UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Elaine Cristina Resende Cândido

Um estudo comparativo de redes neurais profundas para classificação automática de texto

Belo Horizonte
2020

Elaine Cristina Resende Cândido

Um estudo comparativo de redes neurais profundas para classificação automática de texto

Versão Final

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador(a): Marcos André Gonçalves

Belo Horizonte

2020

Elaine Cristina Resende Cândido

A comparative study of deep neural networks for automatic text classification

Final Version

Thesis presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais Departamento de Ciência da Computação. in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Marcos André Gonçalves

Belo Horizonte

2020

Resende Cândido, Elaine Cristina

D1234p A comparative study of deep neural networks for automatic text classification / Elaine Cristina Resende Cândido. — Belo Horizonte, 2020
xiv, 81 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais Departamento de Ciência da Computação.
Orientador(a): Marcos André Gonçalves

1. Computação — Teses. 2. Banco de Dados — Teses. 3. Machine Learning — Teses. I. Orientador. II. Título.

CDU 519.6*82.10

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz - CRB 6ª Região nº 819

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

FOLHA DE APROVAÇÃO

A Comparative Study of Deep Neural Networks for Automatic Text Classification

ELAINE CRISTINA RESENDE CÂNDIDO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. MARCOS ANDRÉ GONÇALVES - Orientador
Departamento de Ciência da Computação - UFMG

PROFA. JUSSARA MARQUES DE ALMEIDA GONÇALVES
Departamento de Ciência da Computação - UFMG

PROF. LEONARDO CHAVES DUTRA DA ROCHA
Departamento de Ciência da Computação - UFSJ

Belo Horizonte, 14 de Fevereiro de 2020.

I dedicate this dissertion first to our Almighty God who has given me the gift of life and the free will to persecute my goals and dreams and acquire wisdom; and second to my lovely husband and family who have supported me during this journey.

Acknowledgments

First of all I would like to express my gratitude to my advisor Marcos André Gonçalves who believed in my capacity and trusted in my work.

Second, I would like to thank all my colleagues from the databases laboratory, specially those who have somehow contributed to this work.

Also, I would like to warmly thank my husband, Cristiano, for all understanding and love.

Last, but not least, my mother Elisabet, my siblings Petrônio, Gizele and Patrick and my grandmother Desi and aunt Eliete for all support and incentive along my path of studies.

"In God we trust, all others must bring data"
(W. Eduards Deming)

Resumo

Classificação Automática de Texto (ATC), também conhecida como Classificação de Documentos ou Categorização de Texto, é uma tarefa desafiante em processamento de linguagem natural que envolve classificar textos em categorias baseando-se nas propriedades e atributos textuais de cada documento. Recentemente, metodologias de aprendizado profundo têm transformado ATC através de novas estratégias de classificação e criação de novas técnicas de vetorização de palavras, que nada mais é do que a representação de palavras em forma de vetor numérico. Contudo, a maioria dos métodos de redes neurais mostram diversos problemas, incluindo a falta de comparações rigorosas de *benchmarks* com outros algoritmos tradicionais mais bem estabelecidos (onde a avaliação é feita partir de conjuntos de dados e procedimentos de preprocessamento padronizados). Nesta dissertação, avaliamos diversas métodos, incluindo diferentes arquiteturas de redes neurais como redes neurais de convolução, de atenção, e transformadores bidirecionais e as comparamos com um dos algoritmos de aprendizado de máquina mais tradicionais, denominado Máquinas de Vetor de Suporte (mais conhecido como SVM). Nossos resultados experimentais indicam que, para conjuntos de dados menores, o método de referência mais tradicional e barata (TFIDF com máquinas de vetor de suporte) está entre os melhores desempenhos no geral, superando significativamente abordagens neurais muito mais sofisticadas e caras quando o custo-benefício é considerado. Nos conjuntos de dados maiores, a abordagem neural mais recente que utiliza transformadores, denominada BERT, ultrapassa alguns métodos com significância estatística.

Palavras-chave: Classificação de texto, aprendizado profundo, SVM.

Abstract

Automatic Text Classification (ATC), also known as Document Classification or Text Categorization, is a challenging Natural Language Preprocessing task that involves classifying texts into various categories based on inherent properties or attributes of each text document. Recently, deep learning methodologies have been transforming ATC through new strategies of classification and creation of new word embedding approaches. However, most neural networks methods show several issues, including the lack of rigorous benchmark comparisons with other more well established traditional algorithms (where the evaluation is based on standard datasets and pre-processing procedures). In this master thesis we evaluate several methodologies including different neural networks architectures such as Convolution Neural Networks, Attention Networks and Bidirectional Transformers and compare them to one of the most traditional known machine learning algorithms called Support Vector Machines. Our experimental results, indicate that for the smaller datasets, the simplest and cheaper baseline (TFIDF with Support Vector Machines) is among the best overall performers, clearly beating much more sophisticated and costly neural approaches when a trade-off effectiveness-cost is considered. In the larger datasets, the recently proposed neural approaches based on Transformers do excel, beating (tied) other neural architectures with statistical significance.

Palavras-chave: Text Classification, Deep Learning, SVM.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Getting meaningful insights from textual data in order to understand text is one of the first, yet most essential tasks in Natural Language Processing (NLP). Automatic Text Classification (ATC), also known as Document Classification or Text Categorization, is a challenging NLP task that involves categorizing texts based on inherent properties or features of each text document. Several methodologies and algorithms have been used to classify, clusterize and extract information from text, such as Nearest Neighbors classifiers, Decision Trees, Latent Semantic Analysis, probabilistic classifiers and ensembles of classifiers Crammer et al. [2012]; Manne et al. [2012]; Joachims [1998]; Allahyari et al. [2017]; Campos et al. [2017a]; Salles et al. [2015a].

Advances in computer architectures (e.g., GPUs) and in processing power have boosted some Artificial Intelligence architectures, among them, Deep Neural Networks (DNNs)[1]. DNNs are able to learn hierarchical features through multiple stacked representation layers in an automatic fashion, allowing NNs solutions to become the state-of-the-art (SOTA) in many applications, such as image processing Guo et al. [2016]; Pavlakos et al. [2017], object recognition Ribeiro et al. [2018] and speech recognition Chiu et al. [2017]; Upadhyay et al. [2018]; Kannan et al. [2018].

In this context, there is a large interest in developing NNs strategies to extract information from text, classify documents, predict next words, etc Conneau et al. [2017]; Lin [2019]; Kim [2014].

[1]We use the terms Neural Networks (NNs), Deep Learning, Deep Neural Networks (DNNs) interchangeably in this dissertation.

1.1 Motivation

Neural Networks (NNs) have been transforming NLP research in recent years. They started being more studied in different communities, such as data mining and information retrieval, and in a vast number of applications, such as medical diagnosis, document organization, sentiment analysis and web searching [Aggarwal and Zhai, 2012; Kakade et al., 2017]. One of the pioneer works in NLP, well known as word2vec Mikolov et al. [2013a], which is a method based in neural networks that can be trained and transform text input to a representation of numerical vector for each word in the text, has opened entirely new venues for NLP research Young et al. [2018].

It has been demonstrated that Neural Networks can approximate any type of learning function K. et al. [1989], since they are a set of layers multiplications, in the most short form, that can be know as a regression (more about how a neural network works in the Section 2.2). However, the costs to do so are extremely high due to the exponential number of configurations (dozens of hyperparameters), architectures and parameters (literally millions of parameters!) that need to be tuned Lin et al. [2017]. Indeed, some industries estimate the cost of training state-of-the-art deep learning models, even pre-trained ones (such as BERT Devlin et al. [2018] and XLNETYang et al. [2019]), in hundreds of thousands of dollars, besides demanding very powerful and costly computational architectures[2].

Moreover, the adoption of NNs in the text domain is still less successful than in other ones Hassan and Mahmood [2017a]; Kim [2014]; Xiao and Cho [2016a] such as computer vision Feng et al. [2019]. In fact, in some text-based applications, as automatic text classification (ATC) and sentiment analysis, despite some interesting proposals Zhang et al. [2015a]; Kim [2014]; Hassan and Mahmood [2017b]; Conneau et al. [2017]; Devlin et al. [2018], it is not clear or well established yet if deep neural networks configure the SOTA. There are several issues, including the lack of rigorous benchmark comparisons with other more well established SOTA algorithms (e.g., Support Vector Machines[3] Campos et al. [2017a]; Salles et al. [2015b]), with standardized datasets and pre-processing procedures. Also, standardized scientific procedures (e.g., folded cross-validation, statistical significance tests with corrections (e.g., Bonferroni Hochberg [1988] or Friedman Friedman [1940, 1937]) are not the rule in the comparisons of existing and novel deep learning proposals. These problems have been confirmed in recent, more rigorous, comparisons of NNs against standard baselines in

[2]https://syncedreview.com/2019/06/27/the-staggering-cost-of-training-sota-ai-models/

[3]Several comparisons are made with the less effective, but faster, Logistic Regression or KNN algorithms

areas such as recommendation systems Dacrema et al. [2019]; Ludewig et al. [2019], where most NNs did not perform better than the simplest baselines in many situations.

In fact, typically the comparisons of novel NN proposals with previous ones in the literature are very shallow, reporting minimal gains with no rigorous (statistical) evaluation of the results, usually using a single metric such as accuracy, which may not even been adequate for some scenarios (e.g., datasets with skewed class distributions). In such cases, it cannot be ruled out that newest results reported in the literature over the last few years are simply statistical ties, with no real advances.

Another pitfall is the absence of the original implementations of the reported methods, which makes it hard to perform a fair comparison. Even when there is some code available, it is so hard to run or poorly documented, that is almost impossible to run it. When available, and possible to run, the actual parameterization of the method is extremely obscure. In many cases it is performed using a process of *trial-and-error*, commonly described in an anecdotal way, making it again impossible to replicate.

Furthermore, some of these methods usually exploit an enormous amount of proprietary data to configure some portions of the network, which are usually only available to huge enterprises such as Google and Amazon and not to the Academy.

Finally, there is a myriad of different deep neural architectures (convolutional NNs, attention NNs, recurrent neural networks Medsker and Jain [2001]; Liu et al. [2017]; Song et al. [2019]; Devlin et al. [2018], etc.) that can be used for ATC. Thus, it is currently not clear which of these alternatives are more suitable for each specific task.

This is the current context in which this dissertation is inserted.

1.2 Our Proposal

Our main goal in this dissertation is to design and run extensive experiments aiming at validating and comparing the effectiveness and efficiency of different neural networks architectures such as Convolution Neural Networks (CNN) and Very Deep Concolutional Neural Networks (VDCNN), Hierarchical Attention Networks (HAN) and Bidirectional Transformers (BERT). We do compare them to one of the most effective SOTA and traditional classifiers: Support Vector Machines, most known as SVM, in the text classification realm. We follow a standardized and controlled experimental set up. This is not an easy task as the training process of the methods may be time-consuming and the results tend to be vastly distinct with regard to different parameters and datasets.

This general objective can be narrowed down into three specific goals, driven by the following research questions:

- *Research Question 1 (RQ1): Can we model and achieve state-of-the-art performance with recent deep learning architectures? Even on small datasets?*
  Most of the published works on text classification that use deep learning architectures do not perform standard statistical comparison procedures, or training/testing splitting set up for model construction. We build a controlled experimental setup aiming at comparing different methods and evaluating how recent works perform in a set of standardized datasets following proper statistical procedures.

- *Research Question 2 (RQ2): Are previous results statistically significant?*
  Our hypothesis follow similar observations and findings from Dacrema et al. [2019], Ludewig et al. [2019] and Lin [2019], where they evidence that new deep learning methods are rarely better in text classification tasks, that they might perform better in some cases, but by a small margin.

  Aiming at answering RQ1 and RQ2, we have worked in the standardization of all datasets, working in a setup with k-fold cross-validation, using the same set of parameters as the authors of each published deep learning architecture. Our evaluation procedure measures the performance of the models using f1-score (micro and macro averages) and log-loss, comparing the results with statistical t-test (with Bonferroni corrections), Friedman ranking and bootstrapping confidence interval.

- *Research Question 3 (RQ3): How efficient are deep learning models?*
  Previous work Conneau et al. [2017]; Yang et al. [2016]; Xiao and Cho [2016a], do not measure or describe any details of the time their methods take for training/testing. The process of tuning neural networks consumes a very long time. That is usually the reason why tuning of parameters is not used. The process of *trial-and-error* is used due to this high cost. To the best of our knowledge, there is no study mentioning and/or comparing time of tuning or training/testing of deep learning models. Towards answering RQ3, we analyze the total time for training, tuning and testing each one of the classifiers in study.

1.3 Main Contributions

Towards achieving the proposed goals, we have accomplished the following contributions:

A comprehensive experimental evaluation which explores the trade-offs between efficiency and classification effectiveness, where we compare the performance and time taken for fitting and testing different methods.

An evaluation of performance of deep learning models using other metrics besides accuracy or error (100 - accuracy)%. More specifically, as we are aware of, we are the first work that evaluates deep learning architectures considering F1-score with macro and micro averages and log loss for measuring the confidence of the model. We also provide extensive statistical analysis of our findings using well-known statistical tests in order to fairly compare all the methods.

In summary, as our experimental evaluation shows, traditional methods such as Support Vector Machines are very powerful and effective in most of the datasets, mainly in the small ones. Moreover, it is more viable than any other method we have compared in this dissertation due to its very small time consumption and interpretation.

1.4 Outline

The rest of this dissertation is organized as follows. Chapter 2 discusses the background, highlighting main breakthroughs in the deep learning area and most recent works in text classification, plus, it also explains some important concepts around the methods used in this dissertation. Chapter 3 describes our experimental methodology, presenting characteristics of our datasets, methods and experimental setup. Chapter 4 presents and discusses our results and finally, Chapter 5 presents the conclusions and directions for future work.

Chapter 2

Background and Related Work

In this Chapter we introduce a timeline of the history of the Neural Networks with very important breakthroughs concerning deep learning architectures and highlight some related work. Moreover, we explain some important concepts related to the architectures explored in this master thesis.

2.1 A Brief Timeline History

We briefly describe how deep learning techniques evolved over the years. Figure 2.1 illustrates a timeline of the evolution.

Neurons were mentioned by Warren McCulloch and Walter Pitts in their paper titled “A logical calculus of the ideas immanent in nervous activity”, in which the first mathematical model of a neuron was built with a electrical circuit McCulloch and Pitts [1943]. During the 1950’s, Nathanniel Rochester from IBM research lab led the first effort to simulate a neural network. In 1958 Frank Rosenblatt created the oldest Neural Network still in use today: the perceptron Rosenblatt [1958].

One year later Stanford researchers Bernard Widrow and Marcian Hoff developed the first application of neural networks (Adaline and Madaline) applied to phone lines, eliminating echoes Attoh-Okine and Ayyub [2005]. After a long recession, Convolutional Neural Networks emerged with the work of Fukushima Fukushima [1980] in the 1980s, who created Neucognitron, a Neural Network that recognizes visual patterns. At that same time the first type of Recurrent Neural Network (RNN), named as Hopfield Networks Hopfield [1982] surged. Later, in 1997 Jürgen Schmidhuber and Sepp Hochreiter proposed the Long Short Term Memory Neural Networks (LSTM). LSTMs were able to "remember" information for a longer period of time. In 1998, the stochastic

descent algorithm was born with the work of Yann LeCun LeCun et al. [1998], which became a very popular algorithm in deep learning.

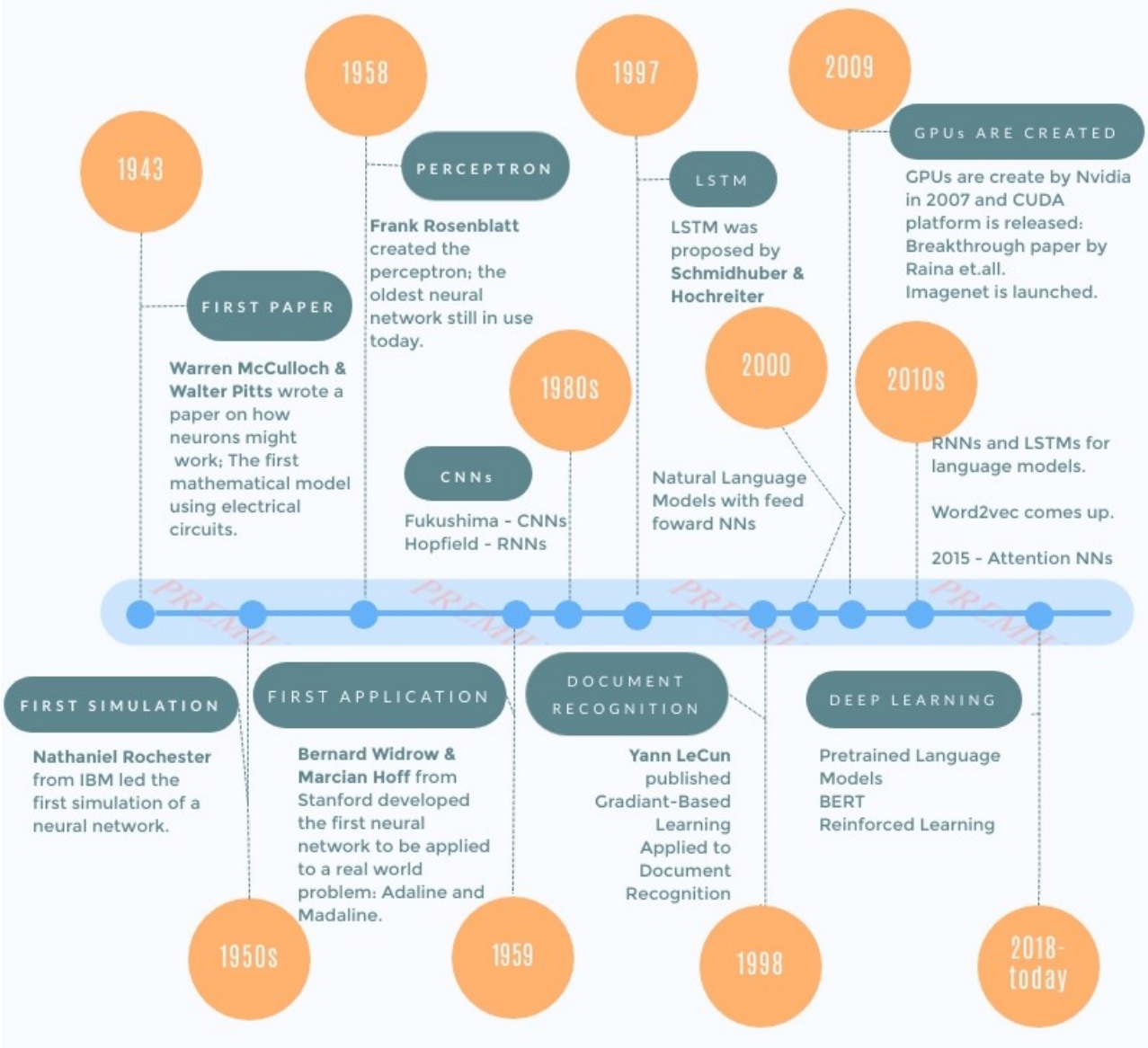


Figure 2.1. Timeline of Neural Networks

The development of neural networks in the visual recognition realm started to bring insights to text-related areas in the beginning of 2000, mainly language models that compute the probability of a word w_t given its $n-1$ previous words. By that time, language models evolved from n-gram based to the first classic neural network language model proposed by Bengio et al. [2003], which learned distributed representations for words. The architecture consists of an one-hidden layer feed-forward neural network that predicts the next word in a sequence. This was also one of the first works to

introduce the term word embeddings (a real-valued word feature vector in *IR*) and train them in a neural network.

The creation of GPUs and release of CUDA platform near 2010 was a breakthrough in the processing of big datasets and consequently a huge step towards the training of neural networks. After that, language models evolved to recurrent neural networks and LSTMs Mikolov et al. [2010]; Graves [2013] and the main innovation was proposed in 2013 by Mikolov et al. [2013c] with the creation of the most well known methodology: Word2vec.

Word2vec is a two-layer neural network that processes text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus, also called word embeddings. Specifically, Mikolov et.al proposed the continuous bag-of-words (CBOW), which uses context to predict a target word and skip-gram, which uses a word to predict a target context Mikolov et al. [2013b]. CBOW and skip-gram are models created to efficiently construct high-quality distributional vector representations.

Another type of word embedding that emerged along with word2vec is a count-base model, called Global Vectors (Glove) by Pennington et al. [2014]. In summary, Glove generates word embeddings by aggregating global word-word co-occurrence matrix from a corpus. By training Word2vec and Glove or any of these types of models on a large corpus of words, we can capture certain relationships between words such as gender, verb tense, and country-capital relationships, which has not been possible in early years. Many other frameworks for creating word embeddings are currently available as pre-trained vectors to be incorporated into deep learning models[4] [5] [6] in an fully automatic way.

The aforementioned embedding representations Mikolov et al. [2013c]; Pennington et al. [2014] consider all the sentences where a word is present in order to create a global vector representation of that word. A new method, called Embedding from Language Model (ELMo) emerged in 2018 Peters et al. [2018] proposing contextual word embeddings instead, where a word can have two different vector representations (each one with a different word sense). It provides deep contextual embeddings, producing word embeddings for each context a word is used. Moreover, the authors claim that combining ELMo with Glove or Word2vec is beneficial. However, the latest findings in transformers (explained later) have shown better results using BERT Devlin et al. [2018], surpassing ELMo.

[4] https://radimrehurek.com/gensim/
[5] https://fasttext.cc/
[6] https://tfhub.dev/google/elmo/2

During 2014-2016 neural networks, such as Convolutional Neural Networks and Recurrent Neural Networks started to be more adopted in several other tasks in natural language processing. Both RNNs and CNNs treat texts as sequences, but CNNs are more parallelizable because the states at every time step only depend on the local context (convolution operation) rather than on all past states (as in RNN) Kim [2014]; Kalchbrenner et al. [2014]; Razavian et al. [2014]; Zhang et al. [2015a]; Xiao and Cho [2016b]; Zhang et al. [2015b].

Following the evolution of deep learning models, attention models also evolved from sequence-to-sequence models Sutskever et al. [2014] and started a new era for NLP tasks. The use of attention neural networks is the most recent advance in NLP. Basically, this model encompasses an encoder and a decoder, both containing several RNNs in their architecture. The attention is an improvement from encoder-decoder model that, besides translating, also identifies which parts of the input sequence are relevant to each word in the output, whereas translation is the process of using the relevant information to select the appropriate output Bahdanau et al. [2014]; Cho et al. [2014].

The transformer was introduced by Vaswani et al. [2017] in 2017 in their groundbreaking paper named "Attention is all you need". It allowed more parallelization because it did not use any recurrent or convolution layers. The model architecture eschews recurrence and instead relies entirely on an attention mechanism.

Bidirectional Encoder Representations from Transformers (BERT) emerged from this idea. It is a bi-directional transformer created by Devlin et al. [2018] and uses different pre-training tasks for language modeling being applicable to several other tasks by means of fine-tuning. One of the pre-training tasks is to predict masked words and the other one is to predict the next sentence given a sentence. This pre-training methodology helps BERT, the most well known representative of transformers architecure, to outperform state-of-the-art techniques on key NLP tasks such as Question-Answer (QA) and Natural Language Inference (NLI), where understanding relations among two sentences is very important.

Right after BERT several other algorithms based on the same architecture appeared, such as RoBERTa Liu et al. [2019], DistilBERT Sanh et al. [2019], XLNet Yang et al. [2019] and many more. Those architectures improvements are either due to increased data, computation power or training procedure. Training them is very expensive and they tend to rely on a trade-off between computation and prediction results. The current effectiveness results (accuracy) reflect a huge advance in the research of NLP tasks, but there is still much study to be done in order to bring these algorithms to everyday (industry) tasks.

2.2 Neural Architectures

In the next subsections some neural network architectures are explained and discussed for understanding of some concepts. We clarify that deep learning architectures have lots of theories, concepts and details and the focus of our work is to compare the performance of the existing algorithms, rather than explaining each architecture in detail.

We introduce some important concepts, but do not deeply explain all of them. The reader is suggested to read references in the last section such as Zhang et al. [2015a]; Vaswani et al. [2017]; LeCun et al. [1995]; Patterson and Gibson [2017]; Foster [2019]; Fontaine [2018] for further understanding. Specifically, we focus on describing architectures such as multilayer perceptron, convolutional neural networks, attention models and transformers that are used in text-related tasks, most notably in automatic text classification.

2.2.1 Multilayer Perceptron - MLP

Deep feedforward networks, also called feedforward neural networks, or multilayer perceptron are the quintessential deep learning models Fontaine [2018]. Rosenblatt [1958] created the first perceptron, which was built in hardware. Later, Rumelhart et al. [1988] introduced the concepts of backpropagation and hidden layers, officially creating the multilayer perceptron.

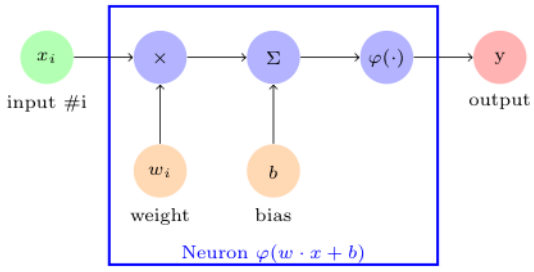


Figure 2.2. A neuron by Bonaccorso et al. [2018]

A neural network is formed by several single units, which are called artificial neurons, shown in Figure 2.2. In the illustration we can understand how each neuron is composed. Specifically, each neuron receives inputs, weights them on the basis of their importance, computes a weighted sum of the inputs and, finally, adds up an additional input b called bias to properly tune its output value. The output of a neuron is $n_i = h(w_i \cdot x + b_i)$, where w_i and b_i are the weights and bias of the linear transformation

and φ is a nonlinear activation function. The nonlinear activation function enables the neural network to model complex non-linearities of the underlying relations between the regressors and the target variable Pal and Prakash [2017]; Schifano et al. [2018].

A multilayer neural network chains many of these neurons and can create models that are more powerful than single neurons. Conceptually, as shown in Figure 2.3, a MLP is composed of an input layer, an output layer that makes a prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP.

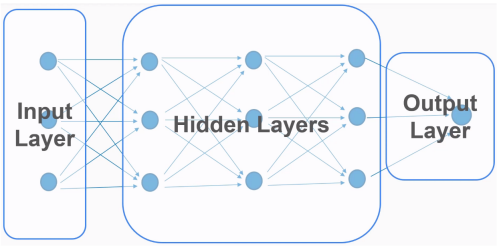


Figure 2.3. MLP by Fontaine [2018]

During the training, the weights are usually initialized to small random values and the bias to zero. After that, each input is passed from one layer to the next one until it reaches the output layer. In this phase, the network computes the output and the error corresponding to the input. The error is calculated on a training or validation set, and then propagated backwards changing the values of weights and bias in order to reduce the error itself Schifano et al. [2018].

An input layer is determined by the number of features in the dataset. For textual data, it can be words or characters and for images, it can be raw pixels values from different color channels. The output layer for classification problems, can have n neurons for n-way classification and utilize a softmax function to output the probability of belonging to each class. There are other hyperparameters that should be chosen somehow - via cross-validation or usually in trial-and-error setting. Specifically, hyperparameters are numerical presets, which are values assigned prior to the start of the learning process. It is critical to select good hyperparameters and very difficult to determine their optimal values. Most hyperparameter optimization algorithms depend on searching a generic range of values and these are imposed blindly on all sequences Dong et al. [2018]. Following, we explain some important hyperparameters that can be tuned during training of deep learning architectures.

Activation function: It is used in every neuron and it is responsible for deciding whether the incoming signals have reached the threshold and should output signals for the next level. In practice the activation function activates a neuron

given the value it receives exceeds a certain threshold. Different types of activation functions are possible. The most common is rectified linear unit (ReLU) (sigmoid and tanh converge slowly) .

Optimization algorithm: It is used to learn weights of the network. Specifically, it is the process that minimizes the error and adjusts the network coefficients step by step. The most used algorithms are based on gradient descents, each one applying different types of improvement, such as Adagrad, Adam and RMSProp.

Loss function: the function that will produce the quantity that will be minimized/maximized by the optimizer (e.g. accuracy, categorical loss).

Weight initialization: In most cases, weights are initialized randomly. In some finely-tuned settings, weights are initialized using a pre-trained model strategy.

Regularization strategy: It is used to decrease overfitting of models. Usually, the regularization is done by adding some constraints on the parameters, such as L1 or L2 regularization, which prevent the weights or coefficients of the networks growing too big.

These same hyperparameters are present in CNNs, RNNs and all the other neural network architectures. As previously described, setting each of these hyperparameters is very important. More complex architectures have many different hyperparameters and they are very sensitive to even small changes to these hyperparameters. There are many possible values (there is a huge number of possibilities), which brings more complexity during tuning or choice of parameters. In our implementation of the neural network architectures we tuned a set of hyperparameters during training, which is shown in Table 3.3.

In this work we develop two neural networks based in MLP architecture, we feed them with feature selection of TFIDF weights. More details are described in Subsection 3.2.1.

2.2.2 Convolutional Neural Networks - CNN

A CNN typically involves two methods: convolution and pooling. This architecture was inspired in the visual cortex, shown in Hubel and Wiesel [1959]. It has a small local receptive field, meaning that they react only to visual stimuli located in a limited region of the visual field. Figure 2.4 illustrates how the local receptive fields work in the picturing of an image by an eye.

The neurons in the visual cortex react to a small or limited area of the overall image due to the presence of a small local receptive field. A local receptive field is an

area on the neuron that excites or activates that neuron to fire information to other neurons. Thus, they are activated by viewing a local area of the image via its local receptive field.

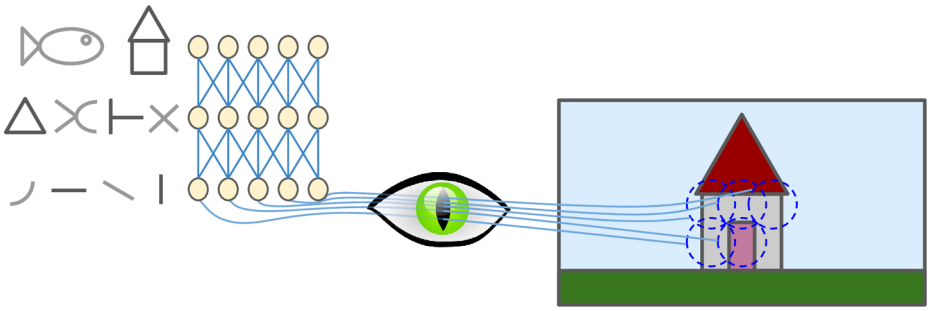


Figure 2.4. Visual Cortex Example Géron [2018]

Each neuron in the visual cortex reacts to a different type of visual information, some recognize horizontal lines, while others react on different orientations. These, among other very interesting observations led to the creation of the convolutional neural networks. In CNNs, each local receptive field can be seen convolving the image above, with a kernel (represented by the blue circle), which recognizes different structures such as lines, shapes, etc.

As previously described, CNNs emerged in the 80's, but only after the creation of GPUs and parallel processing that algorithms of neural networks reemerged. In 2009, Raina et al. [2009] showed that training neural networks on massively parallel graphics processors greatly surpassed the traditional methods of training on multi-core CPUs, what stimulated many other works. Initially, CNNs were created to recognize shapes and patterns in images and audio Lee et al. [2009]; Krizhevsky et al. [2012]; Masci et al. [2013]; Sermanet et al. [2013] but recently started to be used to train language modelsMelis et al. [2017] and to perform classification and other natural language processing tasks Mikolov et al. [2013c]; Conneau et al. [2017]; Zhang et al. [2015a].

2.2.2.1 Convolutions

The convolution layer is where most of the computational operations are done. During a convolution, as shown in Figure 2.5, the kernel (sliding window over the image) is multiplied by the input data values within the same boundary as the kernel and creates a single entry in the output convolved feature (activation map or feature map or feature detector). This architecture allows the network to concentrate on low-level features in the first hidden layer, before assembling them into higher-level features in the next

hidden layer, and so on as shown in Figure 2.6 Patterson and Gibson [2017]; Kulkarni and Shivananda [2019]; Géron [2018].

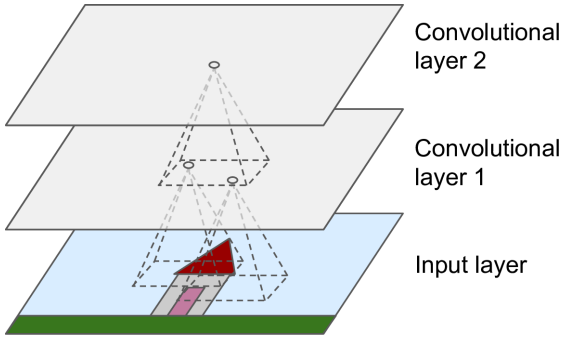


Figure 2.5. Convolution Operation Géron [2018]

The filter size and stride of the filter are important hyperparameters when designing a convolutional layer. We explain each one in the following explanation of a convolution.

The convolution operation begins at the top of the input matrix. The values of the input matrix are multiplied by the corresponding values in the convolution filter. All of the multiplied values are added together resulting in a single scalar, which is placed in the convoluted feature matrix. After that, the kernel moves to the right by the length of x pixels, where x is called stride (a parameter of the CNN structure), as illustrated in Figure 2.6. This process is repeated, covering an entire row, and then shifted down the columns by the same stride length, until the whole input matrix is convoluted.

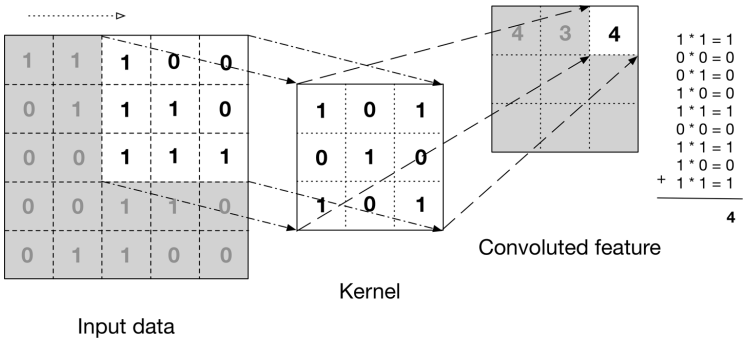


Figure 2.6. Convolution Operation Patterson and Gibson [2017]

Usually, after a convolutional layer there is a layer called pooling, which reduces the size of the resulting feature map through a aggregation operation. We explain it next.

2.2.2.2 Pooling

In order to reduce the computational load, the memory usage, and the number of parameters, a sub-sampling technique called Pooling is applied. A pooling neuron does not have weights. It aggregates the inputs using an aggregation function, such as max or mean. Figure 2.7 examplifies how the pooling operation works with a max pooling kernel with stride 2. Specifically, the pooling layer summarizes the features learned in the previous layers, helping to prevent overfitting. The advantage of the pooling layer is its ability to inject location invariance into the network, which means that features can be detected by the network wherever they are on the input.

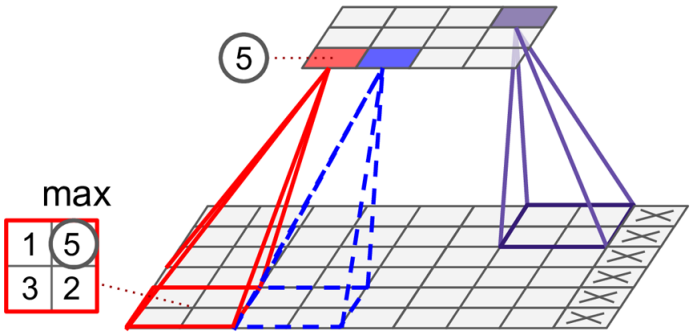


Figure 2.7. Pooling Operation Géron [2018]

2.2.2.3 CNNs Applied to Text

Convolutions were first applied to textual tasks by Collobert et al. [2011] in semantic-role labeling and later by Kim [2014] in sentiment and question classification. The convolution is applied to text in a similar way as to images, but slightly different because the process of convolution occurs in 1D.

Figures 2.8 and 2.9 illustrate how a filter works in word and character based inputs. Each word or character is represented by a vectorized encoded representation, for instance, a one-hot encoded or even real-valued representation.

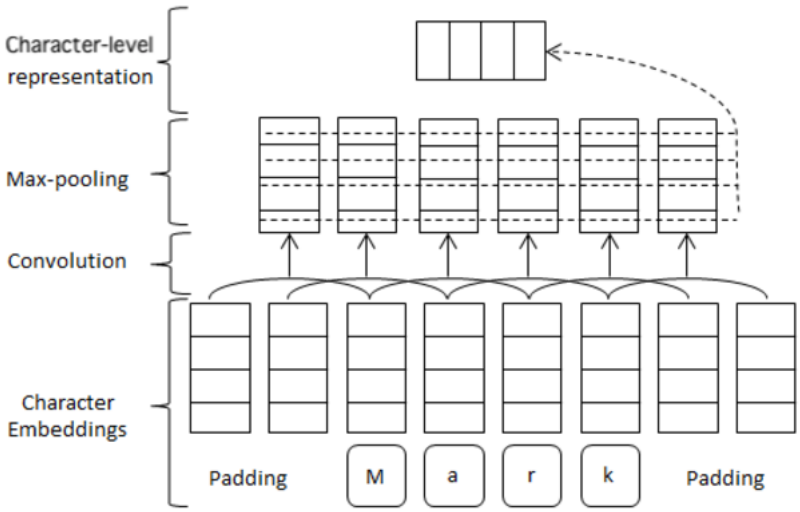


Figure 2.8. Character Convolution over text - adapted from Zhai et al. [2018]

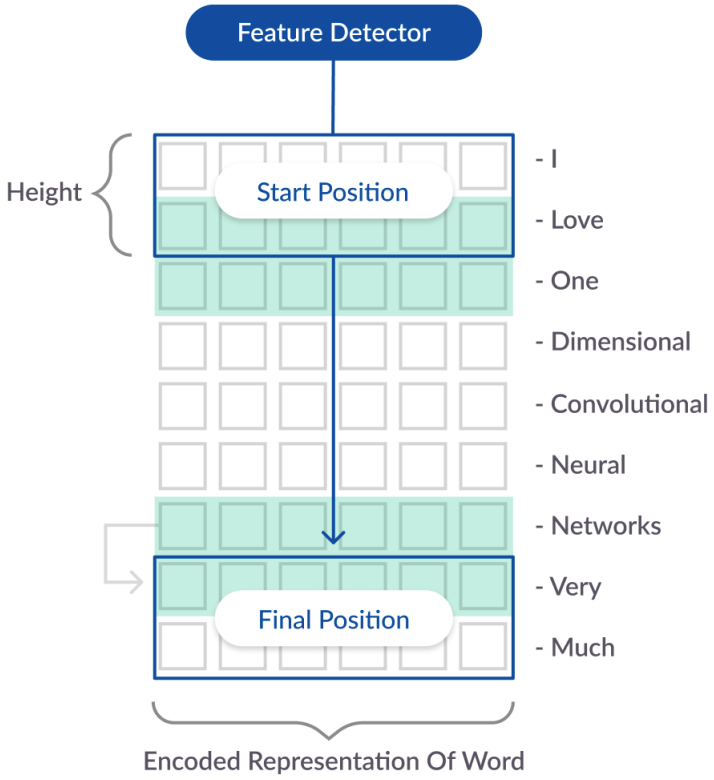


Figure 2.9. Word-level Convolution

When dealing with a character-level representation is necessary to: (i) define an alphabet that will be considered and (ii) determine the max length of the documents size to work on batches. Figure 2.8 exemplifies the process of convolution over the word *Mark*. Each character is represented in a vector of the length of the considered alphabet, which is generally summed up, so that smaller sentences have the same length

as the maximum document length. Following, there is a layer of convolution, which has a kernel of size 3 and stride 1. After that a max-pooling layer takes the maximum active value across the respective convolved matrix.

The process of convolution over text can capture k-grams from the sequence of m vectors Goldberg [2017]. Usually CNNs have many layers, because each layer capture distinct feature maps. Number of kernels, their dimension, number of convolutions, and types of pooling are the most import hyperparameters.

In this master thesis we have considered architectures working on both structural levels (character and word). We will explain each one of them in the subsections 3.2.2 and 3.2.3..

2.2.3 Attention and Transformers

Sequence-to-sequence (or encoder-decoder) models are a relatively recent architecture that have created many possibilities for machine translation, speech recognition, and text summarization. An encoder-decoder maps an input sequence to an output sequence, which can be of a different length Salvaris et al. [2018]. Encoder-decoder networks were initially applied to text translation. In this method when the sentence becomes large, the information from the start of it might not be learned well and the network struggles to retain all information. Thus, attention mechanisms were introduced in order to solve this kind of problem Foster [2019].

Attention models were first introduced in NLP for machine translation tasks by Bahdanau et al. [2014]. It can be used as a tool for interpreting the behavior of neural networks, which are very difficult to understand.

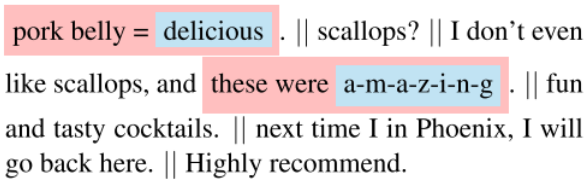


Figure 2.10. Yelp 2013 sentence attention example

Figure 2.10 shows a sentence example from Yang et al. [2016] of how an attention helps to understand results from the neural network. The first and third sentence have stronger meaning and the words delicious and a-m-a-z-i-n-g contribute the most in defining the sentiment of the two sentences. The attention helps to create visualizations that highlights attention weights, making it possible to investigate and understand the outcome.

Mainly, attention allows the model to focus on the relevant parts of the input sequence as needed. A nice screenshot from Alammar [2018] post is shown in Figure 2.11 and illustrates how attention works in a high level. We will explain it briefly and also in a high level due to its high complexity; for further understanding we suggest to continue on specialized literature Alammar [2018]; Vaswani et al. [2017].

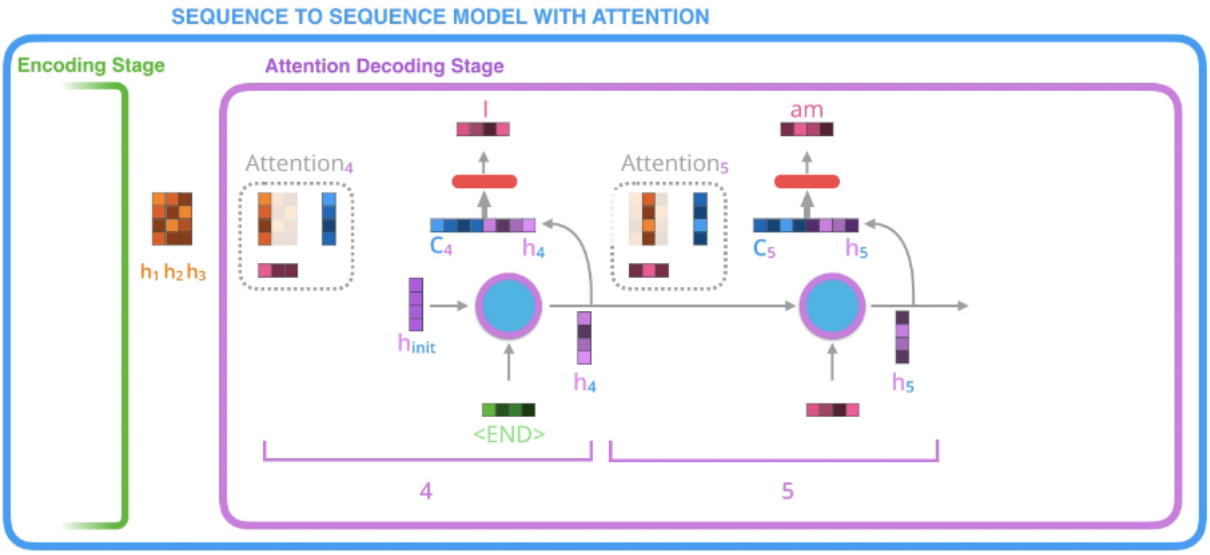


Figure 2.11. An attention on a high level Alammar [2018]

We start by reading the image above from left to the right side. The encoding state (in green) is a set of recurrent neural networks, which receive text as input data and process it to become basically a vector of numbers, that we call hidden states (a representation of the input given to the encoder). After that, the encoder passes the hidden states to the decoder, that receives a current word and initial hidden state. The RNN (blue circle) process its inputs, producing an output and a new hidden state vector ($h4$). Once the RNN outputs $h4$, it uses the hidden states from the decoder plus $h4$ to calculate a context vector $c4$, that represents the scores from the words around the current word, for the current time step. Then both are concatenated into one vector, passing through a feedforward neural network and its output indicate the word for this time step. Repeating this process for the next time steps (next words).

The use of attention have introduced a challenge of increased computational complexity of computing a separate context vector for every step of decoder. Transformers were proposed by Vaswani et al. [2017] to mitigate this problem, being an evolution of attention models as they do not rely on recurrent units. The authors Vaswani et al. [2017] affirm:

The Transformer is the first transduction model relying entirely on self-

attention to compute representations of its input and output without using sequence-aligned RNNs or convolution.

Specifically it has stacked layers of encoders and decoders. The encoder is composed by a self-attention model and a feed forward neural network and the decoder has 3 layers: a self-attention, a encoder-decoder attention and a feed forward. As it is shown in Figure 2.12.

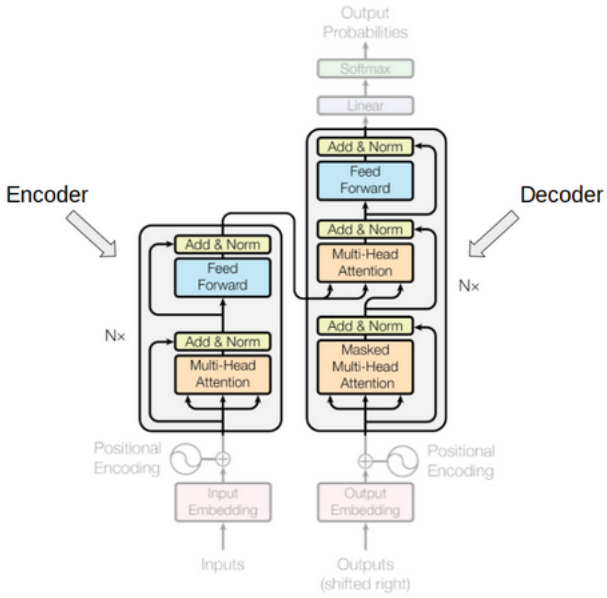


Figure 2.12. The transformer

Self-attention is defined by the authors as:

> Self-attention, sometimes called intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

In short, self-attention allows the model to look at the other words in the input sequence to get a better understanding of a certain word in the sequence. The transformer computes self-attention multiple times, thus they name this process as Multi-head Attention.

Transformers demonstrate to be a huge improvement over the current NN architectures and the most recent technique applied to NLP tasks. They have overcome some problems but there are still some limitations such as fixed-length of text input, which leads to context fragmentation. This means that important sentences might be split without considering the semantics.

In this master thesis we study the performance of an attention neural network based on encoding words and sentences as well as transformer model. More details in the subsections 3.2.5 and 3.2.4, respectively.

2.3 Related Work

In this section we discuss some important related work that classifies text using deep learning models. Mainly, we highlight the architectures and discuss how each one of them was evaluated. Although there are many papers using different architectures for NLP tasks (and much more are surging every day) we summarize some important references in order to demonstrate that there might be some questionable comparisons even in the most popular and cited works. Furthermore, we show that several works make strong assumptions where they might not be significantly true.

Table 2.1 summarizes important information about some of the recent published papers, highlighting the year, type of the architecture, input of the neural network (words, characters or sentences), whether any type of statistical test was applied, if there was tuning and how it was done, and finally, if the input uses pre-trained word embeddings.

Table 2.1. Research Papers

Paper	Year	Architecture	Input Level	Statistical Tests	Tuning	Embeddings
CNN Kim [2014]	2014	CNN	Word	No	Grid search on the SST-2 dev set and tuning only on early stopping	Word2vec
GRNN Tang et al. [2015]	2015	CNN, RNN	Word	No		Word2vec
CNN Zhang et al. [2015b]	2015	CNN	Char	No	Not mentioned	—
ConvRec Xiao and Cho [2016a]	2016	CNN+RNN	Char	No	Not mentioned	—
HAN Yang et al. [2016]	2016	Attention	Sentence and word	No	Tuning on validation set; grid-search on the learning rate	Word2vec
VDCNN Conneau et al. [2017]	2017	CNN	Char	No	Not mentioned	—
BERT Devlin et al. [2018]	2018	Transformer	Sentence	No	Finetuning of some hyperparameters	—
XLNet Yang et al. [2019]	2019	Transformer	Sentence	No	Fine-tuning of some hyperparameters	—

Kim [2014] a pioneer work proposed a simple and shallow CNN for sentence-level classification with one layer of convolution on top of word vectors obtained from an unsupervised neural language model (word2vec). The CNN is composed of multiple filters, in order to obtain different n-gram sizes from the text input. The tuning of the architecture occurred only in one dataset and the best values found were used in all the other datasets. Additionally the best early stopping criteria was chosen on all validation sets (called as dev-sets by the authors).

Their CNN was evaluated on seven small sentiment datasets and compared to more than 10 other methods. None of the baselines was evaluated in all datasets, as the authors could not reproduce them in their experimental setup. One of the baselines was a Support Vector Machine, which was evaluated only in one of the datasets, in which it produced the best results. Although Kim [2014]'s architecture showed gains on 4 out of 7 datasets, the results of some baselines were not computed in many of the datasets, which leaves uncertainty about the real contribution of the CNN on the comparison.

Bengio et al. [1994] affirms that Recurrent Neural Networks (RNNs) suffer from vanishing gradients, where gradients may grow or decay exponentially over long sequences. One of Tang et al. [2015] experiments is an analysis of the impact of using gated recurrent units (GRU) instead of standard RNNs. In that experiment they demonstrated that GRUs obtain much better results than standard RNNs. Such improvement is justified by the fact that the document representation encodes rare semantics on the beginning of sentences that standard RNNs might not capture, but GRUs can.

Tang et al. [2015] conducted experiments with CNNs and RNNs in sentiment classification at word level on four-large scale review datasets. They have described, as one of their main contributions, that adding neural gates boost performance on RNNs and improves the vanishing gradient problem. In their experiments Tang's architectures were compared to algorithms such as SVM (with inputs such as bigrams, unigrams, among others), CNN Kim [2014] and paragraph vector Le and Mikolov [2014]. Their method showed great performance over chosen baselines, but there was no information of tuning of the algorithms or training process, which also leaves doubt on results.

There are similar researches dealing with document classification at character-level using embeddings of characters. Zhang et al. [2015b] was the first to implement this idea, followed by Xiao and Cho [2016a]. The former one designed two CNNs: using 1024 and 256 kernels in the so called large and small CNNs, respectively. The latter one proposed a hybrid of convolutional and recurrent layers, that process an input sequence of characters with a number of convolutional layers followed by a single recurrent layer. Both proposals evaluated their models on eight large-scale document classification tasks, including sentiment analysis, ontology classification, question type classification and news categorization.

However, Zhang et al. [2015b], in turn, compared their results with only a logistic regression (LR) classifier using bag-of-words, n-grams, term frequency variants as traditional classifier and a vanilla long-short term memory (LSTM) neural networks. Xiao and Cho [2016a] compared themselves to Zhang et al. [2015b] and its baselines.

Zhang et al. [2015b] results seem to indicate that traditional methods might be better for smaller corpus of data (up to several hundreds of thousands). However, these works did not show any analysis or described any process of tuning neither for the proposed methods or for traditional algorithms. Thus concerns about the significance of results can be raised in the analysis and understanding of the results.

Yang et al. [2016] proposed a hierarchical attention network (HAN) applied to word and sentence levels, which can construct different document representation to recognize more and less important contents in the text. The intuition in their model is that, to determine relevant sections, it is necessary to model interactions of the words, not just their presence in isolation. Thus, HAN includes two levels of attention mechanisms: one at word level and one at the sentence level, allowing the model to pay more or less attention to individual words and sentences when constructing the document representation. Their architecture is complex with encoders and attention layers for word and sentences. We recommend further reading of Yang et al. [2016] for more details and understanding of the architecture.

In their experiments, Yang et al. [2016] used a default split of 80-10-10% for training, validation and testing and when there is no validation in the original data, 10% of the training is randomly selected as validation, for tuning. The authors only used results of the baselines reported in previous papers, having not run the baselines in their own datasets, which is a drawback in their experiments.

In contrast to Kim [2014], Conneau et al. [2017] proposed a very deep convolutional neural network (called VDCNN) for text classification also at character-level, which was the first work to go deeper than six convolutional layers for sentence classification. The VDCNN architecture is inspired in the idea of stacks of convolutional blocks, considering different sizes of filters and pooling and same sizes of kernel (kernel=3). In the experiments, different depths were explored. They observed that depths of 9, 17, 29 and 49 performed better than other depths and showed that the time of training of each model varies from 24 minutes to 7 hours for each epoch, where the number of epochs vary between 10 to 15 until convergence. Thus, the training process is very time consuming even without any tuning of hyperparameters. However, authors showed that the increase of the network depth improves performance in all datasets and that CNNs at character-level can be an effective method.

In the recent years (2018-2019), many deep learning models emerged using transformers in their architectures Devlin et al. [2018]; Sanh et al. [2019]; Yang et al. [2019]; Liu et al. [2019]. Initially, transformers were developed to solve problems related to neural machine translation but their use has broadening to almost all NLP tasks. BERT is one of the most popular transformer based methods, which is essentially a

language representation strategy that created SOTA models for a wide range of NLP tasks such as language inference, question answering and classification. The authors found a set of possible values for fine-tuning and indicated how to run an exhaustive search over the parameters for specific tasks. Models using transformers can be trained faster than architectures based on RNNs or CNNs as shown in Vaswani et al. [2017].

There is not a single paper in all listed works in the Table 2.1 that evaluates the proposed architectures with cross-validation or any type of replication using any other metric besides accuracy (or testing error) - a few use MSE - or with validation using any statistical tests. Moreover, in order to it is also important to know: (i) how confident the model is in the classification (e.g. with log loss), (ii) how is the performance varies across the classes (f1-macro score), (iii) how time consuming is to train such models and (iv) how the performance of the method when compared to other methods properly tuned.

2.4 Summary

In this section we introduced different architectures and problems present in text classification when neural networks are applied. We also highlighted how each prior study was evaluated. Throughout this section we can clearly see that there are some pitfalls in the evaluation of deep learning methodologies for text classification. In this dissertation we aim at overcome many of these issues.

Chapter 3

Experimental Setup

Our experimental setup is presented in this chapter. We provide a description of datasets in Section 3.1, in terms of size, skeweness, etc. We also provide details of the compared deep learning algorithms, including their architectures in Section 3.2. Finally, we describe and explain the evaluation metrics used to measure and compare the quality of the methods.

3.1 Datasets

We evaluate the effectiveness of the models on three large scale document classification datasets and four small real-world textual datasets, namely, AG News corpus (AGNEWS), Sogou News corpus (SOGOU), Yelp Review Full, 20 Newsgroups (20NG), Four Universities (4UNI), Reuters (REUT), ACM Digital Library (ACM), respectively. Table 3.1 describes class distribution and Table 3.2 summarizes the distributions of characters and words for each dataset.

Zhang et al. [2015a] constructed a set containing eight large scale datasets. We have chosen three of them: AG' news, Sogou news and Yelp full because several publications have compared their methods using them. Thus, we are capable of comparing the algorithms using the same datasets. A brief description of all datasets follows next.

AGs news corpus (AGNEWS) Zhang et al. [2015a] obtained the AGs corpus of news articles on the web[7]. The whole corpus contains 496,835 categorized news articles from more than 2000 news sources. The four largest classes (World, Sports, Business and Sci/Tech) from this corpus were chosen to construct the

[7] `http://www.di.unipi.it/gulli/AG_corpus_of_news_articles.html`

dataset, using only the title and description fields. The number of training samples for each class is 30,000 and testing 1900.

Sogou news corpus (SOGOU) is a Chinese dataset from the combination of the SogouCA and SogouCS news corpora, containing 2,909,551 news articles in various topic channels. They were labeled by manually classifying their domain names. Five categories were chosen: sports, finance, entertainment, automobile and technology. There are 90,000 training samples and 12,000 testing samples for each class. The models for English can be applied to this dataset without change. The fields used are title and content Zhang et al. [2015a].

Yelp reviews full (YELP) is obtained from the 2015 Yelp Data Challenge. The original dataset, contains 1,569,264 review text samples. Zhang et al. [2015a] selected 130,000 training samples and 10,000 testing samples each star of Yelp website (1-5 starts), adding up to then 650,000 training and 50,000 testing instances.

4 Universities (4UNI), a.k.a, WebKB contains Web pages collected from Computer Science departments of four universities (Cornell (867 pages), Texas (827), Washington (1205), Wisconsin (1263) and 4,120 miscellaneous pages collected from other universities) by the Carnegie Mellon University (CMU) text learning group[8]. There is a total of 8,282 web pages, classified into 7 categories: "student", "faculty", "staff", "department", "course", "project" and "other". Table 3.1 shows 8199 instances. This is due to a preprocessing step which removed documents with few words Campos et al. [2017b].

20 Newsgroups (20NG) is a classical dataset, that has become a popular data set for experiments in text applications of machine learning techniques. It contains 18,846 newsgroups documents[9], partitioned almost evenly across 20 different newsgroup categories. 20NG has become a popular dataset for experiments in text applications of machine learning techniques, such as text classification and text clustering.

ACM-Digital Library (ACM) a subset of the ACM Digital Library with 24, 897 documents containing articles related to Computer Science. We considered only the first level of the taxonomy adopted by ACM, where each document is assigned to one of 11 classes.

[8] `http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data`
[9] `http://qwone.com/~jason/20Newsgroups/`

Reuters (REUT) this is a classical text dataset, composed by news articles collected and annotated by Carnegie Group, Inc. and Reuters, Ltd. We consider here a set of 13, 327 articles, classified into 90 categories.

Table 3.1 describes the classes distribution for each dataset. The large scale datasets are balanced and the small ones are unbalanced.

				Skewness						
Dataset	Size	# Features	Features/Size	# Classes	Minor Class	1st Quartile	Median	Mean	3rd Quartile	Major Class
4UNI	8199	23047	2.81	7	137	342	926	1171	1374	3705
REUT	13327	27302	2.03	90	2	8	29	148	91	3964
20NG	18846	97401	5.17	20	628	957	984	942	990	999
ACM	24897	48867	1.96	11	63	761	2041	2263	3278	6562
AGNEWS	127600	39837	0.31	4	31900	31900	31900	31900	31900	31900
SOGOU	510000	98974	0.19	5	102000	102000	102000	102000	102000	102000
YELP	700000	115371	0.16	5	140000	140000	140000	140000	140000	140000

Table 3.1. Datasets Statistics

Table 3.2 shows word and character distributions for each dataset. The highest number of characters occurs in 4UNI, 20NG and SOGOU. AGNEWS has the smallest number of characters and words, followed by YELP (with the highest number of instances) and REUT.

	Characters Distribution						Words Distribution					
Dataset	Minor	1st Quartile	Median	Mean	3rd Quartile	Major	Minor	1st Quartile	Median	Mean	3rd Quartile	Major
4UNI	2	304	654	1267	1310	149522	1	53	106	209	207	38528
REUT	29	244	479	748	948	7473	7	61	10	163	198	2383
20NG	1	356	663	1257	1218	152134	1	71	134	244	248	11765
ACM	5	61	95	358	584	19388	1	10	16	63	103	3967
AGNEWS	79	165	195	199	224	911	8	32	37	38	43	177
SOGOU	31	508	1613	2192	3067	181883	2	131	416	567	791	43737
YELP	1	234	438	597	779	4761	1	52	99	1341	175	1052

Table 3.2. Datasets Vocabulary Statistics

The large datasets are available online[10]. Their original splits (training and testing) have different percentages of training and testing. In the thesis we concatenate original training and testing sets and split them into 5-folds for cross-validation procedure in order to have equal quantities for standardized comparison. In the small datasets we use 10-fold cross validation in order to have bigger training sets (and hopefully less variation).

3.2 Algorithms and Methods

In this section we explain the algorithms and parameters for each compared method. Our main goal is to reproduce the architectures in a controlled experimental environment in order to evaluate the effectiveness of the chosen methods. The reasons of the

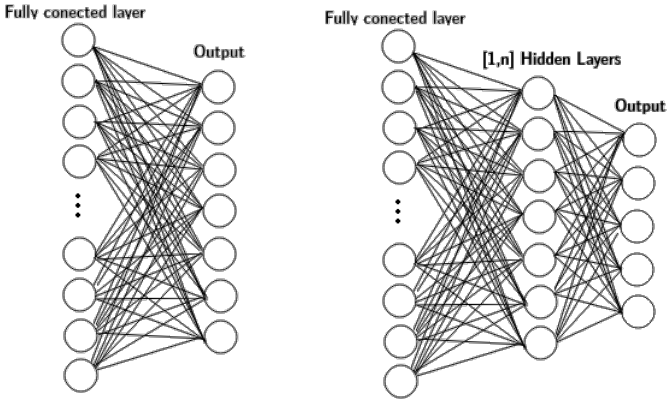
[10]Shorten link for big datasets: `http://tiny.cc/2hichz`

choice of each one of them is two fold: (i) the popularity in the NLP academy; (ii) the easiness of reproducing the implementations as faithfully as possible to original publication.

During replication of the neural networks we have used hyperparameters reported by the authors of each architecture and kept 100 epochs for smaller datasets and 18 epochs for bigger datasets or otherwise described.

3.2.1 Multilayer Perceptron - MLP and DMLP

We have implemented a Multilayer Perceptron (MLP) and a deep MLP (DMLP). We ran MLP and DMLP only in the small datasets due to their high computational cost. Our goal with these architectures is to have a simple and vanilla baseline with which we can compare more complex architectures to understand better the potential of recent trends.



(a) MLP **(b)** Deep MLP

Figure 3.1. Multilayer Perceptron Architectures

Our MLP model is composed by an input layer and an output layer with no hidden layers such as Figure 3.1 shows. Both neural networks are fully connected (also called dense layer), which means that all neurons are connected to each other in the next layer. The DMLP is very similar to MLP, but it adds up some hidden layers and dropout (random removal of neurons) as a way to control overfitting by randomly omitting subsets of features at each iteration of the training procedure Hinton et al. [2012]. Neural networks have many hyperparameters to be tuned. Unfortunately it is impossible to use a grid-search or random-search over all of them. We have chosen some hyperparameters to be tuned in MLP and DMLP architectures.

Table 3.3 shows the range we select for each hyperparameter. DMLP and MLP present distinct architectures and their tuning also differ in the parameters.

Hyperparameters	Range
#Neurons	32, 64, 128
Weight Initialization	random-normal, random-uniform
Activation Function	Relu, Linear
Optimizer	SGD, Adam, Adamgrad, RMSprop
Optimizer Learning rate	1×10^{-2}, 1×10^{-3}, 1×10^{-4}
Regularization	$l1$, $l2$, $l1l2$
Regularization Learning rate	1×10^{-2}, 1×10^{-3}, 1×10^{-4}, 1×10^{-5}
#Layers	1,2,4
Dropout	0, 0.1, 0.3, 0.5, 0.7

Table 3.3. Hyperparameters

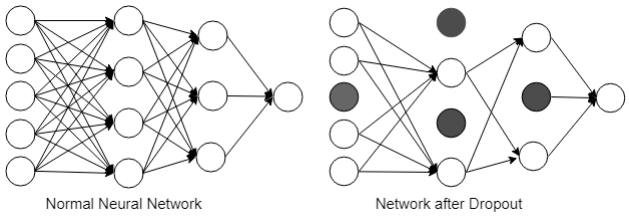
Each range shown in the Table 3.3 was chosen in a trial-and-error process and then the best set was chosen to perform the tuning. The MLP contains one input and one output layers, regarding those layers we chose number of neurons, weight initialization, activation function, optimizer and regularization during tuning. DMLP was tuned in the same manner plus dropout and number of layers.

As we previously stated, the number of neurons in the input and output layers are determined by the type of input and output the task requires, but the hidden layers need to be set. We have set the number of possible values to 32, 64 and 128 because larger numbers affect very much the overfitting of the models.

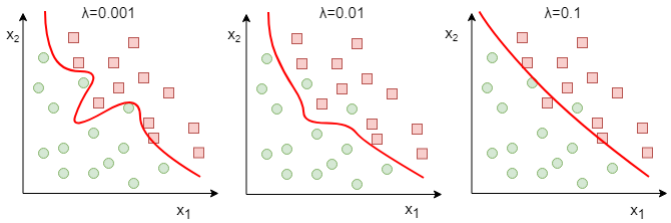
When the optimal initial weights are determined, the initial error is substantially smaller and the number of iterations required to achieve the error criterion is significantly reduced Yam and Chow [2000]. Then choosing the right weight initialization is very important and we set the ranges random-normal or random-uniform to be chosen in the parameterization.

Ganju [2017] affirms that the learning rate determines how quickly (and whether at all) the network reaches the optimum. A high learning rate while proceeding into the training iterations has a high probability to fall into a local minima. In other words, a low learning rate slows down the learning process but converges smoothly while a larger learning rate speeds up the learning but may not converge. Hence we tried to choose the range values according to the time the model takes to converge and to have the least overfitting (as assessed during initial tests).

Overfitting can be reduced by using dropout or other regularization techniques. We use Dropout and Regularization (l1, l2 or l1l2). Dropout removes some of the neurons randomly (along with their input and output connections) from the network. Regarding regularization, the L1 penalizes the absolute value of the weight and tends to make the weights of the neural network zero and L2 penalizes the squared value of the weight and tends to make the weight smaller during the training Vasilev [2019]. Figure 3.2 illustrates how each one of these both regularization techniques work in a neural network.



(a) Dropout



(b) Regularization

Figure 3.2. Regularization Techniques by Arumugam and Shanmugamani [2018]

In the right of Figure 3.2(a) we see a neural network with dropout and in Figure 3.2(b) we see an example of how decreasing the regularization might affect the overfitting of the model (from right to left plot). A high regularization leads to a network nearly linear that cannot shape complicated decision boundaries.

The optimizer function aims at updating the network weights in a way that is going to minimize the training loss error Arumugam and Shanmugamani [2018]. We have chosen some popular optimizers that are available for tuning in Keras package[11]. During our experiments we have set a patience parameter, which is an early stopping criteria that indicates when a neural network might stop its training. Thus, for example, if after n epochs the loss error keeps steady and does not improve anymore the model stops its training.

3.2.2 Very Deep Convolutional Neural Networks - VDCNN

VDCNN was developed by Facebook research group Conneau et al. [2017]. It is another very cited work that classifies text using very deep convolutional neural networks at character-level. The architecture uses many layers of small convolutions (size 3). It starts with a lookup table containing embeddings of 1024 characters, followed by 64 convolutions of size 3 and a stack of temporal (applied in 1 dimension) convolutional blocks.

[11] `https://keras.io/`

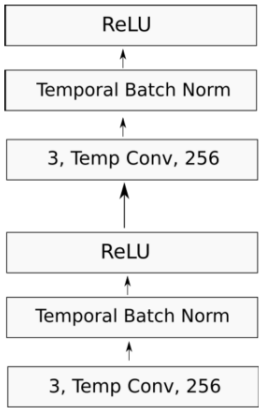


Figure 3.3. Convolutional block by Conneau et al. [2017]

Each convolutional block, shown in Figure 3.3, is a sequence of two convolutional layers, each one followed by a temporal BatchNorm layer and an ReLU activation. A batch normalization aims at improving the performance, speed and stability of neural networks Ioffe and Szegedy [2015]; Santurkar et al. [2018]; Kohler et al. [2018].

The depth of VDCNN is defined through the number of convolutional blocks in the architecture. For each convolutional block we count 2 for depth; hence if an architecture has 4 convolutional blocks we have 16 summing one convolutional layer at the beginning of the network summing up to 17. An example of such depth is shown in the Figure 3.4.

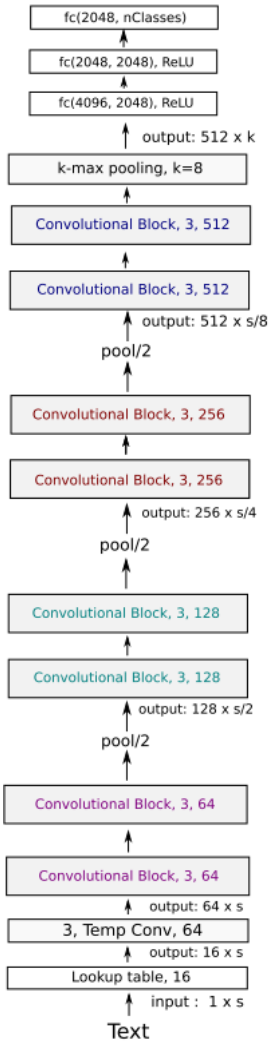


Figure 3.4. VDCNN - 17 layers by Conneau et al. [2017]

Conneau et al. [2017] use 69 characters and a fixed text size of 1014 to define the input. All convolutions have kernel size equals to 3 and the number of filters in each convolutional block is 64, 128, 256 and 512. During experiments, Conneau et al. [2017] show results classifying on four different depths: 9, 17, 29 and 49 as illustrated in Figure 3.4, but in our reproduction of VDCNN we have explored fewer blocks due to limitations in our GPU memory. Table 3.4 describes the number of convolutional blocks we use in this master thesis.

Depth	5	9	15
Convolutional block 512	1	2	2
Convolutional block 256	1	2	2
Convolutional block 128	1	2	5
Convolutional block 64	1	2	5
First convolution	1	1	1

Table 3.4. Number of convolutional layers per depth

The VDCNN has many parameters to be set, which impacts directly in the training and tuning time. Thus as the authors, we do not tune the model and use reported

hyperparameters.

3.2.3 Character-level Convolutional Neural Network

Zhang et al. [2015a] developed 2 CNN architectures at character-level for text classification, shown in Figure 3.5.

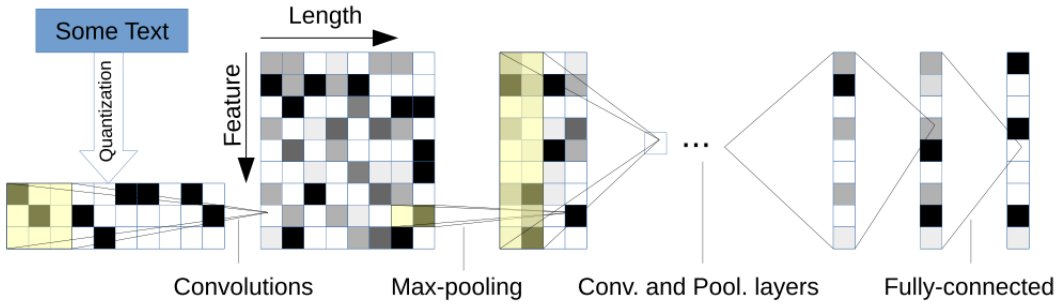


Figure 3.5. Zhang et al. [2015a] CNN

The first step in the neural network is the character quantization, where each character from the input is quantized using one-hot encoding, which is one if the character appears and zero otherwise. Then, each sequence of characters is transformed to a fixed length of 1014 (exceeding characters are ignored).

Both architectures are 9 layers deep with 6 convolutional layers and 3 fully connected layers. The difference between them is the number of filters: 1024 and 256 for the large and small architecture, respectively. The authors define the input size equal to 70 due to the character quantization method, which considers 70 characters (including 26 English letters, 10 digits, 33 other characters and the new line character), and the input feature length is 1014. Thus our input matrix has 70×1014 shape. Two dropout modules are defined in between the 3 fully-connected layers with probability of 0.5. The models are initialized using a Gaussian distribution with mean and standard deviation as $(0, 0.02)$ for the large model and $(0, 0.05)$ for the small one.

In this master thesis we follow the same architecture parameters for reproducing the models. In Chapter 4 we show the results we reproduced with the small and large architectures.

3.2.4 Bidirectional Encoder Representations from Transformers - BERT

BERT is a model developed and introduced by Google Devlin et al. [2018]. It is a massive Transformer-based model (with 110 million parameters in its smallest version and

340 million parameters in the largest), pre-trained on Wikipedia and the BookCorpus dataset. In short, it predicts missing words from a sentence. The authors proposed two pre-training tasks using unsupervised prediction that explain the model's strength:

Masked language model (MLM) - This technique masks words with a probability of 15% and the model is trained to predict the masked words. For example, if the original sentence is I finished my master thesis last month then the model may be given the sentence I <mask> my master thesis <mask> month and it must predict the words finished and last. Each selected word has an 80% chance of being masked, a 10% chance of being replaced by a random word, and a 10% chance of being left alone.

Next sentence prediction (NSP) - It trains the model to predict whether two sentences are consecutive or not. During training 50% of the time A and B are consecutive and 50% of the time is a random sentence. For example, given the sentences I finished my master thesis last month and "I will defend today", the model should predict that they are consecutive, while I finished my master thesis last month and I ate hot dog are not consecutive.

BERT uses a multi-layer bidirectional Transformer encoder. Its self-attention layer performs self-attention in both directions. For text classification tasks a fine-tuning to the target domain is necessary. The authors suggest a set of hyperparameters to be searched while fine-tuning: Batch size: 16, 3; Learning rate (Adam): 5e-5, 3e-5, 2e-5; Number of epochs: 3, 4. The search for best hyperparameters is done in the validation split for bigger datasets.

3.2.5 Hierarchical Attention Networks - HAN

The HAN architecture used in this master thesis is inspired by Yang et al. [2016]. In their work, the HAN has two levels of attention mechanisms: one applied at the word-level and another at sentence-level. This enables the neural network to understand content that is more or less important for constructing the document representation. Specifically, the model consists of: an encoder and attention layer for words and an encoder and attention layer for sentences, as it is shown in Figure 3.6

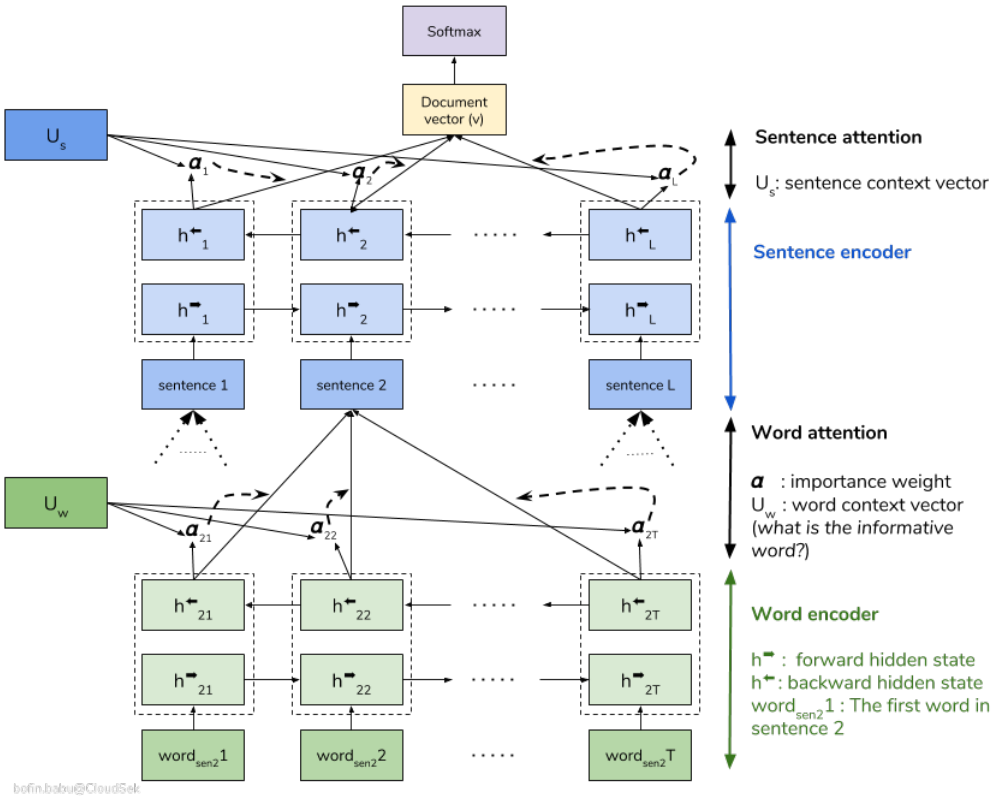


Figure 3.6. HAN architecture

The architecture builds sentence representations by first encoding the word of a sentence and then applying the attention mechanism on them resulting in a sentence vector. Document representation is built in the same way, however, it only receives the sentence vector of each sentence of the document as input.

Beginning at the word encoder, given a sentence with words w_{it}, $t \in [0, T]$, the words are embedded in to vectors through an embedding matrix $We, xij = W_{ewij}$. Thus, we input the raw data into the neural network through a lookup table (the authors use a word2vec to build W_e), representing them as input to a bidirectional GRU that contains the forward and backward steps that can read the sentence from w_{i1} to w_{iT} and from w_{iT} to w_{i1}. To summarize the information around a word w the forward and backward hidden states are concatenated. The word attention extracts words that are important to the meaning of the sentence and aggregate their representations to form a sentence vector. The same aforementioned process also happens at sentence level.

In the experiments we follow same preprocess of the datasets when building the input matrix vector, just words that appear more than 5 times are retained and the ones that appear 5 times are replaced by a special <UNK> token.

3.2.6 Summary

The neural networks VDCNN, CNN and HAN implementations are available online in the github repository[12] and BERT also [13]. We use their default parameters and the parameters we set throughout our study are in the Table 3.5.

Models	small
CNN	lr=0.01;momentum=0.9;gamma=0.9;maxlen=1014;epochs=100;batch size=128
VDCNN	lr=0.01;momentum=0.9;gamma=0.9;maxlen=1024;epochs=100;batch size=128;kernel_size=3;padding=1
HAN	lr=0.0005;momentum=0.9;gamma=0.9;epochs=100;batch size=16
BERT	batch_size=32;lr=5e-5;patience=5
big	
CNN	lr=0.001;momentum=0.9;maxlen=1014;epochs=18;batch size=128
VDCNN	lr=0.01;momentum=0.9;maxlen=1024;epochs=18;batch size=128;kernel_size=3;padding=1
HAN	lr=0.0005;momentum=0.9;gamma=0.9;maxlen=1024;epochs=100;batch size=16
BERT	batch_size=32;lr=5e-5;patience=5

Table 3.5. Neural Networks Parameters

3.2.7 Support Vector Machine - SVM

As the traditional machine learning algorithm we use the Support Vector Machine, which is SOTA when classifying texts and also very used in industry as it is easy to tune and model. It searches for the optimal hyperplane that splits the positive from the negative class, by maximizing the margin between the closest points from either class. SVM is inherently a binary classifier which implies in adopting approaches, such as one-versus-one or one-versus-all, in order to adapt binary SVM for multi-class classification tasks. Linear SVM is specially popular for text classification problems, since it is robust to high-dimensional data, being a top performer in such scenarios.

In this master thesis we use the scikit-learn[14] implementation of Linear SVM in an adapted implementation of Campos et al. [2017b][15] using tuning set of C parameter within training considering different values - (0.03, 0.13, 0.5, 2.0, 8.0, 32.0, 128.0) during fitting in 5-fold cross validation. This set of values were found to be better values for this process in Campos et al. [2017b].

[12] `https://github.com/ArdalanM/nlp-benchmarks`
[13] `https://github.com/yaserkl/`
[14] `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`
[15] `https://github.com/raphaelcampos/stacking-bagged-boosted-forests`

3.3 Experimental Setup

3.3.1 Tuning Process

As previously mentioned, we have tuned MLP and DMLP according to a set of hyperparameters using random search. During the optimization process, drawn in the Figure 3.7, we defined 80 trials of different sets of parameters, thus 80 different models were created per fold ($M1, M2, M3, M4, M5$) in order to find the best model.

The optimization process chooses the best set of parameters for each fold based in a cross-validation (CV) methodology. At the beginning of this process several parameters are given and they can be randomly chosen to create new models. We use a library called Hyperopt[16], which implements an algorithm called Tree of Parzen Estimators (TPE) Bergstra et al. [2013]. It basically searches for the best set of parameters that minimizes a function (loss) randomly.

In Figure 3.7 we illustrate how the process work. A dataset is split into 5 folds for training and 5 folds for testing in a stratified manner, which guarantees same distribution of classes in all folds. Then, for each training set a new cross-validation is executed for random searching the best set of values (we set 80 different trials). At the end, we measure the mean of the F1-macro (our loss function) of each CV across 80 trials and obtain the highest value to perform the final model.

[16] `http://hyperopt.github.io/hyperopt/`

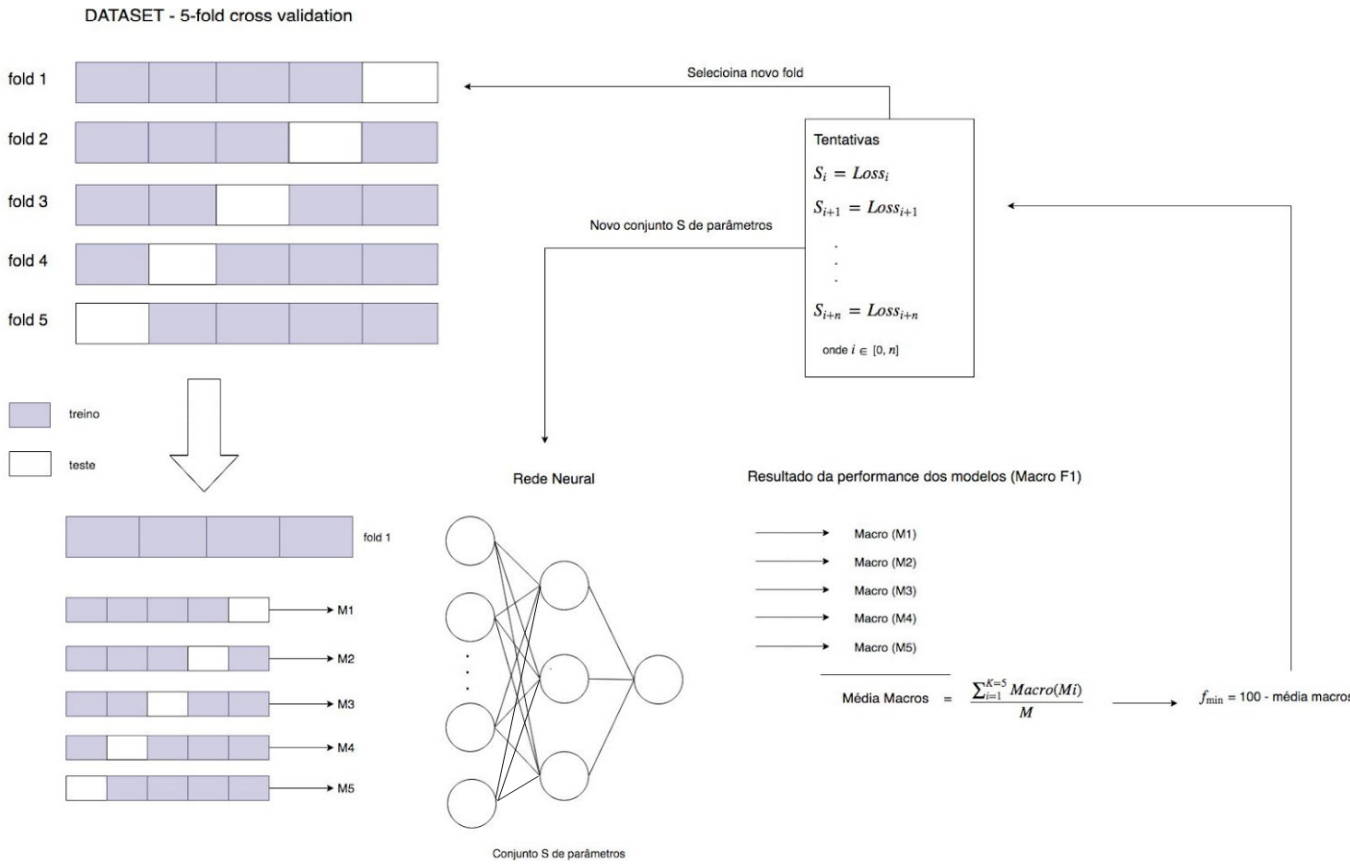


Figure 3.7. Tuning process of MLP and DMLP

3.3.2 Evaluation Metrics

All models are compared using *micro averaged* F_1 (MicroF_1), and *macro averaged* F_1 (MacroF_1), which are standard information retrieval measures Yang and Pedersen [1997]; Yang [1999]; Lewis et al. [1996]. Moreover, we use log loss in order to measure the confidence of each model. Following we explain each one of these metrics.

In order to further understand MicroF_1 and MacroF_1, let $C = \{c_1, c_2, ..., c_k\}$ be the set of classes of a given collection. For each class c_i, we can define the following values:

- True positives for c_i (TP_i): the number of test documents that the true class is c_i and that were classified as c_i
- True negatives for c_i (TN_i): the number of test documents that the true class is not c_i and that were not classified as c_i
- False positives for c_i (FP_i): the number of test documents that the true class is not c_i and that were classified as c_i

- False positives for c_i (FN_i): the number of test documents that the true class is c_i and that were not classified as c_i

Recall and precision can be assessed per class or globally. The recall $r(c_i)$ for a given class c_i is defined as:

$$r(c_i) = \frac{TP_i}{TP_i + FN_i}, \tag{3.1}$$

which means the fraction of test documents of class c_i that were correctly classified. The precision $p(c_i)$ for a given class c_i is given as

$$p(c_i) = \frac{TP_i}{TP_i + FP_i}, \tag{3.2}$$

which means the fraction of documents correctly classified from all documents attributed to the class c_i.

The global recall and precision are respectively defined as:

$$r(C) = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k}(TP_i + FN_i)} \tag{3.3}$$

$$p(C) = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k}(TP_i + TN_i)} \tag{3.4}$$

The F_1 measure combines the recall and precision metric by means of their harmonic mean. The MicroF$_1$ measures the classification effectiveness overall decisions, which is defined as:

$$MicroF_1 = 2\frac{p(C)r(C)}{p(C) + r(C)} \tag{3.5}$$

The MacroF$_1$ measures the classification effectiveness for each individual class and averages them, as follows:

$$MacroF_1 = \frac{\sum_{i=1}^{k} 2\frac{p(c_i)r(c_i)}{p(c_i)+r(c_i)}}{k} \tag{3.6}$$

MicroF_1 tends to be dominated by the classifiers performance on more frequent classes and the MacroF_1 is more influenced by the performance on rare ones.

The logarithmic loss or log loss is another metric we use to measure the quality of the models. It quantifies the accuracy of a classifier by penalizing false classifications. It takes into account the uncertainty of the prediction based on how much it varies from the actual label Collier [2014]. Its value increases as the predicted probability

diverges from the actual label, a perfect model would have log loss of 0. Figure 3.8 illustrates the range of possible values for log loss given a true observation.

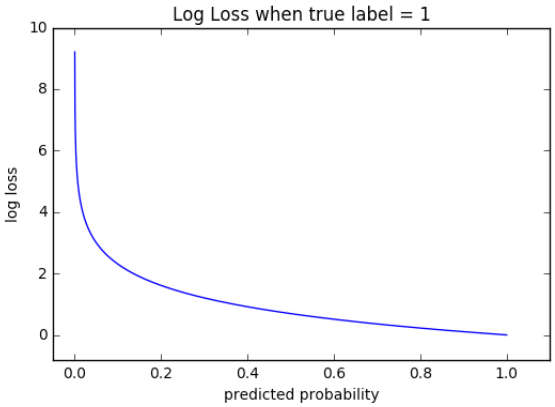


Figure 3.8. Log loss curve log [2017]

In order to calculate a log loss of a model we need to have the probability prediction for each class of each instance, thus we calculate as:

$$logloss = \sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}), \tag{3.7}$$

where N is a number of samples (or instances), M is a number of possible labels, y_{ij} takes value 1 if label j is the correct classification for instance i, and 0 otherwise and p_{ij} is the model probability of assigning label j to instance i.

We see that the log loss gradually declines as the predicted probability improves. Thus, log loss heavily penalizes classifiers that are confident about an incorrect classification.

To compare the average results of our k-fold cross-validation experiments, we assess their statistical significance by applying a paired (unpaired in some cases - explained in the results discussion) t-test with 95% confidence and Bonferroni correction. Also, we evaluate the comparison of all models using Friedman with posthoc Nemenyi test to assure differences among multiple methods for multiple comparisons of mean rank sums.

The Friedman test is used to compare multiple methods Zar [1999]. It is is a non-parametric statistical test similar to the parametric repeated measures ANOVA. It is also used to detect differences in treatments across multiple test attempts. In our case, the test procedure is as follows: for each fold of a dataset in the cross-fold validation, for all folds of all datasets, the best performing method is ranked first (1), the second-best gets rank 2, and so on. The methods are compared based on the average of the

assigned ranking across all folds. The $H0$ hypothesis (null-hypothesis) considers that all methods have the same average ranking position for a predefined p-value (i.e. 0.05), that is, there is no statistical difference across the methods. Moreover, as Connor [2004] highlights:

> [...] confidence intervals communicate much more information in a clear and efficient manner than those using p-values[...] which also prevents readers from drawing erroneous conclusions caused by common misunderstandings about p-values.

Aiming at overcoming such issue, we use a bootstrap technique Efron and Tibshirani [1994] to conduct a pairwise statistical comparison between the classifiers. We use 95% confidence intervals (CIs) for the differences in average performance between pairs of classifiers on the small and large datasets. Specifically, we implement the following steps:

1. Choose an algorithm to be the control level. We choose SVM as our control algorithm due to its performance and lowest cost.
2. For each method we:
    a) Take the average results computed on the datasets (4 small and 3 large) and subtract them from the results obtained with the SVM;
    b) Draw, with replacement, 10,000 samples of size 4 and 3 of small and large datasets, respectively, from the differences obtained in step 2(a);
    c) Take the mean of each bootstrap sample;
    d) Calculate limits of the 95% CI are the quantiles of 2.5% and 97.5% of the estimated means.

In summary, 95% confidence intervals containing the zero suggest small or no difference between algorithms, while strictly negative confidence intervals suggest average performance greater and strictly positive suggest smaller performance than the control algorithm, at a significance level of 0.05.

In addition to effectiveness, we also assess the cost of each method in terms of the training execution time aiming at analyzing the effectiveness-cost trade-offs in all methods. The metric is the overall time in seconds measured in the first fold of each classifier.

3.3.3 Dataset Splitting and Preprocessing

In our experiments we use the K-fold cross-validation, which consists of randomly splitting the data into K independent folds. At each iteration, one fold is retained as the test set, and the remaining $K - 1$ folds are used as training set. Figure 3.7 gives a practical example and further exemplifying, if we split a dataset into 5-fold cross-validation we obtain 5 different splits, where in each split has 80% of the data as training and 20% as testing.

Experiments on large datasets were executed using a 5-fold cross-validation procedure and in 10-fold cross-validation on small datasets (20NG, WEBKB, ACM and REUT). Also, we have executed 5-fold CV for MLP and DMLP due to tuning time consumption. When parameter tuning was necessary (e.g., BERT), nested cross-validation within the training set was performed.

MLP, DMLP and SVM classifiers had a term-frequency inverse document frequency (TFIDF) input for each corpus of data. The preprocessing strategy used was lowering case, removing of stop-words, filtering of words with frequency smaller than 1 document for small datasets and frequency smaller than 2 for big datasets. Specifically, for MLP and DMLP a feature selection was done in order to run these architectures in the GPU memory.

For feature selection (FS), we consider the importance score of Random Forests applied to the original features (TF-IDF) as the method of choice Louppe et al. [2013], at a reduction rate of 30%, established again in preliminary experiments in the validation sets. Those numbers were based on the largest reduction possible without hurting effectiveness. Only SOGOU dataset did not receive any further preprocessing because of the language (Chinese), thus TFIDF was generated without preprocessing of text.

The architectures VDCNN, CNN and HAN transformed the original datasets input to character representations, thus the datasets were inputted in their original form. Finally, for BERT, Glove[17] was used as pre-trained embedding according to words present in the original dataset.

[17] `https://nlp.stanford.edu/projects/glove/`

Chapter 4

Results and Discussion

4.1 Effectiveness of Different Deep Learning Architectures

In this section we describe the results of our experiments on the deep learning architectures, considering different metrics (F1-score and log loss). In the first subsections we describe experiments on VDCNN and CNN in order to analyze the best configuration among the ones originally presented by their authors. For VDCNN, we assess how deeper (more layers) architectures might affect the effectiveness and in CNN we study how the number of filters affect the model's performance.

4.1.1 Results on VDCNN

Tables 4.1 and 4.2 show results of VDCNN classification on small and large datasets, respectively. We evaluate three different configurations of the model, considering different depths: 5, 9 and 15 total layers. Our goal is to measure the impact of depth on the effectiveness.

		20NG	4UNI	ACM	REUT
VDCNN-5	microF1	68.95 ± 0.60	**78.82 ± 0.82**	**67.94 ± 0.80**	**62.82 ± 1.05**
	macroF1	68.91 ± 0.61	**68.04 ± 1.29**	**53.85 ± 1.42**	**20.73 ± 1.42**
VDCNN-9	microF1	73.60 ± 0.62	**78.78 ± 0.61**	**69.02 ± 0.96**	62.22 ± 1.03
	macroF1	73.35 ± 0.61	**68.07 ± 0.94**	**54.91 ± 1.47**	20.22 ± 1.20
VDCNN-15	microF1	**75.80 ± 0.90**	**79.30 ± 1.04**	**67.05 ± 4.35**	**63.27 ± 0.91**
	macroF1	**75.42 ± 0.88**	**68.25 ± 2.46**	**54.71 ± 4.58**	**21.76 ± 1.10**

Table 4.1. VDCNN F1-score Results on Small Datasets: 20NG, 4UNI and ACM

Each result in Table 4.1 represents the mean and standard deviation of a 10-fold cross-validation experiment. We can see that the results in 20NG improve significantly (around 7 percentage points) in the deeper architecture when compared to shallow

ones. This is not observed in the other datasets. In some of them (ACM and 4UNI) there was even an increase in the deviation. This happens because of the size of these datasets. They have few instances and the network can not generalize well. The largest improvements in 20NG may be due to the large average document size (more words per documents), the largest among the small datasets.

In Table 4.2 where the results for large datasets are shown, we also see no significant enhancement in the use of deeper architectures. Conneau et al. [2017] affirm that the increase in the depth improved performance, but their results were not backed up by statistical significance tests. What we actually see in our experiments is an overall tie, mainly due to the large confidence intervals (high variability).

			AGNEWS	SOGOU	YELP-FULL
Mean 5-fold	VDCNN-5	microF1	**85.53 ± 3.16**	94.86 ± 0.11	**59.16 ± 1.11**
		macroF1	**85.70 ± 3.00**	94.86 ± 0.12	**59.22 ± 1.02**
	VDCNN-9	microF1	**86.89 ± 2.66**	**95.15 ± 0.49**	**61.18 ± 0.74**
		macroF1	**86.93 ± 2.38**	**95.15 ± 0.49**	**61.22 ± 0.62**
	VDCNN-15	microF1	**87.26 ± 3.32**	**95.34 ± 0.30**	**61.78 ± 1.02**
		macroF1	**87.22 ± 3.42**	**95.35 ± 0.30**	**61.52 ± 1.36**
Original	VDCNN-5	microF1	89.22	95.00	60.36
		macroF1	89.15	95.00	59.75
	VDCNN-9	microF1	88.39	95.55	62.06
		macroF1	88.45	95.55	61.19
	VDCNN-15	microF1	89.21	95.75	62.08
		macroF1	89.17	95.76	62.58

Table 4.2. VDCNN F1-score Results on Big Datasets: REUT90, SPAM and MED

Results on the original splits of the large datasets from Zhang et al. [2015a] are also reported on Table 4.2. Although our version of VDCNN contains half of the layers for each depth, the results are highly compatible to the original ones.

Although microF$_1$ and macroF$_1$ results show no improvements and suggest that deeper architectures do not help, Table 4.3 shows some improvements in the results of models considering the log loss metric. We see larger values for small datasets, due to the size of the dataset, and better (smaller) values for large datasets, mainly in SOGOU, with the best values. When the depth increases, the log loss decreases for 20NG, AGNEWS, SOGOU and YELP, demonstrating that the model is more assertive regarding to the right label as the depth increases.

	20NG	4UNI	ACM	REUT	AGNEWS	SOGOU	YELP
VDCNN-5	2.48 ± 0.07	**1.38 ± 0.08**	**2.40 ± 0.26**	**1.84 ± 0.08**	0.62 ± 0.18	0.24 ± 0.01	1.06 ± 0.05
VDCNN-9	2.21 ± 0.08	1.59 ± 0.11	**2.34 ± 0.16**	2.05 ± 0.11	0.49 ± 0.09	0.21 ± 0.03	**0.95 ± 0.04**
VDCNN-15	**2.07 ± 0.09**	1.65 ± 0.06	**2.74 ± 1.09**	1.93 ± 0.07	**0.40 ± 0.08**	**0.18 ± 0.01**	**0.92 ± 0.04**

Table 4.3. Log loss of VDCNN models

We can see by mean of this analysis of VDCNN that there is no significant improvements in deeper CNNs on most of the datasets. From what is known in the

literature Le et al. [2018] and the results presented, our main conclusion is that deep models have not yet proven to be more effective than shallow models for text classification tasks. In the comparison of all methods we use the model VDCNN-15 because it was the best in log-loss for most datasets.

4.1.2 Results on CNN

We evaluate CNN on the same datasets aiming at measuring the impact of the number of filters on the effectiveness. Tables 4.4 and 4.5 report results on small and large datasets respectively. As mentioned we evaluate a "small" CNN Zhang et al. [2015b] with 256 filters, and a large one with 1024 filters. The more filters, the more information we have about the inputs.

The results on Table 4.4 were computed running 100 epochs and halving the learning rate of the model at each 15 epochs in order to avoid falling into local minima. We clearly see improvements from CNN-small to CNN-large.

		20NG	4UNI	ACM	REUT
CNN-small	microF1	26.95 ± 13.48	52.25 ± 6.02	45.96 ± 2.62	46.27 ± 6.36
	macroF1	25.00 ± 15.07	26.51 ± 10.30	33.60 ± 2.10	5.72 ± 2.02
CNN-large	microF1	**55.02 ± 2.53**	**69.48 ± 2.49**	**51.61 ± 1.54**	**52.18 ± 1.10**
	macroF1	**55.64± 2.40**	**53.66 ± 2.99**	**39.19 ± 1.16**	**8.96 ± 0.89**

Table 4.4. CNN F1-score Results on Small Datasets: 20NG, 4UNI and ACM

Due to small amount of filters applied to the small datasets, CNN-small presents poor results and large deviations. The combination of few instances and filters do not help the neural network to capture the most important features in the sentences. Thus, for small datasets this architecture does not perform well, mainly due to the huge number of weights it tries to update during training, without enough data.

This architecture is very sensitive to some parameters. We have experimented with different number of starting learning rates and epochs, noticing that some results may vary a lot depending on the set of hyperparameters. This happens because the neural network fall in a local minima during training, stopping the learning process. Thus, there is a trade-off between time and model quality. If we use few epochs and a small learning rate, or half the learning rate down every N epochs, for example, the results might improve. However the time for training also gets higher.

The results on large datasets are reported in the Table 4.5. In the rows named Original we show the authors results in each dataset. We have selected a suitable configuration for the experiments based on these trade-offs. We fixed the learning rate at 0.001 and set the number of epochs to 18 in order to avoid long training times.

			AGNEWS	SOGOU	YELP-FULL
Mean 5-fold	CNN-small	microF1	83.80 ± 1.08	92.03 ± 1.09	**56.50 ± 0.73**
		macroF1	67.00 ± 0.86	92.08 ± 1.04	**56.16 ± 1.34**
	CNN-large	microF1	**88.60 ± 0.47**	**94.23 ± 0.44**	**56.27 ± 1.20**
		macroF1	**88.57 ± 0.48**	**94.23 ± 0.44**	**56.51 ± 1.03**
Original	CNN-small	microF1	88.12	93.65	55.96
		macroF1	88.07	93.64	54.08
	CNN-large	microF1	88.68	95.06	56.88
		macroF1	88.67	95.06	56.91

Table 4.5. CNN F1-score Results on Big Datasets: AGNEWS, SOGOU and YELP

The results above show that CNN-large is significantly better than CNN-small on AGNEWS and SOGOU and also that results in the mean 5-fold are fairly agreeing to original results.

The results considering log loss of CNN models are shown in Table 4.6. The same pattern of high values for small datasets and smaller values for large datasets is seen. The best values on large datasets come from CNN-small. We believe that this happens because of the overfitting with many more filters.

	20NG	4UNI	ACM	REUT	AGNEWS	SOGOU	YELP
CNN-small	**4.17 ± 0.83**	**2.02 ± 0.43**	6.65 ± 0.65	2.51 ± 0.16	**0.45 ± 0.02**	**0.26 ± 0.04**	**1.00 ± 0.02**
CNN-large	**4.39 ± 0.44**	**2.03 ± 0.27**	**5.15 ± 0.28**	**2.26 ± 0.09**	0.60 ± 0.02	0.34 ± 0.05	2.03 ± 0.06

Table 4.6. Log loss of CNN models

To summarize, basically, CNN-small and CNN-large do not perform well in the small datasets, differently from what happens in the large ones. This is due to the amount of data available for training. In the comparison with all methods, we use CNN-large due to its higher $F1$-scores.

4.1.3 Model Learning Analysis

In this section we analyze deep learning methods during training. This is important to understand issues related to variability and convergence.

We show MLP and DMLP results aiming at having a baseline for a simple architecture. As we have previously discussed, MLP and DMLP are two very simple architectures and their input is basically a feature selection of the original TFIDF. Their architecture is very sensitive to parameters and depending on the dataset, some folds might have poor results and overfit very much, as shown in Figure 4.1 (y-axis is the training accuracy and x-axis represents the epochs) and in the results shown in the Table 4.7 with high variability. Even though we have used many techniques (dropout, regularization and early stopping) to prevent the overfitting, with so few data is hard not to overfit.

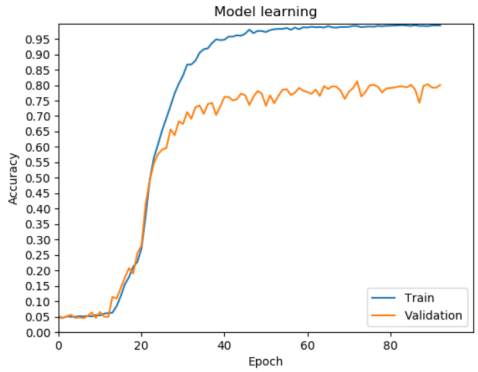


Figure 4.1. DMLP 20NG learning curve

Figures 4.2 and 4.3 show learning curves of HAN, VDCNN-15 and CNN-large models for 20NG and AGNEWS, respectively. In the y axis we show the accuracy and in the x axis the number of epochs, 100 for smaller datasets and 18 for bigger datasets. The lines in blue correspond to the training accuracy and the ones in orange to testing accuracy (as we do not have validation set up for tuning the models for these architectures).

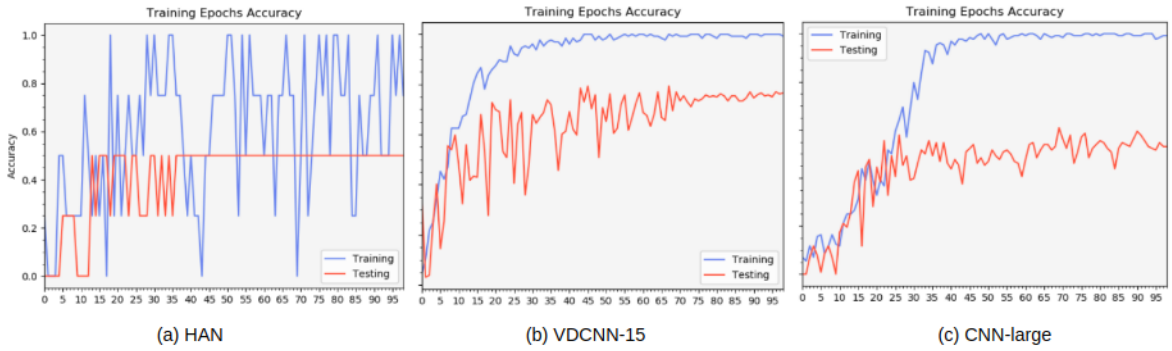
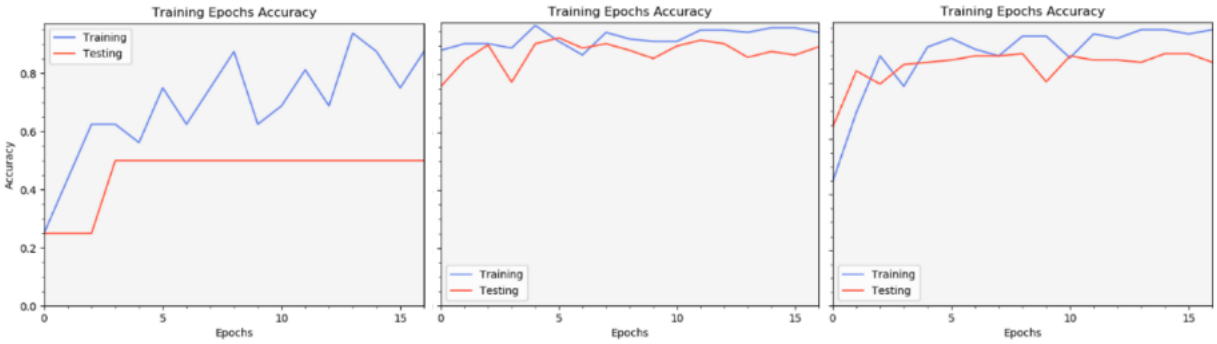


Figure 4.2. Learning curves on 20NG

In Figure 4.2 we can see that the models overfit a lot. This is expected from deep learning models when trained on small datasets, because there is not enough data points for the neural network to generalize. Figure 4.2 (a) represents the learning curve of the HAN model. We observe that there is a very high oscillation of the learning process during training and most of the time the testing accuracy stays still. We believe that this happens mainly because of the way the architecture was constructed. Each time HAN pays attention to a specific part of a word or sentence is where oscillation occurs. This fluctuation is more predominant in the 20NG because of the few instances, but we see a similar behavior on AGNEWS.

The plots in Figures 4.3 (a), (b) and (c) illustrate the learning curve for HAN, VDCNN-15 and CNN-large on AGNEWS. They also show some overfitting of the

model, but much less predominant than in small datasets.



(b) HAN (b) VDCNN-15 (c) CNN-large

Figure 4.3. VDCNN-15 and CNN-large learning curves on AGNEWS

AGNEWS has 91872 training and 25520 testing samples. Hence, as the images illustrates, with more data it is possible to reduce the overfitting in these architectures.

In summary, we clearly see that the deep learning classifiers we study in this dissertation have poor learning curves for small datasets, which explains high log loss and poor performance. The learning curves for large datasets are much more stable and also confirms that the dataset size is crucial to the performance mostly due to better generalization.

4.1.4 Effectiveness of the Models

We here compare the effectiveness of all deep learning methods. We use as baseline SVM, as described in Section 3.2.7. This machine learning algorithm was chosen due to its very robust and efficient text classification capacity as it is shown in Rashid et al. [2017]; Campos et al. [2017a].

We can see the results on small datasets in Table 4.7. We highlight that the results shown in this comparison were calculated with the mean of 10-fold cross validation in all methods but MLP and DMLP (5-fold) due to their high cost for tuning. Thus, we perform a unpaired statistical test for this analysis. We can notice that SVM ties with BERT, MLP and DMLP in 4UNI (macro$F1$) and with MLP in REUT (macro$F1$). SVM is also the single best overall method in 20NG. In the most unbalanced dataset (REUT) BERT, MLP and DMLP tie in micro$F1$ beating SVM.

		20NG	4UNI	ACM	REUT
SVM	microF1	**90.43 ± 0.57**	82.40 ± 0.76	**78.89 ± 0.56**	66.72 ± 0.80
	macroF1	**90.15 ± 0.60**	**70.57 ± 1.79**	**67.00 ± 1.46**	**33.20 ± 2.19**
CNN-large	microF1	55.02 ± 2.53	69.48 ± 2.49	51.61 ± 1.54	52.18 ± 1.10
	macroF1	55.64 ± 2.40	53.66 ± 2.99	39.19 ± 1.16	8.96 ± 0.89
VDCNN-15	microF1	75.80 ± 0.90	79.30 ± 1.04	67.05 ± 4.35	63.27 ± 0.91
	macroF1	75.42 ± 0.88	68.25 ± 2.46	54.71 ± 4.58	21.76 ± 1.10
HAN	microF1	67.54 ± 0.63	61.52 ± 1.48	66.75 ± 0.77	64.59 ± 1.01
	macroF1	66.19 ± 0.70	31.24 ± 1.86	50.00 ± 0.79	8.07 ± 0.60
BERT	microF1	84.54 ± 0.90	**85.22 ± 0.90**	71.80 ± 1.05	**70.62 ± 2.19**
	macroF1	84.04 ± 1.00	**72.24 ± 1.90**	53.37 ±2.47	19.31 ±3.08
∗ MLP	microF1	85.62 ± 2.20	81.18 ± 1.64	**74.63 ± 2.33**	**70.90 ± 3.25**
	macroF1	85.29 ± 2.28	**73.45 ± 4.48**	58.14 ± 1.69	**32.56 ± 3.55**
∗ DMLP	microF1	77.97 ± 4.11	79.73 ± 2.48	73.34 ± 1.06	**70.64 ± 1.03**
	macroF1	77.76 ± 4.02	**69.94 ± 5.29**	60.94 ± 1.31	29.54 ± 4.24

Table 4.7. Micro$F1$ and Macro$F1$ results on Small Datasets with standard deviation. Results in bold are the best (with ties) according to the statistical tests. Methods with ∗ were executed in 5-fold cross-validation

	20NG		ACM		REUT		WEBKB		Cross datasets	
	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro
BERT	2	2	2.9	2.2	2.7	1.2	1.3	1	2.2	1.6
VDCNN	3	3	2.5	3.1	2.3	3.8	2.7	3	2.6	3.2
CNN	5	5	4.9	4.9	4.3	5	4	4	4.5	4.7
HAN	4	4	3.7	3.8	4.7	2.9	5	5	4.3	3.9
SVM	1	1	1	1	1	2.1	2	2	1.2	1.5

Table 4.8. Intra and Cross-Dataset Ranking Comparison of Methods based on the Friedman ranking on small datasets

In Friedman-Nemenyi test we must consider the same number of folds for all methods, thus in the Table 4.8 we consider all methods but MLP and DMLP, due to their different split (5-fold, different from all the other methods). Although these architectures showed competitive results, from our point of view their tuning time makes them impractical methods.

As we can see in the Table 4.8 SVM is overall the best classifier considering both intra-dataset and cross-dataset analysis in most small datasets, closely followed by BERT. We also perform the confidence interval analysis in Table 4.9, with 95% CIs for the differences in average performance for macro$F1$ and micro$F1$. Table 4.9 shows the confidence intervals for each classifier pairwised with SVM describing the minimum and maximum values for each metric.

Classifier	Macro		Micro	
SVM-BERT	2.13	12.75	-2.79	6.93
SVM-CNN	17.78	33.04	13.30	39.09
SVM-HAN	18.59	35.80	7.39	22.19
SVM-VDCNN	4.96	13.91	3.62	13.14
SVM-MLP	-2.17	6.68	-1.97	4.78
SVM-DMLP	1.09	9.80	-1.84	9.87

Table 4.9. Bootstrapped 95% confidence intervals of the differences in average performance between SVM and the other deep learning classifiers on small datasets.

The confidence intervals of SVM-MLP (micro$F1$ and macro$F1$), SVM-BERT and SVM-DMLP (macro$F1$) contain zero, then we can see that SVM ties with MLP for both metrics and with BERT and DMLP for macro$F1$, what confirms the analysis in the Table 4.7.

Regarding the large datasets, in Table 4.10 we can see that SVM outperforms CNN-large and HAN, and ties with VDCNN and BERT in AGNEWS. In SOGOU the results of SVM are very close to the best, but in YELP, SVM shows poor results when compared to the other methods. Zhang et al. [2015a] state that traditional methods as SVM, remain strong methods for data sets up to several hundreds of thousands of documents and when the datasets get bigger neural networks start to perform better. These results show evidence of that.

		AGNEWS	SOGOU	YELP
SVM	microF1	89.19 ± 0.21	92.35 ± 0.23	49.69 ± 1.13
	macroF1	**89.16 ± 0.22**	92.35 ± 0.23	49.17 ± 1.06
CNN-large	microF1	88.60 ± 0.47	94.23 ± 0.44	56.27 ± 1.20
	macroF1	88.57 ± 0.48	94.23 ± 0.44	56.51 ± 1.03
VDCNN-15	microF1	**87.26 ± 3.32**	**95.34 ± 0.30**	**61.78 ± 1.02**
	macroF1	**87.22 ± 3.42**	**95.35 ± 0.30**	**61.52 ± 1.36**
HAN	microF1	85.19 ± 0.36	**95.46 ± 0.52**	59.11 ± 0.19
	macroF1	85.10 ± 0.37	**95.46 ± 0.52**	58.88 ± 0.32
BERT	microF1	**90.54 ±0.59**	**94.46 ± 0.20**	**63.27± 0.20**
	macroF1	**90.47 ±0.65**	**94.46 ± 0.20**	**62.88 ± 0.20**

Table 4.10. Micro$F1$ and Macro$F1$ results on Large Datasets. Results in bold are the best (with ties) according to the statistical tests.

In the analysis of the results on large datasets with bootstrap strategy we can see in the Table 4.11 that taking SVM as the control algorithm we confirm that all deep learning architectures are tied with SVM with 95% certainty. We also analyze with BERT as the control classifier (shown as the best classifier on the large datasets). Results in Table 4.12, confirm this claim. BERT is the best performer despite its similar effectiveness when compared to HAN and VDCNN (the interval includes the value zero).

Classifier	Macro		Micro	
SVM - BERT	-13.7	-1.306	-13.552	-1.352
SVM - CNN	-7.344	0.59	-6.574	0.59
SVM - HAN	-9.71	4.06	-9.418	4
SVM - VDCNN	-12.35	1.94	-12.082	1.928

Table 4.11. Bootstrapped 95% confidence intervals of the differences in average performance between the control algorithm SVM and the other deep learning classifiers on large datasets.

Classifier	Macro		Micro	
BERT - CNN	0.234	6.356	0.226	6.978
BERT - HAN	-0.994	5.366	-0.998	5.352
BERT - SVM	1.306	13.7	1.352	13.552
BERT - VDCNN	-0.884	3.246	-0.88	3.28

Table 4.12. Bootstrapped 95% confidence intervals of the differences in average performance between the control algorithm BERT and the other classifiers on large datasets.

	AGNEWS		SOGOU		YELP		Cross datasets	
	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro
BERT	1	1	3.4	3.4	1	1	1.3	1.3
VDCNN	3.6	3.6	1.8	1.8	2	2	1.8	1.8
CNN	3.4	3.4	3.6	3.6	4	4	2.7	2.7
HAN	4.8	4.8	1.2	1.2	3	3	2.2	2.2
SVM	2.2	2.2	5	5	5	5	3	3

Table 4.13. Intra and Cross-Dataset Ranking Comparison of Methods based on the Friedman ranking on large datasets

In Table 4.13 we can see that BERT is the best in all datasets but SOGOU (HAN is the best followed by VDCNN). SVM is in last place of the ranking for SOGOU and YELP and second in AGNEWS. In the cross-dataset evaluation we notice BERT as the best for micro$F1$ and macro$F1$, with VDCNN as the second and SVM as the last method.

To summarize, BERT and VDCNN show better performance on large datasets and SVM on small datasets. This is confirmed in all analysis we have completed.

4.2 Time Consumption Analysis

In the computation of time consumption we consider the total time spent for training, tuning and testing each dataset[18]. BERT and SVM spent most of the the total time for tuning while the other classifiers took more time in the training process itself.

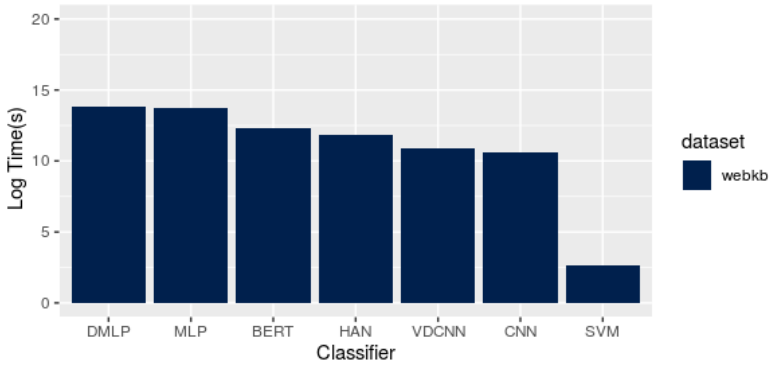
Figures 4.4 and 4.5 plot graphs showing the logarithm of total time spent in each method per each dataset. Specifically, we can see in Figure 4.4 that the DMLP and MLP are the classifiers that take the longest time in all small datasets, having BERT with the third largest time. HAN, VDCNN and CNN show similar time and SVM is the best performer. In Figure 4.5 we can also see that BERT takes the longest times on large datasets, also showing SVM as the best performer.

Further analyzing the time consumption we plot the trade-off of effectiveness and time cost for each dataset in Figures 4.6 and 4.7 for small and large datasets,

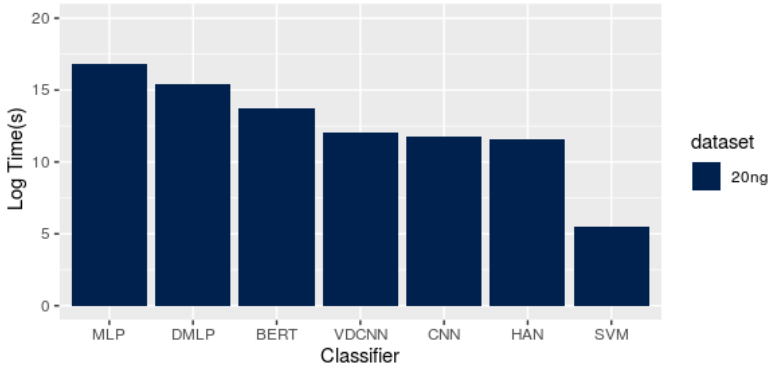
[18]Total time correspond to one fold due to the cost of the experiments, but variability is low and magnitude of times is similar.

respectively. In the y-axis we show the macro$F1$ and in the y-axis the log of the time in seconds.

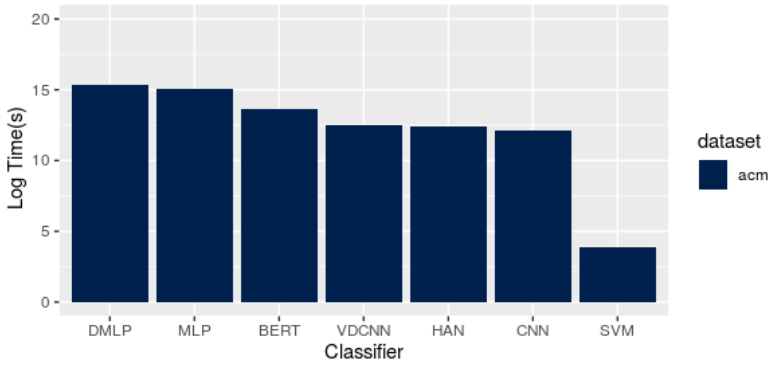
In Figure 4.6 we can see that SVM is the best when compared to the other methods in all small datasets. We can see easily how SVM loses its strength on the large datasets in Figure 4.7 as the size of the dataset increases. Also, we can see that BERT is clearly the worst time, but the best in macro$F1$ in AGNEWS and YELP and that SVM is the best in time, but the worst in macro$F1$ (AGNEWS and YELP).



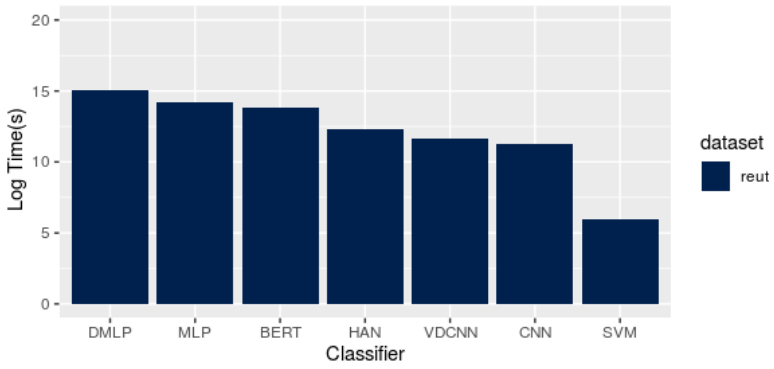
(a) Webkb



(b) 20NG

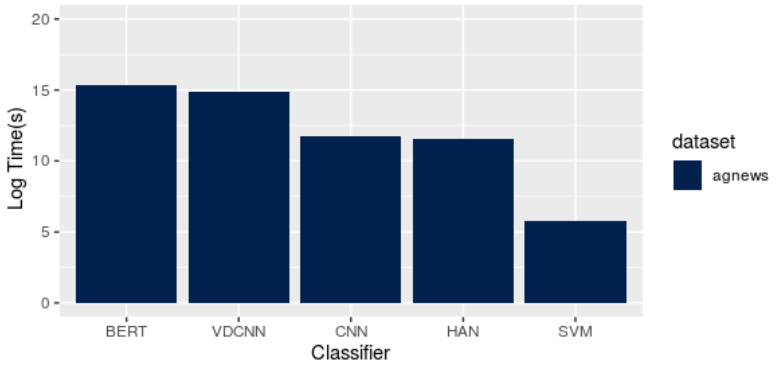


(c) ACM

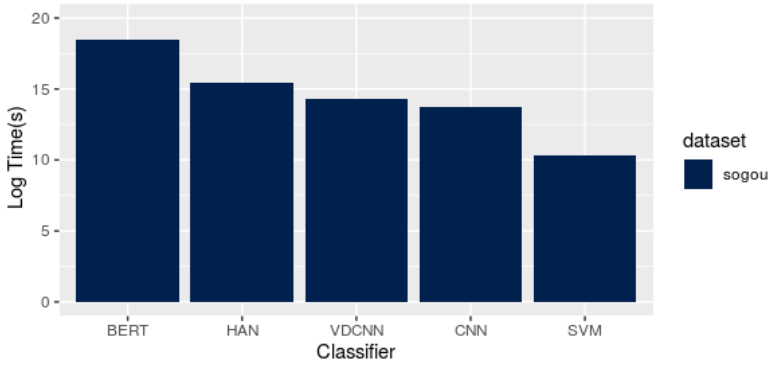


(d) REUT

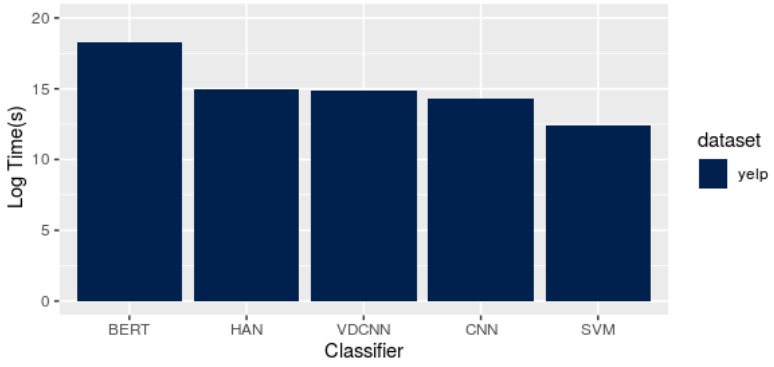
Figure 4.4. Methods Time for each Small Datasets



(a) Webkb

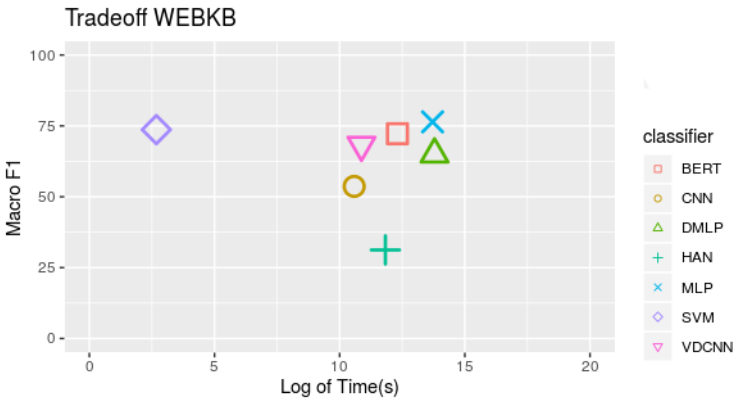


(b) 20NG

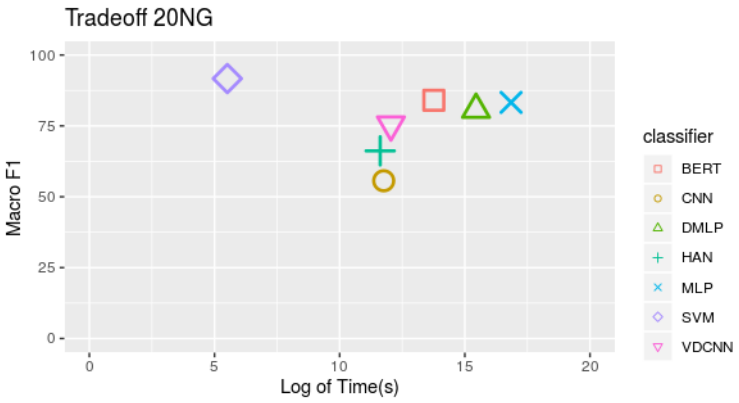


(c) ACM

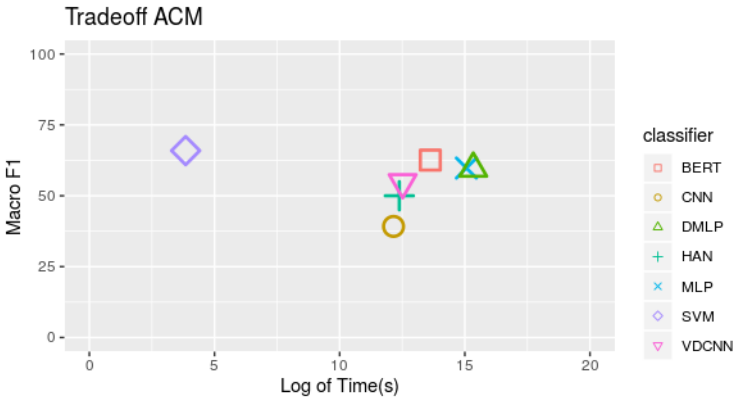
Figure 4.5. Methods Time for each Large Datasets



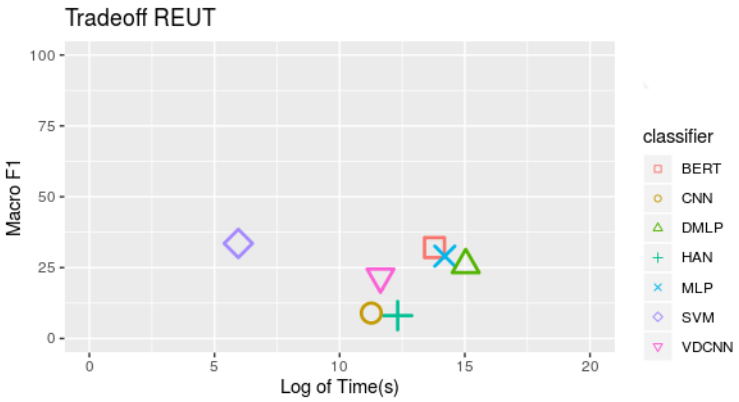
(a) Webkb



(b) 20NG

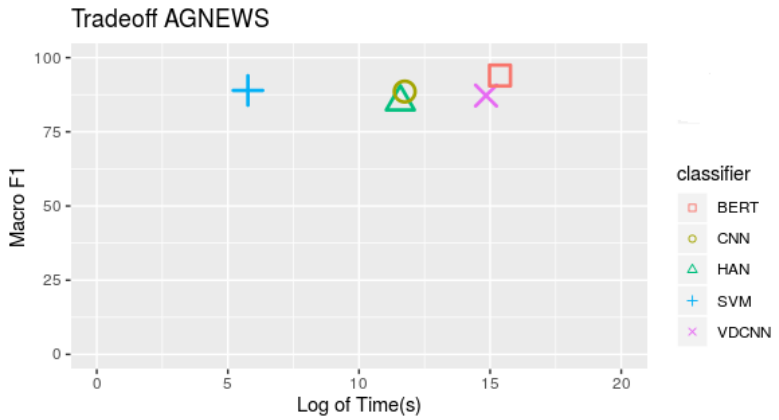


(c) ACM

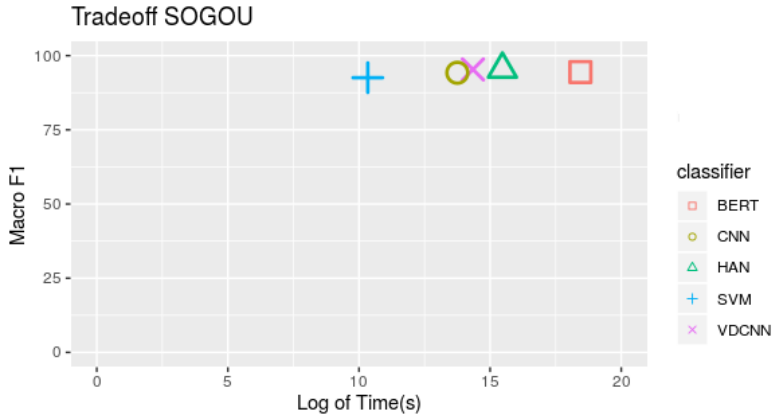


(d) REUT

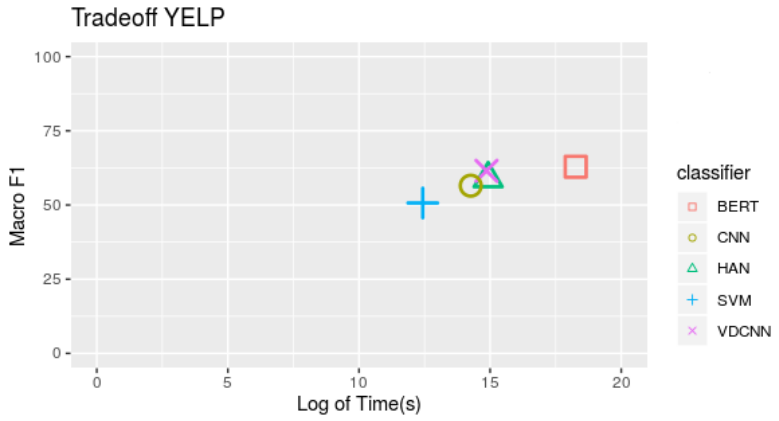
Figure 4.6. Effectiveness-cost Trade-off for each Small Datasets



(a) AGNEWS



(b) SOGOU



(c) YELP

Figure 4.7. Effectiveness-cost Trade-off for each Large Datasets

In summary, we can observe that SVM is by far the best of all methods in the small datasets. We can also observe that BERT is much more time consuming than the other methods. HAN and VDCNN show similar (high) timing. Although BERT is among the best classifiers in terms of effectiveness, it is the worst in terms of

training efficiency, with a training time that for some applications may be considered prohibitive. SVM still keeps its good trade-off in all datasets.

Chapter 5

Conclusions and Future Work

In this chapter we summarize the research contributions of this master thesis and point out some directions for future work.

5.1 Summary and Implications

We studied some of the most recent deep learning architectures in the literature, that classify text using character and word level features. We introduced some of the most important works and pointed out several flaws that occur in their evaluation. Aiming at overcoming some of the lack of scientific rigour in a significant portion of recent work and potential unfair comparisons, we explicitly stated our research questions which we tried to properly answer.

Specifically, in our analysis we have seen and confirmed that deep learning methods are powerfull on large datasets and their performance on small datasets is behind traditional methods such SVM. BERT led to improvements on classification performance in large datasets but not by a large margin when compared to the other architectures. In fact, VDCNN was shown to perform similar to BERT. Additionally we show that deep learning architectures do not perform well on small datasets due to generalization issues.

We also show that deeper architectures such as VDCNN do not improve performance significantly by adding more layers as previously shown in other works. Finally, we illustrate the trade-off effectiveness-cost and show that the very recent methodologies are not efficient, what corroborates to the reason why most of the industry do not consider such approaches in real applications.

5.2 Future Work

Our study is by no means exhaustive. We have not evaluated different parameters in the recent deep learning architectures besides the reported values by the original authors. Thus, we want to consider in future work in order to further analyze their impact. Also, it might be interesting to use other strategies to achieve better performance such as ensembles of features generated from each architecture.

Bibliography

(2017). Log loss. http://wiki.fast.ai/index.php/Log_Loss. [Online; accessed 2019-11-21].

Aggarwal, C. C. and Zhai, C. (2012). A survey of text classification algorithms. *Mining text data*, pages 163--222.

Alammar, J. (2018). Visualizing a neural machine translation model. https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attentior [Online; accessed 2019-10-20].

Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., and Kochut, K. (2017). A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*.

Arumugam, R. and Shanmugamani, R. (2018). *Hands-On Natural Language Processing with Python: A practical guide to applying deep learning architectures to your NLP applications*. Packt Publishing Ltd.

Attoh-Okine, N. O. and Ayyub, B. M. (2005). *Applied research in uncertainty modeling and analysis*. Springer.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137--1155.

Bengio, Y., Simard, P., Frasconi, P., et al. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157--166.

Bergstra, J., Yamins, D., and Cox, D. D. (2013). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, page 20. Citeseer.

Bonaccorso, G., Fandango, A., and Shanmugamani, R. (2018). *Python: Advanced Guide to Artificial Intelligence: Expert machine learning systems and intelligent agents using Python*. Packt Publishing Ltd.

Campos, R., Canuto, S., Salles, T., de Sá, C. C., and Gonçalves, M. A. (2017a). Stacking bagged and boosted forests for effective automated classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 105--114. ACM.

Campos, R., Canuto, S., Salles, T., de Sá, C. C., and Gonçalves, M. A. (2017b). Stacking bagged and boosted forests for effective automated classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 105--114, New York, NY, USA. ACM.

Chiu, C., Sainath, T. N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., et al. (2017). State-of-the-art speech recognition with sequence-to-sequence models. *CoRR*, abs/1712.01769.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Collier, A. B. (2014). Making sense of logarithmic loss. `https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/`. [Online; accessed 2019-09-01].

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493--2537.

Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. (2017). Very deep convolutional networks for text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 1107--1116.

Connor, J. T. (2004). The value of a p-valueless paper. *American Journal of Gastroenterology*, 99(9):1638--1640.

Crammer, K., Dredze, M., and Pereira, F. (2012). Confidence-weighted linear classification for text categorization. *Journal of Machine Learning Research*, 13(Jun):1891--1926.

Dacrema, M. F., Cremonesi, P., and Jannach, D. (2019). Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 101--109. ACM.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dong, X., Shen, J., Wang, W., Liu, Y., Shao, L., and Porikli, F. (2018). Hyperparameter optimization for tracking with continuous deep q-learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 518--527.

Efron, B. and Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

Feng, X., Jiang, Y., Yang, X., Du, M., and Li, X. (2019). Computer vision algorithms and hardware implementations: A survey. *Integration*.

Fontaine, A. (2018). *Mastering Predictive Analytics with Scikit-learn and TensorFlow: Implement Machine Learning Techniques to Build Advanced Predictive Models Using Python*. Packt Publishing. ISBN 178961774X, 9781789617740.

Foster, D. (2019). *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. O'Reilly Media.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675--701.

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86--92.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193--202.

Ganju, S. (2017). Contouring learning rate to optimize neural nets description. `https://www.oreilly.com/ideas/`

contouring-learning-rate-to-optimize-neural-nets. [Online; accessed 2018-09-30].

Géron, A. (2018). *Neural networks and deep learning*. O'Reilly.

Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1--309.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., and Lew, M. S. (2016). Deep learning for visual understanding: A review. *Neurocomputing*, 187.

Hassan, A. and Mahmood, A. (2017a). Deep learning approach for sentiment analysis of short texts. In *Control, Automation and Robotics (ICCAR), 2017 3rd International Conference on*, pages 705--710. IEEE.

Hassan, A. and Mahmood, A. (2017b). Deep learning approach for sentiment analysis of short texts. In *ICCAR IEEE*.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochberg, Y. (1988). A sharper bonferroni procedure for multiple tests of significance. *Biometrika*, 75(4).

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554--2558.

Hubel, D. H. and Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574--591.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, pages 137--142.

K., H., M., S., and H., W. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5).

Kakade, A., Dhumal, K., Das, S., Jain, S., and Ranjan, N. (2017). A neural network approach for text document classification and semantic text analytics. *Journal of Data Mining and Management*, 2(2).

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kannan, A., Wu, Y., Nguyen, P., Sainath, T. N., Chen, Z., and Prabhavalkar, R. (2018). An analysis of incorporating an external language model into a sequence-to-sequence model. *ICASSP*.

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

Kohler, J., Daneshmand, H., Lucchi, A., Zhou, M., Neymeyr, K., and Hofmann, T. (2018). Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. *arXiv preprint arXiv:1805.10694*.

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

Kulkarni, A. and Shivananda, A. (2019). *Natural Language Processing Recipes: Unlocking Text Data with Machine Learning and Deep Learning Using Python*. Springer.

Le, H. T., Cerisara, C., and Denis, A. (2018). Do convolutional networks need to be deep for text classification? In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188--1196.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10).

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278--2324.

Lee, H., Pham, P., Largman, Y., and Ng, A. Y. (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096--1104.

Lewis, D. D., Schapire, R. E., Callan, J. P., and Papka, R. (1996). Training algorithms for linear text classifiers. In *SIGIR*, volume 96, pages 298--306.

Lin, H. W., Tegmark, M., and Rolnick, D. (2017). Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168.

Lin, J. (2019). The neural hype and comparisons against weak baselines. In *ACM SIGIR Forum*, volume 52, pages 40--51. ACM.

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Louppe, G., Wehenkel, L., Sutera, A., and Geurts, P. (2013). Understanding variable importances in forests of randomized trees. In *Advances in neural information processing systems*, pages 431--439.

Ludewig, M., Mauro, N., Latifi, S., and Jannach, D. (2019). Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 462--466. ACM.

Manne, S., Kotha, S., and Sameen Fatima, S. (2012). Text categorization with k-nearest neighbor approach. In *Proceedings of the International Conference on Information Systems Design and Intelligent Applications 2012 (INDIA 2012) held in Visakhapatnam, India, January 2012*, pages 413--420. Springer.

Masci, J., Giusti, A., Ciresan, D., Fricout, G., and Schmidhuber, J. (2013). A fast learning algorithm for image segmentation with max-pooling convolutional networks. In *2013 IEEE International Conference on Image Processing*, pages 2713--2717. IEEE.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115--133.

Medsker, L. and Jain, L. (2001). Recurrent neural networks. *Design and Applications*, 5.

Melis, G., Dyer, C., and Blunsom, P. (2017). On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*.

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

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

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013c). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111--3119.

Pal, A. and Prakash, P. (2017). *Practical Time Series Analysis: Master Time Series Data Processing, Visualization, and Modeling using Python*. Packt Publishing Ltd.

Patterson, J. and Gibson, A. (2017). *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.".

Pavlakos, G., Zhou, X., Derpanis, K. G., and Daniilidis, K. (2017). Harvesting multiple views for marker-less 3d human pose annotations. *CoRR*, abs/1704.04793.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532--1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Raina, R., Madhavan, A., and Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873--880. ACM.

Rashid, T. A., Mustafa, A. M., and Saeed, A. (2017). A robust categorization system for kurdish sorani text documents. *Information Technology Journal*, 16(1):27--34.

Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512--519. IEEE.

Ribeiro, M., Lazzaretti, A. E., and Lopes, H. S. (2018). A study of deep convolutional auto-encoders for anomaly detection in videos. *Pattern Recognition Letters*, 105. Machine Learning and Applications in Artificial Intelligence.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Salles, T., Gonçalves, M., Rodrigues, V., and Rocha, L. (2015a). Broof: Exploiting out-of-bag errors, boosting and random forests for effective automated classification. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 353--362. ACM.

Salles, T., Gonçalves, M., Rodrigues, V., and Rocha, L. (2015b). Broof: Exploiting out-of-bag errors, boosting and random forests for effective automated classification. In *SIGIR ACM*.

Salvaris, M., Dean, D., and Tok, W. H. (2018). *Deep Learning with Azure: Building and Deploying Artificial Intelligence Solutions on the Microsoft AI Platform*. Apress.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483--2493.

Schifano, S. F., Sgarbanti, T., and Tomassetti, L. (2018). Authorship recognition and disambiguation of scientific papers using a neural networks approach. In *International Symposium on Grids and Clouds 2018*.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2019). Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104--3112.

Tang, D., Qin, B., and Liu, T. (2015). Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422--1432.

Upadhyay, S., Faruqui, M., Tur, G., Hakkani-Tur, D., and Heck, L. (2018). (almost) zero-shot cross-lingual spoken language understanding. In *ICASSP IEEE*.

Vasilev, I. (2019). Python deep learning: exploring deep learning techniques and neural network architectures with pytorch, keras, and tensorflow.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998--6008.

Xiao, Y. and Cho, K. (2016a). Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*.

Xiao, Y. and Cho, K. (2016b). Efficient character-level document classification by combining convolution and recurrent layers. *CoRR*.

Yam, J. Y. and Chow, T. W. (2000). A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1-4):219--232.

Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Information retrieval*, 1(1-2):69--90.

Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Icml*, volume 97, page 35.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480--1489.

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55--75.

Zar, J. H. (1999). *Biostatistical analysis*. Pearson Education India.

Zhai, Z., Nguyen, D. Q., and Verspoor, K. (2018). Comparing cnn and lstm character-level embeddings in bilstm-crf models for chemical and disease named entity recognition. *arXiv preprint arXiv:1808.08450*.

Zhang, X., Zhao, J., and LeCun, Y. (2015a). Character-level convolutional networks for text classification. In *NIPS*.

Zhang, X., Zhao, J., and LeCun, Y. (2015b). Character-level convolutional networks for text classification. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 649--657. Curran Associates, Inc.