Junior Management Science 4(1) (2019) 35-62

Junior Management Science

journal homepage: www.jums.academy

Word embedding, neural networks and text classification: what is the state-of-the-art?

Estevan Vilar

ESCP Europe

Abstract

In this bachelor thesis, I first introduce the machine learning methodology of text classification with the goal to describe the functioning of neural networks. Then, I identify and discuss the current development of Convolutional Neural Networks and Recurrent Neural Networks from a text classification perspective and compare both models. Furthermore, I introduce different techniques used to translate textual information in a language comprehensible by the computer, which ultimately serve as inputs for the models previously discussed. From there, I propose a method for the models to cope with words absent from a training corpus. This first part has also the goal to facilitate the access to the machine learning world to a broader audience than computer science students and experts.

To test the proposal, I implement and compare two state-of-the-art models and eight different word representations using pre-trained vectors on a dataset given by LogMeIn and on a common benchmark. I find that, with my configuration, Convolutional Neural Networks are easier to train and are also yielding better results. Nevertheless, I highlight that models that combine both architectures can potentially have a better performance, but need more work on identifying appropriate hyperparameters for training. Finally, I find that the efficacy of word embedding methods depends not only on the dataset but also on the model used to tackle the subsequent task. In my context, they can boost performance by up to 10.2% compared to a random initialization. However, further investigations are necessary to evaluate the value of my proposal with a corpus that contains a greater ratio of unknown relevant words.

Keywords: neural networks; machine learning; word embedding; text classification; business analytics

1. Introduction

> "Innovation is hard. It really is. Because most people don't get it. Remember, the automobile, the airplane, the telephone, these were all considered toys at their introduction because they had no constituency. They were too new." Nolan Kay Bushnell

1.1. Data availability

Data has been called by The Economist the "new oil" (Economist, 2017) as they are now "abundant, ubiquitous and far more valuable [than before]". Internet, social media, sensors, and smartphones have all contributed to the production of electronic information whether structured or not. Daily, 2.5 quintillion bytes of data are created (IBM, 2018). With this increasing amount of data, a need to accurately extract, integrate and classify these resources has appeared in the last two decades.

Among this electronic information, a plethora of textual resources such as tweets, reviews, comments, emails or news but also scanned documents or handwritten notes are produced, and therefore techniques in the field of Natural Language Processing (NLP) and machine learning have been developed to get meaningful knowledge from this information.

The first goal of this bachelor thesis in collaboration with the company LogMeIn Inc.[1] is to evaluate the current state-of-the-art of classification techniques with neural networks, select the appropriate algorithms and subsequently tackle the automated classification and performance analysis. These tasks will be performed to pinpoint the most effective method to sort textual reviews of customers about the use of GoToMeeting[2] - an online meeting, desktop sharing and video conferencing software - by subject (audio, non-audio).

Second, this work is also exploratory as a new method to deal with out-of-vocabulary words is tested and compared

[1] https://www.logmein.com/
[2] https://www.gotomeeting.com/

DOI: https://doi.org/10.5282/jums/v4i1pp35-62

with the state-of-the-art. The goal is to improve the generalization power of classification methods, without deep and heavy implementations.

1.2. Feedback loops

Part of the agile methodology, experimentation is favoured over elaborate planning and so is customer feedback over intuition (Rahimian and Ramsin, 2008). As a consequence, one component of the methodology is to enter quickly what is commonly called feedback loops. It consists of building a minimum viable product, getting customers' feedback and used it to improve the product. In that context, many online tools have been developed to conduct surveys, but also many applications such as AirBnB[3] or Uber[4] include reviews as part of their product to gain the trust of their users. If these tools allow developers and managers to collect a significant amount of data, there is, however, a need to efficiently analyse these data to perform qualitative analysis and infer where resources should be allocated. The third goal of this thesis is, therefore, to present tools that managers or entrepreneurs can leverage to build better products faster. As a consequence, this thesis has been written with a goal in mind to facilitate the access to modern tools for analysis, more specifically neural networks, to add a new card in the hands of managers to understand their customer concerns better.

In Section 2, I introduce the theoretical background and different concepts necessary to understand the functioning of neural networks. I describe two commonly used architectures namely Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) and compare their performance when it comes to classification tasks. In Section 3, I discuss the conversion of textual information in a format recognisable by computers. I introduce three techniques to extract information from texts: GloVe, Word2Vec, and FastText. I also propose a method to deal with words that are not present in the training data. In Section 4, I describe the benchmark to compare the models introduced in Section 2 and techniques mentioned in Section 3. Section 5 includes the results and discussions following the experiment and Section 6 is the concluding part of this thesis.

2. Text Classification and Machine Learning

> "Science is the systematic classification of experience" George Henry Lewes

Text Classification (TC) (also called text categorisation or topic spotting) refers to the identification and labelling of themes or topics of a sentence or document (Sebastiani, 2002). An example would be to label a comment based on the topic it covers like "audio", "screen" and "video". In the early 90's, the emergence of digital data, and the growing computational power of machines contributed to the development of the field. Also, the broad applicability of the task in activities such as spam detection, metadata generation or organisation of documents attracted the interest of technological companies. Before that time, techniques involved knowledge engineering (KE) which consists of classifying a textual document based on knowledge encoded in a set of rules manually defined (Faraz, 2015). However, in the 90's the machine learning (ML) paradigm shifted the attention of researchers away from KE. Rather than imposing classification rules to machines, researchers started to build solutions that let the computer deduce the attributes that will lead to efficient classification. From a pre-classified set of documents, the machine would thus learn the characteristics of interests to build an automated classifier.

Formally, TC tasks assign a Boolean value to each pair $\langle d_j, c_j \rangle \in D \times C$, with D being a training set of documents and $C = \{c_1 ..., c_n\}$ a set of predefined categories. The goal is to approximate the target function f: $D \times C \rightarrow [T, F]$ where T indicates that d_j must be classified under c_i whereas F indicates that d_j must not be classified under c_i. As f is unknown, the function g: $D \times C \rightarrow [T, F]$ that approximate f - also called classifier (or model, or rule) - is used. Then, the effectiveness of the classifier - or accuracy - refers to the degree to which f and g coincide. Ultimately, classifying a document D under $C = [c_1, ..., c_i, ..., c_n]$ with i=1...,n can be seen as n independent problems with $f_i : D \rightarrow [T, F]$ as an unknown target function for c_i and $g_i : D \rightarrow [T, F]$ a classifier for c_i (Sebastiani, 2002).

The first challenge lies in the so-called inter-indexed inconsistency based on the first law of Jesse H. Shera (Cleverdon, 1984). It states that "No cataloguer will accept the work of any other cataloguer". This law highlights the subjectivism of classification tasks and therefore points to the non-existence of a deterministic solution - a function f - for the classification problem. Nevertheless, in the last decades, researchers have been looking for an optimal function g to solve specific classification problems.

In ML, building a classifier relies on the availability of a pre-classified corpus from which to deduce the relevant characteristics i.e., a corpus on which the values of every pair $\langle d_j, c_j \rangle \in D \times C$ are known. Besides, to evaluate the effectiveness of the classifier, it is common practice to split this pre-classified corpus between a training- set - used to build the classifier - and a test set - to assess the effectiveness of the classifier. Once the classifier is built, each d_j from the test set are used as input which produces a corresponding c_i. The effectiveness is measured by how often the pairs $\langle d_j, c_i \rangle$ matches the values of the pre-classified corpus while testing.

Since the beginning of machine learning techniques for TC, a broad range of model including rule induction, naïve-bays, decisions trees, K-nearest neighbours (KNN), support vector machines (SVM) and neural networks have been used to build classifiers. A comparative study of the techniques is available in (Kaur and Kaur, 2017; Khan et al., 2010; Nikam, 2015). As pointed out in (Young et al., 2018), deep learning architecture such as deep neural networks have increasingly

[3] https://www.airbnb.com/
[4] https://www.uber.com/en-MX/

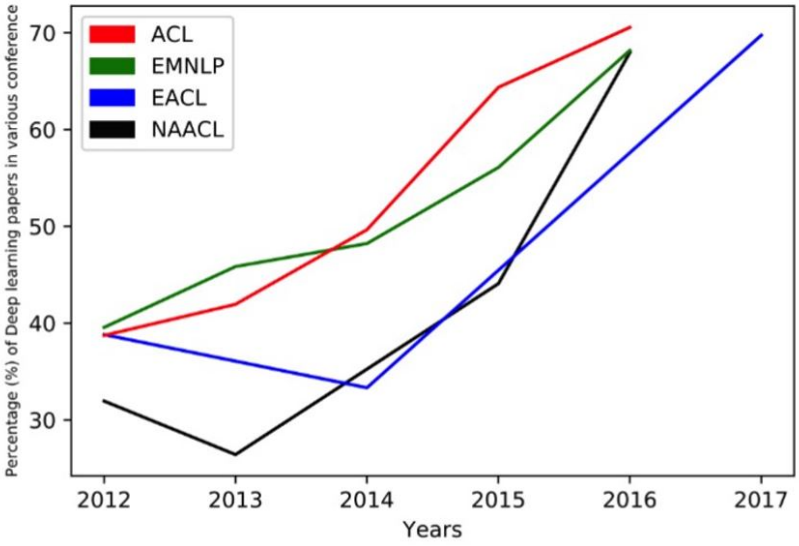

Figure 1: Percentage of deep learning papers in ACL, EMNLP, EACL, NAACL over the last six years; Source: (Young et al., 2018)

attracted the attention of researchers as shown in Figure 1. For that reason; this work is focusing on neural networks for TC tasks.

2.1. Neural Networks for text classification

Artificial neural networks as defined by Dr. Robert Hecht-Nielsen quoted in Neural Network Primer: Part 1 is:

> "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." (Caudill, 1986).

If the essential components of neural networks remain the same, their architecture can change a lot. For this section, I aim to identify the state-of-the-art model for TC among Convolutional Neural Networks (CNN) and Recurrent Neural Network (RNN). First, I describe the functioning of a multilayer perceptron (MLP), a feed-forward neural network, which represents one of the most straightforward architectures of neural networks[5]. It is done to introduce the fundamental concepts necessary to understand the CNN and RNN. I describe the models and their most up-to-date applications for TC tasks.

2.2. Feed-forward neural networks

The definition mentioned previously encompasses the essential components of modern neural networks. Hecht-Nielsen refers to what are today called neurons with the word "processing elements". This computational unit receives a set of scalar x_i or vector x as input, (1) multiplies them by their importance - their weights w_i-, (2) and apply a function f such as summation or max operation. Finally, (3) it applies a non-linear function g - also called activation function - on the result, which represents the output - a single scalar y or vector y as shown in Figure 2.

Artificial neural networks are made out of a multitude of neurons that are interconnected in different layers as illustrated in Figure 3.

They have the power to approximate any Borel functions from a finite dimensional space to another as shown in (Hornik et al., 1989), a category under which classifiers defined in the previous paragraph fall.

In mathematical notations, the feed-forward neural network represented in Figure 3 with two hidden layers would be expressed as follow[6]:

$$NN_2(x) = g^2(g^1(xW^1)W^2)W^3$$

with $x \in R^{input}$ an input vector (dimension of the Figure is 3), $W^1 \in R^{input} \times R^{output}$ is the weight matrix from the input to the first hidden layer, $W^2 \in R^{input} \times R^{output}$ is the weight matrix from the first hidden layer to the second hidden layer $W^3 \in R^{input} \times R^{output}$ is the weight matrix from the second layer to the output layer, $g^1()$ is the activation function in the first layer and $g^2()$ is the activation function of the second layer. In Figure 3, W^1, W^2, and W^3 are of dimension 3x3, 3x2 and 2x4 respectively.

[5]The simplest one is a single layer perceptron

[6]Some feed-forward neural networks include a bias term in some layer, which is a neuron that is not connected with the previous layer. Figure 3 does not have any bias and therefore the bias in not included in the mathematical notation.

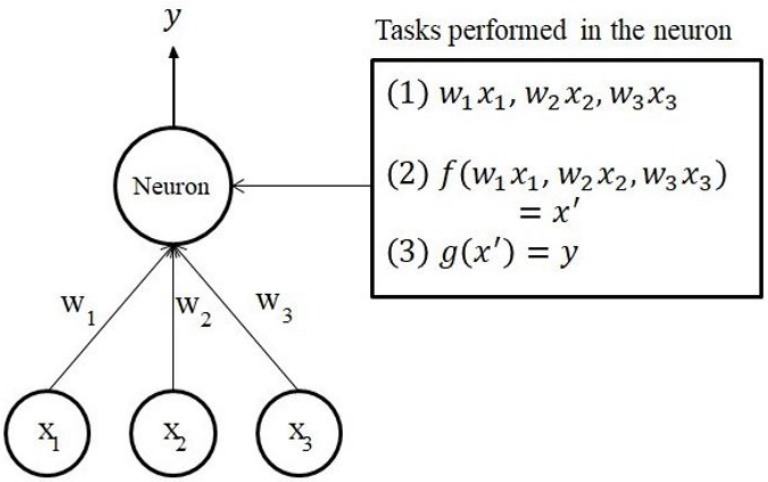

Figure 2: Illustration of the tasks performed in a neuron; Source: Author's own representation

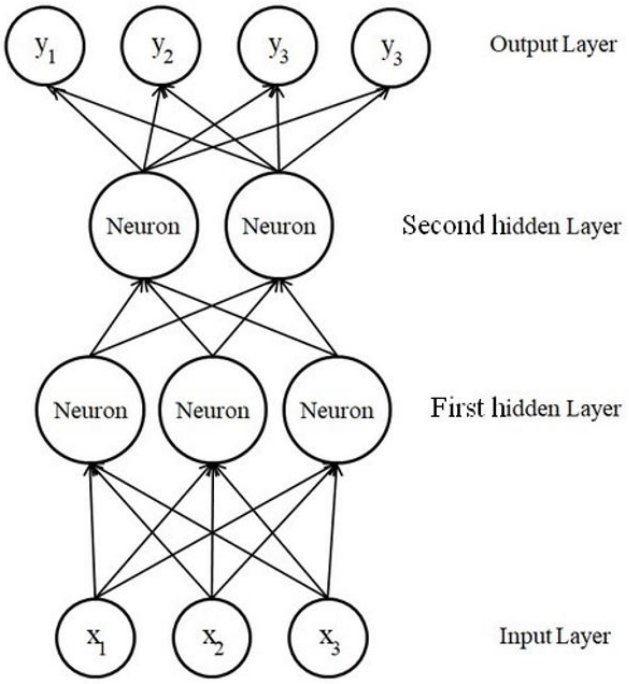

Figure 3: Feed-forward neural network with two hidden layers; Source: Author's own representation

Alternatively, the hidden layers could be expressed as:

$$h^1 = g^1(xW^1) \text{ for the first layer}$$
$$h^2 = g^2(h^1W^2) \text{ for the second layer}$$

This gives us:

$$NN_2(x) = h^2W^3$$

The collection of matrices W^1,W^2,W^3 is referred in the literature as the parameters θ of the neural network. In classification problems, feed-forward neural networks are often designed such as each element in the output layer is positive and that they sum to 1. The output vector can, therefore, be interpreted as a probability distribution over the different classes $[c_1..., c_n]$. This final transformation is often performed with a softmax function[7].

[7] $softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$ for $x = x_1...x_k$

Table 1: Summary of the most commonly used activation functions and their first derivative; Source: Compiled by author

Name	f(x)	f'(x)
Sigmoid	$\frac{1}{1+exp(-x)}$	$sigmoid(x)(1-sigmoid(x))$
Tanh	$\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$\frac{d}{dx}tanh(x) = 1-tanh(x)^2$
ReLu	$max(0,x)$	$\begin{cases} 0, & x<0 \\ 1, & x \geq 0 \end{cases}$
ELU	$\begin{cases} \alpha(exp(x)-1), & x<0 \\ x, & x \geq 0 \end{cases}$	$\begin{cases} \alpha(exp(x)-1), & x<0 \\ 1, & x \geq 0 \end{cases}$
Swish	$x * sigmoid(\beta x)$	$\beta swich(x) + sigmoid(\beta x)(1-\beta(swich(x))$

2.2.1. Input layer

The input of the neural network is usually a vector $x = (x_1 ... x_k)$. For TC problems, this vector is the result of a transformation of textual data to a vector representation. It is often referred as an embedding layer. I discuss the vector representation of text in paragraph 3.1.

2.2.2. Activation functions

In the machine learning literature, many activation functions including sigmoid, Rectified Linear Units (ReLu) (Hahnloser et al., 2000), Exponential Linear Unit (ELU)(Clevert et al., 2015) and tanh have been considered yielding different results. In its paper, Alcantara (2017) provides a comparison of different activation functions and concludes that ELU performs the best with ReLu nevertheless yielding great results. However, in the recent work of Ramachandran et al., 2017, the authors claimed that no other function had been more adopted than ReLu thanks to its simplicity and effectiveness. It is also concluded that Swish, a function similar to the Sigmoid-weighted Linear Unit (SiL) (Elfwing et al., 2018) performed better than ReLu. As far as I know, no comparison between Swish and ELU has been made and it is still an open-question to determine which one performs best. A summary of common activation functions is available in Table 1. Also, plots of these functions are available in Figure 4 and Figure 5.

2.2.3. Training a neural network

Neural networks must be trained to be efficient. Training a neural network involves setting the right weights in the various matrices W: in another word, tuning the parameters θ the best possible so the neural network approximates the desired function. To do so, a loss function is optimised and various techniques to perform the task exist.

Loss functions

As mentioned in paragraph 2.1, it is common practice in machine learning to split the data into a training set and a validation one. Let us define the output of the neural network as $\hat{c}$ and the actual output as c. In the training set, all the pairs $\langle d_j, c_j \rangle \in D \times C$ - each document and their corresponding class c_i - are known. The objective of the training is to minimise the function $L(\hat{c}, c)$ - a loss function - that gives a score to $\hat{c}$ based on c. The score is therefore null if $\hat{c}_l = c_i$ and positive otherwise.

Recently, a comparison between several loss functions has been performed for TC purpose (Janocha and Czarnecki, 2017). Out of 12 loss functions - showed in Table 2 - the authors conclude that non-log losses are preferable for classification purpose. In particular, they identify the squared hinge loss (formula present in Table 1) to be the best performing function. They note however that if much noise[8] is present in the data set, the expectation loss is the preferable choice.

Training techniques

The loss function is what needs to be minimised, and the computer must be told how to do it i.e., defining a training algorithm for the neural network.

As pointed out in (Goodfellow et al., 2016, Chapter 8), the training of the parameters θ is indirect as we hope by minimising $L(\hat{c}, c)$ we will obtain the best parameters. Therefore the techniques differ from classic optimisation problems. This include for example not evaluating the loss on the whole data set but rather on small batches and then average the results for computation power purpose. Indeed, as the standard error of the mean from a sample n is $\frac{\sigma}{\sqrt{n}}$ where σ is the true standard error, training a set of 10'000 examples takes 100 times more computational power than training a set of 100 examples, but reduces the error only by a factor 10 (Goodfellow et al., 2016, Chapter 8). Using less than all the training examples available is referred as mini-batch methods.

The most used category of optimisation algorithm are named back propagation or backward propagation of errors (Rumelhart et al., 1988) and its best representative is currently the stochastic gradient descent (SGD) (Goodfellow et al., 2016; Ruder, 2016). It consists of an iterative approach that reduces $L(\hat{c}, c)$ by moving the parameters θ in the direction opposite to sign of $L'(\hat{c}, c)$ - the derivative of the loss function. The algorithm is shown in Figure 6.

The learning rate ϵ_k present in Figure **??** as a required output is a parameter that defines how quickly the old pa-

[8] i.e. a big variability

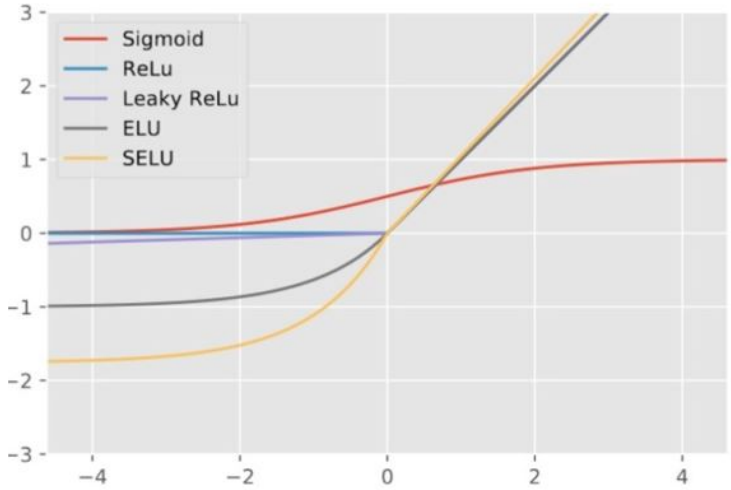

Figure 4: Plots of activation functions including Sigmoid, ELU, ReLU; Source: (Alcantara, 2017)

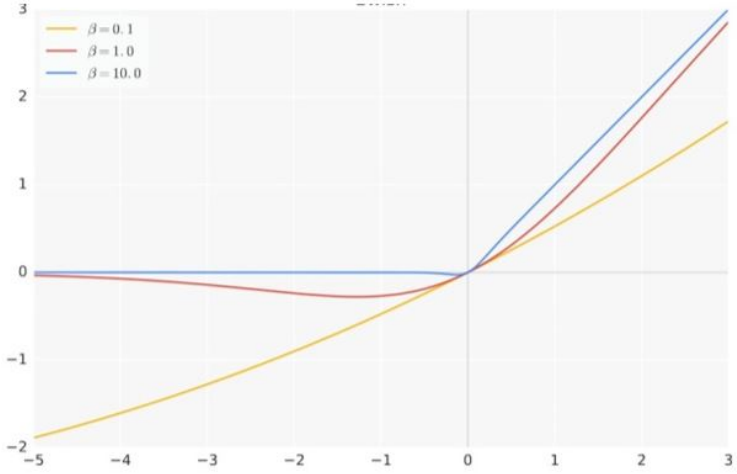

Figure 5: Plot of the Swich function with different betas; Source: (Ramachandran et al., 2017)

Algorithm Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ with corresponding targets $\boldsymbol{c}_i$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(g(\boldsymbol{x}_i; \boldsymbol{\theta}), \boldsymbol{c}_i)$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

Figure 6: Algorithm of SGD. Note that the function $g(x_i; \theta) = \hat{c}_i$ is the one referred to in 2; Source: (Goodfellow et al., 2016, Chapter 8)

rameters are forgotten compared to the new one. It has been demonstrated that, if the learning rate is appropriately set, using SGD, the function will surely converge to a global minimum or local minimum, if the function is convex (such as the one presented in 2.2.3.1) (Bottou, 1998; Kiwiel, 2001). Furthermore, Bottou, 2012 suggest to update the learning rate in function of the iteration - also called epoch - as follow:

Table 2: List of loss functions tested in (Janocha and Czarnecki, 2017). The authors name "y" the true value. I use the notation c. Similarly, the output of the neural network is named "o" whereas I name it $\hat{c}$; Source: (Janocha and Czarnecki, 2017)

symbol	name	equation
Λ_1	L_1 loss	$\| y-o \|_1$
Λ_2	L_2 loss	$\| y-o \|_2^2$
$\Lambda_1 \circ \sigma$	expectation loss	$\| y-\sigma(o) \|_1$
$\Lambda_2 \circ \sigma$	regularised expectation loss	$\| y-\sigma(o) \|_2^2$
$\Lambda_\infty \circ \sigma$	Chebyshev loss	$max_j \lvert\sigma(o)^{(j)} - y^{(j)}\rvert$
$hinge$	hinge (margin) loss	$\sum_j max(0, \frac{1}{2} - \hat{y}^{(j)} o^{(j)})$
$hinge^2$	squared hinge (margin) loss	$\sum_j max(0, \frac{1}{2} - \hat{y}^{(j)} o^{(j)})^2$
$hinge^3$	cubed hinge (margin) loss	$\sum_j max(0, \frac{1}{2} - \hat{y}^{(j)} o^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j y^{(j)} log \sigma(o)^{(j)}$
log^2	squared log loss	$-\sum_j [y^{(j)} log \sigma(o)^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(o)^{(j)} y^{(j)}}{\|\sigma(o)\|_2^2 + \|y\|_2^2 - \sum_j \sigma(o)^{(j)} y^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence	$-log \frac{\sum_j \sigma(o)^{(j)} y^{(j)}}{\|\sigma(o)\|_2 \|y\|_2}$

$$\epsilon_k = \epsilon_0 \frac{1}{1 + \epsilon_0 \delta k}$$

With ϵ_0 the initial learning rate and δ a hyperparameter[9] to be set. However, as pointed out in (Zeiler, 2012), setting the hyperparameters alter the results of the neural networks, and the tuning can be tricky. He, therefore, presents an improvement of the standard SGD, ADAELTA, that, when used, the performance of the neural networks is not sensitive on the hyperparameter of the learning rate. The algorithm is shown in Figure 7.

Moreover, similar algorithms to ADADELTA exist such as ADAM (Kingma and Ba, 2015) or Nadam (Dozat, 2016). Finally, it is worth to point out that Ranganathan and Natarajan (2018) recently developed a new method of backpropagation without using SGD but rather Moore-Penrose Pseudo Inverse[10] with promising results.

Initialisation of the network

At the beginning of the training, the weights in the different matrices W must be set. This point can determine whether the loss function - regardless of its form - will converge or diverge. Therefore, two underlying questions emerge from this issue: what is the ideal magnitude of the initial weights and what is the range in which they must be included? Before 2006, deep neural networks tended to produce inaccurate results and one reason for that is that initialisation of the network was usually totally random (Erhan et al., 2009; Glorot and Bengio, 2010; Sutskever et al., 2013). This resulted in errors such as vanishing (converging close to 0) or exploding (becoming high) gradients which does not allow the neural network to approximate the required function. In addition, neurons tended to become saturated - setting output value to 0 due to very small gradients. Likewise, output values could become too high - or die - resulting in a gradient of 0 due to inputs being negative caused by a big negative change in the gradient during the previous iteration. These issues can be solved with a wise choice of the loss function, learning algorithm, and effective initialisation of the network. An initialisation method - the xavier initialisation - introduced in (Glorot and Bengio, 2010), has become a popular technique among researchers (Goldberg, 2015). It consists of initialising the matrix as follow:

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{d_{in} + d_{out}}}; \frac{\sqrt{6}}{\sqrt{d_{in} + d_{out}}}]$$

With $U[a, b]$ being a uniformly sampled value between a and b, d_in is the dimension of the input vector, and d_{out} is the dimension of the output vector. Using this initialisation makes sure that the distribution of the input is centred around 0 and of variance 1. However, this method assumes that the activation function is linear which is not the case for ReLu for instance. Also, this method seems not to work for very deep models (Glorot and Bengio, 2010). He et al.

[9] In machine learning, the word "hyperparameter" is used to distinguish from the parameters θ. Hyperparameters are higher level parameters set to configure properties of the neural network.

[10] A generalisation of the notion of inverse matrix that satisfies the four Moore-Penrose conditions (Penrose, 1955)

Algorithm Computing ADADELTA update at time t

Require: Decay rate ρ, Constant ϵ
Require: Initial parameter θ
Initialize accumulation variables $E[g^2]_0 = 0$, $E[\Delta\theta^2]_0 = 0$
for $t = 1 : T$ **do** %% Loop over # of updates
Compute Gradient: g_t
Accumulate Gradient: $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2$
Compute Update: $\Delta x_t = -\frac{\text{RMS}[\Delta\theta]_{t-1}}{\text{RMS}[g]_t}\, g_t$
Accumulate Updates: $E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1-\rho)\Delta\theta_t^2$
Apply Update: $\theta_{t+1} = \theta_t + \Delta\theta_t$
end for

Figure 7: Algorithm of ADAELTA. Note that $RMS[x]_t = \sqrt{E[x]_t + \epsilon}$ as in (Becker et al., 1988). The hyperparameters ρ and ϵ do not alter the performance of the model significantly; Source: (Zeiler, 2012).

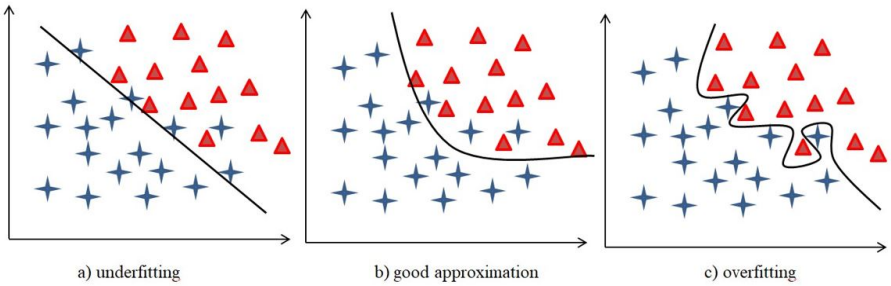

Figure 8: Illustration of 1) underfitting, 2) a good approximation and 3) overfitting; Source: Author's own representation

(2015) offer, therefore, to solve these two issues by doing an initialisation as follow:

$$W \sim U(0; \sqrt{\frac{2}{d_{in}}})$$

With $N(a, b)$ being a normal distribution of mean a and standard deviation b.

Generalisation challenges and regularisation

As defined at the beginning of this section, the accuracy of the classification algorithm is how often the couple $\langle d_j, c_i \rangle$ matches the values of the pre-classified corpus. Also, it was mentioned that the data set is usually split between a training set and a test set. When training the algorithm, we therefore obtain a training error - the proportion of examples for which the model produces an incorrect output. Similarly, we obtain a test error, when running the model on the test set. One challenge in training a model is to avoid a training error that is too high - problem named underfitting - which is the result of a high bias. It produces a model that is too general and not capable of proper predictions with unknown inputs. A second challenge is to have a gap between the training error and test error to be too wide - which is called overfitting - which makes the model to be too specific to the training set and thus not generalizable for new data. Both problems are illustrated in Figure 8 with an analogy to regressions. It must be pointed out that no classification algorithm exists that outperforms other on all possible data distribution. It is known as the no free lunch theorem (Wolpert, 1996) which is a generalisation of the inter-indexed inconsistency mentioned earlier in this section. Nevertheless, we can find algorithms that perform well on a specific distribution. As expressed in paragraph 2.1, neural networks are particularly capable of approximating any Borel functions. However, it makes them also particularly prone to overfitting. To minimise it, one could get more and better data or regularize the model.

Regularization " is any modification we make to a learning algorithm that is intended to reduce its generalization error, but not its training error" (Goodfellow et al., 2016, Chapter 5). Regularization is a widely researched topic in machine learning but the most common forms of regularization are weight penalties, early stopping, and dropout.

Weight penalties

Weight penalties consist of adding an element to the loss function $L(\hat{c}, c)$ depending on the magnitude of the weights in the matrix W and a hyperparameter γ controlling for the amount of penalty. Two common weight penalties used are called L1 - also called Lasso regression (Tibshirani, 1996) - and L2 regularisation - also called Tikhonov regularisation or ridge regression (Ng, 2004). Let's define a new loss function $L^*(\hat{c}, c)$, below the equations of L1 and L2:

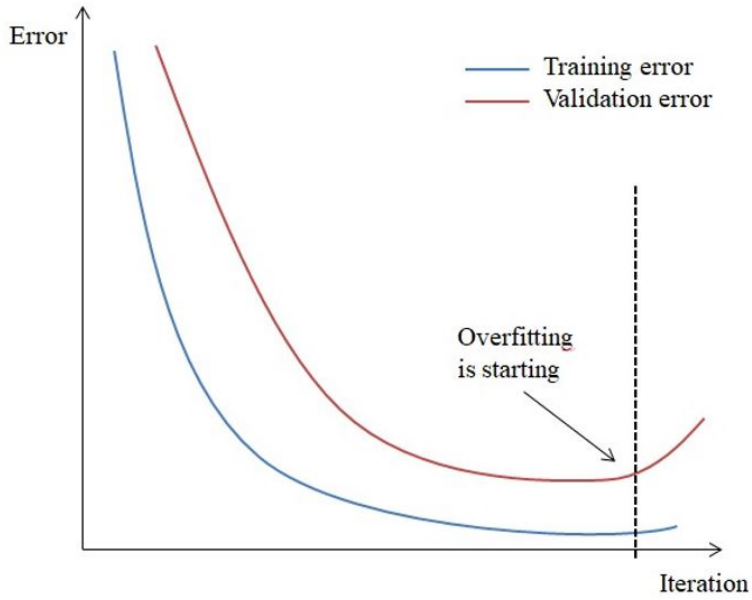

Figure 9: Illustration of overfitting and when the training should stop; Source: Author's own representation

$$L1 : L^*(\hat{c}, c) = L(\hat{c}, c) + \gamma \sum_{w \in W} |w|$$
$$L2 : L^*(\hat{c}, c) = L(\hat{c}, c) + \frac{\gamma}{2} \sum_{w \in W} w^T W$$

Both methods tend to penalise large values in W by shrinking them towards 0, however, in L2 values are squared due to the matrix multiplication and are therefore more penalised. In machine learning literature, L1 appeared first, but L2 has been outperforming L1 in most cases (Ng, 2004).

Early stopping

Early stopping merely consists of stopping the training session before the model starts to learn too much specificity on the training set. This is achieved by stopping when the validation error starts to become greater than for the previous epoch. Indeed, that would mean that the gap between the validation error and the training error is widening and therefore the model starts to become too specific to the training set as shown in Figure 9.

Dropout

Dropout is a method introduced in (Srivastava et al., 2014) that consists of temporarily removing random neurons of the network as shown in Figure 10. A neuron has a probability p of being removed and the authors suggest starting with a value of 0.5 and then adjust if necessary. The rationale behind it is inspired from the role of sex in evolution (Livnat et al., 2010): sexual reproduction generally involves taking half the genes of the male and half of the women ones forcing the genes to "work" together. Similarly, by dropping out neurons from the network, they are obliged to work with randomly selected neurons. It means that a neuron will not overly rely on a specific underlying neuron and learn to adapt from different inputs, which is the end goal of regularisation. Empirical studies have suggested that dropout is a very effective method of regularisation, in particular with the ReLu activation function (Dahl et al., 2013; Warde-Farley et al., 2013).

2.3. Convolutional Neural Network

For TC tasks, the input of the neural networks is often a sentence or a set of phrases. These have to be encoded in a vector representation (discussion about it in Section 3). This could easily be achieved by considering the sentence as a bag-of-words. However, this method does not take into account the word order. Yet the meaning of a sentence is highly dependent on the word order. CNNs are designed to take into account the context around each word and therefore avoid to consider the input as a bag-of-words. They have been first used in image recognition and then introduced to the NLP community with the work of Collobert et al. (2011) and then showed excellent results even with shallow architecture (Kalchbrenner et al., 2014; Kim, 2014). Since then, CNNs have been continuously used for TC tasks representing the state-of-the-art of text classification techniques (Agrawal and Awekar, 2018; Georgakopoulos et al., 2018; Le et al., 2018; Salinca, 2017; Sundström, 2017). Zhang et al. (2015) developed a similar model to Kim's working at a character-level rather than word level with results varying from a data set to another. Finally, in (Johnson and Zhang, 2017) a deep pyramid CNN model with 15 weight layers was developed. To avoid extravagant computing costs, they decrease the computation time allowed to perform the task in function of the layer depth (from which the pyramid reference comes from). So far this architecture has been the best performing one on several TC tasks.

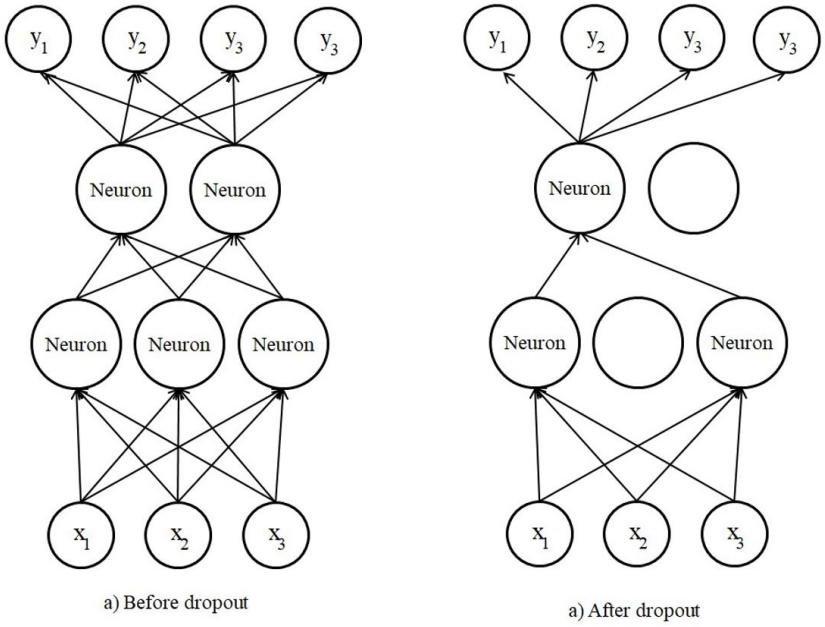

Figure 10: a) represents a two-layer neural network. b) is the same network with a dropout deactivating two neurons; Source: Author's own representation

The architecture of CNN is similar to the MLP model introduced in section 2.2. The difference lies in the addition of a convolutional layer and a pooling layer represented in Figure 11.

2.4. Convolutional layer

The convolutional layer is present to extract from the input the most salient information - also called feature (more discussion about it in paragraph 3.3) - around a particular window of h words referred as the filter[11] in (Kim, 2014). For a filter of size 2 and the sentence "we unlock the potential of the modern workforce"[12], the convolutional layer extracts the features from "we unlock", then "unlock the", then "the potential" and so forth. Similarly, a window of size 3 on the same sentence extracts features in "we unlock the", "unlock the potential", "the potential of" etc. For each filter, a feature map is created that, from each extraction, stores the different features.

Formally, the layer receives an input vector $s \in R^s$ constructed from a sentence for instance. A dot product is performed between a vector of weights $w \in R^w$ and each w-gram[13] in s resulting in a new set of features $e = [e_1 ... e_n]$. The value of n will change depending on the dimension of s and w. If s≥w then n=s-m+1 (narrow convolution), else $n = s + m - 1$ (wide convolution) with all the $e_i = 0$ for $i > s$.

In the model presented by Kim, a multichannel architecture has been designed - that is a single layer that applies multiple filters with different sizes on the input and stores the features. Kalchbrenner et al. (2014) later added multiple convolutional layers in their model.

2.4.1. Pooling layer

After the convolutional layer, a set of features is stored for each filter in the filter map. The pooling layer will simply extract the most important feature in each filter map with a function such as $max(x)$ called 1-max pooling. This is performed to reduce the size of the output, reducing thus the computation power required. In addition, as it reduces the number of parameters θ (w is included in θ), it reduces the risk of overfitting. Kalchbrenner et al. (2014) replaced the 1-max pooling layer by a dynamic k-max pooling layer in charge of extracting the k most important features from the different feature maps. It is called dynamic as the value of k varies in function of the number of the current layer l, the total number of layers L, the dimension of the input s and k_{last} the number of features that are extracted from the last convolution:

$$k_l = max(k_{last}; \frac{L-l}{L}s)$$

Several methods of pooling exist also using the average or the summation of the features in e, but $max()$ is the most widely used.

2.5. Recurrent Neural Network

While basic feed-forward neural networks are not able to take into account the word order, we have seen that, by adding a convolutional layer to the architecture, they become

[11] The filter has the goal to capture the context

[12] Sentence from www.logmeininc.com/

[13] "we unlock" would be a 2-gram in the sentence "we unlock the potential of the modern workforce". "we unlock the potential" would be a 4-gram.

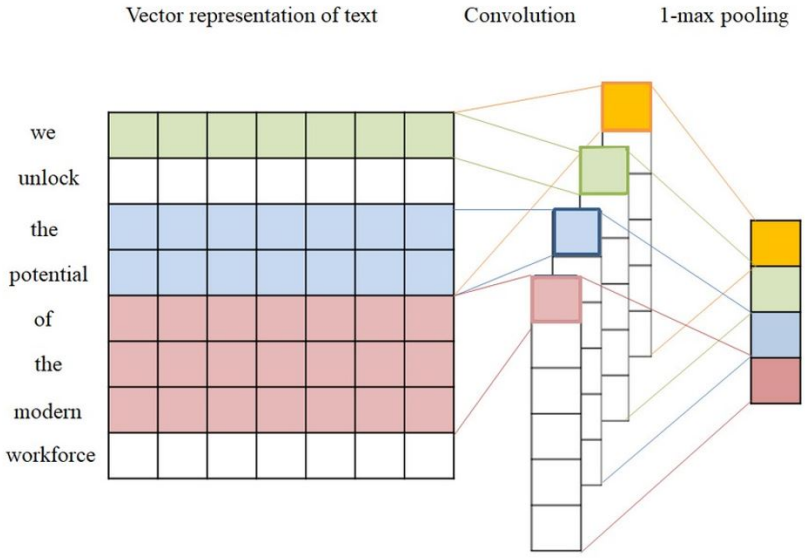

Figure 11: Illustration of a convolution and a 1-max pooling layer. Each word in the sentence "we unlock the potential of the modern workforce" is represented by a vector of dimension 7. In green, a filter of size 1 is applied, in blue a filter of size 2, in red a filter of size 3 and in yellow a filter of size 4. For each filter, the result is a filter map. For each filter map, a 1-max pooling operation is applied. As 4 different filters were used, the output is of size 4 since only one feature is extracted from the 1-max pooling layer; Source: Author's own representation

capable of taking into account the context of a word. However, they are not able to take into account the full context as filter sizes are set as hyperparameters. Also, the size of the input vector has to be fixed and therefore during the preprocessing of the data a padding operation - i.e., setting all the input vectors to the same size - must be performed. It is usually done by setting the vector size as big as the longest input in the data set.

Recurrent Neural Network (Elman, 1990), shown in Figure 12, have been particularly suited to work with textual data (Mikolov et al., 2010; Mikolov et al., 2011) because they allow processing variable-length inputs. They do that by being recurrent as they perform the same task for every element of a sequence. The output is then dependent on the previous computation. Compared to MLPs or CNNs, RNNs have an additional component - a hidden state vector - that memorises the previous information. Using the same sentence "we unlock the potential of the modern workforce", the model first processes the word "we", then the word "unlock" taking into account the computation performed for "we". Then, it processes the word "of" taking into account the computation performed for the word "unlock" which was computed using the computation for the word "we". The algorithm goes one until the end of the sentence. The output includes, therefore, all the computation performed for every single word in the sentence. We understand therefore, why RNNs have first been used for language modelling (Martens, 2011; Mikolov et al., 2010, Mikolov et al., 2010): if the output of the computation is a conditional probability based on the previous words, they can thus predict the next word (Sundermeyer et al., 2014). Similarly, for TC, after each word the conditional probability of the sentence being in a class category is updated until the end of the sentence. The output represents the probability of the whole sentence being classified in a certain category based on all the words in it.

Formally, for an input vector $x = [x_1, ...x_{input}] \in R^{input}$, a scalar v_1 is formed by concatenating the vector representing a word in x, and s_0 is the hidden state at iteration 0. Then for i starting at 1[14]

$$\begin{aligned} v_i = concatenate(x_i) = [x_1; ...; x_{input}] \\ s_i = f(v_i W + s_{i-1} V) \\ y_i = g(S_i V) \end{aligned}$$

Where W and V are weights matrices, fan activation function and g another function that results in a probability distribution $y = [y_1, \dots y_n]$. In their original paper, Mikolov et al. used a sigmoid function for f and a softmax for g. For classification tasks, the intermediate values of y_i are often ignored and only the final one, x_n, is used as it represents the probability of the whole sentence being in a certain class. From the notation above we can observe the recursive nature of the neural network. If we model the hidden state at the 3rd iteration, the equation would be:

$$\begin{aligned} s_3 &= f(v_3 W + s_2 V) \\ &= f(v_3 W + f(v_2 W + s_1 V) V) \\ &= f(v_3 W + f(v_2 W + f(v_1 W + s_0 V) V) V) \end{aligned}$$

[14]Please note that I use the notion $[a, \dots, b]$ to define a set and $[a; \dots; b]$ for the concatenation operation

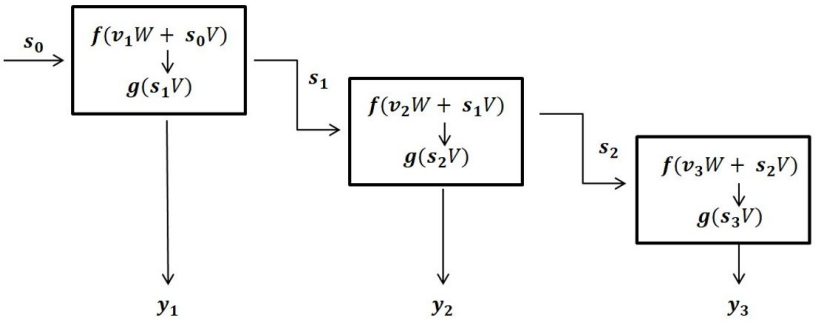

Figure 12: Representation of a Recurrent Neural Network (RNN) with an input vector of dimension 3; Source: Author's own representation

As the process is iterative, we understand how the size of the input can be flexible. Conceptually, the RNN is very similar to an MLP, however, the number of hidden layers is the same as the dimension of the input. Therefore, a layer is "created" for each word that is present in the sentence that we want to classify.

This structure also has some drawbacks. Indeed, due to their recursive nature, RNNs are often difficult to train as they can become very deep neural networks. As a consequence, they often face the problem of vanishing gradient explained in paragraph 2.2.3.4. Also, when it comes to language modelling or classification tasks, sometimes a big gap between relevant information is found. Indeed, it can be that relevant words are at the beginning and the end of the sentence. Therefore, it would be hard for the last iterations to capture the relevance of the first word as it is "drawn" by all the iterations that have been previously performed (Olah, 2017). For these reasons, several improvements in their architecture have been made to tackle classification tasks. The most commonly used are Long Short-Time Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and its variant Gated Recurrent Unit (GRU)(Cho et al., 2014).

2.5.1. Long Short-Time Memory

Simple RNNs introduced the hidden state layer to memorise information. LSTM has an additional variable that tracks the value of the gradients - the memory cell - and three additional layers to monitor and control the memorising of information commonly named input gate, forget gate and output gate. The different gates can be thought as neurons as introduced in paragraph 2.2. In LSTMs, the hidden state layer and the output are the same and therefore $y_i = s_i$.

Conceptually, the memory cell is an object that is going to be updated during the whole iterative process. For each iteration, the memory cell goes through the output gate which gives the final output $y_i = s_i$. How much information comes from the previous iteration that must be forgotten in the memory cell is monitored by the forget gate. Similarly how much of the new information from the current iteration should be added to the memory cell is monitored by the input gate.

Formally, we define $i, fo, o, n \in R^{dim_s}$ vectors referred as input gate, forget gate and output gate and new candidate respectively. Also are defined $c = [c_1, ...c_{2*dim_s}$ the memory cell and c_j the memory components. Then:

$$n_i = tanh(W_n[c_{i-1}, y_{i-1}; x_i]) \quad (1)$$

$$fo_i = \sigma(W_{fo}[c_{i-1}, y_{i-1}; x_i]) \quad (2)$$

$$i_i = \sigma(W_i[c_{i-1}, y_{i-1}; x_i])$$

$$c_i = fo_i * c_{i-1} + n_i * i_i \quad (3)$$

$$o_i = \sigma(W_o[c_i, y_{i-1}; x_i]) \quad (4)$$

$$y_i = tanh(c_i) * o_i \quad (5)$$

First, the new candidate (1) is computed through a tanh() function which represents the new information coming from the new word x_i. The tanh() function is applied to make sure that the values are included in the range $[-1 : 1]$. Then, the forget gate (2) and the input gate (2') are computed simultaneously through a sigmoid function σ. This step, thanks to the sigmoid function, tells that value close to 0 must be forgotten and values close to 1 must be saved. From there, the memory components c_j (3) can be computed from the new candidate, the input gate and the forget gate again with a sigmoid function. The output gate (4) is computed from the new memory component c_j. Finally, the output y_i (5) is computed from the dot product between the tanh of the memory component and the output gate. y_n therefore represents the final probability distribution over the different classification categories.

2.5.2. GRU

The GRU architecture is a simplification of the classic LSTM model but has shown to be competitive for TC tasks (Berger, 2014). Like normal RNNs, GRUs use a hidden state layer but have an update gate and a reset gate.

Mathematically, we define $u, r, n \in R^{dim_s}$, vector referred as update gate, reset gate and new candidate respectively, then:

$$u_i = \sigma(W_u[y_{i-1}; x_i])$$
$$r_i = \sigma(W_r[c_{i-1}, y_{i-1}; x_i])$$
$$n_i = tanh(W_n[y_{i-1} \times r_i; x_i])$$
$$y_i = (1-u_i) \times y_{i-1} + u_i \times n_i$$

Similar to the LSTM architecture, the gates monitor the quantity of new information that should be added at each iteration. The output is simply an interpolation between the previous iteration - controlled by the update gate - and the new iteration-controlled by the reset gate through the computation of the new candidate.

Even if LSTMs, GRUs, and variants are better suited for language modelling, they have been able to compete against CNNs for TC tasks (Ding et al., 2018; Lee and Dernoncourt, 2016; Liu et al., 2016; Zhou et al., 2016a). Recently, Yu et al. (2018) successfully mimicked skimming, re-reading and skipping techniques performed by humans during TC tasks with an LSTM design. They achieved that by adding a cost function that is minimised during the whole process, providing a better accuracy and higher efficiency than previous approaches. Also, Ma et al. (2018) provide an extension of LSTM that has a separate output gate that incorporates the explicit knowledge such as common sense facts for accomplishing a specific task. The architecture achieved promising results.

2.6. Comparison

We have seen that CNNs are efficient machines in extracting local features around words, but weak at deriving features from sequential treatments because of their rigid structure. On the other hand, RNNs are effective at learning features from sequential correlations, but unable to do it in a parallel way (Zhou et al., 2015b). The two methods seem complementary and in (Yin et al., 2017) the authors point out that which architecture performs better depends on "how important it is to understand the whole sequence". Indeed, they found that RNNs are not particularly well suited when critical information has to be identified in a sentence to take a classification decision. It includes identifying a particular word to determine the topic or the sentiment of the sentence. They also note that CNNs and GRUs are comparable when sentences are small (<10), but GRU becomes better when the sentences become longer. Finally, according to Baidu Research DeepBench benchmark[15], CNNs are approximately 5x faster to train than RNNs. The iterative nature of RNNs may explain this result.

As it is not clear which one performs better, Zhou et al., 2015b developed a model combining a convolutional layer and an LSTM one. Their model has been able to outperform both CNNs and LSTMs based models. Xiao and Cho (2016) also developed a hybrid model made out of a recurrent layer (LSTM) and several convolutional layers. However, the input of their model is not working at word level but at character level. Their model has not been able to outperform either simple CNNs or RNNs model on all common classification benchmarks as their results were highly dependent on the data set.

In this section, the classic methodology of solving text classification problems using machine learning has been introduced. Then, the main components of neural networks namely the neurons, the parameters, activation functions and output layer have been described. From there, an explanation of the training procedure of neural network by initialising the parameters and using a loss function to minimise through a training algorithm has been provided. Finally, regularisation techniques to improve the generalisation power of the neural networks were presented.

Building on the previous explanations, the functioning of CNNs as powerful tools to learn local features thanks to a convolutional layer and a pooling layer has been highlighted. Also, the ability of RNNs to learn sequential features has been explained, and a comparison of both models has been provided together with their most up-to-date applications.

The next section is dedicated to explaining how to convert textual information in a format suitable to be fed in the neural networks described.

3. Document Representation

> "Translation is not original creation - that is what one must remember. In translation, some loss is inevitable" Joseph Brodsky

As computers work with binary information, they are not able to directly interpret a human language. Consequently, the second challenge of TC is to determine the best representation of the input for the classifier to extract the syntactic structure and semantics of texts. Indeed, the effectiveness of most classifiers is heavily dependent on the choice of the representation of the data (Bengio et al., 2013; Wolfram and Zhang, 2008). This task is often referred as learning representation (or document indexing).

Several approaches have been developed and are based on the idea that a document can be described based on a set of the words contained in it commonly called the set-of-words or the bag-of-words approach (Apté et al., 1994; Fuhr et al., 1991; Lewis, 1992; Tzeras and Hartmann, 1993). Furthermore, a word has been proved to be the best unit for text representation (Song et al., 2005) despite promising recent results of representations built at character level (Conneau et al., 2016). However, not all words have the same representative value. Indeed, words such as "and" or "or" would not provide the same information as "music" or "image" about the topic of a document. A solution has therefore been to develop a vector representation of the document where the "importance" of each word is stored. Determining such importance has been a highly investigated field of NLP and will

[15] https://github.com/baidu-research/DeepBench#results The website compares different hardware components for data science tasks including training RNNs and CNNs.

be discussed in 3.1. In the past decade, various methods have been approached and the currently predominant one is the vector space model (VSM) introduced by Salton et al. (1975).

3.1. The vector space model

In the vector space model, documents are represented as a vector where each dimension represents a separate term (i.e., word), and weights are ranging between [0, 1]. 0 is used to express the absence of a term in the document and all value bigger aim to represent the importance of the word in the document.

For $D = \{d_1...,d_n\}$ a set of documents, we define $L = \{l_1...,l_m\}$ being the dictionary (or lexicon), i.e., the set of all different terms occurring in D. Then we define, a document vector as $d_i = \langle w_{1i}...,w_{ni}\rangle$ with w_{ki} representing the weight of the k^{th} term in d_i. Given the vector documents for two documents, it is then possible to determine the similarity - product of vector or inverse function of the angle between the two vectors - between them (Salton et al., 1975). Also, to give all the documents the same importance, each vector document is normalized to have lengths of one.

Encoding the vectors, i.e., determine the weight w_i of a word l_i in a document d_j has been subjects to many discussions (Baeza-Yates and Ribeiro-Neto, 1999; Gövert et al., 1999), but a common approach has been to use the tfidf function introduced in (Salton & Buckley, 1988):

$$w_i(d_j,l_i) = \frac{tf(d_i,l_i)log(\frac{N}{n_t}}{\sqrt{\sum_{j=1}^{m} tf(d_i,l_i)^2(log(\frac{N}{n_t}))^2}}$$

With N being the number of documents in D, n_t the number of documents in D that have an occurrence of l and $tf(d,l)$ the number of time l appears in d. With such a method, deriving the similarity between two documents d_1 and d_2 becomes handy has it can be represented by the Euclidian distance between the two document vectors d_1 and d_2.

However, the drawback is the high dimensionality of the representation. Indeed, for a set D of size N with M unique words in L, the matrix representation is of size NxM whose rows are words and columns are documents (Sánchez et al., 2008). To overcome this issue, some pre-processing can be done on the data which is discussed in the next paragraph.

3.2. Tokenization, filtering and stemming

As exposed before, the most common unit in text classification task is the word. Therefore for each document d, a tokenization is required, i.e removing all punctuations marks and replacing non-text characters by single white spaces (Murty et al., 2011). It has been highlighted that by representing the set of documents on a VSM, we end up with a representation that has a high dimensionality. To reduce it, the first method is to diminish the size of the lexicon L. This can be done by filtering, i.e., removing words from the lexicon. Frakes (1992) point out that words that appear really often bear no particular statistical relevance and can be removed. Also, words such as prepositions or articles do not have content information. In addition, stemming can be performed on the data which consists of grouping words with the same roots and replacing it with the most basic form or stem. It is indeed assumed that words with a common stem will usually have similar meanings (Porter, 1980). Therefore plural forms from nouns or the “ing” from verbs will be removed and the dictionary will contain a list of unique stems.

3.3. Distributed representation of words

Although pre-processing techniques have been able to reduce the dimensionality of the document representation efficiently, the modelling presented earlier has other drawbacks. They include not being able to represent the distance between individuals terms (Kusner et al., 2015) that means it does not capture sense about the semantics of the words. Also, the high dimensionality is often not suitable for computing document distance as they produce matrixes that are almost orthogonal (also called diagonal dominance[16]) (Greene and Cunningham, 2006). Finally, word order is disregarded when constructing such a representation. Some studies have been trying to solve this issue producing a more coherent approach, yet without improving the performance of the downstream classification task. (Blei et al., 2003; Deerwester et al., 1990; Robertson and Walker, 1994).

A breakthrough in document representation occurred when researchers leveraged the distributional hypothesis that states that words that are used and occur in the same contexts tend to purport similar meanings (Harris, 1981). Additionally, the pioneering work of Hinton et al. (1986) on distributed representations contributed to improvement of document representation: rather than representing a word with a single high dimensionality vector, it can be represented as a combination of low dimensional vectors. Each vector is used to represent a feature (such as the tfidf of a word, Chi-Squared, Information Gain (Debole and Sebastiani, 2003)) and the number of features is smaller than the size of the lexicon (reducing thus the dimensionality). Also, relevant features can be selected from a set of all the features (feature selection) and used for the representation. Alternatively, a machine learning approach can be implemented to pick and transform the features (feature extraction) into a lower dimension. This distributed representation of a word is called word embedding. A word is thus represented as a word vector with each dimension representing a feature.

Formally, for each word l in L, a set of linguistic features $[e_1...e_k]$ is extracted or constructed. Each e_i is encoded in a vector $v(e_i)$. l is then represented by a combination of each vector (summation, concatenation or both). The model is therefore made out of dense and low-dimensional vectors

[16] A square matrix A is called diagonally dominant if $|A_{ii}| \geq \sum_{i\neq j} |A_{ij}|$ for all i.

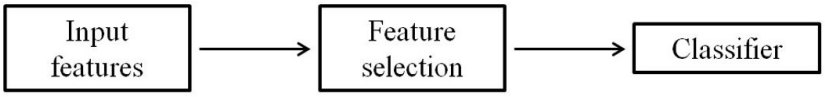

Figure 13: Feature filter model; Source: (John et al., 1994)

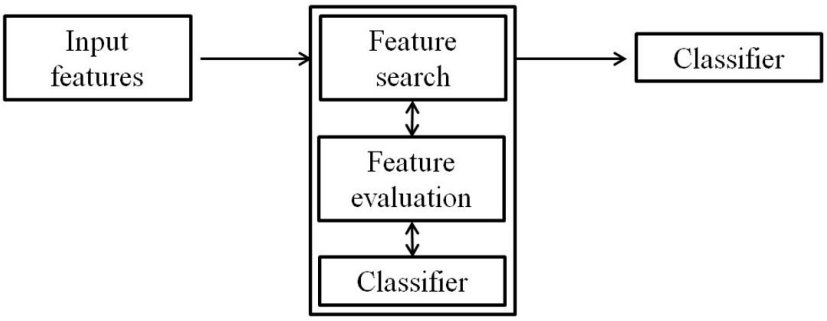

Figure 14: Feature wrapper model; Source: (John et al., 1994)

which lowers the dimension of the representation of the document significantly. These vectors are usually the input of the classifier mentioned in paragraph 2.2.1.

3.4. Feature selection

In the previous section, the notion of feature for words has been introduced. A plethora a features exists concerning words and to generate the representation of a document, some criteria can be used to filter out if a feature is relevant[17] for prediction purpose or not. Feature selections methods are categorized in three different types: filter, wrapper, and embedded methods.

3.4.1. Filter

Filter method (illustration of process in Figure 13) refers to algorithms that treat a possible set of features and rank them independently of the classifier. The top-ranked features are selected (Forman, 2003). Examples of such algorithm include some built on similarity measures such as Pearson's correlation coefficient (Saeys et al., 2008), statistical methods and heuristic search or ensemble learning (Kira and Rendell, 1992) . These methods have the advantage of being fast and thus scalable. However, they do not yield particularly accurate results as they increase bias and are exposed to the selection of redundant features (Jashki et al., 2009).

3.4.2. Wrapper

Wrapper methods (illustration of the process in Figure 14), on the other hand, test every feature in the context of the classifier (Kohavi and John, 1997). They usually involve automated search techniques such as the greedy search strategy (Guyon and Elisseeff, 2003). These methods are more accurate than filter methods but come with high-computing costs.

[17]Discussions about the meaning of relevance and its definition can be found in (Sag et al., 2002)

3.4.3. Embedded

Finally, embedded methods perform feature selection during the execution of the classifier (being therefore embedded in the classifier). Therefore, the feature selection and the training methods of the classifier are not separated steps. Conventional methods may use decision three algorithm (Genuer et al., 2010) or multinomial logistic regression (Cawley et al., 2007). These methods are similar to wrappers but are specific to classifiers, which makes them computationally less expensive as they are optimized for them.

3.5. Feature extraction

As mentioned, a board range of features exists and some that humans find useful will not necessarily be useful for the models and vice-versa. Therefore, all the features known based on basic statistics about a document can be used, referred as count based methods. Alternatively, machine learning techniques such as neural networks can be used to let the model determine which features are important or not.

3.5.1. Count based methods

In feature extraction, the feature space - set of all possible features - is converted to another space with a lower dimension keeping the most informative and discriminative features (Gomez et al., 2012). Methods include Principal Components Analysis (PCA) and Latent Semantic Analysis (LSA).

PCA is a statistical method that transforms the set of features (possibly correlated) into new features that are uncorrelated called principal components using a linear transformation. Like in feature selection, the best new features are then selected.

LSA (Deerwester et al., 1990) - also referred as Latent Semantic Indexing - is a technique developed to address the problems deriving from the use of synonymous, near-synonymous, and polysemous words as features of document representations (Sebastiani, 2002). The process involves

identifying the relevant words - using, for example, the tfidf of words - and then constructs a term-document matrix as described in paragraph 3.1 . Then the matrix is decomposed using Singular Value Decomposition - a technique closely related to PCA. The result is a set of lower dimension features vectors that were constructed looking at patterns of word usage in the documents. In essence, the features are usually hardly interpretable as there are meant to capture latent (hidden) relationship between words. LSA provided a significant step forward in document representation as it accounted for semantic characteristics of texts, synonymy of words and partially polysemy (Deerwester et al., 1990).

Glove: the state-of-the-art of count-based model

In the paper introduced by Pennington et al. (2014), the authors argue that the count of words in a document carries meaningful information, but also the count of a word w_i in the context of another word w_j called co-occurrence probability. Following their example, for the context of steam and ice, it is expected that the ratio of the probability of observing solid in the context of ice and the probability of observing solid in the context of steam - $\frac{p(\text{solid}|\text{ice})}{p(\text{solid}|\text{steam})}$ - to be high. Likewise, this ratio for the word gas in the same contexts is expected to be small. The model therefore constructs a matrix X_{ij} based on word-context co-occurrences and factorise it to obtain the vectors. To complete the latter step, the authors use a weighted least squares regression model that is able to encode the information available in the probability of co-occurrence. When constructing the word vectors, the objective is to minimize the difference between the product of the two word vectors w_i and w_j (word and context), and the logarithm of the probability of co-occurrence (plus a bias for each word) which is expressed as follow:

$$J = \sum_{i,j=1}^{V} f(X_{i,j})(w_i^T W_j + b_i + b_j - log(X_{ij}))^2$$

$$\text{with} f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{xmax})^{3/4} & \text{if } X_{ij} < X_{max} \\ 1 & \text{otherwise} \end{cases}$$

3.5.2. Neural networks for words embedding

Methods to represent words explained so far are referred as count-based methods in (Baroni et al., 2014) as values in vectors are derived from co-occurrence counts. The authors point out the weaknesses of these models namely problem of scalability, poor performance on word analogy evaluation and task-dependent (except for GloVe that performed pretty well on the latter). To deal with these issues, new models have appeared referred as predictive-based methods. This new generation of models where first exposed in 1981 (Hinton et al., 1986), but have demonstrated their utility in (Collobert and Weston, 2008) building up on previous research on deep neural network (Bengio et al., 2003) challenging the previous state-of-the-art methods. Rather than counting words co-occurrence, generating the vectors and reducing the dimensionality, these methods try to directly generate the vectors by predicting a word from its neighbours or vice versa. Thus, as similar words occur in similar contexts, the system assigns similar vectors to similar words (Baroni et al., 2014). The comparisons between count-based and predictive-based methods have demonstrated the superiority of the latter in lexical semantics tasks including semantic relatedness, synonym detection and analogy, (Cambria et al., 2017; Socher et al., 2011; Turney and Pantel, 2010; Weston et al., 2010), but have failed to leverage statistical information from documents as they are based on context windows of a few words. It must however be pointed out that no methods of adequately evaluating the quality of vector representations have been developed. Indeed, so far they have been evaluated on word similarity or analogy metrics, but these only correlate weakly with downstream tasks performance such as TC (Tsvetkov et al., 2015).

The next section is dedicated to presenting the most famous predictive-based methods using neural networks.

Word2Vec

In the paper (Mikolov et al., 2013), the authors offer two models. One model predicts a word given a context (Continuous Bag-of-Words model) and the other one given a context, predicts a word (Skip-gram model). Using these models with such objectives will not result in word vectors per se in the output layer. Indeed, the word vectors will be present in the different weight matrices of the models. The intuition behind it was previously expressed: if two words are similar, they should appear in a similar context and thus their representation should be similar.

The results of the learned embedding were a big step forward in the vector representation of words. Indeed, they were not only capable of training a huge list of words (1.6 billion) in less than one day, but also captured semantic meaning of words. It is illustrated by an example that has become famous in the NLP community: having the vector for the words queen, women, men and king, they have performed the following calculation successfully:

$$queen - women + men = king$$

In the same vein, they were capable of capturing the semantic behind the sentence “France is to Paris as Germany is to Berlin”.

Continuous Bag-of-Words model

To achieve this amazing result, they leverage a modification of the MLP presented in 2.2 as pictured in Figure 15. They used the same structure with an input layer that represents one-hot-encoded[18] words, a single hidden layer and an output layer with the goal to classify. They looked at four words and given these words try to find a word that would fit in the middle of these four words which can be defined as a classification problem. With this architecture, they therefore end

[18] A one-hot-encoded vector consists of 0s in each dimension with the exception of a 1 in a dimension used uniquely to identify the word by its position in the sentence.

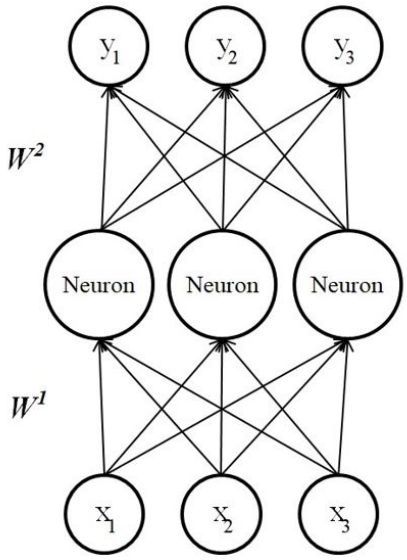

Figure 15: x_1, x_2, x_3 is the input layer made out of one-hot-encoded vectors. The hidden layer is represented by neurons and the output layer is y_1, y_2, y_3. This representation would fit a sentence made out of three words. Source: Author's own representation

up with two weights matrices W^1 (from the input layer to hidden layer) and W^2 (from the hidden layer to the output layer) that form the parameters θ. The output layer, thanks to a softmax function, represents a multinomial distribution of all words given a context[19]. We understand that the goal is to maximize the probability of a word given a context, which consists of minimizing the opposite probability:

$$Lossfunction = -log(p(word|words_{context}))$$

The log appears because we are using the softmax function to transform the last layer in a probability distribution.

With this setting, the actual output of interest are the matrices W^1 and W^2. Indeed, after training, the matrix W^1 contains in its lines vectors that, for a word, represents the context. On the other hand, W^2 has a vector representation of a word in its columns, which is precisely what we are looking for (Rong, 2014).

Skip-Gram model

The Skip-Gram model is very similar to the CBOW model. It is just doing the opposite: given a word, predict the context. Indeed, for a word given, it will pick another word and estimate the probability of that word being around[20] it. Consequently, the rows of W^1 will now represent the vector representation for a word and the column of W^2 will represent context vectors.

From the two models, the authors have been able to create vectors that were capable of representing words better syntactically (with the CBOW model) and semantically (with the Skip-gram model) than previous neural models (Mikolov et al., 2009; Mikolov et al., 2010). However, the models have limitations. The first one is that for one word, they assign one vector and therefore they are unable to represent polysemy words. To (partially) solve the issue, Upadhyay et al. (2017) developed an algorithm that learns word representation jointly across language. The intuition behind it is that a polysemy word in language could be translated into distinctive words in another language. Using the authors' example, the word bank in English which has several meanings can be translated to banque or banc in French which capture two different meanings with two different words. Therefore, by learning using multiple languages, the algorithm can identify which sense to use. The second caveat is that the meaning of multi-word expressions[21] is not captured. Indeed, expressions such as "in short" or "Los Angeles" are poorly encoded as they will be represented in two vectors. Some methods have been developed to capture phrasemes without however improving the performance of downstream tasks such as text classifications (Hashimoto and Tsuruoka, 2016; Yu and Dredze, 2015). Finally, training the CBOW or Skip-gram is computationally expensive on large datasets. Thus, rather than generating an embedding for every task performed, it is common practice to use pre-trained vectors[22]. However, it is often the case that some words in the datasets are not part of the pre-trained vectors. These words are referred in the literature as out-of-vocabulary words (OOV). Being able to assign a proper representation to the input, including OOV

[19] The context can be made out of one word or several words preceding or following the word of interest

[20] "around" is predefined and can be for instance 2 words before and 2 words after. The authors of the model found that increasing the size of the context resulted in better quality of word vectors.

[21] Multiword expressions are also called phraseme. An accurate discussion about the typology of multiword expression is present in (Sag et al., 2002)

[22] Mikolov et al., after developing their model, published a set of pre-trained vectors on Google News with 3 million words and phrases.

words, can alter the performance of the downstream task by up to 6% over random initialization (Dhingra et al., 2017). One way of dealing with OOV words is to replace them with a unique token, UNK (Chen et al., 2016; Shen et al., 2017), and use it for training. Another method is to assign each OOV word a randomly generated vector at test time (Kim, 2014) or a unique randomly generated vector (Dhingra et al., 2017). A recently suggested method in (Dong and Huang, 2018) is to combine pre-trained vectors and vectors generated during training. When a word is present in the pre-trained vectors and the training set, then a new vector is constructed concatenating both vectors. If one of them is missing, it is replaced by a null vector.

Proposal

I would like to propose a variation of the method developed in (Dong and Huang, 2018). Rather than pre-trained or embedded vectors concatenating with a null one, I suggest to concatenate them with a unique vector sampled from a distribution such as the vector has the same variance as the other ones. The idea of generating vectors from such a distribution is not new as it was already expressed in (Kim, 2014). However, I combine both approaches with a concatenation operation as shown in Figure 16.

FastText

As expressed earlier, training word vectors can be computably expensive to learn and dealing with OOV words can be challenging. Bojanowski et al. (2016) offer an extension of Word2Vec named FastText to learn a vector representation of word quickly and to (partially) deal with OOV words. Rather than learning vector representation of words, they learn representations of character n-grams. Then a word is simply the sum of this character n-gram[23]. For instance, for the word “hello” , extracting a character 3-gram, will give the vector representations of: “he”, “hel”,”ell”,”llo”,”lo”. This allows leveraging the morphology of words and therefore reducing the number of necessary computations. Also, when dealing with OOV words, it is likely that new words can be expressed as a combination of the learned character n-gram.

While Section 1 described various neural networks models, Section 2 of this work has been first dedicated to explaining the rationale behind the conversion of words into vectors as inputs for the models. From the simple bag-of-words method that uses a high dimensional representation, the notion of feature and tricks to diminish the size of that representation have been introduced. Furthermore, methods to select features but also techniques that extract them were explained. For the latter, the dichotomy that exists between count-based solutions - with its best representative GloVe - and prediction-based solutions such as Word2Vec and FastText has been presented. Finally, a solution to deal with words that are not present in pre-trained vectors data set has been proposed.

As the state-of-the-art of neural networks models for TC and word embedding methods have been identified, the next section describes the benchmark used to evaluate them on the LogMeIn data. Also I introduce another dataset to assess the proposal.

[23] In their study, they extract all the n-gram with n ranging from 3 to 6.

4. Experiment

> “Experience is simply the name we give our mistakes” Oscar Wilde

To evaluate the state-of-the-art classifiers on the data provided by LogMeIn, the CNN model as described in (Kim, 2014) and a hybrid CNN+LSTM model as described in (Zhou et al., 2015b) were implemented. Also, in order to evaluate the proposal, the implementation was tested on two datasets: the LogMeIn one and the TREC (Li and Roth, 2006) dataset.

Moreover, I test different techniques of embedding, a random initialization with different dimensions, using pre-trained vectors generated by Word2Vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), FastText (Bojanowski et al., 2016), FastText with subwords information (Mikolov et al., 2017), the method introduced in (Dong and Huang, 2018) and my proposal.

4.1. Data

The LogMeIn dataset is made out of customer reviews based on the product GoToMeeting - an online meeting and video conferencing software. It has been annotated such as reviews are classified under the categories “screen”, “video”, or “audio”. Unfortunately, reviews may appear in several categories in the original dataset. Therefore the dataset is used for a binary classification problem whether the review is under the category ”audio” or not. Also, to avoid bias, an undersampling procedure has been performed on the “non-audio” category to get a 1:1 ratio of “audio” and “non-audio” entries. It consisted of randomly dropping data points until parity was reached.

The TRAC dataset is a common benchmark used for multi-topic categorisation. It is made out of a question that refers to a person (884 samples), a location (616), numeric information (664), an abbreviation (62), and an entity (937). The task is to classify a question under one of these categories. Statistics for both datasets are available in Table 3.

4.2. Models

The CNN is the same as described in Section 2.3. As explained, a CNN takes a fixed length of input; therefore each sentence is padded to the maximum sample length. It is done by adding symbols for sentences that are shorter than the maximum sentence length in the training set and taking off words for samples that are longer in the test set. The window sizes of the filters of the convolutional layer are 3, 4 and 5 words with 100 feature maps each which are combined with a ReLu non-linearity. A 1-max pooling operation is performed on the output of the filters and then a 0.5 dropout is applied. The batch size is 32 and the loss function is the softmax cross entropy function shown below:

Algorithm : Combine pre-trained word embedding with those generated on training set.

Input : Pre-trained word embedding set $\{U_w|w \in S\}$ where $U_w \in \mathbb{R}^{d_1}$ is embedding vector for word w. Word embedding $\{V_w|w \in T\}$ are generated on training set where $V_w \in \mathbb{R}^{d_2}$. P is a set of word vocabulary on the task dataset. a such as $U[a; a]$ has the same variance as the vectors in $\{V_w|w \in T\}$. b such as $U[b; b]$ has the same variance as the vectors in $\{U_w|w \in S\}$

Output: A dictionary with word embedding vectors of dimension $d_1 + d_2$ for $(S \cap P) \cup T$.

res = dict()
for $w \in (S \cap P) \cup T$ **do**
  if $w \in S \cap P$ and $w \in T$ **then** res$[w] = [U_w; V_w]$;
  else if $w \in S \cap P$ and $w \notin T$ **then** $V_w \sim U[a; a]$ and res$[w] = [U_w; V_w]$;
  else $U_w \sim U[b; b]$ and res$[w] = [U_w; V_w]$;
end
Return res

Figure 16: Algorithm suggested to deal with OVV word, using the same notation as in (Dong and Huang, 2018); Source: Author's own representation

Table 3: Statistics about LogMeIn and TREC datasets Source: Data compiled by author

	LogMeIn	TREC
Total Samples	1137	4000
Train Samples	1023	3600
Test Samples	114	400
Number of categories	2	6
Average length of samples	16.975	10.1545
Median length of samples	10	10
Maximum length of samples	205	37
Total unique words	2786	6987

$$L(\hat{c}, c) = -log(\frac{e^c}{\sum_{i=1}^{C} e^{\hat{c}}}$$

The CNN+LSTM model is first made out of convolutional layer that extracts higher-level sequences of word features. It is the same convolutional layer as the simple CNN model. Unlike in (Zhou et al., 2015a) a 1-max pooling operation is kept after the convolutional layer. Then an LSTM capture long-term dependencies over each window feature created by the convolutional layer. After the LSTM, a 0.5 dropout is applied just before the softmax cross entropy layer. The batch size is also 32.

4.3. Word embedding

I test 8 forms of word embedding. First, I try two random assignations of vectors to words from the uniform distribution U[-0.25,0.25]. The first set of vectors is of dimension 300 and the second of dimension 600. This is to check the effect of the size of word embedding on the downstream classification task.

Also, a third embedding is generated from pre-trained vectors with Word2Vec made available by Mikolov et al. (2013). It includes a vocabulary of 3 million words and phrases that were trained on about 100 billion words from a Google News dataset. The vectors are of dimension 300.

The fourth embedding is generated from pre-trained vectors with GloVe made available by Pennington et al. (2014). They were trained on a Wikipedia and Giga word datasets[24] and consist of 400'000 words. The vectors are of dimension 300.

The fifth and sixth embedding are generated from pre-trained vectors with FastText (Bojanowski et al., 2016; Mikolov et al., 2017). They consist of 1 million word vectors trained on Wikipedia 2017[25], UMBC web base corpus[26] and statmt.org news dataset[27]. One is trained with subword information the other is not.

The sixth embedding method is the proposal of Dong and Huang (2018) which consists of vectors of dimension 600 made out of the concatenation of Word2vec pre-trained vectors and vectors trained directly on the database with the CBOW algorithm. If a word is not present in either of the two sources it is represented by a null vector of size 300. Finally, the last embedding is my proposal. It is the same as the sixth except that vectors that are not found are represented by a unique vector initialized from a uniform distribution U[-0.25,0.25] of size 300.

The code was written in Python using TensorFlow[28] and

[24] https://catalog.ldc.upenn.edu/LDC2012T21
[25] http://wiki.dbpedia.org/Datasets
[26] https://ebiquity.umbc.edu/resource/html/id/351
[27] http://www.statmt.org/
[28] https://github.com/tensorflow/tensorflow/releases/tag/v1.8.0

the code of Jie Zhang available on GitHub[29] as a basis for the implementation of the CNN and CNN+LSTM. Also, the gensim[30] implementation of Word2Vec is used to generate the word embedding of the TREC and LogMeIn dataset. The code is available in Appendix 8.1, Appendix 8.2, and Appendix 8.3. Finally, the tests were performed on a Central Processor Unit (CPU) Intel ® Core ™ i7-7500U @ 2.70 GHz and 8GB LPDDR3-1866Mhz RAM. As some randomness is part of each model, they are tested 5 times. The average performance as well as a 95% confidence interval is reported. The 16 models, their name, and characteristics are summarized in Table 4.

5. Results and Discussion

> "I am just a child who has never grown up. I still keep asking these 'how' and 'why' questions. Occasionally, I find an answer" Stephen Hawking

The first part of this section presents the results necessary to compare the effectiveness of the CNN and the CNN+LSTM models. Then, the effects of the different word embedding methods are presented and discussed.

5.1. CNN and CNN+LSTM

A first remark is that I was not able to achieve the same results as (Kim, 2014) and (Zhou et al., 2015a) on the TREC dataset. As pointed out in Wang et al., 2018), measures can change depending on the pre-processing of the data. In my experiment, short forms such as "I'm" or "He'll" are split in two distinctive words, uppercase characters are replaced by lowercase ones and non-alphanumeric characters except punctuation symbols were removed. Also, I do not use the same hyperparameters and architecture. Indeed, when Kim use ADADELTA (Zeiler, 2012) as the algorithm to update gradients, I use ADAM (Kingma and Ba, 2015). Indeed, it was shown in the aforementioned paper that both methods efficiently lower the cost of training on CNNs, but ADAM is better at that task than ADADELTA, especially on deep neural networks. However, despite being more efficient, ADAM and ADADELTA should converge toward the same local minimum which should therefore not change the performance of the downstream task. Tests with an ADADELTA function have been performed on the TREC dataset and the CNNFXT model and no significant changes were perceived confirming the previous statement. Also, Kim uses 25 training cycle (or epochs) whereas on my benchmark I only use 3. The more epoch is used, the better trained the model is, however the higher the risk of overfitting. I have personally chosen 3 epochs to run more tests in the benchmark. Indeed, multiplying the amount of epochs inevitably increases the training time of each model. However, after testing on the CNNW2V model on the TREC dataset with 25 epochs, again no significant differences were observed as it appears that the model plateaus at about 84% accuracy. Despite these differences, I have not been able to identify other sources responsible for the performances differences.

Similarly, the results of the CNN+LSTM models do not replicate the ones from C. Zhou et al. The first difference is the presence of the max pooling layer after the convolutional layer. The authors argue that the operation breaks the sequence order as the selected features from the convolutional layer are discontinuous. However, the role of the max-pool is first reducing the computation for the next layer, but also to extract the most salient features in the sample. A test has been performed on the LSTMW2V model, but again no significant changes in term of performance have been observed.

The number of epochs, however, affects much more the LSTM models. Indeed, the results reported in Table 5 come from a training procedure of 3 epochs, but by increasing it to 25, the accuracy for the model LSTMW2V jumped from 63% and 77% to 86% and 97% on TREC and LogMeIn datasets respectively[31]. To investigate it, I changed the gradient updating algorithm of the CNN+LSTM models. In their initial configuration (and the one tested in this work), the algorithm used for the update of the gradient of the CNN+LSTM models is the RMSprop (Tieleman and Hinton, 2012). A test has been performed using the ADAM algorithm with 3 epochs on both datasets and results are conclusive achieving similar results than with the RMSprop with 25 epochs. Therefore, the first recommendation when using a CNN+LSTM model is to use the ADAM algorithm as gradient update function. It requires less training time while yielding better results than RMSprop for the same number of epochs.

In the lights of the first conclusion, a second benchmark has been performed using a CNN+LSTM with an ADAM gradient update function and 3 epochs to compare the model directly with the simple CNN models. The results are reported in Table 6.

Besides, for both configurations, I use filter sizes of 3, 4 and 5 on the convolutional layer. In their original paper, C. Zhou et al. conclude that a filter of size 3 yields better results for the CNN+LSTM architecture. However, Kim reports better results using filter size of 3, 4, 5 on a simple CNN one. I find better results using filter size of 3, 4 and 5 on both datasets with both configurations. As few data were collected (5 per model), to investigate differences between CNN and CNN+LSTM, I aggregate the measures of all CNN tests and all CNN+LSTM tests. The mean and a 95% confidence interval are reported in Table 7.

There are no statistical differences observed between the two models on the dataset experimented. The CNN results are similar in magnitude to the results in (Zhou et al., 2016b) on the TRAC dataset, but no improvement is observed by adding the LSTM layer to the architecture unlike in (Zhou et al., 2015a). First, a better fine-tuning of the CNN+LSTM

[29] https://github.com/jiegzhan

[30] https://radimrehurek.com/gensim/

[31] These figures might be inflated as I did not check whether an overfitting problem was appearing or not.

Table 4: Summary of the different model tested and their features; Source: Data compiled by author

Name	Model	Dimension	Embedding
CNN300	CNN	300	Random
CNN600	CNN	600	Random
CNNW2V	CNN	300	Pre-trained Word2Vec (Mikolov et al., 2013)
CNNGVE	CNN	300	Pre-trained GloVe (Pennington et al., 2014)
CNNFXT	CNN	300	Pre-trained FastText (Bojanowski et al., 2016)
CNNFXT_SUB	CNN	300	Pre-trained FastText (Mikolov et al., 2017)
CNNW2V600_NULL	CNN	600	Word2Vec + pre-training on dataset (Dong and Huang, 2018)
CNNW2V600	CNN	600	Pre-trained Word2Vec + pre-training on dataset (proposal)
LSTM300	CNN+LSTM	300	Random
LSTM600	CNN+LSTM	600	Random
LSTMW2V	CNN+LSTM	300	Pre-trained Word2Vec (Mikolov et al., 2013)
LSTMGVE	CNN+LSTM	300	Pre-trained GloVe (Pennington et al., 2014)
LSTMFXT	CNN+LSTM	300	Pre-trained FastText (Bojanowski et al., 2016)
LSTMFXT_SUB	CNN+LSTM	300	Pre-trained FastText (Mikolov et al., 2017)
LSTMW2V600_NULL	CNN+LSTM	600	Pre-trained Word2Vec + pre-training on dataset (Dong and Huang, 2018)
LSTMW2V600	CNN+LSTM	600	Pre-trained Word2Vec + pre-training on dataset (proposal)

Table 5: Classification accuracy of the different models on the LogMeIn and TREC datasets. The best result is in bold; the second best is in italic. The 95% confidence interval is reported in parentheses. Here the CNN+LSTM models are trained with RMSprop; Source: Data compiled by author

	LogMeIn	TREC
CNN300	0.9035 (± 0.0241)	0.7765 (± 0.0169)
CNN600	0.8614 (± 0.0190)	0.7810 (± 0.0107)
CNNW2V	0.9333 (± 0.0253)	0.8425 (± 0.0831)
CNNGVE	0.9123 (± 0.0108)	0.7730 (± 0.0162)
CNNFXT	0.9386 (± 0.0139)	0.8455 (± 0.0161)
CNNFXT_SUB	0.9193 (± 0.0385)	0.8300 (± 0.0121)
CNNW2V600_NULL	0.9351 (± 0.0268)	0.8445 (± 0.0176)
CNNW2V600	0.9273 (± 0.0268)	0.8165 (± 0.0099)
LSTM300	0.6069 (± 0.0478)	0.7000 (± 0.0438)
LSTM600	0.5825 (± 0.0321)	0.7005 (± 0.0231)
LSTMW2V	0.6316 (± 0.0311)	0.7745 (± 0.0348)
LSTMGVE	0.7175 (± 0.0419)	0.7380 (± 0.0185)
LSTMFXT	0.6070 (± 0.0575)	0.7550 (± 0.0360)
LSTMFXT_SUB	0.5316 (± 0.0476)	0.6835 (± 0.0176)
LSTMW2V600_NULL	0.6386 (± 0.0228)	0.7845 (± 0.0127)
LSTMW2V600	0.5912 (± 0.0275)	0.7535 (± 0.0261)

model is necessary. As pointed out in the previous paragraph, LSTM based model are very sensitive to the number of epochs and update algorithm function. Further investigations must be performed on the CNN+LSTM model to identify the right number of epochs, but also the ideal batch size. Indeed, a test has been conducted with the LSTMW2V model with 6 epochs showing an accuracy of 84.25% on TREC, which is higher than all other tests (Appendix 8.4).

In both models, the convolutional layer performs the same task which explains the similarity of results, but the addition of the LSTM layer requires further work to leverage the memory cell capacity. Also, as pointed out in (Yin et al., 2017), CNNs and RNNs are expected to yield comparable results when sentences are short which is the case as shown in Table 3. Finally, both algorithms perform better on LogMeIn than TREC, but tasks are also slightly different as one is a binary classification and the other one is a 6 categories classification task.

5.2. Effect of Word Embedding

Despite not being able to replicate other state-of-the-art results, effects regarding the word embedding are captured by both models by holding the rest of the parameters constant. To capture only the effect of the word embedding method, an aggregation has been made between results of CNN and CNN+LSTM based models. Results on the LogMeIn dataset is present in Figure 17 and results on TREC are available in Figure 18.

Table 6: Classification accuracy of the different models on the LogMeIn and TREC datasets. The best result is in bold; the second best is in italic. The 95% confidence interval is reported in parentheses. Here the CNN+LSTM models are trained with ADAM; Source: Data compiled by author

	LogMeIn	TREC
CNN300	0.9035 (± 0.0241)	0.7765 (± 0.0169)
CNN600	0.8614 (± 0.0190)	0.7810 (± 0.0107)
CNNW2V	0.9333 (± 0.0253)	0.8425 (± 0.0831)
CNNGVE	0.9123 (± 0.0108)	0.7730 (± 0.0162)
CNNFXT	0.9386 (± 0.0139)	0.8455 (± 0.0161)
CNNFXT_SUB	0.9193 (± 0.0385)	0.8300 (± 0.0121)
CNNW2V600_NULL	0.9351 (± 0.0268)	0.8445 (± 0.0176)
CNNW2V600	0.9273 (± 0.0268)	0.8165 (± 0.0099)
LSTM300	0.87016 (± 0.0331)	0.7855 (± 0.0201)
LSTM600	0.89474 (± 0.0300)	0.7895 (± 0.0082)
LSTMW2V	0.8895 (± 0.0083)	0.835 (± 0.0271)
LSTMGVE	0.91924 (± 0.0162)	0.816 (± 0.0141)
LSTMFXT	0.90596 (± 0.0162)	0.8365 (± 0.0180)
LSTMFXT_SUB	0.91756 (± 0.0122)	0.8145 (± 0.233)
LSTMW2V600_NULL	0.91754 (± 0.0176)	0.825 (± 0.0258)
LSTMW2V600	0.91404 (± 0.0100)	0.833 (± 0.0220)

Table 7: Results of CNN and CNN+LSTM based models; Source: Data compiled by author

Models	LogMeIn	TREC
CNN	0.916348 (± 0.0116)	0.813688 (± 0.0116)
CNN+LSTM	0.90359 (± 0.0119)	0.816875 (± 0.0094)

First, it can be observed that, as expected, the effects of the word embedding method are dependent on the dataset. Indeed, as results are not necessarily conclusive on the LogMeIn data, they are on the TREC one. Here I assume two effects must be taken into account. First, the larger the size of the vocabulary (i.e., total unique words in Table 3), the higher the model can leverage pre-trained vectors for similar ratios of words found/total words. Also, the higher the noise in the dataset, the higher will be the variance in performance of the downstream model.

Furthermore, looking in Table 6, it is also striking that the improvement in accuracy induced by the use of pre-trained vectors is dependent on the downstream model used to tackle the classification task. Indeed, compared to a random initialization, using the pre-trained FastText vectors can improve the accuracy by up to 10.2% using a CNN and 8.9% using a CNN+LSTM. Also, the impact is even greater if the subsequent model is not ideally trained. Indeed, in Table 5, from a random initialization of dimension 300 to the use of Dong & Huang the accuracy is potentially jumping from 65.62% to 79.72%, a 14.1% gain. The figures found are higher than the ones that in (Dhingra et al., 2017). As pre-trained vectors already carry information when fuelled to the subsequent classification model, they enhance the performance of the classifier. However, the nature of the gain, whether linear or not, has not, as far as I know, been investigated. It could be investigated by studying the relation between the number of epochs and the relative gains by using pre-trained vectors. My hypothesis is that the marginal gain of using pre-trained vectors is diminishing as the number of epochs increases.

Second, simply doubling the dimension of the word embedding does not change the performance of the classification task with random initialization. However, doubling the dimension, allows reducing the variance of downstream results as observed in Figure 18.

Third, using pre-trained vectors yields indeed better results over random initialization which can be observed on the TREC results as well. As the matter of fact, except for GloVe pre-trained vectors, all embedding methods give better results than random initialization.

Fourth, using subword information from the pre-trained vectors of FastText does not improve either the performance. Further investigations using an architecture that uses character-level information such as in (Xiao and Cho, 2016; Zhang et al., 2015) should be performed to investigate whether these models can leverage these subword features better. In addition, it can be observed that methods leveraging the CBOW algorithm such as Word2Vec and FastText outperform GloVe. Looking at a 2D projections of the TREC vocabulary generated from FastTtext pre-trained vectors using t-distributed stochastic neighbour embedding (T-SNE[32])

[32] A technique used to reduce the dimensionality of the vectors while keeping some features. The implementation has been done using the sckit-learn library (http://scikit-learn.org/stable/index.html) and matplotlib (https://matplotlib.org/)

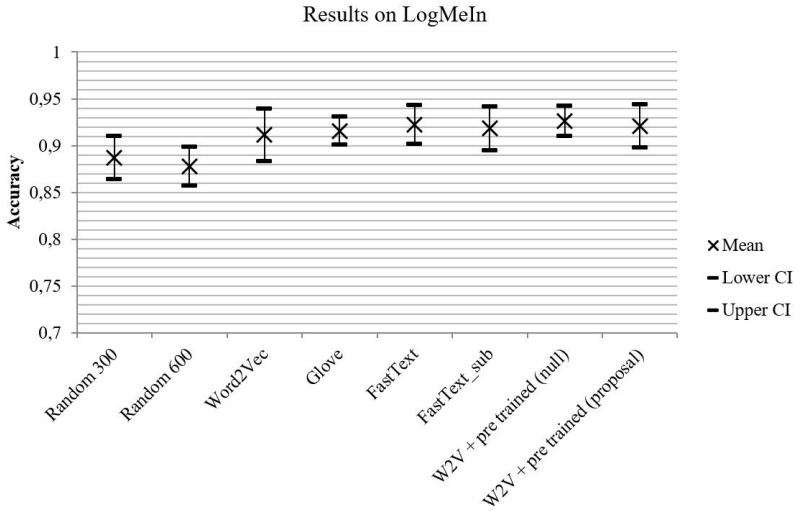

Figure 17: 95% Confidence intervals of the results of the models on the LogMeIn dataset; Source: Author's own representation

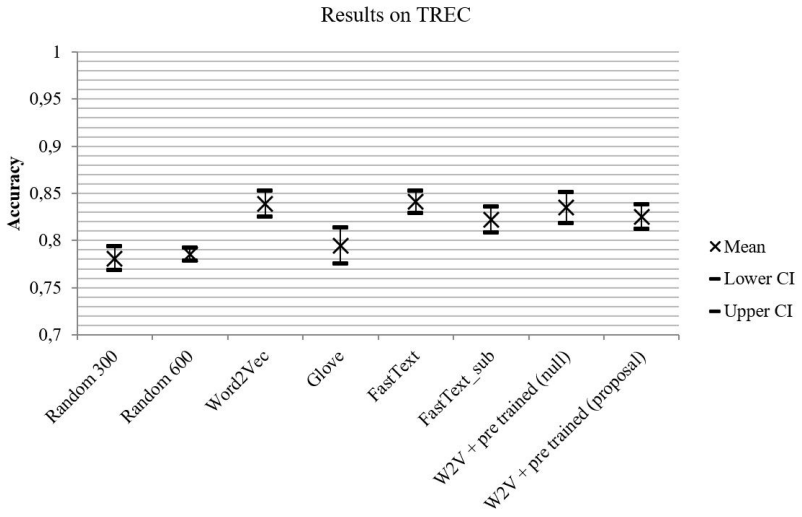

Figure 18: 95% Confidence intervals of the results of the models on the TREC dataset; Source: Author's own representation

in Figure 19, we can observe that FastText vectors are better at capturing semantic information as words with similar meanings are clustered. However looking at the GloVe projections in Figure 20, we observe that fewer clusters appear and that the projection is similar to a random initialization with however increased variance. As a consequence, this variance also spills over the variance of performance of the models.

Finally, my proposal does not show a statistical difference with Dong & Huang's algorithm. As shown in Table 8, the number of out-of-vocabulary words is relatively low and the effectiveness of both methods is therefore hard to evaluate as a few information is added by the algorithm.

Further tests on different datasets with a greater number of OOV words should be performed. Indeed, words whose initialization is not random due to pre-training on the dataset include "gotomeeting", "gotowebinar" or "seminario" which do not help much on determining whether the review is audio or not (low representative value). Likewise, on the TREC dataset these words include "spielberg", "mozambique", "gould". In TREC dataset, totally missing words include numbers such as "1991", "1967" or "327", or words such as 'occam', 'rockettes', 'quetzalcoatl', 'khrushchev', On the LogMeIn dataset missing words also includes numbers such as "995" or "65", misspelled words such as "'presentationbefore" or 'probleme', and words in another language such as "perfekt", "einfache" or "reiniciar'. A suggestion to improve the performance on the LogMeIn dataset is to use a combination of pre-trained vectors from different language as the dataset includes samples in another language than English.

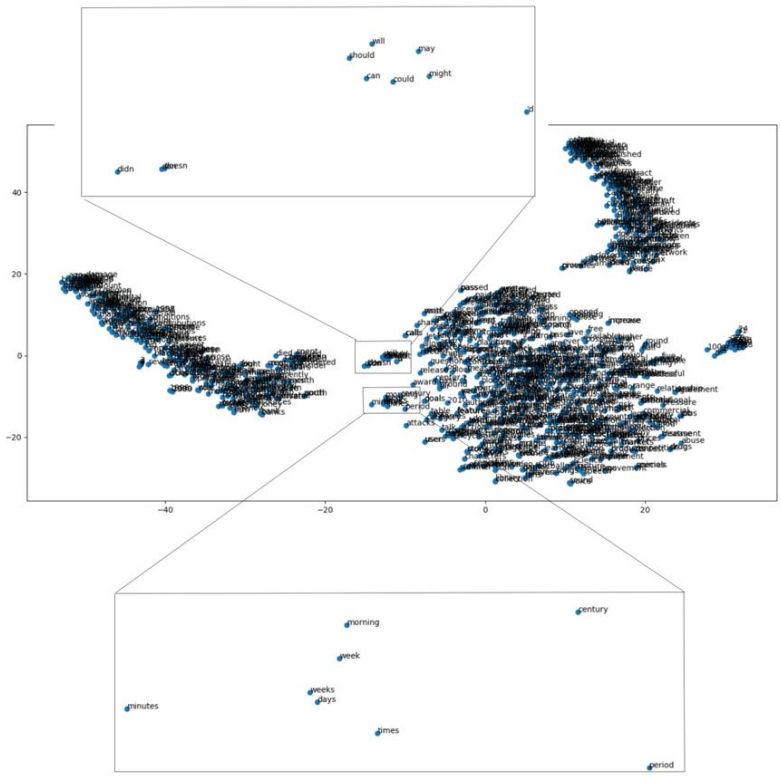

Figure 19: T-SNE projection of FastText pre-trained vectors for the TREC vocabulary. Two clusters are shown; one presenting words about time (bottom) and the other one with modal verbs (top); Source: Author's own representation

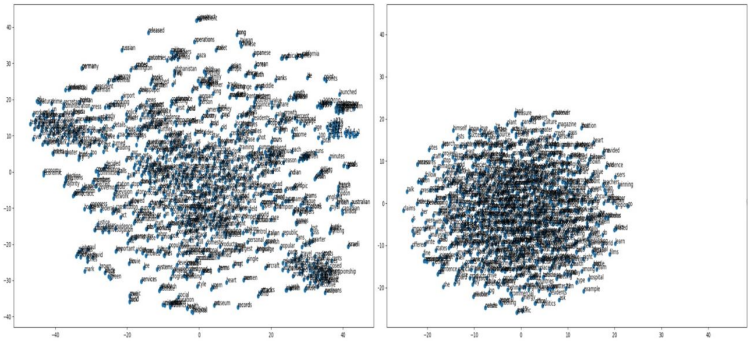

Figure 20: TT-SNE projection of word vectors for the TREC vocabulary. On the left vectors from GloVe. On the right random initialization of dimension 300; Source: Author's own representation

Table 8: Descriptive statistics about the words found using different embedding methods; Source: Data compiled by author

	LogMeIn	TREC
Total unique words	2786	6987
Found in Word2Vec	2504	6036
Found in Glove	2575	6814
Found in FastText	2603	6501
Proposal		
Found in both Word2Vec and generated vectors	393	863
Found only in Word2Vec	2111	5173
Found only in generated vectors	10	25
Not found	272	926

6. Conclusion

> "As machines become more and more efficient and perfect, so it will become clear that imperfection is the greatness of man." Ernst Fischer

As the literature in deep learning is flourishing, so is the range of models and their application. In this thesis, I have first described two common architectures used for text classification tasks namely convolutional neural networks (CNNs) and recurrent neural networks (RNNs). I have compared their performance and training procedure on a text classification task and found that CNNs are easier to train and yield better results. Nevertheless, according to the literature review, hybrid models combining both architectures can yield better results. Through my benchmark, I could not verify this statement as I could not reach optimal performance through my implementation but could highlight that sensitive factors for RNNs include the gradient update function and the number of epochs. I could also show that they are computably more expensive to train.

Also, I have discussed ways to convert textual data into inputs that the aforementioned models can leverage to improve their performance. I have highlighted that the use of pre-trained vectors can increase by up to 10.2% the performance of the subsequent model. Concretely, I have found that methods that generate word vectors based on a Continuous Bag of Word (CBOW) algorithm such as Word2Vec or FastText yield better results than count-based methods such as GloVe. Moreover, after observing the empirical results, I have stated that this gain is probably diminishing and therefore not linear as the subsequent models become fine-tuned. This could be subject to further research to confirm or not my hypothesis. I could also confirm that the gain was dependent not only on the subsequent models but also on the dataset used.

Finally, I proposed an algorithm for these models to deal with words that are unknown with unfortunately inconclusive results. Further evaluations are necessary with datasets that include a higher proportion of unknown words with a higher representative value. The benchmark used was designed to assess models on a classification task and not sufficient to evaluate my proposal.

While this thesis has been narrowed down to classification tasks for qualitative analysis, the use of neural networks is broad ranging from autonomous cars to automatic trading. The same way economists embraced the development of differential calculus to expand their models; entrepreneurs leveraged the spreading of the internet to create new business models, I expect managers and researchers to incorporate big data analytics into their day-to-day activities to understand better the world around us. However, as demonstrated during the hearing of Mark Zuckerberg, Facebook's CEO, in front of the U.S. Congress about the Cambridge Analytical scandal, even policymakers do not have a sound understanding of the current capabilities of modern techniques despite growing concerns about machines taking over human jobs and big data techniques hijacking democracy. I, therefore, call for a democratisation of programming languages and a sensitisation of machine learning techniques as tools to solve problems, but also about the issues they raise. As a consequence, I hope this work demystified the functioning of neural networks and could be used as a gate by business students, entrepreneurs, managers, and teachers to enter the machine learning world.

References

Agrawal, S. and Awekar, A. Deep Learning for Detecting Cyberbullying Across Multiple Social Media Platforms, 2018. URL https://arxiv.org/pdf/1801.06482.pdf. Retrieved from.

Alcantara, G. Empirical analysis of non-linear activation functions for Deep Neural Networks in classification tasks, 2017. URL https://arxiv.org/pdf/1710.11272.pdf. Retrieved from.

Apté, C., Damerau, F., and Weiss, S. M. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems (TOIS)*, 12(3):233–251, 1994.

Baeza-Yates, R. and Ribeiro-Neto, B. Modern information retrieval. New York, 1999. URL https://doi.org/10.1080/14735789709366603.

Baroni, M., Dinu, G., and Kruszewski, G. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 238–247, 2014.

Becker, S., Le Cun, Y., et al. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37, 1988.

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

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Berger, M. J. Large Scale Multi-label Text Classification with Semantic Word Vectors. Technical Report, 2014. URL https://cs224d.stanford.edu/reports/BergerMark.pdf. Retrieved from.

Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. Enriching Word Vectors with Subword Information, 2016. URL https://doi.org/1511.09249v1.

Bottou, L. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

Bottou, L. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

Cambria, E., Poria, S., Gelbukh, A., and Thelwall, M. Sentiment analysis is a big suitcase. *IEEE Intelligent Systems*, 32(6):74–80, 2017.

Caudill, M. Neural Networks Primer, Part 1. AI Expert (Vol. 2). [CL Publications], 1986. URL https://dl.acm.org/citation.cfm?id=38295. Retrieved from.

Cawley, G. C., Talbot, N. L., and Girolami, M. Sparse multinomial logistic regression via bayesian l1 regularisation. In *Advances in neural information processing systems*, pages 209–216, 2007.

Chen, D., Bolton, J., and Manning, C. D. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858*, 2016.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Cleverdon, C. Optimizing convenient online access to bibliographic databases. *Information services and Use*, 4:37–47, 1984.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

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

Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. Very deep convolutional networks for natural language processing. *arXiv preprint arXiv:1606.01781*, 2, 2016.

Dahl, G. E., Sainath, T. N., and Hinton, G. E. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

Debole, F. and Sebastiani, F. Supervised team weightening for Automated Text Categorization. Istituto Di Scienza E Tecnologie dell'Informazione, (Ml), pp. 784-788, 2003. URL http://nmis.isti.cnr.it/sebastiani/Publications/NEMIS04.pdf. Retrieved from.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

Dhingra, B., Liu, H., Salakhutdinov, R., and Cohen, W. W. A comparative study of word embeddings for reading comprehension. *arXiv preprint arXiv:1703.00993*, 2017.

Ding, Z., Xia, R., Yu, J., Li, X., and Yang, J. Densely connected bidirectional lstm with applications to sentence classification. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 278–287. Springer, 2018.

Dong, J. and Huang, J. Enhance word representation for out-of-vocabulary on ubuntu dialogue corpus. *arXiv preprint arXiv:1802.02614*, 2018.

Dozat, T. Incorporating Nesterov Momentum into Adam. ICLR Workshop (1), 2013–2016, 2016. Retrieved from.

Economist. The world's most valuable resource is no longer oil, but data. May 6th, 2017. URL https://www.economist.com/news/leaders/21721656-data-economy-demands-new-approach-antitrust-rules-worlds-most-valuable-resource. Retrieved from.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Elman, J. L. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., and Vincent, P. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Artificial Intelligence and Statistics*, pages 153–160, 2009.

Faraz, A. An elaboration of text categorization and automatic text classification through mathematical and graphical modelling. *Computer Science & Engineering: An International Journal (CSEIJ)*, 5(2/3):1–11, 2015.

Forman, G. An extensive empirical study of feature selection metrics for text classification. *Journal of machine learning research*, 3(Mar):1289–1305, 2003.

Frakes, W. B. Information retrieval, data structures & algorithms. Prentice Hall, 1992. URL https://users.dcc.uchile.cl/~rbaeza/iradsbook/irbook.html. Retrieved from.

Fuhr, N., Hartmann, S., Lustig, G., Schwantner, M., Tzeras, K., and Knorz, G. Air/x: A rule-based multistage indexing system for large subject fields. In *Intelligent Text and Image Handling-Volume 2*, pages 606–623. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, 1991.

Genuer, R., Poggi, J.-M., and Tuleau-Malot, C. Variable selection using random forests. *Pattern Recognition Letters*, 31(14):2225–2236, 2010.

Georgakopoulos, S. V., Tasoulis, S. K., Vrahatis, A. G., and Plagianakos, V. P. Convolutional neural networks for toxic comment classification. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*, page 35. ACM, 2018.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Goldberg, Y. A Primer on Neural Networks Models for Natural Language Processing 1–76, 2015. URL https://doi.org/10.1613/jair.4992.

Gomez, J. C., Boiy, E., and Moens, M.-F. Highly discriminative statistical features for email classification. *Knowledge and information systems*, 31 (1):23–53, 2012.

Goodfellow, I., Bengio, Y., and Courville, A. Deep Learning. The MIT Press, 2016. URL http://www.deeplearningbook.org/front_matter.pdf. Retrieved from.

Gövert, N., Lalmas, M., and Fuhr, N. A probabilistic description-oriented approach for categorizing web documents. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 475–482. ACM, 1999.

Greene, D. and Cunningham, P. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proceedings of the 23rd international conference on Machine learning*, pages 377–384. ACM, 2006.

Guyon, I. and Elisseeff, A. An introduction to variable and feature selection.

Journal of machine learning research, 3(Mar):1157–1182, 2003.

Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.

Harris, Z. S. Distributional structure. *Papers on Syntax*, pages 3–22, 1981.

Hashimoto, K. and Tsuruoka, Y. Adaptive joint learning of compositional and non-compositional phrase embeddings. *arXiv preprint arXiv:1603.06067*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Hinton, G., McClelland, J. L., and Rumerlhart, D. E. Distributed representations. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, 1, 1986.

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

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

IBM. What is big data?, 2018. URL https://www-01.ibm.com/software/in/data/bigdata/. Retrieved February 14, 2018.

Janocha, K. and Czarnecki, W. M. On loss functions for deep neural networks in classification. *Classification. Schedae Informaticae*, 25:1–10, 2017.

Jashki, M.-A., Makki, M., Bagheri, E., and Ghorbani, A. A. An iterative hybrid filter-wrapper approach to feature selection for document clustering. In *Canadian Conference on Artificial Intelligence*, pages 74–85. Springer, 2009.

John, G. H., Kohavi, R., and Pfleger, K. Irrelevant features and the subset selection problem. In *Machine Learning Proceedings 1994*, pages 121–129. Elsevier, 1994.

Johnson, R. and Zhang, T. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 562–570, 2017.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 655–665, 2014.

Kaur, G. and Kaur, P. ISSN: 2454-132X Impact factor: 4.295 Review on Text Classification by NLP Approaches with Machine Learning and Data Mining Approaches. International Journal of Advance Research , Ideas and Innovations in Technology, 3, 767–771, 2017. Retrieved from www.ijariit.com.

Khan, A., Baharudin, B., Lee, L. H., and Khan, K. A review of machine learning algorithms for text-documents classification. *Journal of advances in information technology*, 1(1):4–20, 2010.

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic gradient descent. *arXiv preprint arXiv:1412.6980*, 2015.

Kira, K. and Rendell, L. A. A practical approach to feature selection. In *Machine Learning Proceedings 1992*, pages 249–256. Elsevier, 1992.

Kiwiel, K. C. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90(1):1–25, 2001.

Kohavi, R. and John, G. H. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, 1997.

Kusner, M., Sun, Y., Kolkin, N., and Weinberger, K. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

Le, H., Pham, Q., Sahoo, D., and Hoi, S. C. Urlnet: Learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162*, 2018.

Lee, J. Y. and Dernoncourt, F. Sequential short-text classification with recurrent and convolutional neural networks. *arXiv preprint arXiv:1603.03827*, 2016.

Lewis, D. D. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–50. ACM, 1992.

Li, X. and Roth, D. Learning question classifiers: the role of semantic information. *Natural Language Engineering*, 12(3):229–249, 2006.

Liu, P., Qiu, X., and Huang, X. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.

Livnat, A., Papadimitriou, C., Pippenger, N., and Feldman, M. W. Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences*, 107(4):1452–1457, 2010.

Ma, Y., Peng, H., Khan, T., Cambria, E., and Hussain, A. Sentic lstm: a hybrid network for targeted aspect-based sentiment analysis. *Cognitive Computation*, 10(4):639–650, 2018.

Mikolov, T., Kopecky, J., Burget, L., Glembek, O., et al. Neural network based language models for highly inflective languages. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4725–4728. IEEE, 2009.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, pages 1045–1048, 2010.

Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J., and Khudanpur, S. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

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

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*, 2017.

Murty, M. R., Murthy, J., and PVGD, P. R. Text document classification based-on least square support vector machines with singular value decomposition. *International Journal of Computer Applications*, 27(7):21–26, 2011.

Ng, A. Y. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

Nikam, S. S. A comparative study of classification techniques in data mining algorithms. *Oriental journal of computer science & technology*, 8(1):13–19, 2015.

Olah, C. Understanding LSTM Networks – colah's blog, 2017. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Retrieved April 14, 2018.

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

Penrose, R. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge University Press, 1955.

Porter, M. F. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

Rahimian, V. and Ramsin, R. Designing an agile methodology for mobile software development: A hybrid method engineering approach. In *2008 Second International Conference on Research Challenges in Information Science*, pages 337–342. IEEE, 2008.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Ranganathan, V. and Natarajan, S. A new backpropagation algorithm without gradient descent. *arXiv preprint arXiv:1802.00027*, 2018.

Robertson, S. E. and Walker, S. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94*, pages 232–241. Springer, 1994.

Rong, X. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

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

Saeys, Y., Abeel, T., and Van de Peer, Y. Robust feature selection using ensemble feature selection techniques. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 313–325. Springer, 2008.

Sag, I. A., Baldwin, T., Bond, F., Copestake, A., and Flickinger, D. Multiword expressions: A pain in the neck for nlp. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 1–15. Springer, 2002.

Salinca, A. Convolutional neural networks for sentiment classification on

business reviews. *arXiv preprint arXiv:1710.05978*, 2017.

Salton, G., Wong, A., and Yang, C.-S. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

Sánchez, D., Martín-Bautista, M. J., Blanco, I., and de la Torre, C. J. Text knowledge mining: an alternative to text data mining. In *2008 IEEE International Conference on Data Mining Workshops*, pages 664–672. IEEE, 2008.

Sebastiani, F. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

Shen, Y., Huang, P.-S., Gao, J., and Chen, W. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1047–1055. ACM, 2017.

Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

Song, F., Liu, S., and Yang, J. A comparative study on text representation schemes in text categorization. *Pattern analysis and applications*, 8(1-2): 199–209, 2005.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Sundermeyer, M., Alkhouli, T., Wuebker, J., and Ney, H. Translation modeling with bidirectional recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 14–25, 2014.

Sundström, J. Sentiment analysis of swedish reviews and transfer learning using convolutional neural networks, 2017. URL http://uu.diva-portal.org/smash/get/diva2:1174477/FULLTEXT01.pdf. Retrieved from.

Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28(1139-1147):5, 2013.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 2012.

Tsvetkov, Y., Faruqui, M., Ling, W., Lample, G., and Dyer, C. Evaluation of word vector representations by subspace alignment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2049–2054, 2015.

Turney, P. D. and Pantel, P. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188, 2010.

Tzeras, K. and Hartmann, S. Automatic indexing based on bayesian inference networks. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 22–35. ACM, 1993.

Upadhyay, S., Chang, K.-W., Taddy, M., Kalai, A., and Zou, J. Beyond bilingual: Multi-sense word embeddings using multilingual context. *arXiv preprint arXiv:1706.08160*, 2017.

Wang, B., Wang, L., Wei, Q., and Liu, L. Textzoo, a new benchmark for reconsidering text classification. *arXiv preprint arXiv:1802.03656*, 2018.

Warde-Farley, D., Goodfellow, I. J., Courville, A., and Bengio, Y. An empirical analysis of dropout in piecewise linear networks. *arXiv preprint arXiv:1312.6197*, 2013.

Weston, J., Bengio, S., and Usunier, N. Scaling up to large vocabulary image annotation. In *Proceedings of the 22st International Joint Conference on Artificial Intelligence*, pages 2764–2770, 2010.

Wolfram, D. and Zhang, J. The influence of indexing practices and weighting algorithms on document spaces. *Journal of the American Society for Information Science and Technology*, 59(1):3–11, 2008.

Wolpert, D. H. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

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

Yin, W., Kann, K., Yu, M., and Schütze, H. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

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

Yu, K., Liu, Y., Schwing, A. G., and Peng, J. Fast and accurate text classification: Skimming, rereading and early stopping. 2018.

Yu, M. and Dredze, M. Learning composition models for phrase embeddings. *Transactions of the Association for Computational Linguistics*, 3:227–242, 2015.

Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zhang, X., Zhao, J., and LeCun, Y. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

Zhou, C., Sun, C., Liu, Z., and Lau, F. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015a.

Zhou, C., Sun, C., Liu, Z., and Lau, F. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015b.

Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., and Xu, B. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. *arXiv preprint arXiv:1611.06639*, 2016a.

Zhou, Z., Zhu, X., He, Z., and Qu, Y. Question classification based on hybrid neural networks. In *2016 4th International Conference on Electrical & Electronics Engineering and Computer Science (ICEEECS 2016)*. Atlantis Press, 2016b.