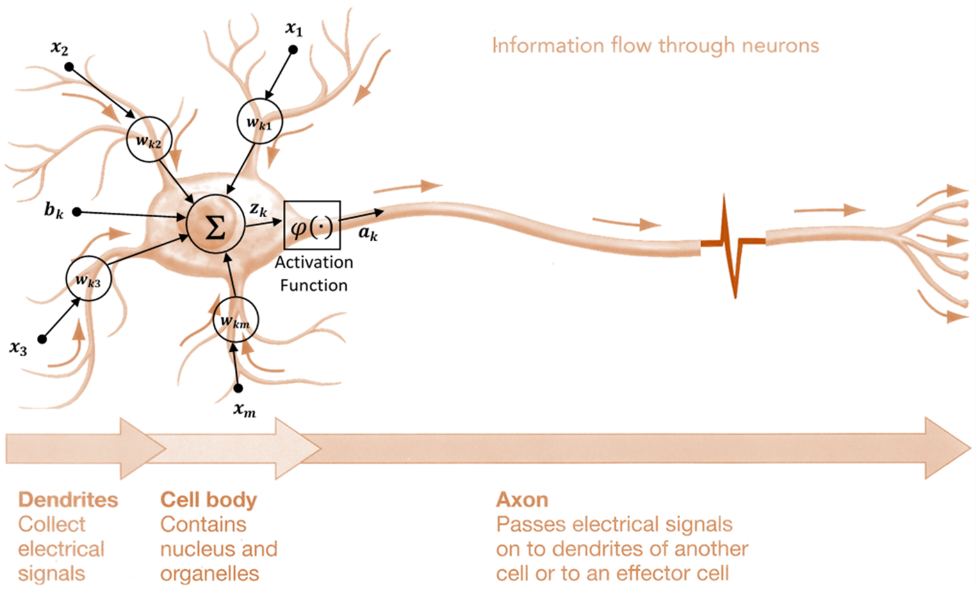
Information flow through neurons
x_1
x_2
x_3
x_m
w_{k1}
w_{k2}
w_{k3}
w_{km}
b_k
Σ
z_k
$\varphi(\cdot)$
a_k
Activation Function
Dendrites
Collect electrical signals
Cell body
Contains nucleus and organelles
Axon
Passes electrical signals on to dendrites of another cell or to an effector cell

Figure 2.1. Neuron model projected over a typical neuron cell.

In Haykin [1999], the author defines an ANN as a massively parallel distributed system made up of simple processing units, which has a natural propensity for storing experimental knowledge. An ANN resembles the human brain in that the knowledge is acquired by the network from its environment through a learning process and the strength of neuron connections, known as synaptic weights, are used to store the acquired knowledge.

In the context of ANNs, learning is the process by which the synaptic weights, or network parameters, are adjusted through a process of stimulation known as training. The type of learning is determined by the way the parameters changes take place. There are many types of learning mechanisms. In ANNs, the most used learning mechanism is the error-correction algorithm.

The error-correction learning algorithm compares the network output with a target value through an objective or cost function. The cost function associates the network parameters with a measure of the error produced by the network output. In feedforward NNs, the most used error-correction learning algorithm is the backpropagation.

Backpropagation is about understanding how adjusting the weights and biases in a network changes the error given by the cost function. Because the cost function depends on the network output value, which in turn is a function of the output layer activation function that depends on the previous layers weights and bias and so on, we can recursively use the chain rule to calculate the gradient of the cost function with respect to the network parameters. This way, we know how the changes on each network parameter contribute to the error measured by the cost function.

The backpropagation algorithm is executed in four steps. In the feedforward step, the network output is calculated. In the error step, the cost function gradient of with respect to the network output is calculated. In the backward step, the error is back propagated calculating the gradient with respect to the previous layers outputs. In the update step, the values of the network parameters are adjusted using some updating rule. In general, the updating rule used is the gradient descent algorithm. The network parameters are subtracted from its gradient multiplied by a constant. This constant is called the learning rate.

The backpropagation algorithm was originally introduced in the 1970s, but it became popular only in 1986 after the publication of a paper in which the authors showed that the speedup aroused from the use of the backpropagation algorithm made it possible to use NNs to solve problems that had previously been insoluble [Rumelhart, Hinton, & Wilson, 1986].

What is clever about the backpropagation algorithm is that it enables us to compute all the gradients partial derivatives simultaneously using just one forward pass through the network, followed by one backward pass. Roughly speaking, the computational cost of the backward pass is about the same as the forward pass.

Even in the late 1980s, people ran up against computational limits, especially when attempting to use backpropagation to train deep NNs. The backpropagation algorithm is based on common linear algebraic operations like vector additions and matrix multiplications. In 2006, the improvement of the algorithms and the popularization of the use of the GPUs for scientific computation made the use of the backpropagation algorithm feasible in deep NNs models [Hinton, Osindero, & Teh, 2006] [Kirk & Hwu, 2010].

Figure 2.2 shows the steps of the backpropagation algorithm in a small segment of a typical feedforward neural network.

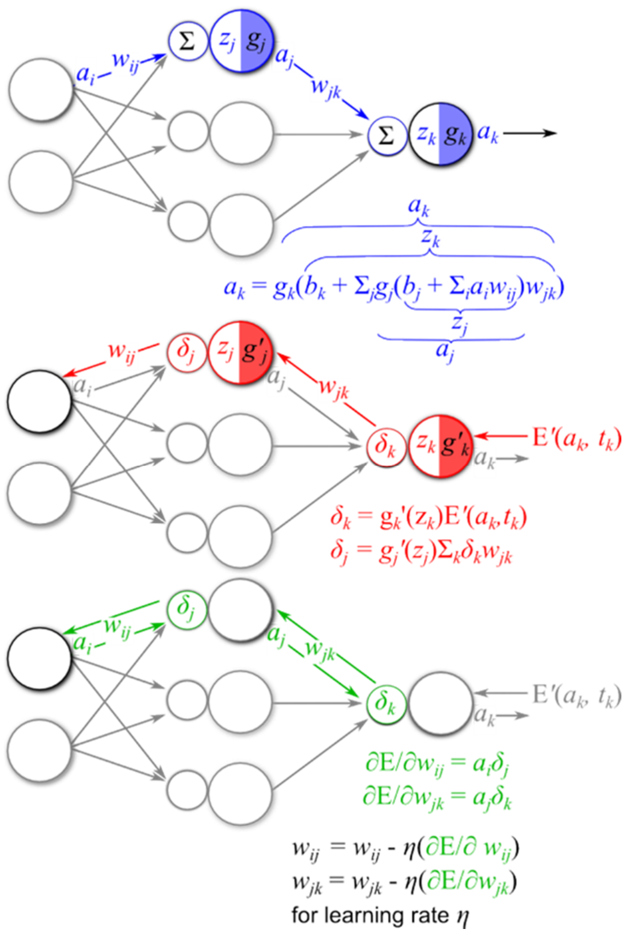
a_i
w_{ij}
Σ
z_j
g_j
a_j
w_{jk}
z_k
g_k
a_k
$a_k = g_k(b_k + \Sigma_j g_j(b_j + \Sigma_i a_i w_{ij}) w_{jk})$
δ_j
g'_j
δ_k
g'_k
$E'(a_k, t_k)$
$\delta_k = g_k'(z_k)E'(a_k,t_k)$
$\delta_j = g_j'(z_j)\Sigma_k \delta_k w_{jk}$
$\partial E/\partial w_{ij} = a_i\delta_j$
$\partial E/\partial w_{jk} = a_j\delta_k$
$w_{ij} = w_{ij} - \eta(\partial E/\partial w_{ij})$
$w_{jk} = w_{jk} - \eta(\partial E/\partial w_{jk})$
for learning rate η

Figure 2.2. Steps of the backpropagation algorithm.

2.2 Neocortex Deep Structure

Another concept used in ANNs inspired by the human cortex structure is the concept of hierarchy. Humans organize their ideas and concepts hierarchically first learning simpler concepts and then composing them to represent abstract concepts. It is believed that this behavior is due to the physical structure of the human neocortex.

The human neocortex is organized into regions and the typical neocortex tissue is made up by six layers of neurons cells. The lower layers, sixth and fifth, have a higher concentration of neurons than the upper layers. They receive input signals from other cortex regions and pass the extracted features to the upper layers, which in turn pass the information to other neocortex regions.

Within the neocortex, the information flows serially from one region to another. For example, the visual cortex is built by a sequence of regions, each of which contains a representation of the input and the signals flow from one region to the next. Each level of this feature hierarchy represents the input at a different abstraction level, with more abstract features further up in the hierarchy, defined in terms of the lower-level ones [Kandel, Schwartz, & Jessel, 2000].

The upper layers and regions also have feedback connections to the lower ones. For many years, most scientists ignored these feedback connections. They are essential for the brain to accomplish one of its most important functions, which is to make predictions. Predictions requires a comparison between what is happening and what you expect to happen. What is actually happening flows up in the hierarchy, and what you expect to happen flows down [Hawkins & Blakeslee, 2004].

Figure 2.3 shows on the left a histological structure of the human neocortex tissue and on the right a schematic representation of some sensory regions layers hierarchy. The appearance of the histological structure depends on what was used to stain it. The Golgi stain reveals the neuronal cell bodies and the dendritic trees. The Nissl method shows the cell bodies and the proximal dendrites. The Weigert stain for myelinated fibers reveals the pattern of axonal distribution [Kandel, Schwartz, & Jessel, 2000].

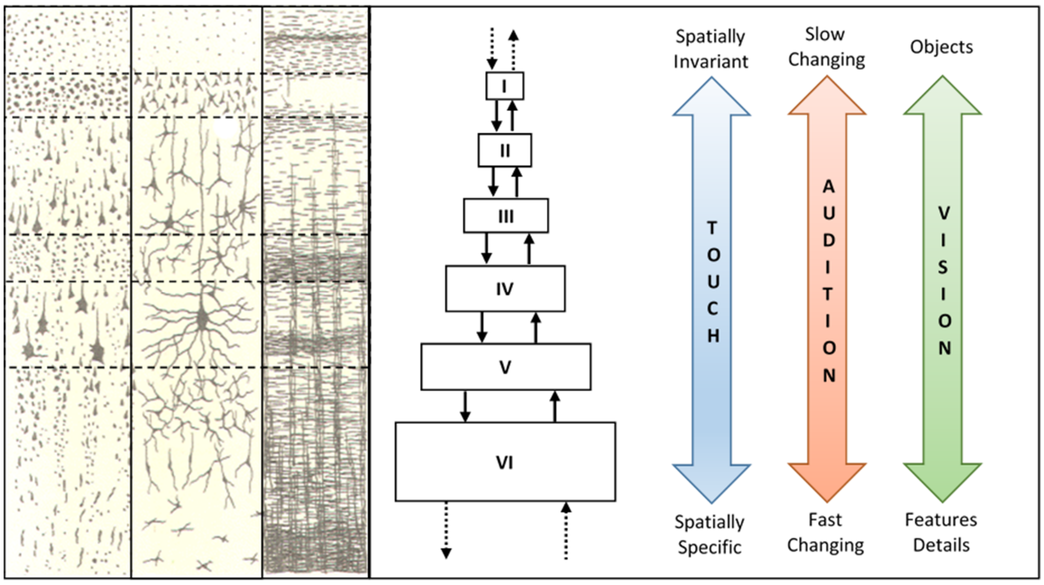

Figure 2.3. The six layers of the human neocortex.

2.3 Deep Learning

2.3.1 Depth Matters

The deep learning paradigm can be characterized by the use of two strategies inspired by the working of the human brain. The first strategy is the learning from experience, which was already adopted in the ANNs. The second strategy is to understand the world in terms of a deep hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts.

The approach of gathering knowledge from experience avoids the need to specify the formal rules that allow the computer programs to solve abstract problems. The approach of viewing an abstract problem as a hierarchy of concepts allows the computer programs to learn complicated concepts by building them out of simpler ones.

The building of a hierarchy of concepts is induced by the deep architecture of layers. The use of a deep architecture can be viewed as a kind of function factorization. The depth of two layers may be enough to represent some families of functions with a given target accuracy. Theoretical results showed that there are families of functions for which the insufficient depth makes the number of parameters grows exponentially with the input size [Bengio, 2009]. The Kolmogorov's Mapping Neural Network Existence theorem assures that an arbitrary continuous function, mapping values from an n-dimensional compact set to the real numbers vector space, can be implemented by a feedforward neural network with at least three layers of depth [Hecht-Nielsen, 1990].

Figure 2.4 illustrates a classification problem of a two-class data set represented by two curves. Each layer of the network transforms the data, creating a new representation and making the data easily separable by a linear classifier.

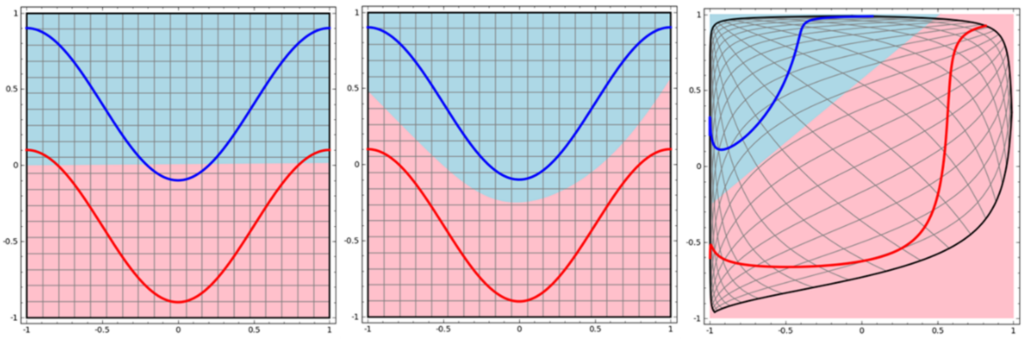

Figure 2.4. Representations learned by layers make data separable.

Deeper models tend to perform better not only because they are larger. Increasing the number of parameters in models having less than three layers, called shallow models, does not allow them to reach the same level of performance as deeper models. This is primarily due to overfitting. Figure 2.5 presents a chart with the results of an experiment comparing the number of parameters with the performance of models having different depths [Goodfellow, Bulatov, Ibarz, Arnoud, & Shet, 2014].

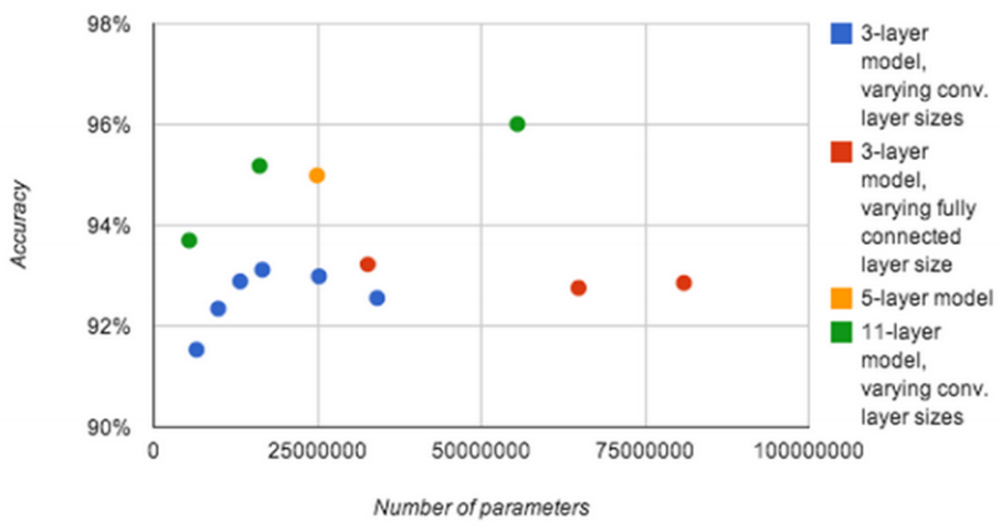

Figure 2.5. Effect of the number of parameters on the performance of models with different depths.

It is clear that only the deepest models had their accuracy increased with the growth on the number of parameters.

2.3.2 The Renascence

Until 2006, attempts of training a deep supervised feedforward neural network architecture yielded worse results then shallow architectures. In Bengio et al. [2006], the authors extended the pionner work done in Hinton et al. [2006,] showing that the initialization of Deep Belief Networks (DBN) parameters with pre-trained unsupervised learned representations values could improve its generalization. Since then, the development of new algorithms and techniques made possible the implementation of deeper architecutes and the adoption of the deep learning paradigm to solve problems in many domains [Bengio, Learning Deep Architectures for AI., 2009].

In 2012, a dramatic moment in the meteoric rise of deep learning came when a deep CNN architecture won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) for the first time and by a wide margin, bringing down the state-of-the-art error rate from 26.1% to 15.3% [Krizhevsky, Sutskever, & Hinton, 2012]. Since then, these competitions are consistently won by deep CNNs and the advances in deep learning have brought the latest top-5 error rate in this contest down to3.6% [Goodfellow, Bengio, & Courville, 2016].

Two main facts, besides the development of new algorithms and techniques, contributed to the recent success of the deep learning paradigm. The increase on the massively parallel processing power of the GPUs for scientific computation made it possible to implement deeper models having a huge number of parameters.

Figure 2.6 shows comparative charts between the processing power of the GPUs, on the left, and the number of neurons of ANNs implemented over time on the right [Goodfellow, Bengio, & Courville, 2016].

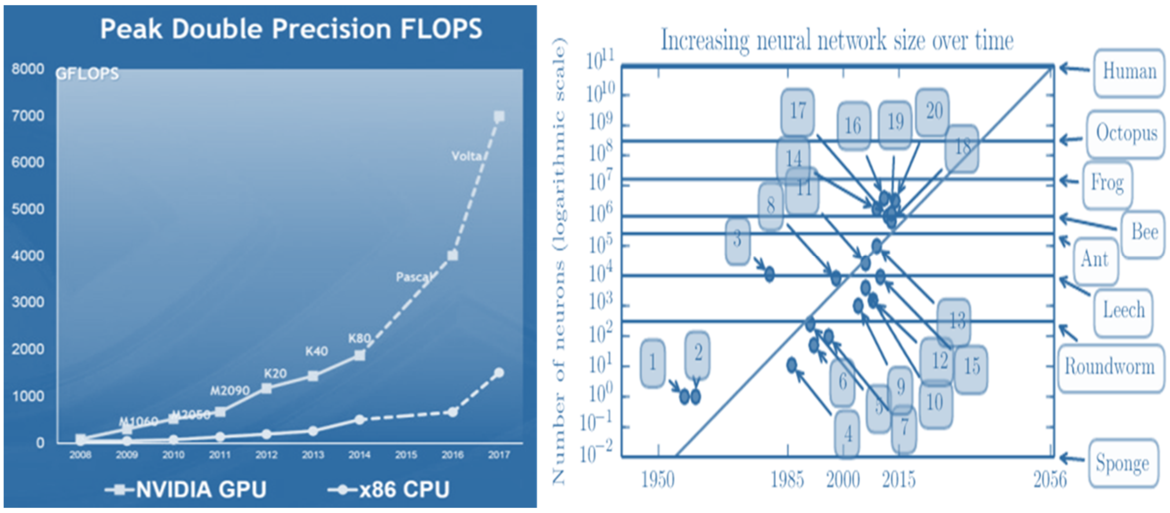

Figure 2.6. Evolution of GPUs and NNs over time.

The other fact that contributed to the recent success of the deep learning paradigm is the increase on the data sets size. In the 1980s and 1990s, machine learning became statistical in nature and began to leverage larger data sets containing tens of thousands of examples such as the MNIST data set. As the models become more complex, the number of parameters increases and more data is required to train the model.

Figure 2.7 shows a chart of the data sets size over time [Goodfellow, Bengio, & Courville, 2016].

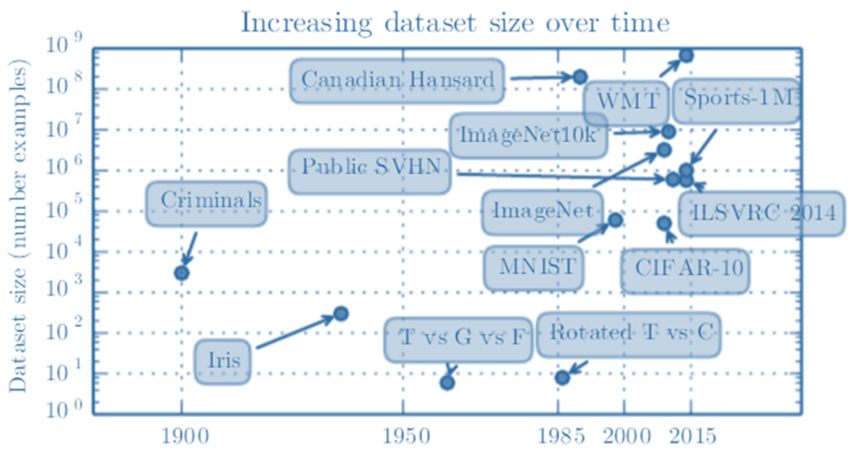

Figure 2.7. Evolution of data sets size over time.

2.3.3 CNN Architecture

There are many types of ANN architectures. Each architecture has been developed for a specific task. The Convolutional Neural Network (CNN) architecture was developed for computer vision tasks and it was inspired by the discoveries of the neurophysiologists about how the mammalian vision system works [Hubel & Wiesel, 1959]. They observed how neurons in the cat's brain responded to images projected in precise locations on a screen in front of the cat. Their great discovery was that neurons in the early visual system responded most strongly to very specific patterns of light, such as precisely oriented bars, but responded hardly at all to other patterns.

The visual cortex contains a complex arrangement of cells that are sensitive to small sub-regions of the visual field, called a receptive field. The sub-regions are tiled to cover the entire visual field. These cells act as local filters over the input space and are well suited to exploit the strong spatially local correlation present in natural images. Simple cells respond maximally to specific edge-like patterns within their receptive field. Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

The term convolutional comes from a mathematical operation called convolution. Convolution is a specialized kind of linear operation. The convolution operation used in ANNs does not correspond precisely to its definition in mathematics. Convolutional networks are simply ANNs that use convolution in place of general matrix multiplication in at least one of their layers [Goodfellow, Bengio, & Courville, 2016].

Figure 2.8 shows a schematic view of a 2D convolution operation as it is used in ANNs. The small letters correspond to the values of each position of the input and of the filter.

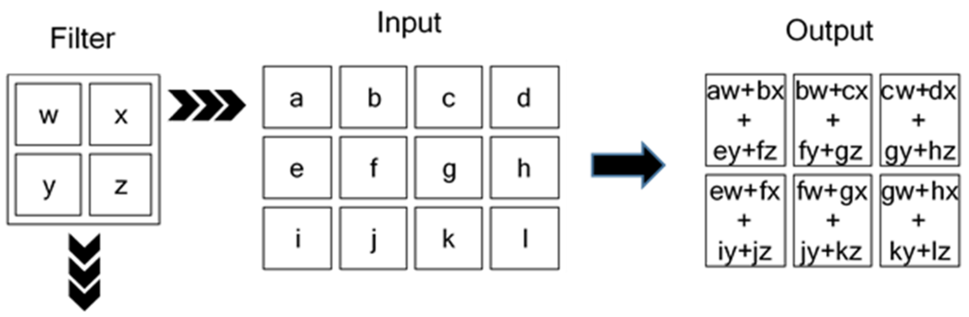

Figure 2.8. 2D convolution operation.

It has been showed that deep CNN models are able not only to discover the features of the data, but also they are able to learn a hierarchical representation for the data through the discovered features. The revealed features have desirable properties such as compositionality, increasing invariance and class discrimination as they ascend the network layers [Zeiler & Fergus, 2013]. Figure 2.9 shows the images generated by a visualization technique called deconvolution. The images reveal the patterns learned by each layer of a deep CNN. In the lower layers, the discovered patterns, like edges, correspond to small regions of the image. In the upper layers, the discovered patterns, like objects, correspond to larger regions of the image.

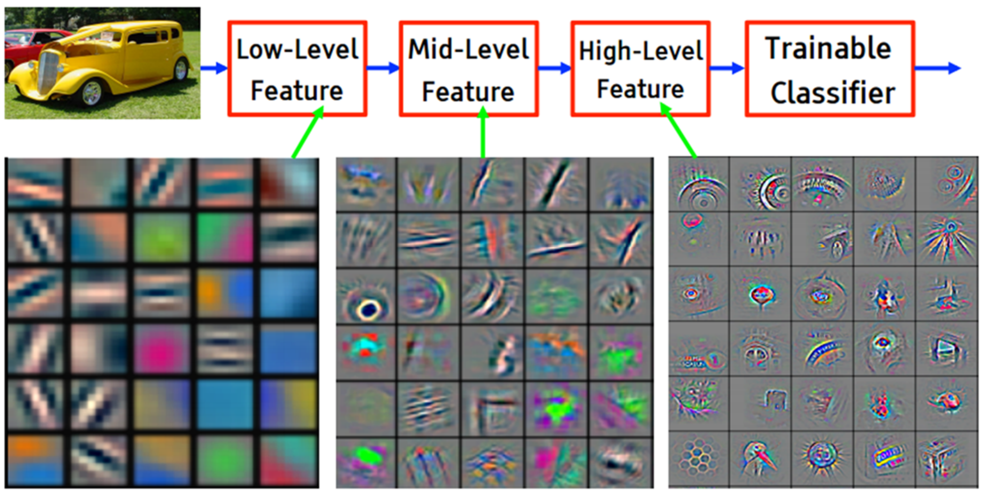

Figure 2.9. Patterns learned by the layers of a deep CNN.

Another key consideration about the architecture design of ANNs is the connection between the layers. Traditional ANN layers use a matrix multiplication to describe the interaction between each layer. This means that every element of a layer is connected to every element of the previous and next layers. CNNs have sparse connections. This is accomplished by making the filter smaller than the input. For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with filters that occupy only tens or hundreds of pixels. This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency. It also means that computing the output requires fewer operations. These improvements in efficiency are usually quite large.

Another strategy present in CNNs that helps reduce the memory requirements is the parameter sharing. Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional ANN, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. As a synonym for parameter sharing, one can say that a network has tied weights, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere. In a CNN, each element of the filter is used at every position of the input. The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, the model learns only one set. CNNs are thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency [Goodfellow, Bengio, & Courville, 2016]. Figure 2.10 shows a schematic view of the sparse connectivity and parameters sharing effects caused by a 1D-convolution operation.

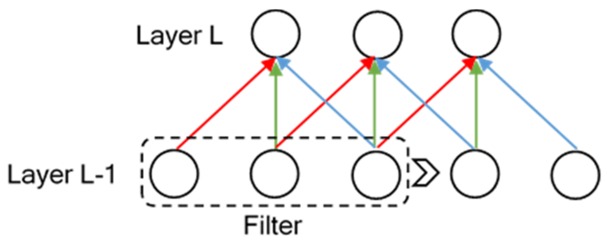

Figure 2.10. 1D convolution sparse connectivity and parameters sharing.

Figure 2.11 shows the three stages of a CNN typical layer. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the detector stage. In the third stage, we use a pooling function to modify the layer output further.

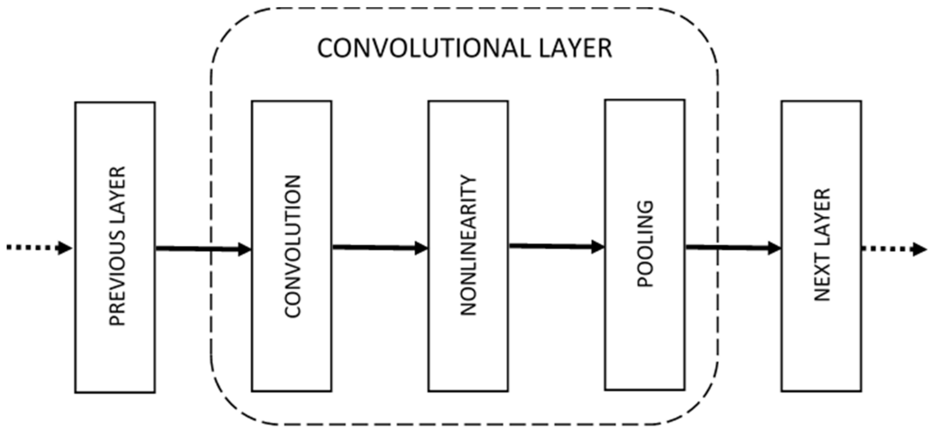

Figure 2.11. A typical CNN layer.

A pooling function replaces the layer output at a certain location with a summary statistic of the nearby outputs. For example, the max pooling operation reports the maximum output within a rectangular neighborhood. The pooling operation helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.

2.4 Word Embedding

In the NLP domain, when we decide to consider the words as the building blocks of a text, we have to find a way to represent these words. This choice is a trade-off between robustness and computational efficiency.

The most obvious choice is to use the one-hot representation. In this type of representation, each vocable of the text is represented by a vector having the size of the vocabulary. The position in the vector that corresponds to the id of the vocable is set to one.

There are two main problems with this type of representation. The first is the dimensionality of the vectors. For example, for a vocabulary with the size of 30K, each word in the text will be represented by a vector of size 30K. A sentence with 20 words will be represented by an input having 600K parameters.

Another problem with the one-hot representation is that it treats the words as atomic units; there is no notion of similarity between the words. All words are equally distant from each other. A way to solve this problem is to create a representation based on a statistical language model.

The goal of the statistical language modeling is to learn the joint probability function of word sequences in a language. This probability function can be used to create a distributed representation where more statistically dependent words are closer. In this distributed representation, each word corresponds to a point in a feature space, so that similar words get to be closer to each other in that space [Vincent, Bengio, & Ducharme, 2000].

The main limitations of the statistical language modeling approach are the curse of dimensionality and the generalization of the representation learned. As we increase the number of words in a learned sequence from the training corpus, the computational cost to calculate the joint probability function becomes expensive and it is likely that this sequence will not occur again.

To overcome these limitations, neural network based language models are used to modeling continuous variables that generate distributed representations that have some local smoothness properties. For example, the sentences “The cat is walking in the bedroom” and “A dog was running in a room” should have similar representations because the words “dog” and “cat”, “the” and “a”, “room” and “bedroom”, “walking” and “running” have similar semantic and grammatical roles [Vincent, Bengio, & Ducharme, 2000].

In our work, the initial vocable representations are obtained from a pre-trained unsupervised neural language model proposed in Mikolov et al. [2013].

Figure 2.12 shows the architecture of two neural language models proposed by the authors.

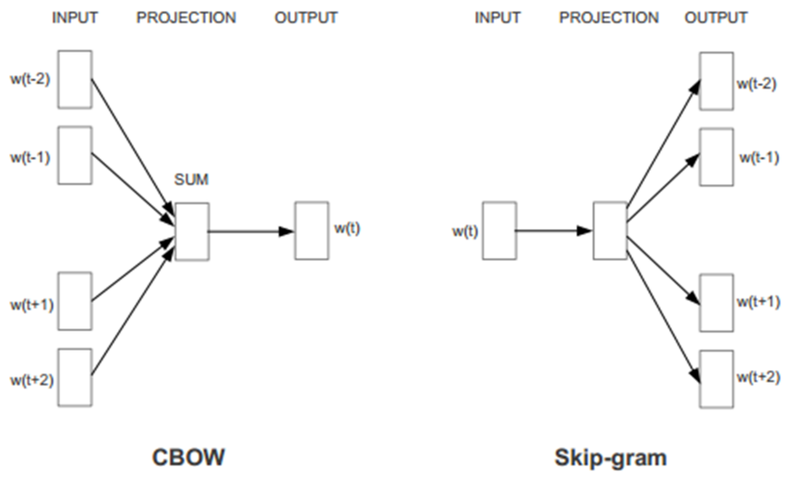

Figure 2.12. Architectures of the CBOW and Skip-gram neural language models. [Mikolov, Chen, Corrado, & Dean, 2013]

The Continuous Bag-of-Words (CBOW) neural language model predicts the current word based on the context, and the Skip-gram model predicts the neighborhood words given the current word.

The similarity between the words whose distributed representations are generated by these models can be measured using a word-offset technique where simple algebraic operations are performed on the word vectors. It was shown for example that the vector (”King”) minus vector (”Man”) plus vector (”Woman”) results in a vector that is closest to the vector representation of the word “Queen” [Zweig, Mikolov, & tau Yih, 2013].

Figure 2.13 shows a pictorial representation of this example.

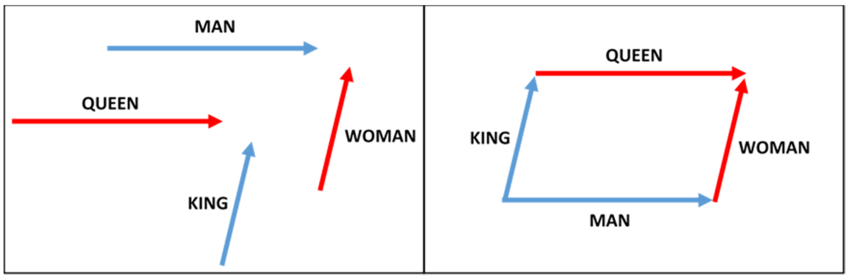

Figure 2.13. Algebraic operations on word vectors.

In ANN models, the initial values of the network parameters determine the quality of the learned representations. The same model trained with the same data set using different initial values for the network parameters can yield different solutions that differ substantially in quality. Different initial values will bias the learning algorithm to develop some type of feature detection units at the hidden layers, but not others [Golden, 1996].

In the context of NLP, the use of word embeddings, in the models that consider the words as the text building blocks, can be viewed as a prior knowledge information strategy [Gülçehre & Bengio, 2013].

Although some controversies exist about the ability of the word embeddings to capture semantics of word sequences, there are experiments showing that their use can improve the performance of some models on NLP benchmarks [Lev, Klein, & Wolf, 2015].

Chapter 3

Related Work

In this chapter, we present the related work that apply or develop similar concepts used in this work. In the course of our research, we made an extensive literature review including tens of papers, books and online references, but we present only the works that are closer related to our work. We summarize five works that employ CNN architectures to solve the text categorization problem.

In Kalchbrenner et al. [2014], the authors proposed a deep CNN architecture to make semantic modelling of sentences. The model is named Dynamic Convolutional Neural Network (DCNN). It is based on the architecture of a Time Delay Neural Network (TDNN) [Collobert & Weston, 2008]. The authors addressed the limitations of TDNN while preserving its advantages.

The proposed deep CNN architecture has four layers. In the first layer, the input sentences are represented using word embeddings initialized using a pre-trained unsupervised model that predicts the contexts of occurrence for the words [Turian, Ratinov, & Bengio, 2010]. In the second and third layers, the resulting representations from the previous layers are convolved by a set of filters. The convolution operators are followed by dynamic k-max pooling and non-linearity operators. The term dynamic means that the number of the k maximum values selected by the pooling operators changes according to the sentence size and to the layer level where the operation happens. The output of the third layer is fully connected to a softmax non-linearity layer that predicts the probability distribution over the classes given the input sentence.

The network was trained to minimize the cross-entropy of the predicted and true class labels distributions by backpropagation using mini- batches. The 1D convolution operator was implemented using a Fast Fourier Transform function. The code was implemented in Matlab and the experiments were processed on a GPU device.

The authors tested the DCNN in four experiments: small-scale binary and multi-class sentiment prediction, six-way question classification and Twitter sentiment prediction by distant supervision. The network achieved excellent performance in the first three tasks and the error reduction with respect to the strongest baseline was greater than 25% in the last task.

Although their model deals only with sentences, the architecture proposed by the authors inspired most of the works that use the CNN architecture in the NLP domain, including our work. Our model accepts input texts of any size, which makes it usable in real NLP applications.

In Kim [2014], the author proposed four variants of a CNN architecture based on the work of Bottou et al. [2011]. The proposed CNN architecture has three layers. The four architecture variants are created changing the way that the word representations are initialized and updated during the training.

In the first variant, the word representations are initialized randomly and updated during the training. In the second variant, the word representations are derived from Google pre-trained vectors (Word2Vec) and they are not updated during the training. The third variant is the same as the second one, except by the fact that the word representations are updated during the training. The fourth variant is the innovation proposed by the author. It is a mixture from the second and third variants. It creates the concept of channels. Each channel has its own copy of pre-trained word representations. In one channel, the word representations are updated during the training, and, in the other channel, they are not updated.

The author made experiments with seven data sets. Five of them are for sentiment analysis tasks on user reviews. The performance of the models was compared with strong base lines like DCNN [Kalchbrenner, Grefenstette, & Blunsom, 2014]. The proposed models improved upon the state of the art on four out of seven tasks.

The results showed that unsupervised pre-training of word vectors is an important ingredient in deep learning models for NLP tasks. To avoid overfitting on one specific task, one can use two channels for the word representations. One is kept static and the other one is optimized for the specific task that the model is being trained.

Their model also deals only with sentences and, although it has three layers, it is not considered a deep model because it has only one convolutional layer. The sentence representations learned by their model is limited because of the lack of depth. Our model overcomes these limitations using a deep architecture.

In Johnson & Zhang [2015], the authors proposed a shallow CNN architecture using high dimensional word representations. The convolution operator is applied over sequences of words called regions. Two variants of high dimensional word representations are used. One of them is the traditional one-hot vector. The other one, is named bag of words CNN. In this variant, the words of a region share the same vector representation, where each position of the vector represents one index of the vocabulary. This approach is a balance in the trade off between the representations high dimensionality and the order of the words. It preserves the order of the regions in the sentence but the order of the words in each region is lost.

The models were implemented using the C++ programming language and they explore the parallel processing power of the GPUs. Two data sets of user reviews and one of topic classification are used to compare the performance of the model with other strong baseline algorithms. The results showed that the proposed architecture achieved an excellent performance compared with the state of the art algorithms that use low dimensional pre-trained word representations.

Although the use of an efficient implementation combined with a powerful GPU makes it feasible the adoption of one-hot representations, the lack of context of this type of representation makes it harder to their model to extract good semantics from the text. Our model makes use of the word embeddings as the initial representations for the vocables and updates them in the training process. This strategy helps our model to extract good semantics from the text, starting with generic representations and adjusting them to the context of the specific task.

In Denil et al. [2014], the authors proposed a deep CNN architecture that explicitly extract representations for the input text at the sentence and document levels. The network has four layers and it is similar to the one presented in Kalchbrenner et al. [2014], except for the fact that, in the third layer, the sentence representations are concatenated to form the document representation. The convolution, k-max pooling and non-linearity operations are the same used in Kalchbrenner et al. [2014].

The innovation introduced by the authors is the use of a deconvolution technique used in the computer vision domain to generate interpretable visualizations of the deep layers activations in convolutional neural networks [Taylor, Fergus, & Zeiler, 2011]. To generate the saliency map for a given document, the authors applied the same technique used in Simonyan et al. [2013].

The authors proposed a way to measure the extraction quality of the most relevant sentences using them as a summarization for the reviews of the IMDB data set. The model is trained using the whole text of the reviews and the accuracy of the predicted sentiment is compared with the accuracy of the model trained using only the sentences extracted through the deconvolution process. The results show that the proposed model outperforms the baseline methods on the task of extracting the most relevant sentences from text documents.

Although the architecture of their model induces the creation of a hierarchy of representations, as our model does, the use of only two convolutional layers and the restriction on the number of words of the input text make the use of their model restricted to documents of small size. Our model has three convolutional layers and accepts input texts of any size, which makes it usable in real NLP applications.

In Zhang et al. [2015], the authors proposed a deep CNN architecture for text categorization using features extracted from character level representations. The network has nine layers composed of six convolutional and three full-connected layers.

In the input layer, it is created a representation for the input text using the one-hot encoding of the 70 alphabet symbols that represents the last 1014 text characters. The first six layers are made up by a sequence of 1D convolution, non-linearity and max pooling operators. The last three layers are made up by a sequence of linear and dropout operators. The last layer has a log softmax operator that gives the class labels log probabilities for the input text representation. The gradients are obtained by backpropagation and the optimization is done through Stochastic Gradient Descent (SGD) using mini-batches.

To evaluate their model, the authors built eight large-scale data sets. The model was trained using these data sets to make sentiment analysis and topic classification tasks. The authors implemented traditional models such as bag of words, n-grams and their TFIDF variants, and deep learning models such as word-based CNNs and LSTM to be used as baselines. The character-level CNN models achieved the state of the art performance on four of the eight tasks.

The use of characters as semantic units demands a huge number of samples to their model to learn good representations for a sequence of characters. Our model adopts the prior knowledge principle making use of the word embeddings as the initial representations for the vocables. This strategy makes our model learn good semantics using significantly less training samples.

Chapter 4

Model

In this chapter, we detail the architecture of our model starting by presenting an overview of the data flow and describing the text encoding mechanism, then we exam the design of the deep architecture and finally we talk about the network optimization algorithm used to update the network parameters.

4.1 Data Flow

Figure 4.1 presents a flowchart representing the data flow of our model. The data sets are split in training and testing sets. The vocables occurring in the training set are used to generate the vocabulary in the text encoding process. The word embeddings are read from a binary file obtained from a pre-trained model.

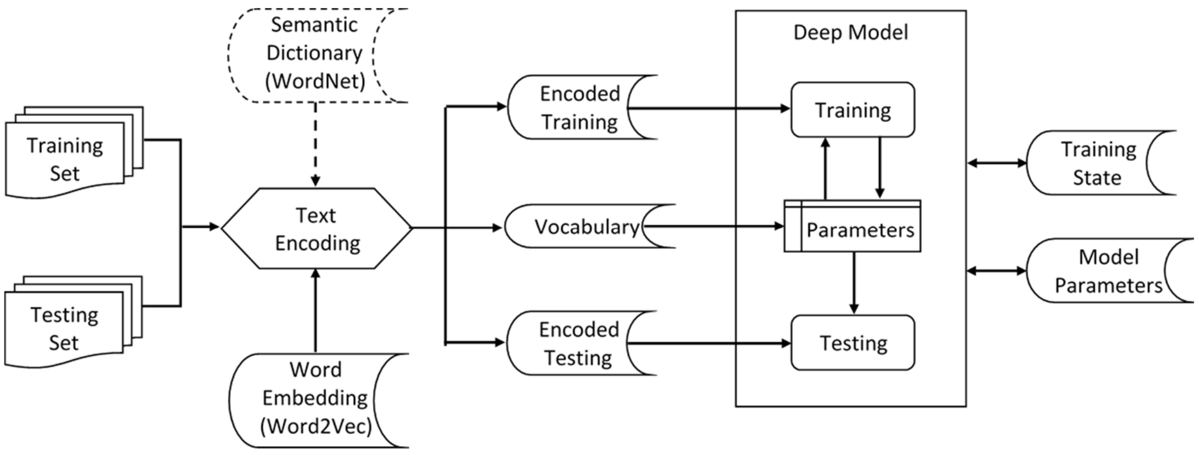

Figure 4.1. Model data flow.

We implemented two models. The WordNet semantic dictionary corpus is used in the Deep NLP WordNet model to get synonyms for the vocables in the text encoding process. The doted lines in the chart denote that the WordNet corpus is not used in the Deep NLP model.

The vocabulary generated by the text encoding process is used to encode the texts of the training and testing data sets and it is stored in a binary file that will be loaded by the deep CNN. The encoded texts of the training and testing data sets are also stored in binary files that will be used by the deep CNN in the training and testing process. The updating of the vocabulary representations can be enabled in the training process.

The training state and the network parameters are saved in binary files, so they can be loaded later in the testing process.

4.2 Text Encoding

The first step in the text encoding process is the text tokenization. Because of our model explicitly creates intermediate representations for sentences, we first tokenize the text into sentences, then we tokenize the sentences into words.

The second step in the text encoding process is the vocabulary generation. There are two steps in the process of building the vocabulary. The first step is to select the vocables that will compose the vocabulary. In compliance with the principle that the content of the testing samples should not be viewed by the model before the testing phase, we take into account only the vocables present in the training samples to build the vocabulary. In this step, there is an important decision to be made, the vocabulary size.

Because our model learns its parameters in a supervised way and the vocable initial representations are considered parameters of the network, the vocabulary size has a huge impact on the number of parameters that have to be learned by the model.

Although there is not a rule of thumb to determine the vocabulary size, one point that must be considered is the equilibrium between the number of training samples per class and the number of parameters that have to be learned. To constraint the vocabulary size, we use the strategy of selecting only the vocables that appear in the training samples at a minimum frequency.

The second step in the vocabulary generation process is to assign an initial representation to the vocables. In our model, the vocable initial representations are obtained from a pre-trained unsupervised neural language model publicly available[2] (Word2Vec). These initial representations are adjusted to the specific context of the training samples during the training phase.

When a vocable is not found in the Word2Vec, we assign a random value to its initial representation. In the model implemented using the WordNet corpus, before assigning a random value to the initial representation of a vocable, we first try to find a WordNet synonym, lemma or stem whose vocable is present in the Word2Vec.

The WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The WordNet's structure makes it a useful tool for computational linguistics and natural language processing [Miller, 1995].

The last step in the text encoding process is to associate each text word of the training and testing samples with its correspondent vocable in the vocabulary. This association is made assigning to each text word an integer value that is the index of its correspondent vocable present in the vocabulary.

Instead of ignoring the words whose vocables are not present in the vocabulary, we assign to them the index of one of the generic vocables specifically created for this purpose (#NUMBER#, #SYMBOL#, #UNKNOWN#). In the model implemented using the WordNet corpus, before assigning the index of a generic vocable to a unknown word, we first try to find a WordNet synonym, lemma or stem whose vocable is present in the vocabulary.

This strategy enhances the robustness of our model through the generalization of its learned representations. Even when the model encounter a text with many vocables that it cannot find in its vocabulary, it is able to replace them by some cognitive synonym that is present in the vocabulary. This is similar to what the humans do when they encounter an unknown word in a text. They search the unknown word in a dictionary or thesaurus and

[2] https://code.google.com/archive/p/word2vec/

replace it by a word whose semantic is already known in a similar context [Gülçehre & Bengio, 2013].

4.3 Deep Architecture

The design of the network architecture of our model is inspired by the deep CNN architectures used in the computer vision domain [LeCun, Bottou, Bengio, & Haffner, 1998] [Krizhevsky, Sutskever, & Hinton, 2012]. Figure 4.2 shows a diagram with the main components of our model architecture.

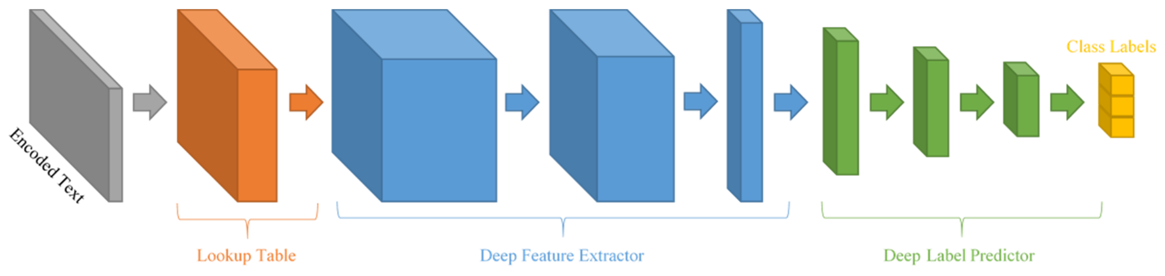

Figure 4.2. Deep NLP model architecture.

Our model implements a sequential standard feedforward architecture. The model is made up by seven layers that can be grouped into three main components. The first component is the lookup table. It stores the vocable representations assigned by the vocabulary building process. This component is responsible for translating the word encodings into word embeddings. The vocable initial representations are obtained from the publicly available[3] pre-trained Word2Vec binary file using 300-dimensional vectors. Because these representations are updated in the training phase, this component has the larger number of the network parameters.

The second component of our model architecture is the deep feature extractor. This component is responsible for extracting complex features from the text. Because we adopted the sentence compositional approach, we force the text to pass through layers that explicitly create intermediate representations for the sentences. At the upper layers of this component, the sentence representations are concatenated to create the text representation. This

[3] https://code.google.com/archive/p/word2vec/

component is made up by three layers. Each of these layers is arranged as a sequence of temporal convolution, non-linearity and max pooling modules. The temporal convolution module is responsible for creating new words and sentence representations.

At the sentence level layers, the temporal convolution module convolves a set of filters through the words of each separated sentence. The filters of the same module have a fixed size and they are shared among the input text sentences. This approach reduces the number of parameters that have to be learned by the model. The filter size does not depend on the number of sentences. The main consequence of this design decision is that there is no restriction on the size of the input text that can be processed by our model.

At the document level layer, the temporal convolution module convolves a set of filters through the concatenated sentence representations created in the previous layer. The output of this layer is a set of features that represent the whole text.

The non-linearity modules are responsible for extracting complex features from the data and making them more easily separable [Goodfellow, Bengio, & Courville, 2016].

The max pooling modules are responsible for selecting the most important features and consequently reducing the dimensionality of the learned representations [Boureau, Ponce, & LeCun, 2010].

The third component of our model architecture is the deep label predictor. This component is made up by three fully connected layers followed by a classifier. Each of these layers is arranged as a sequence of linear transformation, non-linearity and dropout modules.

The sequence of linear transformation modules is responsible for extracting features that are more abstract. Each layer narrows the number of features from previous layer reducing the dimensionality of the learned representations.

The dropout modules are responsible for reducing the model overfitting. They induce the network to learn features that are more robust [Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012]. They are activated only in the training phase.

The classifier module is responsible for associating a class label probability distribution for the text representation produced by the model.

4.4 Optimization

Our model is trained to minimize the Negative Log-Likelihood (NLL) loss function. The gradients are accumulated using the backpropagation algorithm. [Rumelhart, Hinton, & Wilson, 1986].

The network parameters are updated using a mini-batch version of the Stochastic Gradient Descent (SGD) algorithm called momentum update [Sutskever, Martens, Dahl, & Hinton, 2013]. This approach helps to accelerate the learning of the network parameters.

We also randomly shuffle the training data set before each epoch, which tends to provide better convergence [LeCun, Bottou, Orr, & Müller, 2012].

Chapter 5

Implementation

In this chapter, we talk about the programming languages and packages used to implement our model. We also give some details about the modules used to implement the network architecture presented in Chapter 4.

5.1 Programming Languages

We used the Python programming language to implement most of the text encoding process. We chose this language because of its aptitude for data manipulation and for the convenience of the Natural Language Toolkit (NLTK) platform implemented in Python [Bird, Klein, & Loper, 2009]. In the model implemented using the WordNet corpus, we also used the Python language to implement the vocabulary generation module.

The Lua programing language was used to implement the deep CNN modules [Ierusalimschy, 2006]. In the model implemented not using the WordNet corpus, we also used Lua to implement the vocabulary generation module. We chose this language because it is used to implement the computing framework we selected.

5.2 Computing Framework

We selected the Torch7 computing framework to implement our deep CNN model [Kavukcuoglu, Farabet, & Collobert, 2011]. We chose this framework because of its wide support for deep learning algorithms, its modularity and its efficiency on the use of the GPUs.

The Torch7 Neural Network (NN) package provides an easy and modular way to build and train neural networks. Each module implements the fundamental methods and the necessary state variables for training a neural network. The modules are grouped into containers that in turn can be assembled like Lego building blocks to create complex models.

Figure 5.1 shows the main components of a Torch7 NN package module. The forward method computes the module's output from its input and it stores the result in a state variable. The backward method computes the gradients with respect to the module's input and with respect to the module's parameters. During the backward pass, the gradient with respect to the module's parameters is accumulated and it is zeroed after being updated.

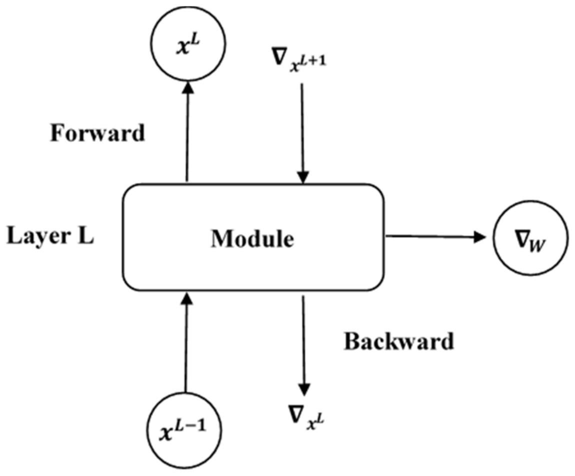

Figure 5.1. Main components of a Torch7 NN package module.

The cuTorch and cuNN packages provide a GPU implementation for many of the Torch7 backend and NN package modules. They are implemented using the CUDA API and they inherit all the CUDA's efficiency on the use of Nvidia's GPUs.

These packages give total control over the RAM to/from GPU's memory data transfers. This issue is critical for a successfully implementation of deep learning models that process huge data sets. In our experiments, the implementation of our model sustains the utilization rate of the Nvidia's K40 GPU at 95% on average during the training and testing processing. The transfer of a whole model from CPU to/from GPU is made merely through the call of a single method.

5.3 Modules

Figure 5.2 displays a Lua code fragment excerpted from the module that implements our deep CNN model. The code fragment shows how the NN package modules are stacked to create the network architecture presented in Chapter 4.

```
-- First layer: input (encoded text: nSentences x nWords)
  model.modules[1] = {name = "LookupTable", parameters = {vocabulary = model.config.vocabulary}}

  -- Second layer.
  model.modules[2] = {name = "TemporalConvolution", parameters = {inputFrameSize = 300,
outputFrameSize = 200, kW = model.config.minWordsSentence, dW = 1}}
  model.modules[3] = {name = "Threshold", parameters = {}}

  -- Third layer.
  model.modules[4] = {name = "TemporalConvolution", parameters = {inputFrameSize = 200,
outputFrameSize = 200, kW = 3, dW = 1}}
  model.modules[5] = {name = "Threshold", parameters = {}}
  model.modules[6] = {name = "SpatialAdaptiveMaxPooling", parameters = {outputWidth = 200,
outputHeight = 3}}

  -- Fourth layer.
  model.modules[7] = {name = "View", parameters = {}}
  model.modules[8] = {name = "TemporalConvolution", parameters = {inputFrameSize = 200,
outputFrameSize = 100, kW = 3, dW = 1}}
  model.modules[9] = {name = "Threshold", parameters = {}}
  model.modules[10] = {name = "View", parameters = {}}
  model.modules[11] = {name = "SpatialAdaptiveMaxPooling", parameters = {outputWidth = 100,
outputHeight = 15}}

  -- Fifth layer.
  model.modules[12] = {name = "View", parameters = {}}
  model.modules[13] = {name = "Linear", parameters = {inputDimension = 1500, outputDimension =
1000}}
  model.modules[14] = {name = "Threshold", parameters = {}}
  model.modules[15] = {name = "Dropout", parameters = {probability = 0.5}}

  -- Sixth layer.
  model.modules[16] = {name = "Linear", parameters = {inputDimension = 1000, outputDimension = 500}}
  model.modules[17] = {name = "Threshold", parameters = {}}
  model.modules[18] = {name = "Dropout", parameters = {probability = 0.5}}

  -- Seventh layer.
  model.modules[19] = {name = "Linear", parameters = {inputDimension = 500, outputDimension =
model.config.outputClasses}}
  -- Output layer.
  model.modules[20] = {name = "LogSoftMax", parameters = {}}
```

Figure 5.2. Lua code fragment of the deep CNN model implementation.

Figure 5.3 displays an excerpt from the execution log of the code showed in Figure 5.2. The network layers were grouped using two sequential modules that, in turn, were grouped into a sequential container. This approach makes easy to disable the vocable initial representations updating in the lookup table through the setting of a configuration parameter.

```
nn.Sequential {
  [input -> (1) -> (2) -> output]
  (1): nn.Sequential {
    [input -> (1) -> output]
    (1): nn.LookupTable
  }
  (2): nn.Sequential {
    [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9) -> (10) -> (11) -> (12) -> (13) -> (14) ->
(15) -> (16) -> (17) -> (18) -> (19) -> output]
    (1): nn.TemporalConvolution
    (2): nn.Threshold
    (3): nn.TemporalConvolution
    (4): nn.Threshold
    (5): nn.SpatialAdaptiveMaxPooling
    (6): nn.View
    (7): nn.TemporalConvolution
    (8): nn.Threshold
    (9): nn.View
    (10): nn.SpatialAdaptiveMaxPooling
    (11): nn.View
    (12): nn.Linear(1500 -> 1000)
    (13): nn.Threshold
    (14): nn.Dropout(0.500000)
    (15): nn.Linear(1000 -> 500)
    (16): nn.Threshold
    (17): nn.Dropout(0.500000)
    (18): nn.Linear(500 -> 10)
    (19): nn.LogSoftMax
  }
}
```

Figure 5.3. Execution log excerpt of the deep CNN model.

In the next subsections, we give some details about the NN package modules that we used to implement our deep CNN model.

5.3.1 Lookup Table

In the first layer of our model, we used a lookup table module from the NN package. This layer is responsible for decoding the input text words into word embeddings. The first reason for using a lookup table has to do with the efficient use of the GPU. The bus bandwidth is one of the bottlenecks that prevents the efficient use of the GPUs. Instead of decoding the input text words in the host's memory and send them to the GPU device, we make the input text decoding directly in the GPU's memory. This approach saves the bus bandwidth of sending 2,392 bytes per word when we use a 300-dimensional word embedding.

The other reason for using a lookup table for decoding the input text has to do with the updating of the vocable initial representations. When we enable the vocable representations updating, they become part of the network parameters, therefore they must stay together with the other network parameters in the GPU's memory.

The vocabulary is sent to the GPU device only once when the model is instantiated for the first time. The vocable initial representations are stored in a matrix and become the weight parameters of the lookup table.

The encoded input text is stored in a matrix. The lines of the matrix correspond to the text sentences and the columns correspond to the sentence words. The number of lines is unlimited. The number of columns is equal to the number of words in the largest sentence of the input text. The smaller sentences are zero padded to the right.

Figure 5.4 shows a diagram of how the input text is decoded through the forward method of the lookup table module. The output of the lookup table forward method is stored in a 3D tensor.

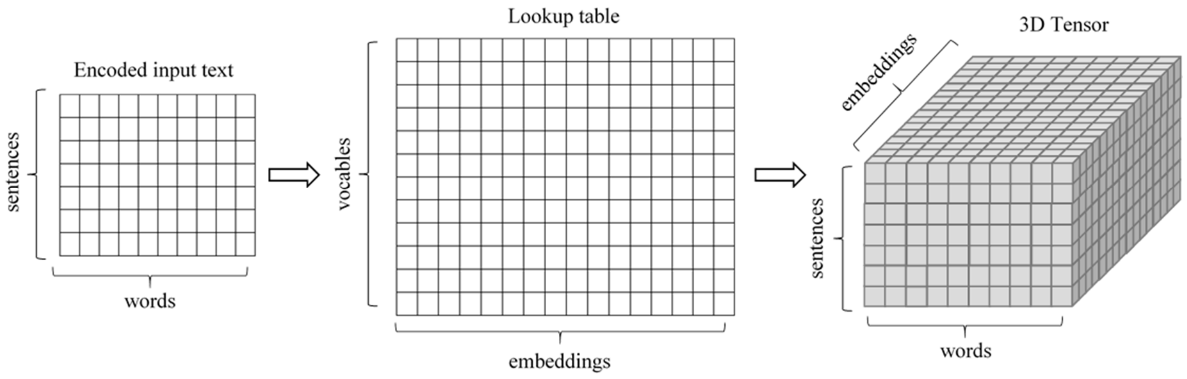

Figure 5.4. Input text decoding through the lookup table forward method.

5.3.2 Temporal Convolution

The temporal convolutional module applies a 1D convolution using a set of filters over an input sequence made up of input frames. Each filter generates an output frame. The size of the filter is determined by the number of input frames and by the width of the convolution.

Figure 5.5 shows a diagram of how the forward method of the temporal convolution module operates over the decoded input text in the second layer of our model. In this example, each embedding dimension corresponds to an input frame. Each color represents a different filter. The temporal convolution produces an output with the same number of dimensions of the input.

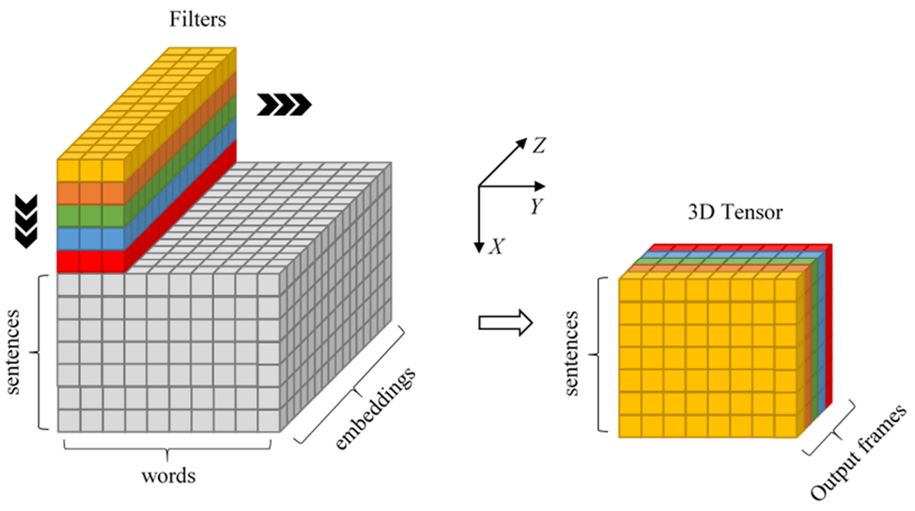

Figure 5.5. Temporal convolution over the decoded input text.

In our model, the temporal convolution module is used to extract new representations for words and sentence sequences. At the document level layer, the temporal convolution module convolves a set of filters through the concatenated sentence representations created in the previous layer.

The value of each element of the output produced by the temporal convolutional operation over a 3D input tensor can be precisely defined as:

$$\mathcal{O}_{x,y,z} = \sum_{j=1}^{W} \sum_{k=1}^{N} \mathcal{F}_{z,j,k} * \mathcal{I}_{x,y-1+j,k} \tag{5.1}$$

where, $\mathcal{F}$ is the set of filters, $\mathcal{I}$ is the 3D input tensor, W is the width of the filters and N is the number of input frames.

5.3.3 Threshold

The non-linearity function used in our model is implemented by the threshold module. The threshold function is similar to Rectified Linear Units (ReLUs) [Nair & Hinton, 2010]. It is defined as:

$$f(x) = \max\{0, x\} \tag{5.2}$$

Figure 5.6 shows a plot of the threshold function and its derivative.

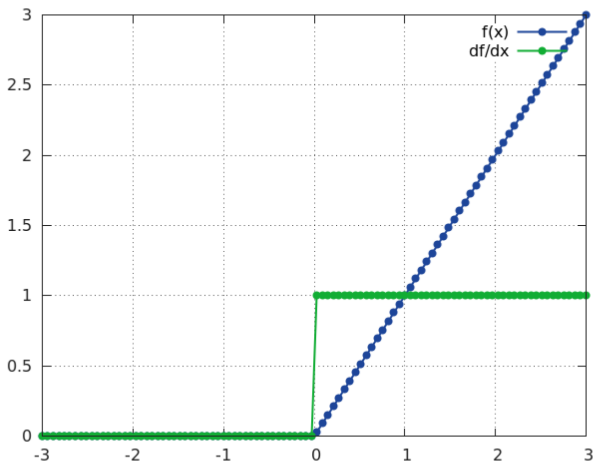

Figure 5.6. Plot of the threshold function and its derivative.

5.3.4 Spatial Adaptive Max Pooling

The spatial adaptive max-pooling module is a 2D version of the temporal max-pooling operation, which adapts its parameters dynamically such that the output has a fixed size. Differently from the traditional max-pooling operators that select the maximum values among all the features of a dimension, the spatial adaptive max pooling splits the dimension into segments, according to the desired output size, and it selects the maximum value from each segment.

This approach helps to avoid the adverse effect caused by the sentences right padding made in the input layer. When the number of words in a sentence is too small, depending on the weights and biases values associated by the convolution operation to the paddings, the traditional max-pooling operator could select only these values as being the most important features of the sentence.

5.3.5 View

The view module creates a new view for the input tensor using the sizes passed to the class constructor. This module is used in the fourth layer of our model to concatenate the sentence representations created in the previous layer. It is also used in the fifth layer to flattening the fourth layer output transforming it into an 1D tensor.

5.3.6 Linear

This module applies a linear transformation to an 1D input tensor. This module is used to implement the fully connected layers of the deep feature extractor component of our model.

5.3.7 Dropout

The dropout module forwards the input masking its elements using binary samples from a Bernoulli distribution. The input elements associated with a mask position that has a zero value are dropped, that is the value of their correspondent output elements are set to zero.

The input elements that are not dropped have the value of their correspondent output elements scaled by a factor of $1/(1-p)$, where p is the probability of an element being dropped. The dropout module is activated only in the training phase.

In our model, the dropout module is used in the output of the fully connected layers and the drop out probability is set to 0.5.

5.3.8 Log Softmax

The log softmax module implements the log normalized exponential function. The log softmax function is the gradient-log-normalizer of the categorical probability distribution. It is defined as:

$$f_i(x) = \log \frac{e^{x_i}}{\sum_1^k e^{x_j}} \quad , \begin{cases} f_i(x) \geq 0 \\ \sum_{i=1}^{k} f_i(x) = 1 \end{cases} \tag{5.3}$$

where, k is the number of classes and $f_i(x)$ is the log-probability associated with the class i.

The log softmax module is used as the classifier in the output layer of our model. It associates a log-probability to each class label for the text representation produced in the seventh layer output.

5.3.9 ClassNLLCriterion

Differently from the modules presented above, this module belongs to a set of the NN package's modules called criterions. Criterions are helpful to train a neural network. Given an input and a target, they compute a gradient according to a given loss function.

This module implements the Negative Log-Likelihood (NLL) loss function. It is used in the training module of our model. The log-probability for each class label, given by the forward method of the log softmax module in the output of our network, is used as input to the forward and backward methods of the ClassNLLCriterion module. The computed gradient is back propagated through the network using the backward method.

5.4 Detailed Network

Figure 5.7, Figure 5.8 and Figure 5.9 show a detailed diagram of a complete forward pass through the entire deep CNN of our model.

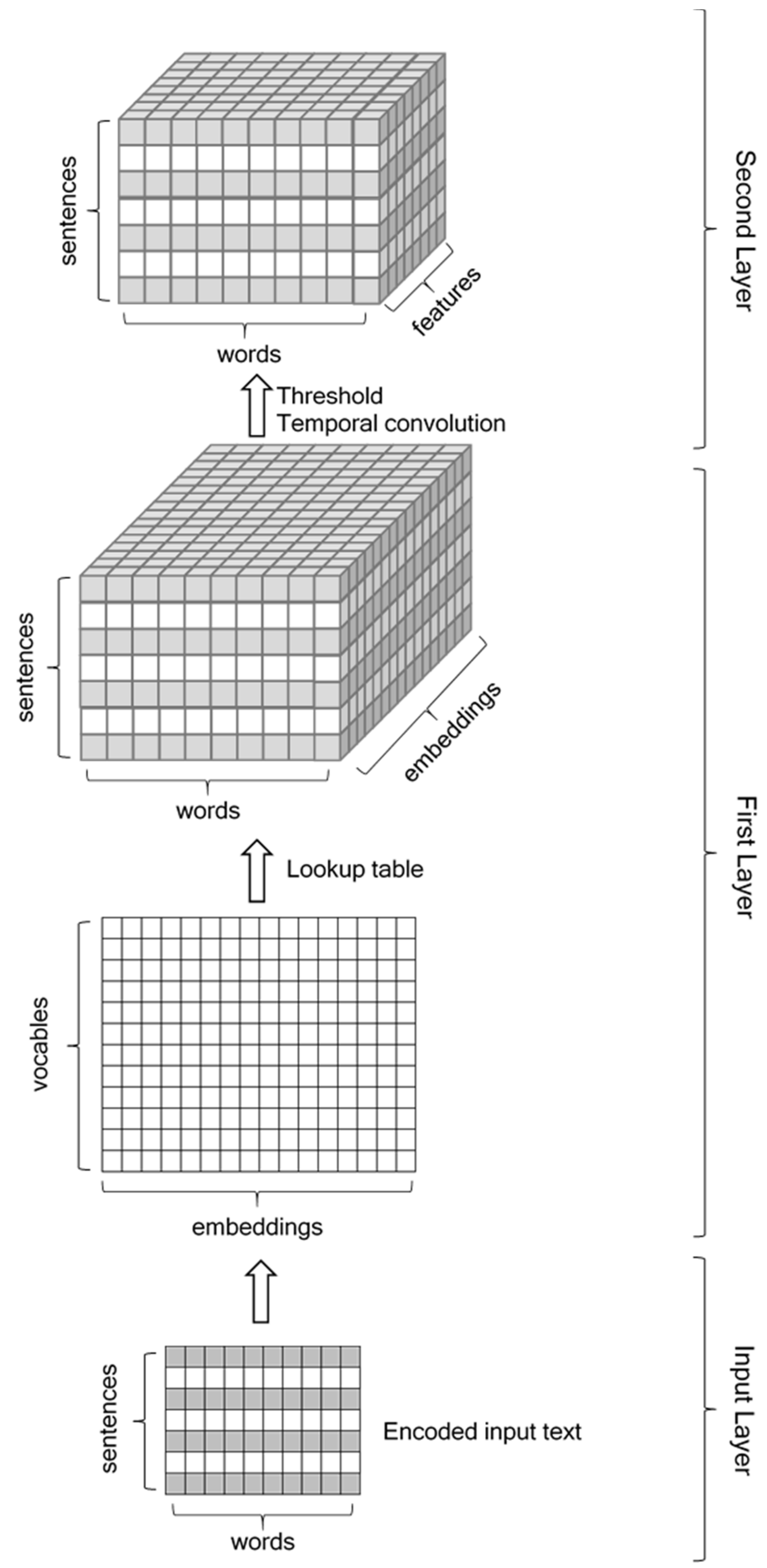

Figure 5.7. Detailed diagram of our deep CNN architecture.

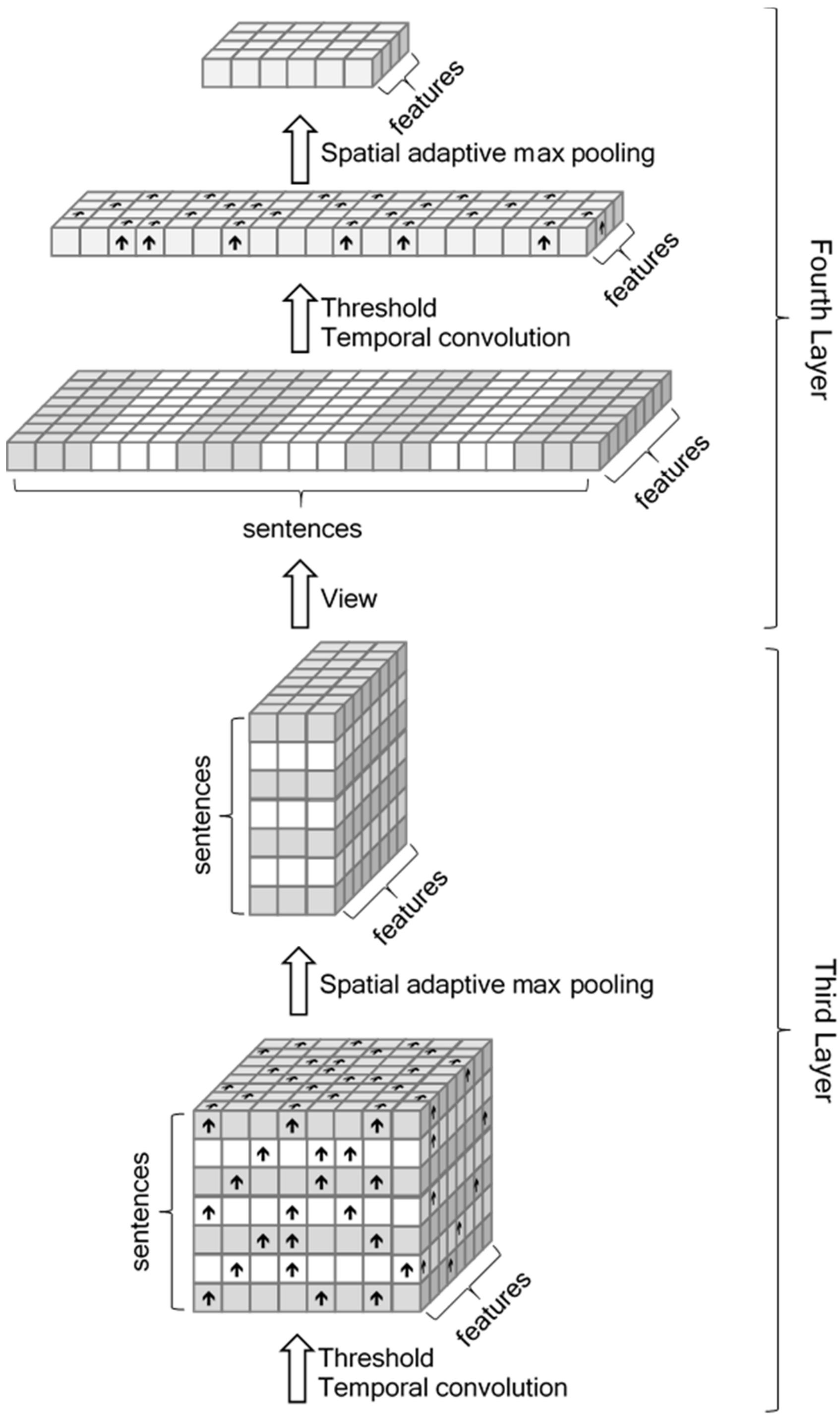

Figure 5.8. Detailed diagram of our deep CNN architecture.

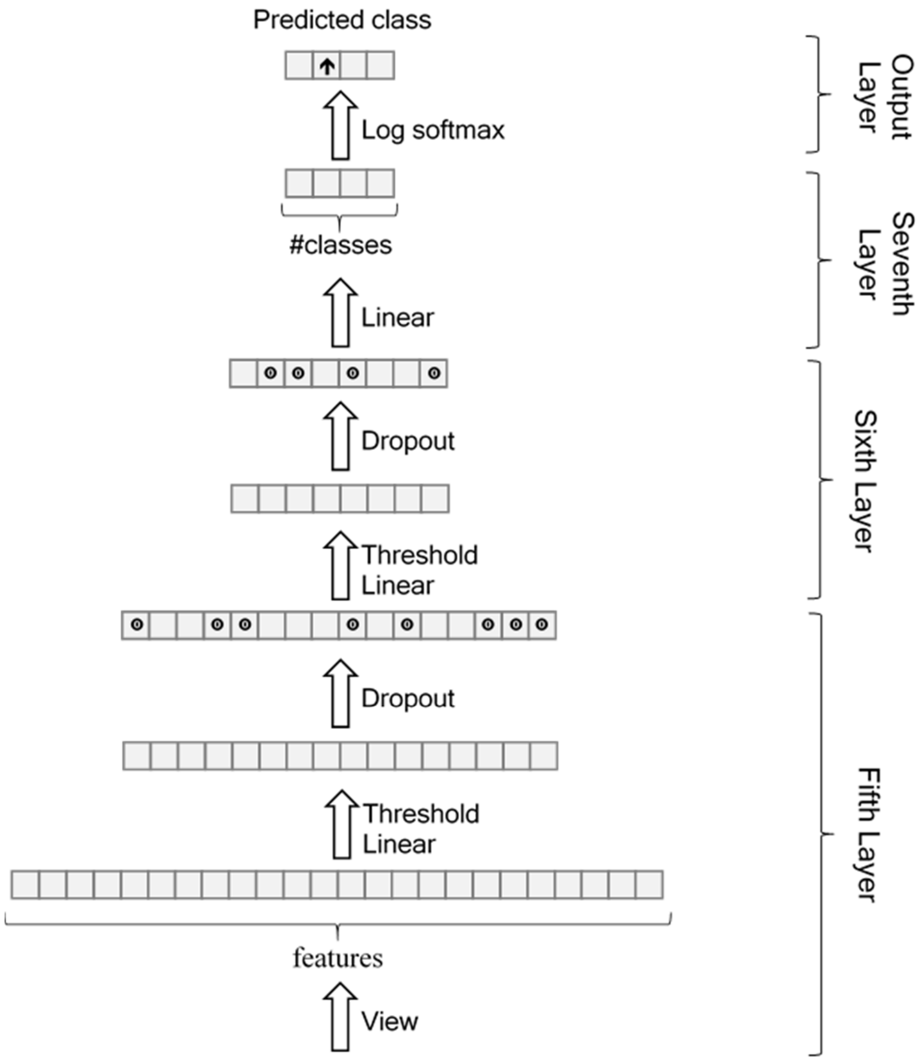

Figure 5.9. Detailed diagram of our deep CNN architecture.

The size of the data structures showed in these diagrams are not scaled. In the input layer, the number of sentences and words of the encoded input text are not fixed.

In the first layer, the number of vocables in the lookup table depends on the data set and it is determined by the minimum vocable frequency parameter. The model does not impose a limit on the size of the vocabulary, but, in most of the cases, the number of vocables is at least one hundred times greater than the size of the word embeddings. In our experiments, we used word embedding having a size of 300.

In the second layer, the size of the word embeddings is decreased to 200 through a temporal convolutional operation.

In the third layer, the sentences width is decreased by a temporal convolutional operation, using a filter width of three, followed by a spatial adaptive max pooling operation using a fixed output size of three. The symbol ⬆ denotes the highest values selected by the spatial adaptive max pooling operation.

In the fourth layer, a view operation concatenates the 3D tensor sentence representations into a 2D tensor. The sentences width and the number of features are decreased by a temporal convolutional operation using a filter width of three and output frame size of one hundred, followed by a spatial adaptive max pooling operation using an output size of fifteen.

In the fifth layer, a view operation concatenates the 2D tensor sentence representations into a 1D tensor. The number of features is decreased to 1000 by a linear operation. The symbol ⓿ means that the value of a feature is considered as being zero by the dropout module in the training phase.

In the sixth layer, the number of features is decreased to 500 by a linear operation.

In the seventh layer, the number of features is decreased to the number of classes by a linear operation.

Chapter 6

Evaluation

In this chapter, we describe the experiments carried out to evaluate our model. We initially introduce the data sets used, then we talk about the methodology and hardware used to run the experiments.

6.1 Data Sets

To evaluate our model, we used several large-scale data sets, publicly available[4], built specifically to assess deep CNN architectures [Zhang, Zhao, & LeCun, 2015]. The data sets are used for topic classification and sentiment analysis tasks.

Table 6.1 presents the characteristics of the large-scale data sets used in our experiments.

Table 6.1. Characteristics of the large-scale data sets used in the experiments.

Data Set	Classes	Training Samples	Testing Samples	Task
AG's News	4	120,000	7,600	Topic
DBPedia	14	560,000	70,000	Topic
Yelp Review Polarity	2	560,000	38,000	Sentiment
Yelp Review Full	5	650,000	50,000	Sentiment
Yahoo! Answers	10	1,400,000	60,000	Topic
Amazon Review Polarity	2	3,600,000	400,000	Sentiment

[4] http://goo.gl/JyCnZq

6.1.1 AG’s News

The original AG data set is a collection of more than 1 million news articles. News articles has been gathered from more than 2,000 news sources by ComeToMyHead in more than one year of activity. ComeToMyHead is an academic news search engine, which has been running since July 2004. The dataset is provided by the academic community for research purposes in data mining and information retrieval.

The AG's news data set was constructed by choosing the four largest classes from the original AG corpus using only the title and description fields. Each class contains 30,000 training samples and 1,900 testing samples. The total number of training samples is 120,000 and testing is 7,600 [Zhang, Zhao, & LeCun, 2015].

The AG's news data set classes and their corresponding labels used for topic classification are:

1- World 2- Sports 3- Business 4- Science/Technology

Table 6.2 shows some samples of the AG's news data set. In our experiments, we concatenated the title and description fields on the training and testing processing.

Table 6.2. AG's news data set samples.

Class	Title	Description
1	On front line of AIDS in Russia.	An industrial city northwest of Moscow struggles as AIDS hits a broader population.
2	Giddy Phelps Touches Gold for First Time.	Michael Phelps won the gold medal in the 400 individual medley and set a world record in a time of 4 minutes 8.26 seconds.
3	Fears for T N pension after talks.	Unions representing workers at Turner Newall say they are 'disappointed' after talks with stricken parent firm Federal Mogul.
4	IBM Chips May Someday Heal Themselves.	New technology applies electrical fuses to help identify and repair faults.

Table 6.3 presents some statistics of the AG's news data set documents. This statistics were acquired in the text encoding process.

Table 6.3. AG's news documents statistics.

Data Set	Sentences per sample				Words per sentence			
	Min	Max	Mean	Stdev	Min	Max	Mean	Stdev
Testing	1	24	2.64	0.99	1	131	16.86	11.95
Training	1	19	2.64	0.99	1	128	16.92	11.97

6.1.2 DBPedia Ontology

The DBPedia is a large-scale data set emerged from the crowd-sourced community effort to extract structured information from Wikipedia [Lehmann, et al., 2015].

The DBPedia ontology classification data set was constructed by picking 14 non-overlapping classes from the original DBPedia 2014 using only the title and abstract fields of each Wikipedia article. From each of these 14 ontology classes, 40,000 training samples and 5,000 testing samples were randomly chosen. Therefore, the total size of the training data set is 560,000 and testing data set is 70,000 [Zhang, Zhao, & LeCun, 2015].

The DBPedia ontology data set classes and their corresponding labels used for topic classification are:

1- Company	2- Educational Institution	3- Artist
4- Athlete	5- Office Holder	6- Mean Of Transportation
7- Building	8- Natural Place	9- Village
10- Animal	11- Plant	12- Album
13- Film	14- Written Work	

Table 6.4 shows some samples of the DBPedia ontology data set. In our experiments, we used only the abstract field on the training and testing processing.

Table 6.4. DBPedia ontology data set samples.

Class	Title	Abstract
1	Export-Import Bank of Romania.	Exim Bank is The Export-Import Bank of Romania based in Bucharest.
2	Strong Vincent High School.	Strong Vincent High School is a public high school in Erie Pennsylvania.
3	Lizzy Pattinson.	Elizabeth Lizzy Pattinson is an English singer and songwriter.
4	Henry Nicoll (cricketer).	Henry Russell Nicoll (1883–1948) was a Scottish cricketer.
5	Samuel Douglas.	Samuel Douglas (1781–July 8 1833) was a Pennsylvania lawyer and state Attorney General.
6	INS Sharada (P55).	INS Sharada (P55) is a Sukanya class patrol vessel of the Indian Navy.
7	Château de Sauvebœuf (Aubas).	Château de Sauvebœuf is a château in Dordogne Aquitane France.
8	Lake Pacucha.	Lake Pacucha is a lake in Peru.
9	Vindornyaszőlős.	Vindornyaszőlős is a village in Zala county Hungary.
10	Bertula.	Bertula is a genus of moths of the Noctuidae family.
11	Dracula Polyphemus.	Dracula Polyphemus is a species of orchid.
12	O Corpo Sutil (The Subtle Body).	O Corpo Sutil (The Subtle Body) is an album by musician Arto Lindsay.
13	Rahgir.	Rahgir is a Bollywood film. It was released in 1943.
14	Red Claw (novel).	Red Claw is a 2009 science fiction novel by Philip Palmer.

Table 6.5 presents some statistics of the DBPedia ontology data set documents. This statistics were acquired in the text encoding process.

Table 6.5. DBPedia ontology documents statistics.

Data Set	Sentences per sample				Words per sentence			
	Min	Max	Mean	Stdev	Min	Max	Mean	Stdev
Testing	1	32	2.88	1.59	1	519	17.70	9.63
Training	1	39	2.87	1.58	1	1327	17.73	9.66

6.1.3 Yelp Review Polarity

The original Yelp reviews data set consists of 1,569,264 reviews extracted from the Yelp Data Set Challenge 2015 data[5].

The Yelp reviews polarity data set was constructed by considering stars 1 and 2 negative and stars 3 and 4 positive. For each polarity, 280,000 training samples and 19,000 testing samples were taken randomly. In total, there are 560,000 training samples and 38,000 testing samples. Negative polarity is class 1 and positive is class 2. [Zhang, Zhao, & LeCun, 2015].

Table 6.6 shows some samples of the Yelp reviews polarity data set. In our experiments, we used the whole review text field on the training and testing processing.

Table 6.6. Yelp reviews polarity data set samples.

Class	Review text
1	The food is good. Unfortunately, the service is very hit or miss. The main issue seems to be with the kitchen, the waiters and waitresses are often very apologetic for the long waits and it's pretty obvious that some of them avoid the tables after taking the initial order to avoid hearing complaints.
2	Arrived around midnight and the front desk was ready for us, check in was quick and we were able to turn in. The room was clean, bed comfy, the desk was huge...but the bathroom was small. Breakfast in the morning was very convenient, several choices, and the coffee hit the spot.

[5] http://www.yelp.com/dataset_challenge

Table 6.7 presents some statistics of the Yelp reviews polarity data set documents. This statistics were acquired in the text encoding process.

Table 6.7. Yelp reviews polarity documents statistics.

Data Set	Sentences per sample				Words per sentence			
	Min	Max	Mean	Stdev	Min	Max	Mean	Stdev
Testing	1	105	9.80	8.21	1	545	15.33	10.25
Training	1	148	9.83	8.24	1	745	15.35	10.26

6.1.4 Yelp Review Full

The original Yelp reviews data set consists of 1,569,264 reviews extracted from the Yelp Data Set Challenge 2015 data[6].

The Yelp reviews full star data set was constructed by randomly taking 130,000 training samples and 10,000 testing samples for each review star from 1 to 5. In total, there are 650,000 training samples and 50,000 testing samples [Zhang, Zhao, & LeCun, 2015].

Table 6.8 shows some samples of the Yelp reviews full star data set. In our experiments, we used the whole review text field on the training and testing processing.

Table 6.8. Yelp reviews full star data set samples.

Class	Review text
1	Don't waste your time. We had two different people come to our house to give us estimates for a deck (one of them the OWNER). Both times, we never heard from them. Not a call, not the estimate, nothing.
2	Service was okay, at best. I wouldn't go there again. They quoted me at thousands of dollars of repairs for my car to pass inspection. I took it somewhere else and had it done for a fraction of the quote.
3	The pizza is great. Other food items might disappoint. They do deliver! Service is hit and miss. There is one rude, smile-less bartender... I have actually seen him through the window and decided to go somewhere else because I was in the mood for good service.

[6] http://www.yelp.com/dataset_challenge

Table 6.8. Yelp reviews full star data set samples.

Class	Review text
4	A good Starbucks. There is always a line at this one due to its location but they do a great job of getting people served quickly. Today I had a salted camel mocha. It was pretty amazing. This location also has a fireplace, which is a nice touch for cold days.
5	I am a big fan of Max's for their local flair, real German food, and authentic Pittsburgh feeling. They did not sell out, are not overly commercialized, and should be supported for the long standing quality service to the city. Thanks, Max's.

Table 6.9 presents some statistics of the Yelp reviews full star data set documents. This statistics were acquired in the text encoding process.

Table 6.9. Yelp reviews full star documents statistics.

Data Set	Sentences per sample				Words per sentence			
	Min	Max	Mean	Stdev	Min	Max	Mean	Stdev
Testing	1	110	9.92	8.27	1	441	15.38	10.18
Training	1	131	9.89	8.20	1	796	15.41	10.21

6.1.5 Yahoo! Answers

The original Yahoo! Answers Comprehensive Questions and Answers corpus contains 4,483,032 questions and their answers.

The Yahoo! Answers topic classification data set was constructed from the original Yahoo! Answers Comprehensive Questions and Answers corpus using the question title, question content and best answer fields of the 10 largest main categories. Each class contains 140,000 training samples and 6,000 testing samples. Therefore, the total number of training samples is 1,400,000 and testing samples is 60,000 [Zhang, Zhao, & LeCun, 2015].

The Yahoo! Answers data set classes and their corresponding labels used for topic classification are:

1- Society & Culture	2- Science & Mathematics
3- Health	4- Education & Reference
5- Computers & Internet	6- Sports

7- Business & Finance	8- Entertainment & Music
9- Family & Relationships	10- Politics & Government

Table 6.10 shows some samples of the Yahoo! Answers data set. In our experiments, we concatenated the question title, question content and best answer fields on the training and testing processing.

Table 6.10. Yahoo! Answers data set samples.

Class	Question title/Question content/Best answer
1	what are the mining of 'jerban'or 'jarban'? i think this is a arabic or ibree word. You may have heard "juban" which means coward.
2	Why does Zebras have stripes? What is the purpose or those stripes? Who do they serve the Zebras in the wild life? this provides camouflage - predator vision is such that it is usually difficult for them to see complex patterns
3	Why is it desirable to have a 'grill' on your teeth in the hip-hop community? You know?...the gold caps and designs. They even have a whole song dedicated to this trend playing on Mtv. I think pearly whites are better vs putting any type of rare metal in your mouth.
4	What year did the stock market crash? That caused the so called GREAT DEPRESSION The stock market crashed in October 1929. This launched the "Great depression" Hope this helps!
5	Whos better, Yahoo or Google? Out of both email services which is better, Yahoo Mail or Gmail Though they are not comparable. Yahoo is the best.
6	what happen to Eddie Guerero? cause of his death He died of Heart failure do to his past use of drugs and of extensive exercising.
7	is it good habit to keep ur PC on when u r going somewhere for 5-10 minutes? i dont like to on it again n again Yeah you can simply lock it, if you are using window XP. You can use Ctrl+Alt+del or WindosButton+L

Table 6.10. Yahoo! Answers data set samples.

Class	Question title/Question content/Best answer
8	Do someone know what is the origin of Lenore "the cute little dead girl"? I saw in the cartoon network, and I am think is disturber. it's inspired by the poem "lenore" by edgar allen poe.
9	i have an interview in a new state. my husband wants me to move there alone for it. should i just cancel it? i don't want a divorce or to live without/away from my husband. i say go he's letting you know that its over read between the lines you will see it to.
10	Have married a Chinese National. What is the best visa option for her travel with me when I return to the USA Currently on overseas assignment in Beijing. Expected return date to US May 2006' Apply for a visa in Beijing

Table 6.11 presents some statistics of the Yahoo answers data set documents. This statistics were acquired during the text encoding process.

Table 6.11. Yahoo! Answers documents statistics.

Data Set	Sentences per sample				Words per sentence			
	Min	Max	Mean	Stdev	Min	Max	Mean	Stdev
Testing	1	129	7.05	6.63	1	765	15.04	13.05
Training	1	650	7.03	6.69	1	1816	15.06	13.39

6.1.6 Amazon Review Polarity

The original Amazon reviews data set consists of product reviews and information about the users who rated the products. The data span a period of 18 years, including ~35 million reviews up to March 2013 [McAuley & Leskovec, 2013].

The Amazon reviews polarity data set was constructed by taking reviews with scores 1 and 2 as negatives, and with scores 4 and 5 as positives. Samples with score 3 were ignored. In the Amazon reviews polarity data set, class 1 is the negative and class 2 is the positive.

Each class has 1,800,000 training samples and 200,000 testing samples [Zhang, Zhao, & LeCun, 2015].

Table 6.12 shows some samples of the Amazon reviews polarity data set. In our experiments, we concatenated the title and review text fields on the training and testing processing.

Table 6.12. Amazon reviews polarity data set samples.

Class	Review title	Review text
1	DVD menu select problems	I cannot scroll through a DVD menu that is set up vertically. The triangle keys will only select horizontally. So I cannot select anything on most DVD's besides play. No special features, no language select, nothing, just play.
2	The Scarlet Letter	I really enjoyed this book. It shows the judgmental tendencies in our human race and how one woman strove to live a life of service to others to gain redemption for her mistake. Can't go wrong with the classics.

Table 6.13 presents some statistics of the Amazon reviews polarity data set documents. This statistics were acquired during the text encoding process.

Table 6.13. Amazon reviews polarity documents statistics.

Data Set	Sentences per sample				Words per sentence			
	Min	Max	Mean	Stdev	Min	Max	Mean	Stdev
Testing	1	38	6.19	2.96	1	321	14.45	9.93
Training	1	81	6.20	2.97	1	384	14.45	9.92

6.2 Experiments

6.2.1 Methodology

To evaluate the accuracy of our model, we used as the baseline the results reported in Zhang et al. [2015].

In all experiments, we used the same values for the hyperparameters of our model.

Table 6.14 shows the names of the hyperparameters and the values used in the experiments.

Table 6.14. Values of the model hyperparameters used in the experiments.

Parameter	Value
model.minWordsSentence	1
model.updateLookupTable	true
train.epoches	10
train.batchSize	100
train.learningRate	1e-2
train.momentum	0.9
train.parametersDecay	1e-19
train.collectgarbage	100
train.validationSize	0
train.shuffle	true

The values of the hyperparameters were determined empirically training and testing the model using the first 200,000 samples of the Amazon reviews polarity data set and comparing the accuracy with the values reported in [Zhang, Zhao, & LeCun, 2015].

The text of the data sets samples were encoded with and without the use of WordNet synonyms. The vocabularies were constructed considering only the content of the training samples of each data set.

Table 6.15 shows the minimum vocable frequencies used to build the vocabularies for each data set.

Table 6.15. Minimum vocable frequencies used in experiments.

Data Set	Minimum Vocable Frequency
AG's News	10
DBPedia	12
Yelp Review Polarity	5
Yelp Review Full	5
Yahoo! Answers	12
Amazon Review Polarity	12

The value showed for the Amazon Review Polarity data set corresponds to the vocabulary built using 2,400,000 training samples. The minimum vocable frequency values were determined empirically.

The number of network parameters is affected by the vocabulary size that, in turn, is determined by the minimum vocable frequency parameter.

Table 6.16 shows the number of distinct vocables, the vocabulary size, generated using the minimum vocable frequency showed in Table 6.15, and the total number of model parameters for each data set.

Table 6.16. Vocabulary size and number of parameters of the model.

Data Set	Distinct Vocables	Vocabulary Size	Model Parameters
AG's News	100,039	21,028	8,552,404
DBPedia	718,985	63,739	21,370,714
Yelp Review Polarity	372,994	75,670	24,945,202
Yelp Review Full	414,403	82,080	26,869,705
Yahoo! Answers	1,450,085	104,775	33,680,710
Amazon Review Polarity	1,146,245	108,810	34,887,202

The values showed for the Amazon Review Polarity data set corresponds to the vocabulary built using 2,400,000 training samples.

In all experiments, we trained our model for 10 epochs. We did not use any validation data set. After each epoch, we tested the model using the data set testing samples. We reported the model accuracy for each data set as the best accuracy achieved among the 10 epochs.

We made an experiment with the purpose of evaluating the impact of the training size on the accuracy of our model. We chose the Amazon Review Polarity data set to make this experiment because of its huge size. We trained our model using 200,000 samples of the training data set and repeated the training adding up chunks of 200,000 samples up to the size of 2,400,000 training samples. The testing data set samples were used in the same proportion of the samples used in the training data set. We run the experiments encoding the text with and without the use of WordNet synonyms.

Table 6.17 shows the minimum vocable frequencies used to build the vocabularies, the size of the vocabularies and the total number of model parameters for each size of the training set used.

Table 6.17. Amazon reviews polarity training data set size experiment.

Training Samples	Minimum Frequency	Vocabulary Size	Model Parameters
200,000	5	48,405	16,765,702
400,000	5	70,070	23,265,202
600,000	10	57,419	19,469,902
800,000	10	67,131	22,383,502
1,000,000	10	75,571	24,915,502
1,200,000	10	83,300	27,234,202
1,400,000	12	81,382	26,658,802
1,600,000	12	87,373	28,456,102
1,800,000	12	92,994	30,142,402
2,000,000	12	98,535	31,804,702
2,200,000	12	103,751	33,369,502
2,400,000	12	108,810	34,887,202

The minimum vocable frequency values were determined by targeting the total number of parameters of the model to the 15~35 million interval.

In this experiment, in addition to comparing the accuracy achieved by our model with the results reported in [Zhang, Zhao, & LeCun, 2015], we also trained the state of the art model implemented by the authors, named Crepe[7], using 200K, 600K, 1,200K and 1,800K samples to evaluate the impact of the training size on the accuracy of their model.

6.2.2 Hardware

Table 6.18 shows the hardware specification for the computer used to run all the experiments. The Graphical Processing Unit (GPU) was donated by NVIDIA through the Academic Hardware Grant Program.

Table 6.18. Computer hardware specification.

Component	Manufacturer	Model
Motherboard	Gigabyte	GA-X99-UD3
CPU	Intel	Core i7-5820K @3.3GHz LGA 2011-v3
RAM	G.SKILL	Ripjaws 4 - DDR4 - F4-2800C15Q-32GRBB
Hard disk	Seagate	Barracuda ST2000DM001
GPU	NVIDIA	Tesla K40 Accelerator Board
Case	Nilco	NK211 EATX-TF
Power supply	EVGA	120-G2-1300-XR

[7] https://github.com/zhangxiangxiao/Crepe

Chapter 7

Results

In this chapter, we report and analyze the results of the experiments we described in Chapter 6. We initially report the results of the experiments carried out to evaluate the accuracy of our model, then we report the results of the experiments that we made to evaluate the impact of the training data set size on the accuracy of our model.

7.1 Accuracy

The accuracy of our model is compared with the models described in Zhang et al. [2015]. In their paper, the authors implemented 22 models divided into 4 classes. The first class encompasses 5 traditional models that use a handcrafted feature extractor and linear classifiers. In the second class, the authors implemented the common vanilla architecture of LSTM using Word2Vec as the initial representation for the words. The third class is composed by 8 variations of the word based Convolutional Neural Network (CNN) model. The architectures of this class models are the most comparable to our model. This class is subdivided into 2 classes based on the type of the initial representation used for the words. The names of the models make reference to these subclasses. The term “Lk.” stands for lookup table, which means that the model uses one-hot as the initial word representations. The models of the other subclass use the Word2Vec as the initial word representations. The models, whose names have the term “Th.”, make use of the thesaurus for data augmentation. The terms “Lg.” and “Sm.” designate the size of the upper fully connected layer and correspond respectively to the 2,048 and 1,024 sizes. The fourth class is composed by 8 variations of the character based CNN model. The models labeled “Full” are those that distinguish between lower and upper letters.

Table 7.1 shows a summary of the results. The numbers are in percentage. The best accuracy for each data set is printed in bold face. The table is subdivided into model classes.

Table 7.1. Accuracy results summary.

Model	Amazon Polarity	Yelp Polarity	Yelp Full	DBPedia	AG’s News	Yahoo! Answers
BoW	90.40	92.24	57.99	96.61	88.81	68.89
BoW TFIDF	91.00	93.66	59.86	97.37	89.64	71.04
ngrams	92.02	95.64	56.26	98.63	92.04	68.47
ngrams TFIDF	91.54	95.44	54.80	98.69	92.36	68.51
Bag-of-means	81.61	87.33	52.54	90.45	83.09	60.55
LSTM	93.90	94.74	58.17	98.55	86.06	70.84
Lg. w2v Conv.	94.12	95.40	59.84	98.58	90.08	68.03
Sm. w2v Conv.	94.00	94.44	57.87	98.29	88.65	68.50
Lg. w2v Conv. Th.	94.20	95.37	60.42	98.63	90.09	68.77
Sm. w2v Conv. Th.	94.37	94.64	58.91	98.47	89.12	70.14
Lg. Lk. Conv.	94.16	95.11	59.48	98.28	91.45	70.94
Sm. Lk. Conv.	94.15	94.46	58.59	98.15	89.13	69.98
Lg. Lk. Conv. Th.	94.48	94.97	59.48	98.42	91.07	71.16
Sm. Lk. Conv. Th.	94.49	94.63	58.83	98.23	90.88	71.08
Lg. Full Conv.	94.22	94.75	61.60	98.34	90.15	70.10
Sm. Full Conv.	94.22	94.33	61.18	98.11	88.41	69.99
Lg. Full Conv. Th.	94.49	95.12	61.96	98.45	90.49	70.42
Sm. Full Conv. Th.	94.34	94.58	62.05	98.31	89.11	70.10
Lg. Conv.	94.49	94.11	60.38	98.27	87.18	70.45
Sm. Conv.	94.50	93.47	59.16	98.02	84.35	70.16
Lg. Conv. Th.	95.07	94.18	60.70	98.40	86.61	71.20
Sm. Conv. Th.	94.33	93.51	59.84	98.15	85.20	70.16
Deep NLP	95.32	96.05	64.76	98.66	92.26	74.02
Deep NLP WordNet	**95.65**	**96.32**	**65.62**	**98.82**	**92.61**	**74.53**

The Deep NLP WordNet model surpasses all other models in all tasks. The Deep NLP model surpasses all models of the other classes in all tasks with the exception of the DBPedia and AG's News data sets, in which it is surpassed by the ngrams TFIDF model.

This result can be explained by two facts. The first one is the fact that the amount of training samples per class on both data sets are the smallest among all data sets used.

The second fact is that the sample documents of both data sets have less than three sentences on average. Because of our model explicitly creates intermediate representations for the sentences, texts with small number of sentences have a poorest semantic context.

The worst performance of our model on these two data sets can be justified by the linguistic theory called Poverty of the Stimulus (POS) [Chomsky, 1980].

Table 7.2 shows a comparison between the number of training samples per class, the mean number of sentences per sample and number of model parameters of the data sets.

Table 7.2. Training data sets comparison.

Data Set	Training Samples per Class	Mean Number of Sentences per Sample	Model Parameters
AG's News	30,000	2.64	8,552,404
DBPedia	40,000	2.87	21,370,714
Yelp Review Polarity	280,000	9.83	24,945,202
Yelp Review Full	130,000	9.89	26,869,705
Yahoo! Answers	140,000	7.03	33,680,710
Amazon Review Polarity	1,200,000	6.20	34,887,202

This adverse scenario helps to show why the use of the WordNet synonyms provides robustness to our model making it to surpass the accuracy of all other models. Table 7.3 shows the number of vocables in the vocabulary whose initial Word2Vec representations were replaced and the number of words in the samples text replaced by WordNet synonyms in the AG's News and DBPedia Ontology data sets.

Table 7.3. Vocabulary generation and text encoding statistics using WordNet.

Data Set	Vocables Replaced	Words Replaced in Training Samples	Words Replaced in Testing Samples
AG's News	50	44,594	3,166
DBPedia	239	107,477	14,651

7.2 Training Size

Figure 7.1 summarizes the results of the experiment that we made to evaluate the impact of the Amazon Review Polarity data set size on the accuracy of our model.

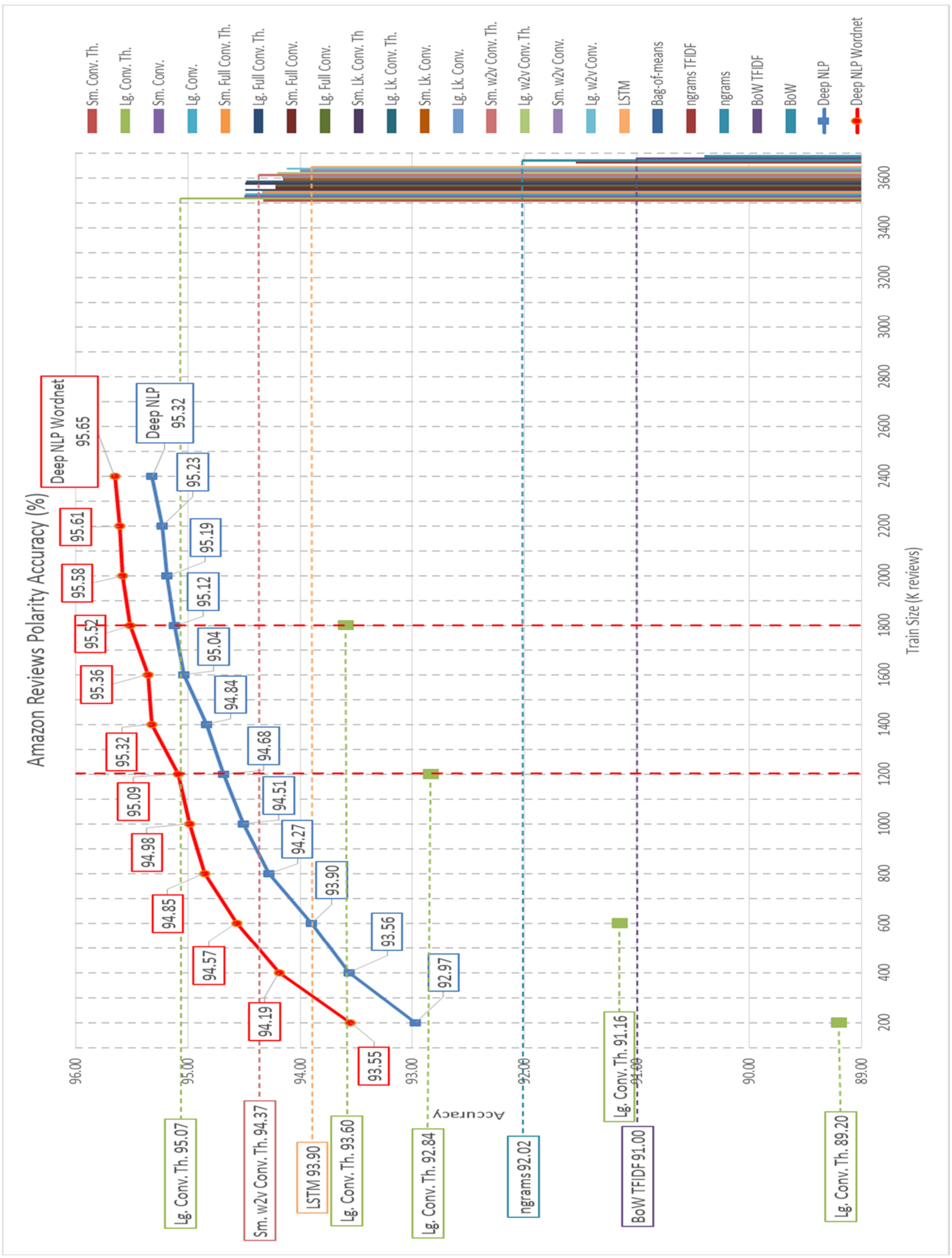

Figure 7.1. Experiments with the size of the Amazon Review Polarity data set.

This chart highlights the best accuracy achieve by each class of the models reported in [Zhang, Zhao, & LeCun, 2015]. Our model without the use of WordNet synonyms surpasses the state of the art model accuracy using half of the training samples. Our model using the WordNet synonyms surpasses the state of the art model accuracy using one third of the training samples. The accuracies achieved by the state of the art character based CNN model using these volumes of training are significantly lowers than the ones achieved by our models.

This chart shows that the accuracy of our models consistently increases as the size of the training data set grows. It is also clear that the model that make use of the WordNet synonyms consistently achieved better accuracies than the ones achieved by the model that do not use the synonyms.

The results presented on this chart suggest that the use of the WordNet synonyms not only decreases the demand for training samples, but also improves the accuracy of our model.

Table 7.4 shows the time spent by the Deep NLP and Crepe models in the training of ten epochs for each Amazon Review Polarity data set size used in this experiment. The time is presented in hours.

Table 7.4 Training times for the Amazon Review Polarity data set sizes.

Data Set Size	Deep NLP	Crepe
200,000	5	110
400,000	10	-
600,000	15	120
800,000	20	-
1,000,000	25	-
1,200,000	30	130
1,400,000	35	-
1,600,000	40	-
1,800,000	45	140
2,000,000	50	-
2,200,000	55	-
2,400,000	60	-

Chapter 8

Conclusions

In this work, we proposed a robust deep learning CNN model for text categorization tasks. The model is robust in the sense that it can achieve the state of the art accuracy on different text categorization tasks without the need to adjust the model hyperparameters for each task.

To achieve the robustness, we incorporated into the model many deep learning concepts and techniques. The concept of compositionality was used in the design of the deep CNN architecture to induce the creation of a hierarchical representation for the text.

We employed the concept of prior knowledge when we used the word embeddings and semantic synonyms in the text encoding process. We used the concept of specialization when we allowed the initial word representations to be adjusted in the training process, considering them as parameters of the network.

The concept of depth was used in the design of the feature extractor and label predictor components of the network. The parameter sharing and sparse connectivity techniques were used in the convolutional layers. The overfitting was tackled using the dropout technique during the training process.

To accelerate the network convergence, we used the mini-batch momentum version of the SGD update algorithm and we randomly shuffled the training data set before each epoch. To accelerate the training, making viable the use of large datasets, we implemented our model using a language and framework that make effective use of the massively parallel processing power of the GPUs.

We evaluated our model comparing its accuracy against the results reported by some traditional and deep learning models using six large-scale data sets. The results showed that our model outperformed the accuracy of the state of the art models in different text categorization tasks. The results also showed that the use of word embeddings and semantic synonyms helped to generalize the representations learned by the model increasing its accuracy.

The main contribution of our work is to show that, even when a large amount of training samples is available, the use of word embeddings is important to achieve a higher accuracy using less training data, and consequentially in a shorter processing time.

Another contribution comes from the fact that the size of the input text is not limited by the network architecture of our model. The number of words and sentences in the input text is limited only by the amount of GPU's memory. In similar works, the size of the input text is limited by the number of characters, words or sentences.

Another contribution comes from the implementation of our model that makes an efficient use of the massively parallel processing power of the GPU, which makes it possible to train huge data sets in a shorter processing time.

Chapter 9

Future Work

The vocabulary size has a huge impact on the number of network parameters of our model. To limit the number of parameters, we only include in the vocabulary the vocables that have a minimum frequency in the training data set. If a vocable occurs only in a few training samples, it is difficult to the model to learn a good representation for it. To overcome this limitation, as a future work, we propose to employ a different frequency measure that also takes into account the number of training samples that the vocable occurs.

Since our model does not limit the size of the input text, would be interesting to evaluate its performance on data sets that have larger documents.

Deconvolution is a visualization technique used to show the patterns learned by each layer of a deep CNN in computer vision applications. As the design of the deep CNN architecture of our model induces the creation of a hierarchical representation for the input text, in a future work, we propose the use of the deconvolution technique to discover the words and sentences of the input text that most contributed to the class predicted by the model.

Transfer learning is the process of learning new tasks using the experience gained by solving predecessor problems that are somewhat similar. In the context of supervised learning, transfer learning can be used to train a model using a data set and use the trained model to process the samples of a similar data set. In a future work, we intend to make experiments using our model trained on a given data set and evaluate its accuracy on testing samples of other data sets.

Another interesting work would be to evaluate the impact on the accuracy of our model caused by the use of word embeddings obtained from factual texts of a specific domain. For example, we can train an unsupervised language model using documents having health, drugs and other factual contents in the field of medicine. Then we can train our model using these word embeddings on a data set collected from patient's posts in health forums and evaluate the accuracy of the model on the prediction of the rate given by the patients to a drug.

The degree of agreement among humans is also known as human concordance. In experiments, this degree is measured using some coefficients and its quality is measure using inter-rater reliability techniques. There are some works saying that the rate of human concordance is between 70% and 79%, and that a good accuracy for sentiment analysis tools is 70% [Gwet, 2014]. Our model achieved an accuracy higher than 70%, in most of the experiments, using data sets whose documents were written by humans. How this can be explained? One hypothesis is that our model is able to learn the discourse used by the group of people who wrote the content of a given data set. Although deep learning models are inspired by the working principles of the human brain, they do not learn to reason. All they know about the world comes from the training samples presented to them. Differently from deep learning models, human beings reason about something using past experiences acquired in different contexts. In general, a data set is made up by documents published by a group of people expressing their experiences and opinions about some subject. Although people of the same group have different experiences and opinions, they must agree about the discourse used to express them. This can explain why our model has a better performance in some text categorization tasks that surpasses the human concordance.

To verify this hypothesis, in a future work, we intend to train our model using a data set containing product reviews written by specialists and evaluate the accuracy of the trained model on a data set containing product reviews written by lay people.

Bibliography

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning, 2*(1), 1-127.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy Layer-Wise Training of Deep Networks. Em B. Schölkopf, J. C. Platt, & T. Hofmann (Ed.), *NIPS* (pp. 153-160). MIT Press.

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python.* Beijing; Cambridge [Mass.]: O'Reilly.

Bottou, L., Collobert, R., Weston, J., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (November de 2011). Natural Language Processing (Almost) from Scratch. *J. Mach. Learn. Res., 999888*, 2493-2537.

Boureau, Y.-L., Ponce, J., & LeCun, Y. (2010). A Theoretical Analysis of Feature Pooling in Visual Recognition. Em J. Fürnkranz, & T. Joachims (Ed.), *ICML* (pp. 111-118). Omnipress.

Chomsky, N. (1980). *Rules and Representations.* New York: Columbia Univeristy Press.

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: deep neural networks with multitask learning. Em W. W. Cohen, A. McCallum, & S. T. Roweis (Ed.), *ICML. 307*, pp. 160-167. ACM.

de Oliveira Jr., R. L., Veloso, A., Pereira, A. M., Jr., W. M., Ferreira, R., & Parthasarathy, S. (2014). Economically-efficient sentiment stream analysis. Em S. Geva, A. Trotman, P. Bruza, C. L. Clarke, & K. Järvelin (Ed.), *SIGIR* (pp. 637-646). ACM.

Denil, M., Demiraj, A., Kalchbrenner, N., Blunsom, P., & de Freitas, N. (2014). Modelling, Visualising and Summarising Documents with a Single Convolutional Neural Network. *CoRR, abs/1406.3830*.

dos Santos, C. N., & Gatti, M. (2014). Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. Em J. Hajic, & J. Tsujii (Ed.), *COLING* (pp. 69-78). ACL.

Golden, M. R. (1996). *Mathematical Methods for Neural Network Analysis and Design.* Cambridge, MA: MIT Press.

Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., & Shet, V. D. (2014). Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. *International Conference on Learning Representations.*

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning* (Book in preparation ed.). MIT Press.

Gülçehre, Ç., & Bengio, Y. (2013). Knowledge Matters: Importance of Prior Information for Optimization. *CoRR, abs/1301.4083.*

Gwet, K. L. (2014). *Handbook of Inter-Rater Reliability: The Definitive Guide to Measuring the Extent of Agreement Among Raters* (4th ed.). Advanced Analytics, LLC.

Hawkins, J., & Blakeslee, S. (2004). *On Intelligence.* Henry Holt.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation.* Prentice Hall.

Hecht-Nielsen, R. (1990). *Neurocomputing.* Redwood City, CA: Addison-Wesley.

Hinton, G. E., Krizhevsky, A., & Wang, S. D. (2011). Transforming Auto-Encoders. Em T. Honkela, W. Duch, M. A. Girolami, & S. Kaski (Ed.), *ICANN (1). 6791*, pp. 44-51. Springer.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation, 18*, 1527-1554.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR, abs/1207.0580.*

Hubel, D. H., & Wiesel, T. N. (1959). Receptive Fields of Single Neurons in the Cat's Striate Cortex. *Journal of Physiology, 148*, 574-591.

Ierusalimschy, R. (2006). *Programming in Lua (2. ed.).* Lua.org.

Johnson, R., & Zhang, T. (2015). Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. Em R. Mihalcea, J. Y. Chai, & A. Sarkar (Ed.), *HLT-NAACL* (pp. 103-112). The Association for Computational Linguistics.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (June de 2014). A Convolutional Neural Network for Modelling Sentences. *Proceedings of the 52nd Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers) (pp. 655-665). Baltimore, Maryland: Association for Computational Linguistics.

Kandel, E., Schwartz, J., & Jessel, T. (2000). *Principles of neural science* (fourth ed.). McGraw-Hill.

Kavukcuoglu, K., Farabet, & Collobert, R. (2011). Torch7: A Matlab-like Environment for Machine Learning. *BigLearn, NIPS Workshop, EPFL-CONF-192376.*

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Em A. Moschitti, B. Pang, & W. Daelemans (Ed.), *EMNLP* (pp. 1746-1751). ACL.

Kirk, D. B., & Hwu, W.-m. W. (2010). *Programming Massively Parallel Processors: A Hands-on Approach.* Burlington, MA: Morgan Kaufmann Publishers.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, (pp. 1097-1105).

Le, Q. V., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. *ICML. 32*, pp. 1188-1196. JMLR.org.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE, 86*, pp. 2278-2324.

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient BackProp. Em G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural Networks: Tricks of the Trade (2nd ed.)* (Vol. 7700, pp. 9-48). Springer.

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., . . . Bizer, C. (2015). DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web, 6*(2), 167-195.

Lev, G., Klein, B., & Wolf, L. (2015). In Defense of Word Embedding for Generic Text Representation. Em C. Biemann, S. Handschuh, A. Freitas, F. Meziane, & E. Métais (Ed.), *NLDB. 9103*, pp. 35-50. Springer.

Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* Cambridge, Massachusetts: The MIT Press.

McAuley, J. J., & Leskovec, J. (2013). Hidden factors and hidden topics: understanding rating dimensions with review text. Em Q. Y. 0001, I. King, Q. Li, P. Pu, & G. Karypis (Ed.), *RecSys* (pp. 165-172). ACM.

McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysic*(5), 115-133.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Miller, G. (1995). WordNet {A} Lexical Database for {E}nglish. *Communications of ACM, 38*(11), 39-41.

Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. Em J. Fürnkranz, & T. Joachims (Ed.), *ICML* (pp. 807-814). Omnipress.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *ICML (3). 28*, pp. 1310-1318. JMLR.org.

Rosenblatt, F. (1962). *Principles of Neurodynamics.* Spartan, New York.

Rumelhart, D. E., Hinton, G. E., & Wilson, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533-536.

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *CoRR, abs/1312.6034*.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., & Potts, C. (2013). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 1631-1642). Seattle: Association for Computational Linguistics.

Sutskever, I., Martens, J., Dahl, G. E., & Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. *ICML (3). 28*, pp. 1139-1147. JMLR.org.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2014). Going Deeper with Convolutions. *CoRR, abs/1409.4842*.

Taylor, G. W., Fergus, R., & Zeiler, M. D. (2011). Adaptive deconvolutional networks for mid and high level feature learning. Em D. N. Metaxas, L. Quan, A. Sanfeliu, & L. J. Gool (Ed.), *ICCV* (pp. 2018-2025). IEEE Computer Society.

Turian, J. P., Ratinov, L.-A., & Bengio, Y. (2010). Word Representations: A Simple and General Method for Semi-Supervised Learning. Em J. Hajic, S. Carberry, & S. Clark (Ed.), *ACL* (pp. 384-394). The Association for Computer Linguistics.

Veloso, A., Jr., W. M., Cristo, M., Gonçalves, M. A., & Zaki, M. J. (2006). Multi-evidence, multi-criteria, lazy associative document classification. Em P. S. Yu, V. J. Tsotras, E. A. Fox, & B. L. 0001 (Ed.), *CIKM* (pp. 218-227). ACM.

Vincent, P., Bengio, Y., & Ducharme, R. (2000). A Neural Probabilistic Language Model. Em T. K. Leen, T. G. Dietterich, & V. Tresp (Ed.), *NIPS* (pp. 932-938). MIT Press.

Waibel, A., Hanazawa, T., Hinton, G. E., Shikano, K., & Lang, K. (1989). Phoneme Recognition Using Time-Delay Neural Networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 37*, 328-339.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent Neural Network Regularization. *CoRR, abs/1409.2329*.

Zeiler, M. D., & Fergus, R. (2013). Visualizing and Understanding Convolutional Networks. *CoRR, abs/1311.2901*.

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level Convolutional Networks for Text Classification. *CoRR, abs/1509.01626*.

Zweig, G., Mikolov, T., & tau Yih, W. (2013). Linguistic Regularities in Continuous Space Word Representations. Em L. Vanderwende, H. D. III, & K. Kirchhoff (Ed.), *HLT-NAACL* (pp. 746-751). The Association for Computational Linguistics.