Universidad de la República
PEDECIBA Informática

Master Thesis

Data Efficient Deep Learning Models for Text Classification

Author:
Raúl Garreta

Supervisors:
Guillermo Moncecchi,
Dina Wonsever

A thesis submitted in fulfillment of the requirements for the degree of Master of Computer Science

in the

Grupo PLN
Instituto de Computación

October 31, 2020

Abstract

Text classification is one of the most important techniques within *natural language processing*. Applications range from topic detection and intent identification to sentiment analysis. Usually text classification is formulated as a *supervised learning* problem, where a labeled training set is fed into a *machine learning* algorithm. In practice, training supervised machine learning algorithms such as those present in *deep learning*, require large training sets which involves a considerable amount of human labor to manually tag the data. This constitutes a bottleneck in applied supervised learning, and as a result, it is desired to have supervised learning models that require smaller amounts of tagged data.

In this work, we will research and compare supervised learning models for text classification that are *data efficient*, that is, require small amounts of tagged data to achieve state of the art performance levels. In particular, we will study *transfer learning* techniques that reuse previous knowledge to train supervised learning models. For the purpose of comparison, we will focus on opinion polarity classification, a sub problem within *sentiment analysis* that assigns polarity to an opinion (positive or negative) depending on the mood of the opinion holder.

Multiple deep learning models to learn representations of texts including *BERT*, *InferSent*, *Universal Sentence Encoder* and the *Sentiment Neuron* are compared in six datasets from different domains. Results show that transfer learning dramatically improves data efficiency, obtaining double digit improvements in F1 score just with under 100 supervised training examples.

Keywords: Text classification, natural language processing, sentiment analysis, deep learning, transfer learning.

Acknowledgements

First of all, I would like to thank my supervisors Guillermo Moncecchi and Dina Wonsever for their support and advice. In particular, my advisor and friend Guillermo, has been a fundamental force to keep me working on this thesis, reviewing the progress, discussing ideas, suggesting improvements and pushing me forward. Thanks to my friend Javier Couto for reviewing and suggesting improvements to this work and for the great conversations about this fascinating topic.

Second, I want to acknowledge the research community which I referenced throughout this work. We all build on top of the shoulders of giants. Sharing research in the form of papers, datasets and source code is dramatically accelerating the progress in the AI field. I believe this is one of the most wonderful examples of how humankind can achieve outstanding progress based on the collaboration and sharing of knowledge.

Lastly but not less important, I want to thank and dedicate this work to my family. My wife Mayra for being so supportive with me and giving me the extra energy and time I needed to keep working on this, even during weekends. Thanks to Mom, Dad and Grandmas Kuki and Pinky for giving me everything, believing in me and your unconditional support.

Contents

Abstract **iii**

Acknowledgements **v**

Contents **vii**

1 Introduction **1**

1.1 Text Classification Overview and Motivation 1

1.2 Objectives 4

1.3 Contributions 5

1.4 Document Organization 6

2 Deep Learning Background **7**

2.1 Basic Concepts 7

2.1.1 Neural Networks Theory 8

2.1.2 Biological Neurons 8

2.1.3 Artificial Neurons 10

2.1.4 Artificial Neural Networks (ANN) 14

2.1.5 Machine Learning and Neural Networks 15

Learning weights of a single neuron 16

Perceptron Training Rule 16

Gradient Descent 17

Learning weights of a Network: Backpropagation . 18

2.2 Neural Architectures for Text Classification 20

2.2.1 Convolutional Neural Networks 20

2.2.2 Recursive Neural Tensor Networks 22
2.2.3 Recurrent Neural Networks 23
LSTMs 26
GRUs 29
Encoder-Decoder Models 29
Attention Models 31
2.3 Conclusions 35

3 Deep Learning for Text Classification 37

3.1 Text Classification Modeling 38
3.2 Transfer Learning 40
3.3 Text Vectorization 42
3.3.1 Bag of N-grams 42
3.3.2 Word Embeddings 44
3.4 Text Vectorization implementations to be evaluated 46
3.4.1 FastText 46
3.4.2 InferSent 47
3.4.3 Universal Sentence Encoder (USE) 51
3.4.4 Bidirectional Encoder Representations from Transformers (BERT) 53
3.4.5 Sentiment Neuron 57
3.5 Classification Model 60
3.6 Conclusions 60

4 Datasets and Results 61

4.1 Problem definition: Polarity classification for Sentiment Analysis 61
4.1.1 What is an opinion? 62
4.1.2 Formal Definitions 63
4.2 Datasets 65
4.2.1 Movie Reviews (Stanford Treebank) 67

4.2.2 Product Reviews 68
4.2.3 Hotel Reviews 69
4.2.4 Restaurant Reviews 71
4.2.5 Airline Twitter Comments 73
4.2.6 Political Twitter Comments (GOP Debate) 75
4.3 Metrics and Objectives 76
4.3.1 Accounting for binary and imbalanced datasets . . 76
4.3.2 Accounting for imbalance in small subsets 77
4.4 Results and Analysis 79
4.4.1 Movie Reviews (Stanford Treebank) 81
4.4.2 Product Reviews 84
4.4.3 Hotel Reviews 86
4.4.4 Restaurant Reviews 88
4.4.5 Airlines Twitter Comments 90
4.4.6 Political Twitter Comments (GOP Debate, the Outlier) 92
4.4.7 Comparison of Datasets per Model 94

5 Conclusions 95
5.1 General Conclusions 95
5.1.1 Transfer Learning improves data efficiency 96
5.1.2 Sentiment neuron model achieves best performance 97
5.1.3 Domain where representations are trained affect results 97
5.1.4 Benchmarks with code and data reproducibility . . 98
5.2 Future Work 98
5.2.1 Representations in other languages 98
5.2.2 Representations in other domains 99
5.2.3 Representations in a generic domain 99
5.2.4 Active learning 99

5.2.5 Zero-shot learning 100
5.2.6 Transfer learning in other NLP problems 100

Bibliography **103**

Chapter 1

Introduction

This chapter gives a brief introduction and motivation of this work. Finally the objectives and contributions of this thesis will be stated and a general organization of the document will be provided.

1.1 Text Classification Overview and Motivation

Text classification is one of the most important techniques within *Natural Language Processing (NLP)*. Applications range from topic detection, intent identification to sentiment analysis. Usually text classification is formulated as a *supervised learning* problem, where a labeled training set is fed into a *machine learning* algorithm. The process of text classification gets a text as input and returns a class as a result. The class value usually belongs to a discrete and finite set of values that can represent a particular topic of the text, intent or sentiment depending on the problem to solve. Variations of this problem can return multiple classes as a result, known as *multi-label classification*. The training and the prediction processes involved in supervised text classification are summarized in Figure 3.1.

There has been a large body of research on text classification using machine learning techniques. The typical approach consist in a two step scheme of *feature extraction* (automatic or handcrafted) followed by a classification step (Joachims 1997).

The most common feature extraction methods in text classification have been *bag-of-words* or *bag-of-n-grams* with their frequency or *tf-idf*. The objective of this step is to transform the text into a numeric representation in the form of a vector. Usually each component of the vector represents the frequency or tf-idf score of each word in a predefined dictionary.

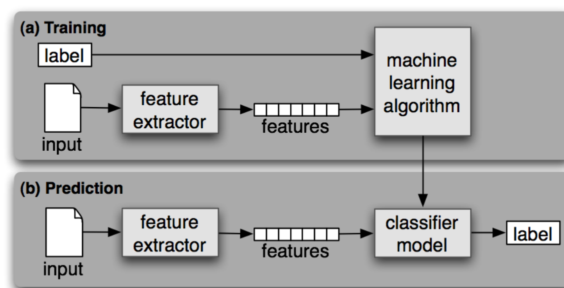


FIGURE 1.1: Supervised Text Classification process. (a) Training process where the feature extractor transfers the text input into a feature vector. Pairs of feature vectors and labels are fed into the machine learning algorithm to generate a model. (b) Prediction process, where the feature extractor is used to transform unseen text input into feature vectors. These feature vectors are then fed into the model, which generates predicted labels (Bird, Klein, and Loper 2009).

The classification step usually involves a statistical model such as *Naive Bayes* (Joachims 1997) or linear models such as *Logistic Regression* and *Support Vector Machines* (Joachims 1998). More details in this implementation will be provided in Subsection 3.3.1: Bag of N-grams.

One of the known limitations of this approach resides in the fact that the bag-of-words representation loses the sense of ordering between words, that is, sequence and order in a sentence are important for meaning, but bag-of-words completely loses this aspect.

The bag-of-n-grams approach tries to fix this issue by taking sequences of 2 or 3 words (bigrams or trigrams) as features, instead of isolated words. That way, it can add more context and the sense of ordering of words in a sentence.

Even though both approaches have naive assumptions, they have proven to work surprisingly well on many practical applications of text classification (Wang and Manning 2012). However, there are various clear pitfalls in these approaches of text representation, some of the most important being that:

- Each component of the vector (word or n-gram) is equidistant in the vector representation. That means that this method fails to represent the fact that words with similar semantics should have closer distance.

- The representation space is sparse, which implies a large amount of labeled training data (in the order of thousands for training examples to achieve reasonable classification performance).

More recently, after a few pioneer works on *Neural Networks* applied to NLP (Collobert and Weston 2008; Collobert, Weston, et al. 2011), new feature extraction techniques have been applied based on *word embeddings* also known as *word vectors* (Bengio et al. 2003; Mikolov, Chen, et al. 2013) where words are projected from the sparse 1 of V encoding (where V is the vocabulary size) into a low-dimensional vector space. This representation is usually obtained through a hidden layer in a neural network trained over a large corpus, with popular implementations such as *Word2Vec* (Mikolov, Ilya Sutskever, et al. 2013) and *GloVe* (Pennington, Socher, and Manning 2014). The advantage of this kind of representation is the fact that semantically close words are also close in euclidean or cosine distance in the lower dimensional vector space. The vector representations of each word in a text can then be combined to obtain a fixed size representation of the whole text, which then is used as input by the classification step. Subsection 3.3.2: Word Embeddings will provide more details of these methods.

Many works based on *Deep Learning* techniques such as *Convolutional Neural Neteworks (CNNs)* (Lecun et al. 1998; Kim 2014) were trained on top of pre-trained word embeddings with very good results in standard text classification datasets. These architectures try to create hierarchical representations of texts inspired by the success in their application to computer vision.

Other related approaches are based on *Temporal Convolutional Neural Networks* (Zhang and LeCun 2015; Zhang, Zhao, and LeCun 2015; Conneau, Schwenk, et al. 2016) have been trained from scratch without any need of previous word representations, just from raw character or word representations. This approach is based on the fundamental idea of CNNs that consider feature extraction and classification as a one single problem. Subsection 2.2.1: Convolutional Neural Networks will provide an introduction to CNNs for text classification.

However it is still not clear what is the best way to combine the individual word embeddings to obtain a representation of a whole text which usually has complicated syntactic and semantic relations. Some works such as *Recursive Neural Networks (RNTNs)* (Socher et al. 2013) propose

to use external resources such as parsers to define the order in which the word embeddings of a text are combined to obtain the whole representation. The disadvantage relies in the need of these kind of resources for different languages and domains. Subsection 2.2.2: Recursive Neural Tensor Networks will provide more detail about this implementation.

Finally, one of the most promising approaches in deep learning applied to NLP comes from the *Recurrent Neural Networks (RNNs)* (Elman 1990). The advantage of this approach is the fact that sentences can be represented as a sequence of tokens (either characters or words) which allows modeling the way text is written and read.

Some examples of this approach are presented in (Tang, Qin, and T. Liu 2015) where word embeddings are first combined to form a sentence embedding (either using CNNs or *LSTMs* (Hochreiter and Schmidhuber 1997)), then the sentence representation is fed into a recurrent neural network using *GRU* units (KyungHyun Cho et al. 2014). Finally the recurrent network hidden state is used as a representation of a variable length text which is fed into a softmax classifier. Xiao and Cho also provide an implementation combining recurrent and convolutional networks but this time, inputing the text at a character level (Y. Xiao and Kyunghyun Cho 2016). Subsection 2.2.3: Recurrent Neural Networks will provide an introduction to RNNs for text classification.

However, one big problem that all of these solutions (the classic approaches and deep learning approaches) have in common is that they require large amounts of labeled training sets which in practice constitutes a bottleneck for practical applications. A considerable amount of human labor is required to manually tag the training sets.

As a result, it is desirable to have supervised learning models that require smaller amounts of tagged data. A lot of research has been done pushing the boundaries of text classification accuracy on full training sets, but few has been done on small datasets.

1.2 Objectives

In this work, we will research and compare supervised learning models for text classification that are *data efficient*, i.e.: require small amounts of tagged data to achieve state of the art performance levels. In particular, we will study *Transfer Learning* (Pan and Yang 2010), a method

within machine learning that allows to reuse or transfer previous knowledge learned on a task A to train supervised learning models on another task B similar to A. Transfer learning has proven to be very effective in computer vision (Oquab et al. 2014) to create supervised models that are data efficient, that is, require lower amounts of tagged images.

Just for the purposes of comparison of different transfer learning approaches, we will focus on opinion polarity classification, a sub problem within text classification, also known as *Sentiment Analysis* (Pang and Lee 2008b) that assigns polarity to an opinion (positive or negative) depending on the mood of the opinion holder about a particular topic. Polarity classification is a very complex and well defined problem where multiple tagged datasets in different domains exist, thus being a very good field of research for our comparison in data efficient models for text classification.

The following are the objectives proposed in this thesis:

- Show that transfe r learning approaches can be used in text classification to obtain data efficient models, that is, models that require less tagged training examples than models that do not leverage transfer learning (learn text representations from scratch).
- Compare different data efficient models for text classification particularly for polarity classification in sentiment analysis.

1.3 Contributions

The contributions of this work are the following:

1. A deep analysis of different transfer learning techniques for text classification applications have been performed. It includes an overview of the evolution and state of the art of deep learning techniques, presenting the fundamentals and practical implementations of text classification that leverage deep learning and transfer learning techniques.
2. We showed that transfer learning approaches can be used for text classification in order to obtain models that are more data efficient (require smaller training sets) than models that learn text representations from scratch.

3. A set of 6 different datasets to train and test text classification models in different domains were gathered. This provides easy access to other works that need to use datasets to reproduce this benchmark or work on new implementations.

4. A Python 3 code base was generated[1] with the benchmarks that reproduce the implementations described. This is very important to actually test and compare the implementations.

1.4 Document Organization

The document will be organized as the following:

Chapter 2: Deep Learning Background will provide a theoretical background to basic concepts of deep learning, applied in particular to text classification. These concepts will be necessary to understand the particular implementations that will be compared in the rest of the document.

Chapter 3: Deep Learning for Text Classification will provide a description of the particular deep learning implementations that will be compared to solve polarity classification in sentiment analysis. A classic supervised machine learning model based on a logistic regression and bag of n-grams will be used as baseline. Two important concepts will be also introduced: *text vectorization* and *transfer learning*. The rest of the models used are based on deep learning approaches with different architectures, ways to represent text and transfer learning techniques.

Chapter 4: Datasets and Results will describe the datasets that will be used for the experimentation. These datasets represent different domains such as politics, product reviews, hotel reviews, restaurant reviews between others. This way we will compare the performance of each model in different situations. The results obtained in the benchmark will be then analyzed both at a quantitative level based on classic machine learning performance metrics and a qualitative level analyzing particular examples and corresponding results. As mentioned before, the data efficiency of each model will be the top metric of comparison.

Chapter 5: Conclusions will state conclusions obtained, challenges and future paths of research.

[1] https://github.com/raulgarreta/data-efficient-text-classification

Chapter 2

Deep Learning Background

In this chapter a brief introduction to neural networks and deep learning will be provided. These concepts are then used as part of the implementations described in Chapter 3: Deep Learning for Text Classification. First the basic concepts of neural networks will be introduced in Section 2.1: Basic Concepts. Then the general architectures of neural networks will be described in Section 2.2: Neural Architectures for Text Classification, with the particular application for text classification and sentiment analysis.

2.1 Basic Concepts

In the last decade, deep learning has become the main Machine Learning algorithm for many artificial intelligence problems, including computer vision and natural language processing. This approach has pushed the state of the art and obtained the best performance metrics in many fronts, including text classification.

However, it is worth mentioning that the base theory behind it has been developed decades ago, under the name of *Artificial Neural Networks*. This new name is not just a rebranding, but a new rise in neural network algorithms thanks to new research, architectures, tools, data and hardware that now allow researchers to solve complex problems that previously have been almost impossible.

A brief overview of the theory of deep learning will be provided. A minimal knowledge in machine learning is required to understand the concepts.

2.1.1 Neural Networks Theory

Artificial Neural Networks (ANNs) theory was inspired by the human brain, so it is important to have a little background on how the brain works to understand how this theory started.

By the end of the nineteenth century, the existence of nerve cells and their interconnection in functional structures was widely accepted. Nerve fibers were known to conduct electrical impulses and by the end of 1930, excitation and inhibition of individual cells was demonstrated. McCullock and Pitts (McCulloch and Pitts 1943) made the observation that, like propositional logic can be *true* or *false*, neurons can be *on* or *off*, either if they are firing a signal or not.

The brain works based on billions of neural cells (neurons) that are interconnected to form higher level structures. Back in 1943, Warren McCulloch and Walter Pitts (McCulloch and Pitts 1943) created the first computational model of an artificial neural network. This work was an important landmark in science that drafted the initial theory not only for artificial intelligence, but also for neuro science and cognitive science. It described a model that found common elements between how animals and machines work.

The theory basically states that small units of processing (neurons) activate, that is, return an output based on their inputs. These outputs can then be inputs (excitatory or inhibitory) to other neurons.

There are two key elements in the McCullock and Pitts work:

- The neurons are interconnected in networks to build higher complexity structures.
- The logical model of activation of neurons.

The result is a symbiosis of elements retrieved from mathematics and physiology, inspired by computational theory (Turing 1936), where a computation can be described as a set of units of computation that are indivisible.

2.1.2 Biological Neurons

The theory of McCullock and Pitts assumes that first, the nervous system is a network of neurons, each having a *soma* and an *axon*, and that

synapses (connections) are always between the axon of one neuron and the soma of another. Their second assumption was that at any instant the neuron has some *threshold*, which excitation must exceed to initiate an impulse. A third important assumption was that excitation occurs mainly from axonal terminations to somata, and inhibition involves the prevention of the activity of one group of neurons by concurrent or antecedent activity of a second group. See Figure 2.1 for a reference on the general structure of a biological neuron. These were all generally accepted assumptions about neurons gained from decades of empirical investigation.

The theory presupposed that the activity of a neuron is an all-or-none process: a neuron either *fires* or it does not. McCulloch and Pitts initially presupposed that the structure of the net does not change with time. Then the authors admitted that these were both abstractions: that the activity of neurons could empirically be shown to be more continuous than discrete, and that phenomena such as *learning* could alter the structure of a net permanently, so that a stimulus which would previously have been inadequate is now adequate.

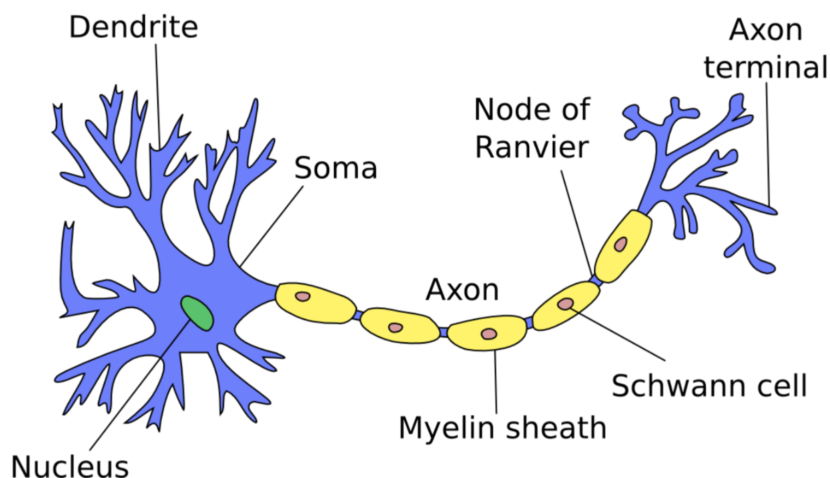


FIGURE 2.1: Graphical representation of a biological neuron.

It is worth mentioning the following facts (Mitchell 1997) to have a reference of the order of magnitude in which the human brain works:

- The human brain is estimated to have approximately 10^{11} neurons densely interconnected (on average, each neuron is connected with 10^4 neurons).

- The fastest neuron activation times are around 10^{-3} seconds, (quite slow compared with computer times (10^{-10} seconds).
- Humans are able to make complex decisions surprisingly quickly. For example, it is estimated that a person can recognize the face of his mother in around 10^{-1} seconds.

All the previous facts can be useful to provide raw conclusions that can be helpful in how to model problems with artificial neural networks. For example, if we divide the average time required by a human to recognize a familiar image by the average activation time of a neuron, that means that there must be at most a few hundred steps or layers of neurons to do all the processing. From this observation, we can conclude that the biological neural system must have high parallel processing and distributed representations. These simple but powerful observations have been one of the many guidelines used in artificial neural networks to try to build architectures that can solve complex problems like image processing or natural language processing.

2.1.3 Artificial Neurons

As we have described in the previous section, biological neurons can be roughly viewed as small units that receive inputs into their dendrites, and depending on that particular input (which of them are activated and which are not) and their internal state, they can activate their output by triggering a signal on their axon.

Artificial neurons were conceived (initially) as a mathematical structure to model this behavior of biological neurons. As with biological neurons, there are different types of artificial neurons which are modeled by different mathematical functions. But in general, an artificial neuron takes a set of inputs and produces an output depending on the particular mathematical transformation that it implements.

The usual transformation in an artificial neuron takes the inputs and performs a weighted sum which is then passed through a nonlinear function commonly known as the *activation function*:

$$y = \phi(\sum_{j=0}^{m} w_j x_j)$$

where w is a vector of weights, x the input vector and ϕ the activation function.

Here is a simple example: if we have a neuron that has 3 inputs and has only one output, then we will have 3 weights (one for each input, i.e. $m = 3$). If the input values are $x_0 = 2, x_1 = 0, x_3 = 5$, with weights $w_0 = 1$, $w_1 = 7$ and $w_2 = -3$ respectively, then the linear combination would be:

$$x.w =$$

$$x_0 \times w_0 + x_1 \times w_1 + x_2 \times w_2 =$$

$$2 \times 1 + 0 \times 7 + 5 \times -3 =$$

$$2 + 0 - 15 =$$

$$-13$$

Then the activation function ϕ would be evaluated over the result of the linear combination (in our example -13). The output of the neuron would be $\phi(-13)$.

This function can be represented using linear algebra as the dot product of vectors:

$$y = \phi(w \cdot x)$$

Where vectors w and x which contain the components of the weights and inputs respectively are combined with the *dot product* and then passed through the activation function.

There are many ways to graphically represent artificial neurons as pictured in Figure 2.2 and Figure 2.3.

The activation function ϕ usually has the following properties:

- **Monotonically increasing**: the magnitude of the output increases as the magnitude of the input increases.
- **Continuous**: roughly speaking, small changes in the input produce small changes in the output.

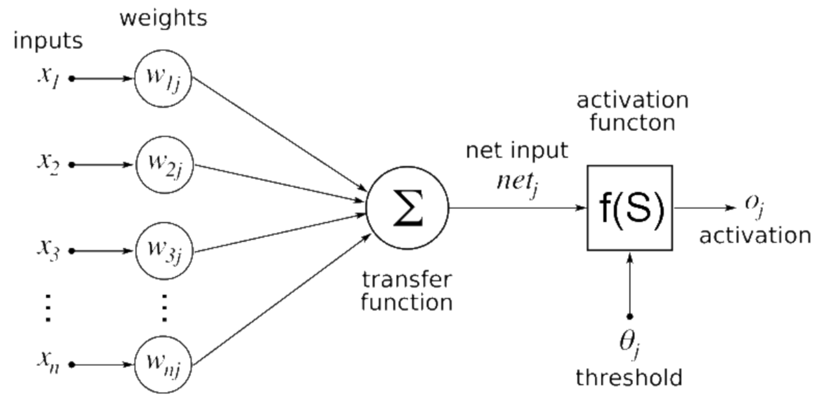


FIGURE 2.2: Graphical representation of a artificial neuron.

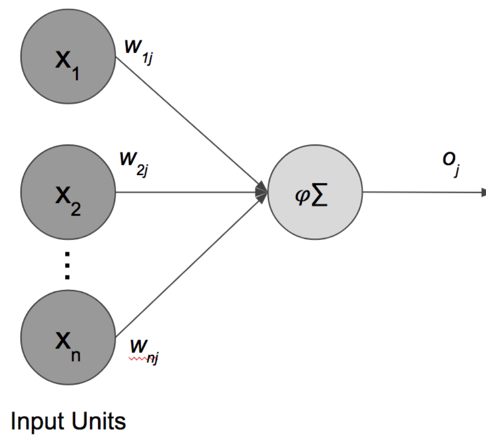


FIGURE 2.3: Simplified graphical representation of a artificial neuron.

- **Differentiable**: that way its derivative can be calculated. In particular with ANNs we want functions where the derivative not only exists, but it is also easy to compute.
- **Bounded**: a function that returns an output that can be bounded, like squashing all the input in a bounded range, e.g.: *Sigmoid* function takes all the real number space into a number between 0 and 1. *Tanh* function does the same but between -1 and 1.

There are a handful of activation functions that meet most of the above properties and are commonly used in neural networks:

- Linear
- Sigmoid
- HardTanh
- Sign
- Hyperbolic Tangent (Tanh)
- ReLU (Rectified Linear Unit)

Figure 2.4 shows a graphical representation of each one.

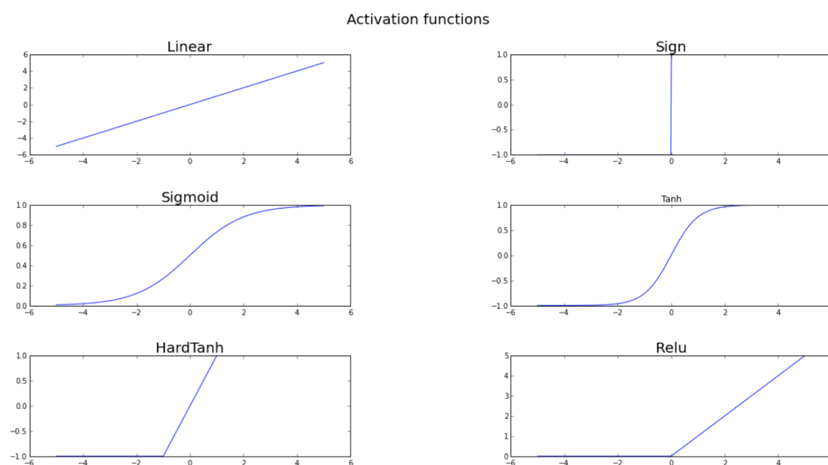


FIGURE 2.4: Activation functions commonly used for artificial neural networks.

2.1.4 Artificial Neural Networks (ANN)

Individual neurons can represent certain basic functions, and training an artificial neuron is equivalent to a *logistic regression*, a model widely used in machine learning. But when combining artificial neurons (connecting outputs of neurons to inputs of others) we can represent a large variety of functions.

Figure 2.5 shows the graphical representation of a neural network with a very popular architecture known as *feedforward*, also known as *dense network*.

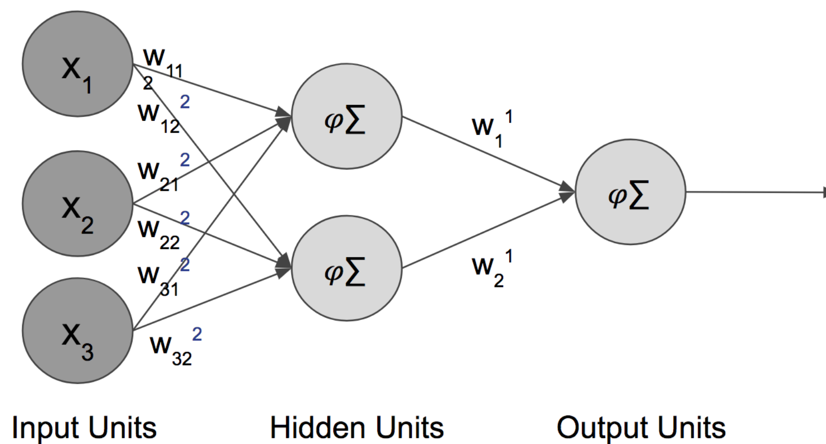


FIGURE 2.5: Graphical representation of an artificial neural network.

This particular network has:

- An input vector x of three components (the *input layer*).
- Two *hidden units*: neurons that are positioned in a *hidden layer* after the input layer and before the *output layer*. The hidden neurons are fed by the outputs of the input layer (the three inputs).
- A single output neuron (the output layer) which is fed by the outputs of the hidden layer (the outputs of the two hidden neurons).

To picture how this network could be used in practice, supposed we wanted to detect sentiment in text, we could use this network in the following manner:

- The inputs could model if a particular word appears or not in a text. So we may have one input for each possible word in the dictionary, where 1 means that the word is present and 0 that it is not. That is the classical implementation of the bag of words representation of texts.
- The output could model if the text has a positive (the network returns 1) or negative sentiment (the network returns 0).

Even though this kind of implementation have a very naive assumption (the text is represented as a bag of words when clearly semantics in text has a sequential and ordering nature), it has been very useful in practice as it works surprisingly well. One of the reasons is that, since the model is very simple, it requires few parameters to be adjusted and as a result, few training examples.

In the following sections more complex architectures and models will be presented. But before diving in more complex architectures, the basic notions of the learning algorithms for neural networks will be introduced.

2.1.5 Machine Learning and Neural Networks

Until now, the structure of neurons and neural networks have been reviewed, but technically nothing used machine learning. The initial models proposed by McCulloch and Pitts, where just to explain their structure, a static structure that does not change over time. But as with real biological neural networks, the interesting part comes when connections between neurons can dynamically change, adapt and improve to create the desired output given a particular input. And that is when machine learning comes into the show.

The objective now is to describe the process in which the learning algorithm will adjust the weights of the network (the w vectors) so that for every particular input in the training set, the desired output is obtained.

Before diving into how to do that with neural networks, first understanding a simpler case is important: adjusting the weights of a single artificial neuron. After that, describing how to generalize the process with a complete network will be easier.

Learning weights of a single neuron

There are many ways to implement a learning algorithm that, starting from a set of training examples, adjust the weights of an artificial neuron to get the desired output. The basics of most of them consist in the following steps:

1. Begin with an initial set of weights (e.g.: start with random weights).
2. Input an example into the unit and obtain the resulted prediction.
3. Adjust weights appropriately whenever the output is different to the expected result.
4. Repeat steps 2 and 3 as many times with all the training set (one *epoch*) as many times as necessary until the outputs for every training example are correct.

The variations of the learning algorithms mainly differ in how they adjust the weights to correct the output of the neuron:

$$w \leftarrow w + dw$$

Perceptron Training Rule

The *Perceptron Training Rule* is a very simple rule to adjust the weights to obtain the desired result given a particular input. This rule makes the weight learning with the following equation:

$$\delta w = \alpha(t - y)x$$

Where t is the desired output (target output) and y is the current output generated by the neuron when the vector x is used as input and the vector w as weights. The α parameter is a constant known as the *learning rate*, which is used to adjust the magnitude of the changes made to the weights. Usually it is a small number and typically decays as the process iterates over new more training examples.

This rule adjusts the weights in the correct direction in order to correct the errors in the outputs that the unit produces comparing to the desired output. It is pretty intuitive and it is demonstrated to converge in a

finite number of iterations to a set of weights that make the unit correctly classify all the training examples, provided that the instances are linearly separable.

The problem with this rule is that it may not converge when the training examples are not linearly separable, which is a very common situation. This is where the *gradient descent* method comes in.

Gradient Descent

The advantage of *gradient descent* is that if the training examples are not linearly separable, the rule still converges to the best possible approximation to the desired output. The basic concept behind the gradient descent is that it searches the best combination of weights in the possible combination space (hypothesis space) by using the gradient of a function.

Gradient descend is basically a method that searches for the minimum in a function. In our case, the minimum in the error of the output of an artificial neuron by minimizing the squared error between the returned and the desired output for every example in the training set. This can be represented with the following equation:

$$J(w) = \frac{1}{2} \sum_{d \in D} (t_d - y_d)^2$$

Where D is the set of training examples, t_d and y_d are the target and obtained outputs for the *d-th* training example. The way to find the minimum of a function (in our case the minimum of the error function) is to calculate the gradient (the derivative in each dimension). After having the gradient, the steepest descent along the error surface towards to the minimum can be found. This is why the rule's name is gradient descent.

The updating rule thus would be:

$$w = w - \alpha \nabla J(w)$$

∇J is a vector whose components are the partial derivatives of the error function J respect to each of the vector w (weights) components. α is again the learning rate and plays the same role as with the perceptron rule, it moderates the modifications made to the weight vector.

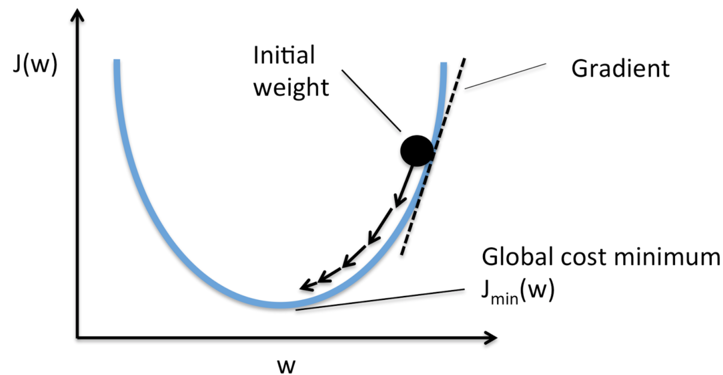


FIGURE 2.6: Graphical representation of gradient descent algorithm with a single scalar weight parameter w. The updates made to w are made in the opposite direction of the gradient, thus minimizing the error function J

The reason why the activation function must be differentiable (i.e. the derivative can be calculated) is mainly to be able to calculate the error derivative with the *chain rule*. Besides being able to get the derivative of the activation function, it is desired also that it can be calculated efficiently because of practical reasons. It will be computed on every example and every iteration (and every unit when working with networks) of the training process. That means a large amount of time, so optimizations in this calculation, have a lot of impact on the total training time.

Learning weights of a Network: Backpropagation

As stated before, single units can only represent linear decision surfaces: they can perfectly separate only linearly separable training sets. However, multilayer neural networks can represent nonlinear functions (provided that nonlinear units are used) and this is why usually networks of artificial neurons are used. Nonlinear functions can solve much more complex problems, e.g.: detect sentiment in text.

In order to train a multilayer network some generalization of the gradient descent rule must be made, that is exactly what *Backpropagation* algorithm (Rumelhart, Hinton, and R. J. Williams 1988) is about.

The backpropagation algorithm adjusts the weights in a neural network in order to minimize the error between the obtained output and the

target output of the network when feeding the network with its inputs.

The main differences are:

- As neural networks can have multiple outputs, the minimization can be calculated, for example, by taking the sum of the errors of all the outputs of the network.
- Weights of all the units in the network must be adjusted (it is a much larger search space).
- The error surface can have multiple local minima, so it is not guaranteed that the algorithm converges to the global minima.

However, despite these difficulties, backpropagation obtains very good results in many practical applications, and it is the base of all neural network learning algorithms.

The process is similar as with a single unit:

1. Initialize the weights of all the units in the network (e.g.: with random values).
2. Iteratively input every training example to the network, get the result, calculate the error and compute the gradients (with backpropagation) to update the weights in all the units of the network.
3. Repeat the step 2 until the error reduces and you obtain the desired outputs.

The algorithm works very similar to the gradient descent, but has some differences. The output units are the only where the error can be directly calculated. In the case of hidden units, the errors in the network's output must be propagated to the outputs in the internal units. Intuitively, the error in the hidden unit output is calculating by summing the error for each of the output units that the hidden input influences (the output units that have as input the output of the hidden unit). That error is weighted by the weight between the connection of the hidden and the output unit. The same process as before can be repeated with lower hidden units, this time the error in an upper hidden unit will be used to propagate the error to a lower hidden unit. This is the rationale behind the name

"backpropagation", we propagate the error from the output through the hidden layers back to the input units.

2.2 Neural Architectures for Text Classification

So far the basic principles of artificial neural networks and the most classical feedforward architecture have been shown. Part of the success in new approaches in deep learning are related to the use of new advanced architectures: the *Recurrent Neural Networks* and the *Convolutional Neural Networks*. In the following subsections the most important architectures will be described in the context of text classification.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) (Lecun et al. 1998) are one of the most popular deep learning architectures, very frequently used in image processing, but also very useful for NLP tasks.

The most representative approach to sentence classification, in particular for sentiment analysis using convolutional networks is shown in the Figure 2.7 by (Kim 2014):

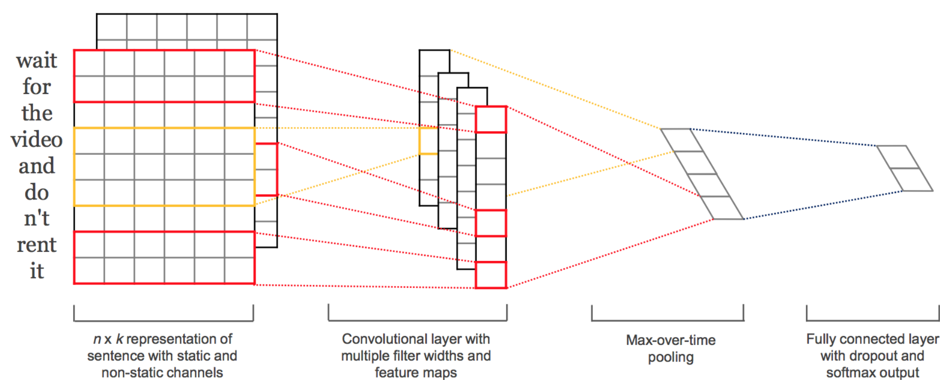


FIGURE 2.7: Graphical representation of a convolutional neural network for text classification.

This architecture is a variant of (Collobert, Weston, et al. 2011) where, each word is represented as a k-dimensional vector. Then a sentence of length n words is encoded as the concatenation of n word vectors.

One layer of convolution is trained on top of word vectors previously obtained with unsupervised techniques (Mikolov, Ilya Sutskever, et al. 2013). The vectors are a result of a neural network trained over a large dataset (100 billion words from *Google News*).

This approach reinforces the idea that pre-trained vectors can be reused as feature extractors for various classification tasks in order to reduce the amount of labeled training examples, which is one of the objectives of this thesis.

The model obtains further improvements in performance by learning "task specific" vectors: word vectors that are adjusted as part of the supervised learning process, besides the static word vectors obtained by (Mikolov, Ilya Sutskever, et al. 2013). Thus the name *multichannel CNN*.

Convolution operations are then applied to windows of h words to produce a feature map. For example, c_i is a feature created that takes $x_i : i + h - 1$ word vectors from the input:

$$c_i = f(w \cdot x_{i:i+h-1} + b)$$

where w are the weights, b the bias and f the activation function. This filter is applied to each possible window of words in the sentence to produce the feature map:

$$c = [c_1, c_2, ..., c_{n-h+1}]$$

with $c \in R^{n-h+1}$.

Then a *max over time pooling* operation is applied over the feature maps. Basically it takes the maximum value of c from a particular set of filters. This way the max pool captures the feature with the highest value for each feature map. It also deals with the variable sentence length: a variable length input is transformed to a variable size feature map with the convolutional filters and then the max pooling transforms the variable size feature map into a fixed size vector.

Multiple filters (with varying window sizes) are calculated to form multiple feature maps. The result of the max pooling layer (which returns a fixed size vector) is finally connected with a fully connected *softmax layer* to produce a probability distributed set of labels (in our case, the sentiment labels).

The model also employs *regularization* by adding a *dropout layer* just after the feature maps. This prevents co-adaptation of hidden units, also known as *overfitting*.

2.2.2 Recursive Neural Tensor Networks

(Socher et al. 2013) presented a work that is one of the most important references in modern sentiment analysis with machine learning. The paper is important for two reasons: designing an advanced deep learning method for sentiment analysis and also creating a sentiment dataset frequently used by other research papers.

It proposes an advanced way to represent meaning in sentences, the *Recursive Neural Tensor Networks* (RNTNs). The motivation starts on the fact that even though word embeddings have been very useful in representing the semantics of words, they cannot express the meaning of sentences an their compositionally.

This work creates a dataset, the *Sentiment Treebank* that will be used in this thesis as part of the benchmark. It has sentences with their corresponding parse tree labeled with the partial sentiment. This dataset has been very useful to do compositionally analysis: how the sentiment of each part of the structure of a sentence affects the overall sentiment of the sentence.

The model first uses a parser to obtain the structure of the sentence. The nodes of the sentences are vectors that represent words (leaves of the tree) and partial representations of the sentence (internal nodes). Then a neural network is trained to compose two vectors (inputs) that could be a single word or an internal node into a new vector (internal node). The learning system then has to transform the last internal representation vector into the sentence sentiment with a softmax classifier. Besides, each node is trained with a softmax classifier as the sentiment of each of the subtrees was also labeled. Figure 2.8 shows a graphical representation of this model.

The RNTN tries to model the compositionally of sentiment based on the parsing trees. In the moment of publication, it outperformed all previous methods.

No further details on the implementation will be provided, since it can be conclude that the model has some practical drawbacks:

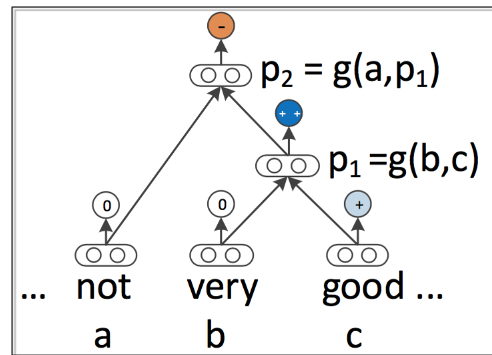


FIGURE 2.8: Graphical representation of a Recursive Neural Tensor Network. A parser is used to establish the order in which the word vectors are combined for the whole sentence (Socher et al. 2013).

- Requires the text to be parsed, which means that it requires extra resources: a parser for the particular language, which is usually something expensive to create.
- Requires a labeled training set not only with the sentiment at the sentence level, but on each of if subtrees obtained in the parsing tree. Again, something expensive for practical purposes.

2.2.3 Recurrent Neural Networks

One of the main constrains of the dense feedforward networks and the convolutional neural networks is that they operate over a fixed size input. If a variable size input, such as, for example, text, has to be processed, first some kind of transformation must be made before in order to turn it into a fixed size vector.

Moreover, these kind of networks have a fixed set of computations on the input, which intuitively would be a constraint for variable length inputs that may require a variable amount of computations to be processed.

Lastly, intuitively when a person reads a text, it does it in sequence, where previous words (or characters) affect the meaning of the following elements in the sequence: the reader has memory of the sequence and ordering affects the result. This is another point where dense feedforward and convolutional networks are not able or struggle to represent.

At the end of the day, using these kind of architectures particularly in text processing, where the nature is to have a variable length sequence implies a large limitation. These limitations are then translated in naive assumptions like bag of word representations to name a very known implementation.

Recurrent Neural Networks (RNNs) (Elman 1990) overcome all these restrictions and this is probably the reason why they are having significant success in NLP applications.

The fundamental property that RNNs have is the possibility to have a network with loops Figure 2.9. These loops connect the state of the current sequence input with the state of the same network obtained after consuming all the previous elements in the sequence.

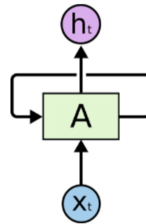


FIGURE 2.9: Graphical representation of RNN with a loop (Olah 2015).

The unrolled version of this graphical representation is the following:

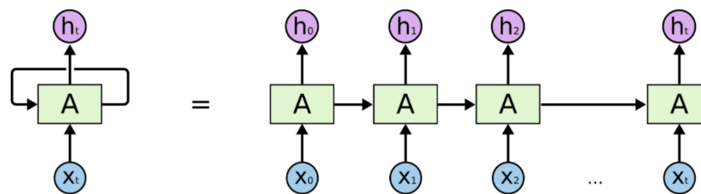


FIGURE 2.10: Unrolled graphical representation of a RNN. Blue circles are input vectors, green rectangles are RNN hidden states and purple circles are output vectors (Olah 2015).

In the notation showed in Figure 2.10, t represents the timestep; in the case of text, if we consume characters, the first character x_0 will be the input at $t = 0$ (the first character in the sequence), x_1 the input at $t = 1$ (the second character), etc. In the same way, the network will produce an

output h_t at each timestep t and have an internal state at each timestep t.

Figure 2.11 shows a toy application of a RNN that could be trained as a character level language model: a model that given a sequence for characters returns a probability distribution of the next character.

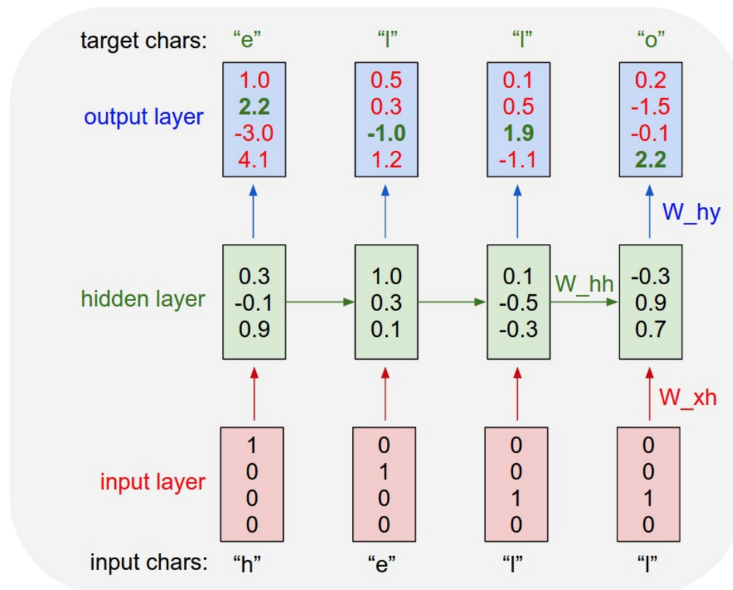


FIGURE 2.11: Graphical representation of a RNN for character level language model (Karpathy 2015).

In this example (Karpathy 2015) the RNN observes the first character in the sequence h represented by the column vector $(1, 0, 0, 0)$. The output of the network at that timestep produces an output vector that represents the probability distribution of the next character in the sequence. In this example assigns 2.2 to letter e, -3.0 to l, and 4.1 to o. Since the training examples is the sequence *hello*, the correct character output in this timestemp should be e. The learning algorithm then would adjust the weights in order to increase the probability of this output.

The process is repeated after inputing every character in the sequence: from the current state (after inputing the first n characters) output the next character.

The RNN network is composed of the same artificial neurons as described in previous sections, so the same bakpropagation algorithm can be used to adjust the corresponding weights to obtain the desired outputs

given a training set.

In the previous example, after training the RNN model, a higher number in the component associated to the letter e should be seen after inputting the letter h.

Another important advantage of RNNs is the fact that they are able to map variable length inputs into variable or fixed length outputs as shown in Figure 2.12, thus the reason they are very useful for NLP applications.

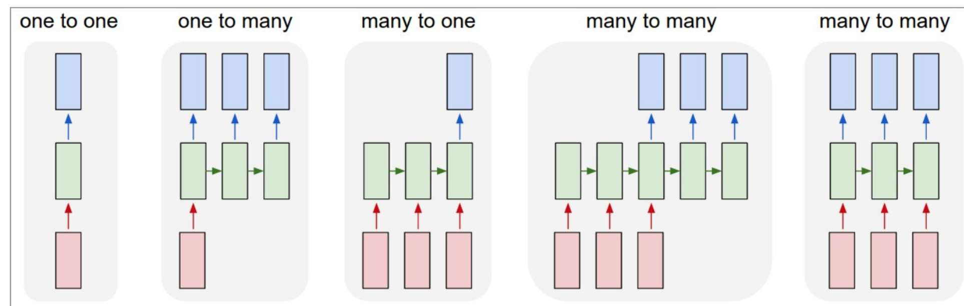


FIGURE 2.12: Graphical representation of RNNs. Red rectangles are input vectors, green rectangles are RNN hidden states and blue rectangles are output vectors (Karpathy 2015).

LSTMs

The *Long Short Term Memory networks* (LSTMs) (Hochreiter and Schmidhuber 1997) are a special kind of RNN that is shown to be more efficient in practical applications. Almost all the current state of the art results that involve RNNs are implemented using LSTMs.

The main advantages of standard RNN model are their capability of connecting previous elements in the sequence to output the correct prediction in the current element. Sometimes the connections are made with elements close to the current timestep, for example, in the sentence:

The monitor has a good brightness.

The word *birghtness* would be predicted because of the previous inputs in the sequence, the words *monitor* and *good* would give the context that the writer is talking about a feature of a monitor. These words are relatively close to the current timestep.

Sometimes the connections must be made with longer distance dependencies, e.g.:

Last week I bought a monitor at BestBuy store, it's a little bit expensive but it has good brightness.

Here for example, the last word can be inferred to be a feature/characteristic since the previous words *has a good* suggest it. But in order to predict that it is a feature of a monitor, a connection with an element longer before that word must be made (the word *monitor* at the beginning of the sentence). Unfortunately in practice, this connections with long gaps are difficult for vanilla RNNs to learn.

This is where LSTMs can help since they were particularly designed to overcome the long-term dependency problem. LSTMs have the capability of remembering information for longer periods of timesteps.

The LSTM has the same loop recurrency, but it substitutes the single layer of *tanh* units with a more sophisticated 4-layer structure that gives the ability to remember or forget previous features of elements in the sequence.

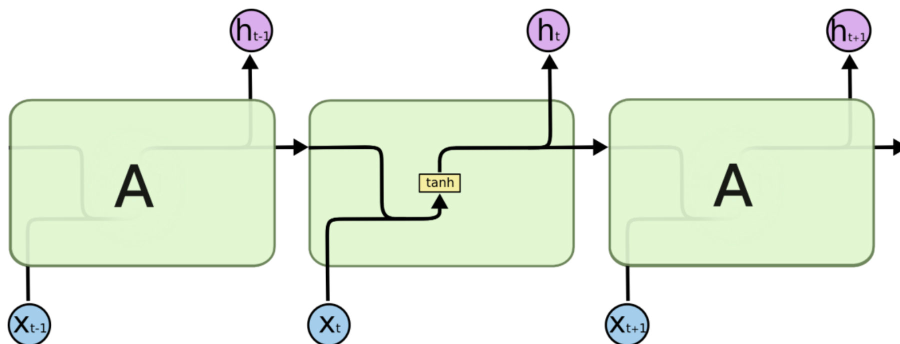


FIGURE 2.13: Internal structure of a RNN (Olah 2015).

The core structure within an LSTM is the *cell state*, a vector that stores a numeric representation of the current state of the network. In the last example, one of the components of the cell state could store if the subject is a monitor or not.

The four internal layers, are in charge of learning how to modify the cell state based on the current state, the previous output and the current

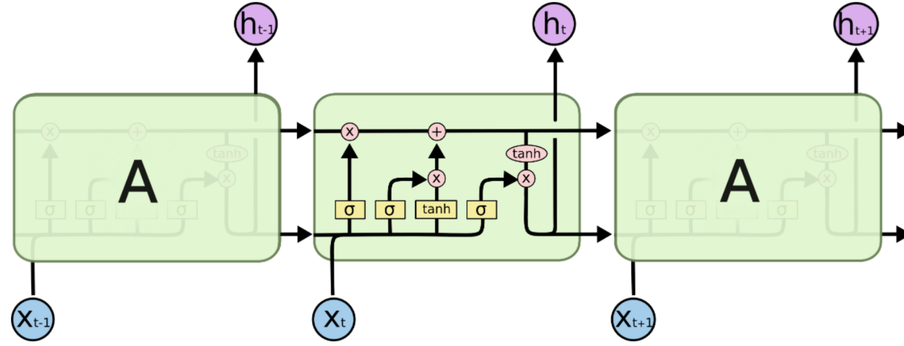


FIGURE 2.14: Internal structure of an LSTM (Olah 2015).

input. This is made with:

1. A **forget gate layer** that learns how to decide which components of the current cell state are kept and which are forgotten. This is implemented with a sigmoid layer that returns a number between 0 (forget) and 1 (keep) for each component of the cell state based on the previous output and the current input.

$$ft = \sigma(W_f \cdot h_{t-1}, x_t + b_f)$$

2. An **input gate layer** that learns to decide which components of the cell state will be updated:

$$i_t = \sigma(W_i \cdot h_{t-1}, x_t + b_i)$$

3. A **tanh layer** that learns to create a new values to be added to the cell state:

$$Cs_t = tanh(W_C \cdot h_{t-1}, x_t + b_C)$$

With these new values, the cell state is updated:

$$C_t = f_t * C_{t-1} + i_t * Cs_t$$

4. Finally the **output layer** that learns to decide which will be the output: which of the current cell state components will be used as

outputs. A *tanh* function is applied in order to get values between -1 and 1:

$$o_t = \sigma(W_o \cdot h_{t-1}, x_t + b_o) h_t = o_t * tanh(C_t)$$

GRUs

There are many variations of the LSTM, one of the most notable are the *Gated Recurrent Units (GRUs)* (KyungHyun Cho et al. 2014). The most dramatic changes are the fact that they combine the forget and input gate layers into a single *update gate layer*. That means that the unit forgets only when it is going to input new values to the cell state. This has also the advantage that only one layer needs to be learned instead of two. The GRU also merges the cell state and the hidden state (output) between other modifications. This model is a simpler version of the LSTM an recently has gained a lot of popularity.

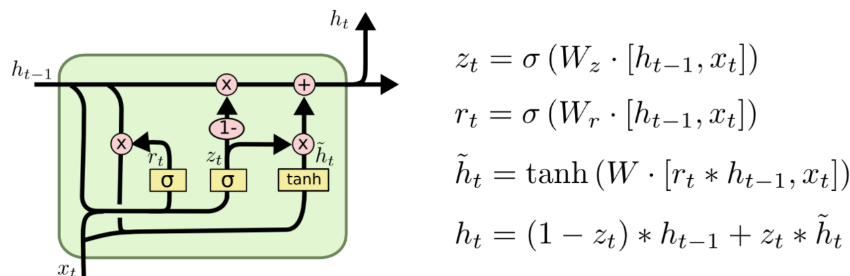


FIGURE 2.15: Internal structure of a GRU (Olah 2015).

Encoder-Decoder Models

Encoder-Decoder models (Ilya Sutskever, Vinyals, and Le 2014; Kyunghyun Cho et al. 2014) for RNNs is a particular architecture designed to address sequence to sequence problems such as language translation, i.e.: how to translated an arbitrary length input sequence into an arbitrary length output sequence. As a result they are of particular interest for this work since they are a very useful representation to model text sequences.

Figure Figure 2.16 shows a graphical representation. At a higher level the encoder-decoder model is composed by two sub-models: the encoder model which is in charge of encoding the input sequence X into a vector

C called *context* and a decoder model that takes the context vector C and decodes it into an output sequence y. Note that the input sequence X could be an arbitrary length sequence of size T (for example: the sequence of words in the sentence "I like oranges", $T = 3$) and the output sequence y of size T' (for example: the sequence of words in the sentence "Me gustan las naranjas", $T' = 4$), where T can be different from T'. Note also that the vector C is a fixed length vector, that is, the encoder is able to encode a variable length sequence into a fixed length vector as described in (Kyunghyun Cho et al. 2014).

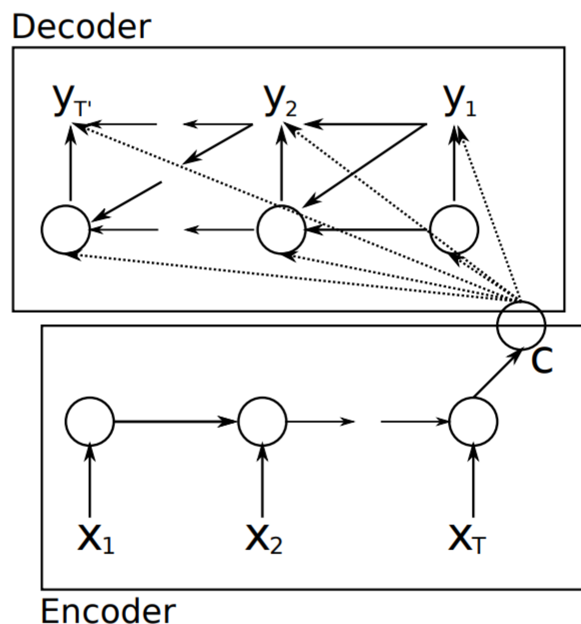


FIGURE 2.16: Graphical representation of an Encoder-Decoder model (Kyunghyun Cho et al. 2014).

There are different flavors on how to implement the encoders and decoders, in the one from (Kyunghyun Cho et al. 2014) both $y(t)$ (the output at step t) and $h(t)$ (the hidden state at step t) are conditioned on $y(t-1)$ (the output of the previous step), $h(t-1)$ (the hidden state at the previous step) and the context vector C of the input sequence.

Attention Models

Attention Models (Bahdanau, Kyunghyun Cho, and Bengio 2014) are an extension of Encoder-Decoder models that were proposed to try to overcome an inherent limitation of the encoder-decoder: they have to encode a variable length sequence into a fixed size vector. Intuitively, if the sequence is very long (e.g.: long sentences, paragraphs or entire documents), it will be very hard for the encoder to compress the representation of the input sequence.

Another problem of recurrent models (even using LSTMs and GRUs optimizations) is the fact that it has a hard time to remember useful inputs that are needed at long range distances in the output. Figure 2.17 shows an example where the last word in the output has its translation far away in the second word of the input. This dependency is very hard to remember.

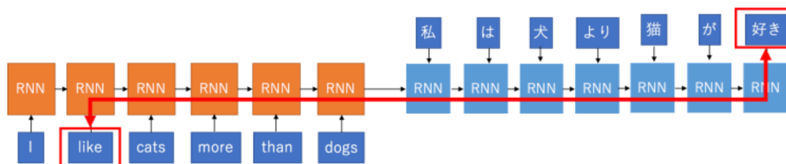


FIGURE 2.17: Example of long-range dependency that is hard for the classical RNN, even with LSTMs or GRUs to remember.

The attention model was developed in the context of machine translation where the authors were interested in solving the alignment problem: identify which parts of the input are relevant to each of the words of the output. The model then learns to jointly translate and align at the same time. It learns the output sequences but also learns to search in the input sequence. As a result, a particular word in the output is predicted from the previous predicted words and also the entire input sequence.

Intuitively, the attention mechanism allows the decoder to "look back" at the entire sentence and selectively extract the information it needs during decoding.

In the paper from (Bahdanau, Kyunghyun Cho, and Bengio 2014) shown in the graphical representation in Figure 2.18, the attention model has a bidirectional recurrent network to process the input sequence. This gives an additional advantage where the input is processed both forward

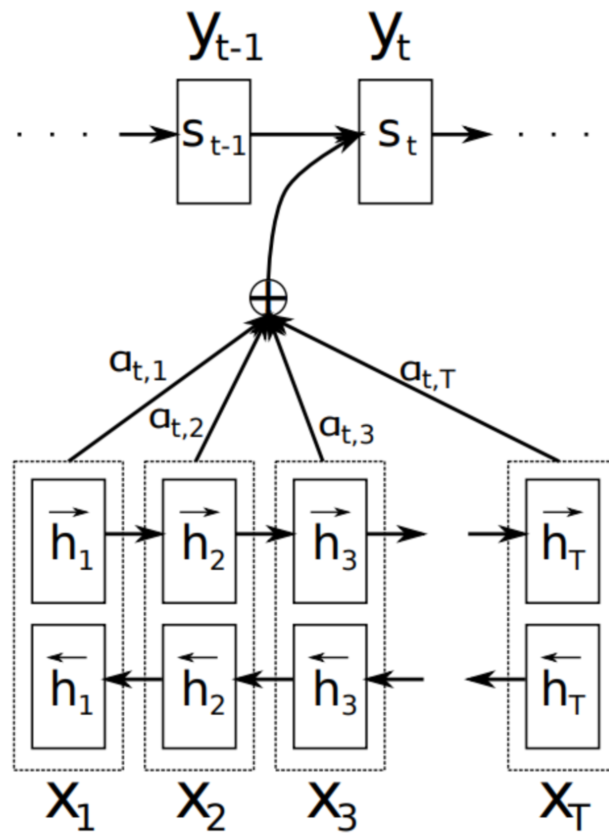


FIGURE 2.18: Graphical representation of an Attention Model (Bahdanau, Kyunghyun Cho, and Bengio 2014).

and backwards, which intuitively is an advantage in cases where not only the last words in a sequence are more important than the first words in order to start predicting the output sequence.

The attention model does not just send the last hidden states of the input sequence into the decoder as a context for the decoder (as in the classic encoder-decoder model), but it also sends a weighted combination of all the hidden states in the input sequence. Those weights are calculated based on scores e learned jointly as part of the model learning process. It basically learns a weighting function a (alignment function) that gives a score for each of the pairs of output steps and input steps.

$$e_{t't} = a(o_{t'}, h_t)$$

For example $e_{t't}$ would be the score or attention weight for the input at step t in order to predict the output at step t'. The alignment function a can be modeled in multiple ways, but originally it was modeled with a one-layer feedfordward neural network with a *tanh* transfer function. Then a softmax function is used on top of the scores in order to normalize and be treated as probabilities. The result is the actual attention weights $a_{t't}$:

$$a_{t't} = \frac{e^{t't}}{\sum_{i=1}^{T} e^{t'i}}$$

Finally the particular context vector c for an output step t' is calculated as the weighted sum of hidden input states:

$$c_{t'} = \sum_{i=1}^{T} a_{t'i} h_i$$

At each output step t', the decoder will take the corresponding context vector $c_{t'}$ and the output hidden state $s_{t'-1}$ at the previous step in order to predict the output $y_{t'}$.

It is interesting to visualize the attention matrix to introspect how the model pays more attention on particular input words to predict particular output words. Figure 2.19 shows an example of the generated attention matrix for a model that translates an input sequence (a sentence in English) into an output sequence (a sentence in French).

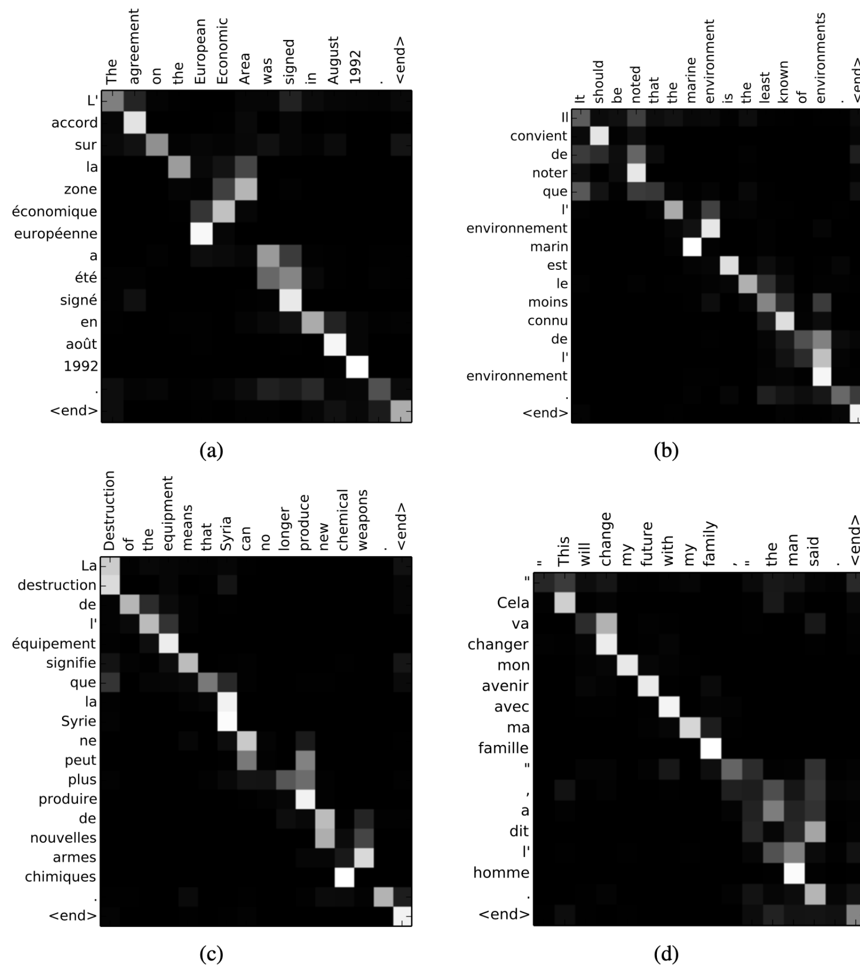


FIGURE 2.19: Example of the values of the attention matrix of the translation between a sentence in English (x axis) into a sentence in French (y axis) (Bahdanau, Kyunghyun Cho, and Bengio 2014).

2.3 Conclusions

In this chapter a brief introduction to neural networks and deep learning has been provided. The most important concepts of neural networks were introduced and general architectures applied to text classification were reviewed. This will be helpful when reviewing the particular implementations introduced in Chapter 3: Deep Learning for Text Classification.

Chapter 3

Deep Learning for Text Classification

In this chapter the particular implementations for text classification that are used in this thesis will be described. The way the different implementations differ are basically in the way they do the text vectorization, while the classification stage implementation is exactly the same and will be described in Section 3.5: Classification Model.

The different text classification implementations will be then compared in Chapter 4: Datasets and Results.

As stated in Chapter 1: Introduction, the motivations and objectives of this thesis are to compare different text classification models that are data efficient, i.e.: that require small amounts of labeled training examples to achieve state of the art performance. In particular, the models will be compared for polarity classification in sentiment analysis.

This chapter is organized in the following way:

Section 3.1: Text Classification Modeling presents how the text classification problem is modeled, introducing the two fundamental stages of text classification: text vectorization stage and the supervised machine learning classification stage.

Section 3.2: Transfer Learning introduces a formal definition of transfer learning that will be used to inject previous learned knowledge into the system.

Section 3.3: Text Vectorization provides an introduction to the classical text vectorization techniques, including Subsection 3.3.1: Bag of N-grams, Subsection 3.3.2: Word Embeddings. In particular *bag of n-grams* will be used as the baseline for the comparisons.

Section 3.4: Text Vectorization implementations to be evaluated provides a description of the different transfer learning based text vectorization models and implementations that will be compared in this thesis: Subsection 3.3.2: Word Embeddings, Subsection 3.4.2: InferSent, Subsection 3.4.3: Universal Sentence Encoder (USE), Subsection 3.4.4: Bidirectional Encoder Representations from Transformers (BERT) and Subsection 3.4.5: Sentiment Neuron

Since there are multiple deep learning models in the state of the art to choose from, this thesis will focus on the implementations that:

- Are good representations of the underlying deep learning methods, e.g.: CNN, RNNs, etc.
- Use representations of text (vectorizations) that could be trained independently with unsupervised or supervised learning techniques, thus allowing transfer learning techniques which is one of the motivations of this thesis.
- Have an implementation publicly available, this is necessary to reproduce the results obtained in the corresponding papers and also can be reused as part of the benchmark comparison in this thesis.

Finally Section 3.5: Classification Model introduces the machine learning classification model that will be used on top of the text vectorizations in order to implement the full text classification model.

It is worth mentioning that all the text classification models compared (including the baseline) will use the same supervised learning classification model: a simple linear regression.

3.1 Text Classification Modeling

Typically text classification problems are modeled with two clear stages:

1. **Text vectorization stage** is in charge of taking the input text and transform it into a feature vector. This vector is expected to be a good representation of the text, where each feature should encode particular and relevant syntactic and semantic characteristics of the input text. It is usually also desired that the vectorization returns a fixed length vector, independently of the length of the input text.

This is very useful for practical purposes (and usually required) for the classification stage to use as input.

2. **Classification stage** takes the vector obtained from the text vectorization stage and transforms it into the desired output. The output usually is a finite set of labels that must be assigned to the text. The output could be single-label when the output should be just a single label for each input text, or multi-label when the output could return multiple labels for a particular input text. In our case (polarity classification for sentiment analysis), the problem is a single-label one: for a particular text, we require the output be either *positive* or *negative*, but not both at the same time.

Figure 3.1 shows a graphical representation.

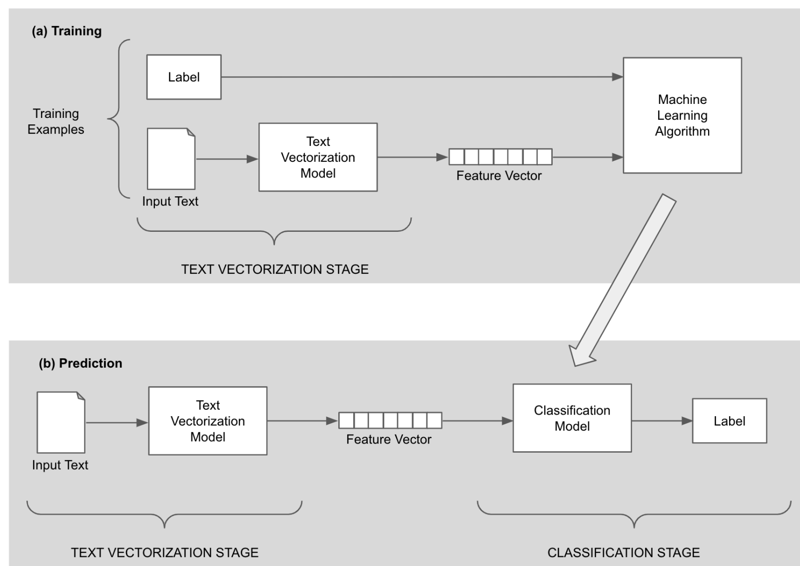


FIGURE 3.1: Supervised Text Classification process. (a) Training process where the feature extractor transfers the text input into a feature vector. Pairs of feature vectors and labels are fed into the machine learning algorithm to generate a model. (b) Prediction process, where the feature extractor is used to transform unseen text input into feature vectors. These feature vectors are then fed into the model, which generates predicted labels.

The way to incorporate previous knowledge into this setup is by using transfer learning techniques to obtain text vectorizations that add previous knowledge into the system, the text classification task (target task)

by transferring a text vectorization learned from another problem (source task).

Then the classification stage will learn the downstream task (target task) with a supervised machine learning model that leans from a training set vectorized with the text vectorization model. The classification will learn to transform the text vectorization into the sentiment labels (positive or negative).

By using a text vectorization model that was previously trained in a source task, the classification stage requires significantly less training examples to learn the downstream task. Intuitively, the system will use pre-existing knowledge from the world (encoded in the feature rich text vectorization model) thus not requiring to learn that knowledge from scratch.

The reader should take into account that a pre-trained vectorization model could have been trained with potentially millions of training examples which could have involved tagging large amounts of training examples, using large untagged corpuses, and consuming large computation resources.

On the other end, in a practical setup where we want to train data efficient models, the order of magnitude of training examples used to train the downstream task ranges from a just few tens to a few hundreds of training examples.

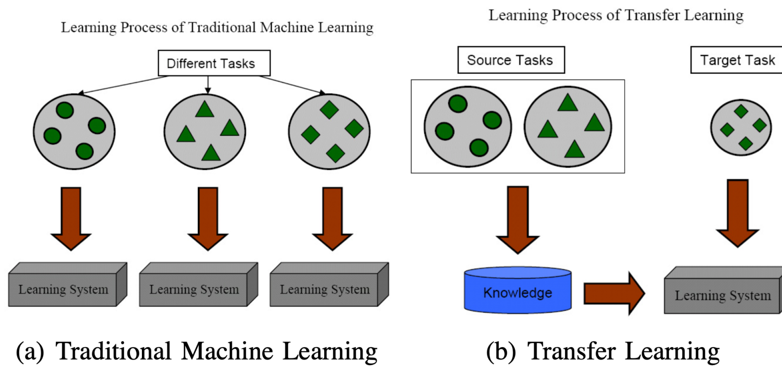
The beauty of transfer learning then resides in the fact that the expensive task of training a feature rich text vectorization with large training costs is made just one time (and will be reused), while the much more cheaper stage of adjusting the downstream tasks of the classification stage is done for every time for the particular domain.

3.2 Transfer Learning

Transfer learning also known as *domain adaptation* is a technique that has been successfully applied to supervised classification tasks where a small training set is available (Pan and Yang 2010; Yosinski et al. 2014). It basically consists in transferring or adapting knowledge from a source task S into a target task T where a small training set is available, see Figure 3.2.

In particular, with deep learning applications, transfer learning has

a direct and intuitive use. If tasks T and S are similar, the knowledge acquired and represented by network S parameters can be useful for the task T.



(a) Traditional Machine Learning (b) Transfer Learning

FIGURE 3.2: Graphic representation of transfer learning processes (Pan and Yang 2010).

The most common implementation consists in initializing the parameters of the lower layers of a neural network model for the target task with the parameters of the lower layers of the source neural network. That can be accomplished if both networks have the same topology.

One of the most successful applications have been in image classification where training a network on a large image dataset has been useful to be transferred to more specific image classification tasks (Krizhevsky, Ilya Sutskever, and Hinton 2012).

Transfer knowledge usually is performed between two tasks that are semantically similar, but trained with different datasets. Sometimes the transfer can also be performed between two semantically different tasks, but that share the same network topology.

Two main approaches of transfer learning have been developed:

- **Transfer learning by parameter initialization**: the knowledge between two tasks S and T is transferred by initializing parameters in task T with those learned in task S.
- **Transfer learning by multi-task learning**: the knowledge between two tasks S and T is transferred by training both tasks simultaneously and sharing a common set of network layers (multi-task learning).

While in image classification tasks, transfer learning has proven to show good results, its application to text classification was initially uncertain. Some previous works have tried to apply transfer learning techniques in the domain of NLP (Semwal et al. 2018; Mou et al. 2016).

Some relevant conclusions have been stated that are relevant to this thesis:

- The effectiveness of transfer learning in neural networks depends largely on the semantic similarity between the source and target tasks.
- Usually the output layers are specific to the task and dataset, and as a result they are not transferable.
- Word embeddings have been successful transferred between semantically different tasks.
- Transfer learning by parameter initialization and multi-task learning have both produced comparable results. The combination of both approaches does not appear to produce additional improvements.

3.3 Text Vectorization

As it was described before, the typical model of text classification is composed by two steps, being the first step the transformation of a text into a vector, also known as *feature extraction* or in the particular case of text classification, *text vectorization*. The following sub-sections give an introduction to the typical text vectorization techniques.

3.3.1 Bag of N-grams

The classical way to implement text vectorization is through *bag of words*. This algorithm is very simple, efficient and has a surprisingly good accuracy in practice.

The bag of words vectorization describes a text by counting the occurrences of words in the text. First a finite set of words (dictionary) is defined. The size of the dictionary will be the size of the vector representation of a text where each component of the vector represents the

frequency of a particular word in the text. Figure 3.3 pictures the bag of word text vectorization with an example.

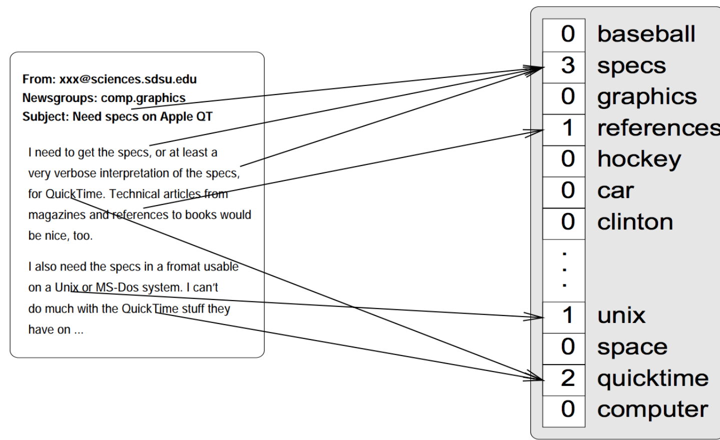


FIGURE 3.3: Bag of words representation (Joachims 1997).

Using the term frequency alone has an important disadvantage: common words such as *a*, *the*, *and*, etc. will be very frequent but will not add meaningful information that can be helpful to classify the sentiment of a text. In order to remove those irrelevant terms, we can use two techniques:

- **Stopword filtering**: a list of common English words or even the particular domain of the problem can be defined such us articles or connectors. Those words are then not considered for the construction of the vectorization dictionary.
- **IDF weighting**: the term frequency *TF* is multiplied by the inverse document frequency of the term *IDF* which counts the number of different texts where the term is present. This intuitively penalizes a term if it's too frequent across the text dataset, thus being little useful to differentiate texts into different classes. This technique is commonly known as *TF-IDF* (term frequency inverse document frequency).

Another important problem with just using isolated words (unigrams) in bag of words is that words alone do not capture multi-word expressions since the frequencies are counted not taking into account the word ordering. This representation completely ignores the relative position of the

words within the texts. A very common approach to reduce this problem is to use n-grams instead of just unigrams. This vectorization method is know as *bag of n-grams*. Instead of building a dictionary of words, the bag of n-grams algorithm builds a dictionary of unigrams, bigrams and trigrams thus capturing expressions such as "not good", "very useful", etc. It's a very simple but effective way to provide word ordering and context on the vectorization stage.

The baseline model used in this thesis will be a bag of n-grams vectorizer. The model vectorizes the text input with the bag of ngrams algorithm with an n-gram range between 1 and 3 (unigrams, bigrams and trigrams). The vectorizer constructs a dictionary with the top 20,000 n-grams with highest TF-IDF score in the training set. Then each component in a feature vector is obtained by calculating the TF-IDF score of the particular n-gram in the text. The TF-IDF dictionary will be calculate on top of the entire training set.

3.3.2 Word Embeddings

The typical vectorization with bag of ngrams has important drawbacks. Some of them can be reduced by using the techniques described in the previous section, but still there are other disadvantages. One of them is that each of the words are represented as a discrete and arbitrary set of indexes, each one at the same distance of the rest (Bengio et al. 2003).

This encoding does not provide useful information about the relation that exists between words. For example, in a text vectorization dictionary composed by four words: *cat*, *dog*, *car* and *bus*, if the words are represented with the vectors:

$$cat = (1, 0, 0, 0)$$

$$dog = (0, 1, 0, 0)$$

$$car = (0, 0, 1, 0)$$

$$bus = (0, 0, 0, 1)$$

Also known as *one-hot-encoder*, the distance between any pair of words in the euclidean vector space would be the same. This means that the classification step that will work on top of this representation does not have any information about the relation between words. In this example, the words *cat* and *dog* are related since both represent concepts that are

somehow similar (both are animals, four-legged, pets, similar size, etc). The same with the words *car* and *bus* (both are vehicles, four wheels, have combustion motors, etc). It would be desirable that the words cat and dog have less distance between them than with the word car or bus.

The one-hot-encoder representation also has the disadvantage that the vector space is sparse which usually means that more training examples will be needed in order to train a text classifier on top of it. In this representation, one new component in the vector is needed for each word that the model needs to index. For example, if the dictionary contains 10,000 words, then the vector representation will have 10,000 components. This makes a large sparse vector space, which in turns means that the classification model that works on top of it will have to adjust the same order of parameters, a problem known in machine learning as the *curse of dimensionality* (Mitchell 1997).

Word embeddings (Bengio et al. 2003; Mikolov, Chen, et al. 2013; Mikolov, Ilya Sutskever, et al. 2013; Mikolov, Yih, and Zweig 2013; Pennington, Socher, and Manning 2014) try to overcome these problems by learning word vector representations where distance has a semantic meaning and the vector space is more dense by reducing sparsity and vector dimensions. Semantic similar words will be mapped to nearby points in the vector space.

The methods used to create word embeddings use the distributional hypothesis, which means that words that are used in the same context would have similar vector representations. Historically two approaches have been used to calculate the embeddings. The most recent ones have been those based on neural networks which will be described and used in this thesis.

One of the most popular and efficient implementations for calculation of word embeddings is the *Word2Vec* (Mikolov, Ilya Sutskever, et al. 2013; Mikolov, Chen, et al. 2013). Besides being efficient, the paper shows that algebraic operations in the vector space map to semantic and syntactic relations between words. Figure 3.4 shows some common examples:

- **Gender**: If we subtract the vector representing the word *man* to the vector representing the word *woman*, we obtain a vector g representing the gender transformation $Male \rightarrow Female$. If we add the vector g to the vector representing the word *king*, we should obtain a word vector representing the word *queen*.

- **Verb tense**: If we subtract the vector representing the word *walking* to the vector representing the word *walked*, we obtain a vector v representing the syntactic verb tense transformation $Gerund \rightarrow PastTense$. If we add the vector v to the vector representing the word *swiming*, we should obtain a word vector representing the word *swam*.

- **Semantic relations**: If we subtract the vector representing the word *Spain* to the vector representing the word *Madrid*, we obtain a vector c representing the semantic transformation $Country \rightarrow Capital$. If we add the vector c to the vector representing the word *Italy*, we should obtain a word vector representing the word *Rome*.

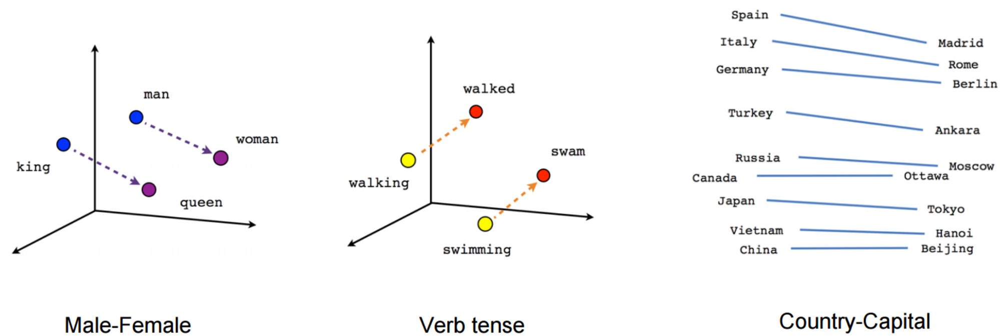


FIGURE 3.4: Semantic relationships in word vectors like male/female, gerund/past tense and country/capital (Mikolov, Yih, and Zweig 2013).

3.4 Text Vectorization implementations to be evaluated

The following subsections will provide a brief overview of the text vectorization implementations that will be used and compared in this thesis.

3.4.1 FastText

FastText (Joulin et al. 2016) is a library for learning word embeddings and text classification created by Facebook's AI Research lab. It has a similar implementation to Word2Vec to obtain word vectors. The most important difference resides in the fact that Word2Vec uses a single word as the

smallest unit, while FastText uses morphological information, modeled as n-grams of characters. Then a word is represented with a vector that is obtained as the sum of the vectors of all the character n-grams contained within the word.

For example, the word *sunny* would be formed by n-grams:

```
sun, sunn, sunny, unny, nny
```

This way of representing word vectors has several advantages:

- Rare words (words that appear infrequently) have better representations since they have chances to share character n-grams with other frequent words.
- It can obtain a vector representation even for unknown words (out of vocabulary words). For example the word:

  ```
  fantaboulouslovelyevent
  ```

  Probably would not appear in a text, and as a result, implementations like Word2Vec or GloVe would return a zero vector. However, FastText would probably return a vector similar to the vectors of the words *faboulous*, *lovely* and *event*.

Besides learning word representations with this novel approach. FastText implements a simple linear neural network that can be trained as a text classifier using the word representations as features. FastText averages the word vectors within a text and uses that as the input for the supervised text classifier. As a result, FastText can leverage transfer learning techniques by reusing the word vectors. In particular, we will perform the benchmarks with FastText by reusing the pre-trained word vectors in English.

3.4.2 InferSent

InferSent (Conneau, Kiela, et al. 2017) is a sentence embeddings method that provides semantic representations for English sentences, also developed by Facebook's AI Research Lab. The authors mentioned that the research community has agreed that word vectors were without doubt

useful for different NLP problems, but still there were no sentence vectors (sentence representations) that can be successfully used in different NLP problems such as text classification.

Most of the approaches that obtain a fixed size vector representation of a sentence are based on unsupervised approaches like *SkipThought* vectors (Kiros et al. 2015) (in a similar fashion to word vectors). In this paper, the authors work on a sentence representation that is based on a supervised learning method. The inspiration comes from the image processing field, where generic low level representations of images have been successfully created based on a supervised learning approach using the *ImageNet* dataset.

This work tries to do a similar approach for NLP to encode sentences. The dataset used is the *Stanford Natural Language Inference (SNLI)* (Bowman et al. 2015) which consists of 570,000 human generated English sentence pairs, manually labeled with one of three categories: *entailment*, *contradiction* and *neutral*. The hypothesis stated by the authors is that the semantic nature of Natural Language Inference (NLI) is a good candidate for learning universal sentence representations (sentence embeddings), useful for transfer learning, through a supervised method.

First, sentences are encoded with a recurrent neural network shown in Figure 3.5.

The words are first transformed into vectors with a word embedding, potentially pretrained with a unsupervised method like Word2Vec or GloVe. Then the word vectors of a sentence are processed in a sequence by a bidirectional LSTM. For a sentence of T words, it generates T hidden vectors. Each hidden vector is composed by the concatenation of the forward LSTM and the backward LSTM:

$$\overrightarrow{h_t} = \overrightarrow{LSTM_t}(w_1, ..., w_T)$$
$$\overleftarrow{h_t} = \overleftarrow{LSTM_t}(w_1, ..., w_T)$$
$$h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$$

Then these T hidden vectors are combined to generate a fixed size representation. Two possible ways exist: a *max pooling* (which take the maximum value of each dimension of the hidden units) or the *mean pooling* (which makes an average of all the hidden vectors).

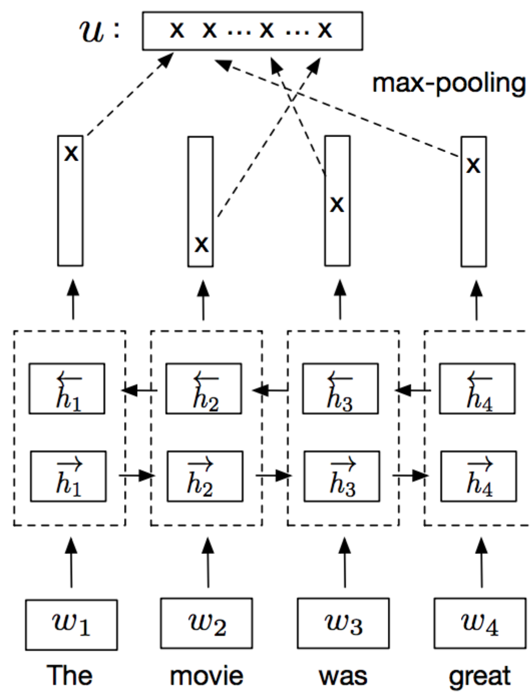


FIGURE 3.5: Bidirectional LSTM max pooling neural network encoder from InferSent (Conneau, Kiela, et al. 2017).

Then the encoder is used to obtain representations u and v for each of the pairs of sentences of the SNLI dataset. The two representations are then combined with different methods to get relations between them:

1. Concatenation of the two representations: (u, v).
2. Element wise product: $u * v$.
3. Absolute element wise difference: $|u - v|$.

The resulting vector captures the information from both representations, the *premise* u and the *hypothesis* v which are feed into a 3-class classifier (which will learn the three labeled NLI relations entailment, contradiction and neutral). This neural network is composed by multiple fully connected layers followed by a softmax layer.

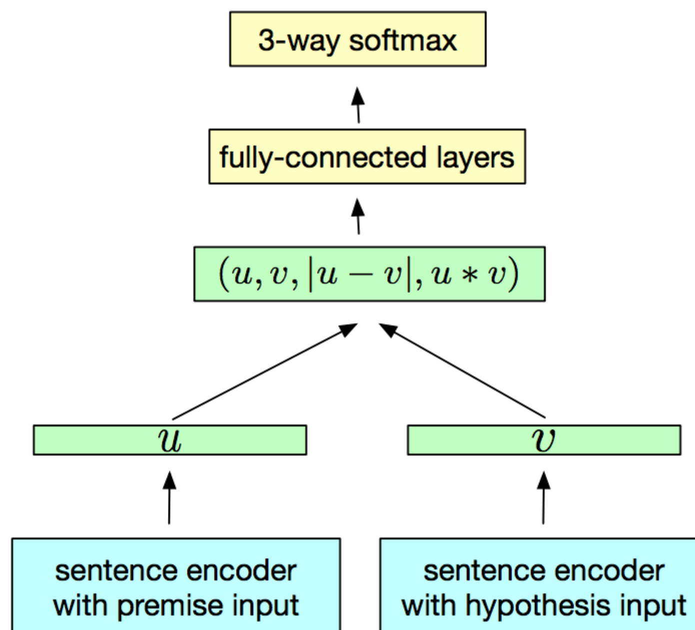


FIGURE 3.6: Neural network architecture for the NLI learning (Conneau, Kiela, et al. 2017).

The model is trained as a supervised model using the SNLI dataset. The authors describe the usage of a stochastic gradient descent algorithm with a learning rate of 0.1 with a weight decay of 0.99. At each epoch they divide the learning rate by 5 when the accuracy on the development set

decreases. The word vectors used have 300 components and are obtained with the GloVe implementation trained on the Common Crawl dataset.

This model, after trained can be used as a feature extractor for a standard supervised machine learning model.

3.4.3 Universal Sentence Encoder (USE)

The *Universal Sentence Encoder (USE)* (Cer et al. 2018) is a model developed by Google Research that presents a way to encode sentences into embedding vectors specifically targeted to be used for transfer learning in NLP tasks. They propose two variants for the encoding model which have a trade-off between accuracy and computing resources needed. In the paper, they show a benchmark comparing different approaches, including baselines with models that use word level transfer learning via pre-trained word embeddings and other baselines with models that do not use any transfer learning. The models are trained with a multi-task transfer learning approach:

The two model implementations are:

- **Transformer model (T)**: A model that creates a sentence encoder using the transformer architecture (Vaswani et al. 2017) This model obtains higher accuracy at the cost of greater model complexity and resource requirements (compute time is quadratical, $O(n^2)$ in the length fo the input sequence). The sentence embedding is constructed by using the encoding sub-model from the transformer architecture. It uses multi-headed attention to compute context aware representations of words in a sentence that take into account both the order and the identity of all the words. Word representations are then used to obtain the fixed length sentence encoding vector by computing the element-wise sum of the words.
- **DAN model (D)**: A model that creates a sentence encoder using deep averaging networks (DAN) (Iyyer et al. 2015). This is a simpler model that obtains slightly less accuracy with less computation requirements (compute time is linear, $O(n)$ in the length fo the input sequence). The sentence embeddings is constructed by inputing word and bigram embeddings that are first averaged together and then passed through a feedforward deep neural network to produce the output vectors.

Both models generate sentence embeddings taking as input an English sentence and producing as output a fixed dimensional embedding of dimension 512.

Figure 3.7 shows an example of a practical use of the computed embeddings: computing the sentence level semantic similarity score between two sentences.

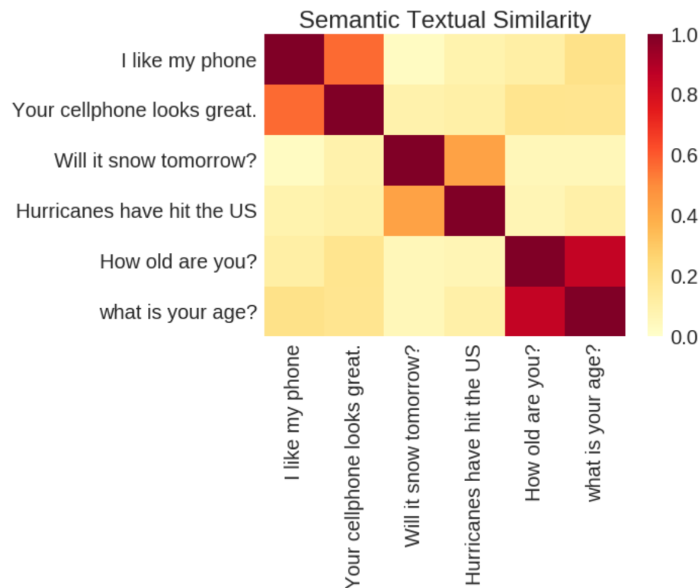


FIGURE 3.7: Similarity matrix between pairs of sentences by computing the similarity sore using universal sentence encoders (Cer et al. 2018).

Both models are trained to obtain general purpose encodings. This is accomplished by using multi-task learning where a single encoding model is used as the input for downstream learning tasks that include:

- **Skip-Thought task** (Kiros et al. 2015) to learn unsupervised representations of sentences.
- **A conversational input-response task** (Henderson et al. 2017) to learn the desired sentence response for an input sentence.
- **Supervised classification task** to learn the desired classification of an input sentence.

3.4.4 Bidirectional Encoder Representations from Transformers (BERT)

The *Bidirectional Encoder Representations from Transformers (BERT)* (Devlin et al. 2018) is a model based on the new architecture named *Transformers* (Vaswani et al. 2017).

A transformer is a new architecture that takes the attention architecture that was described in Figure 2.2.3: Attention Models, and removes the recurrence of the RNNs. The authors highlight an important practical drawback from the RNNs: they require sequential computation that grows with the length of the input and output sequences to be processed. This is not only expensive in practical computations but precludes the parallelized computation which in practice means that it can not take the advantage of parallel computation from GPUs.

The authors mention previous works that attempt to get improvements in this process such as factorization tricks and conditional computation, but those do not fix the constrain of the sequential computation.

The transformer removes the recurrence and instead relies completely on the attention mechanism which obtains significant improvements in the paralellization and achieves new state of the art results with a fraction of computing time compared to previous recurrent approaches.

This means that the dependencies between the input and output (including the long-range dependencies which are a weakness of recurrent models (even with LSTMs, GRUs and attention), are implemented just with attention.

Another problem detected by the authors is the fact that the words in the input may have different meaning depending on the context, that is, the meaning depends on the words in the input that appear before and after. The same happens to the dependencies of the words within the output. So for a sequence translation problem there are three types of dependencies:

1. Dependencies between the input and output tokens.
2. Dependencies between input tokens.
3. Dependencies between output tokens.

The classical attention mechanism solves the first type of the dependencies. The Transformer model generalizes that mechanism to model the dependencies on the second and third types.

The transformers introduce the concept of multi-headed attention. First, the attention mechanism can be generalized with the following notation:

$$A(q, K, V) = \sum_i \frac{exp(e_{qk_i})}{\sum_j exp(e_{qk_j})} v_i$$

Given a query q and a set of key-value pairs (K, V) the query determines which values V to focus (attend) on.

The transformer instead of relying just on a single attention pass over the input values (as with the classic attention model), computes multiple attentions. This gives the model the ability to focus on multiple parts of the input at the same time. Figure 3.8 shows a graphical representation.

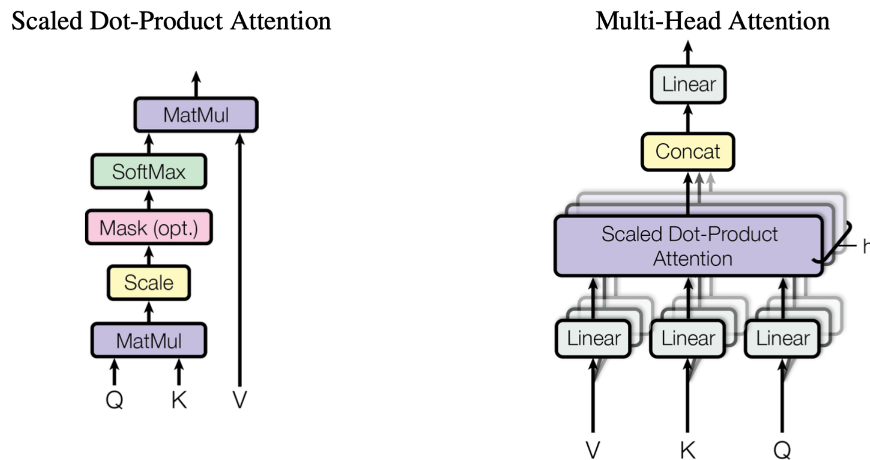


FIGURE 3.8: Graphical representation of a multi-headed attention (Vaswani et al. 2017).

The transformer keeps the encoder-decoder design. The embeddings of the input sequence are the inputs to the encoder (left) and the embeddings of the output sequence so far are inputed to the decoder. Both for the input embeddings and the output embeddings, the positional encoding is incorporated in order to give the model the sense of ordering in the sequences. This is necessary as the transformer does not use the

recurrency that the traditional encoder-decoder model has.

Both the encoder and the decoder are composed of N layers of blocks. Each of the blocks are composed of two sub-layers , the first sub-layer is a multi-headed attention and the second sub-layer is a feedforward network.

BERT's model architecture is basically a multi-layer bidirectional transformer encoder based on (Vaswani et al. 2017). The authors use different hyperparameters: L: the number of layers of transformer blocks, H: the hidden size and A: the number of attention heads. The report includes two model sizes, a base and a large with L=12, H=768, A=12 and L=24, H=1024, A=16 respectively.

The overall architecture is show in Figure 3.9

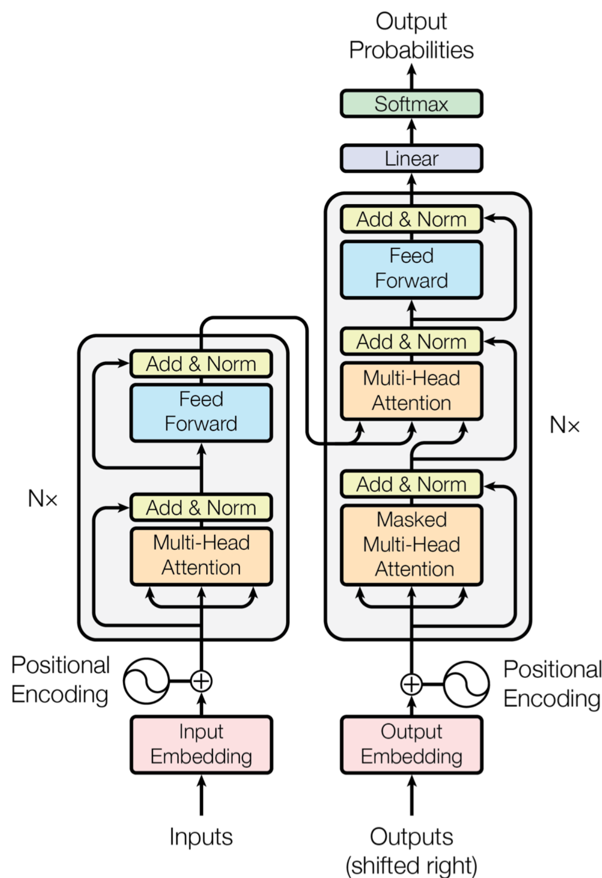


FIGURE 3.9: Graphical representation of the Transformer architecture (Vaswani et al. 2017).

For the purposes of the benchmarks Bert service (H. Xiao 2018) implementation will be used. This implementation uses BERT models developed by Google. In particular a *BERT-Base* model is used with dimensions L=12, H=768, A=12. The model was trained in an unsupervised approach, meaning, it was trained only on a plain text corpus (not annotated), which has the advantage of not requiring human labeling and can leverage large amounts of plain text data publicly available on the web. The training approach is done with two approaches:

Masked language modeling (MLM). The model will be trained by masking 15% of the words and learning to predict the masked words based on the rest of the words in the sentence. For example:

```
Input: the man went to the [MASK1] . he bought a [MASK2] of milk.
Labels: [MASK1] = store; [MASK2] = gallon
```

Next sentence prediction (NSP). Learn relationships between sentences by training a simple task where given two sentences A and B, predict if B is the actual next sentence that comes after A or if it's just a random sentence from the corpus. For example:

```
Sentence A: the man went to the store .
Sentence B: he bought a gallon of milk .
Label: IsNextSentence


Sentence A: the man went to the store .
Sentence B: penguins are flightless .
Label: NotNextSentence
```

One particular interesting feature about BERT is the fact that the model represents a a word with a deep bi-directional approach. As we described en previous sections, word embedding implementations such as word2vec and Glove generate a word embedding representation in a context-free approach, meaning for example that the word *bank* would have the same representation both for *bank deposit* and *river bank*.

The contextual models, on the other hand, will generate a representation of a word based on the other words in the sentence where the word appears. The contextual approach might have two further sub-approaches:

unidirectional or bidirectional. For example, in the sentence *I made a bank deposit*, the unidirectional approach would represent the word bank based on the left-context *I made*. On the other hand, the bidirectional approach will use both, the left and the right context, in this case the left context *I made* and the right context *deposit*.

BERT uses a bidirectional approach (compared to other models such as ELMo (Peters et al. 2018), Generative Pre-Training (GPT) (A. Radford 2018) and ULMFit (Howard and Ruder 2018).

Finally, in order to obtain a fixed length representation of a variable length sentence, Bert service uses an average pooling of the second to last hidden layer of the tokens in the sentence.

3.4.5 Sentiment Neuron

The last implementation used in this thesis is the one described in the paper from (A. Radford, Jozefowicz, and I. Sutskever 2017). The approach is to learn unsupervised representations of texts for sentiment analysis. The authors train a character level language model using an RNN, in particular, a *multiplicative LSTM (mLSTM)* over a very large corpus of product reviews (83 million product reviews from Amazon). The hypothesis is that these models can create good low level representations of text that could be useful to learn higher level concepts like sentiment.

An important hypothesis of the authors is that previous research on creating unsupervised representations may have had low performance because of two main combined problems:

1. First, often the unsupervised models are trained over a corpus that has a completely different domain than the one where the model is then used and tested. For example, a model trained over the *Wikipedia* corpus may have few overlapping with a corpus about e-commerce product opinions.

2. Second, the previous designs based on distributed sentence representations may have limited capacity to capture important syntactic and semantic concepts. This is why exploring a recurrent network at a character level could improve results.

The model processes a text sequence as characters. After processing a character of the sequence, it updates the hidden state and returns a

probability distribution to predict the next character in the sequence, very similiar to the character level model described previously in Figure 2.11, Subsection 2.2.3: Recurrent Neural Networks. Then the hidden states of the network should encode a meaningful representation of the sequence that then could be used as features for a supervised algorithm.

After various configurations, an mLSTM network of 4,096 units is trained. The authors observed that the mLSTM converged faster than the LSTM. The model was trained with minibatches of 128 examples of 256 characters each.

The dataset that the authors selected was a very large dataset about Amazon product reviews introduced by (McAuley, Pandey, and Leskovec 2015). The authors selected this dataset as it has various expressions on sentiment in opinions about products. The dataset has 82 million product reviews in total from May 1996 to July 2014 for a total of 28 billion bytes.

This kind of models (big recurrent networks on long sequences) require a large amount of computations, thus being very time consuming for its training. Adding the fact that will be trained over a very large dataset, the authors used four GPUs to parallelize the computation and speed up the training process. Even with all these optimizations, the authors declared that the training process of 1 epoch with 4 GPUs on the full corpus took approximately 1 month.

The model generated by the authors were tested with various datasets and compared with other top state of the art models outperforms the rest in two datasets, achieving competitive results in the rest. But the most interesting result that will be used in this thesis is its data efficiency, that is, how the model achieves high performance metrics (equal or higher to other algorithms) but with a few dozens of examples (while the rest of the algorithms require thousands of training examples).

The results are shown in Figure 3.10. This is a particularly interesting approach since it solves one of the motivations in this thesis: how to reduce the amount of labeled training data to create supervised sentiment models.

The model used in this thesis will be created using the character level mLSTM language model as feature extractor to vectorize the text examples and training a supervised logistic regression as the classification model. The logistic regression is adjusted using an L1 penalty which was reported by the authors to perform better in low volume training sets.

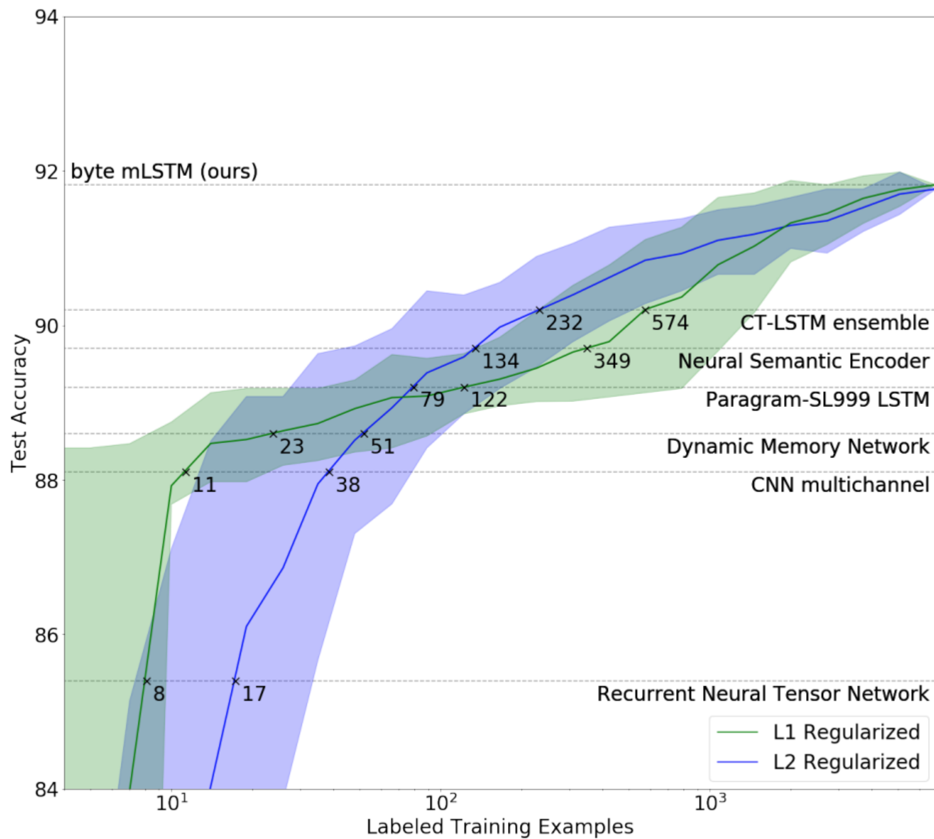


FIGURE 3.10: Performance of the Sentiment Neuron model compared with state of the art sentiment analysis on the Stanford Treebank dataset. The performance of each model is plotted as a function of the labeled training examples used (A. Radford, Jozefowicz, and I. Sutskever 2017). It can be seen that the L2 regularized version outperforms all the state of the art models with just 232 training examples, while the rest of the models use in the order of thousands of them.

3.5 Classification Model

It is worth mentioning that all the text classification models compared (including the baseline) will use the same supervised learning classification model: a simple linear regression.

The reason why that particular model was selected is that because of being a very simple model: the speed at which it can learn (both in terms of computing time and accuracy improvement) makes it a good choice compared to more complex models. This decision is inspired by the results shown in (A. Radford, Jozefowicz, and I. Sutskever 2017) where the same model was used to run the benchmarks.

3.6 Conclusions

In this chapter a baseline model for text classification and five different implementations that leverage transfer learning text vectorization techniques have been described.

In the next chapter, these models will be reproduced and compared. The goal will be, as stated in Chapter 1: Introduction, to measure and compare their data efficiency, that is, how fast they learn (performance improvement) as a function of the number of training examples used. A particular set of datasets from different domains will be used for this purpose.

Chapter 4

Datasets and Results

Chapter 3: Deep Learning for Text Classification described the six models that will be used in this thesis for a comparison of data efficient text classification methods for sentiment analysis. The first one is a classic baseline model (bag of n-grams). The remaining five use state of the art machine learning methods that leverage transfer learning techniques.

This chapter will compare those implementations over different datasets.

First, the particular text classification problem is defined: polarity classification in sentiment analysis. Polarity classification for sentiment analysis was selected as a good field to test data efficient text classification models.

Second, the six different datasets that will be used to run the benchmarks are introduced.

Third, the metrics that will be used to compare the implementations are defined.

Finally, we will show and analyze the results obtained for each implementation over each of the datasets.

4.1 Problem definition: Polarity classification for Sentiment Analysis

In this section a brief introduction to Sentiment Analysis is provided, particularly, polarity classification will be used as the test field for our data efficient text classification benchmarks. For a complete description, the reader shall look at (Pang and Lee 2008b; B. Liu 2010). In the following sections, the most useful definitions and concepts from the mentioned

works are presented.

4.1.1 What is an opinion?

(B. Liu 2010) states that text information can be broadly categorized into two main types: *facts* and *opinions*. Facts are objective expressions about entities and their properties and opinions are usually subjective expressions that describe people's sentiments, appraisals and feelings toward entities, events and their properties.

Sentiment analysis as many NLP problems can be seen from the computational point of view as a classification problem where two sub-topics have been studied:

- Classifying a sentence as subjective or objective (*subjectivity classification*)
- Classifying a sentence as expressing a positive or negative opinion (*sentiment or polarity classification*)

In an opinion, the targets can be: an object, their components, attributes and features. An opinionated object could be a product, service, individual, organization, event, topic, etc. For example, in the following opinion:

"The battery life of this camera is too short."

A negative opinion is expressed about a feature (battery life) of an object (camera).

Opinions can be done in two main ways: *direct appraisal* or *comparison opinion*. A direct appraisal, also known as *direct opinion* gives an opinion of an object directly, for example:

"The picture quality of this camera is poor."

In comparison opinions, the opinion is expressed by comparing an object with another, for example:

"The picture quality of this camera is better than that of Camera-x"

Usually comparative opinions express similarities or differences between two or more objects using comparative or superlative form of an adjective or adverb.

4.1.2 Formal Definitions

The following are some definitions extracted from (B. Liu 2010), that summarize useful concepts to understand the sentiment analysis polarity classification problem.

Definition: Object. An object o is an entity which can be a product, person, event, organization, or topic. An object may be composed by a set of components (parts) and a set of attributes.

Definition: Opinion holder. The holder of an opinion is the person or organization that expresses the opinion.

Definition: Opinion polarity. The orientation or polarity of an opinion on a feature f indicates whether the opinion is positive, negative or neutral.

Definition: Direct opinion. A direct opinion is a quintuple (o, f, oo, h, t), where o is an object, f is a feature of the object o, oo is the orientation or polarity of the opinion on feature f of object o, h is the opinion holder and t is the time when the opinion is expressed by h.

Some works add another attribute that describes the strength of an opinion. For example some opinions are very strong e.g.:

"This phone is a piece of junk."

and some are weak e.g.:

"I think this phone is fine."

Definition: Sentence subjectivity. An objective sentence expresses some factual information about the world, while a subjective sentence expresses some personal feelings or beliefs.

Definition: Explicit and implicit opinion. An explicit opinion on feature f is an opinion explicitly expressed on f in a subjective sentence. An implicit opinion on feature f is an opinion on f implied in an objective sentence. The following sentence expresses an explicit positive opinion:

"The voice quality of this phone is amazing."

The following sentence expresses an implicit negative opinion:

"The earphone broke in two days."

Within implicit opinions we could include metaphors that may be the most difficult type of opinions to analyze since they include much semantic information.

For the purposes of comparing different data efficient text classification implementations, this thesis will focus on the opinion polarity classification. The rationale is that this problem is a very good field to test data efficient classification models:

1. **The problem is complex.** Detecting the polarity of a sentence is not trivial since it requires both syntactic and semantic analysis of the text.
2. **The problem is well defined.** As we reviewed in the previous subsections, polarity classification is clearly defined.
3. **A good amount of public datasets are available.** These datasets are composed of texts manually tagged by human annotators with a consistent criteria. Besides, there is a good diversity of domains including: product reviews, restaurant reviews, politic comments,

airlines reviews, movies reviews. This will allow us to test the classification models with different domains and have a comparison between them.

The following section will describe the different datasets to be used, all of them contain sentences or multiple sentences that are all subjective: they express an opinion about a particular object with a polarity, either positive or negative.

4.2 Datasets

In order to compare the different methods, a set of different datasets were collected. Each dataset is composed by a set of opinions from different domains. Each opinion is composed by a single or multiple sentences, and is tagged with the corresponding polarity label (*positive* or *negative*).

Six sentiment analysis datasets from different domains were collected, the goal was to have a wide spectrum of different domains where different sentiment opinions are expressed:

1. **Movie reviews (Stanford Treebank)**
2. **Product reviews**
3. **Restaurant reviews**
4. **Hotel reviews**
5. **Political Twitter comments (GOP Debate)**
6. **Airline Twitter comments**

The goals for this selection are the following:

- **Domain diversity:** it is desired to have wide spectrum of different domains where different sentiment opinions are expressed. This way the analysis won't be biased to obtain good metrics for transfer learning vectorizations that were trained over a particular domain.
- **Same number of labels:** In order to have comparable results between the different datasets, the same number of labels will be enforced. In particular just the positive and negative labels will be

used. Part of the reason was a practical limitation due to some of the datasets been tagged with positive, negative and neutral, whereas others just with positive and negative. The simplification keeps only the samples tagged with positive or negative labels. It is worth mentioning that each sample has exactly one tag, that means, the comment or review is clearly positive or negative, and no multilabel tagging was performed. As a result the problem is modeled as a single label classification.

Each dataset has been partitioned into disjoint training and testing subsets in a stratified approach to get the same ratio of positive/negative examples in the training and testing sets. The following sections describe each dataset.

4.2.1 Movie Reviews (Stanford Treebank)

The Stanford Treebank corpus is based on the dataset introduced by (Pang and Lee 2008a) and consists of 11,855 single sentences extracted from movie reviews. It was parsed with the Stanford parser and tagged with sentiment by humans as part of the work of (Socher et al. 2013), with a total of 215,154 unique phrases from those parse trees, each annotated by 3 human judges. Table 4.1: Number of training and testing examples for Stanford Treebank dataset. shows the total amount of training and testing examples by category.

TABLE 4.1: Number of training and testing examples for Stanford Treebank dataset.

Label	Training Examples	Testing Examples
Negative	3,305	912
Positive	3,606	909
Total	6,911	1,821

TABLE 4.2: Examples of positive and negative reviews in the corpus.

Label	Example
Positive	Reno himself can take credit for most of the movie 's success .
Positive	Despite the film 's shortcomings , the stories are quietly moving .
Negative	The lead actors share no chemistry or engaging charisma .
Negative	It all comes down to whether you can tolerate Leon Barlow .

As we can see in Table 4.2: Examples of positive and negative reviews in the corpus., the opinions are about movies, usually referencing to performances of actors and directors in a movie. The corpus was originally collected from a movie review site Rotten Tomatoes[1].

[1] https://www.rottentomatoes.com/

4.2.2 Product Reviews

This dataset was collected from different e-commerce sites on the Internet where consumers post reviews about products.

Table 4.3: Number of training and testing examples for Product Reviews dataset.

Label	Training Examples	Testing Examples
Negative	2393	798
Positive	5909	1970

Table 4.4: Examples of positive and negative reviews in the corpus.

Label	Example
Positive	is the best buy I've made in my life. have a good sound and good sharpness. I recommend it 1.00
Positive	It replaces her nite-nite bedtime omega 3 that is no longer available. Ruby had been using the omega 3 from 3 monts on. She is now 3 years old. Switching this was worrisome to me, but apparently not to her.Thanks for quick service.
Negative	Arrived promptly but it is of poor quality. It does not hold a charge very long.
Negative	Consumer Reports must be loosing it. Why they recommended the Plantronics 510 I will never understand. This unit is not just heavy it is so big, one would have to have Dumbo ears to use it. I would return it if I could.

4.2.3 Hotel Reviews

This dataset was collected as part of this work by downloading hotel reviews from users of *TripAdvisor*[2] site. The reviews were labeled by humans with sentiment polarity using *Mechanical Turk*[3].

TABLE 4.5: Number of training and testing examples for Hotel Reviews dataset.

Label	Training Examples	Testing Examples
Negative	3,893	1,298
Positive	8,475	2,825
Total	12,368	4,123

TABLE 4.6: Examples of positive and negative reviews in the corpus.

Label	Example
Positive	Very nice, clean hotel! Staff were friendly and checking in and out was easy.
Positive	The hotel was a reasonable price for SF. The reception staff were very welcoming and helpful. The room decore was a little dated but clean and gave the hotel some character. I liked the hotel which was close to a bus route to give you access to the city. You could walk in to the city but was a little too far to be practical. All in all Id be happy to stay there again which is a good measure of a hotel in my book.
Negative	Low quality facilities. Service was alright. Overpriced though for small rooms and certainly not the cleanest. "Breakfast" not even worth it. I suggest doing a little more research and gliding over this one, you can certainly find a more comfortable and less dingy hotel with a little effort.
Negative	Was given a room that had not been serviced. Then got a new room where the tub would not drain, towels had stains. This is what you get when you go cheap.

As we can see in Table 4.6: Examples of positive and negative reviews in the corpus. the opinions are from users that express their experience during their stay in different hotels around the world. In particular, describing positive or negative sentiment about:

[2]https://www.tripadvisor.com/
[3]https://www.mturk.com/mturk/welcome

- service (general staff performance).
- facilities (ambiance, room, cleanliness, amenities, etc).
- price (value per money).

4.2.4 Restaurant Reviews

This corpus of restaurant reviews was collected as part of this work from the *Yelp*[4] site. The data was labeled with *Mechanical Turk*.

Table 4.7: Number of training and testing examples for Restaurant Reviews dataset.

Label	Training Examples	Testing Examples
Negative	2,552	851
Positive	10,351	3,451
Total		

Table 4.8: Examples of positive and negative reviews in the corpus.

Label	Example
Positive	The clam sauce pasta was awesome. Great place for a romantic evening. Great service. It can get a little busy, but totally worth the wait. I will be returning.
Positive	Came in for happy hour. Suprisingly it wasn't super busy. I had an enjoyable experience and would come back for another visit. Next time to try the lamb burger that I've heard so much of! The roast pork sandwich was what I ordered and it was very good! The also have daily specials so ask them about it!
Negative	Disgusting food that is saturated in salty sauce. Fried rice was burnt with tons of soy sauce. Not a place I will go back to. Too pricy & not worth the price. Also owner & waitress were very rude. My advice go to rose garden on the west side if not there chopstick on lomas.
Negative	Tried the food for the first time and felt like it was mediocre at best. I know Tucson is not the South, but I've had 100 times better in a dumpy BBQ joint in Memphis. I had the Sticky Ribs, slaw, bread and corn on the cob. The ribs were good. The slaw was OK. The bread and corn were flavorless and borderline disgusting. The ribs were worth it but otherwise very over priced.

As we can see in Table 4.8: Examples of positive and negative reviews in the corpus. opinions are from customers attending different restaurants. The opinions usually involve sentiment about:

[4]https://www.yelp.com/

- food (taste, quality, etc).
- service (waiter and general staff performance).
- facilities (ambiance, cleanliness).
- price (value per money).

4.2.5 Airline Twitter Comments

This dataset was downloaded from *CrowdFlower*[5] site. It contains tweets from airlines users that mention experiences with major airlines from the United States. It was scrapped from Twitter in 2015 and was tagged by humans with the CrowdFlower platform. The dataset originally have also the neutral tag, but for this work we only kept the examples with positive or negative labels.

TABLE 4.9: Number of training and testing examples for Airlines dataset.

Label	Training Examples	Testing Examples
Negative	6,884	2,294
Positive	1,772	591
Total	8,656	2,885

TABLE 4.10: Examples of positive and negative reviews in the corpus.

Label	Example
Positive	@USAirways Thank you. And thanks for being so accommodating.
Positive	@VirginAmerica , am I dreaming? Did you really just open up a route between Dallas and Austin?! And does this mean Houston might be next?
Negative	@SouthwestAir Why do airlines change ticket prices in the middle of the day #annoyed
Negative	@AmericanAir oh I did. All I received back was an email saying " delays happen".. Uh huh..

As we can see in Table 4.10: Examples of positive and negative reviews in the corpus. opinions are from customers expressing their experience with an airline company. The opinions usually involve sentiment about:

- In flight experience (food, comfort, etc).
- Service (stewardess and general staff performance).
- Timelines (changes/delays in departure and arrival times, connections).

[5]https://www.crowdflower.com/data-for-everyone/

- Price (value per money).

4.2.6 Political Twitter Comments (GOP Debate)

This dataset was obtained from *CrowdFlower*[6] site. It contains tens of thousands of tweets about the early August GOP debate in Ohio. Contributors were asked if the tweet was relevant, which candidate was mentioned, what subject was mentioned, and then what the sentiment was for a given tweet. The dataset contains 14,000 tagged tweets in total.

TABLE 4.11: Number of training and testing examples for GOP debate dataset.

Label	Training Examples	Testing Examples
Negative	6,370	2,123
Positive	1,677	559
Total	8,047	2,682

TABLE 4.12: Examples of positive and negative reviews in the corpus.

Label	Example
Positive	RT @FrankLuntz: John Kasich got a louder applause from the Q arena in Cleveland than LeBron James. #GOPDebate
Positive	Loved watching the #GOPDebate last night! We need @realDonaldTrump to #MakeAmericagreatagain. #inspiration
Negative	RT @billmaher: Jesus, this is the worst production of Glengarry Glen Ross I've ever seen #GOPDebate
Negative	Do you really think it's a coincidence that so many Fox News employees are defending Megyn Kelly right now? Orders. #tcot #GOPDebate

As we can see in Table 4.12: Examples of positive and negative reviews in the corpus. opinions are from general public expressing their observations during the debate. The opinions usually involve sentiment about:

- Opinions expressed by the candidates during the debate.
- General political opinions expressed by the author of the tweet.
- Opinions about the candidates.
- General opinions about the debate show and venue.

[6]https://www.crowdflower.com/data-for-everyone/

4.3 Metrics and Objectives

It is critical to choose the right metrics to evaluate the models. The design of the metrics will have a decisive impact on how the models are measured in performance. Some of the key points to take into account are summarized in the following sections.

4.3.1 Accounting for binary and imbalanced datasets

Because of the fact that the datasets are binary (labels can be either *positive* or *negative*, it should be expected that the models to be obtained should have high accuracy. For example, if we had a perfectly balanced dataset, e.g.: 50% of the examples are positive and 50% are negative, then if we just toss a coin, we should expect .50 accuracy on our random model. If, on the other hand, the dataset is imbalanced, e.g.: 95% of the examples are positive and 5% are negative. A random model should also expect .50 accuracy. But if we build a naive model that just predicts the *positive* label 100% of the time, then we would get a model with .95 accuracy!

That means that we have to choose a better metric than just the basic accuracy, otherwise we could potentially be giving a high score to a very poor model.

The usual option would be to use F1 score defined as:

$$F1 = 2 * (precision * recall)/(precision + recall)$$

Where precision and recall are the usual metrics:

- **Precision** is the ratio $TP/(TP + FP)$ where TP is the number of true positives and FP the number of false positives. Precision measures the ability of the classifier not to predict an example to be part of a category when it is not.
- **Recall** is the ratio $TP/(TP+FN)$ where TP is the number of true positives and FN the number of false negatives. Recall measures the ability of the classifier to correctly predict the examples that belong to a particular category.

The $F1$ score is a weighted average of the precision and recall, where an $F1$ score reaches its best value at 1 and worst value at 0. The relative contribution of precision and recall to the $F1$ score are equal. But when we have a multiple labels and the dataset is imbalanced, the $F1$ score will depend on the weighting that is provided to each class. Usually there are three ways to weight each class:

- **micro**: Calculate metrics globally by counting the total true positives, false negatives and false positives.
- **macro**: Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.
- **weighted**: Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters *macro* to account for label imbalance; it can result in an F-score that is not between precision and recall.

Given the nature of binary and imbalanced datasets, it is desired to give the same weight to each of the two classes, regardless of their support (number of examples a particular label has). As a result, the ***macro* average seems to be the most appropriate for this thesis**.

4.3.2 Accounting for imbalance in small subsets

The goal is to measure the performance of the models as a function of the amount of training examples. That means that the models will be trained with a growing amount of training data, starting with very small training examples (in the order of a few tens of training examples). In an imbalanced scenario (our case) it is important to use the right methods to keep the same imbalance in the training and testing subsets. The typical way is using a stratified sampling technique which will ensure that the subsets are made by preserving the percentage of examples for each class.

Finally, in small subsets, a particular random stratified partition could get much better performance than another random stratified partition, that could happen because of the particular examples selected, the performance of the model can be significantly higher or lower. This phenomenon is particularly present in small subsets, and can affect the performance metrics just by chance. In order to remediate this problem, besides performing stratification as explained before, we will be also doing k-folding,

training models for each stratified k-fold and get the average performance between all folds. That means that for a particular subset size, k different random stratified folds will be used to train k different models. Then the performance (F1 score) will be tested on a fixed testing set. Finally, the average F1 score is obtained, alongside its standard deviation. Figure 4.1 shows a graphical description of the process.

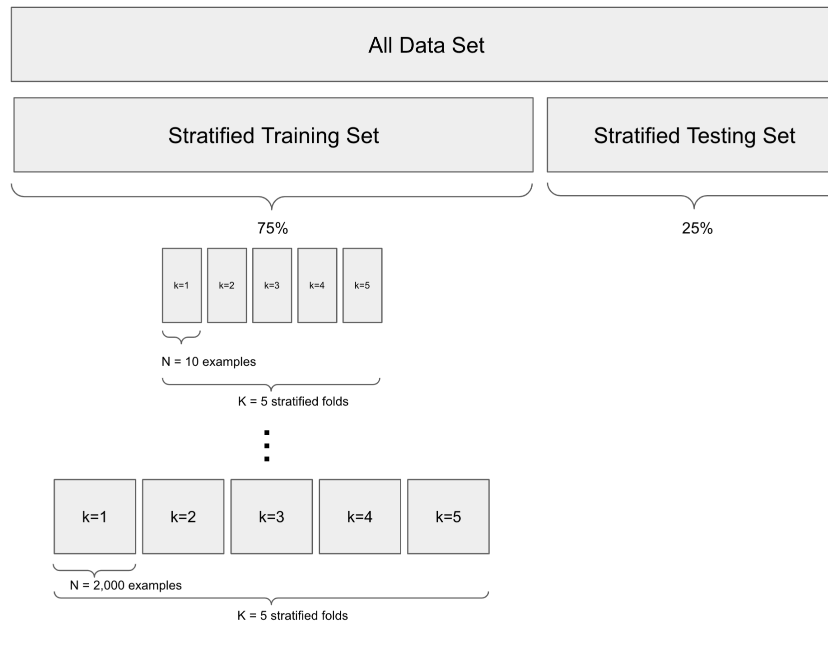


FIGURE 4.1: Graphical description of the training and testing partitions. N is the growing number of training examples. For each N, K=5 different stratified folds are used to train 5 instances of the model that are tested against the testing set. F1 scores for each N are obtained by averaging the K different instances.

The initial dataset (All Data Set) is first partitioned into two stratified and non-overlapping subsets: a training set with 75% of the examples and a testing set with 25% of the examples. Then the training set is divided into K stratified subsets (folds) with a growing size of N examples each. Then the experiment runs as follows: for a particular training set size N, 5 different instances of the model are trained with each of the stratified 5 folds. Then, each of those models are tested against the testing set to obtain their F1 score. Finally, the average F1 score and its standard deviation is calculated for each particular N by averaging the F1 scores of each of the folds. The process is repeated by growing N with the following sequence: N = 10, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240,

260, 280, 300, 320, 340, 360, 380, 400, 420, 440, 460, 480, 500, 750, 1000, 1250, 1500, 1750 and 2000.

4.4 Results and Analysis

This section defines the metrics to be used for a quantitative comparison of the six implementations. Finally these metrics will be obtained of each of the datasets and the six implementations.

The following notations will be used to reference the six benchmarked implementations:

1. **bag_ngrams**: the baseline model which uses bag of ngrams as vectorizer and a linear classification model. Described in Subsection 3.3.1: Bag of N-grams.

2. **fasttext**: the model that uses FastText word embeddings as vectorizer and a linear classification model. Described in Subsection 3.4.1: FastText.

3. **infersent**: the model that uses the InferSent representation as vectorizer and a linear classification model. Described in Subsection 3.4.2: InferSent.

4. **use**: the model that uses the Universal Sentence Encoder representation as vectorizer and a linear classification model. Described in Subsection 3.4.2: InferSent.

5. **bert**: the model that uses the BERT representation as vectorizer and a linear classification model. Described in Subsection 3.4.4: Bidirectional Encoder Representations from Transformers (BERT).

6. **sentiment_neuron**: the model that uses the Sentiment Neuron character level RNN as vectorizer and a linear classification model. Described in Subsection 3.4.5: Sentiment Neuron.

4.4.1 Movie Reviews (Stanford Treebank)

The first corpus tested is the Stanford Treebank. This corpus has been used by many research papers.

Table 4.13, and figures 4.2 and 4.3 show the F1 scores of the six models as a function of the number of training examples used to train the supervised model.

The results obtained are coherent with those presented by (A. Radford, Jozefowicz, and I. Sutskever 2017). As we can see in the figure the *bag_ngrams* model starts with an F1 score in the order of .50 with a few tens of training examples (i.e.: it is incapable of learning with so few examples). On the other hand, the *sentiment_neuron* model starts with a very good performance of around .65 with the same amount of labeled training examples.

The F1 scores on the six models keep improving as the amount of supervised training examples increases. The *sentiment_neuron* maintains a considerable higher F1 through the process ending with the full training set with .72 of F1 for the *bag_ngrams* model and .88 for the *sentiment_neuron* model. BERT is clearly the second best model, it also achieves significant improvements compared to the *bag_ngrams* and *fasttext* methods. At 100 training examples *bert* and *sentiment_neuron* achieve 29 percentual points and 46 percentual points more than the baseline. This clearly shows that both models leverages transfer learning capabilities to accelerate the learning process.

TABLE 4.13: F1 scores for Stanford Treebank dataset. The best performance is highlighted for milestones corresponding to 100, 500 and 2,000 training examples.

# training examples	bag of n-grams	bert	fasttext	infersent	sentiment neuron	use
10	.50	.59	.55	.54	.65	.57
20	.54	.61	.50	.61	.77	.61
40	.55	.66	.58	.70	.83	.65
60	.56	.70	.65	.72	.85	.66
80	.56	.74	.66	.73	.86	.69
100	.58	.77	.71	.75	**.86**	.70
150	.60	.77	.71	.77	.86	.71
200	.60	.79	.71	.78	.88	.70
250	.61	.78	.70	.78	.89	.72
300	.61	.80	.76	.77	.88	.70
350	.63	.80	.76	.79	.89	.72
400	.63	.80	.76	.79	.88	.72
450	.64	.80	.77	.79	.88	.73
500	.65	.80	.76	.79	**.88**	.73
750	.67	.81	.80	.79	.88	.75
1000	.69	.82	.80	.78	.87	.76
1250	.70	.83	.80	.79	.88	.76
1500	.71	.82	.80	.79	.88	.78
1750	.71	.83	.81	.79	.88	.77
2000	.72	.83	.80	.79	**.88**	.78

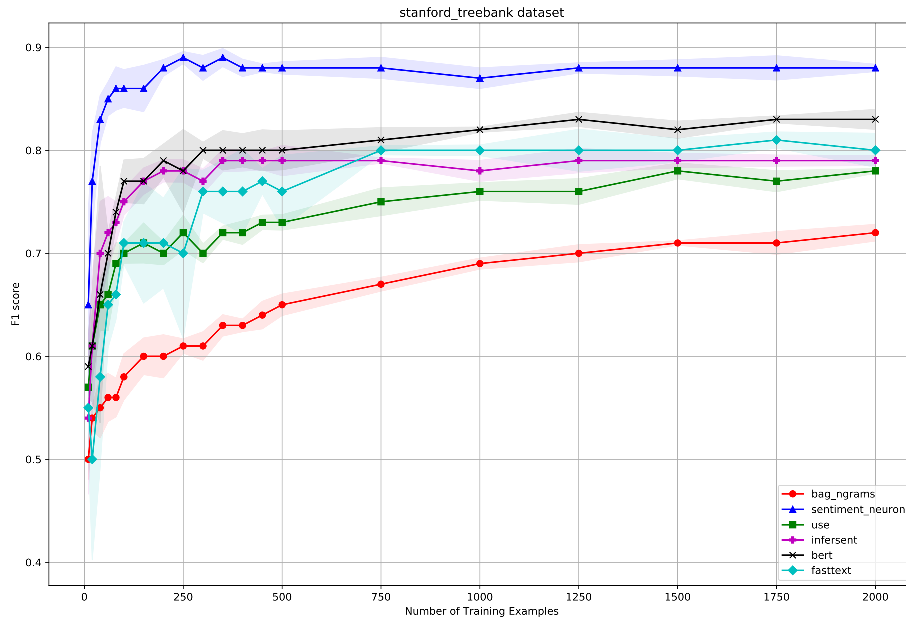


FIGURE 4.2: Stanford Treebank dataset F1 score results with full dataset. The lines are the average F1 score for the K folds, shade shows standard deviation.

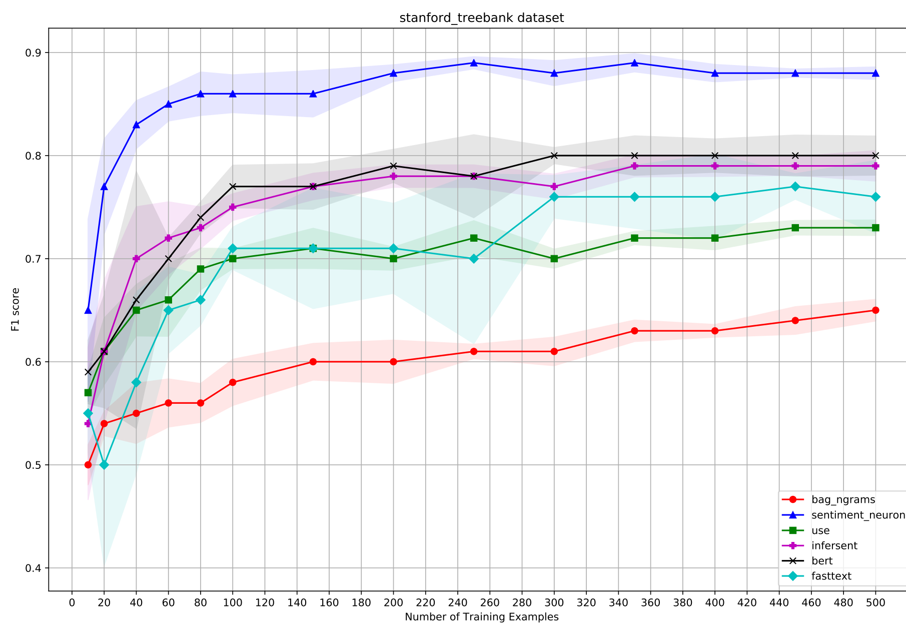


FIGURE 4.3: Stanford Treebank dataset F1 score results zoomed first 500 samples. The lines are the average F1 score for the K folds, shade shows standard deviation.

4.4.2 Product Reviews

Similar to the previous dataset, the *sentiment_neuron* achieves a considerable better performance than the rest of the methods. *infersent*, *use* and *bert* obtain very similar results getting the second best performance.

Table 4.14, and figures 4.4 and 4.5 show the F1 scores of the six models as a function of the number of training examples used to train the supervised model.

All the models improve their performance at 100 training examples, but clearly, the improvements are different: *sentiment_neuron* reaches .89, 46 percentual points more than the baseline *bag_ngrams*. *infersent*, *use* and *bert* reach similar F1 scores at around .70. *fasttext* reaches .60 while *bag_ngrams* does not obtain any improvements. At 1,000 training examples the four best transfer learning models keep improving reaching around .90. At 2,000 training examples, *sentiment_neuron* keeps the first place with .93 of F1 score.

TABLE 4.14: F1 scores for Product Reviews dataset. The best performance is highlighted for milestones corresponding to 100, 500 and 2,000 training examples.

# training examples	bag of n-grams	bert	fasttext	infersent	sentiment neuron	use
10	.45	.51	.42	.48	.67	.55
20	.43	.53	.31	.58	.82	.60
40	.43	.61	.50	.64	.85	.68
60	.43	.71	.48	.70	.86	.68
80	.43	.72	.48	.70	.89	.70
100	.43	.75	.60	.72	**.89**	.72
150	.46	.75	.67	.75	.91	.75
200	.50	.78	.59	.76	.92	.76
250	.54	.79	.62	.77	.92	.77
300	.57	.79	.66	.78	.92	.76
350	.61	.79	.62	.79	.92	.77
400	.63	.79	.67	.79	.92	.78
450	.65	.80	.65	.79	.93	.78
500	.67	.81	.69	.79	**.92**	.79
750	.72	.81	.62	.79	.92	.80
1000	.73	.81	.67	.79	.92	.80
1250	.75	.82	.67	.79	.92	.81
1500	.77	.81	.72	.80	.92	.81
1750	.77	.82	.68	.81	.92	.81
2000	.78	.82	.71	.80	**.92**	.81

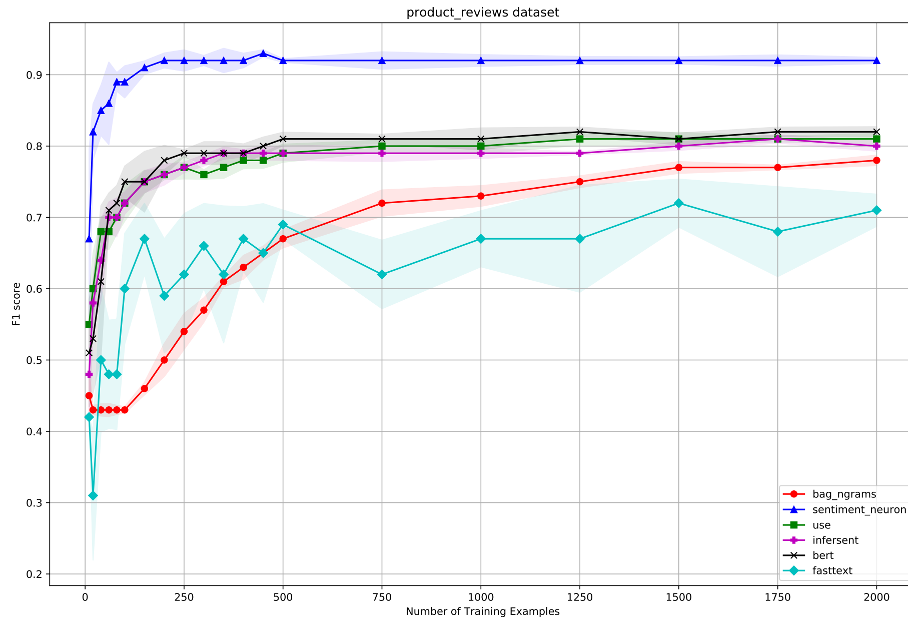


FIGURE 4.4: Product reviews dataset F1 score results with full dataset. The lines are the average F1 score for the K folds, shade shows standard deviation.

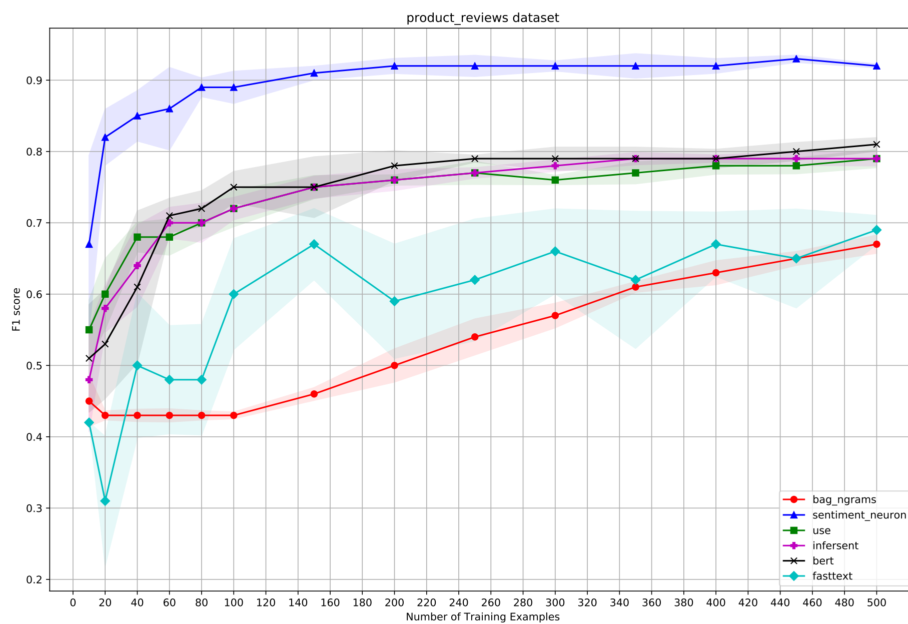


FIGURE 4.5: Product reviews dataset F1 score results zoomed first 500 samples. The lines are the average F1 score for the K folds, shade shows standard deviation.

4.4.3 Hotel Reviews

In this dataset the four best transfer learning models *sentiment_ neuron*, *infersent*, *use* and *bert* obtain a clear improved compared to the baseline with similar results.

Table 4.15, and figures 4.6 and 4.7 show the F1 scores of the six models as a function of the number of training examples used to train the supervised model.

At 100 training examples, clearly demonstrate a much better performance with a mark of .90 F1 score, almost 45 points more than the *bag_ ngrams* model. At the 1,000 training examples mark, the four first models do not get significant improvement as they get their best performance, achieving in the range of .91 of F1 score while *bag_ ngrams* reaches around .88.

TABLE 4.15: F1 scores for Hotels Reviews dataset. The best performance is highlighted for milestones corresponding to 100, 500 and 2,000 training examples.

# training examples	bag of n-grams	bert	fasttext	infersent	sentiment neuron	use
10	.42	.66	.42	.64	.59	.80
20	.41	.80	.46	.78	.73	.82
40	.44	.81	.61	.84	.86	.86
60	.41	.85	.62	.88	.88	.87
80	.43	.88	.68	.88	.86	.87
100	.45	.88	.75	**.90**	**.90**	.88
150	.59	.87	.74	.90	.91	.88
200	.70	.90	.77	.91	.92	.89
250	.74	.90	.82	.91	.93	.90
300	.76	.89	.83	.91	.93	.89
350	.78	.91	.84	.92	.94	.90
400	.80	.90	.84	.92	.93	.90
450	.83	.90	.82	.92	.94	.90
500	.83	.91	.86	.92	**.93**	.90
750	.86	.91	.86	.92	.93	.91
1000	.88	.91	.87	.92	.93	.91
1250	.89	.91	.87	.92	.94	.91
1500	.89	.91	.86	.92	.93	.91
1750	.89	.92	.86	.92	.94	.91
2000	.90	.92	.86	.92	**.93**	.92

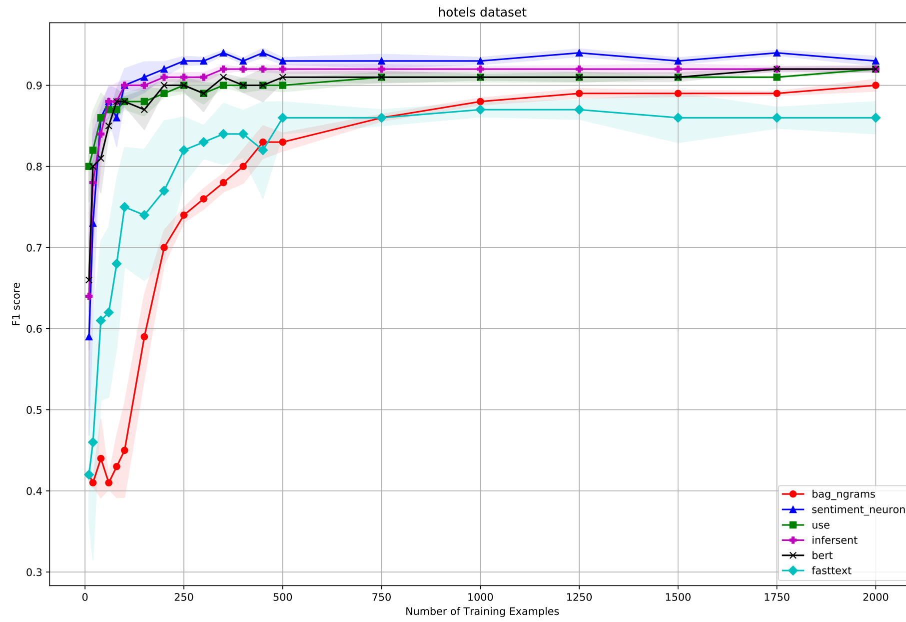


FIGURE 4.6: Hotel reviews dataset F1 score results with full dataset. The lines are the average F1 score for the K folds, shade shows standard deviation.

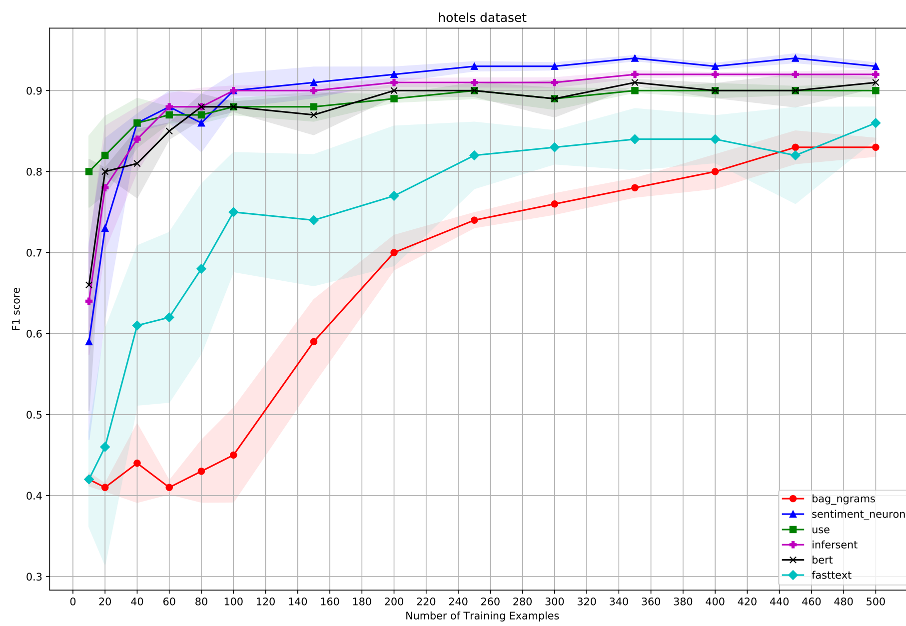


FIGURE 4.7: Hotel reviews dataset F1 score results zoomed first 500 samples. The lines are the average F1 score for the K folds, shade shows standard deviation.

4.4.4 Restaurant Reviews

With the restaurant reviews, the previous patterns are repeated again, the four best transfer learning models *sentiment_neuron*, *infersent*, *use* and *bert* obtain the best performance with similar results between them.

Table 4.16, and figures 4.6 and 4.7 show the F1 scores of the six models as a function of the number of training examples used to train the supervised model.

At the 100 examples mark *sentiment_neuron* obtains the best performance with .89, 44 points higher than the baseline *bag_ngrams* which achieves .45.

The *fasttext* model obtains a poor perfrmance, even compared to the *bag_ngrams* model. At 500 training examples, *sentiment_neuron* is still the best model with .92, 19 points above the baseline.

TABLE 4.16: F1 scores for Restaurants Reviews dataset. The best performance is highlighted for milestones corresponding to 100, 500 and 2,000 training examples.

# training examples	bag of n-grams	bert	fasttext	infersent	sentiment neuron	use
10	.45	.52	.41	.48	.73	.62
20	.45	.69	.47	.58	.70	.75
40	.45	.76	.55	.75	.86	.82
60	.45	.75	.54	.83	.88	.84
80	.45	.81	.58	.82	.90	.85
100	.45	.80	.66	.86	**.89**	.85
150	.47	.83	.64	.87	.90	.86
200	.55	.86	.73	.89	.91	.86
250	.60	.84	.70	.89	.91	.86
300	.62	.86	.77	.89	.92	.88
350	.66	.87	.73	.90	.92	.88
400	.68	.87	.75	.90	.92	.88
450	.71	.87	.74	.90	.92	.88
500	.73	.86	.76	.90	**.92**	.88
750	.77	.88	.80	.90	.93	.89
1000	.80	.88	.77	.90	.93	.90
1250	.83	.88	.75	.90	.92	.90
1500	.84	.89	.78	.90	.93	.90
1750	.85	.89	.78	.90	.93	.90
2000	.86	.89	.79	.90	**.93**	.90

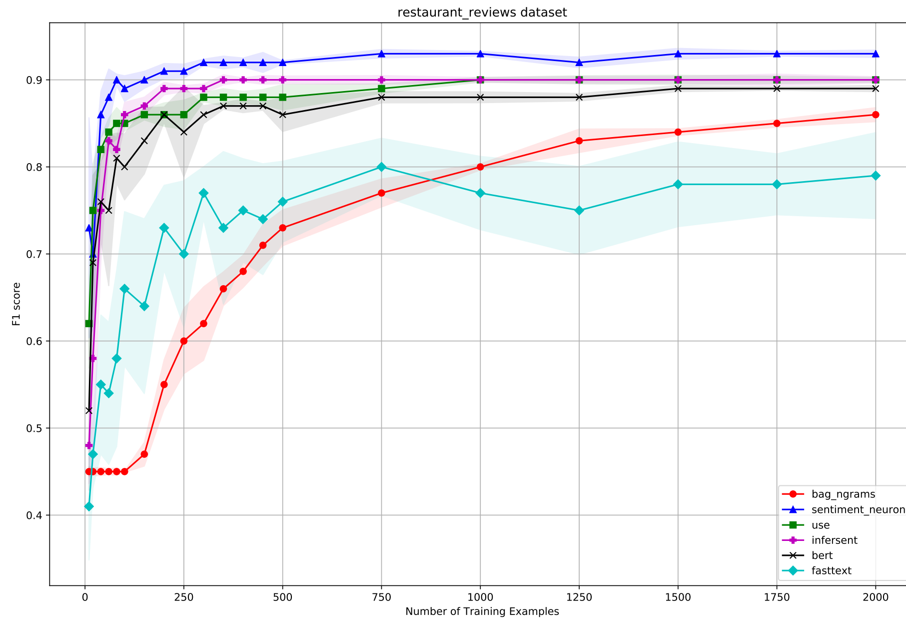


FIGURE 4.8: Restaurant reviews dataset F1 score results with full dataset. The lines are the average F1 score for the K folds, shade shows standard deviation.

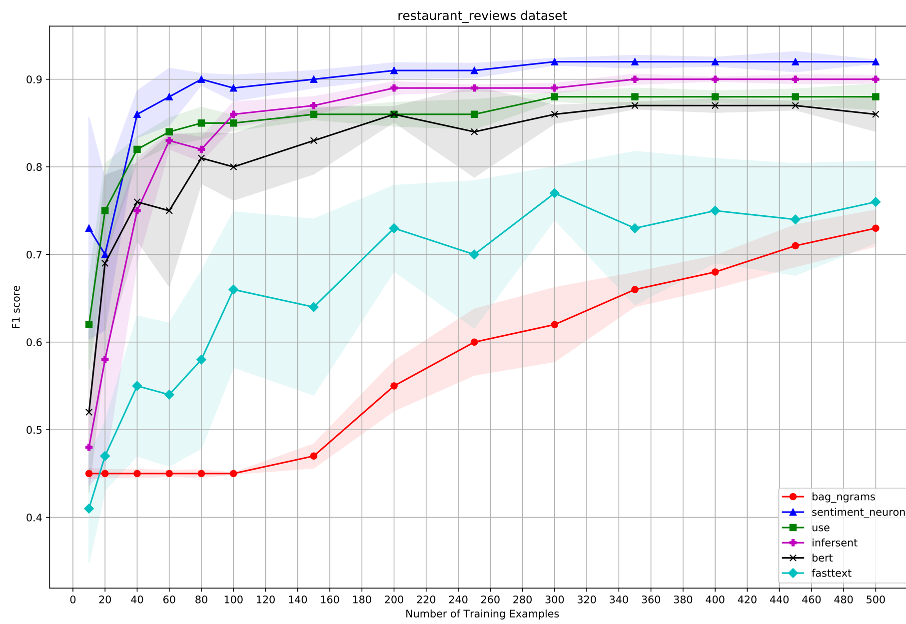


FIGURE 4.9: Restaurant reviews dataset F1 score results zoomed first 500 samples. The lines are the average F1 score for the K folds, shade shows standard deviation.

4.4.5 Airlines Twitter Comments

With the airlines corpus happens something similar to the hotel reviews corpus in the sense that the top three transfer learning models start with a similar level of performance (although *sentiment_neuron* in this case is not the best).

Table 4.17, and figures 4.10 and 4.11 show the F1 scores of the six models as a function of the number of training examples used to train the supervised model.

use, *bert* and *infersent* end up in the first place, with the same level of F1 score at .87. As we can see, this level is slightly lower than in the other corpuses, probably because the domain of this corpus (tweets about airlines) is significantly different than the previous and probably harder, since tweets usually have additional artifacts. Still the transfer learning models, achieve a pronounced difference of 23 points, 14 points and 8 points at 100, 500 and 2,000 training examples respectively.

TABLE 4.17: F1 scores for Airlines Comments dataset. The best performance is highlighted for milestones corresponding to 100, 500 and 2,000 training examples.

# training examples	bag of n-grams	bert	fasttext	infersent	sentiment neuron	use
10	.54	.51	.51	.59	.49	.60
20	.53	.62	.59	.69	.60	.70
40	.55	.65	.64	.75	.69	.77
60	.58	.64	.64	.81	.74	.80
80	.61	.78	.70	.80	.75	.82
100	.61	.78	.75	.83	.76	**.84**
150	.64	.83	.74	.83	.76	.83
200	.64	.81	.78	.84	.81	.83
250	.65	.84	.77	.84	.80	.84
300	.67	.86	.80	.85	.82	.84
350	.68	.84	.78	.85	.82	.84
400	.70	.85	.77	.85	.82	.84
450	.71	.84	.81	.85	.83	.85
500	.72	**.86**	.81	.85	.82	.85
750	.74	.87	.83	.86	.83	.86
1000	.76	.87	.83	.86	.84	.86
1250	.77	.87	.81	.86	.83	.87
1500	.77	.88	.82	.86	.84	.87
1750	.78	.88	.81	.87	.85	.87
2000	.79	**.87**	.83	**.87**	.84	**.87**

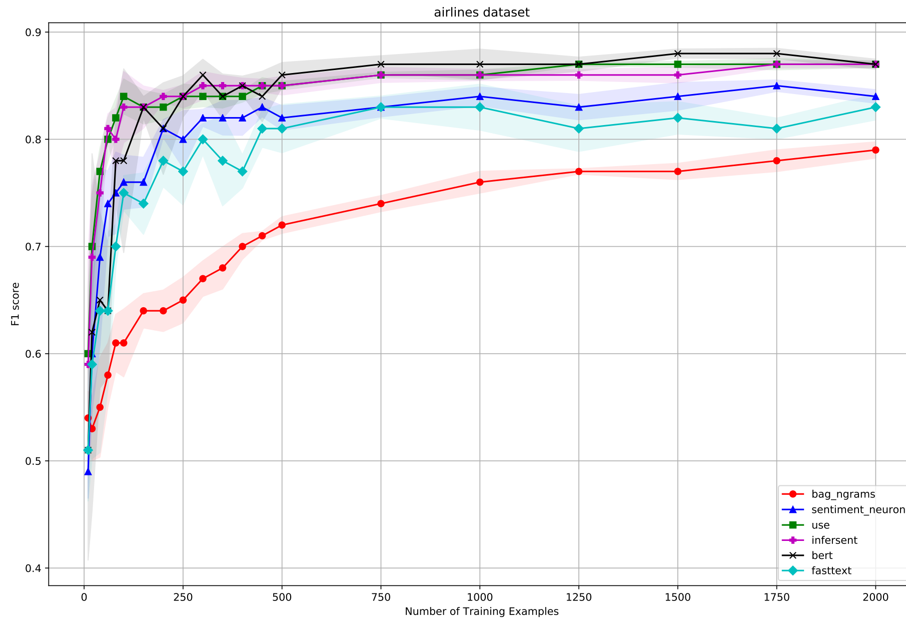


FIGURE 4.10: Airline comments dataset F1 score results with full dataset. The lines are the average F1 score for the K folds, shade shows standard deviation.

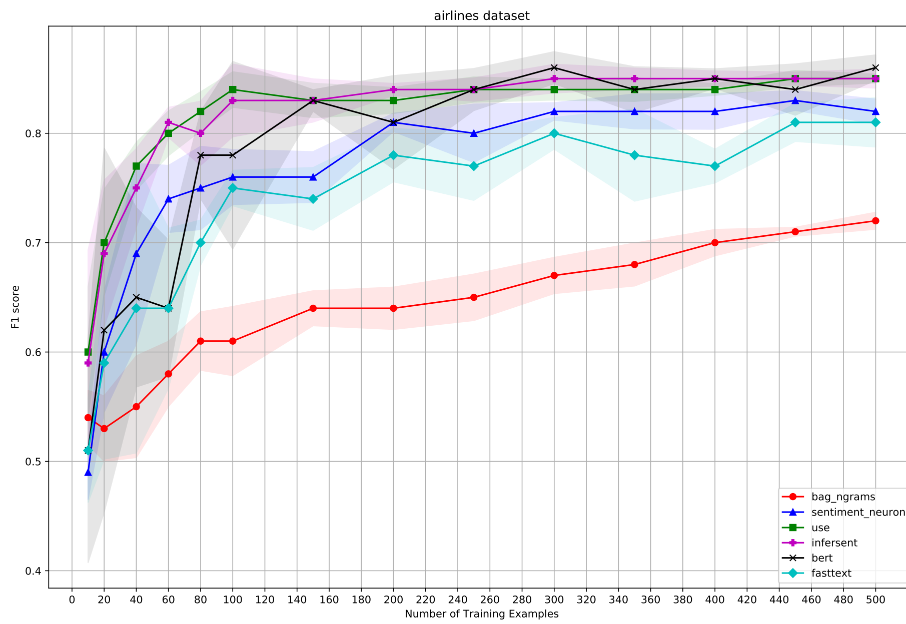


FIGURE 4.11: Airline comments dataset F1 score results zoomed first 500 samples. The lines are the average F1 score for the K folds, shade shows standard deviation.

4.4.6 Political Twitter Comments (GOP Debate, the Outlier)

With the politics dataset the results are very different than with the rest of the datasets. In contrast with the previous cases, the first three models start with a better accuracy than the *sentiment_neuron* model. Then all the models keep improving as more training examples are fed. All of them achieve very similar levels of performance ending with very similar accuracies with all the training set. In particular, the *bag_ngrams* achieves the second place.

Table 4.18, and figures 4.12 and 4.13 show the F1 scores of the six models as a function of the number of training examples used to train the supervised model.

This could mean that none of the models could leverage the advantages of transfer learning. The reason could be due to the fact that the source corpuses used to learn the text representations might be of a very different domain than politics comments in Twitter.

TABLE 4.18: F1 scores for GOP Debate Comments dataset. The best performance is highlighted for milestones corresponding to 100, 500 and 2,000 training examples.

# training examples	bag of n-grams	bert	fasttext	infersent	sentiment neuron	use
10	.46	.45	.46	.47	.47	.47
20	.47	.50	.49	.52	.51	.53
40	.47	.50	.51	.55	.54	.58
60	.50	.58	.53	.58	.58	.59
80	.52	.53	.59	.63	.59	.62
100	.53	.55	.56	**.63**	.59	.62
150	.55	.65	.57	.66	.61	.64
200	.56	.65	.60	.68	.64	.66
250	.56	.63	.56	.68	.63	.66
300	.58	.66	.59	.69	.66	.66
350	.59	.67	.60	.70	.67	.67
400	.59	.68	.61	.69	.67	.68
450	.60	.71	.57	.69	.66	.69
500	.60	.68	.60	**.70**	.67	.68
750	.61	.72	.63	.70	.69	.70
1000	.63	.72	.62	.71	.70	.69
1250	.64	.73	.61	.71	.70	.71
1500	.64	.73	.63	.72	.70	.72
1750	.66	.73	.64	.73	.70	.73
2000	.66	**.74**	.61	**.74**	.69	.71

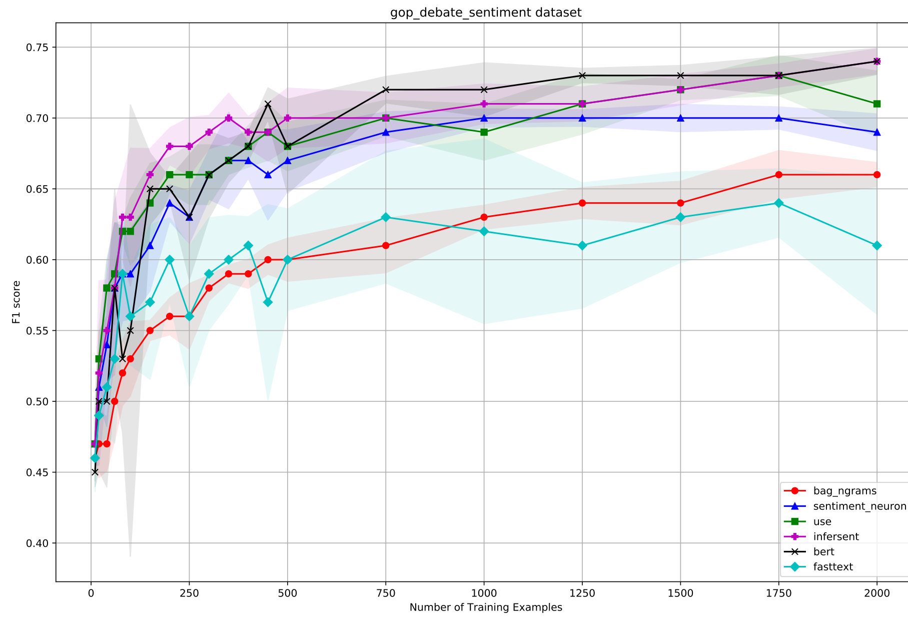


FIGURE 4.12: GOP debate dataset F1 score results with full dataset. The lines are the average F1 score for the K folds, shade shows standard deviation.

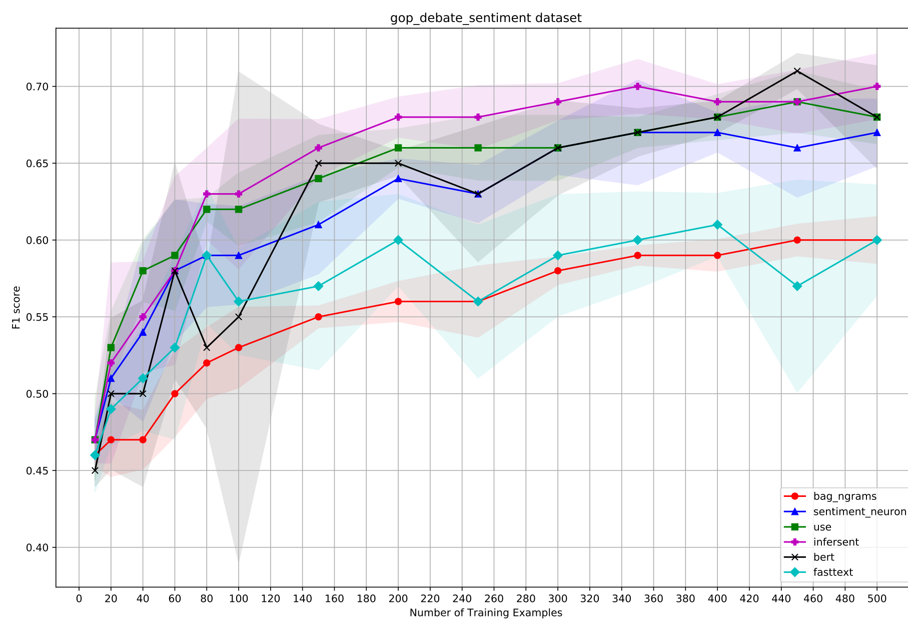


FIGURE 4.13: GOP debate dataset F1 score results zoomed first 500 samples. The lines are the average F1 score for the K folds, shade shows standard deviation.

4.4.7 Comparison of Datasets per Model

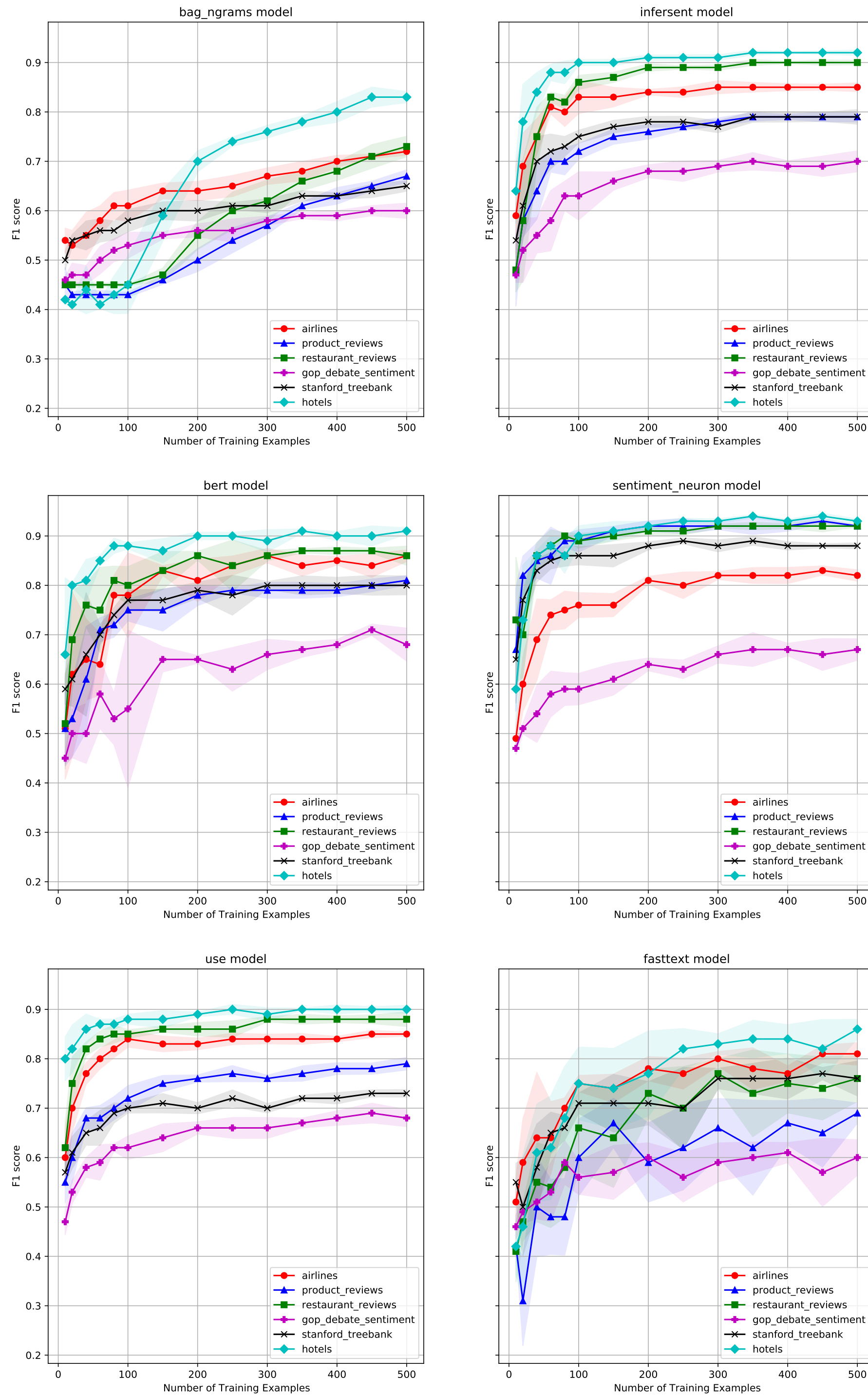


FIGURE 4.14: Comparison of the dataset results, per model.

Chapter 5

Conclusions

In this chapter the general discussion and conclusions of the experiments performed are stated. It also proposes future paths of research not included in the scope of this thesis.

5.1 General Conclusions

In this thesis a deep analysis of different transfer learning techniques for text classification applications have been performed.

Chapter 1: Introduction started with the motivation of this work, based on the need for data efficient classification models for practical applications. Sentiment analysis is used as a good testing ground to prove that, in particular, transfer learning techniques can be used to solve this problem, which established the objectives to compare the different techniques in different scenarios (datasets).

Chapter 2: Deep Learning Background, provided an overview of the evolution and state of the art of deep learning techniques presenting the fundamentals in order to dive deeper into the particular text classification techniques used in the following chapters.

Chapter 3: Deep Learning for Text Classification presented various practical implementations of text classification that leverage deep learning and transfer learning techniques. First, the most popular text vectorization techniques were reviewed, including bag of n-grams which is used as a baseline model. Then an introduction to the transfer learning fundamentals were presented. Finally in that chapter, the particular implementations of text classification that leverage transfer learning techniques were presented.

In Chapter 4: Datasets and Results, the metrics to be used for the benchmarks were introduced alongside important considerations to take into account to select the metric that could accurately measure the performance of each model, in alignment with the thesis objectives. The F1 score with macro averaging was selected. Besides, a stratified k-folding technique was used to generate the subsets to train the models. Both the metrics and the subsampling techniques take into account practical issues related to using small datasets with imbalanced sets of labels. Then the six datasets to run the benchmarks were introduced, the selection of the datasets aimed to generate diversity in domains in order to have a good representation of real world problems. Finally in that chapter, the results of the benchmarks are presented, including F1 scores for each training set size for each of the combinations of datasets and transfer learning models used.

As a result of this thesis, the following sections establish the major contributions and conclusions obtained from this work.

5.1.1 Transfer Learning improves data efficiency

We showed that transfer learning approaches can be used for text classification in order to obtain models that are more data efficient (require smaller training sets) than models that learn text representations from scratch.

From the results shown in section Section 4.4: Results and Analysis, we can clearly observe that transfer learning techniques quickly obtain substantial better F1 score improvements than the baseline. With just a few training examples (under 100 training examples), the transfer learning techniques obtain double digit improvements in the performance. Even at higher training set sizes, 500 training examples, and 1,000 training examples, the transfer learning techniques show substantial improvements compared to a bag of n-grams approach.

What every model implementation has in common is the fact that they use the same supervised technique: a simple linear classifier. That means that the only difference between the different approaches is the underlying text vectorization. It was shown that the previous knowledge learned by the different vectorizations helped each model to learn much faster than using a bag of n-grams approach.

5.1.2 Sentiment neuron model achieves best performance

It is interesting to note that for 4 out of the 6 datasets tested, the sentiment neuron technique got the best performance, both in learning faster than the rest of the models (achieving higher F1 scores with small amounts of training data), but also the final score with the largest training size.

What is even more interesting is the fact that the sentiment neuron, as explained in Subsection 3.4.5: Sentiment Neuron, was trained with a complete unsupervised approach: it was not required to tag data with human annotators. Different from the rest of the transfer learning techniques that got significant results (USE, InferSent and BERT) that require a large corpus of human annotated data. InferSent requires paired sentences, USE uses a combination of un-tagged and tagged datasets as well as BERT.

As a result, the sentiment neuron technique definitely has an extra advantage: it just needs a very large un-tagged corpus to be trained, which usually is something relatively cheap to obtain. The problem though, is that from the paper presented by the authors and the current implementation it is not clear how those results can be reproduced (retrain the model in different corpus). First, the process of training the model required very large computation power, as the authors declared that it took multiple GPU cores running for around 30 days. Second, it is not yet clear how the sentiment neuron emerged from just training a character level language model. This point is connected to one of the suggested future works presented in the following section.

5.1.3 Domain where representations are trained affect results

The dataset where all the models (even the baseline) obtained very similar results was the GOP debate (opinions about politics). An intuitive reason of this result could be that the text representations of the different models where transferred from domains that are not similar to politics (e.g.: movie reviews, restaurants reviews, product reviews, hotel reviews and airline comments). On the other hand, the results for the transfer learning models where much better than the baseline in target domains similar to the source domain. As a result, it could be concluded that the

domain of the source corpus where the transfer learning model is trained affects its potential to be used in other target domains.

5.1.4 Benchmarks with code and data reproducibility

As a result of the analysis, besides the conclusions stated before, this thesis generated two practical contributions:

- A set of 6 different datasets to train and test text classification models in different domains were gathered. This provides easy access to other works that need to use datasets to reproduce this benchmark or work on new implementations.
- A Python 3 code base was generated with the benchmarks that reproduce the implementations described in Chapter 3: Deep Learning for Text Classification. This is very important to actually test and compare the implementations.

Both the datasets and the code have been published with the corresponding references to their authors[1].

5.2 Future Work

The following are future paths of research that could complement the transfer learning techniques to obtain even better results.

5.2.1 Representations in other languages

It would be interesting to reproduce the success of transfer learning techniques in other languages such as Spanish, for example, by training a language model such as the Sentiment Neuron or BERT but with a large corpus in Spanish. In particular for the BERT implementation, HuggingFace (Wolf et al. 2020) has trained multiple BERT models using public web corpus in different languages. In particular, it would be interesting to test the performance of the benchmarks with a multilingual approach (training a single model that recognizes multiple languages). HuggingFace has trained a multilingual model on the top 104 languages with the

[1] https://github.com/raulgarreta/data-efficient-text-classification

largest Wikipedia using an masked language modeling approach. This is particularly interesting for applications where the same classification has to be done on top of texts with different languages. The advantage: just a single text vectorization model must be trained which massively contributes to data efficiency.

5.2.2 Representations in other domains

As mentioned in the previous conclusions, the election of the domain of the corpus of the source domain in the transfer learning process can have impact on the effectiveness. In our case, the knowledge transferred from a corpus about product opinions is not as effective in a target domain such as politics. It would be interesting to validate this hypothesis by transferring knowledge from a politics domain and confirm that then the effectiveness is similar to what we obtained in the rest of the domains.

5.2.3 Representations in a generic domain

Very related to the previous item, it would be interesting to validate if using a generic corpus as the source of knowledge, e.g.: a corpus that combines opinions in different domains like e-commerce, hospitality, politics, etc, can be as effective as having focused source domains for the target task. This would provide to be very practical as a way to have a universal text representation for multiple text classification domains and problems.

5.2.4 Active learning

If the transfer learning models such as the sentiment neuron get very significant improvements in performance with a few training examples, intuitively, the training examples that are selected and the order in which they are fed into the model could make an important change in the speed at which the model improves its performance. As a result, it would be interesting to test *active learning* techniques to try to obtain optimal policies on how to choose the training examples to be fed to the supervised stage of the models. This could improve the data efficiency.

5.2.5 Zero-shot learning

In this thesis we researched how to create data efficient text classification models to reduce the number of supervised training examples necessary to train the model. This approach has been recently known also as *Few-shot learning* (Brown et al. 2020a). The extreme situation would be a classification model that does not need any supervised training examples at all, also known as *Zero-shot learning* (Yin, Hay, and Roth 2019). In this extreme case, the model does not observe any samples with the corresponding labels at training time, but it is directly used to predict labels. Recently there has been a good progress in this direction by using natural language inference models, where the classification task is modeled as an inference of the class (hypothesis) from the input text (premise). In order to build these models, we must have an inference model that takes a pair of sequences A (premise) and B (hypothesis), being A the sequence to classify and B the description of the label to predict. The task is to determine whether the hypothesis B is true (entailment) or false (contradiction) given the premise A. For example, in our case of sentiment polarity classification, the pair of sequences would be:

```
A:"I love this product"
B:"The user is expressing a compliment about a product." (Positive)

A:"The user interface is very easy to use "
B:"The user is expressing a compliment about a product." (Positive)

A:"It was very hard to set up the product, it took days just to build!"
B:"The user is expressing a complaint about a product." (Negative)
```

Recent research using Generative Pre-Trained (GPT) language models trained on massive amounts of corpus has generated impressive results including GPT-2 (Alec Radford et al. 2019) and GPT-3 (Brown et al. 2020b). Other approaches use smaller models based on pre-trained BERT and fine tuning the inference task with the Multi-genre NLI (MNLI) (A. Williams, Nangia, and Bowman 2017) corpus.

5.2.6 Transfer learning in other NLP problems

Lastly, it would be interesting to test the same techniques but for different NLP problems:

- Other text classification problems including:
  - Topic classification: classify texts into a pre defined set of topics or themes, eg: classify items by product category, classify company descriptions by industry.
  - Intent classification: classify text into a pre defined set of intents, eg: Is the customer expressing urgency? is the customer asking for help? is the customer requesting a new functionality?
- Sequential classification problems including:
  - Entity extraction: extract people names, company names, locations, events.
  - Keyword extraction: extract the most relevant multi-word keywords or keyphrases in a text.

Bibliography

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). *Neural Machine Translation by Jointly Learning to Align and Translate*. URL: http://arxiv.org/abs/1409.0473.

Bengio, Yoshua et al. (Mar. 2003). "A Neural Probabilistic Language Model". In: *J. Mach. Learn. Res.* 3, pp. 1137–1155. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944919.944966.

Bird, Steven, Ewan Klein, and Edward Loper (2009). *Natural Language Processing with Python*. 1st. O'Reilly Media, Inc. ISBN: 0596516495, 9780596516499.

Bowman, Samuel R. et al. (2015). "A large annotated corpus for learning natural language inference". In: *CoRR* abs/1508.05326. arXiv: 1508.05326. URL: http://arxiv.org/abs/1508.05326.

Brown, Tom B. et al. (2020a). *Language Models are Few-Shot Learners*. arXiv: 2005.14165 [cs.CL].

— (2020b). *Language Models are Few-Shot Learners*. arXiv: 2005.14165 [cs.CL].

Cer, Daniel et al. (2018). "Universal Sentence Encoder". In: *CoRR* abs/1803.11175. arXiv: 1803.11175. URL: http://arxiv.org/abs/1803.11175.

Cho, KyungHyun et al. (2014). "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *CoRR* abs/1409.1259. arXiv: 1409.1259. URL: http://arxiv.org/abs/1409.1259.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078. arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

Collobert, Ronan and Jason Weston (2008). "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, pp. 160–167. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390177. URL: http://doi.acm.org/10.1145/1390156.1390177.

Collobert, Ronan, Jason Weston, et al. (2011). "Natural Language Processing (almost) from Scratch". In: *CoRR* abs/1103.0398. arXiv: 1103.0398. URL: http://arxiv.org/abs/1103.0398.

Conneau, Alexis, Douwe Kiela, et al. (2017). "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: *CoRR* abs/1705.02364. arXiv: 1705.02364. URL: http://arxiv.org/abs/1705.02364.

Conneau, Alexis, Holger Schwenk, et al. (2016). "Very Deep Convolutional Networks for Natural Language Processing". In: *CoRR* abs/1606.01781. arXiv: 1606.01781. URL: http://arxiv.org/abs/1606.01781.

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

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

Henderson, Matthew L. et al. (2017). "Efficient Natural Language Response Suggestion for Smart Reply". In: *CoRR* abs/1705.00652. arXiv: 1705.00652. URL: http://arxiv.org/abs/1705.00652.

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1997). "Long Short-Term Memory". In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Howard, Jeremy and Sebastian Ruder (2018). "Fine-tuned Language Models for Text Classification". In: *CoRR* abs/1801.06146. arXiv: 1801.06146. URL: http://arxiv.org/abs/1801.06146.

Iyyer, Mohit et al. (July 2015). "Deep Unordered Composition Rivals Syntactic Methods for Text Classification". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1681–1691. DOI: 10.3115/v1/P15-1162. URL: https://www.aclweb.org/anthology/P15-1162.

Joachims, Thorsten (1997). "A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization". In: *Proceedings of the Fourteenth International Conference on Machine Learning*. ICML '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 143–151. ISBN: 1-55860-486-3. URL: http://dl.acm.org/citation.cfm?id=645526.657278.

— (1998). "Text Categorization with Suport Vector Machines: Learning with Many Relevant Features". In: *Proceedings of the 10th European Conference on Machine Learning*. ECML '98. London, UK, UK: Springer-Verlag, pp. 137–142. ISBN: 3-540-64417-2. URL: http://dl.acm.org/citation.cfm?id=645326.649721.

Joulin, Armand et al. (2016). "Bag of Tricks for Efficient Text Classification". In: *CoRR* abs/1607.01759. arXiv: 1607.01759. URL: http://arxiv.org/abs/1607.01759.

Karpathy, Andrej (2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Kim, Yoon (2014). "Convolutional Neural Networks for Sentence Classification". In: *CoRR* abs/1408.5882. URL: http://arxiv.org/abs/1408.5882.

Kiros, Ryan et al. (2015). "Skip-Thought Vectors". In: *CoRR* abs/1506.06726. arXiv: 1506.06726. URL: http://arxiv.org/abs/1506.06726.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., pp. 1097–1105. URL: http://dl.acm.org/citation.cfm?id=2999134.2999257.

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

Liu, Bing (2010). "Sentiment Analysis and Subjectivity". In: *Handbook of Natural Language Processing, Second Edition*. Ed. by Nitin Indurkhya and Fred J. Damerau. ISBN 978-1420085921. Boca Raton, FL: CRC Press, Taylor and Francis Group.

McAuley, Julian J., Rahul Pandey, and Jure Leskovec (2015). "Inferring Networks of Substitutable and Complementary Products". In: *CoRR* abs/1506.08839. arXiv: 1506.08839. URL: http://arxiv.org/abs/1506.08839.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133.

Mikolov, Tomas, Kai Chen, et al. (2013). "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781. arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781.

Mikolov, Tomas, Ilya Sutskever, et al. (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *CoRR* abs/1310.4546. arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

Mikolov, Tomas, Scott Wen-tau Yih, and Geoffrey Zweig (May 2013). "Linguistic Regularities in Continuous Space Word Representations". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics. URL: https://www.microsoft.com/en-us/research/publication/linguistic-regularities-in-continuous-space-word-representations/.

Mitchell, Tom M (1997). *Machine learning. WCB*.

Mou, Lili et al. (2016). "How Transferable are Neural Networks in NLP Applications?" In: *CoRR* abs/1603.06111. arXiv: 1603.06111. URL: http://arxiv.org/abs/1603.06111.

Olah, Christopher (2015). *Understanding LSTM Networks*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs.

Oquab, Maxime et al. (2014). "Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks". In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '14. Washington, DC, USA: IEEE Computer Society, pp. 1717–1724. ISBN: 978-1-4799-5118-5. DOI: 10.1109/CVPR.2014.222. URL: http://dx.doi.org/10.1109/CVPR.2014.222.

Pan, Sinno Jialin and Qiang Yang (Oct. 2010). "A Survey on Transfer Learning". In: *IEEE Trans. on Knowl. and Data Eng.* 22.10, pp. 1345–1359. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191. URL: http://dx.doi.org/10.1109/TKDE.2009.191.

Pang, Bo and Lillian Lee (Jan. 2008a). "Opinion Mining and Sentiment Analysis". In: *Found. Trends Inf. Retr.* 2.1-2, pp. 1–135. ISSN: 1554-0669. DOI: 10.1561/1500000011. URL: http://dx.doi.org/10.1561/1500000011.

— (2008b). "Opinion mining and sentiment analysis". In: *Foundations and Trends in Information Retrieval* 2.1-2, pp. 1–135.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "Glove: Global vectors for word representation". In: *In EMNLP*.

Peters, Matthew E. et al. (2018). "Deep contextualized word representations". In: *Proc. of NAACL*.

Radford, A. (2018). “Improving Language Understanding by Generative Pre-Training”. In:

Radford, A., R. Jozefowicz, and I. Sutskever (Apr. 2017). “Learning to Generate Reviews and Discovering Sentiment”. In: *ArXiv e-prints*. arXiv: 1704.01444 [cs.LG].

Radford, Alec et al. (2019). “Language Models are Unsupervised Multitask Learners”. In:

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1988). “Neurocomputing: Foundations of Research”. In: ed. by James A. Anderson and Edward Rosenfeld. Cambridge, MA, USA: MIT Press. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6. URL: http://dl.acm.org/citation.cfm?id=65669.104451.

Semwal, Tushar et al. (2018). “A Practitioners’ Guide to Transfer Learning for Text Classification using Convolutional Neural Networks”. In: *CoRR* abs/1801.06480. arXiv: 1801.06480. URL: http://arxiv.org/abs/1801.06480.

Socher, Richard et al. (2013). *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). “Sequence to Sequence Learning with Neural Networks”. In: *CoRR* abs/1409.3215. arXiv: 1409.3215. URL: http://arxiv.org/abs/1409.3215.

Tang, Duyu, Bing Qin, and Ting Liu (2015). “Document Modeling with Gated Recurrent Neural Network for Sentiment Classification.” In: *EMNLP*. Ed. by Lluís Màrquez et al. The Association for Computational Linguistics, pp. 1422–1432. ISBN: 978-1-941643-32-7. URL: http://dblp.uni-trier.de/db/conf/emnlp/emnlp2015.html#TangQL15.

Turing, Alan Mathison (1936). “On computable numbers, with an application to the Entscheidungsproblem”. In: *J. of Math* 58, pp. 345–363.

Vaswani, Ashish et al. (2017). “Attention Is All You Need”. In: *CoRR* abs/1706.03762. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

Wang, Sida and Christopher D. Manning (2012). “Baselines and Bigrams: Simple, Good Sentiment and Topic Classification”. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*. ACL ’12. Jeju Island, Korea: Association for Computational Linguistics, pp. 90–94. URL: http://dl.acm.org/citation.cfm?id=2390665.2390688.

Williams, Adina, Nikita Nangia, and Samuel R. Bowman (2017). "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *CoRR* abs/1704.05426. arXiv: 1704.05426. URL: http://arxiv.org/abs/1704.05426.

Wolf, Thomas et al. (2020). *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. arXiv: 1910.03771 [cs.CL].

Xiao, Han (2018). *bert-as-service*. https://github.com/hanxiao/bert-as-service.

Xiao, Yijun and Kyunghyun Cho (2016). "Efficient Character-level Document Classification by Combining Convolution and Recurrent Layers". In: *CoRR* abs/1602.00367. arXiv: 1602.00367. URL: http://arxiv.org/abs/1602.00367.

Yin, Wenpeng, Jamaal Hay, and Dan Roth (2019). *Benchmarking Zero-shot Text Classification: Datasets, Evaluation and Entailment Approach*. arXiv: 1909.00161 [cs.CL].

Yosinski, Jason et al. (2014). "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792. arXiv: 1411.1792. URL: http://arxiv.org/abs/1411.1792.

Zhang, Xiang and Yann LeCun (2015). "Text Understanding from Scratch". In: *CoRR* abs/1502.01710. arXiv: 1502.01710. URL: http://arxiv.org/abs/1502.01710.

Zhang, Xiang, Junbo Jake Zhao, and Yann LeCun (2015). "Character-level Convolutional Networks for Text Classification". In: *CoRR* abs/1509.01626. URL: http://arxiv.org/abs/1509.01626.