REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SEMANTIC ANALYSIS USING NATURAL LANGUAGE PROCESSING METHODS

NABEEL ZUHAIR TAWFEEQ ABDULNABI

M.Sc. THESIS
DEPARTMENT OF COMPUTER ENGINEERING
PROGRAM OF COMPUTER ENGINEERING

ADVISER
ASSIST. PROF. DR. OĞUZ ALTUN

ISTANBUL, 2016

REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SEMANTIC ANALYSIS USING NATURAL LANGUAGE PROCESSING METHODS

A thesis submitted by NABEEL ZUHAIR TAEFEEQ ABDULNABI in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** is approved by the committee on 29.11.2016 in in Department of Computer Engineering, Computer Engineering Program.

Thesis Adviser

Assist. Prof. Dr. Oğuz Altun

Yıldız Technical University

Approved By the Examining Committee

Assist. Prof. Dr. Oğuz Altun

Yildiz Technical University _____

Assist. Prof. Dr. Mehmet Aktaş, Member

Yildiz Technical University _____

Assist. Prof. Dr. Cemal Okan Şakar, Member

Bahcesehir University _____

ACKNOWLEDGEMENTS

I would like to thank God for the kindness, which bestowed on us of health, science, a great family, and good friends in my life.

I would like to thank my thesis advisor Assist. Prof. Dr. Oğuz Altun of Computer Engineering Department at Yildiz Technical University for his guidance which is helping me to complet this research. The door to Assist. Prof. Altun office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this thesis to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to thank my parents and to my wife for providing me with permanent support and continuous encouragement throughout the years of my study. This work is not complete without your support me.

2016

NABEEL ZUHAIR TAEFEEQ ABDULNABI

TABLE OF CONTENTS

LIST OF SYMBOLS

b	Bias of network input
C	Cost Function
D	Dimension of Matrix
E	Error of the output of the network
h	Hidden Layer
p	Probability
R	Real Number
S	The length of Sentence
W	Weight Matrix

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
CR	Customer Review
DCNN	Dynamic Convolutional Neural Network
IMBD	Internet Movie Database
MLP	Multilayer Perceptron
MR	Movie Review
RCV1	Reuters Corpus Volume I
ReLU	Rectifier Linear Unit
SST	Stanford Sentiment Treebank
TREC	Text Retrieval Conference

LIST OF FIGURES

Page

LIST OF TABLES

Page

ABSTRACT

SEMANTIC ANALYSIS USING NATURAL LANGUAGE PROCESSING METHODS

NABEEL ZUHAIR TAWFEEQ ABDULNABI

Department of Computer Engineering

M.Sc. Thesis

Adviser: Assist. Prof. Dr. Oğuz Altun

Sentence classification for shortened texts such as single sentences of a movie reviews typically presents problems due to the limited amount of information that they contain. In response, we developed a multilayer convolutional neural network (CNN) architecture and better hyperparameter values for sentence classification learning.

Among CNN's layers the input layer consisting of concatenated word embeddings, followed by convolution layer with different filter sizes for learning sentence-level features, and a max-pooling layer that concatenates features to form a final feature vector.

In a final fully connected layer, a softmax classifier predicts the class. We allow the network to handle an arbtirary batch size with different dropout ratios in order to aptly regularize the CNN to block neurons from co-adapting, and to impose those neurons in order to learn useful features. By using the CNN with multiple filter sizes, we can detect specific features, including negations such as “not amazing”. We furthermore use a word2vec [1] method for the learning of word representation distribution, which we show can easily capture beneficial syntactic and semantic regularities.

Keywords: Convolutional Neural Network; Sentence Classification; Word Embedding

ÖZET

DOĞAL DİL İŞLEME YÖNTEMLERİ İLE SEMANTİK ANALİZİ

NABEEL ZUHAİR TAWFEEQ ABDULNABİ

Bilgisayar Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi

Tez Danışmanı: Yrd. Doç. Dr. Oğuz Altun

Film eleştirileri üzerinden yapılan cümle sınıflandırma işlemleri cümlelerin kısalığı yüzünden çok zor yapılmaktadır.

Bu tezde, küçük boyuttaki verilerin üzerinde çalışacak bir Konvolüsyonal Neural Network (CNN) daha iyi bir sınıflandırma yapılması amacıyla anlatılmıştır. Bu çalışmada sunulan CNN, bir çok katmandan meydana gelmektedir. Birinci katmanda cümleler ikili tabana dönüştürülmüştür. Sonraki katmanda, konvolüsyonel yapay sinir ağı sayesinde cümlelerin seviye özellikleri çıkarılmıştır.Bu aşamadan alınan sonuçlar ikinci katmanda max-pooling yöntemiyle bir matris üzerinde birleştirilmiştir. Son aşamada ise softmax sınıflandırıcısı kullanılmıştır. Bu çalışmada veriyi işlemek için farklı boyutlarda parti (batch) ve birakma (dropout) oranları kullanılmıştır. Bu sayesde CNN ve yapay sinir ağı blokları cümlelerin özelliklerinin öğrenilmesinde etkin bir hale getirilmiştir. CNN'in çoklu filtrelerle kullanılması sayesinde çok kısa ve anlaşılması güç cümlelerin sınıflandırılması mümkün olmuştur..

Anahtar Kelimeler: Konvansiyonel sinir ağı, Cümle sınıflandırması, Kelime gömme

CHAPTER 1

INTRODUCTION

1.1 Literature Review

Convolutional Neural Networks (CNN) are considered to be responsible for a major breakthrough in sentence classification. Recently, researchers started to use CNNs in Natural Language Processing (NLP) with promising results, especially in sentence classification [2]. CNNs use layers with convolving filters that are applied to local regions. Furthermore, in CNNs, each filter is replicated a long the entire local field . These replicated regions share the same weight vector and create a feature map. This weight sharing enables researchers to use valuable features regardless of their location, while preserving their locations when useful features show up. CNNs are cosidered to be quite powerful in terms of sentence classification, because they used as the better approach of weighting individual words of a fixed size to produce a useful feature for a particular task.

Recently, Kim. [3] reported several experiments on sentence classification using CNNs. A simple CNNs is trained on one layer of convolution on top of word vectors followed by a max-pooling, and, finally, in a fully connected layer as shown in Figure 1.1. For regularization Kim used dropout to prevent over-fitting and co-adaption of the hidden layers. This model has two channels of word vectors the first channel is kept static throughout training, and the second channel is fine-tuned via backpropagation. The datasets used are movie reviews [34] with two labels (positive and negative). They also used a customer reviews dataset [39], which has either positive or negative labels, and the Stanford Sentiment Treebank dataset [4], which has very positive, positive, natural, negative, and very negative movie reviews The researchers discovered that a little tuning of hyperparameters can achieve remarkable results. In addition, they also implement a little change to the architecture to allow for the use of both pre-trained (a

model pre-trained from word2vec) and task-specific vectors (a model pre-trained from word2vec and fine-tuned for each task) by having different channels. However, a CNN with static vectors achieves excellent results, comparable to advanced deep-learning models.

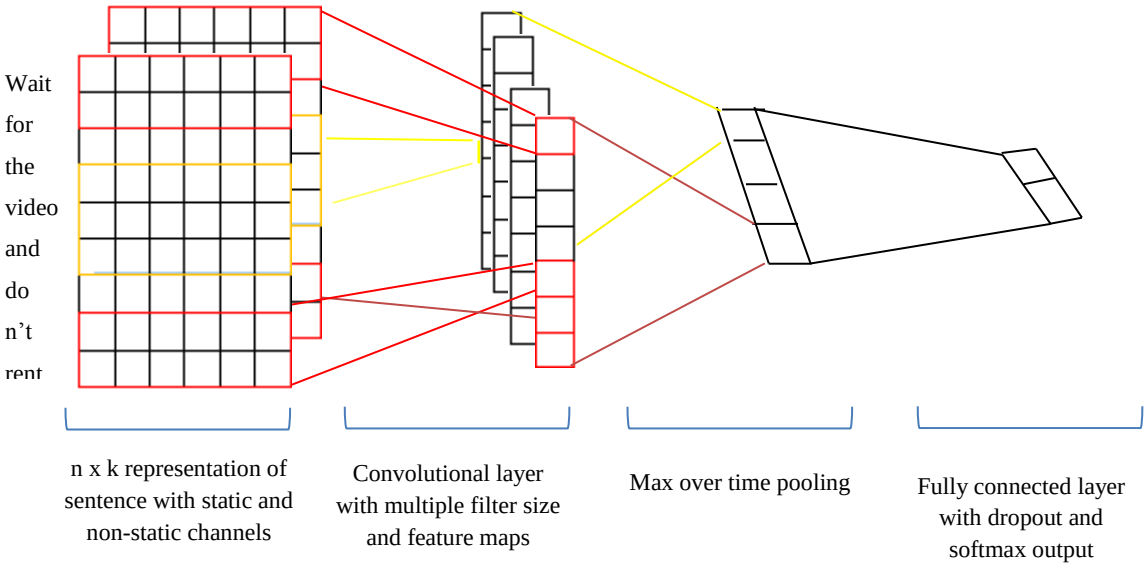


Figure 1.1 Convolutional neural networks for sentence classification [3]

Zhang et al. [5] presented a study on a character-level CNN (ConvNet) for text classification. The model consists of two large and small convNets. Both have nine deep layers followed by six convolutional layers. Finally, a fully connected layer is used for classification as depicted in Figure 1.2. This model is tested on several large –scale datasets (Amazon Review full [40], Amazon review polarity, yahoo! Answers and Sogou News [41]). for regularization, two dropout models are inserted between the three fully connected layers. The dropout probability is set to 0.5. In comparisons between ConvNet and traditional methods such as a bag of words and n-gram, the researchers found that a character-level of ConvNet proved its superiority in many aspects.

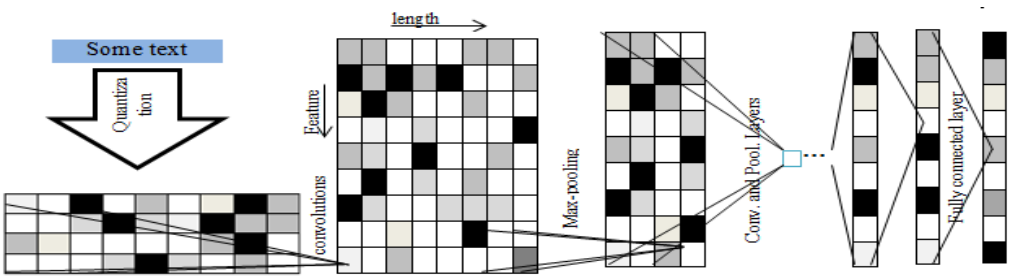


Figure 1.2 CNN model for character level classification [5]

Wang et al. [6] prposed model for short text classification. Extra knowledge is detected through pre-trained word embedding and multi-scale Semantic units (SUs defined as n-grams that provide the dominant meaning of text) in short texts. External knowledge is introduced using observation of semantic cliques. Then, combinations of these significant parts are fed to the convolution layer, and max pooling is used to detect the most important feature. In the final layer a fully connected softmax classifier is used to estimate the probability distribution of classes, as shown in Figure 1.3.

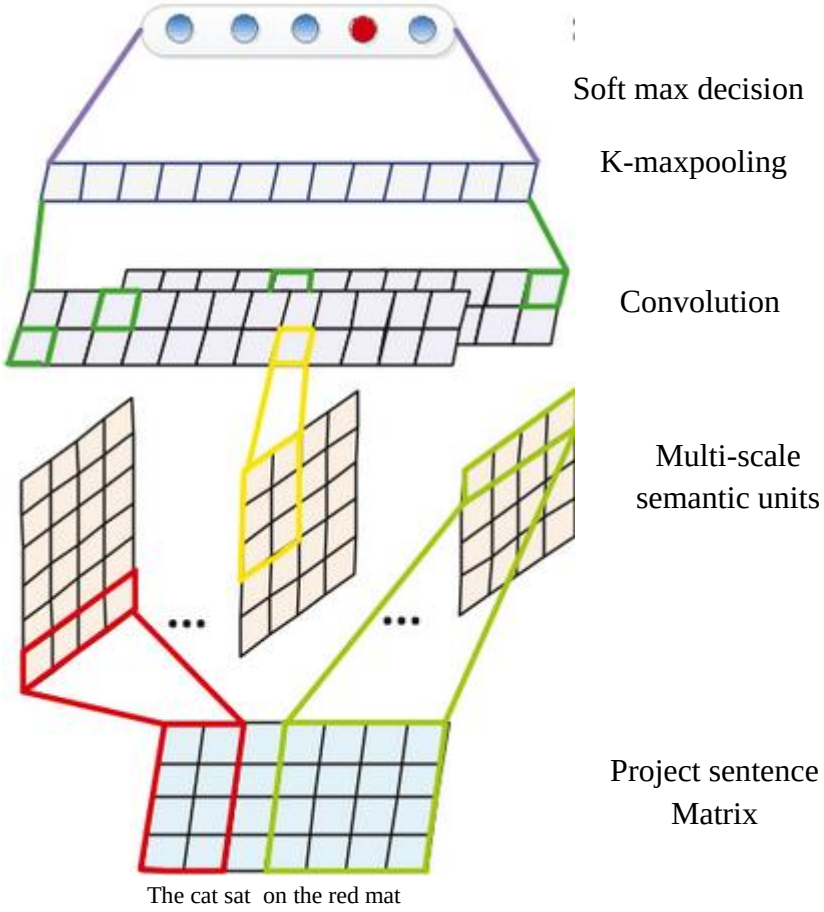


Figure 1.3 CCN Architecture for short text categoraization [6]

The experiment is applied to two datasets (Google Snippets [42] and TREC [43]). Google Snippets has eight categories, and TREC has six type of questions.

Zeng et al. [7] introduced the use of CNNs for question classification. Lexical and sentence-level features are extracted via a deep CNN and word embedding [44]. According to the given nouns the lexical level features are extracted. First, a convolution layers used to learn sentence level features. Then lexical and sentence level features are passed into softmax neural networks to estimate the relationship between a

noun in a sentence [45]. As depicted in Figure 1.4. SemEval-2010 Task 8 dataset [46] is used to evaluate this model.

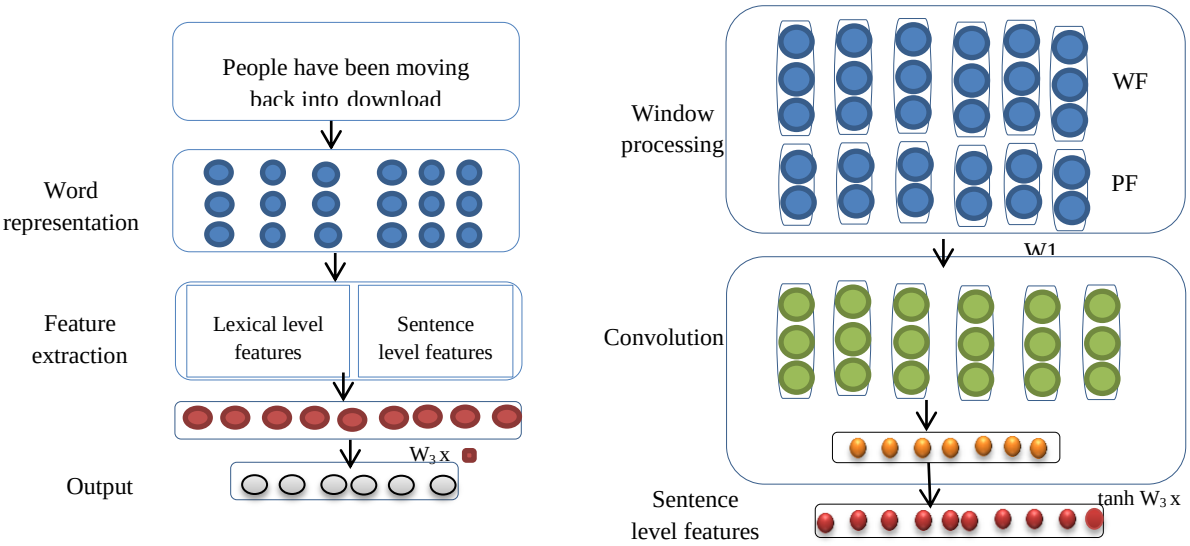


Figure 1.4 Left: Relation classification via neural network architecture, Right:Show the framework utilized for sentence level features extraction [7].

Mass et al. [8] attempted to introduce a model to capture both semantic and sentiment similarities between words. The semantic component of their model learns word vectors via an unsupervised probabilistic model of documents.

Hu et al. [9] have introduced a deep convolutional model for sentences matching The architecture of model consists of 1 dimensional convolution and followed by max-pooling. Then followed by 2 dimensional convolutions and pooling layers followed by fully connected layers as depicted in Figure 1.5. A convolutional model ARC-I and ARC-II, are proposed for matching two sentences. A low level representation of the 2 sentences is acquired after the first convolution. The proposed model is evaluated on matching response and sentence completion to Weibo, MSRP dataset [47]. Researchers report that this method outperforms other competitors on multiple matching approaches.

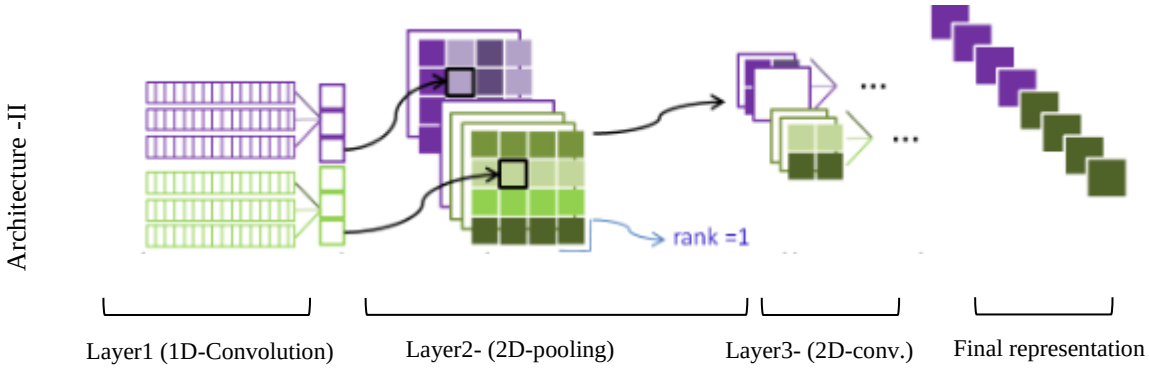


Figure 1.5 CNN for sentences matching [9]

Different machine learning algorithms are presented (support vector machines [48], Naive Bayes [49], and maximum entropy classification [50]) for document classification [10] to try to find out the polarity of document(e.g., whether it is positive or negative). The author is experimenting with the impact of applying a machine-learning mechanism to the sentence-classification problem. The lack of a keyword in the sentence that gives a meaning of negative or positive is the most challenging aspect of detecting review sentiment polarity. For instance, the sentence (”how could anyone sit through this movie”), it does not include any word that indicates negative.

A dynamic convolutional neural network (DCNN) for semantic modeling of sentences is proposed by Kalchbrenner et al. [11]. Multiple convolution layers and dynamic pooling operation is applied to create a feature maps. This is followed by k-max-pooling layer. Finally, a fully conntected layer is used for prediction as shown in Figure 1.6. The author tested the model in four experiments: a six-way question classification in the TREC dataset [51]; SST–1, SST–2, 6–type question categorization; and Twitter sentiment. This model can deal with input sequences of different lengths. DCNN performs better on SST–1 and TREC, but it gets weak results on SST–2.

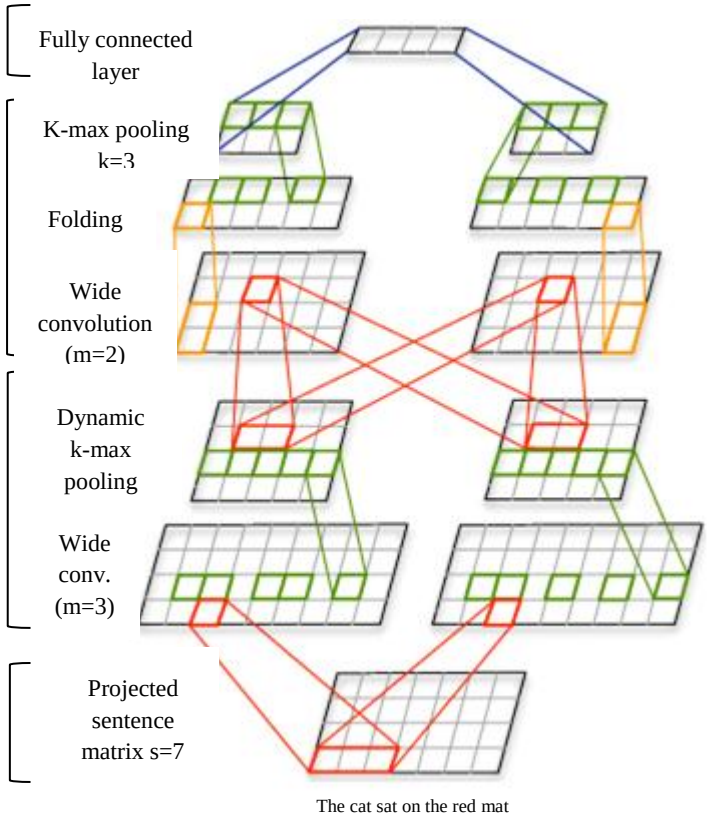


Figure 1.6 DCNN for semantic modeling of sentence [11]

Pang et al. 2002 [52] used a different machine-learning algorithm (support vector machine SVM, naïve Bayes and maximum entropy classification) for document classification. The authors used movie review from Rotten Rotten Tomatoes to test their experiment. The results introduced from by the machine learning methods are good in comparison to human baselines. The naïve Bayes method shows week results in term of performance, whereas SVM shows a good results.

LeCun et al. [12] introduced one-dimensional structure (word order) of the document in which each part of the convolution layer deals with a specific part or region of the document (sequence of words). Johnson et al. [13] experimented with two kinds of CNNs: bow-CNN and seq-CNN. Both methods outperform a baseline approach on all datasets they used. Also the author found that seq-CNN gets better performance than bow-CNN in sentiment analysis. On the other hand, bow-CNN outperforms seq-CNN in topic classification. The proposed model consists of parallel CNN, which is two convolutional layers followed by pooling layer and output layers as shown in Figure 1.7. The experiment is applied to IMDB: movie reviews [8] for sentiment classification; Elec: electronics product reviews for Amazon review dataset; and RCV1 [40]: topic categorization, which is a corpus of Reuters news articles datasets [53].

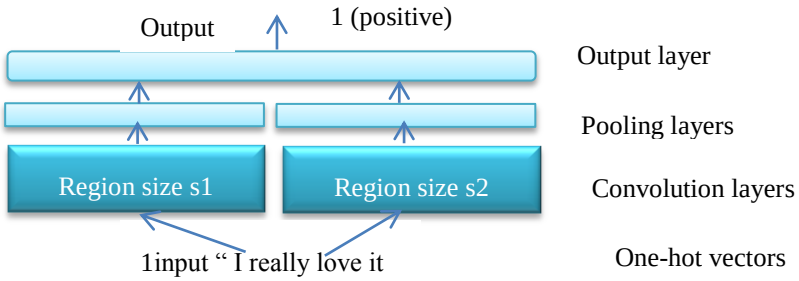


Figure 1.7 Parallel CNN

In most of the CNNs that deal with text classification, the input layer is the vector of the transformed word of the sentence that is either trained by the CNN or via another approach [14]. In this thesis, we use a model similar to [3], which uses a CNN for sentence classification. The first layer is used for embedding words into a low-dimensional vector. The second layer is the convolution layer, which convolves embedding vectors by kernels of different sizes. After that, a max pool is applied to the output of the convolutional layer to extract the feature maps. This feature maps are stacked into a long feature vector and fed to the fully connected layer. A dropout applied to the fully connect layers for regularization. Finally a sotfmax is used to classify the result.

1.2 Objective of the Thesis

There is a huge amount of information on the internet, including opinion-rich resources such as the review sites and blog posts. In this thesis our aim is to develop a system for binary classification of sentences in which the sentence is classified as either positive or negative. In addition, we identify empirically the setting of hyperparameters and find a reasonable range for each hyperparameter. In this research we present the results of a number of tests of various configurations of CNNs applied to sentence classification datasets. Once you understand how the customer feels after analyzing their comments or reviews you can identify what they like and disliked and build things like machine-learning systems or more targeted marketing campaigns for them. For instance, in movie reviews this system can identify whether a movie review is positive or negative based on the text of the review [15]. A fundamental technique in different opinion-mining and sentiment-analysis applications is classification [16]. The importance of classification comes from the observation that many problems of interest can be worked out as implementation of a classification to given text document.

1.3 Hypothesis

Automatic sentence classification focuses on making the machine able to understand text in terms of different aspects (polarity positive/negative, objectivity, etc.). In natural-language processing, sentence classification is the central problem. Recently, multiple methods have been applied to deal with sentence classification. Convolutional Neural Networks (CNNs) are good for sentence classification. We demonstrate how the simple tuning in hyper-parameters of CNNs can achieve remarkable results. Two datasets are used in this thesis. The first one is a movie review data from Rotten Tomatoes. It consists of 10,660 sample review sentences. The vocabulary size is around 20k. The second dataset used is a Stanford sentiment Treebank, which is a widely used dataset for evaluating sentence models and classification. It consists of 11,855 movie reviews, each of which is given one of 5 labels (strongly positive, positive, neutral, negative, and strongly negative).

CHAPTER 2

GENERAL INFORMATION

A CNN is comparable to a traditional neural network in that it consists of neurons that have weights and biases. It can learn these parameters during the training stage of the network. Each neuron is fed some inputs, followed by a linear operation in which the sum of the products of the weights and the inputs is calculated. This is followed by a nonlinear activation function likes (e.g., ReLU, tanh, etc., which is explained in section 2.6). Finally, a fully connected layer is used to predict the class. What distinguishes a CNN is that its network architectures can extract patterns found within local regions of the input layers. Working with local regions helps to speed up and minimize the number of parameters in the network [17].

2.1 Background of Neural Networks

It is worth a simple review of the neural network before diving into details of convolutional neural network.

2.1.1 Perceptron

Perceptron is a simple linear classifier. It considers a supervised learning algorithm. Perceptron consists of two parts [18], as a shown in Figure 2.1. The first part calculates of the weighted sum:

$$\sum_{i=0}^{n} x_i \, . w_i + b \qquad (2.1)$$

Where x is an input vector $x \in \begin{bmatrix} x1 \\ x2 \\ \vdots \\ x_n \end{bmatrix}$, and the second part is activation function which is 1 if the summation is greater than 0 and zero otherwise:

$$f(x) = \begin{cases} 1 \ if \ w.x + b > 0 \\ 0 \ otherwise \end{cases} \quad (2.2)$$

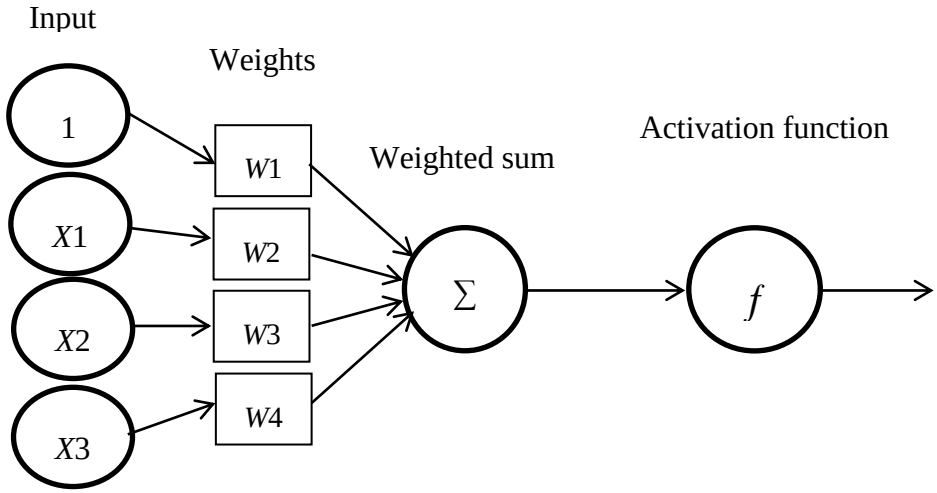


Figure 2.1 Perceptron model

So, if we have a binary classification problem (0, 1), then, perceptron can make a prediction of the sign label of each class. The function $f(x)$ is utilized to classify *x* as zero or one or as positive or a negative.

In two-dimensional spaces the decision boundary is a line that is can be shown by the equation (2.3) as shown in Figure 2.2.

$$w^T x + w_o = 0 \quad (2.3)$$

Our aim is to separate the negative and positive samples. So that the entire positive samples are put in a side of the decision boundary and the negative samples are put in the other side. And learning means we have chosen weights and w_o in a way that this line is in the correct position (i.e., split the positive and negative samples). Learning perceptron means find the optimal values for these weights. However, we usually define an objective function, and we assume a family of linear discriminant function. Therefore, we have to search for best line that separates our data samples. In order to do that we have to define the cost function and error and to try to minimize that error. In Figure 2.2 for instance, if we miss classifying one of the positive samples this would be an error and we have to minimize it.

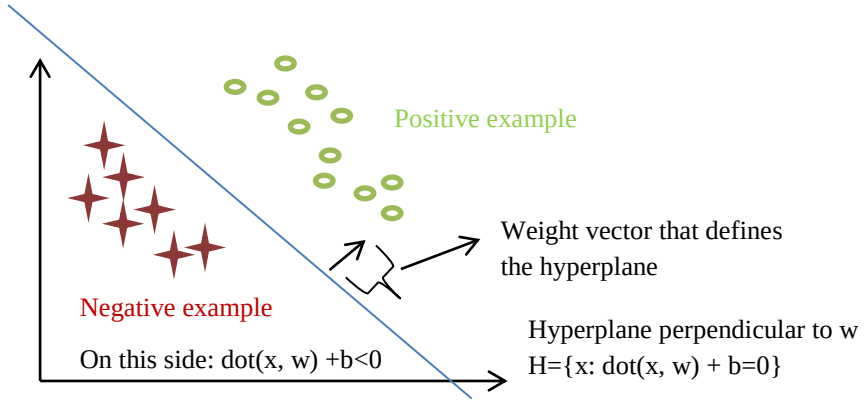


Figure 2.2 Illustrate decision boundary

Let's look at the equation of this line, for any two points x_1 *and* x_2:

$$w^T x_1 + w_o = w^T x_2 + w_o = 0 \quad (2.4)$$

$$w^T (x_1 - x_2) = 0 \quad (2.5)$$

Where w^T is vector and $(x_1 - x_2)$ is also a vector. If the product of two vectors is zero, this means they are orthogonal. It means that w^T is orthogonal to the line$(x_1 - x_2)$. And for any x_o:

$$w^T x_o + w_0 = 0 \quad (2.6)$$

$$w_o = -w^T x_o \quad (2.7)$$

Suppose we have a point x and would like to calculate the distance between this point and the decision boundary. As we say, w^T is orthogonal to the decision boundary line, however, if we have any point on this line x_o, then the distance is computed as in equation (2.8):

$$w^T (x - x_o) \quad (2.8)$$

$$w^T x - w^T x_o \quad (2.9)$$

And we know that $w_o = -w^T x_o$, then:

$$w^T x + w_o \quad (2.10)$$

So, to find a distance between the point *x* and the decision boundary simply, we put this point in equation (2.10). And it will be positive or negative. So, the reason we compute the distance between the point and hyperplane or decision boundary is to define the cost function based on this distance. In Figure 2.2, for example, if there is some point that is misclassified we can define a cost function based on distance of this point that misclassified from the decision boundary as in equation (2.11):

$$Err = \sum_{i \in m} -y_i (w^T x_i + w_o) \quad (2.11)$$

Where, *m* is the set of all misclassified points. So having an error function, it is easy to apply gradient descent. It is just taking a derivative of the error function, and when we take a derivative, it gives us one step toward a direction.

The derivative of the error function is:

$$\frac{dErr}{dw} = \sum_{i \in m} -y_i \, x_i \tag{2.12}$$

The derivative with respect to w_o is

$$\frac{dErr}{dw_o} = \sum_{i \in m} -y_i \tag{2.13}$$

Using the gradient descent, we update our parameters as in equation (2.14):

$$w_{new} = w_{old} - \eta \frac{dErr}{dw} \tag{2.14}$$

Where η is the learning rate, which is the length of the step toward the global minimum.

2.1.2 Multilayer Perceptron(MLP)

As we know from the definition of the perceptron, it is a linear model *a*. But it is insufficient in dealing with a problem that is not linearly separable, like a two-dimensional XOR problem. Therefore, in order to solve more complicated problems we need to add more weights, because that learning in neural networks is occurring throughout the weights [19]. We can insert neurons between the input layer and the output layers, which are called hidden layers, as shown in Figure 2.3

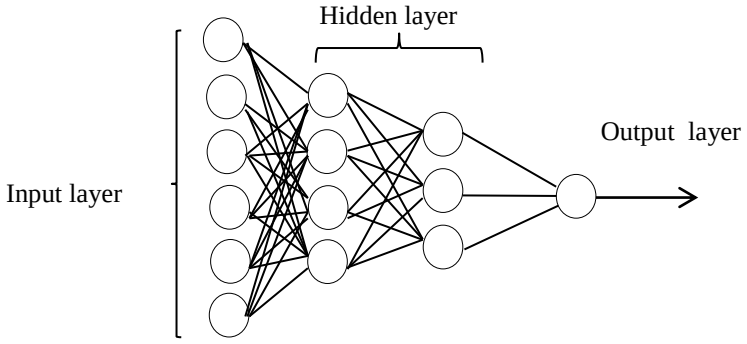


Figure 2.3 Multilayer perceptron network

MLP can be abbreviated by two basic operations. The first one is mapping a set of input samples of the current weight to outputs. And the second one is to update those weights with respect to the loss function, which is the difference between what our network predicts and the target outputs. These are known as feed-forward neural networks [20].

Moreover, what makes MLP perceptron distinct from others is that it uses a nonlinear activation function.

2.1.3 Back-Propagation

Backpropagation is the main algorithm used to train a model in feed-forward neural networks. More specifically, back-propagation is the algorithm used to train the weight in feed-forward neural networks and to find the optimal weight values. The aim of backpropagation is to optimize the weights in a way that networks can learn how to maps the input data points to the output accurately [21].

Usually, in backpropagation there are two stages:

- Forward pass
- Backward pass

In the forward pass, the input layer with the initial value of the weights and biases feed to the neural networks. For each hidden layer neuron for example, h1, as illustrated in Figure 2.5, we compute weighted sum:

$$h1 = w1 * i1 + w2 * i2 + b1 * 1 \quad (2.15)$$

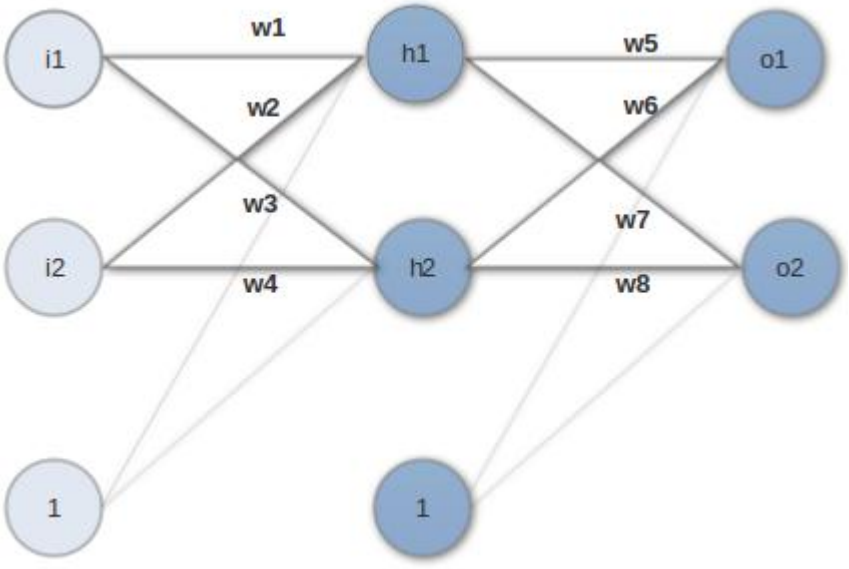


Figure 2.4 Initiate the inputs, weights and biases for feedforward [38]

Then the output will pass to the logistic function:

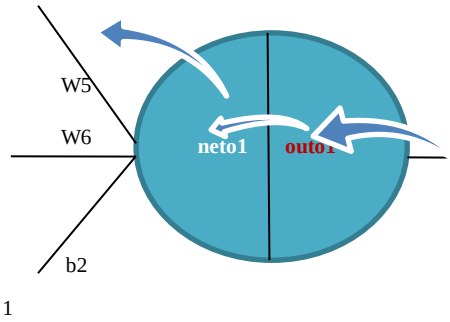
$$h1 = \frac{1}{1+e^{-h1}} \quad (2.16)$$

We repeat this process for the output layer neurons, taking the output from the previous layer (hidden layer) as inputs. After all, we have to compute the total error using the formula:

$$E_{total} = \sum \frac{1}{2}(target - output)^2 \quad (2.17)$$

In the backward pass stage, our aim is to update the weights in a way that the output of the neural networks is very close the target output. Therefore, we have to minimize the error of the output neural networks. In Figure 2.5, for example, in order to know the effect of change w5 on the total error, we have to take the derivative of the total error with respect to w5, $\frac{dE_{totla}}{dw_5}$, and by using the chain rule, we compute this derivative as follows:

$$\frac{dE_{total}}{dw_5} = \frac{dE_{total}}{douto1} * \frac{dout_{o1}}{dnet_{o1}} * \frac{dnet_{o1}}{dw_5} \quad (2.18)$$



$$Eo1 = {}^1/_2\,(targeto1 - outo1)^2$$

$$Etotal = Eo1 + Eo2$$

Figure 2.5 Applying chain rule in backward stage

As depicted in Figure 2.6 for the hidden layer, we complete backward stage by updating the value of weights(w1, w2, w3, and w4). So, w1 can be computed as in equation (2.19):

$$w_1 = w_1 - \eta * \frac{dE_{total}}{dw_1} \quad (2.19)$$

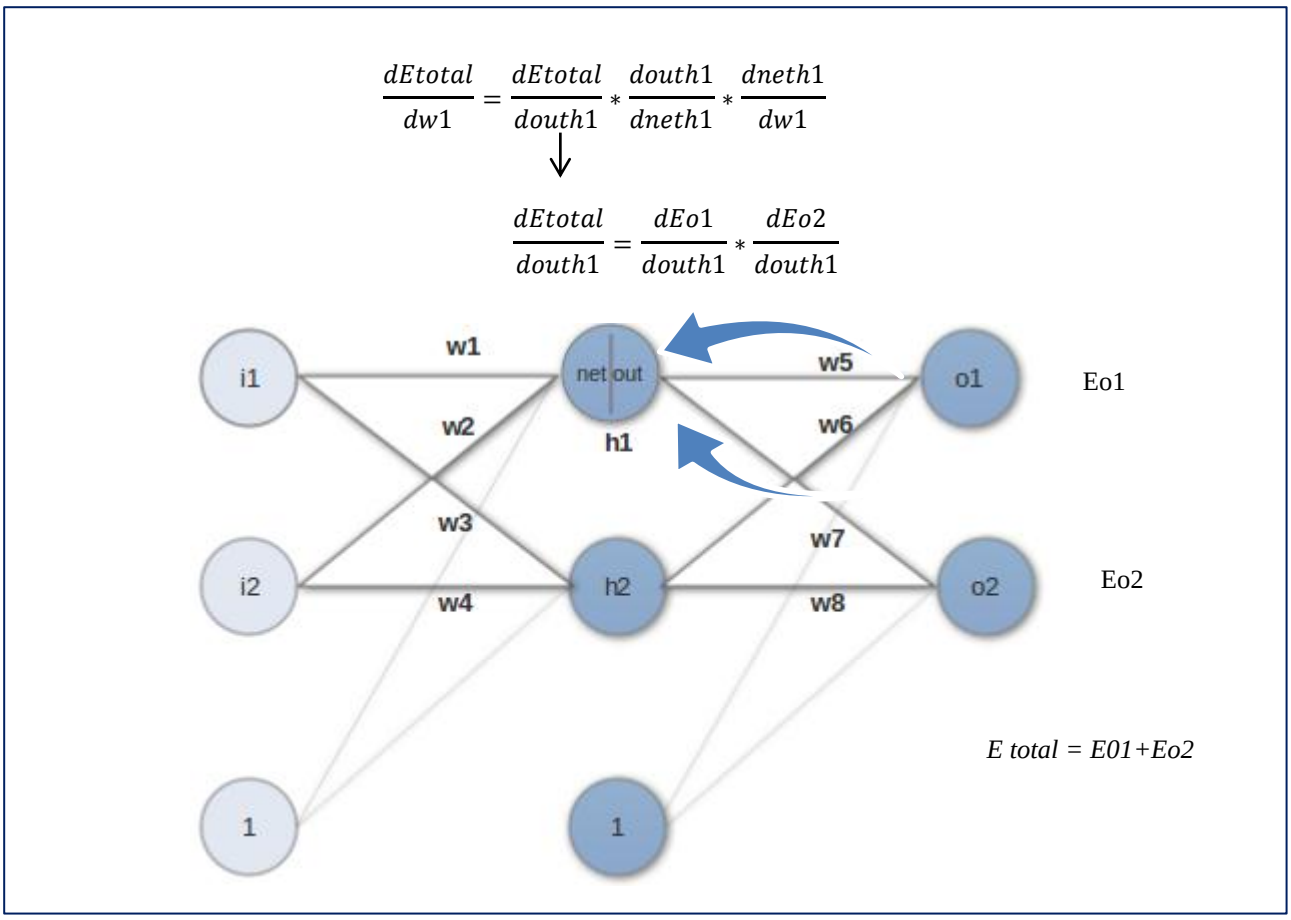


Figure 2.6 Hidden layers backward stage [38]

2.2 Supervised Learning

In machine learning there are different types of learning algorithms. Supervised learning is the task of deducing a function from labeled training data (see Figure 2.7). In supervised learning, the label of the training sample is there [22]. Let us consider we have a training dataset $S = \{x_i , y_i\}_{i=1}^{n}$ where n is the number of examples in our data, x_i is the vector of input data or features, and y_i is the label vector corresponding to each feature in the input vector space. However, our aim in supervised learning is to produce a function that can map $f \epsilon H: X \rightarrow Y$ in a way that the algorithm can classify or find a class label for unseen sample. So, we have to reduce the difference between what our algorithm predicted $f(x_i)$ and the actual label y_i. Therefore, the supervised learning target is to minimize the cost function:

$$C = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2 \qquad (2.19)$$

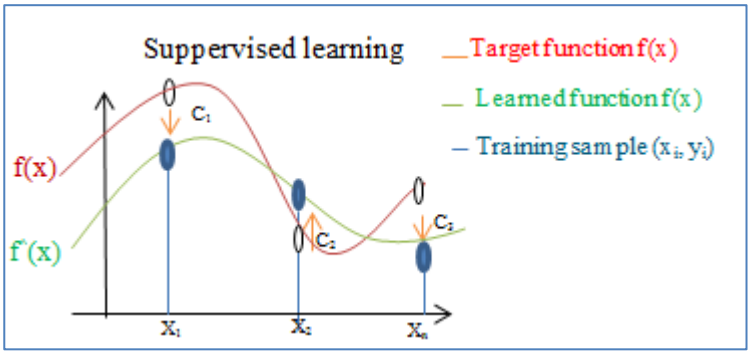


Figure 2.7 Supervised learning

Supervised learning can be divided into two categories, based on the kind of label that has been applied:

1. Classification if the label is discrete value.
2. Regression when the label is a real value.

2.2.1 Linear Classification

The classification can be explained as taking input vectors and deciding or predicting to which class it belongs(see Figure 2.8).

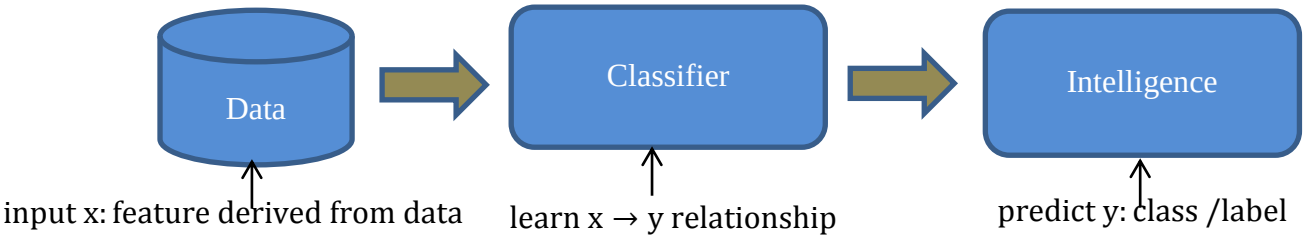


Figure 2.8 Linear classification

The important thing in classification is that it considers each element discretely. Each sample belongs to one class. However, we are dealing with a binary classification y= {0, 1}. A linear classification can be described by the following forms:

$$f(x) = w^T x + b \quad (2.21)$$

Where w ϵR^d is defined as the weight vector and b is the bias. These two parameters will be learned through the training process, so we can tune these parameters in order to minimize the cost function.

2.2.2 Regression

We can define *regression* as a predicting or estimating the relationships between a dependent variable and an independent variable. More precisely, regression can give us an explanation of how tuning or varying independent variables can impact dependent variables. There are different types of regression and here we focus on logistic regression.

2.2.3 Logistic Regression

Logistic regression is utilized to compute the probability of a binary response. It is mostly used with classification problems. In order to predict the probability between independent variable and dependent variable it uses a logistic function as in (2.22):

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.22}$$

In logistic regression there are always two things that must be satisfied in order to produce a reasonable forecasts:

- The probability should be always positive (p>=0)
- It must be less than 1(p<=1) as depicted in Figure 2.9.

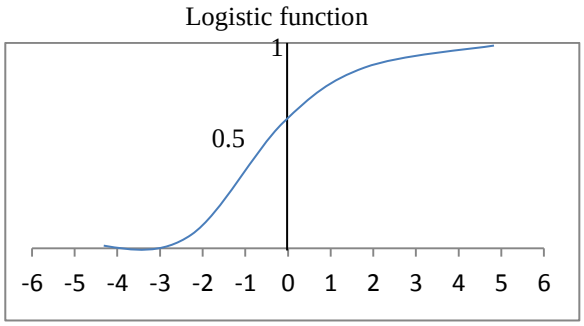


Figure 2.9 Logistic function

2.3 Convolutional Neural Network

CNN is one of the most successful architectures in deep learning [23]. CNN can capture a local features via using a set of learnable filters, which have a small receptive field, but convolved across the whole depth of the input volume [12]. CNN is basically a neural network that utilizes convolution in place of general matrix multiplication in at least one of the layers. A convolution layer has three stages: Convolution stage;

detection stage, which is applying nonlinear activation function in it as application; and pooling stage, as shown in Figure 2.10

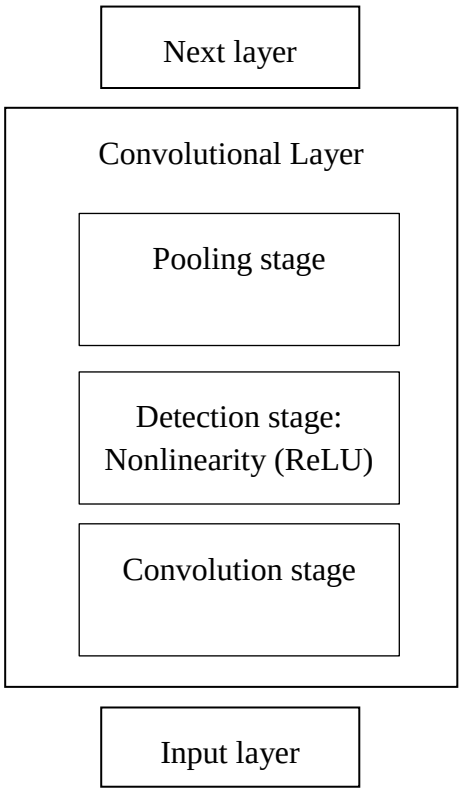


Figure 2.10 Convolutional stages

In the convolution stage, multiple convolution processes are applied in parallel to find pre-activations. In the detection stage a nonlinear activation function is applied for each pre-activation. In the pooling stage, which is a kind of down sampling method, the non-liner max pooling is used to select the maximum value from each sub region of the feature maps. So, the pooling stage is used to reduce the number of parameter computation in the training process of the network. The intuition of CNNs comes from the concept of cross-correlation. In cross-correlation, a multiplication between two signals is applied, summed up and then one signal is shift to the right and the process is repeated [24]. So, when these two signals are similar, we get a large value, and we get a small value when they are not. If we assume that one signal is a kernel, then we can capture a useful featueatedre. For example, in image processing when we are computing a dot product between the kernel and the input and producing the feature map of that

kernel, the network learns kernels that activate when they found some correlation between the kernel and the input. This is shown in Figure 2.11.

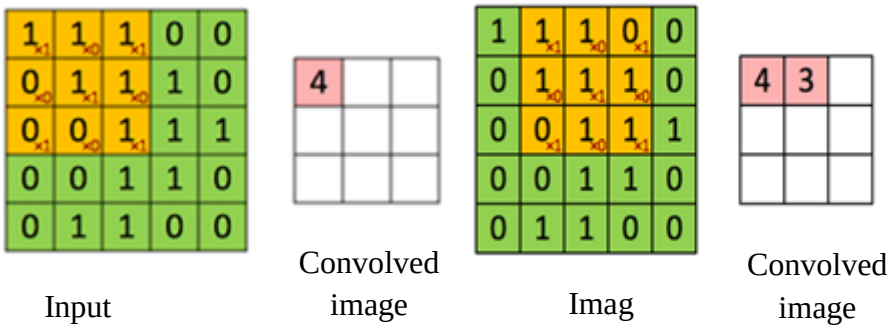


Figure 2.11 Left: The kernel is dot product with input pixel, Right: Convolution after shifting the kernel one position to the right [36].

Kernels with different shapes capture different features. In convolutional neural networks, we are going to learn this kernel in a supervised setting. So, we have some output to predict and the CNN can learn the useful feature to get the best prediction. More precisely, we learn some kernel weights to get this prediction. The difference between a feed forward neural network and a CNN is that the CNN has sparse weights, which is obtained by making a kernel smaller than the input nodes. Weights of the same kernel share the same layer as depicted in Figure 2.12.

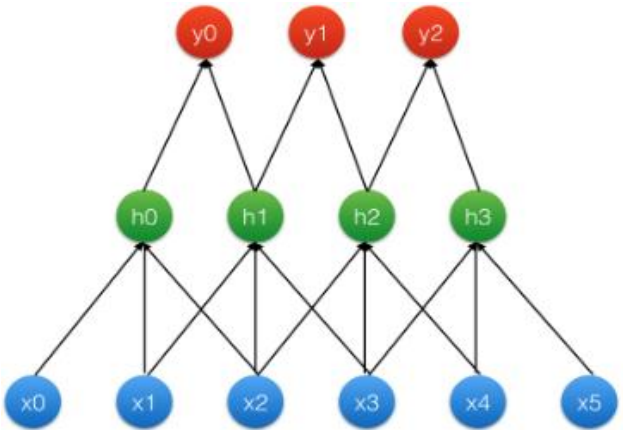


Figure 2.12 Sketch the shared weights of the same layer

For instance in the Figure 2.12, node $h_0 = f(W \begin{pmatrix} x0 \\ x1 \\ x2 \end{pmatrix} + b)$ and it is equal to $f(w_0 x_{0+} w_1 x_{1+} w_2 x_2 + b)$ and for $h_1 = f(W \begin{pmatrix} x1 \\ x2 \\ x3 \end{pmatrix} + b)$ which is equal

to $f(w_1x_{1+}w_2x_2 + w_3x_3 + b)$, which mean that W is shared by the same kernel in the same layer [25].

2.4 Architecture of CNN

Mainly, CNNs consist of three layers: the convolution layer and the pooling and fully connected layers.

2.4.1 Convolution Layer

Convolution has three important features that are very useful for machine learning: sparse interactions, parameter sharing, and equivariant representations. Furthermore, convolution can deal with inputs of variable size [26]. In normal neural networks, to define the relationship between each input neuron and each output neuron we need to do matrix multiplication, which means that every output is affected by the input layer. However, in a CNN this is not the case. Instead, each neuron will connect to a local region of input units. The convolution layer has sparse interactions, and this can be applied by selecting filter by size smaller than input size, although an image may have millions of pixels we can detect useful features by applying kernels to hundreds of pixels. Therefore, the number of parameters that the network has to learn decreases and efficiency improves [27]. If we have m input and n outputs, then we need $m \times n$ parameters to train the network. if we reduce the number of connections to k, then we need only $k \times n$ parameters to estimate as shown in Figure 2.13.

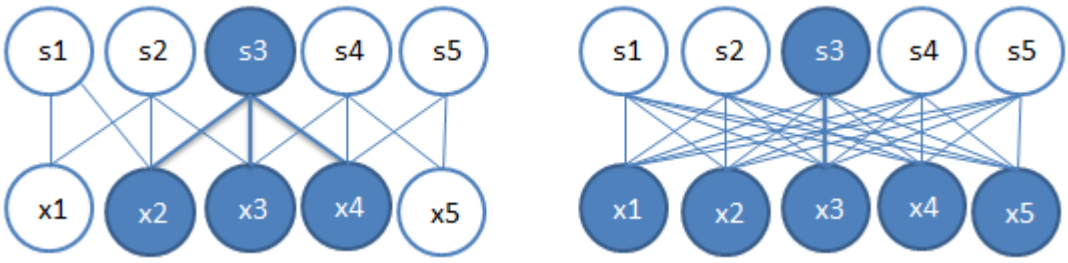


Figure 2.13 Left: Sparse connectivity, only three inputs of X, $(x2, x3\ and\ x4)$ are affected by $s3$ output. Right: all the input affect $S3$

Weight or parameter sharing leads to improved learning efficiency by decreasing the number of weights that we have to learn. That means the same weight will be used for more than one function of the network. This is shown in Figure 2.14.

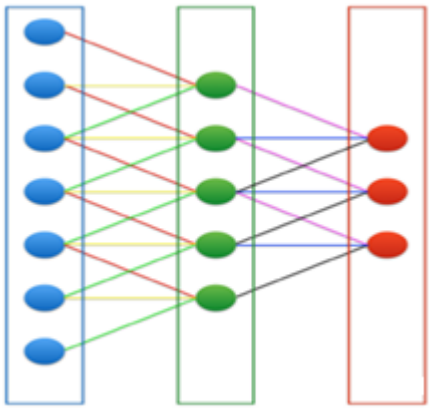


Figure 2.14 Connections with same colour share weights

In convolution, *equivariant* means that if we let g be any function that translates the input (i.e., shifts it), then the convolution function is equivariant to g. For instance, let $g(x)$ be a function such that for all i, $g(x)[i] = x[i - 1]$. This shifts every element of x one step to the right. If we perform this transformation to x and then perform the convolution, the result will be the same as if we applied the convolution to x and then applied the transformation to the output.

2.4.2 Pooling

Pooling can be defined as the operation of changing the output of the network at a restricted location with a statistical summary of the nearby outputs [26]. There are different types of pooling functions (e.g.,max-pooling, or selecting the maximum value in the rectangular nearby output). Another kind of pooling function takes the average of neighborhoods, using the L2 norm of the nearby rectangule. All of these types of pooling functions allow the representation of small translation of inputs to be invariant. Therefore, the value of the output after the pooling operation does not change if we translate a small part of the inputs as depicted in Figure 2.15.

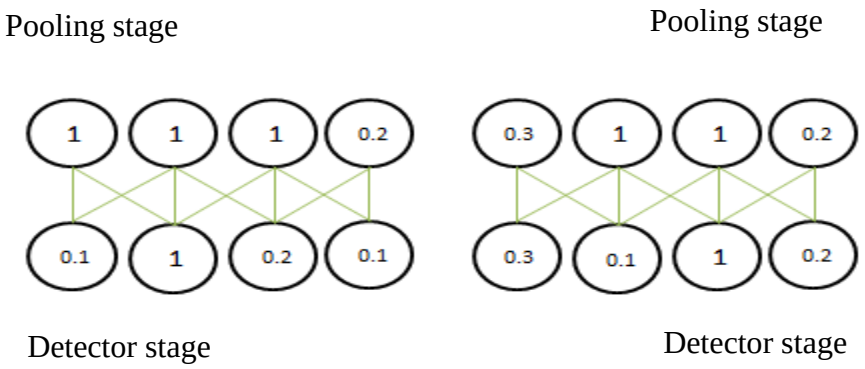


Figure 2.15 Invariance in max pooling, Left: The undermost row represents the outputs on nonlinear functions, the upper row represents the output of max pooling function, the width of the region is three with stride(how much we want to shift our filter at each step) equal one, right: The same network after shifted by one step to the right.

As we can see, all the values in the lower row have been changed, but only half of the upper row has changed. This is because a max pooling is influenced only by the maximum value in neighborhoods. Invariance is considered very useful in terms of local translations when we are concerned about whether a feature exists or not, rather than its exact position. In NLP we use pooling over the all of the filtering output, yielding a single number for each kernel. One advantage of using pooling layers is to produce a fixed size output matrix, which is required for classification. For instance, if we have 100-kernels and we apply max pooling to each, we will get a 100-dimensional output, regardless of the size of our kernel and of the size of our input. This allows us to classify sentences of different sizes. By applying pooling we minimize the dimension of output, but retain the useful features. We can think of each kernel as detecting a particular feature, such as capturing whether the sentence has a negative meaning (e.g., “not good”).

2.4.3 Fully Connected Layer

Fully connected layers can be defined as convolutions with kernels that cover their entire input region [28]. In a fully connected layer, each neuron has a connection with the previous layer as shown in Figure 2.16.

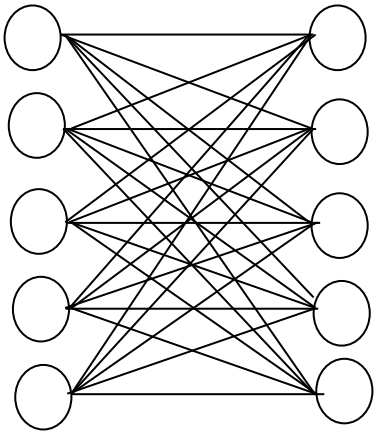


Figure 2.16 Fully connected layers

2.5 Dropout

In machine learning, it is important to train algorithms that can do well in both training and testing data. Overfitting is a big problem in neural networks. Dropout can be defined as a way of preventing overfitting in neural networks, and it is an efficient way to combine neural nets **[**29**].** During the training of the model, some of the hidden output units that would be ignored are set to zero. That means these units will be dropped during the computing of the forward pass and also in the back propagation stage [30]. For each training case, we randomly omit some of the hidden units so we end up with a different architecture for each training case. We can think of this as having a different model for every training case. Dropout is an efficient way to average many large neural nets. Suppose we have a neural net with one hidden layer as shown in Figure 2.17. Every time we present a training example, we randomly omit each hidden unit with probability 0.5. Effectively, we crossed out three of the hidden units. We then run the example of the net with those hidden units absent. We are randomly sampling from 2^*H* different architectures, where *H* is the number of hidden units. It is a huge number of architectures. All of these architectures share weights. That is, whenever we use a hidden unit, it is gets the same weight as the weight it gets in other architectures. Thus, we can consider dropout a type of model averaging. The sharing of the weights between all the models means that each model is very strongly regularized by the others. In testing, we use all the hidden units. Another way to think about dropout is that, if a hidden unit knows which other hidden units are present, it can co-adapt to them on the training data. But complex co-adaptations are likely to go wrong with new test data. So, by using the dropout method, we force a hidden unit to work combination with many

other sets of hidden units. And this helps the nets with dropout to have good performance.

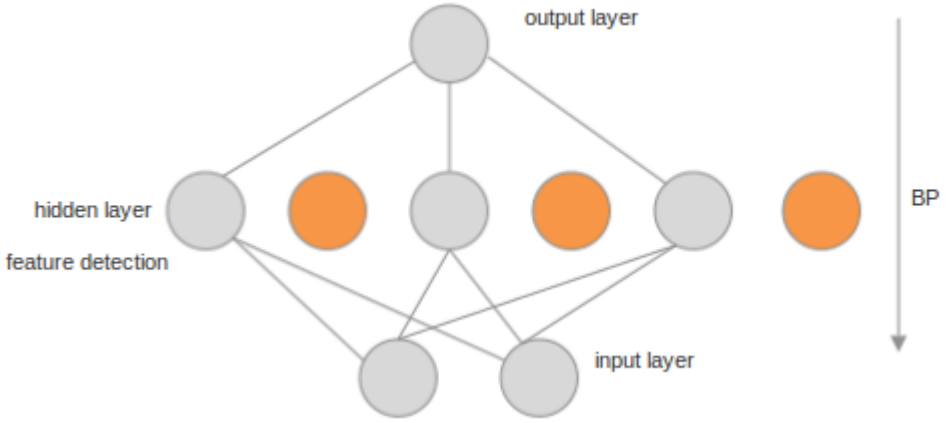


Figure 2.17 Dropout in hidden layer, coloured neurons are set their weight to zero during training stage

2.6 Activation Function

Activation functions are mathematical equations that can be used to scale the output of a neural network. They are used to make sure that the output value is in the desirable range. Some different types of activation functions [31] are listed below:

1. Linear activation function: A simple activation function that does not scale the output at all. In the linear activation function, we cannot use back propagation training because the linear function has no useful derivative.
2. Sigmoid activation function: The formula for sigmoid function is given by equation (2.23):

$$f(x) = \frac{1}{1+e^{-z}} \tag{2.23}$$

$$where\ z = \sum W_i X_i + b$$

$$f: \mathcal{R} \rightarrow [0,1]$$

The sigmoid gets its name from the shape of the graph as shown in Figure 2.18. Often, sigmoid function refers to a special case of the logistic function. In general, a sigmoid function is real-valued and differentiable.

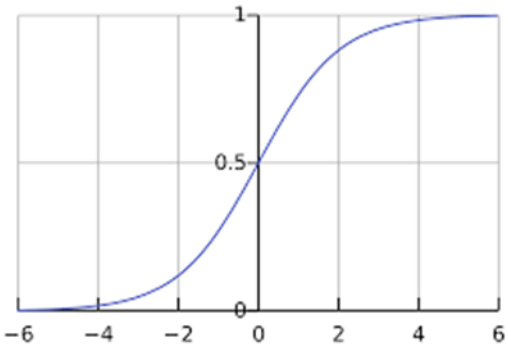


Figure 2.18 Sigmoid function

3. Hyperbolic Tangent (tanh): It defined as in equation (2.24):

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.24}$$

$$f: \mathcal{R} \rightarrow [-1,1]$$

4. Rectified Linear unit(ReLU): The ReLU is an activation function that sets the negative part to zero and retain the positive parts as shown in Figure 2.19.

$$f(z) = max(0, z) \tag{2.25}$$

$$f: \mathcal{R} \rightarrow [0, +\infty]$$

The advantages of using the ReLU activation function include:

- It solves what is called the exploding/vanishing gradient [54].
- It accelerates the convergence speed [32].

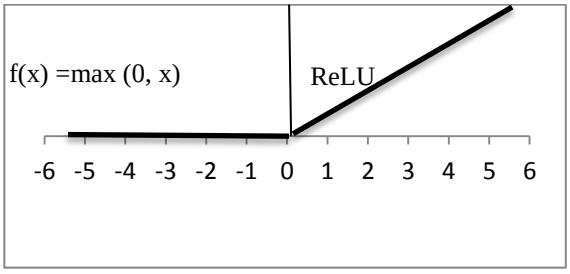


Figure 2.19 ReLU activation function

2.7 Cost Function

In machine learning, cost functions help us find out how to fit the most accurate straight line to the dataset. For instance, if we have a number of training samples, the hypothesis we use to find predictions is:

Hypothesis: $h_\theta = \theta_0 + \theta_1 x$

Where θ_i, are parameters. The question is how to choose these parameters (θ_0, θ_1)?

The idea is to choose (θ_0, θ_1) in a way that makes the hypothesis $h_\theta(x)$ very close for training samples (x_i, y_i). In other words, the choose parameters for (θ_0, θ_1) that minimize the difference between $h_\theta(x)$ and the actual output y. As a result, the objective function will be:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_i^m(h_\theta(x_i) - \mathrm{y}_i)^2 \quad (2.26)$$

Where *m* is the number of training samples, this cost function is sometimes called *squared error function* orthe *lost function.*

CHAPTER 3

OVERVIEW

In this research, we focus on classifying single sentences that are either positive or negative. To achieve this, we use CNNs for sentence classification, and we test the CNNs with different hyperparameters. We find that a little tuning in these hyperparameters can achieve good results using different benchmarks. Recently, a CNN model achieves good classification performance among a range of text classification tasks (e.g., sentiment analysis) and has since become a standard baseline for new text classification architectures. To regularize our CNNs, we use the dropout methods [33]. A dropout method is used to prevent the neurons from co-adapting. Furthermore, we define a loss function. The loss function calculates the error of our model, and we aim to minimize it. In our model we use the cross-entropy loss function [55]. In order to evaluate our models on sentence classification tasks, we use the accuracy, which is a good method for testing our model. And we used a gradient descent for optimization.

3.1 Datasets

The first dataset used in this thesis is the Movie Review (MR) data from Rotten Tomatoes [37]. The dataset contains 10,662 movie reviews with one sentence per review. Each review is classified as either positive or negative. The dataset has a vocabulary size of 20k (the set of terms of interest, denoted by v). We split the dataset as follows: 20% testing, 20% validation and 60% training.

Second dataset used is the Stanford Sentiment Treebank (SST) [34]. This dataset is widely used for evaluating sentence models. It consists of 11,855 movie reviews, each of which is given one of five labels (i.e., strongly positive, positive, neutral, negative,

strongly negative). The dataset consists of three sets: 8,544 sentences for training, 2,210 sentences for testing, and 1,101 sentences for validation[1].

3.2 Data Preprocessing

- Tokenization: For input as character arrangement, tokenization is a method of dividing data in to units called tokens and simultaneously eliminating certain characters, such as diacritical marks. A token is a case of succession of characters that are gathered together as a valuable semantic unit for handling.

- Stemming: This is the process of reducing derived words to their stem, or root form. Stemming programs are commonly referred to as "stemmers" or "stemming algorithms." A simple stemmer finds the inflected form in a lookup table. This approach is simple and fast. The disadvantage is that all inflected forms must be explicitly listed in table (.e.g., "*developed*", and "*development*", *developing*" are reduced to the stem *"develop*").

- Load positive and negative sentences from the raw data files (Load MR Polarity data). Next, split the data into words and generate labels.

- Clean the text data by converting the all the upper case letters to lowercase and get rid of the white space.

- Pad each sentence to the maximum sentence length, which turns out to be 59. We append special <PAD> tokens to all other sentences to make them 59 words. Padding sentences to the same length is a powerful process allows us to efficiently batch the data because each example in a batch is of the same length.

- Build a vocabulary index and map each word to an integer between 0 and 18,765 (the vocabulary size). Each sentence becomes a vector of integers.

[1] http://nlp.Stanford.edu/sentiment/

3.3 Tools and Environment

1. CUDA is a platform that is used in this thesis to execute our experiment on the GPU. The CUDA driver and toolkit is required for NVidia's GPU programming[2].
2. Tensor Flow is an open source software library for numerical computation using data flow graphs.
3. Experiment Environment: The following configuration describes all of the experiment tested with a computer:

- OS system: Ubuntu 16.04 LTS
- Processor: Intel® Core™ i7-6500U CPU @ 2.50GHz × 4
- Memory: 12GB 1333 MHz DDR4
- GPU: GeForce 940MX/PCIe/SSE2
- Python: 2.7.6
- NumPy: 1.9.2

3.4 The Model

The network presented in this thesis consists of multiple layers. The first layer is the input layer which embeds the words into low dimensional vectors. It is followed by a convolutional layer that performs convolutions over the embedded word vectors. The convolution is performed by using multiple filter sizes. Using different filter sizes enables us to capture different feature maps. Normally, sliding over 3, 4, or 5 words at a time is used. The next layer is a max-pooling layer, which is responsible for max-pooling the result or for stacking the result from the convolutional layer into a long feature vector. Later on, we add dropout for regularization, and finally, the soft-max layer is used to find the probability distributions over labels.

[2] CUDA Toolkit can be downloaded from the developer at nvidia.com.

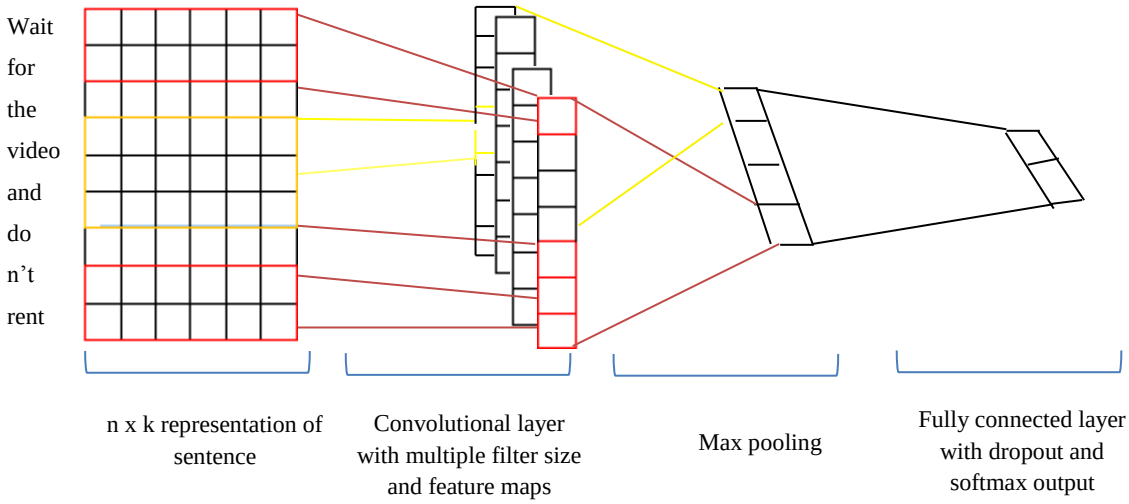


Figure 3.1 Convolutional Neural Networks model

3.5 Flowchart

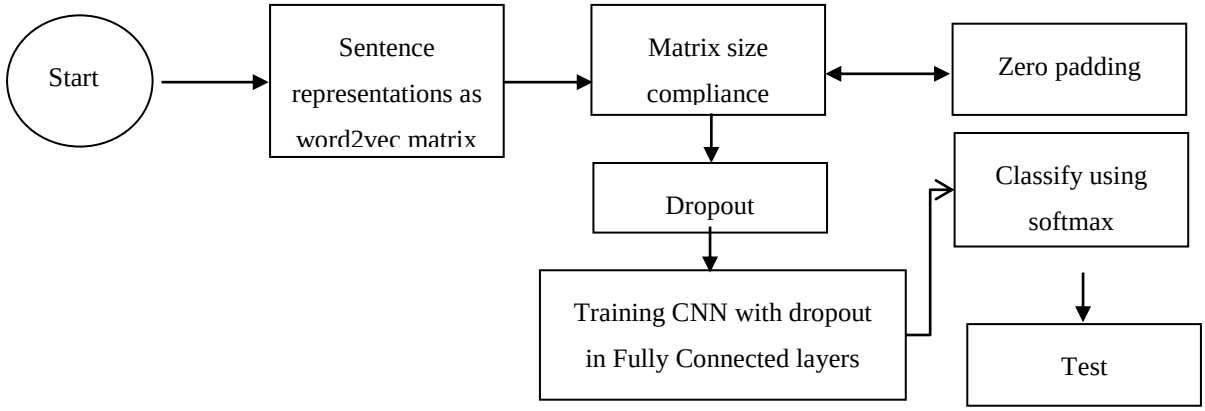


Figure 3.2 Basic flowchart

3.6 Methodology

In this section we describe the method used in this thesis. A document/sentence represented as a two-dimensional (2-D) matrix that is a concatenated vector of its sentences/words (each row is a words). The size of the input to the CNN is computed with respect to the dataset. The height of the 2-D input matrix is determined as the maximum length of sentence (in words). Short sentences are zero-padded and pass to the CNN. The convolutional neural network is trained with 0.5 dropouts in the fully connected layers. A simple softmax layer is applied over the output to find the probability distribution.

Different hyper-parameters are needed to create a model graph:

- Sequence length: The length of sentences. We padded all our sentences to have the same length (59 for our data set).
- Number of classes: Number of classes in the output layer, in our case, is two (positive and negative).
- Vocabulary size: The size of our vocabulary is required in order to define the size of our embedding layer
- Embedding size: Dimensionality of embedding.
- Filter sizes: The number of words that are covered by convolutional filters. The number of filters for each size is specified here. For example, [3, 4, 5] means that we will have filters that slide over 3, 4 and 5 words.
- Number of filters: The number of filters per filter size.

3.7 Our Approach

- We begin with a tokenized sentence, and then create a sentence matrix, in which each row of the matrix is the word vector representation.
- We denote the dimensionality of the word vectors by **d** and the length of sentence by **s.** The dimensionality of the sentence matrix is **s** x **d.**
- We apply a convolution on the sentence matrix via liner filters as shown in Figure 3.3.

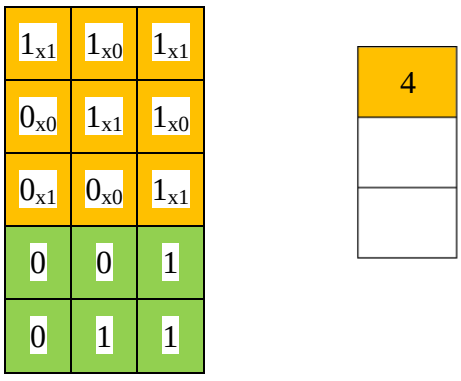


Figure 3.3 Convolution via linear filter

- The filter's width is equal to the dimensionality of the word vectors, and the height (region size) of the filter can be varied depending on the number of words we want to convolve.

- We denote the filter parameters by weight matrix w with the region size *h***,** and the matrix weight w will contain (*h* x *d*) parameters to estimated.
- We denoted the sentence matrix by $A \in \mathbb{R}^{s \times d}$, and use $A[i: j]$ to represent a sub matrix of *A* from row *i* to row *j*.
- Then the output sequence $o \in \mathbb{R}^{s-h+1}$ of the convolution process is obtained by applying the filter on sub-matrices of A as in equation (3.1):

$$o_{i=w \,.\, A[i:i+h-1]} \tag{3.1}$$

Where $i = 1 \ldots s - h + 1$

- We add bias *b* and the activation function *f* to each output o_i, inducing the feature map $c \in \mathbb{R}^{s-h+1}$ for this filter as in equation (3.2):

$$c_i = f\left(o_i + \mathrm{b}\right). \tag{3.2}$$

- A pooling function is applied to each feature map to create a fixed-length vector.
- We apply 1-max pooling, which is a non-linear down-sampling. It partitions input into non-overlapping rectangles. The output of 1-max pooling is the maximum value for each sub-region. Furthermore, it minimizes computation for next layer. This is shown in Figure 3.4.

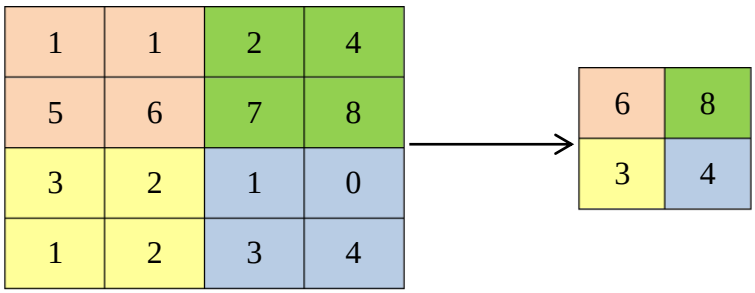


Figure 3.4 Max pooling applied to the feature maps

- Concatenate the outputs generated from each filter map into a fixed-length feature vector.
- Then the output features vector fed to the fully connected softmax layer to generate the final classification, Figure 3.6 illustrates these steps. Softmax turn the scores into probabilities [35]. This is shown in Figure 3.5.

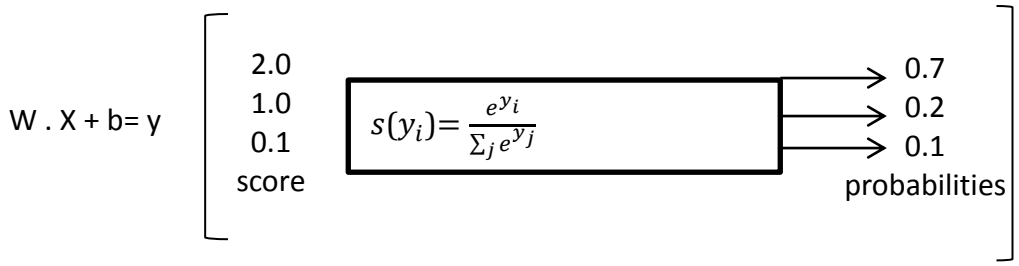


Figure 3.5 Softmax function

Probabilities sum to 1, and they will be large when the scores are large.

At this softmax layer, we apply "dropout" [29] as a means of regularization. This entails randomly setting values in the weight vector to 0.

A sensible training objective to be minimized is the categorical cross-entropy loss. For example, if we have several input variables, $x1, x2, ...,$ corresponding weights $w1, w2, \ldots,$ and a bias b As in Figure (3.6)

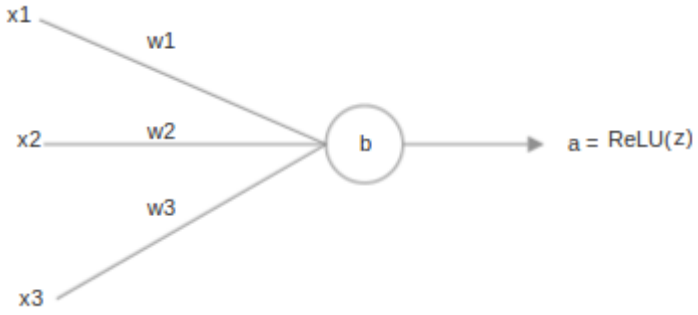


Figure 3.6 Example of simple neuron

The output from the neuron is, *a= ReLU* (*z*), where *z* is the weighted sum of inputs as in equation (3.3):

$$z = \sum_j w_j x_j + b \tag{3.3}$$

We define the cross-entropy for this neuron as in equation (3.4):

$$C = -\frac{1}{n}\sum_x [y ln\, a + (1-y)\ln(1-a)] \tag{2.26}$$

Where n is the total number of items of training data, the sum is over all training inputs, x, and the corresponding desired output$y$.

The parameters to be estimated include the weight vector(s) of the kernel and the bias term of the activation function. Figure 3.7 illustrate all of our steps.

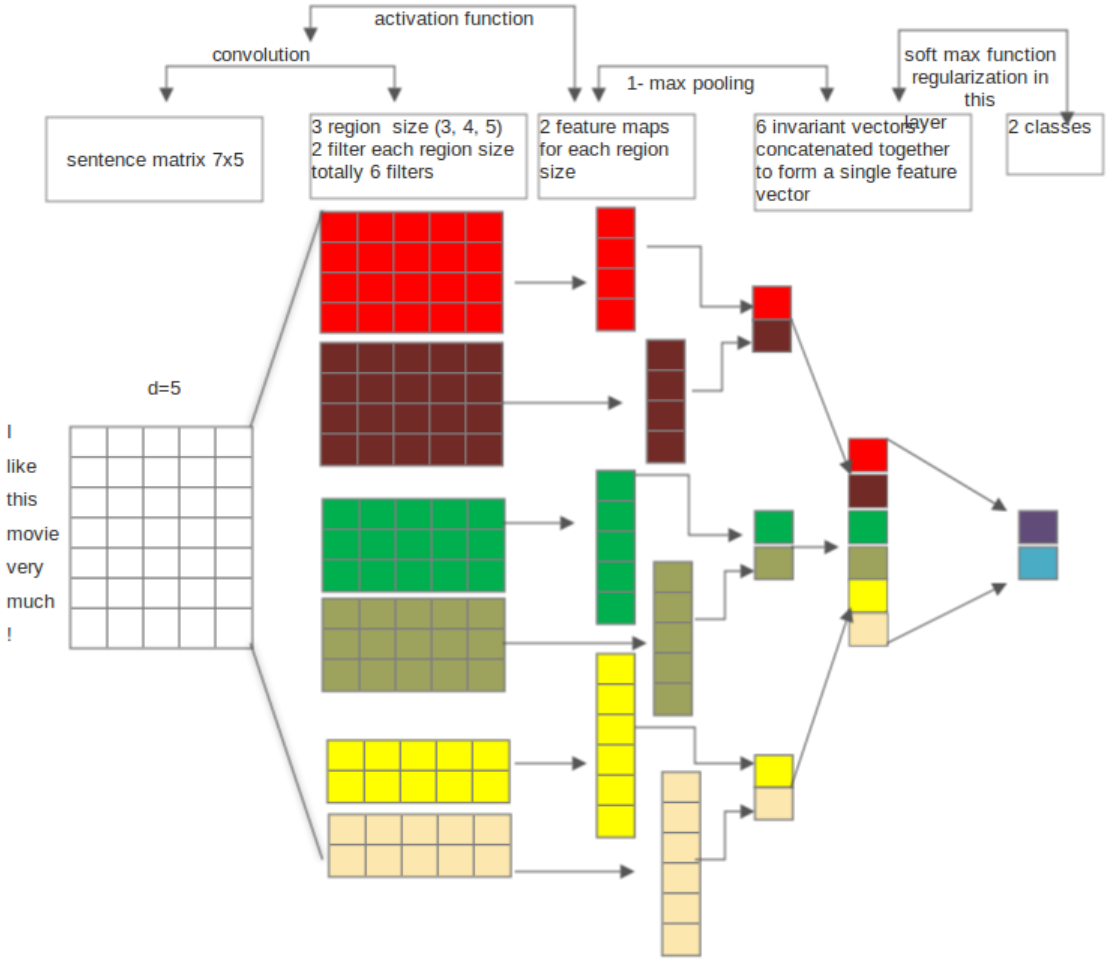


Figure 3.7 Convolutional Neural Network architectures

3.8 Embedding Layer

The embedding layer is the first layer we define, in which the vocabulary word indices map into low-dimensional vector representations. It's essentially a lookup table that we learn from data.

The weight (W) is the parameter that we will learn through training in our embedding matrix. The weight matrix is initialized randomly using a random uniform distribution function in tensor flow, which is creating the actual embedding operation.

3.8.1 Understanding word embedding

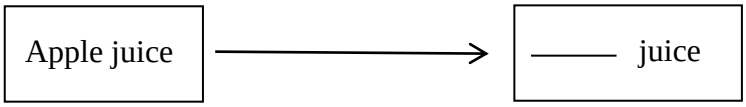
In order for a computer to process natural language, we need to create a representation of language. Often, it is called the representation of the words in the early years [14]. Many NLP systems use atomic word representations. For example, "Apple is represented by a vector whose length is the same as the number of words in the vocabulary. It is a very long vector, and only one component is one as shown below:

Apple [000010000000000….000]

Orange [000000000000010.....000]

Car [000000000100000….000]

This kind of representation creates a serious problem. For example, in the training data, if you see "apple" and 'juice" co-occurring, then in the test data, only using this representation, there is no way to figure out the relationship between these words. For example, we cannot know whether between "orange" and "car" which one better fits that blank spot in the second box below:



We want to represent words by the context appear in. We can deduce the meaning of a word by its neighbors. For example, we see that "apple" and "orange" here are similar in context, but they differ considerably from "car".

I eat apple

I eat an orange

I like driving my car to work

The idea of using word-embedding models is to preserve those kinds of similarity relationships, (e.g., if a pair of words is similar in the contextual-similarity sense). Then when they converted to low-dimensional representation, we expect them to be close to each other as well, as compared to other words, such as "bus," "train," and "car," as shown below, which should be quite far away because they are distinct from "foods" or "fruits".

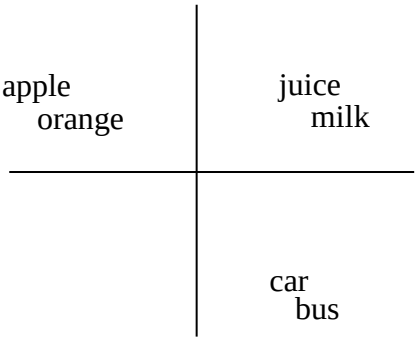


Figure 3.8 Word vectors

Recently, word2vec and other word embedding models have achieved remarkable results in NLP [56]. The idea is that they are not only able to preserve the distance between similar words with low dimensions. But, they are also able to preserve certain kinds of relations in the low-dimensional space.

For example, if we apply vector subtractions after putting these words in the low-dimensional space as shown in Figure 3.9, then subtract vector “king” from vector “queen” would be equal to vector “man” subtracted from vector “woman.”

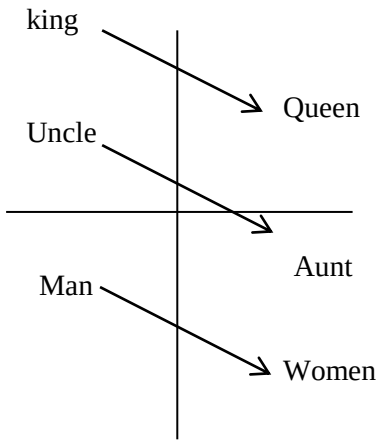


Figure 3.9 Word analogy

3.8.2 Training a text embedding model

In NLP in order to classify a document we are going to have to look at the words in that document. The ambiguity problem is one of the biggest challenges in NLP. The words that you rarely see tend to be the most important ones. Another problem is that we often use different words to mean almost the same thing. For example we can say “cat” or we can say “kitty”, but they are not the same. But their meaning is similar. Therefore, we really want to share parameters between them. If we want to share anything between

"kitty" and "cat", we are going to have to learn that they are related. Therefore, we would like to see those important words often enough to be able to learn their meaning automatically. And we have to learn how words relate to each other so that we can share parameters between them. And that would mean collecting a lot of labels. There are many ways to use the idea that says similar words occur in similar contexts. In our case we are going to map words to small vectors called embedding. Words that have similar meaning will be put close to each other; otherwise, they will be far. Once we have embedded our words into these small vectors, we have a word representation in which the entire catlike spectrum of cats, kitties, kittens, pets and lions are all represented by vectors that are similar.

3.9 Convolution and Max-Pooling Layers

After the embedding layer we can build our convolution layer, followed by max pooling. Different filter sizes are used in our model. Therefore, each convolution will create a different shape, so we need to iterate through them and produce a layer for each of them, and then merge the created layer into one big feature vector. A nonlinearity function is applied to the convolution output and is shown in Figure 3.10.

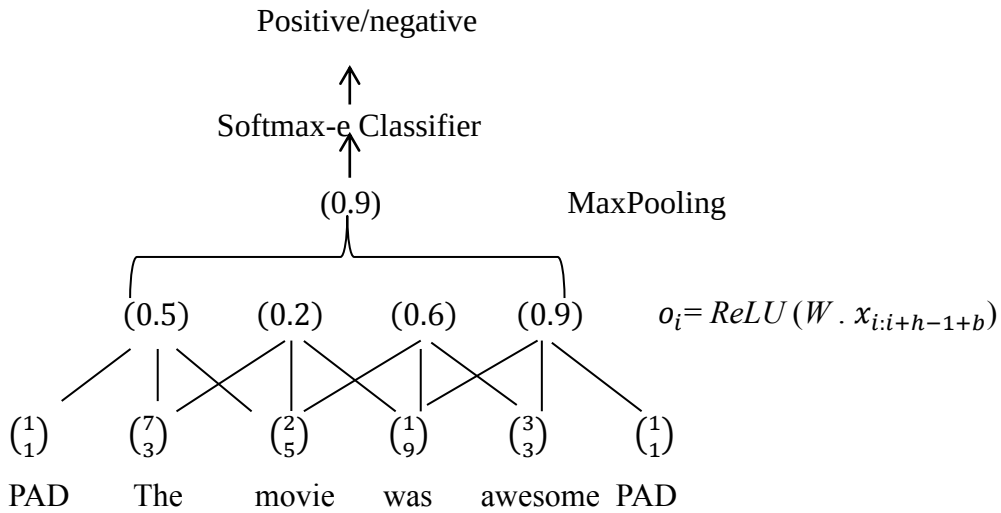


Figure 3.10 Convolution on three words and stride=1

Each filter slides over the all embedding, but differs in how many words it covers. Finally, max-pooling were applied to a specific filter size. This is essentially a feature vector, where the last dimension corresponds to our features. Once we have all the pooled output tensors from each filter size, we combine them into one long feature vector.

3.10 Loss and Accuracy

Loss function is a good way of tracking the error of our network in training stage. After finding the scores, we can calculate the loss function. Therefore, it is the function that tells us the degree of error our networks make and aims to minimize it. In this thesis the cross-entropy loss function is used as a standard loss function. The lower the loss indicated the better the model. The loss is computed on training and validation, and its translation of how well the model is doing for these two sets. However, loss can be defined as the summation of the errors made by each sample in training/validation sets. So, the main goal in a learning stage is to reduce the loss function's value regarding the parameters of the model. This can be applied by tuning the weight vector values through various optimization algorithms. Loss values used to evaluate the level of performance after each iteration of optimization. Ideally, we expect the loss value minimal after several iterations. The accuracy of a model is usually determined after the network reaches convergence status. At this point, model parameters are learned and fixed, and no learning is taking place. After that, the test examples are fed to the network and the number of errors the model makes is calculated. The percentage of misclassification is then computed by comparing the output of our network to the actual or target output. For instance, if we have 100 samples and our network classified 95 of them as true, the accuracy is 95%.

3.11 Narrow vs Wide Convolution

In order to apply the kernel to the first element of a matrix that does not have any neighbor elements to the top and left, we can use zero-padding, in which all elements located outside the matrix are zero. Therefore, we can apply the kernel to each element of our input matrix. If we add zero padding it is called *wide convolution*, and non-zero padding is called *narrow convolution*.

3.12 Stride Size

The other hyperparameter of our CNN is called *stride size*, which can be defined as how much we want to shift our kernel at each step. When the stride is 1, we move the filters one step at a time. When the stride is 2, the filters move two steps at a time. A stride of 2 produces less output volumes spatially.

CHAPTER 4

RESULTS AND DISCUSSION

The work done in this thesis is of text classification as it used movie reviews as classes. We have conducted an extensive experimental analysis of CNNs for sentence classification. To evaluate our model we used two datasets. The first is a Movie Reviews (MR) dataset with one sentence per review. The second dataset used is Stanford Sentiment Treebank (SST). We experimented with different hyperparameter values in order to find the optimal values for these parameters).

We tested the efficiency of various word embedding techniques (e.g., word2vec embedding).

4.1 Results

The training procedure is operated with default parameters (in 128-dimensional embedding, the sizes of the filter are 3, 4, and 5 with a dropout of 0.5 and 128 filters per filter size). Figure 4.1 shows the convergence of loss value and Figure 4.2 depicts the accuracy plots (the blue line is training data and the red line is validation data).

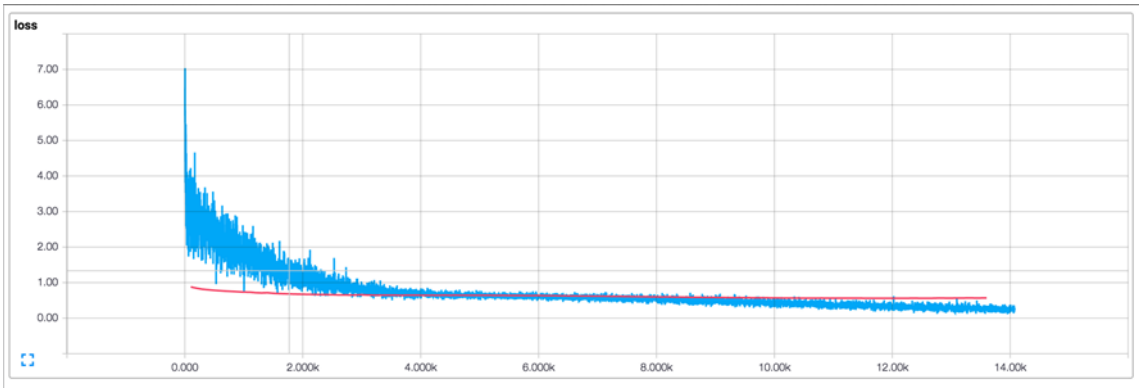


Figure 4.1 Loss value

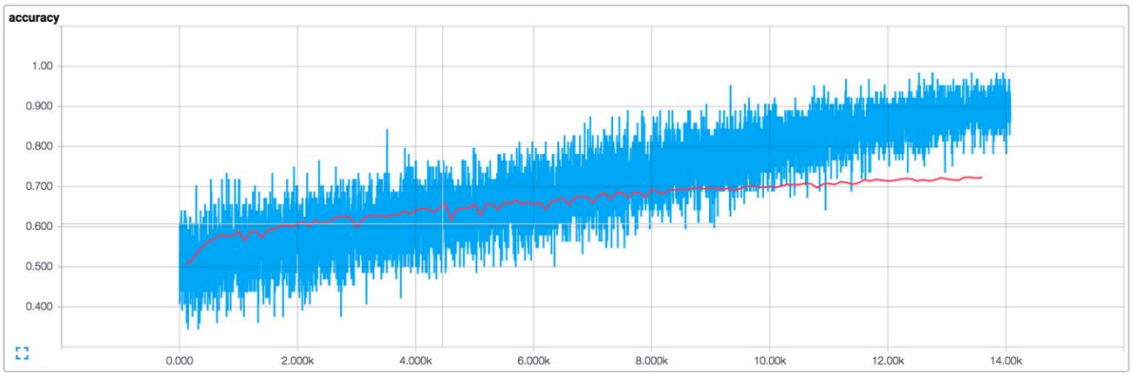


Figure 4.2 Accuracy value

We evaluate our model using various hyperparameters (e.g., batch size, dropout, filter size, number of feature maps). Then we tried to tune these parameters in order to improve the performance of our model. For all the hyperparameters we run the model ten times and takes the average. First, we trained our model using a batch size of 64 and a dropout equal to 0.1 to evaluate the net with respect to the accuracy(i.e., corret predictions from all predictions the model made), which can be define as in equation (4.1):

$$accuracy = (tp + tn)/(tp + tn + fp + fn) \quad (4.1)$$

and loss as shown in Table 4.1. Notice that this result is for the MR dataset, which learned embedding from scratch.

Table 4.1 the Accuracy percentage on batch size 64 and droupout 0.1

Method	Dropout	Batch size	Accuracy	Loss
CNN	0.1	64	70.50%	6.2

4.2 Effect of Dropout Values in regularization

The traditional way to regularize a CNN is by using L2 and dropout methods. The experiments are applied to a range of dropout values between 0.1 and 0.9. We measured the accuracy for each value of dropout used as shown in Table 4.2. We also use different kernel sizes (3, 4, and 5). We test the network with different values of hyperparameters in order to find optimal values for these parameters. We define a loss function to compute the error of our model, and our aim is to minimize it. We apply cross-entropy loss to measure the loss for each class, given the true label sample and our output scores of the network. After that we take the average of the losses.

Table 4.2 Accuracy percentage for different values of dropout

Method	Batch size	Loss	Dropout	Accuracy
CNN	64	6.2	0.1	70.50%
CNN	64	6.5	0.2	69.10%
CNN	64	6.3	0.3	71.50%
CNN	64	6.5	0.4	71.80%
CNN	64	6,2	0.5	73.90%
CNN	64	6.5	0.6	73.45%
CNN	64	6,2	0.7	73.50%
CNN	64	6.5	0.8	72.70%
CNN	64	6,2	0.9	73.73%

4.3 Effect of Batch Size

In Table 4.3 we examine our model in different batch sizes (8, 16, 32, 64). and set a fixed value for dropout equal to 0.5. We get the best accuracy value with a batch size of 64, and the loss value decreases to 6.2 as shown in Figures 4.3 and 4.4. All of the hyperparameters values (batch size and dropout) summarized in Table 4.4

Table 4.3 Accuracy and loss values over different values of batch size

Batch size	Dropout	Loss	# of Epoch	Accuracy
8	0.5	27.6	200	70.6%
16	0.5	16.5	200	71.3%
32	0.5	9.9	200	72.7%
64	0.5	6.2	200	73.9%
128	0.5	6.1	200	73.5%
256	0.5	6.9	200	70.0%

Table 4.4 Illustrate all the case of hyperparameters value

Batch Size	Loss	Dropout	Number of Epochs	Accuracy
8	27.6	0.5	200	70.60%
16	16.5	0.5	200	71.3%
32	9.9	0.5	200	72.7%
64	6.2	0.5	200	73.90%
64	6.2	0.1	200	70.50%
64	6.5	0.2	200	69.10%
64	6.3	0.3	200	71.50%
64	6.2	0.4	200	72.00%

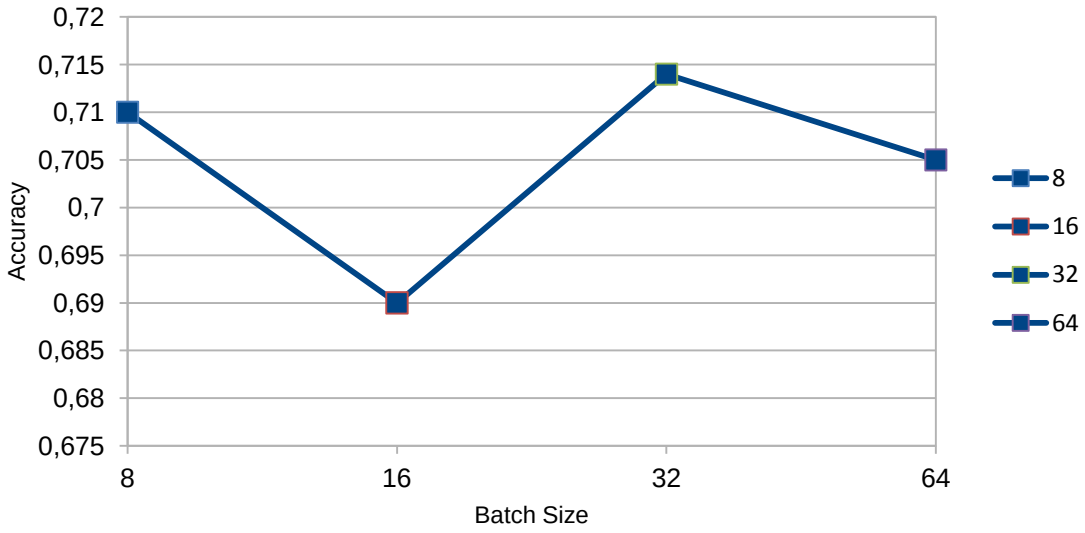


Figure 4.3 Impact of batch size on accuracy percentage value

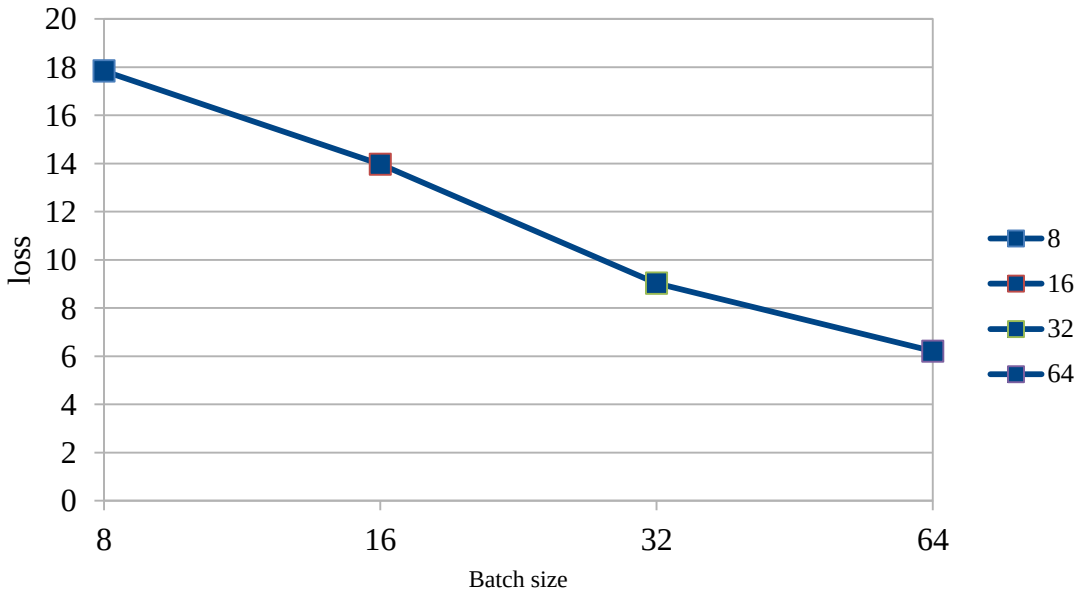


Figure 4.4 Impact of batch size on loss rate

4.4 Effect of Activation Function

We tested the proposed model with different activation functions in the convolution layer, including: hyperbolic tangent (tanh), rectified linear unit (ReLU), ReLU6, exponential linear (elu), sigmoid function, softsign and softplus. Table 4.5 shows the results of applying the various activation functions. For the MR dataset, the best activation function was ReLU and softplus.

Table 4.5 Effective of activation function on accuracy

Embedding dimension	Number of epoches	Dropout	Batch size	Activation function	Accuracy
128	**200**	**0.5**	**64**	**ReLU**	**0.744**
128	200	0.5	64	softplus	0.729
128	200	0.5	64	softsign	0.716
128	200	0.5	64	ReLu6	0.725
128	200	0.5	64	elu	0.713
128	200	0.5	64	tanh	0.710
128	200	0.5	64	sigmoid	0.712

4.5 Effect of Filter Size

Table 4.6 and Figure 4.5 show the effect of different single filter sizes (i.e., how many words the filtes slides around). We start with a single filter size and keep the number of

feature maps equal to 128. We explore the effect of region sizes 1, 3, 5, 7, 10, 15, 20, 25, and 30. We report the percentage changes in accuracy. The results show that the best filter size for sentence classification in the MR dataset is 3.

Table 4.6 Effective of single filter region size

Embedding dimension	filter size	# of epoch	Dropout	Batch size	Accuracy
128	1	200	0.5	64	0.729
128	**3**	**200**	**0.5**	**64**	**0.746**
128	5	200	0.5	64	0.732
128	7	200	0.5	64	0.735
128	10	200	0.5	64	0.733
128	15	200	0.5	64	0.724
128	20	200	0.5	64	0.719
128	25	200	0.5	64	0.721
128	30	200	0.5	64	0.720

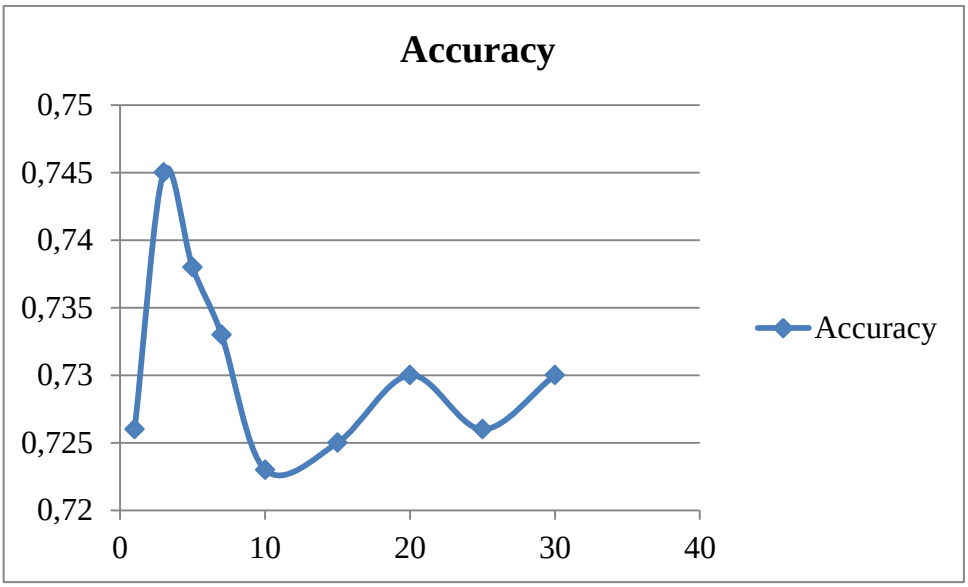


Figure 4.5 Effective of single region size on accuracy

We also explore the effects of multiple region sizes, when we hold the number of feature maps fixed at 128. We discover that using several filters with region sizes close to the optimal single region size can enhance performance, while using region sizes far

from the optimal range may decrease the performance of the model. For example, when using a single filter size, the best region size for the MR dataset is 3. Therefore, combining several different filter region sizes close to this optimal range can enhance the performance of our model. Comparing the results of this region to the approaches that use regions far away from the optimal single region size, we find that it decreases the performance of the model. For example, Table 4.7 and Figure 4.6 shows that using (2, 3, 4) and (3, 4, 5) which are near to the optimum single-region size 3 gives the best results.

Table 4.7 Effective of multiple region size

Multi Region Size	Accuracy
2,3,4	0.743
3,4,5	0.738
4,5,6	0.735
5,6,7	0.733
7,8,9	0.731

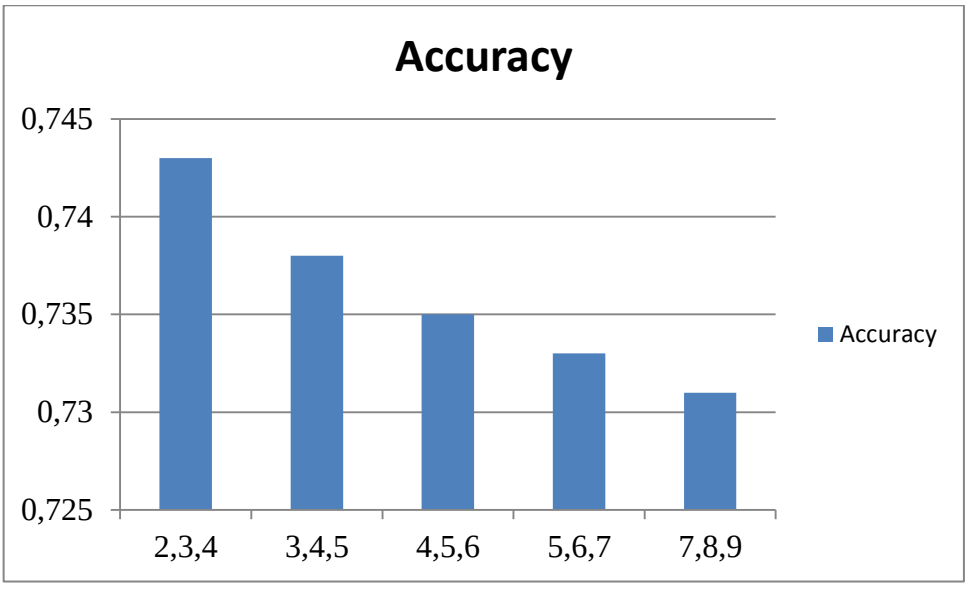


Figure 4.6 Effect of multiple region size

4.6 Effect of Number of Feature Maps in Each Filter Size

We keep the region size of filters fixed (i.e.,3, 4, and 5), and simple tune the number of feature maps for each region size. We experiment with values (10, 50, 100, 200, 400, 600, 1,000, 2,000). Table 4.8 shows the results of applying these values of feature maps. However, when the number of feature maps increases, it takes longer to train the model. The best number of feature maps depends on the dataset. In our case, for the MR dataset, the best number of feature map is 1,000, but it takes a long time to train the model.

Table 4.8 Effective of number of feature maps on accuracy

Embedding dimension	filter size	# of epoch	Dropout	Batch size	number of feature maps	Accuracy
128	3,4,5	200	0.5	64	10	0.723
128	3,4,5	200	0.5	64	50	0.735
128	3,4,5	200	0.5	64	100	0.736
128	3,4,5	200	0.5	64	200	0.738
128	3,4,5	200	0.5	64	400	0.731
128	3,4,5	200	0.5	64	600	0.735
128	3,4,5	200	0.5	64	1000	0.744
128	3,4,5	200	0.5	64	2000	0.742

4.7 Comparison of Word Embedding Methods

In this section we introduced the word2vec [1] tool based upon the consideration that word2vec has more semantic features. The word2vec can advance the sentence classification approach. Grouping similar words together using a distributed representation of words in a long vector space can increase the performance of learning algorithms.

We use filter sizes of 2, 3, and 4 with 128 feature maps each. The dropout rate is set to 0.5 after the convolutional layer on the last fully connected layer. A mini-batch size is set to 64. Finally, for each epoch, the training set is randomly shuffled.

Table 4.9 Accuracy of sentence classification tasks in comparison with using two kind of word embedding

	Methods	MR Dataset
Our work	**CNN pre-trained on top of Word2vec**	**81.60%**
	CNN learn embedding from scratch	**74.5%**

Table 4.10 Accuracy of sentence classification tasks in comparison with various models

Group	Method	SST
Baseline	CNN-non-static	47.4
Our Work	**CNN learn embedding using Word2vec**	**49.70%**
	CNN learn embedding from scratch	**46.5**
Recursive NN	Deep RNN	49.80%
	Constituency Tree-LSTM	51.00%
Recurrent NN	Bidirectional LSTM	49.10%
CNN Variants	Dependency CNN (ancestor+sibling+sequential)	49.50%
	Dynamic CNN	48.5
	d-TBCNN	51.40%

In Table 4.9 presents the comparison between the results of our model in two different kinds of embedding. For the MR dataset, our approach to CNN learn embedding using Word2vec shows a good improvement over the CNN learn embedding from scratch.

In Table 4.10 for the SST dataset, CNN learn embedding using Word2vec performs a little better than our baseline. However, the performance of a CNN traind by word2vec achieves a remarkable result overall.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this thesis, we described various experiments using a CNN for sentence classification tasks. We tested a CNN model that consists of multiple layers. The first layer embed words into dimensional vectors. In the first layer, we use word embedding to get word vectors. Useful features are extracted from the word embedding vectors by convolving filters with different sizes over this vector. The idea is to capture the most important features with the highest score for each feature map. A max pooling is used to create a long feature vector. For regularization, we apply a dropout to fully connected layer. Finally, a softmax is used to find the probability distribution over labels. We find that a little tuning of the hyperparameters of the network can achieve remarkable results in terms of accuracy and loss rate. We can summarize our work as follows:

- The method for input vector representation (e.g., word2vec) has an effect on the performance of the model. For sentence classification, word2vec achieves a remarkable enhancement compared to the one-hot-vector approach.
- Batch size has an impact on accuracy and loss value. We get the minimum loss value at a batch size of 64.
- Choosing the best number of feature maps can improve the performance of the model, even if increasing the number of feature maps can increase the training time of the network. We have a trade-off between these options.
- The filter region size can have a large impact on model performance. In our model the best single region size was 3. And we find that selecting multiple filter sizes close to the optimal single region size can improve the performance of the network.

5.2 Future Work

For future work, we suggest building a multi-class document classification task based on the results of this approach. Also we suggest trying to use sentence vectors instead of word vectors to test multi-sentence document-level classification tasks. Furthermore, using different CNN architectures (i.e., multiple convolution and pooling layers) might be a good idea.

Finally, we suggest adding L2 regularization and increasing the dropout rate help to prevent overfitting.

REFERENCES

[1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. doi:1301.3781.

[2] Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. doi:1510.03820.

[3] Kim, Y. (2014). Efficient estimation of word representations in vector space. doi:1408.5882.

[4] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. (2013). Recursive deep models for semantic compos itionality over a sentiment treebank. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP) , 1631: 1642.

[5] Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. In Advances in Neural Information Processing Systems. 649-657.

[6] Wang, P., Xu, J., Xu, B., Liu, C. L., Zhang, H., Wang, F., & Hao, H. (2015). Semantic clustering and convolutional neural network for short text categorization. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, 2: 352-357.

[7] Zeng, D., Liu, K., Lai, S., Zhou, G., & Zhao, J. (2014, August). Relation Classification via Convolutional Deep Neural Network. doi: 2335-2344.

[8] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011, June). Learning word vectors for sentiment analysis. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, 1: 142-150. Association for Computational Linguistics.

[9] Hu, B., Lu, Z., Li, H., & Chen, Q. (2014). Convolutional neural network architectures for matching natural language sentences. In Advances in Neural Information Processing Systems, 2042-2050.

[10] Pang, B., Lee, L., & Vaithyanathan, S. (2002, July). Thumbs up?: sentiment classification using machine learning techniques. In Proceedings of the ACL-02

conference on Empirical methods in natural language processing, 10: 79-86. Association for Computational Linguistics.

[11] Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. doi:1404.2188.

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

[13] Johnson, R., & Zhang, T. (2014). Effective use of word order for text categorization with convolutional neural networks. doi: 1412.1058.

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

[15] Wawre, S. V., & Deshmukh, S. N. (2013) Sentiment Classification using Machine Learning Techniques, 5(4): 2319-7064.

[16] Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. Foundations and trends in information retrieval, 2(1-2): 1-135.

[17] Zeiler, M. D., & Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. doi:1301.3557.

[18] Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. Machine learning, 37(3): 277-296.

[19] Marsland, S. (2015). Machine learning: an algorithmic perspective. CRC press.

[20] Ruck, D. W., Rogers, S. K., Kabrisky, M., Oxley, M. E., & Suter, B. W. (1990). The multilayer perceptron as an approximation to a Bayes optimal discriminant function. IEEE Transactions on Neural Networks, 1(4), 296-298.

[21] Williams, D. R. G. H. R., & Hinton, G. E. (1986). Learning representations by back-propagating errors. Nature, 323: 533-536.

[22] Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). Foundations of machine learning. MIT press.

[23] Ali Ghodsi, Lec [6], Deep Learning: convolutional network, Data Science Courses https://www.youtube.com/watch?v=ZMBp7_qqtLE, 21 Sep. 2015.

[24] Damelin, S. B., & Miller Jr, W. (2012). The mathematics of signal processing (No. 48). Cambridge University Press.

[25] Chen, Y. (2015). Convolutional Neural Network for Sentence Classification. Waterloo, Ontario, Canada.

[26] Bengio, Y., Goodfellow, I. J., & Courville, A. (2015). Deep learning. An MIT Press book in preparation.

[27] Chen, Y., Jiang, H., Li, C., Jia, X., & Ghamisi, P. (2016). Deep Feature Extraction and Classification of Hyperspectral Images Based on Convolutional Neural Networks. IEEE Transactions on Geoscience and Remote Sensing, 54(10): 6232-6251.

[28] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3431-3440.

[29] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1): 1929-1958.

[30] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. doi: 1207.0580.

[31] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, 1026-1034).

[32] Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network, doi: 1505.00853.

[33] Le, Q. V., & Mikolov, T. (2014, June). Distributed Representations of Sentences and Documents. In ICML , 14: 1188-1196.

[34] Pang, Bo, and Lillian Lee. (2005). "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales." Proceedings of the 43rd annual meeting on association for computational linguistics. Association for Computational Linguistics, 115-124.

[35] Boureau, Y. L., Ponce, J., & LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In Proceedings of the 27th international conference on machine learning (ICML-10): 111-118.

[36] Denny Britz , Implementing a CNN for Text Classification in TensorFlow (October 12 2016), Retrieved from Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution.

[37] Pang, Lee, (2005). Movie Review data, National Science foundation, United States, http://www.cs.cornell.edu/people/pabo/movie-review-data/.

[38] Matt Mazur, A Step by Step Backpropagation Example, (October 15 2016), Retrieved from Source: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example, Jun 2016.

[39] M. Hu, B. Liu. 2004. Mining and Summarizing Cusomer Reviews.In Proceedings of ACM SIGKDD, 168-177.

[40] McAuley, J., & Leskovec, J. (2013, October). Hidden factors and hidden topics: understanding rating dimensions with review text. In Proceedings of the 7th ACM conference on Recommender systems, 165-172.

[41] Wang, C., Zhang, M., Ma, S., & Ru, L. (2008, April). Automatic online news issue construction in web environment. In Proceedings of the 17th international conference on World Wide Web, 457-466 ACM.

[42] Phan, X. H., Nguyen, L. M., & Horiguchi, S. (2008, April). Learning to classify short and sparse text & web with hidden topics from large-scale data

collections. In Proceedings of the 17th international conference on World Wide Web, 91-100 ACM.

[43] Li, X., & Roth, D. (2002, August). Learning question classifiers. In Proceedings of the 19th international conference on Computational linguistics-1: 1-7. Association for Computational Linguistics.

[44] Levy, O., & Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In Advances in neural information processing systems, 2177-2185.

[45] Bishop, Christopher M. "Pattern recognition." Machine Learning 128 (2006), New York, NY, USA.

[46] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid O. S´eaghdha, Sebastian Pad o, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2010. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In Proceedings of the 5th International Workshop on SemanticEvaluation , SemEval '10: 33–38.

[47] Rus, V., McCarthy, P. M., Lintean, M. C., McNamara, D. S., & Graesser, A. C. (2008). Paraphrase Identification with Lexico-Syntactic Graph Subsumption. In FLAIRS conference, 201-206.

[48] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3): 273-297.

[49] Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In CAI 2001 workshop on empirical methods in artificial intelligence 3(22): 41-46. IBM New York.

[50] Cover, T. M., & Thomas, J. A. (2012). Elements of information theory. John Wiley & Sons.

[51] Xin Li and Dan Roth. 2002. Learning question classifiers. In Proceedings of the 19th international conference on Computational linguistics, 1: 1–7. Association for Computational Linguistics.

[52] Pang, B., Lee, L., & Vaithyanathan, S. (2002, July). Thumbs up?: sentiment classification using machine learning techniques. In Proceedings of the ACL-02 conference on Empirical methods in natural language processing- 10: 79-86, Association for Computational Linguistics.

[53] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li.2004. RCV1: A new benchmark collection for text categorization research. Journal of Marchine Learning Research , 5:361–397.

[54] Glorot, X., Bordes, A., & Bengio, Y. (2011, April). Deep Sparse Rectifier Neural Networks. In Aistats, 15(106): 275.

[55] Michael A. Nielsen, (2015). Neural Networks and Deep Learning, Determination Press.

[56] Yin, W., & Schütze, H. (2015). Learning Word Meta-Embeddings by Using Ensembles of Embedding Sets, doi: 1508.04257.

CURRICULUM VITAE

PERSONAL INFORMATION

Name Surname : NABEEL ZUHAUR TAWFEEQ ABDULNABI

Date of birth and place : 4th September1978, Mosul-Iraq.

Foreign Languages :Arabic and English

E-mail :Nabil78.nz@gmail.com

EDUCATION

Degree	**Department**	**University**	**Date of Graduation**
Master			
Undergraduate	Computer Science	Mosul University	2000-2001
High School	Scientific Department	Abdurrahman Al-Ghafeqee	1996-1997

WORK EXPERIENCE

Year	**Corporation/Institute**	**Enrollment**
2009-Currently	Mosul University-Iraq	Researcher
2005-2008	Employee at AsiaCell Telecommunication Company	IT at AsiaCell

PUBLISHMENTS

Papers

1. N. Z. T. Abdulnabi, O.Altun,	"Batch Size for Training Convolutional Neural Networks for Sentence Classification", Journal of Advances in Technology and Engineering Studies, 2016, 2(5): 156-163.

Conference Papers

1. N. Z. T. Abdulnabi, O.Altun,	"Batch Size for Training Convolutional Neural Networks for Sentence Classification", 30th International Conference on Engineering & Technology, Computer, Basic & Applied Sciences (ECBA 2016), 2 August 2016, Istanbul

.