SCHOOL OF COMPUTER SCIENCE AND INFORMATICS
CARDIFF UNIVERSITY

CM3203 - ONE SEMESTER INDIVIDUAL PROJECT

Automated Essay Marking with Machine Learning and Artificial Neural Networks

Angelos Charitidis

Supervisor: Dr. Frank C Langbein
Moderator: Dr. Jing Wu

Date of Submission: 11/05/2018

ABSTRACT

The aim of this paper is to investigate the application of Machine Learning in Automated Essay Marking. After a thorough background of Machine Learning, Artificial and Convolutional Neural Networks and fundamental word-to-vector representations, six of the main approaches of an Automated Essay Marking system using Convolutional Neural Networks are presented. The approaches differ about the word-to-vector representation, the essay representation and the network structure they are using. The approaches are then implemented to different software versions and their implementation is fully explained. The results of the various versions are first presented and then evaluated using six different evaluation metrics such as Accuracy, Confusion Matrix, Loss Graph etc. The results are very encouraging with some of the versions classifying testing examples with an Accuracy of more than 50%, but most importantly capturing the dataset's ordinal class structure. After evaluating the models and their results, the following outcomes were reached. The essay representation, as 1-dimensional or 2-dimensional, is insignificant given that the feature extraction part of the classification is performed with similar parameters. The word-to-vector representation is directly affecting the size of the essay and dimensions of the word-vector. As a result, a simpler word-to-vector representation can outperform a more complex word-to-vector representation, if the complex word-to-vector representation is not combined by a deep neural network that can "handle" its size.

ACKNOWLEDGEMENTS

This project would never happen without the invaluable advice and help by my supervisor, Dr. Frank Langbein.

I would like to dedicate this to my family who I dearly miss.

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

The aim of this project is to thoroughly investigate the application of Machine Learning in Automated Essay Marking. The reason Automated Essay Marking is a very important and interesting subject to investigate under the light of Machine Learning is because it gives us the chance to philosophically but also scientifically question whether a machine is able to capture patterns so obscure and complex such the patterns of essays in relation to their grades.

The combination of Machine Learning and Natural Language Processing in Automated Essay Marking will hopefully give us these answers. A human essay marker does not only assess an essay grammatically and/or syntactically, but also assesses the arguments made, and even more importantly, the support of the arguments made. So, this project will investigate whether a Machine Learning model can extract those features and classify the essays using them.

The dataset used originates from Kaggle and its purpose was "The Hewlett Foundation: Automated Essay Scoring" competition (Kaggle, 2012). It's a set of 1750 essays together with a grade from 2 to 12. All essays were written by students ranging from grade 7 to grade 10. Each essay was marked by two raters, who then agreed to an average, which is the mark that will be used for the classification.

This report includes 6 of the main Artificial Neural Network approaches, which were implemented and trained to 6 main versions. Through these different approaches/versions, I investigated the effect of three different factors of essay classification. First, the effect of the different representation of words as the essays are inserted into the Artificial Neural Network, implementing two completely opposite approaches and evaluating their performance. Second, the effect of the essay representation as the essays are inserted into the Artificial Neural Network, implementing 1-dimensional and 2-dimensional essay representation. Third, the effect of the depth or size of the Artificial Neural Network, by experimenting with various different model depths. Last but not least, I also investigated the dependency between those three factors by combining them in different ways.

The results were more than satisfactory. Four out of six models classified the essays with high accuracy showing signs of capturing the order of the grades and the natural structure of the data.

Most importantly, the three factors were fully examined and resulted to very important outcomes that will be vital in the next step of the project.

Both word representations, and thus the project, are based on the assumption that the order of the words in the essay is insignificant and that specific words will lead to a class regardless their position in the text.

The intended audience of this project is the research community. Although the results show that the produced models can be used as supplementary marking guides, if not as sole marking systems, the outcomes of this projects regarding the three factors effect in the classification can result to a highly accurate Automated Essay Marking in the very near future.

The approach used to carry out was a modified Agile methodology (Beck, et al., 2001). The first step of the methodology is to decide the approach. The initial model approach must be as simple as possible, in order to avoid building a model unnecessarily big or even worse overfitted on the training set which will be unable to successful predict new (or testing) examples. More information about model overfit will be provided later. The second step of the approach is to implement the approach and then the third step is to test it. The fourth and final step is to evaluate the results of the model. Using the results and the evaluation of the results from the fourth step, the first step is repeated by reconsidering or extending the approach. Since the intended audience is the research community, the version created by every cycle is stored and referenced to justify the new approach.

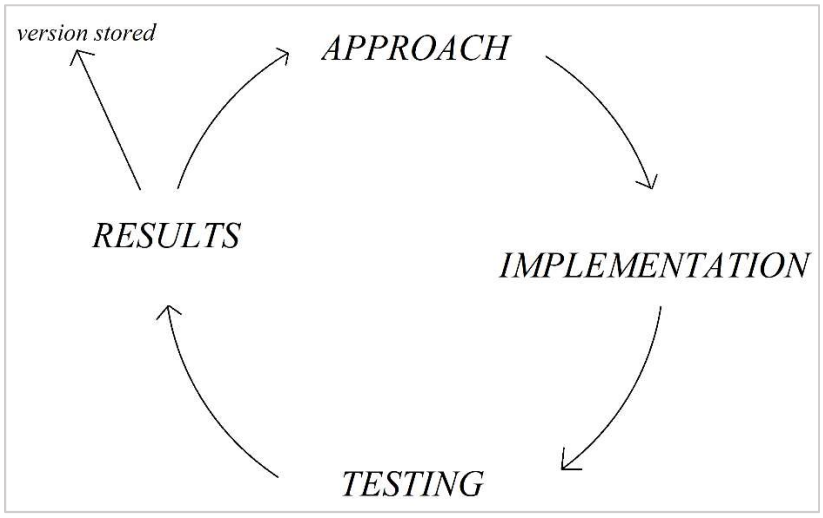

Figure 1: Software Development Cycle

2. BACKGROUND

2.1. MACHINE LEARNING

2.1.1. INTRODUCTION TO MACHINE LEARNING

Machine Learning is the notion of a machine progressively changing its structure, program, or data (based on its inputs or in response to external information) in such a manner that its expected future performance improves (Nilson, 1998). In other words, a Machine learns when it adapts to the new data and information.

Machine Learning tasks can be divided into two broad categories; Supervised and Unsupervised Learning, which differentiate based on their inputs and outputs. In Supervised Learning, for each observation there is an associated response measurement/class (James, et al., 2013). The most common Supervised Learning method is Classification, where the model is trained by the instance-class pairs of already classified examples (the training set) and can be then used to map other instances to the already existing classes. Taking for example the insurance industry, an example of a classification task could be to classify a number of instances of people with specific income, age, industry of employment, marital status and expected retirement age to two classes, one for insurance offer and one for insurance rejection. The inputs of the classification model will be the instances' information (specific income, age, industry of employment, marital status and expected retirement), whereas the output would be whether that person should or should not be offered an insurance plan. The model will be first trained by examined offer or rejection cases and can be then used to examine new cases. On the other hand, in Unsupervised Learning, the model is used to draw conclusions about a set of instances without an associated response/label (James, et al., 2013). The most common Unsupervised Learning method is Clustering, where the model is used to find possible grouping patterns of similar instances. Taking for example an online e-shop, Clustering could be used to find groups/clusters of similar products, so that after a costumer has viewed a specific product, the e-shop can recommend other products from the same cluster. The inputs of the Clustering model could be the products' information, such as price, sales, buyers' id, buyers' location and product category, whereas there are no set output options. There are also subcategories/special cases of Supervised Learning, which are Semi-Supervised Learning, Active Learning and Reinforcement Learning. The field of Machine Learning and the

general notion of a machine "learning" by progressively improving its performance in a specific task (Mitra, et al., 2018) is one of the most trending fields of computer science at the moment. In the last decade, scientists and engineers have tested its applications in various problems such as Medicine, Gaming, Insurance, Marketing, Fraud Detection and many other. So far, Machine Learning has achieved great results and is used more and more every day.

2.2. ARTIFICIAL NEURAL NETWORKS

2.2.1. INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks is a Machine Learning approach inspired by the biological neural networks which can be found in the human brain (Schalkoff, 1997). The human brain is able to capture complex information as patterns and then compare any new instance to the previously stored patterns in order to understand and classify it. For example, after a human has seen an elephant at least once, he is able to identify any new elephants as "elephant" by comparing it to the pattern of the previously seen elephant(s). Although Artificial Neural Networks are groundbreaking in Science's attempt to digitize the human brain, it is questionable whether the research community is any close to achieving that. The very mechanism of the human brain is a mystery (and an ongoing research) let alone the way to digitize this mechanism.

2.2.2. ARTIFICIAL NEURAL NETWORK vs OTHER MACHINE LEARNING APPROACHES

In comparison to some of the other Machine Learning approaches, such as regression models, Artificial Neural Networks can be used to capture very complex non-linear relationships. If we wanted to find the relationship between word count and the grade of the essay we could construct a simple regression model. However, the relationship between (the context of) an essay and the grade of the essay, which is the aim of this project, is far too complex to be described by a regression model. Thus, what an Artificial Neural Network is doing is divide a complex task, like the one investigated in this project, into several (maybe in the hundreds or thousands) smaller tasks and solve each one of those subtasks independently. In a classification task, for example, the task is to find a function f, which will satisfy $f(x) = y$, so that this function f is able to map an instance x to its class y. Artificial Neural Networks instead of attempting to find the

function f, are attempting to find a family of functions F, where $f_1, f_2, f_3, ..., f_n \in F$ and $f_n(\mathrm{f}_{n-1}(...(f_3(f_2(f_1(x))))...)) = f(x) = y$, thus achieving the same goal (Goodfellow, et al., 2016). Those function $f_1, f_2, f_3, ..., f_n$ are connected in a chain in order to achieve the task of f. This can be seen from the structure of Artificial Neural Networks.

2.2.3. BASIC STRUCTURE OF AN ARTIFICIAL NEURAL NETWORK

As said, Artificial Neural Networks are inspired by the Human Neural Network and so is their basic structure. Neural Networks are organized in layers. The layers of an Artificial Neural Network compose the Artificial Neural Network in the exact same way that $f_1, f_2, f_3, ..., f_n$ compose f. Each layer takes a value from the previous layer as an input, modifies it and then passes on the output to the next layer. As the task undertaken by the Artificial Neural Network gets more complex and demanding, more functions are required in order to divide the task. The number of layers in an Artificial Neural Network are also referred to as the depth of the Network.

The Artificial Neural Network's layers, in turn, consist of a number of interconnected (connected to the nodes of the previous and following layer) nodes called neurons (D.T. Pham, 1995). In summary, the neuron's function is to take an input from one or more neurons in the previous layer which are connected to it and pass on an output to the neuron or neurons in the next layer which the neuron is connected to. The actual function of the neuron depends on some parameters which are set by the Artificial Neural Network's architect. A weight is multiplied to the value of each connected neuron from the previous layer.

The first parameter is the "way" in which the inputs of the neuron from the previous neurons are combined. The simplest way is a weighted sum of their values, or in other words a sum of the neuron's input values v_i (output values of the neurons in the previous layer connected to this neuron) each of them calculated with a weight w_i.

$$\sum_{i=0}^{\mathrm{N}} w_i \cdot v_i$$

Equation 1: Artificial Neuron: Weighted Sum of Input values

where N is the number of neurons from the previous layer connected to the neuron.

The second parameter is set for the second phase of the neuron's function which is to input this sum in an Activation function f. This Activation function can be chosen from a number of different ones such as *sigmoid*, *tanh*, *Rectified Linear Unit* or *ReLU* to name some of the most important one. The purpose of the Activation function is to map the sum of weighted values to a value whose range corresponds to the range of the Activation function in order to normalize it. A table summarizing the properties of the main Activation function options is provided below.

Name	**Equation**	**Range**
Identity Function	$f(\mathrm{x})=x$	$(-\infty,\infty)$
Binary Step	$f(\mathrm{x})=\begin{cases}0 & for\ x<0\\1 & for\ x\geq 0\end{cases}$	$\{0,1\}$
Rectified Linear Unit (ReLU)	$f(\mathrm{x})=\begin{cases}0 & for\ x<0\\x & for\ x\geq 0\end{cases}$	$[0,\infty)$
Sigmoid	$f(x)=\frac{1}{1+e^{-x}}$	$(0,1)$
Arctan (Inverse Tangent)	$f(x)=\tan^{-1}x$	$\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$
Tanh (Hyperbolic Tangent)	$f(x)=\tanh(\mathrm{x})=\frac{2}{1+e^{-2x}}-1$	$(-1,1)$
Gaussian Distribution	$f(x)=e^{-x^2}$	$(0,1]$

Table 1: Artificial Neuron: Activation Functions

Thus, the aggregate function of a neuron can be expressed by the following equation:

$$f\left(\sum_{i=0}^{N} w_i \cdot v_i\right)$$

Equation 2: Artificial Neuron: Aggregate Function

where f is the Activation function

An importance notice to be made is that there are N neurons in the previous layer connected to the neuron, but there are N+1 values starting from 0 to N. The zeroth value v_0 is usually +1 and when calculated with its weight w_0 will give the bias input $b = w_0 \cdot v_0 = w_0 \ (since\ v_0\ is\ +1)$. The weights and the bias are very important concepts in Artificial Neural Networks and the concept of a machine "learning" and will be explained in depth later. To summarize, a neuron connected to N neurons from the previous layer will receive a sum of N+1 values, each multiplied by a weight out of a total of N+1 weights. The neuron will then apply an activation function to that sum and pass on the output to the next neurons to which it is connected to.

The mathematical model of the Artificial Neuron, described above, also known as the McCulloch-Pitts neuron was proposed by Warren McCullock and Walter Pitts in 1943 (Mcculloch & Pitts, 1943) and it is based on the theoretical point of view of creating a complex model out of very basic components, like the neuron. Much progress has been done in the field of Artificial Neural Networks but this model still remains the fundamental model of the Artificial Neuron. However, other models of the Artificial Neuron have also been introduced such as the Fourier-like IN/OUT function (Silvescu, 2000).

The following figure summarizes the function of the Artificial Neuron.

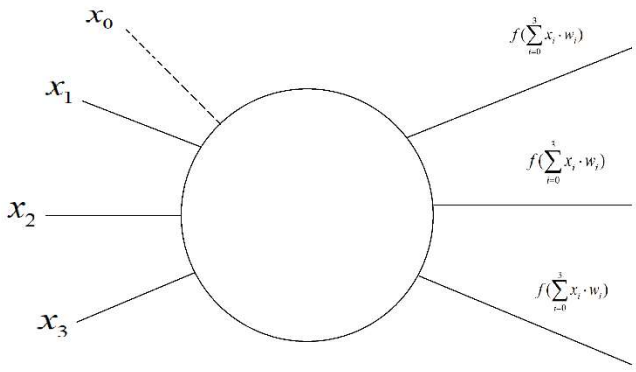

Figure 2: The Artificial Neuron

The most basic Artificial Neural Network consists of an input layer, an output layer and (between the input and output) one hidden layer. However, most Artificial Neural Networks consist of more than one hidden layer and depending on their task and the type/size of input could have hundreds of hidden layers. Figure 2 graphically represents the structure of the basic Artificial Neural Network, which was just described.

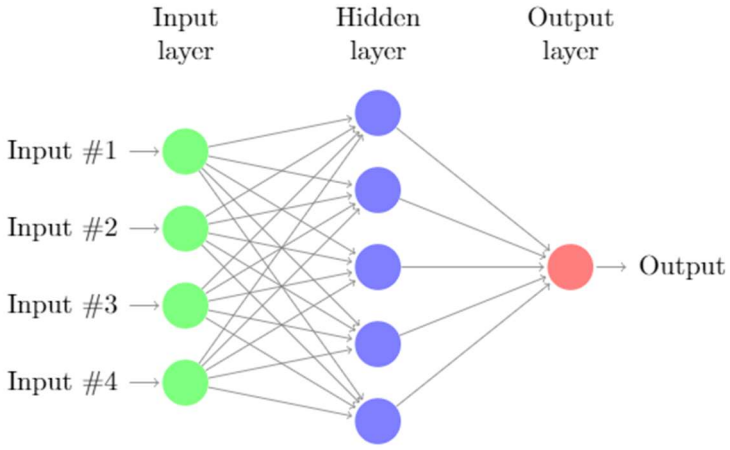

Figure 3: Basic Artificial Neural Network Structure (Fauske, 2006)

The Artificial Neural Network in figure 1 is an example of an Artificial Neural Network that could be used for a regression task. An example of this regression task could be predicting the GDP (Gross Domestic Product) of a country based on its population, percentage of population between 18 and 65, literacy rate and number of hospitals per citizen. So, the Artificial Neural Network would take in as input those four properties and produce an estimation for the GDP. The number of neurons in the input layer depends on the number of parameters/inputs needed to produce the estimation, in this case four. The number of neurons in the output layer depends on the type of output. In regression, the number of neurons in the output layer would be one as there can be only on dependent variable, whereas in classification (like in this project), the number of neurons in the output layer would equal to the number of possible classes.

Let's say for example that we want to classify a number of short 10-word movie or book reviews as positive or negative using an Artificial Neural Network. Each word is considered a parameter input, so there would be 10 neurons in the input layer, whereas there would be 2 neurons in the output layer, one for each of the different classes (positive or negative).

Then the basic structure necessary to perform this task would be the following:

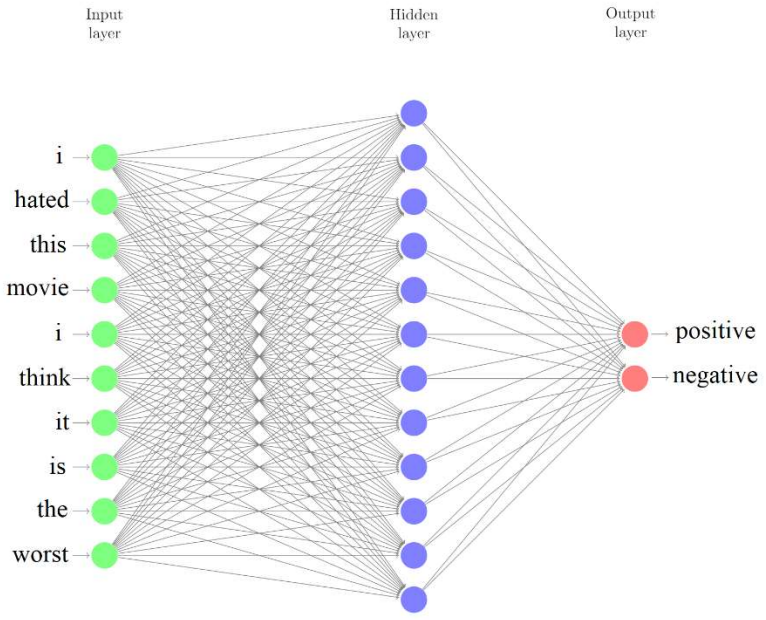

Figure 4: Basic Artificial Neural Network Structure for Text Classification

This is an oversimplified illustration as the input would not be a list of words but a list of vectors representing the words. The reason is that Artificial Neural Networks cannot process words. More information about Word Embedding in general and the Word Embeddings used in this project is provided later. Another reason the illustration above is a simplification of the Artificial Neural Network is that an Artificial Neural Network of this size (only one hidden layer) is unlikely to successfully carry out a task like this with adequate results. Instead, there should be some hidden layers before the hidden layer in the illustration that are responsible for extracting the features necessary for the classification.

2.2.4. TYPES OF ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks can be categorized based on various properties. Probably, the most unclear categorization is between ("Non-Deep") Neural Networks and Deep Neural Networks, where Neural Networks are classified based on the number of their hidden layers. The minimum number of hidden layers required for a Neural Network to be characterized as "Deep" is controversial topic. Part of the scientific community states that more than one hidden layers constitute a Deep Neural Network, whereas the other states that there needs to be a considerable number of hidden layers.

Another very important ground of classification is the type of connection between the neurons. In feed-forward neural networks, each neuron is only connected to other neurons in the next layers. On the contrary, in recurrent neural networks, there are connections called feedback which connect a neuron to itself and the output of the neuron is fed back to the neuron as a feedback.

2.2.5. ARTIFICIAL NEURAL NETWORKS OPTIMIZATION

As already mentioned, Machine Learning is achieved when the system's behavior is progressively improved in a certain task until it reaches a point where it can effectively and successfully perform this task. The measure, which is used to assess the performance of the Artificial Neural Network is called loss or cost. Thus, the objective of an Artificial Neural Network is to minimize the loss measure.

An Artificial Neural Network takes in the input in batches (e.g. the default batch size is 128 inputs, in our case, essays). Initially, the Neural Network assigns random values of weights w and biases b. After a batch has gone through the Artificial Neural Network, the loss is calculated by comparing the prediction calculated by the Artificial Neural Network to the actual label which was input. After the Artificial Neural Network calculates the loss, it reassigns the values of weight w and bias b of the neurons, modifying their output, so that loss is minimized. This procedure is repeated thousands of times.

LOSS CALCULATION

In order to develop a basic Artificial Neural Network, it may be deemed enough to know that the smaller the loss the better. However, in order to really understand how the Artificial Neural Network works it is vital to understand how the loss is calculated.

There are various loss calculation functions depending on the task. Cross-Entropy loss is one of the most popular and is highly recommended in multiclass classification tasks (classification tasks where there are more than 2 classes) such as this task. Cross-entropy loss, also called log entropy (because of the presence of the natural log in the formula) is calculated for each instance (prediction-label pair) and the sum of the Cross-Entropy loss for all instances gives us the total Cross-Entropy loss of the model. Cross-Entropy for multiclass classification is calculated for each instance using the following formula:

$$L_{ins} = -\frac{1}{M}\sum_{cl=1}^{M} y_{ins,cl}\log(\mathrm{P}_{ins,cl})$$

Equation 3: Cross Entropy Loss (Deep Learning Course Wiki, n.d.)

where - M is the total number of classes,

- ins is the index of the instance,
- cl is the index of the class,
- $y_{ins,cl}$ is a binary indicator (range = $\{0,1\}$) indicating whether an instance belongs to a class, e.g. if instance $ins = 3$ belongs to the class $cl = 2$ then $y_{3,2} = 1$ and - $y_{3,1} = y_{3,3} = y_{3,4} = 0$,
- $(\mathrm{P}_{ins,cl})$ is the probability given from the classifier that instance ins belongs to class cl

Suppose a batch of 4 training examples have gone through the Artificial Neural Network, which classifies the examples to 3 different classes, and calculates the following prediction probabilities [[0.9, 0.05, 0.05], [0.6, 0.4, 0.0], [0.51, 0.39, 0.1], [0.3, 0.2, 0.5]]. The labels of the training examples will first be converted to one-hot encoding. In one-hot encoding, each label is converted to a format which can be more easily compared with the prediction probabilities. If the labels of the 4 training examples are [1,2,2,3], then the one-hot encoding of those labels will be [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]].

Calculating for the 1st instance:

first instance prediction probabilities = [0.9, 0.05, 0.05]

first instance one-hot encoding of label = [1.0, 0.0, 0.0]

$$
\begin{aligned}
L_1 &= -\frac{1}{M}\sum_{cl=1}^{M} y_{ins,cl}\log(\mathrm{P}_{ins,cl}) = \\
&= -\frac{1}{M}(y_{1,1}\log(\mathrm{P}_{1,1}) + y_{1,2}\log(\mathrm{P}_{1,2}) + y_{1,3}\log(\mathrm{P}_{1,3})) \\
&= -\frac{1}{3}(1\cdot\log(0.9) + 0\cdot\log(0.05) + 0\cdot\log(0.05) = \\
&= 0.0351
\end{aligned}
$$

The correct labels was predicted with an accuracy of 0.9 or 90%, as a result the loss calculated is very low.

Calculating total loss:

$$
\begin{aligned}
L_{total} &= L_1 + L_2 + L_3 + L_4 = \\
&= 0.0351 + 0.3054 + 0.3139 + 0.2311 = \\
&= 0.8855
\end{aligned}
$$

LOSS MINIMIZATION (OPTIMIZATION)

As stated before, each neuron receives an input from the previous N neurons connected to it and applies weights to the values resulting to a sum in the form of $\sum_{i=0}^{N} w_i \cdot v_i$. It then applies an activation function, such as sigmoid, and passes this value to the next neuron(s) connected to it. The recalibration of the Artificial Neural Network in order to minimize the loss is achieved by changing the weights w and so the biases b. As stated before, the aim of the Artificial Neural Network is to find the function f which will satisfy $f(x) = y$. This is achieved by finding the functions $f_1, f_2, \ldots, f_n \in F$ which will satisfy $f_n\left(..\left(f_2\left(f_1(x)\right)\right)\ldots\right) = y$ and thus compose the function f. Thus, the task of the Artificial Neural Network layer n is to find the respective function f_n.

Suppose a neuron with the identity function $f(x) = y$ is connected to one neuron from the previous layer with output value x. A weight w_1 is multiplied with this value and bias b_1 is also added, resulting to the value $y = w_1 x + b_1$. The plot of the value y for every x will be a straight line of gradient w_1 intercepting the y-axis at $(0, b_1)$ and the x-axis at $\left(\frac{-b_1}{w_1}, 0\right)$.

By changing the weight and bias, thus changing the gradient and intercepts of the function, the Artificial Neural Network can form an infinite number of linear functions until it finds the function that satisfies the aforementioned conditions and each input x is mapped to a correct. The effect of different weights w and biases b can be seen in the plot of 5 different $f(x) = w_1 x_1 + b_1$ functions with different weight w_1 and bias b_1 provided bellow.

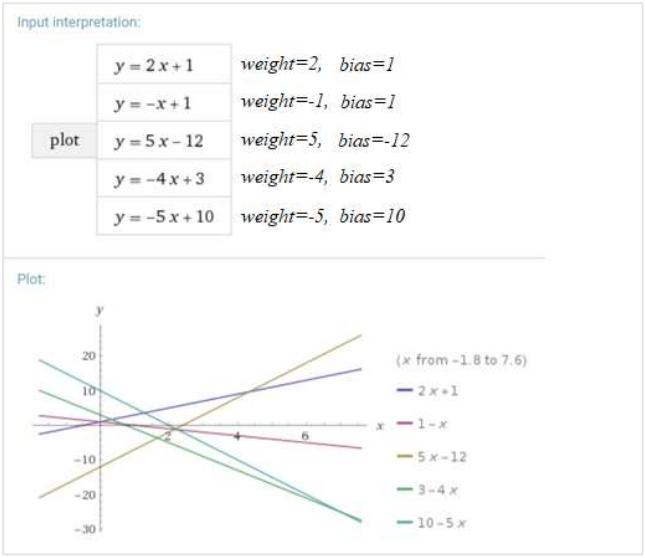

Figure 5: Effect of Weight and Bias using the Identity function as Activation function

When the correct weight w_1 and bias b_1 are found so that every x is mapped to the correct y, the loss will be minimum. So the loss can be thought as a function $L(w,b)$ which reaches a global minimum for a weight w and bias b. At the global minimum (as well as all local minimums), as we know from calculus, the derivative of the loss function $L'(w,b)$ at that particular point will be 0.

The weights w and biases b that will minimize the loss are calculated by optimization algorithms. Feed forward networks, like Convolutional Neural Networks which are used in this project, use an optimization algorithm called Backpropagation, short for "backwards propagation of errors". During training, the input will start from the input layer and propagating through the hidden layers will reach the output layer and the Artificial Neural Network will produce an output. This is called a forward propagation as the information flows from the start to the end of the network. Using this output to calculate the loss, the network will then start from the end of the network and using the loss calculated, will compute the gradient of the loss $L'(w,b)$ .The information of loss flows backwards from the end of the network to the start to compute the gradient. Thus, the name back-propagation (in contrast to forward propagation). The Back-propagation algorithm was introduced by Rumelhart et al. at their paper "Learning representation

by back-propagating errors" (Rumelhart, et al., 1986). This is done by applying very small increments to weights w and biases b.

It is important to notice that Back-propagation is only responsible of finding the gradient of the loss function and not the recalibrating the weights and biases to minimize it. Most Artificial Neural Networks use a gradient descent algorithm to calculate the new weight. As we said, the Loss function $L(w,b)$ is minimum at the point where $L'(w,b)=0$. Applying the back-propagation algorithm and getting the gradient of $L(w,b)$ at a particular point, the gradient descent algorithm then changes weights towards the negative gradient or gradient descent in order to minimize the loss function $L(w,b)$ until it reaches a global or local minimum where $L'(w,b)=0$.

The optimization algorithm that was used in this project is called Stochastic Gradient Descent and will be explained in depth in the "Approach" section.

2.2.6. SAMPLES REPRESENTATION IN ARTIFICIAL NEURAL NETWORKS

The samples that are inserted in an Artificial Neural Network have a very specific representation dictated by their type.

Starting from images, the representation of an image depends on its color model. A 9x9 image for example (height=9, width=9) would be of the form 9x9x3 if it was in rgb color model (red, green, yellow) or 9x9x1 if it was in greyscale model. This third value after height and width, in Artificial Neural Networks, is called channels. It can be confusing that although it is a 2-dimensional sample, an image, it is composed by three different values. However, considering how computers conceive images it is absolutely understandable. In greyscale, there is only one channel, where each of the 9x9=81 pixels is a value from 0 to 1 (including 0 and 1) representing the intensity of black. A white pixel would have the value 0, a black pixel would have the value 1 and a grey pixel would have a value somewhere in between 0 and 1. In RGB, however, there are three different and independent channels, each consisting of 81 pixels that describe the intensity of the channel's color. There is one channel for each of the three colors (Red, Green, Blue).

Text samples are not that different. A 25-word sample can be either inserted in the Artificial Neural Network in the form 25x3 or it can be inserted in the form 5x5x3. Both forms have been

investigated in this project. The channels, however, in this case do not represent the intensity of a color, but a specific characteristic of the word. It can be either practical, such as the frequency of the word in the essay (or sentence etc.) or the index in a word dictionary, or it be a semantic characteristic of the word, such as whether this word has a political meaning or a scientific meaning. For example, the value of the word "Democracy" would have a high value in the political meaning channel and a low value in the scientific meaning channel, whereas "Cross-Entropy" would the opposite.

2.3. CONVOLUTIONAL NEURAL NETWORKS

The neural network I am using is a Convolutional Neural Network. Although the main use of Convolutional Neural Networks is Image and Video (objects') recognition, it has also been applied to Natural Language Processing tasks, such as Text Classification and Prediction, and has shown great results. One of the most popular examples of Text Classification with CNNs is "Convolutional Neural Networks for Sentence Classification" (Kim, 2014). Other examples are:

- "A Convolutional Neural Network for Modelling Sentences" (Kalchbrenner, et al., 2014),

- "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (Zhang & Wallace, 2015) and

- "Multichannel Variable-Size Convolution for Sentence Classification" (Yin & Schütze, 2016)

The reason Artificial Neural Networks with Convolutional (and Pooling) Layers are successful in text classification is because they are able to detect strong local word/indicators that will lead to a class without taking into consideration their position in the input text (Goldberg, 2016). Suppose an Artificial Neural Network with Convolutional Layers has been trained to classify and mark 100 essays with the topic "The Applications of Machine Learning in Business Management". Words and phrases like "decision-making" or "perfect information" which would be good arguments will probably be strong indicators that will point out to a good grade.

2.3.1. BASIC STRUCTURE OF A CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Network is a type of feed-forward network. Its name is attributed to the presence of at least one Convolutional Layer (followed by one or more down-sampling layers) in it. The first layer of a Convolutional Neural Network is always a Convolutional Layer.

2.3.2. CONVOLUTIONAL LAYER

Convolution can be summarized as a number of filters containing weights "sliding" over an input sample. However, the actual function of a Convolutional Layer is a lot more complicated. In Convolution, a 1-dimensional or 2-dimensional filter, consisting of a number of weights goes over its input performing a series of calculations similar to the ones we described in the Artificial Neural Network.

The function of the Convolutional Layer is determined by a number of parameters, such as kernel size, stride, number of filters and activation function.

The exact function of a Convolutional Layer can be better described using an example and such is provided below. In the Samples Representation section, we analyzed the different Samples representation and the importance of channels. Let a text sample of the form 8x3. This would be an 8-word sample where each word is described by 3 different features.

CONVOLUTION INPUT

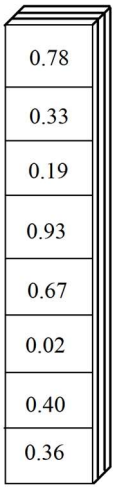

Figure 7: Convolution Illustration - Input

FILTER(S)

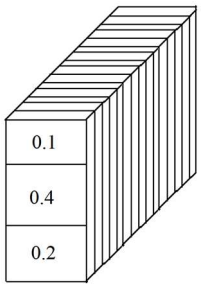

Figure 6: Convolution Illustration - Filter

The parameters of the above convolution are kernel size = 3, filters = 15, activation function= sigmoid and stride=1. The first two parameters can be identified in the above figure 4 since each of the kernels (or filter) is consisted of 3 values and there is a total of 15 filters. The other two parameters are explained during the demonstration of the example.

When we introduced the Artificial Neuron, we mentioned the weights that are multiplied to the neuron. The filters that "slide" over the input sample are nothing more and nothing less than a "group" of weights. So the above Convolutional Layer is consisted of (3(each filter) times 15(number of filters)) 45 weights.

STEP NUMBER 1

In the first step of convolution in the above example, the first filter will be "applied" on the input. Since the filter size is 3, the filter will be applied on the first 3 values (values 1 to 3) of the input. The sum of the three values will then be inserted to the activation function (in this case sigmoid, $f(x)=\frac{1}{1+e^{-x}}$) and will give the first value.

O.78	x	0.1	= 0.078
0.33	x	0.4	= 0.132
0.19	x	0.2	= 0.038

$$\sum = 0.078 + 0.132 + 0.038 = 0.248$$

$$f(0.248) = \frac{1}{1+e^{-0.248}} = 0.5617$$

0.5617

Figure 8: Convolution Illustration - Step 1

STEP NUMBER 2

The stride parameter mentioned before determines the next step of the convolution. In this case (stride=1), the filter will "slide" one position below (values 2 to 4) and will perform the same operation again.

0.33	x	0.1	= 0.033
0.19	x	0.4	= 0.076
0.93	x	0.2	= 0.186

$$\sum = 0.033 + 0.076 + 0.188 = 0.295$$

$$f(0.295) = \frac{1}{1+e^{-0.295}} = 0.5732$$

0.5732

Figure 9: Convolution Illustration - Step 2

STEPS NUMBER 3-6

The procedure described in steps 1 and 2 is repeated for 3, 4, 5 and 6. Every time the filter "slides" one position below. In Step 6, the filter performs the operation on values 6 to 8, so the filter has sled over the entire first channel of the input channels. The output is shown in figure 8:

0.5617
0.5732
0.6283
0.5903
0.5387
0.5582

Figure 10: Convolution Illustration - Output from Steps 1-6

STEPS NUMBER 7-18

After the first filter has “sled” over the entire first channel of the input sample, then the second filter will do the same with the second channel and the third filter will do the same with the third and final channel of the input sample, each adding one more channel to the output.

STEPS NUMBER 18-90

After the three input channels have been convoluted by the first three filters, then the same procedure will be repeated between the three input channels and the second three filters. So, the first input layer will be convoluted by the fourth filter, the second with the fifth etc. Thus, in the end the three input channels will be convoluted by all the filters in the same way. So, the output, also called the feature map, will have the size of the output in Step 6 and the number of channels of the output will be determined by the number of filters (15 filters), resulting in a 6x15 output.

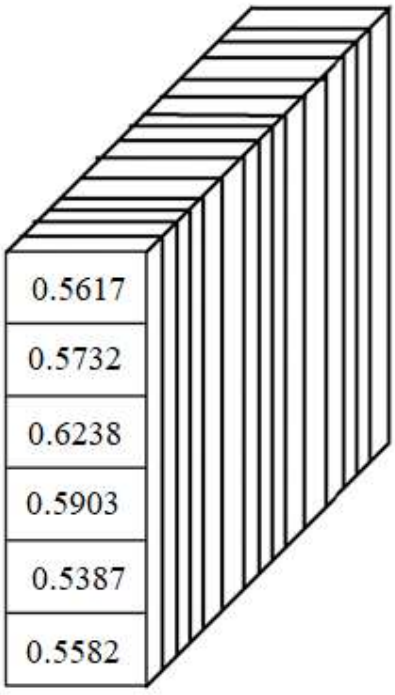

Figure 11: Convolution Illustration - Output

2.3.3. POOLING LAYER

Several successive Convolutional Layers may result in an output, the size of which makes the training of the model (especially in the fully connected layers) inefficient. In the previous example, the Convolutional Layer's input is a 8x3 sample, thus consisting of (8x3=)24 values, whereas the Convolutional Layer's output is a 6x15 sample, thus consisting of (6x15=)90 values. This may be deemed as an insignificant difference which will not affect the efficiency of the model. However, it causes a very significant chain affect, especially if there are several successive Convolutional Layers. The bigger the samples, the bigger the layers in order to process the samples and the bigger the layers, the more the weights and biases, which will have to be optimized by the optimization function in order to minimize the loss function. For that reason, it is very common for Convolutional Layers to be succeeded by Pooling Layers which aim to the down-sampling of the sample.

There are various types of Pooling Layers such as Max-Pooling, Min-Pooling and Average-Pooling etc. Suppose that we wanted to down-sample the output of the Convolutional layer which was used as an example in the previous section (shown in figure 11). We could use a Max-Pooling layer using pooling filters of size 2 and stride 1.

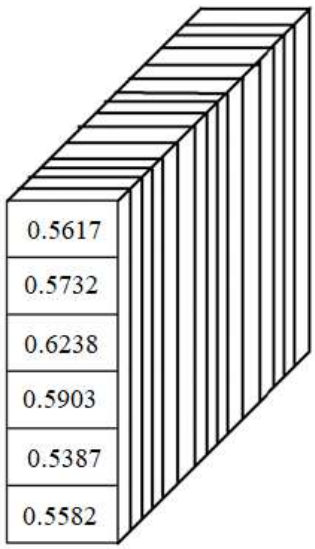

Figure 12: Pooling Illustration - Input

STEP NUMBER 1

Since we are using pooling filters of size 2, the first two values of the first channel will be combined by keeping the maximum of the two. So the first value of the output will be max(0.5617,0.5732)=0.5732.

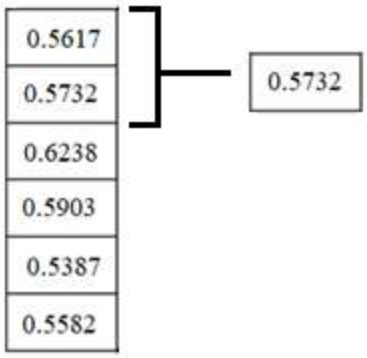

Figure 13: Pooling Illustration - Step 1

STEP NUMBER 2

Since the stride is one, the pooling filter will slide one position and thus the second and the third values will be combined by keeping the maximum of the two. So the second value of the output will be max(0.5732,0.6238)=0.6238.

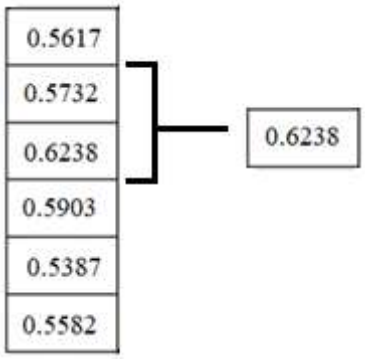

Figure 14: Pooling Illustration - Step 2

STEPS NUMBER 3-5

The above procedure will be continued for the third and fourth, the fourth and fifth, the fifth and sixth values respectively and combine the values of the first channel by keeping the maximum of the two. The result of the Pooling Layer on the first channel of its input and thus the first channel of the output is the following.

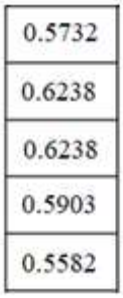

Figure 15: Pooling Illustration - Steps 3-5

STEPS NUMBER 6-75

Steps 1-5 will be repeated for the second, the third, the fourth and so on channels until it is repeated for the fifteenth and last channel. The output of the Pooling Layer is the following.

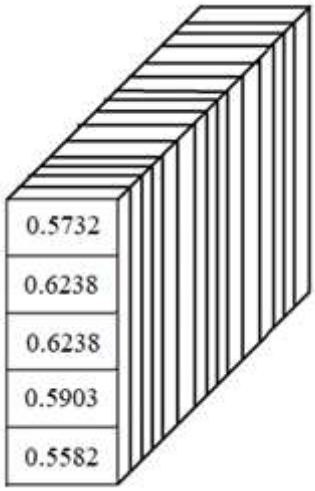

Figure 16: Pooling Illustration - Output

The output of the Pooling Layer does not look a lot different from the input and it is not. The parameters (filter size and stride) used were the minimum which resulted in a very slight down-sampling. The Pooling Layer's input was of size 6 and 15 channels, for a total of 90 values, whereas the Pooling Layer's output was of size 5 with 15 channels for a total of 75 values.

2.3.4. CONVOLUTIONAL AND POOLING LAYER PARAMETERS EFFECT

Designing a basic Artificial Neural Network is not very difficult. However, when designing an Artificial Neural Network, it is essential to understand the exact effect of the parameters, which is very challenging. Going back to the Pooling Layer illustration, if we were to use the same input (6, 15 channels, total 90 values) but increase the Pooling Layer's filter size from 2 to 3 and the stride from 1 to 3, then the output of the Pooling Layer would be of size 2 with 15 channels for a total of 30 values. So, a slight change of the filter size and stride resulted to an output less than half the initial.

For both Convolutional and Pooling layers, a bigger stride parameter will result to a smaller output. The same is true for the filters' size as a bigger filter parameter will result to a smaller output. Last but not least, the number of filters used by the Convolutional layer will not affect the size of the output, but its channels, as the output has the same number of channels as the number of filters used by the Convolutional layer.

2.4. WORD-TO-VECTOR REPRESENTATION

One of the many challenges when undertaking a Machine Learning project where the input is in Natural Language is that there needs to be a conversion of the text to a machine-understandable form. This is usually in the form of vectors. However, the methods of converting a word to a vector are various.

2.4.1. BAG-OF-WORDS (B-O-W) MODEL

The standard representation of a document is called Bag-of-Words. In Bag-of-Words, each word is represented by a vector of identifiers. These identifiers can be a dictionary index, the frequency of the word in the text or others. The Bag-of-Words model is based on the assumption that the order of the words in the text is unimportant and thus the text can be represented as a bag of words (Manning, et al., 2009). An alternation of the Bag-of-Words model is the n-gram model

where instead of splitting the text to single words and each word being represented by a vector, the text is split in groups of n words, where each word is converted to a vector with the n next words (in order) in the text. The difference between the models can be better illustrated with the following example.

Let the sentence be: “I am Angelos and I am studying Computer Science”. The Bag-of-Words model would disregard the order of the words and the sentence could be split in the following way, [“angelos”, “computer”, “science”, “i”, “studying”, “am”, “and”]. The exact order of the words depend on the identifiers and could be, for example, in ascending dictionary index. On the contrary, the split of the sentence in the 3-gram representation of the sentence could be [“I am studying”, “studying Computer Science”, “and I am”, “am Angelos and” ...]. Thus, n-gram in comparison to bag-of-words achieved some preservation of the words’ order in the sentence, which may be very useful, but may also be utterly useless depending on the task in question.

2.4.2. CONTINUOUS BAG-OF-WORDS (B-O-W) vs CONTINUOUS SKIP-GRAM MODEL

In 2013, a researching team led by Tomas Mikolov introduced word2vec (Mikolov, et al., 2013) and revolutionized word-to-vector representation. They introduced two different model architectures for computing the vector representation called Continuous Bag-of-Words model (or C-B-o-W) and continuous Skip-gram model. Both models were tested in word similarity tasks and demonstrated greater accuracy than previous techniques.

The difference between the two models lies on their inputs and outputs. In Continuous Bag-of-Words, the input of the model is the context of the word and the output is the word. Let the word be w_t, then its context would be the words before it, w_{t-2} and w_{t-1} and the words after it w_{t+1} and w_{t+2}. So, the input of the model would be w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} and the output would be w_t. Hence, the continuous bag-of-words model can be thought as predicting the meaning (thus its vector representation) of a word based on its context.

In Continuous Skip-gram, the procedure is exactly the opposite. The input of the model is the word and the output is the context of the word. So, considering the example for bag-of-words,

the input of the model would be w_t and the output would be w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}. Hence, the Skip-gram model can be though as predicting the context of a word based on the word.

In both models, the size of the context of the word (either that is the output or the input of the model) is set as its window size. In the example above, the window size is 2 (two words before and two words after).

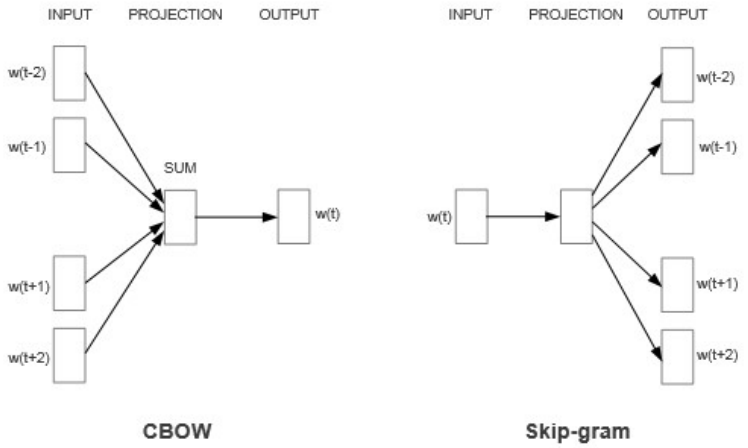

Figure 17: Continuous Bag-of-Words model vs Continuous Skip Gram model (Mikolov, et al., 2013)

2.5. RELATED WORD: MACHINE LEARNING IN EDUCATION AND AUTOMATED ESSAY MARKING

As mentioned, Machine Learning is one of the important and interesting field which has not been adequately investigated is Automated Essay Marking in order to produce a trustworthy system. There have been a few attempts, which however have not been widely put to work in the Education System. The first research made on Automated Essay Scoring dates back to 1967 when Ellis Batten Page, known as the father of Automated Essay Scoring, published his research with the title “"Statistical and linguistic strategies in the computer grading of essays” (Page, 1967) and later the same year he, also, published "The imminence of grading essays by computer". He, then, went on to publish “The analysis of essays by computer” and “The Use of the Computer in Analyzing Student Essays”. Although his work was revolutionary, it was not developed until the 1990s, because of the high cost and low performance of (even high-

performance) computers at that time. His work was purchased by Measurement Incorporated in 2002.

Measurement Incorporated's current Project Essay Grade ((PEG), n.d.) is one of the market's most popular solutions and the only one that is used as a sole scoring system by one state in the US. It is on the other hand, only used by one state in the US. PEG has been also introduced by three other but only as a counseling assessment tool and essays are also marked by human raters, in comparison to be used as the sole assessment tool. In 2011, more than 7000 students were incorrectly graded by Measurement Incorporated's PEF on the standardized test required for admission to most New York City private schools (Nir, 2011). Thus, it is obvious that the status quo is far from a widely-used sole assessment solution to Automated Essay Marking.

Other attempts which are used by the general public are eRater (eRater, n.d.), Intellimetric (Intellimetric, n.d.) and PaperRater (PaperRater, n.d.). For obvious, copyright and intellectual property, reasons, all four are quite vague or not willing to disclose their exact methods of assessment. However, they mention some of the features they use such as "fluency, diction, grammar and construction" ((PEG), n.d.) or "content analysis based on vocabulary measures, lexical complexity/diction, proportion of grammar errors, proportion of usage errors, proportion of mechanics errors, proportion of style comments, organization and development scores, features rewarding idiomatic phraseology" (eRater, n.d.). However, the most interesting disclaimer is the one from PaperRater which was obtained when I attempted to mark part of this section, "Grade: 92 A. The grade above is NOT complete! We do not actually use a crystal ball to generate your grade. Instead, this grafde takes into account spelling, grammar, word choice, style, vocabulary, and more; but it does NOT examine the meaning of your words, how your ideas are structured, or how well your arguments are supported. We should also mention that our automated grader does not always get things right. So, please consider this grade to be one facet of your paper's overall grade." It can be deduced from the above disclaimers that all commonly used Automated Essay Marking software solutions limit their function to classifying essays based on features that are, in my opinion, secondary. An essay, in any school grade, is assessed primarily based on the arguments made and their support, and secondarily based on their grammatical and syntactical performance. As a result, the existing solutions to Automated Essay Marking are inadequate and do not effectively address the problem.

What differentiates this project to other related attempts is that it does not specify the exact features to be investigated (such as word count, large-words count, technical words count, grammatical and syntactical mistakes). It is using small to medium-sized neural networks with some parameter tuning and attempts to classify the essays using local indicators that will map an essay to a class/grade membership.

3. APPROACH

As already mentioned in the "Introduction", the aim of the project is to investigate whether a Machine Learning approach and more specifically an Artificial Neural Network is able to classify a set of essays after being trained by a subset of them. In this section, I will be discussing the design of the Convolutional Neural Network, as well as the design of the preparation of data and the word-to-vector conversion before it is inserted to the network.

3.1. BUSINESS MODEL OF THE SOFTWARE

Before deciding the approach to take, it is vital to think about the aims of the project. In the first place, this project has been developed for the purposes of Scientific Research. It is not destined for commercial use or profit, so a user-friendly graphic user interface (GUI) was not a priority. For the same reason, the project consists of different software versions, whose differences together with their performance evaluation will be vital in evaluating the project and determining the project's next step or future work. The short time frame and other "adverse conditions", which will be discussed in the "Conclusions", make the primary aim of this project to conduct an investigation and a satisfactory starting point rather than a complete and successful software. Although it has not been developed for commercial use, it has been developed so that a user with minimum developing knowledge can train it and evaluate it.

3.2. GENERAL STRUCTURE OF THE SYSTEM-DATA FLOW

Although, we can summarize the software as a Convolutional Neural Network that performs text classification, it consists of various components and subcomponents out of which only one is the Convolutional Neural Network. The data flows through the components from the 1st to the 2nd until it reaches the 3rd and final component and is outputted to the user. The aggregate input of the system is a tsv file containing the essays and the labels in a very simple form whereas the aggregate output of the system is the set of probabilities that each instance belongs to each class together with some evaluation metrics of the training such as Accuracy.

Even the Convolutional Neural Network component of the system can be thought of as a component which consists of two smaller subcomponents. A list of the general structure of the software is provided below:

- 1st Component: Training/Evaluation Set Preprocessing
- 2nd Component: Word To Word Embedding (Word Vectors) Conversion
- 3rd Component: Convolutional Neural Network
    - 1st Sub-Component: Feature Extraction (Convolutional and Pooling Layers)
    - 2nd Sub-Component: Classification (Fully Connected Layers)

3.2.1. 1st COMPONENT: DATASET PREPROCESSING

This first component is responsible for preprocessing the dataset. It is a program coded in Python. The input of the program is the tsv file containing the full dataset and the output of the program is three python lists containing the essays (one for the training, one for the testing set and one for both training and testing) and two python lists containing the labels/grades of each essay (one for the training and one for the testing set). After running the training set on a various different Convolutional Neural Networks of different depths, I came to the realization that regardless the depth of the Network, the classifier was mapping/classifying all essays of the testing set to grade 8. After investigating the dataset in depth, I realized that the training set was consisted of a very high number of essays graded/labeled as 8. Although it is expected for the grades to follow a normal distribution, the frequency of essays classified as 8 were clearly distorting the grades distribution. More specifically, about 39% of the essays in the training set were classified as 8 as can be seen in the tables below. As a measure of comparison, the second most frequent grade/label was 10 with 18%.

FULL DATASET		
GRADE	NUMBER OF INSTANCES	FREQUENCY
2	9	0.514%
3	1	0.057%
4	17	0.971%
5	15	0.857%
6	110	6.286%
7	133	7.600%

TRAINING SET		
GRADE	NUMBER OF INSTANCES	FREQUENCY
2	5	0.571%
3	1	0.114%
4	7	0.800%
5	9	1.029%
6	60	6.857%
7	73	8.343%

8	675	38.571%	8	338	38.629%
9	329	18.800%	9	149	17.029%
10	308	17.600%	10	156	17.829%
11	108	6.171%	11	52	5.943%
12	45	2.571%	12	25	2.857%

Initially, I decided to implement a threshold (maximum number of instances with each grade at the training set) mechanism so that the training set would consist of roughly the same number of instances for each grade/label. However, in order for grade 8 to have a comparable number of instances with grade 2 or 3, the threshold should be as low as 70 instances. Thus, similarly to economics and maximum prices, this would lead to training set "shortage" as the training set would no longer be large enough to train the model.

So, I decided to modify the grading system of the dataset, group instances with close grades increasing their aggregate number of instances and then apply a threshold, resulting to the dataset below.

FULL DATASET			TRAINING SET		
GRADE	NUMBER OF INSTANCES	FREQUENCY	GRADE	NUMBER OF INSTANCES	FREQUENCY
2,3,4,5	42	2.40%	2,3,4,5	35	4.90%
6,7	243	13.89%	6,7	150	20.98%
8	675	38.57%	8	150	20.98%
9	329	18.80%	9	150	20.98%
10	308	17.60%	10	150	20.98%
11,12	153	8.74%	11,12	80	11.19%

The results of the above dataset modification were more than satisfactory and will be analyzed thoroughly in "Results and Evaluation".

3.2.2. 2nd COMPONENT: WORD TO WORD EMBEDDING (WORD VECTORS) CONVERSION

This second component is responsible for converting the lists of essays "transferred" from the first component to a list of vectors. The input of this component is the five Python lists from the first component file containing the essays and the labels/grades. The output of this file is the

word embeddings (word vectors). Although, the input is the same for all versions of the software, the output is different as two different word-to-vector representations have been implemented. More information about the two different word-to-vector representations are provided in the "Word-To-Vector" section which follows.

3.2.3. 3rd COMPONENT: CONVOLUTIONAL NEURAL NETWORK

The third and final component of the software is the Convolutional Neural Network. It is Python file but it also imports Tensorflow among others. More information about Tensorflow is provided in the "Implementation" section. The input of this component is two numpys arrays of the word embeddings (one numpy array containing the training set and one numpy array containing the test set) and two numpy arrays containing the grades/labels (one numpy array containing the grades/labels of the training set and one numpy array containing the grades/labels of the test set). This component is also different between the versions, since different Convolutional Neural Networks were constructed, trained and tested in order to evaluate their performance.

Although this component is one file and can be considered as one Artificial Neural Network, for the purposes of this components' analysis, it is wiser to further divide this component into two smaller subcomponents.

1st SUBCOMPONENT: FEATURE EXTRACTION

Feature Extraction is achieved by a series of Convolutional and Pooling Layers, whose purpose is to extract the necessary features for the classification. So, the subcomponent's input is the NumPy arrays and the output is a modified tensor, whose size and modification varies depending on the version.

2nd SUBCOMPONENT: CLASSIFICATION

Classification is achieved by two fully connected layers. So, the subcomponent's input is the tensor from the first subcomponent and the output is a tensor containing the probabilities of the training or testing example being in each of the classes. This subcomponent is the same in all versions.

3.3. WORD-TO-VECTOR

The aim of word-to-vector is to take as input the set of 1750 essays and convert them to something that can then be inserted to the neural network. In other words, we are trying to achieve Semantic Parsing, which is "the task of mapping natural language sentences to formal representations of their underlying meaning" (Grefenstette, et al., 2014).

We are going to do so by splitting the essays to words and then converting the words to vectors. The vectors will then be padded to have the same size. The vector representation can be done in many different ways depending on the objective of the neural network. The elements that compose the vector represent features of the word. When the essay is inserted in the Artificial Neural Network either as a training or testing example, the elements of the vectors (or features of the words) are the channels which were described in the "Samples Representation" section in the "Background".

In this project, I have implemented two different vector representations, the results of which are going to be analyzed in the "Results and Evaluation" section.

3.3.1. FIRST WORD-TO-VECTOR REPRESENTATION

The first representation describes each word by two features/indicators and so converts each word into a two-dimensional vector. After collecting all distinct words from all essays with frequency more than 2, a dictionary of the words is constructed. The reason there is a minimum frequency of 2 required is that a word which exists only once in one essay has no use as it cannot be used as a class indicator (as it cannot be used as a similarity instance of two essays). So, the first dimension of the vector is the index of the word in the dictionary. The second dimension of the vector is the frequency of the word in the essay. If the vector representation of the word "network" is [7,3], this indicates that the index of the word "network" in the word dictionary is 7 (in other words, the word "network" is the 7^{th} word in the dictionary) and that the word "network" is repeated 3 times in the essay. The exact implementation will be analyzed in the section "Implementation".

3.3.2. SECOND WORD-TO-VECTOR REPRESENTATION

The second representation describes each word by 300 features/indicators and so converts each word into a 300-dimensional vector. For this representation, I used the pre-trained Google News corpus, which can be download from GitHub (Mihăltz, 2016). It is word-to-vector model which consists of 3 billion words and phrases was trained on a vast Google News training dataset using the Continuous Bag-of-Words which was described in the "Word-To-Vector Representation" section in the "Background".

3.3.3. COMPARISON OF THE TWO WORD-TO-VECTOR REPRESENTATIONS

Those two different vector representations allow us to investigate two extreme solutions of the same problem. In the first representation, the input of the Convolutional Neural Network is the simplest representation possible. A major drawback of the first representation is that it does not consider the similarity of the words or their meaning, as the dictionary is sorted alphabetically and two similar or even synonym words may have entirely different dictionary indexes. On the second representation, the input of the Convolutional Neural Network is a lot more complex. The 300 features for each word may not be enough to completely capture the meaning of the word. However, they capture the meaning of the words a lot more effectively than the two features in the first representation. Another difference between the two word-to-vector representations is that since the second word-to-vector captures the correlation between two non-equal words, it doesn't use a minimum frequency of words as two similar words, but not equal, can be used as class indicators. So, in the second word-to-vector representation, the bag-of-words representation of the essays is larger. It is, thus, obvious that they allow us to see the effect of the vector representation of the words to the classification by comparing their performance results.

3.4. ARTIFICIAL NEURAL NETWORK APPROACH

Different tasks require different Convolutional Neural Network structures. However, the required structure is only discovered after the Convolutional Neural Network has been implemented and trained. Other than the task, the amount of data available for training is also a factor that affects the required structure. A demanding task will require a large number of training data, but also an

Convolutional Neural Network deep enough in order to adequately describe the patterns in the training data.

The following figure graphically illustrates three different models for describing the same time-series (a plot of a value against time).

Figure 18: Model Underfitting/Overfitting (edited) (Bhande, 2018)

Starting from left to right, in the first case, the model is not deep enough in order to adequately represent the training data and capture the underlying pattern of the data. In the second case, the model is too deep for the size of the training data, which results to the model being too closely correlated with it. In the third case, the model is deep enough to adequately represent the patterns of the data but not too deep to fit the data too closely. It is important to notice that model overfitting may seem "harmless" but an overfitted model will fail to predict the testing data even if it obeys the pattern. The objective of any Machine Learning approach is to achieve the third fit by attaining a balance between model overfitting and underfitting.

3.4.1. ARTIFICIAL NEURAL NETWORKS STRUCTURE

Although the exact Convolutional Neural Network structure differentiates between the versions, the general layers' structure is constant. As already mentioned, the Artificial Neural Network implemented can be divided to two different, independent, chain-connected components; the feature extraction component and the classification component. Their layers' structures are provided bellow.

3.4.1.1. FEATURE EXTRACTION COMPONENT

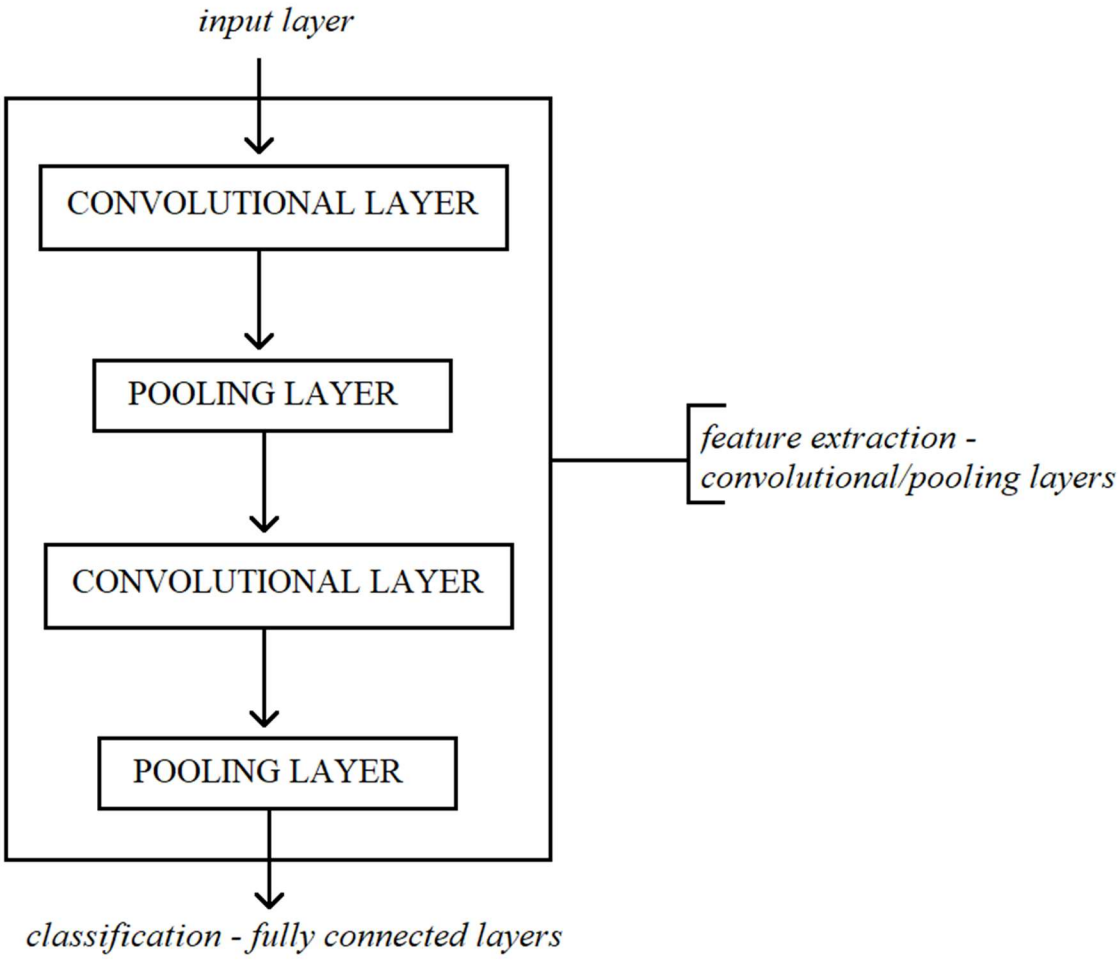

Figure 19: Artificial Neural Network's Feature Extraction Component

The Feature Extraction component succeeds the Input layer and precedes the Classification component. The Feature Extraction component is consisted of two Convolutional layers, each succeeded by a Pooling layer. The Convolutional and Pooling layers' function was described in the "Convolutional Neural Network" in the "Background". Each training sample is first convoluted by the first Convolutional layer, then down-sampled by the first Pooling layer, then convoluted by the second Convolutional layer and finally down-sampled by the second Pooling layer. Depending on the version, the Convolutional layer may use 1-dimensional or 2-dimensional filters and so will the Pooling layer. The samples are in 1-dimensional or 2-dimensional form, but since there are multiple channels, they can be considered 2-dimensional or

3-dimensional respectively. However, the Classification component can only process and classify 1-dimensional samples. Thus, before the sample is transferred to the Classification component, it is modified by a flattening "layer", which flattens the sample, by converting it from 2-dimensional or 3-dimensional form to 1-dimensional, by adding up their dimensions and channels. For example, a 3-dimensional sample of 10 6x6 channels will be converted to 1-dimensional with 1 (10x6x6) 360 values channel. However, this is not an actual Artificial Neural Network layer and thus has not been included in the above graph.

3.4.1.2. CLASSIFICATION COMPONENT

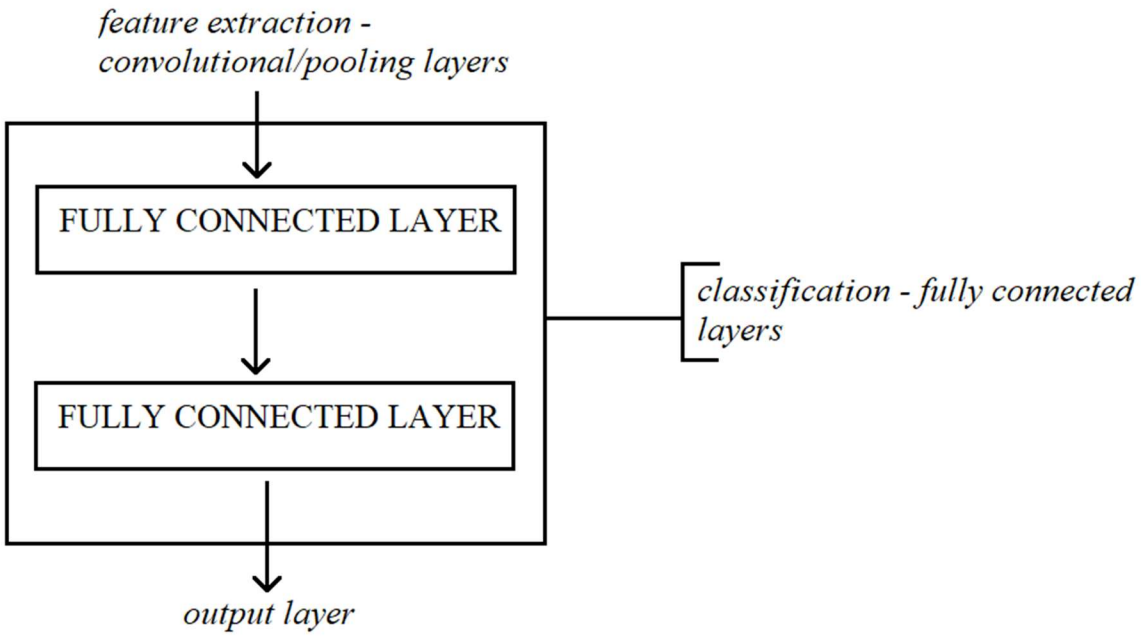

Figure 20: Artificial Neural Network's Classification Component

The Classification component's structure is identical in all Versions of the software. However, the number of neurons in each layer vary. After the necessary features have been extracted by the Feature Extraction component, they are transferred to the Classification component which will use them to classify the sample. Fully Connected layers are simple Artificial Neural Network layers, whose name is due to the fact that all neurons of the previous layer are connected to each neuron of the Fully Connected layer. The first Fully Connected layer is consisted of 1024 or 2048 neurons, each of which is connected to the 6 neurons of the second Fully Connected layer. So, after the Feature Extraction component has extracted the necessary features, those features

will then be weighted and accumulated to 1024 or 2048 neurons of the first Fully Connected layer, which will in turn be accumulated to 6 neurons of the Fully Connected layer. The number of neurons in the first Fully Connected layers are adjustable and subject to judgment. However, the number of neurons in the second Fully Connect layers are unmodifiable and must be equal to the number of classes (thus, 6). The second Fully Connected layer contains an unnormalized set of probabilities of the instance to be in each class. The reason there is a layer afterwards, the output layer, is that the set of probabilities must be normalized using an activation function (predominantly the SoftMax function) in order to map the unnormalized set of probabilities with range from minus infinite to infinite to a normalized set of probabilities with range from 0 to 1.

3.4.2. ARTIFICIAL NEURAL NETWORKS LOSS-OPTIMIZATION

As stated in the "Artificial Neural Network Optimization" section in the "Background", the Artificial Neural Network calculates a loss after training or evaluating a batch of examples. The Artificial Neural Network's optimization function will then recalculate the weights and biases in order to recalibrate the model and minimize the loss. The exact loss and optimization functions to be used depend on various parameters, the most important of which is the classification task.

The loss function used is called Cross-Entropy Loss or Log Loss and was fully explained in the "Artificial Neural Network Optimization" section in the "Background". The main motivation of using it is that the classification task of this project is a multiclass classification, as there is a total of 6 grades from A to F, and Cross-Entropy is the typical loss function for multiclass classifications.

The optimization function used is called Stochastic Gradient Descent (or SGD). Stochastic Gradient Descent is a stochastic approximation of the batch Gradient Descent algorithm of the Gradient Descent algorithms family which we described in the "Artificial Neural Networks Optimization" section in the "Background". Batch Gradient Descent sums the gradients $L'(w,b)$ of the Loss function $L(w,b)$ for the entire batch of training examples and then makes an update to optimize the loss. On the contrary, Stochastic Gradient Descent uses the gradient $L'(w,b)$ of the Loss function $L(w,b)$ for one training example and updates the weights. As a result, Stochastic Gradient Descent is a lot faster and requires less memory as it does not store the entire

gradients for the entire batch (Renals, 2016). The Stochastic Gradient Descent's efficiency advantage was the main motivation of using it as the memory capabilities were limited. The pseudocode of the Stochastic Gradient Descent or SGD is provided below.

```
1: procedure SGDTRAINING(X, T, W)
2:     initialize W to small random numbers
3:     randomize order of training examples in X
4:     while not converged do
5:         for n ← 1, N do
6:             for k ← 1, K do
7:                 y_k^n ← Σ_{i=1}^{d} w_ki x_i^n + b_k
8:                 δ_k^n ← y_k^n − t_k^n
9:                 for i ← 1, d do
10:                    w_ki ← w_ki − η · δ_k^n · x_i^n
11:                end for
12:                b_k ← b_k − η · δ_k^n
13:            end for
14:        end for
15:    end while
16: end procedure
```

Figure 21: Stochastic Gradient Descent Pseudocode (Renals, 2016)

The Stochastic Gradient Descent algorithm will first assign random small numbers for weights w (and thus biases b, since $b_k = w_k \cdot (+1)$). Then it will slowly change the weights until loss converges to zero. The rate of change of the weights is called the learning rate and is the variable η of the above pseudocode. The higher the learning rate the bigger the change of the weights w after each iteration. The learning rate used was 0.001.

3.5. VERSION-SPECIFIC APPROACH

Artificial Neural Networks are generally easy to develop. However, their parameter tuning is very challenging. Given a version and its results, only (calculated) speculations can be made to deduce what went wrong or what went right. The only way to be absolutely sure that a version can be improved in one way or another is to implement and test both ways. For the purposes of this project, more than one hundred models were implemented. However, many of the models implemented were not trained as the GPU capacity of both the Linux lab machines and the insista@cs.cf.ac.uk server machines were inadequate. The following 6 versions are some of the most important models implemented, trained and evaluated. Some of them had great results, while the results of other were discouraging. However, all of them were instrumental in the general approach as they gave invaluable feedback about the next step. The "Version-Specific

Results", which have been used and referred to at this section can be found at the "Results and Evaluation" section. Figures depicting the exact Artificial Neural Network structure as well as the input and output of each layer used in each version can be found in the "Version-Specific Artificial Neural Network Structure Figures", which follows.

3.5.1. VERSION 1.0

As word to word embeddings method, Version 1.0 used the first word-to-vector representation in which each word is represented by a 2-dimensional vector, where the first dimension is the dictionary index of the word and the second dimension is the frequency of the word in the essay.

The essays are inserted into the network and then converted to a 2-dimensional representation. More specifically, each 363-word essay (all essays have the same size as a result of the padding) is converted to a 33x11 tensor (matrix). The reason is that, since Convolutional Neural Networks are mostly used for image classification which are 2-dimensional, it will allow us to see how the Convolutional Neural Network will perform treating the essays as images.

The feature extraction component of the Artificial Neural Network is consisted by two 2-dimensional convolutional layers. The first convolutional layer uses 64 3x3 filters, whereas the second uses 128 3x3 filters. Both convolutional layers are using sigmoid as the activation function and padding in order the output of the convolutional layers to have the same size as the input. Stride has not been manually set, thus it has been automatically set to the default value 1. The two 2-dimensional convolutional layers are each followed by a 2-dimensional pooling layer. Both pooling layer use 2x2 filters with stride 2. The output of the second pooling layer with size 8x2 and 128 channels is flattened by a flattening "layer" to a (8x2x128=) 1-dimensional tensor of size 2048.

The classification component is consisted by two fully connected layers. The first layer is consisted by 1024 nodes and is using sigmoid as activation function. The second and last layer is consisted by 6 nodes, as it is dictated by the number of classes.

3.5.2. VERSION 2.0

The results of version 1.0 were more than encouraging. As a result, I decided to use the first word-to-vector representation again. However, this time, the essays will not be converted to 2-dimensional representation. In order to effectively compare the two essay representations, we are going to use similar parameters in feature extraction, whereas classification will be identical.

The feature extraction component of the Artificial Neural Network is consisted by two 1-dimensional convolutional layers. Both convolutional layers are using filters of size 9 (again 64 and 128 filters respectively), sigmoid as the activation function and stride 1. Again, the two 1-dimensional convolutional layers are each followed by a 1-dimensional pooling layer. The first pooling layer uses filters of size 12 with stride 4 and the second pooling layer used filters of size 12 with stride 5. The output of the second pooling layer with size 16 and 128 channels is flattened by a flattening "layer" to a (16x128=) 1-dimensional tensor of size 2048. Thus, the feature extraction component in Version 1.0 and Version 2.0 both outputs a 1-dimensional tensor of size 2048.

As mentioned, the classification component was the same as Version 1.0.

3.5.3. VERSION 3.0

Both Version 1.0 and 2.0 has great results but it is doubtful that the first word-to-vector representation where each word is represented by a 2-dimensional vector can perform even better. As a result, Version 3.0 is using the second word-to-vector representation where each word is represented by a 300-dimensional vector. Although Version 2.0 had only slightly better results than 1.0, the 1-dimensional representation of word has been used like in Version 2.0.

As mentioned in the comparison of the two word-to-vector representations in the "Comparison of the two word-to-vector representations" section, since the second word-to-vector considers the correlation of words, it doesn't use minimum word frequency. As a result, the maximum size and as a result of the padding, all essays' size in the second word-to-vector representation is 676 (about twice the size of the essays in the first word-to-vector).

Like in Version 2.0, the feature extraction component of the Artificial Neural Network is consisted by two 1-dimensional convolutional layers, each followed by a pooling layer. However, as a result of the essays' size (about twice the size) and the word-vectors' dimensions (300 instead of 2), both convolutional and pooling layers need to be a lot "bigger". Both convolutional layers are using 300 filters of size 16, sigmoid as the activation function and stride 1. The first pooling layer uses filters of size 12 with stride 6 and the second pooling layer used filters of size 18 with stride 9. The output of the second pooling layer with size 11 and 300 channels is flattened by a flattening "layer" to a (11x300=) 3300-sized 1-dimensional tensor. Thus, the feature extraction component in Version 3.0 outputs a 3300-sized, compared to the 2048-sized 1-dimensional tensor output from the feature extraction component in Version 1.0 and Version 2.0.

The classification component is the same as in Version 1.0 and 2.0.

3.5.4. VERSION 4.0

On the one hand, the results from Version 3.0 were disappointing. On the other, it provided invaluable feedback for the next step, as it showed that the model was not big enough to successfully manipulate the input. This is clear by comparing Versions 1.0 and 2.0 to Version 3.0. In Version 1.0 and 2.0, the input of the feature extraction component is 363 2-dimensional vectors and the output is a 2048 1-dimensional tensor. In Version 3.0, the input of the feature extraction is 676 300-dimensional vectors and the output is a 3300 1-dimensional tensor. So, during this "compression" in the feature extraction, many features are mistreated and the classification component is unable to classify the essays effectively. Version 3.0 was designed on the false assumption that the 300 filters in the convolutional layers are enough and that the significant down-sampling by the pooling layers will not affect important features. As a result, Version 4.0 used the second word-to-vector representation, but also applied a combination of considerably bigger convolutional layers and considerably smaller pooling layers in the feature extraction part.

Like in Versions 2.0 and 3.0, the feature extraction component of the Artificial Neural Network is consisted by two 1-dimensional convolutional layers, each followed by a pooling layer. The

first convolutional layer is using 300 filters of size 3, sigmoid as the activation function and stride 1. The second convolutional layer is using 600 filters of size 3. Both pooling layers are using filter of size 5 and stride 2. The output of the second pooling layer with size 166 and 600 channels is flattened by a flattening "layer" to a (166x600=) 99600-sized 1-dimensional tensor. Thus, the feature extraction component in Version 4.0 outputs a 99600-sized, compared to the 3300-sized 1-dimensional tensor output from the feature extraction component in Version 3.0.

The classification component is the same as in Versions 1.0 to 3.0.

3.5.5. VERSION 5.0

The justification for the unsatisfactory results of Version 3.0 and thus the hypothesis for improvement in Version 4.0 proved valid. As a result, I decided to even more increase the size of the feature extraction part. Many attempts were made to increase the convolutional layers' size and also add a third one without, however, adding a third pooling layer which would reduce the size of the inputs (but not the channels). However, the GPU memory capacity of the Linux labs computers and the insista@cs.cf.ac.uk server provided by my supervisor was not enough to undertake a task like this. Thus, adding a third pooling layer but also increasing the existing pooling layers was the only way to add a third convolutional layer and see the effect of it.

Thus, the feature extraction part in Version 5.0 is consisted by three convolutional layers each followed by a pooling layer. The first convolutional layer is using 300 filters of size 3, sigmoid as the activation function and stride 1. The second convolutional layer is using 600 filters of size 3 and the third convolutional layer is using 1200 filters of size 3. The first pooling layer is using a filter of size 8 with stride 2, the second pooling layer is using a filter of size 12 with stride 2 and the third pooling layer is using a filter of size 16 with stride 4. The output of the third pooling layer with size 37 and 1200 channels is flattened by a flattening "layer" to a (37x1200=) 44400-sized 1-dimensional tensor. As expected, because of the third convolutional layer, the output has double the channels, but because of the addition third pooling layer and the increase of the other two, the output has significantly smaller size.

The classification component is the same as in Version 1.0 to 4.0.

3.5.6. VERSION 6.0

The results of Version 5.0 were disappointing but expected, as in order to increase the dimension/channels of the features the size of the features was decreased to a point that feature extraction became unsuccessful. As a result, I decided to reuse the feature extraction part of Version 4.0 which proved successful, but this time increasing the classification part instead of the feature extraction part. In Version 4.0, the output of the feature extraction part is of size 99600. This output is then propagated to the first fully connected layer which consists of 1024 neurons. It is possible than during this condensation, many important features are condensed excessively. Thus, I have increased the first fully connected layer from 1024 neuron to 2048.

3.5.7. VERSION 7.0

The improvement of Version 6.0 compared to Version 5.0 was triggered by making the first fully connected layer (classification part) bigger. Each of the neuron in the fully connected layer is connected to all the neurons of the last feature extraction layer consisting of 99600 neurons. As a result, adding 1024 neurons in the first fully connected weights results to (1024*99600=) more than 100 million extra weights to be optimized. However, 2048 neurons may still not be enough. Doubling the neurons to 4096 would still not solve the problem as the condensation from the first fully connected layer (consisting of 4096 neurons) to the second fully connected layer (consisting of 6 neurons) would make a successful classification impossible. The only way to successfully classify the 99600 features extracted from the feature extraction part (which seems to be successful) is to add a third fully connected layer. Adding a third fully connected layer would make the condensation of the 99600 features from the feature extraction more stable by dividing the condensation in two layers. The first fully connected layer would consist of 4092 neurons, the second fully connected layer would consist of 512 and the third and final fully connected layer would consist of 6. Version 7.0 has been implemented but could not be trained as the significant memory requirements of the model could not be satisfied by the limited memory capabilities available.

3.6. VERSION-SPECIFIC ARTIFICIAL NEURAL NETWORK STRUCTURE FIGURES

VERSION 1.0	VERSION 2.0
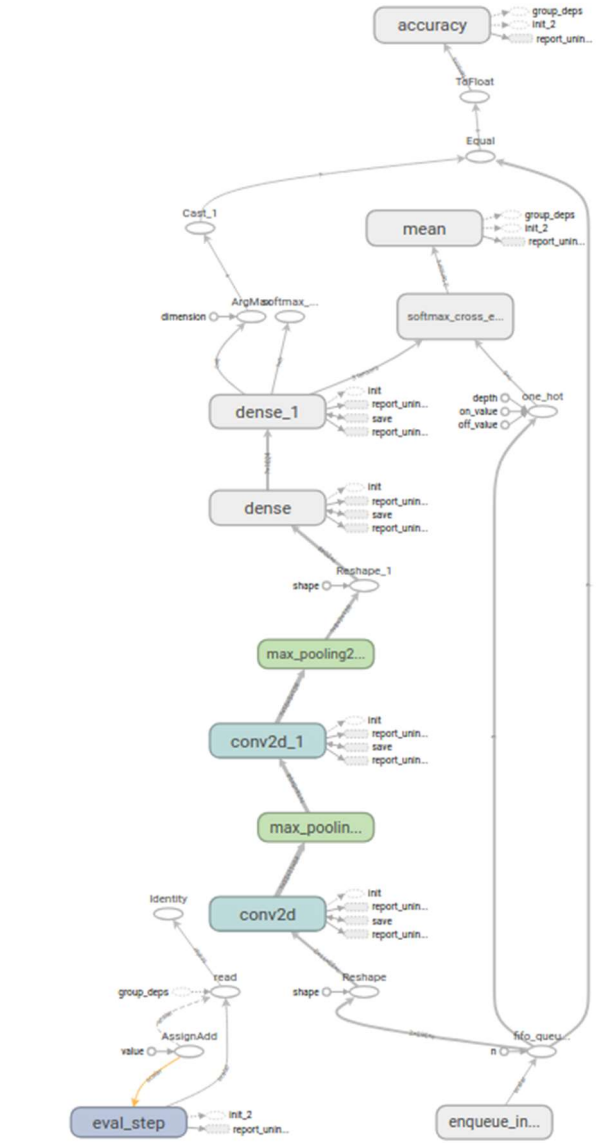	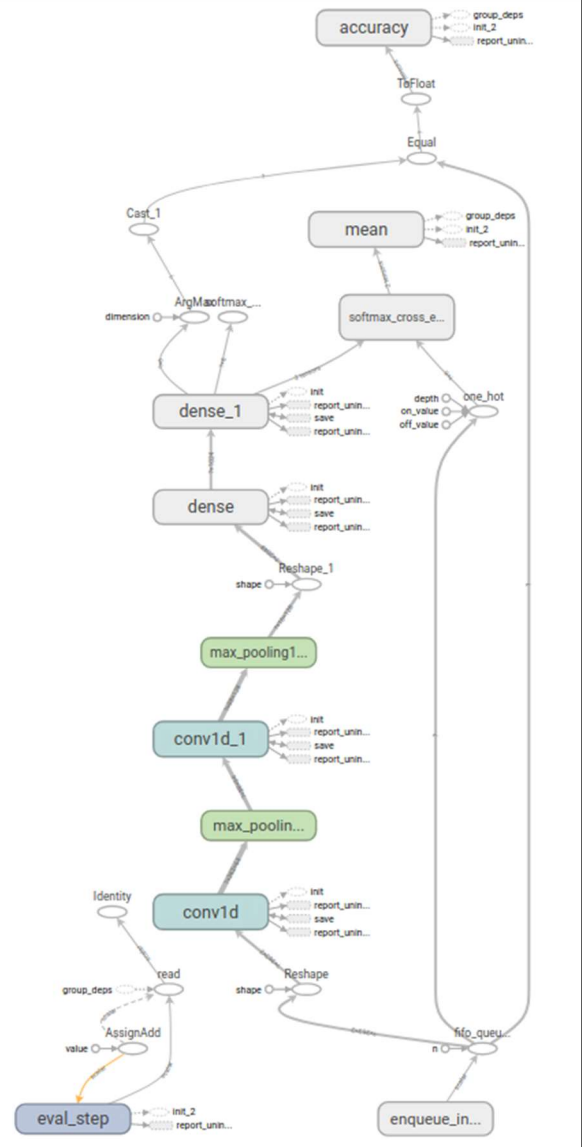
Figure 22: Version 1.0 Artificial Neural Network Structure	*Figure 23: Version 2.0 Artificial Neural Network Structure*

VERSION 3.0	VERSION 4.0
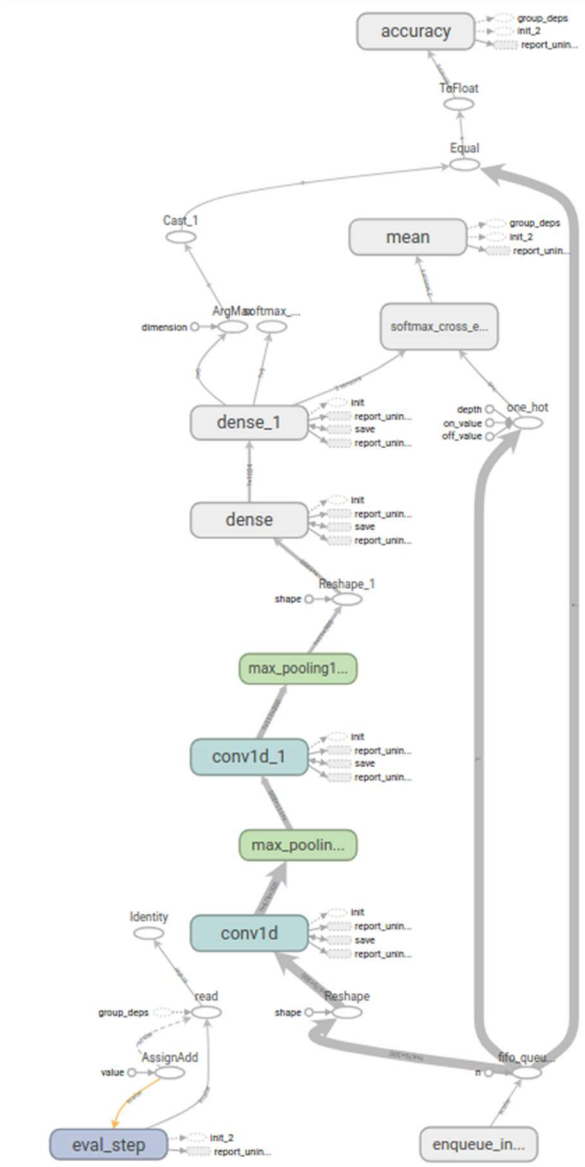	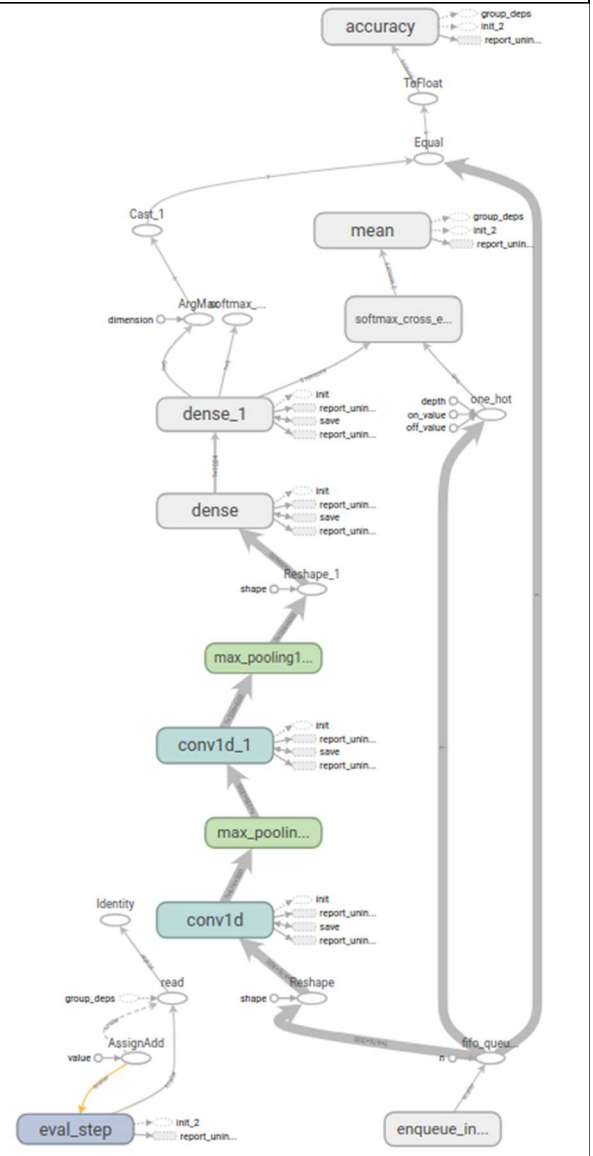
Figure 24: Version 3.0 Artificial Neural Network Structure	*Figure 25: : Version 4.0 Artificial Neural Network Structure*

VERSION 5.0	VERSION 6.0
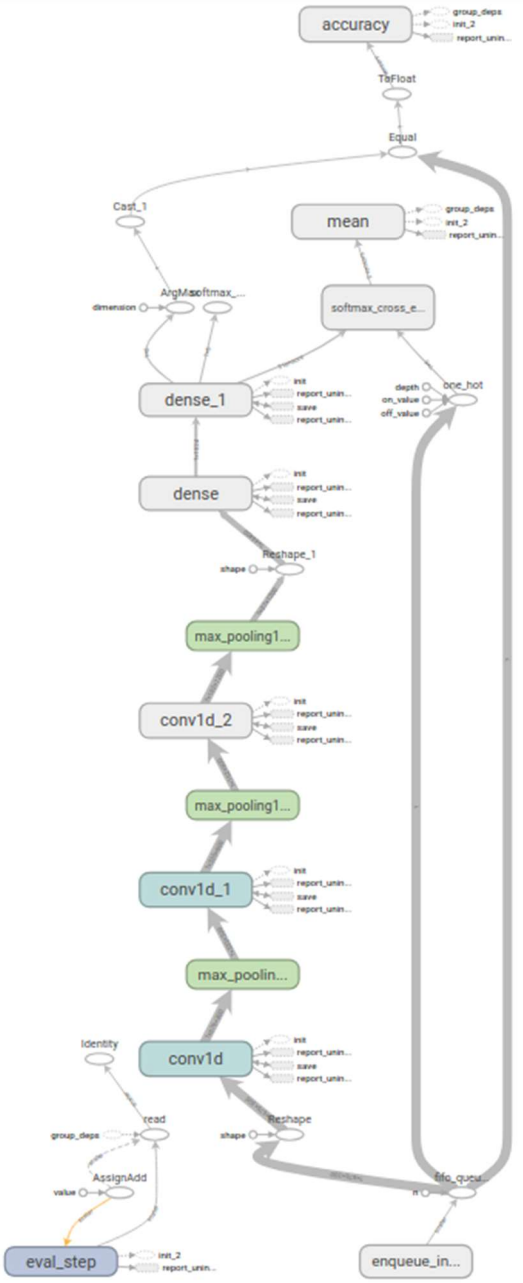	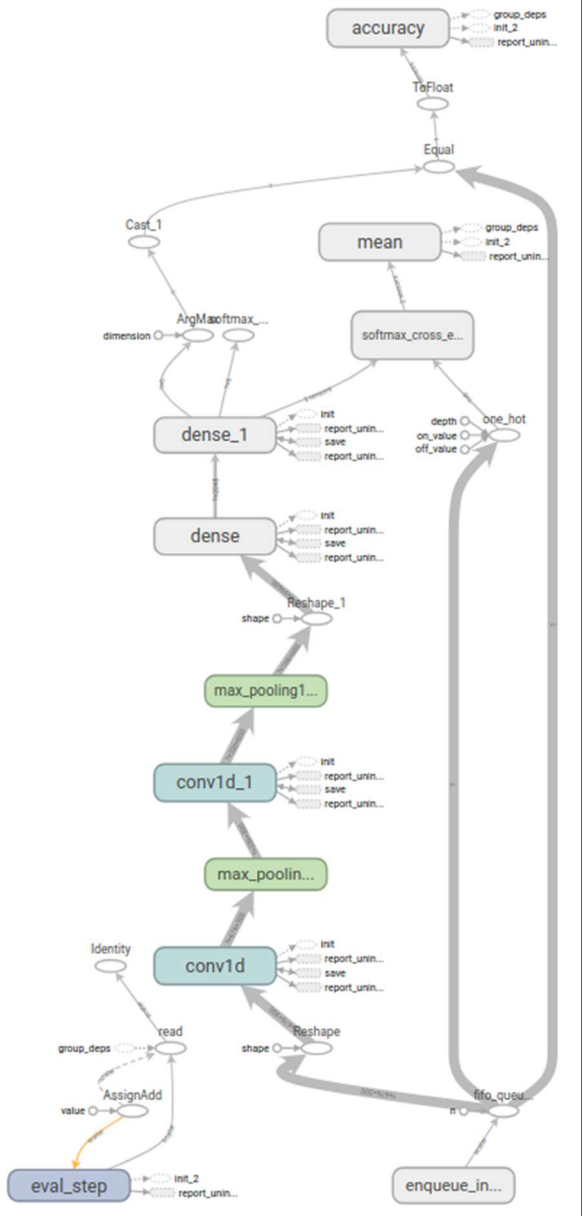
Figure 26: Version 5.0 Artificial Neural Network Structure	*Figure 27: Version 6.0 Artificial Neural Network Structure*

4. IMPLEMENTATION

Most of the models' implementation as well as the models' training and evaluation were performed on my laptop (main specifications using DirectX Diagnostic Tool: *System Manufacturer: Dell Inc., System Model: Inspiron 7548, Processor: Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz (4 CPUs), ~2.4GHz, Memory: 16384MB RAM, Card name: AMD Radeon R7 M270, Dedicated Memory: 4075 MB*, full specifications attached in the Appendix). I also used the Linux Lab machines and the insista@cf.cs.ac.uk server.

4.1. GENERAL IMPLEMENTATION

4.1.1. 1st COMPONENT: DATASET PREPROCESSING

The first component was implemented in Python, also importing the csv and itertools modules. Although, the csv module is mostly used for csv (comma-separate variable) files, it can be also used for tsv (tab-separated variable) files such as the dataset used. One of the challenged faced was to extract the necessary elements of the dataset which are the essays and their grades. The dataset consists of a line containing the columns (elements of each essay) and the rest of the essays occupy one line each. The itertools module was used in order to separate the various parts of the dataset and create lists containing the necessary elements without any unnecessary symbols such as quotation marks ("") or slashes (/) attached to the elements that would be a burden and cause problems over the next steps. An example of these problems would be the same word having two different dictionary indexes as a result of an unnecessary symbol attached to one of them.

```
05. with open ('training_set_rel3.tsv','r',encoding="cp1252") as tsvfile:
06.     reader = csv.reader(tsvfile, delimiter='\t')
10.     essay_section = itertools.islice(reader, 0, 1750)
11.     list_of_essays_labels = []
12.     list_of_essays = []
13.     list_of_labels = []
33.
34.     for row in essay_section:
35.         list_of_essays_labels.append(row)
36.
37.     yy=[2,6]
38.
39.     for i in range(len(list_of_essays_labels)):
40.         for y in reversed(range(len(list_of_essays_labels[i]))):
41.             if y not in yy:
42.                 del list_of_essays_labels[i][y]
43.
44.
45.
46. for i in range(len(list_of_essays_labels)):
47.     list_of_essays.append(list_of_essays_labels[i][0])
48.     list_of_labels.append(list_of_essays_labels[i][1])
```

Figure 28: Dataset Preprocessing: Essays/Labels Extraction

Figure 28 displays the essays/labels extraction from the dataset. After the text is divided to its columns and essays parts, a for loop goes through the essays part and stores each essay in a list. A second for loop removes all the unnecessary elements of the essays (grade given by each rater, essay set, essay number etc.) and only the essay text (index 2) and the grade agreed between the raters (index 6) are kept. A third for loop then makes two different lists, one for the essays and one for the grades.

The dataset grouping, which was mentioned in the "1st Component: Training Set Preprocessing" section in the Approach was implemented using the code in figure 29.

```
marks =[]
training_list_of_essays = []
training_list_of_labels = []
evaluation_list_of_essays = []
evaluation_list_of_labels = []

groups=[[["2","3","4","5"],35,"0"], [["6","7"],150,"1"], [["8"],150,"2"],[["9"],150,"3"],
[["10"],150,"4"],[["11","12"],80,"5"]]


# GRADES: 2-5/ 35 INSTANCES FOR TRAINING / NEW GRADE "0"
# GRADES: 6-7/ 150 INSTANCES FOR TRAINING / NEW GRADE "1"
# GRADES: 8/ 150 INSTANCES FOR TRAINING / NEW GRADE "2"
# GRADES: 9/ 150 INSTANCES FOR TRAINING / NEW GRADE "3"
# GRADES: 10/ 150 INSTANCES FOR TRAINING / NEW GRADE "4"
# GRADES: 11-12/ 80 INSTANCES FOR TRAINING / NEW GRADE "5"



for i in range(len(list_of_labels)):
    for y in range(len(groups)):
        if list_of_labels[i] in groups[y][0] and training_list_of_labels.count(groups[y][2])
<groups[y][1]:
            training_list_of_essays.append(list_of_essays[i])
            training_list_of_labels.append(groups[y][2])
        elif list_of_labels[i] in groups[y][0]:
            evaluation_list_of_essays.append(list_of_essays[i])
            evaluation_list_of_labels.append(groups[y][2])
```

Figure 29: Dataset Preprocessing: Dataset Grouping

The "*groups*" list contains all the necessary information/configurations of the grouping. For each group, the first element is a list of the grades to be grouped together, the second element is the threshold of the group's training data and the third is the new group grade. The "*groups*" list minimized the hardcoding as it is limited to the list and no hardcoding was required in the rest of dataset grouping implementation.

4.1.2. 2nd COMPONENT: WORD TO WORD EMBEDDING (WORD VECTORS) CONVERSION

The second component was also implemented in python, also importing NumPy, corpora from gensim or just gensim (depending on the version), defaultdic from collections and the python file

from the 1st component. NumPy is "the fundamental package for scientific computing with python" (NumPy, 2018). The reason NumPy was imported is that it includes NumPy array, which is the default array input for Tensorflow which was used for the construction of the Convolutional Neural Network and is explained later. According to its website, "gensim is […] the most robust, efficient and hassle-free piece of software to realize unsupervised semantic modelling from plain text" (Řehůřek, 2009). Gensim is one of the most popular free Python libraries specialized in vector space modelling (Řehůřek, 2009), which is an algebraic model for representing texts documents as vector of identifiers, such as index terms. Gensim can be used both for training vector space models, as well as using already trained ones. As said in the Approach section, the aim of this Component is exactly that; to convert the essays to their vector representation using their identifiers. Since this project implements two different word-to-vector representations as was explained in the "Word-To-Vector" section in the approach, it was considered more suitable to fully describe their implementation in the versions they were used, since they use different features and modules of gensim. Last but not least, defaultdic (defaultdict, 2018) from collection was used to calculate the frequency of each word in the essay, by creating a dictionary-like object containing word-frequency pairs. A comparison of the two word-to-vector representations is provided in the "Results and Evaluation" section.

4.1.2.1. FIRST WORD-TO-VECTOR REPRESENTATION

The implementation of this component was coded in Python importing Gensim using the "Corpora and Vector Spaces" (Řehůřek, 2009) tutorial. The second component imports five python libraries from the first component. It imports a list containing the training essays, a list containing the testing essays, but also a list containing both training and testing essays. The reason it imports both lies on the fact that the dictionary which will later be used to generate the indexes is consisted by both the training and testing examples. The python method "*wordlist_to_vectorlist*" in figure 30 takes as input the list of both training and testing examples and returns the dictionary of words with frequency more than 1, which will be later used for the index generation.

```
08.
09.  def wordlist_to_vectorlist(lista):
10.      documents = []
11.      for i in range(len(lista)):
12.          documents.append(lista[i])
13.
14.
15.      texts = [[word for word in document.lower().split()]
16.              for document in documents]
17.      frequency = defaultdict(int)
18.      for text in texts:
19.          for token in text:
20.              frequency[token] += 1
21.      texts = [[token for token in text if frequency[token] > 1]
22.              for text in texts]
23.
24.      dictionary = corpora.Dictionary(texts)
25.      dictionary.save('/tmp/deerwester.dict')
26.
27.      corpus = [dictionary.doc2bow(text) for text in texts]
28.      corpora.MmCorpus.serialize('/Users/Angelos/Desktop/UNI/DISSERTATION/source_code/VERSIONS/dict
29.
30.      return {'corpus':corpus, 'dictionary':dictionary}
```

Figure 30: First Word2Vector - Dictionary creation

The python method "*new_wordlist_to_vectorlist*" in figure 31 will then take the dictionary and each essay from the training data and the testing data and convert it to vector representation.

```
33.  def new_wordlist_to_vectorlist(string, dictionary):
34.      new_doc = string
35.      new_vec = dictionary.doc2bow(new_doc.lower().split())
36.      return new_vec
```

Figure 31: First Word2Vector - Essay to Bag-of-Words

Previous versions of the second component were only using the first method "*wordlist_to_vectorlist*" shown in figure 30 and created a dictionary and corpus containing the vector representation of both training and testing essays. The corpus was then divided to training and testing essays in word-to-vector . However, after implementing the dataset grouping, this division was made on the first component. If we were to use the same method and put in the list of training and testing essays separately, the vector representations would use two different dictionaries. As a result, the index of a word in an essay in the training set and the index of the same word in the testing set would be different making the word index completely useless. Thus, there was a need for a second method which would take as input an essay and produce its bag-of-words representation using an existing dictionary consisting of both training and testing essays.

After the definition of the methods, there is a for loop, which iterates through the training set, converts each essay to its Bag-of-Words representation and saves the Bag-of-Words list in a list of lists containing all the essays of the training set. A second for loop follows, which performs the exact same procedure for the testing essays. The two for loops can be seen in figure 4.

```
res = wordlist_to_vectorlist(list_of_essays)
train_essays_1 = []
eval_essays_1 = []
for i in range(len(training_list_of_essays)):
    ress = new_wordlist_to_vectorlist(training_list_of_essays[i], res['dictionary'])
    train_essays_1.append(ress)

for i in range(len(evaluation_list_of_essays)):
    ress = new_wordlist_to_vectorlist(evaluation_list_of_essays[i], res['dictionary'])
    eval_essays_1.append(ress)
```

Figure 32: First Word2Vector - for loops creating a list of the training and a list of testing essays of the in Bag-of-Words form

Artificial Neural Networks require all training and testing examples to have the same size. As a result, there needs to be a padding which will pad all essays to have the same size. This padding is achieved by the for loops in figure 33.

```
max_len = 0
for i in range(len(train_essays_1)):
    if len(train_essays_1[i])>max_len:
        max_len = len(train_essays_1[i])

for i in range(len(train_essays_1)):
    while len(train_essays_1[i])<max_len:
        train_essays_1[i].append((0,0))


for i in range(len(eval_essays_1)):
    if len(eval_essays_1[i])>max_len:
        max_len = len(eval_essays_1[i])

for i in range(len(eval_essays_1)):
    while len(eval_essays_1[i])<max_len:
        eval_essays_1[i].append((0,0))
```

Figure 33: First Word2Vector - padding

A for loop first iterates through the training essays in order to calculate the maximum essay size and a second for loop then iterates through the training essays and pads them, so that all essays have the same length. Two more for loops performs the exact same procedure for the testing essays.

Last but not least, as already mentioned, numpy was imported in order to convert the lists to numpy arrays which are the default arrays input for Tensorflow. The conversion can be shown in figure 6.

```
72. train_essays_1 = np.asarray(train_essays_1,dtype=np.float32)
73. eval_essays_1 = np.asarray(eval_essays_1,dtype=np.float32)
74. train_labels_1 = np.asarray(training_list_of_labels,dtype=np.int32)
75. eval_labels_1 = np.asarray(evaluation_list_of_labels,dtype=np.int32)
```

Figure 34: First Word2Vector - Lists to Numpy arrays conversion

The default datatype for training and testing examples is float32 bit (single precision float: sign bit, 8 bits exponent, 23 bits mantissa), whereas the default datatype for training and testing labels/classes is int32 (an integer from -2147483648 to 2147483647)) (NumPy, 2018).

4.1.2.2. SECOND WORD-TO-VECTOR REPRESENTATION

The second representation converts each word into a 300-dimensional vector. For this representation, I used the word2vec model pre-trained on the Google News corpus using the Continuous Bag of Words model, which was described in the "Background". It is word-to-vector model which consists of 3 billion words and phrases in word-to-vector form. This component is coded in Python using Gensim and its Keyed Vectors module.

```
05. model = gensim.models.KeyedVectors.load_word2vec_format("GoogleNews-vectors-
    negative300.bin", binary=True)
```

Figure 35: Second Word2Vector - word2vec pre-trained import

The word2vec pre-trained model is first imported using the Keyed Vectors gensim module.

```
08. def split_list_of_essays(list_of_essayss):
09.     list_of_essays_splitted=[]
10.     for i in range(len(list_of_essayss)):
11.         list_of_essays_splitted.append(list_of_essayss[i].lower().split())
12.     return list_of_essays_splitted
```

Figure 36: Second Word2Vector - split_list_of_essays function

The *split_list_of_essays* function takes as an input a list of essays (either training or evaluation) and, using a for loop, iterates through the list of essays and splits the essays into words creating a new list of essays (where each essay is a list of words).

```
26. def list_of_essays_to_bow(list_of_essayss):
27.     list_of_essays_in_bow = []
28.     list_of_essays_splitted = split_list_of_essays(list_of_essayss)
29.     for i in range(len(list_of_essays_splitted)):
30.         list_of_essays_in_bow_single=[]
31.         for y in range(len(list_of_essays_splitted[i])):
32.             temp = ""
33.             for z in range(len(list_of_essays_splitted[i][y])):
34.                 if (list_of_essays_splitted[i][y][z]) not in ".,-+=!@#$%^&*()":
35.                     temp+= list_of_essays_splitted[i][y][z]
36.             if temp in model:
37.                 list_of_essays_in_bow_single.append(model.word_vec(temp))
38.         list_of_essays_in_bow.append(list_of_essays_in_bow_single)
39.         list_of_essays_in_bow = padding(list_of_essays_in_bow)
40.     return list_of_essays_in_bow
```

Figure 37: Second Word2Vector - list_of_essays_to_bow function

The function *list_of_essays_to_bow* takes in as input a list of essays and converts it to bag-of-words format (explained in the "Word-to-Vector" section of the "Background"). This is done by a series of nested for loops which call the *split_list_of_essays,* scan through the words of the new list and "remove" any punctuation marks that got "attached" to the words during splitting. Then, it checks if the scanned word is in the 3 billion words and phrases of the pre-trained model and if it is it adds the word to the new list containing the essays in bag-of-words format. Last but not least, they call the *padding* function.

```
14. def padding(list_of_essays_in_bow):
15.     max_len = 0
16.     for i in range(len(list_of_essays_in_bow)):
17.         if len(list_of_essays_in_bow[i])>max_len:
18.             max_len = len(list_of_essays_in_bow[i])
19. 
20.     for i in range(len(list_of_essays_in_bow)):
21.         while len(list_of_essays_in_bow[i])<max_len:
22.             list_of_essays_in_bow[i].append([0]*300)
23.     return list_of_essays_in_bow
```

Figure 38: Second Word2Vector - padding function

The padding function calculates the size of the biggest (in word count) essay in bag of words format and then pads all the essays (adding vectors of 300 0s) in order all of them to have the same size. If they don't have the same size they cannot be converted to a NumPy array which is the required type of input in Tensorflow.

```
43. train_essays_1 = list_of_essays_to_bow(training_list_of_essays)
44. eval_essays_1 = list_of_essays_to_bow(evaluation_list_of_essays)
45. 
46. 
47. train_essays_1 = np.asarray(train_essays_1,dtype=np.float32)
48. eval_essays_1 = np.asarray(eval_essays_1,dtype=np.float32)
49. train_labels_1 = np.asarray(training_list_of_labels,dtype=np.int32)
50. eval_labels_1 = np.asarray(evaluation_list_of_labels,dtype=np.int32)
```

Figure 39: Second Word2Vector - list_of_essays_to_bow function call and list to numpy array conversions

The lists of essays are first converted to lists of "bags of words" by calling the *list_of_essays_to_bow* and then converted to NumPy arrays like in the first word-to-vector representation.

4.1.3. 3rd COMPONENT: ARTIFICIAL NEURAL NETWORK

The implementation of this component was coded in Python importing Tensorflow using the tutorial on handwriting classification using the MNIST dataset (TensorFlow, 2018). A distinction between the two subcomponents of the 3rd components has not been made here, in contrast to the Approach section. The reason this distinction is unnecessary lies on the fact that the general implementation is the same between the versions in both components, but differentiates on the exact Artificial Neural Network process which will be analyzed at the respective versions.

All Artificial Neural Networks were implemented in Python using Tensorflow (TensorFlow, 2015). Tensorflow is an open-source Python framework, which can be used for various Machine Learning approaches. Its name, Tensorflow, is due to the multidimensional arrays used for the Artificial Neural Network operations called Tensors. Some of the advantages of Tensorflow is a fairly adequate documentation and Tensorboard, which can be used for model debugging and evaluation, such as the Cross-Entropy Loss to Global Steps graph which was used in the "Results & Evaluation". Some of the most popular current uses of Tensorflow are Deep Neural Networks used for Speech and Image Recognition and Classification by organizations such as Google, Mozilla, Snapchat, eBay, DeepMind and many more.

The 3rd component programs are consisted by three parts. The first is the model, both Feature Extraction and Classification components, the second is the modes and the last is the main function which sets some basic parameters and the inputs. Although the model is the most important part of the Tensorflow and the part which is mostly modified from version to version, the model cannot stand alone, which makes the other parts of the program equally important. It is understandably too confusing for someone with minimum Tensorflow or generally Machine Learning experience to understand those different parts, thus they will be explained thoroughly. As said, each of the different versions implemented and tested uses a different Convolutional Neural Network. However, the general implementation remains the same. Version 1.0 of the software has been used for the purposes of the general implementation analysis. Some lines of code have been repeated in order to avoid large figures as well as referring to other figures.

PART 1: The Model

The Artificial Neural Network model is the part of the Artificial Neural Network implementation which is subject to the most changes from version to version as it incorporates the most parameters and thus effects the classification more than the rest of the parts. The model part of the Artificial Neural Network implementation is a model function which is then called by the main function (Part 3) to create the Tensorflow estimator object. The model function definition is displayed in Figure 16.

```
15. def cnn_model_fn(features, labels, mode):
```

Figure 40: Artificial Neural Network Model -Model function Definition

The first layer of the model is the input layer.

```
16.     input_layer = tf.reshape(features["x"], [-1, 33, 11, 2])
```

Figure 41: Artificial Neural Network Model - Input Layer

The input layer's function is to reshape the input to a desirable form. In the above input layer (Version 1.0), the input essays (features["x"]) of size/word count 363 and 2-dimensional vectors for each word are converted to 2-dimensional representation with height 33, width 11 and 2 channels (1 for each word representation). The -1 before the dimensions is the batch size. It can be either set to be the actual bath size or it can be -1 and the batch size is set in the Main function (Part 3).

```
18.     conv1 = tf.layers.conv2d(
19.         inputs=input_layer,
20.         filters=64,
21.         kernel_size=[3, 3],
22.         padding="same",
23.         activation=tf.nn.sigmoid)
```

Figure 42: Artificial Neural Network Model – First Convolutional Layer

As mentioned in the "Background", the first layer of a Convolutional Neural Network is always a Convolutional layer. The first Convolutional layer takes as input the output of the input layer and then performs convolution using 64 filters of size 3x3. It uses sigmoid as the activation function and pads the output of the convolutional layer (by adding 0s) in order for it to have the same size as the input. Stride is not set, so it is automatically set to 1 which is the default value.

```
25.     pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
```

Figure 43: Artificial Neural Network Model – First Pooling Layer

The first pooling layer takes the output of the Convolutional layer as input and performs pooling/down-sampling using filter of size 2x2 with stride 2.

The implementation of the second Convolutional layer and the second Pooling layer are the same.

```
27.     conv2 = tf.layers.conv2d(
28.         inputs=pool1,
29.         filters=128,
30.         kernel_size=[3, 3],
31.         padding="same",
32.         activation=tf.nn.sigmoid)
```

Figure 44: Artificial Neural Network Model – Second Convolutional Layer

```
34.     pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
```

Figure 45: Artificial Neural Network Model – Second Pooling Layer

The second Convolutional layer takes in as input the output of the first Pooling layer and performs convolution but this time with 128 filters. The second Pooling layer takes in as input the output of the second Convolutional layer and performs exactly the same operation as the first Pooling layer.

```
36.     pool2_flat = tf.reshape(pool2, [-1, 8*2*128])
```

Figure 46: Artificial Neural Network Model – Flatenning Layer

The second Pooling layer is followed by a flattening "layer" which uses the reshape function (like the input layer) and flattens the output of the second Pooling layer of size 8x2 and 128 channels and outputs a 1-dimensional tensor with size (8x2x128=) 2048.

```
38.     dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.sigmoid)
```

Figure 47: Artificial Neural Network Model – First Fully Connected Layer

The flattened output of the feature extraction part is then propagated to the first fully connected layer with sigmoid as the activation function and consisted by 1024 neurons.

```
40.     logits = tf.layers.dense(inputs=dense, units=6)
```

Figure 48: Artificial Neural Network Model – Second Fully Connected / Logits Layer

The output of the first fully connected layer is then transferred to the second and last fully connected layer consisted of 6 neurons (same as the number of classes). After the example has been propagated from the input layer through the hidden layers (Convolutional-Pooling-Convolutional-Pooling-Dense) each neuron in the logits layer will output a probability that the example is in the respective class. However, it is important to notice that those probabilities will range from minus infinity to plus infinity. Thus the predictions in the logits layer will have to inputted into a SoftMax (Bishop, 2006) function in order to get the probabilities with range from 0 to 1.

PART 2: The Modes

The Modes part of the Artificial Neural Network is the part where the specific modes and their operations are set. There are three Artificial Neural Network modes; the training, the evaluation and the prediction mode.

```
54.     predictions = {
55.         "classes": tf.argmax(input=logits, axis=1),
56.         "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
57.     }
62.     onehot_labels = tf.one_hot(indices=tf.cast(labels, tf.int32), depth=6)
63.
64.     loss = tf.losses.softmax_cross_entropy(onehot_labels=onehot_labels, logits =logits)
65.
66.
67.     if mode == tf.estimator.ModeKeys.TRAIN:
68.         optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
69.         train_op = optimizer.minimize(
70.             loss=loss,
71.             global_step=tf.train.get_global_step())
72.         return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)
```

Figure 49: Artificial Neural Network Modes: Training Mode

Figure 49 displays the training mode setting. The loss has been calculated using the logits layer (after applying a SoftMax function), containing the model prediction probabilities, from the Artificial Neural Network model and the labels input in one hot label representation. The training mode setting sets the optimizer to be the stochastic gradient descent algorithm with learning rate equal to 0.001 and then sets the optimizer to minimize the loss function. When the Artificial Neural Network finishes training, it will return the final loss and global steps.

```
54.     predictions = {
55.         "classes": tf.argmax(input=logits, axis=1),
56.         "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
57.     }
74.     eval_metric_ops = {
75.         "accuracy": tf.metrics.accuracy(
76.             labels=labels, predictions=predictions["classes"])
77.         }
78.     return tf.estimator.EstimatorSpec(
79.         mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)
```

Figure 50: Artificial Neural Network Modes: Evaluation Mode

Figure 50 displays the evaluation mode setting. After the model has been trained using the training set, it is evaluated using the testing set. After each testing example has been input into the network and propagated through the hidden (feature extraction and classification) layers it reaches the output layer where it produces a predicted class. This class is compared to the actual label of the testing example in order to calculate the accuracy of the estimator model (total matches over total number of instances). It is important to notice that the Accuracy is comparing prediction class to label, whereas Loss was comparing prediction probabilities to label (in one hot format). When the Artificial Neural Network finishes evaluating, it will return the Loss and the evaluation metrics (in this case Accuracy).

```
54.     predictions = {
55.         "classes": tf.argmax(input=logits, axis=1),
56.         "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
57.     }
58.
59.     if mode == tf.estimator.ModeKeys.PREDICT:
60.         return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)
```

Figure 51: Artificial Neural Network Modes: Prediction Mode

Figure 51 displays the prediction mode setting. After the model has been trained by training examples' instance-class pairs and evaluated by testing examples' instance-class pairs to assess its performance, it can be used to classify unlabeled/unclassified examples. When in prediction mode, the model will return the predictions dictionary containing the probabilities of each instance to be in each class.

PART 3: The Main Function

The main function starts by importing the training/evaluation essays and labels.

```
82. def main(unused_argv):
83.     train_data = train_essays_1
84.     train_labels = train_labels_1
85.     eval_data = eval_essays_1
86.     eval_labels = eval_labels_1
```

Figure 52: Artificial Neural Network Main Function– Training/Testing Essays/Grades Import

The name change is only used for convenience and separating the output of the Word Embeddings (2nd) component to the input of the Artificial Neural Network (3rd) component (although they are the same).

```
89.     project_classifier = tf.estimator.Estimator(
90.         model_fn=cnn_model_fn, model_dir='cnn_model2.0')
```

Figure 53: Artificial Neural Network Main Function – Estimator Creation

The main function then creates an estimator using the model created by Part 1 and Part 2 and a directory to save the model's checkpoints and other data.

```
98.     train_input_fn = tf.estimator.inputs.numpy_input_fn(
99.         x={"x": train_data},
100.        y=train_labels,
101.        num_epochs=None,
102.        shuffle=True)
103.
104.
105.    project_classifier.train(
106.        input_fn=train_input_fn,
107.        steps=20000,
108.        hooks=[logging_hook])
```

Figure 54: Artificial Neural Network Main Function: Training Mode Activation

The main function then activates the training mode from Part 2. This is done in two steps. First it sets the input of the estimator's training mode to be the training essays and labels and also sets some parameters to define the way that the training set is manipulated. The number of epochs is the number of times that the entire dataset will go through the network. It is set to none, since the training examples will be divided and go through the network in batches. When the number of batches is not explicitly set, like in this case, it is automatically set by Tensorflow to the default value which is 128. Shuffle is set to true as shuffling the training set will make sure that there aren't any batches consisted mainly by essays of the same grade. Ideally, we want each batch to

accurately represent the entire training set. A batch consisted by essays of the same grade would distract the loss function and the optimization algorithm.

The actual activation of the training mode is achieved on the second step (seen in figure 54) using *"project_classifier.train"*. The input of this function is the input parameters set by the previous step, the number of global steps and the logging hook. The number of global steps is the number of times a batch will go through the network and is set to 20000, which means that the model will predict, calculate the loss and optimize until 20000 batches of 128 training examples have been through the network. The logging hook is a method of returning results such as loss and predictions while the network is training and its parameters were set but are not explained as the logging hook is optional and insignificant.

```
110.     eval_input_fn = tf.estimator.inputs.numpy_input_fn(
111.         x={"x": eval_data},
112.         y=eval_labels,
113.         num_epochs=1,
114.         shuffle=False)
115.
116.     eval_results = project_classifier.evaluate(input_fn=eval_input_fn)
117.
118.     print(eval_results)
```

Figure 55: Artificial Neural Network Main Function - Evaluation Mode Activation

After the model has been trained using the training set, it is time to be evaluated using the testing set. The evaluation mode activation is very similar to the training mode activation and is again done in two steps. First it sets the input of the estimator's evaluation mode to be the evaluation essays and labels. The evaluation set goes through the network in batches, too (again number of batches is set to default value). However, in contract to the training mode, in evaluation mode, the number of epoch is set to 1, as we want evaluation to stop as soon as the entire evaluation set is through the network and predicted, but only once. Last but not least, shuffle is set to false, as there is no need to shuffle the evaluations set as the network is no longer optimizing and thus, the loss function is not affected.

```
120.    predictions_input_fn = tf.estimator.inputs.numpy_input_fn(
121.        x={"x": eval_data},
122.        y=None,
123.        shuffle=False)
124.
125.    predictions = project_classifier.predict(
126.        input_fn=eval_input_fn,
127.    )
```

Figure 56: Artificial Neural Network Main Function - Prediction Mode Activation

After the model has been trained, evaluated and assessed that it performs adequately, it can be used to predict unlabeled instances. The prediction mode activation, similarly to training and evaluation mode activation, is done in two steps. First it sets the input of the estimator's prediction mode to be the prediction essays. Obviously, there are no prediction input labels as the prediction examples are unlabeled. The prediction input was set to be the evaluation examples in order to get the confusion matrix. However, replacing eval_data by the prediction data will produce the predictions for the prediction data.

```
107.    predictions_input_fn = tf.estimator.inputs.numpy_input_fn(
108.        x={"x": eval_data},
109.        y=None,
110.        shuffle=False)
111.
112.    predictions = project_classifier.predict(
113.        input_fn=eval_input_fn,
114.    )
115.
116.
117.    # print(list(predictions))
118.
119.    l_predictionss = list(predictions)
120.    predictionss = []
121.    for pred in l_predictionss:
122.        predictionss = np.append(predictionss,pred['classes'])
123.
124.    confusion_matrix = tf.contrib.metrics.confusion_matrix(eval_labels, predictionss)
125.    with tf.Session() as sess:
126.        print(sess.run(confusion_matrix))
```

Figure 57: Artificial Neural Network Main Function - Confusion Matrix Implementation

Figure 51 displays the method used to obtain the confusion matrix. The confusion matrix can be also outputted as an evaluation metric (like Accuracy) but Tensorflow does not provide confusion matrix as an evaluation metric for multiclass classifications. Thus, this was overridden by predicting the class of the evaluation data and then comparing it to the labels of the evaluation data.

5. RESULTS AND EVALUATION

5.1. ANALYSIS OF THE EVALUATION METRICS AND METHODS USED

There is a vast number of different metrics that are being used to evaluate the Artificial Neural Networks and other Machine Learning approaches. However, without underestimating the significance of the rest, the six evaluation metrics that were used to evaluate the performance of the Artificial Neural Network are enough to fully assess them.

The first evaluation metric that was used is Accuracy. Accuracy is calculated as the number of matches between labels (actual grades) and predictions (predicted grades output from the Artificial Neural Network) over the total number of instances. In simpler words, Accuracy is the percentage of the essays that were correctly classified by the Artificial Neural Network.

The second evaluation metric that was used is the Confusion matrix. The Confusion matrix is an $n_x n$ matrix where n is the number of possible classes. It is a comparison between the predicted class and actual class of the evaluation examples. The table below is an example of a Confusion Matrix as a result of the evaluation of a model classifying a number of images of animals as Mammal, Fish, Bird and Reptile.

		PREDICTED CLASS			
ACTUAL CLASS		**MAMMAL**	**FISH**	**BIRD**	**REPTILE**
	MAMMAL	103	2	5	7
	FISH	6	89	2	8
	BIRD	9	3	29	6
	REPTILE	11	2	12	33

The total number of classes is 4 and as a result the Confusion Matrix is a 4x4 matrix. Interpreting the Confusion Matrix is very easy. Each cell of the Confusion Matrix contains the number of instances with a specific predicted class and actual class. For each cell the predicted class is its column and the actual class is its row. For example, looking at the Confusion Matrix above, one can understand that 12 images were classified as Bird although they contain Reptiles. The cells on the diagonal of the Confusion Matrix is the number of instances for each class that were classified correctly since in each of them the Predicted and the Actual class match. As a result, in a perfect model, all cells other than the cells on the diagonal would be zero.

It is important to notice that the Accuracy can also be calculated from the Confusion Matrix by dividing the sum of the cells in the diagonal by the sum of all cells, in other words dividing the correctly classified instances by the total number of instances. However, having Accuracy as a separate evaluation metric is recommended for convenience.

The table below is an example of a second Confusion Matrix as a result of the evaluation of classifying a number of costumer reviews as very satisfied, satisfied, unsatisfied and very unsatisfied.

		PREDICTED CLASS			
		very satisfied	**satisfied**	**unsatisfied**	**very unsatisfied**
ACTUAL CLASS	**very satisfied**	103	2	5	7
	satisfied	6	89	2	8
	unsatisfied	9	3	29	6
	very unsatisfied	11	2	12	33

When studying the Confusion Matrix, it is important to consider the type of the classes. The classification of the second Confusion Matrix is very similar to the classification of this project. Not only because they both classify text, but more importantly because they have the same type of classes. Comparing the classes from the first Confusion Matrix (Mammal, Fish, Bird and Reptile) and the classes from the second Confusion Matrix (very satisfied, satisfied, unsatisfied and very unsatisfied), it is important to notice that the first set of classes is unordered (or categorical) data, whereas the second set of classes is ordered (or ordinal) data. The difference between the two class sets, is that the animal types do not have an implicit or natural order, whereas the satisfaction index classes do. The "mistake" of the classifier when classifying a Bird as a Reptile compared to a Bird as a Mammal is exactly the same. However, the "mistake" of the classifier if it classifies a review with (actual) class "very unsatisfied" to have a (predicted) class "very satisfied" is a lot bigger than if it classifies the same review as "unsatisfied" (although it still did not predict the class correctly). As said, the type of classes in this investigation are ordinal, since there is an order of the data from grades A to F.

As a result, the two Confusion Matrix should be studied differently. Thus, we are also going to use a fourth evaluation metric, which will be called **Accuracy±1** and will calculate the number of instances where the predicted and the actual grade are considered matched if they are the same or they have a deviation of 1 class/grade. Since the grades are ordinal classes and they follow a natural order, this evaluation metric will be vital in assessing the extend of which the model is able not only to capture the characteristics of each class but also the ordinal nature of the classes. The metric can be calculated from the Confusion Matrix if we sum the instances on the diagonal but also add to that sum the instances next to the diagonal and then divide by the total number of instances.

The fifth evaluation metric that was used for the purposes of Results and Evaluation is a table showing the **Accuracy** and **Accuracy±1 per class** and then the average of those values. Before the data grouping, the Accuracy of the model was 38.17% which was exactly the percentage of essays with grades 8 in the evaluation set. The "congestion" of essays with grades 8 caused the model to over-focus on grade 8 and thus classified all essays of the evaluation set as grade 8. A model with a 30% Accuracy which classifies the essays in all classes is more preferable than a

model with 40% Accuracy which classifies all essays in all class. Thus, it is also important to see how the **Accuracy** and the **Accuracy±1** fluctuates from one class to another.

The sixth evaluation metric that was used is Loss. The function and importance of the Loss function (or Cost function) in Artificial Neural Networks was fully explained in the "Artificial Neural Networks Optimization" in the "Background". Tensorboard allows us to visualize the function of the Stochastic Gradient Descent optimizer which was used to minimize loss, by plotting a graph of "Loss vs Global Steps". After a Global Step has been completed, the optimizer calculates weights and biases again to minimize Loss. As a result, Loss is expected to decrease as Global Steps increase. However, as the Loss is decreasing, it is more difficult for the optimizer to decrease it even more. So, it is also expected to decrease in a decreasing manner.

ACCURACY vs LOSS

In a first glance, it is understandable to confuse the function of the loss and the accuracy measure. They both describe the performance of the Artificial Neural Network and thus they are inversely proportional. As the Artificial Neural Network improves, its loss decreases reaching a minimum at 0 and its accuracy increases reaching a maximum at 1. However, those two metrics are completely different regarding their calculation and meaning. The basic difference is that loss is using the probabilities of the predicted instances to be in each class and contrasts them to the labels, whereas accuracy is using the predicted classes and contrast them to the labels. This means that there can be a loss even if the classifier classifies an instance correctly. Let, for example, that an Artificial Neural Network is evaluated by classifying a test set of 10 instances and it performs brilliantly classifying all of the 10 instances correctly. So comparing predicted classes to the labels will give us an accuracy of 1, whereas comparing probabilities of the predicted instances to be in each class to the labels will give us a low loss but most likely not 0. The only case an Artificial Neural Network will have loss equal to 0 is if it achieves not only to correctly classify all instances but to assign a probability of 1 to all instances to belong to the correct class and thus a probability of 0 to all instances to belong to the rest of the classes.

5.2. VERSION-SPECIFIC RESULTS

5.2.1. VERSION 1.0

Version 1.0 was trained on the training examples and then classified the evaluation examples with a 0.5004831 or **50.05%** Accuracy and the Confusion matrix following.

PREDICTED GRADE/LABEL							
ACTUAL GRADE/LABEL		**F**	**E**	**D**	**C**	**B**	**A**
	F	6	1	0	0	0	0
	E	0	71	21	1	0	0
	D	0	125	241	137	22	0
	C	0	4	46	79	46	4
	B	0	0	13	44	86	15
	A	0	0	0	3	35	35

$$Accuracy \pm 1 = \frac{988}{1035} = 0.9546$$

So the **Accuracy±1** of the model in version 1.0, is 0.9546 or **95.46%**.

ACCURACY AND ACCURACY±1 FOR EACH CLASS OF VERSION 1.0

Actual Grade/Label	Accuracy	Accuracy±1
F	85.7	100.0
E	76.3	98.9
D	45.9	95.8
C	44.1	94.9
B	54.4	91.8
A	47.9	95.9
Average of Classes	**59.05**	**96.22**

LOSS TO GLOBAL STEPS GRAPH OF VERSION 1.0

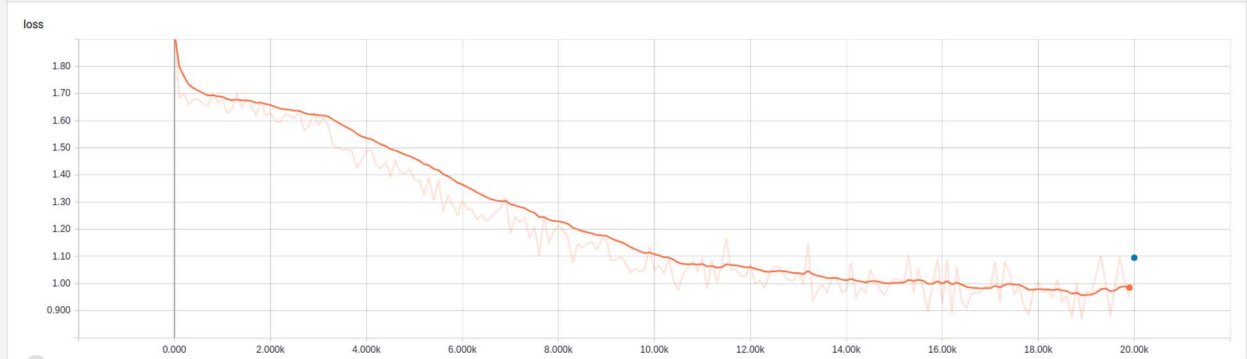

Figure 58: Results and Evaluation - Version 1.0 - Loss to Global Steps Graph

ANALYSIS OF THE RESULTS OF VERSION 1.0

The Accuracy of the model in classifying the evaluation set is, without a doubt, admirable, considering the simplicity of the Word-To-Vector representation. Both the Accuracy±1 value and the Confusion Matrix show that the model not only classifies half of the examples correctly but also successfully captures the ordinal structure of the data. All essays with grade A are not classified with a lower grade than C, all essays with grade F are not classified with a higher grade than E and all essays with grade D are classified between grade E and B. The loss can be seen declining and reaching a minimum of 1 at around 18000 steps. Loss of 1 is considered a very low value for this task. The important, though, is that the loss is steadily declining and is not at any point increasing. This means that during training, the model is improving as more batches are used for training. To summarize, the model in Version 1.0 is successfully capturing the ordinal structure of the data classifying the essays with an Accuracy of 50.05% and a Accuracy±1 of 95.46%.

ACCUMULATED RESULTS OF VERSION 1

Metric	Value
Accuracy	50.05
Accuracy±1	95.46
Average Accuracy of Classes	59.05
Average Accuracy±1 of Classes	96.22

5.2.2. **VERSION 2.0**

Version 2.0 was trained on the training examples and then classified the evaluation examples with a 0.50821257 or **50.82%** Accuracy and the Confusion matrix following.

PREDICTED GRADE/LABEL							
ACTUAL GRADE/LABEL		**F**	**E**	**D**	**C**	**B**	**A**
	F	6	1	0	0	0	0
	E	0	69	22	1	0	0
	D	0	117	248	145	15	0
	C	0	5	43	96	31	4
	B	0	0	12	59	72	15
	A	0	0	0	4	34	35

$$Accuracy \pm 1 = \frac{994}{1035} = 0.9603$$

So the **Accuracy±1** of the model in version 2.0, is 0.9603 or 96.03%.

ACCURACY AND ACCURACY±1 FOR EACH CLASS OF VERSION 2.0

Actual Grade/Label	Accuracy	Accuracy±1
F	85.7	100.0
E	74.2	98.9
D	47.2	95.8
C	53.6	94.9
B	45.6	91.8
A	48.0	95.9
Average of Classes	**59.05**	**96.22**

LOSS TO GLOBAL STEPS GRAPH OF VERSION 2.0

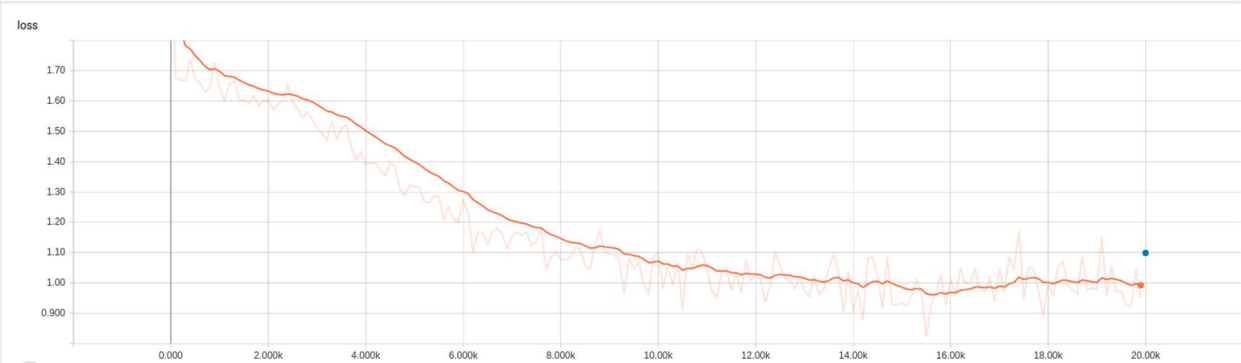

Figure 59: Results and Evaluation - Version 2.0 - Loss to Global Steps Graph

ANALYSIS OF THE RESULTS OF VERSION 2.0

The results of Version 2.0 are very similar to those of Version 1.0. Both versions classify the testing examples with a similar Accuracy of 50.82 % (compared to 50.05%) and Accuracy±1 of 96.03 (compared to 96.22%). Comparing their Average Accuracies, both versions have the exact same values. Comparing their loss graphs, Version 2.0 seems to reach a lower value of loss of about 0.97 at around 16000 steps but then stops declining and stabilizes at around 1.

So to summarize, Version 2.0, like Version 1.0, achieves not only to classify half of the essays correctly but also to capture the ordinal structure of the essays and classifies all of the essays with a very small divergence from the real class. The loss graph of both models seems to reach a satisfactory low, but then stabilize and the model seems unable to reach a lower loss. This is caused by the simplistic word to vector representation. No matter how good the models are, it is impossible to extract features and then use them to classify the examples if the features are not there in the first place.

ACCUMULATED RESULTS OF VERSION 2.0

Metric	Value
Accuracy	50.82
Accuracy±1	96.03
Average Accuracy of Classes	59.05
Average Accuracy±1 of Classes	96.22

5.2.3. **VERSION 3.0**

Version 3.0 was trained on the training examples after they were converted to bag-of-words using the second word to vector representation and then classified the evaluation examples with a 0.2415 or **24.15%** Accuracy and the Confusion matrix following.

PREDICTED GRADE/LABEL							
ACTUAL GRADE/LABEL		**F**	**E**	**D**	**C**	**B**	**A**
	F	0	7	0	0	0	0
	E	0	84	0	9	0	0
	D	0	186	0	339	0	0
	C	0	13	0	166	0	0
	B	0	4	0	154	0	0
	A	0	0	0	73	0	0

$$Accuracy \pm 1 = \frac{936}{1035} = 0.904347$$

So the **Accuracy±1** of the model in version 3.0, is 0.904347826 or **90.43**%.

ACCURACY AND ACCURACY±1 FOR EACH CLASS OF VERSION 3.0

Actual Grade/Label	Accuracy	Accuracy±1
F	0	100
E	90.32	90.32
D	0	100
C	92.75	92.74
B	0	97.47
A	0	0
Average of Classes	**30.51**	**80.09**

LOSS TO GLOBAL STEPS GRAPH OF VERSION 3.0

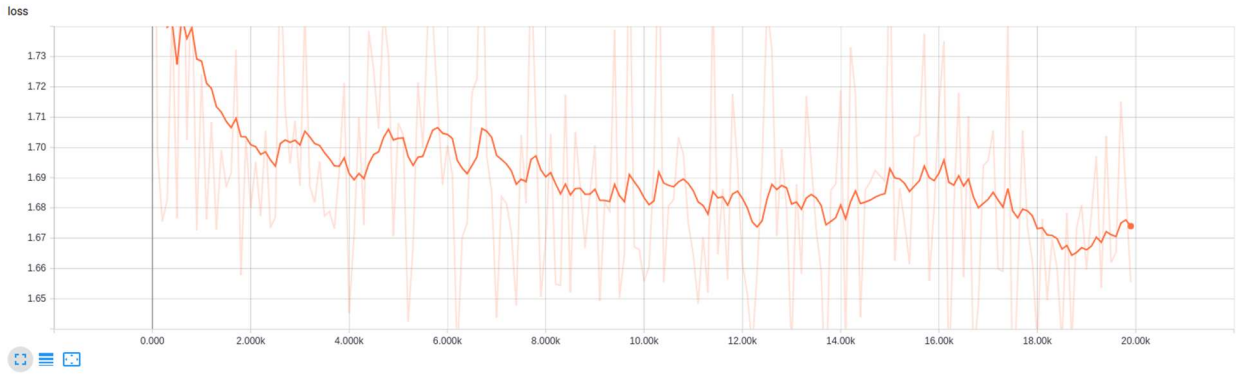

Figure 60: Results and Evaluation - Version 3.0 - Loss to Global Steps Graph

ANALYSIS OF THE RESULTS OF VERSION 3.0

The results of Version 3.0 are disappointing. The Accuracy of the model is less than half the Accuracy achieved by the models in Version 1.0 and 2.0. Although, the Accuracy±1 is not very declined, the Confusion Matrix is a very clear indicator of the model's failure. As can be seen from the Confusion Matrix and the Accuracy for each class table, none of the essays were classified as A, B, D or F (0% Accuracy in those classes) although the actual class of 763 classes of the testing set (73.7% of the testing set) is A, B, D or F. Another indicator of the model's failure is its loss graph, as the loss doesn't seem to follow a downward trajectory. On the first 2000 steps it reaches a local minimum of about 1.7 and then fluctuates around this value reaching a minimum of around 1.65 at around 19000 steps, which comparing to the minimum of the loss function in Version 1.0 and 2.0 (less than 1) is very high. As said in the Approach, the reason of these results is the insufficient feature extraction part which is not big enough to manipulate the input and provide the necessary features for the classification.

ACCUMULATED RESULTS OF VERSION 3.0

Metric	Value
Accuracy	24.15
Accuracy±1	90.43
Average Accuracy of Classes	30.51
Average Accuracy±1 of Classes	80.09

5.2.4. **VERSION 4.0**

Version 4.0 was trained on the training examples and then classified the evaluation examples with a 0.5342995 or **53.43%** Accuracy and the Confusion matrix following.

PREDICTED GRADE/LABEL							
ACTUAL GRADE/LABEL		**F**	**E**	**D**	**C**	**B**	**A**
	F	0	7	0	0	0	0
	E	0	56	37	0	0	0
	D	0	44	389	0	92	0
	C	0	0	95	0	82	0
	B	0	0	50	0	108	0
	A	0	0	9	0	64	0

So the **Accuracy±1** of the model in version 4.0, is 0.852173913 or **85.22%**.

ACCURACY AND ACCURACY±1 FOR EACH CLASS OF VERSION 4.0

Actual Grade/Label	Accuracy	Accuracy±1
F	0	100
E	60.22	100
D	74.1	82.48
C	0	98.88
B	68.35	68.35
A	0	87.67
Average of Classes	**33.78**	**89.56**

LOSS TO GLOBAL STEPS GRAPH OF VERSION 4.0

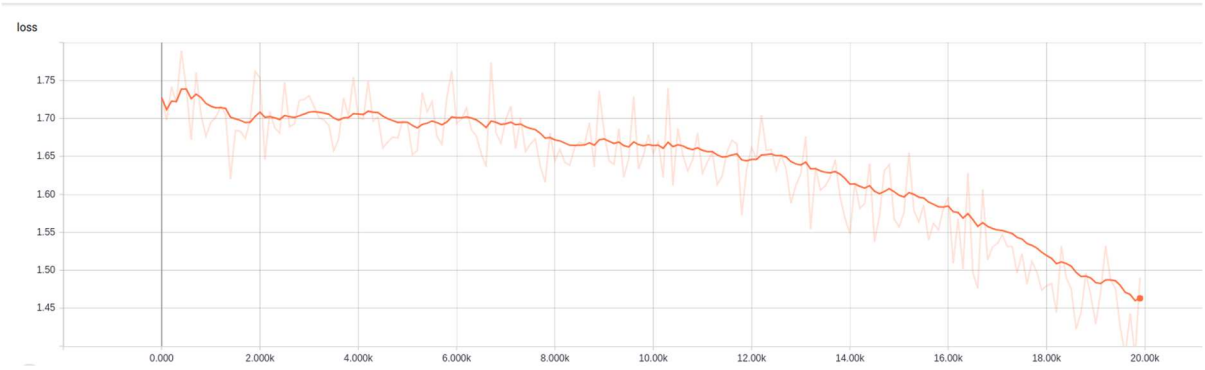

Figure 61: Results and Evaluation - Version 4.0 - Loss to Global Steps Graph

ANALYSIS OF THE RESULTS OF VERSION 4.0

The results of Version 4.0 are clearly improved compared to Version 3.0, as the Accuracy is more than doubled (53.43% to 24.14%). This improvement can be also seen by The Loss to Global Steps graph, where the loss seems to be steadily declining and reaching a minimum of about 1.47 at around 20000 steps (compared to 1.65 in Version 3.0). If Version 4.0 was to be trained for more steps, the loss could be even less. However, there are still significant deficits. Similarly, to Version 3.0, none of the testing examples were classified to classes A, B and F, whereas only 1 essay was classified to class E. This can be seen by both the Confusion Matrix and the Accuracy for each class, where the average Accuracy is 33.78%. However, it is unclear whether the feature extraction should be even bigger or it is already to big for the classification, thus there should be more or bigger (or both more and bigger) layers in the classification part.

ACCUMULATED RESULTS OF VERSION 4.0

Metric	Value
Accuracy	53.43
Accuracy±1	85.22
Average Accuracy of Classes	33.78
Average Accuracy±1 of Classes	89.56

5.2.5. VERSION 5.0

Version 5.0 was trained on the training examples and then classified the evaluation examples with a 0.297584541 or **29.76%** Accuracy and the Confusion matrix following.

PREDICTED GRADE/LABEL							
ACTUAL GRADE/LABEL		**F**	**E**	**D**	**C**	**B**	**A**
	F	0	1	2	4	0	0
	E	0	0	70	23	0	0
	D	0	0	140	385	0	0
	C	0	0	11	168	0	0
	B	0	0	5	153	0	0
	A	0	0	0	73	0	0

So the **Accuracy±1** of the model in version 5.0, is 0.896618357 or **89.66%**.

ACCURACY AND ACCURACY±1 FOR EACH CLASS OF VERSION 5.0

Actual Grade/Label	Accuracy	Accuracy±1
F	0	14.29
E	0	75.27
D	26.67	100
C	93.85	100
B	0	96.84
A	0	0
Average of Classes	**20.09**	**64.40**

LOSS TO GLOBAL STEPS GRAPH OF VERSION 5.0

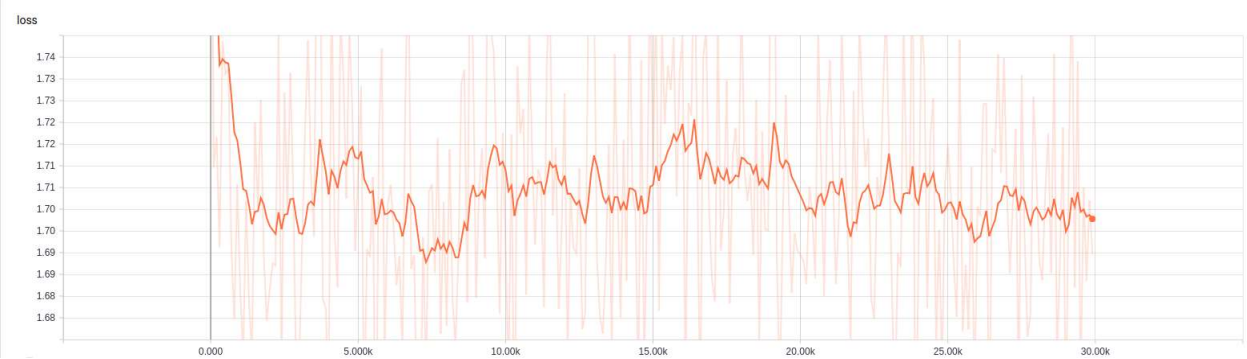

Figure 62: Results and Evaluation - Version 5.0 - Loss to Global Steps Graph

ANALYSIS OF THE RESULTS OF VERSION 5.0

The results of Version 5.0 are very disappointing. Starting from the Accuracy, only 29.76% of the testing examples were classified correctly. Accuracy±1 is reasonably high but it does not reflect the Confusion Matrix, where all of the essays but one were predicted to be either C or D and since most of the essays in the testing set are either C or D, the Accuracy and Accuracy±1 are lifted and do not reflect the actual model's performance. This can shown by the Average Accuracy and Average Accuracy±1, which compared to Accuracy and Accuracy±1, are both reduced. The Loss to Global Steps graph also confirms that the estimator is unable to classify the essays as the cross-entropy loss is fluctuating over and under around 1.71 without showing any decline.

ACCUMULATED RESULTS OF VERSION 5.0

Metric	Value
Accuracy	29.76
Accuracy±1	89.66
Average Accuracy of Classes	20.09
Average Accuracy±1 of Classes	64.40

5.2.6. **VERSION 6.0**

Version 6.0 was trained on the training examples and then classified the evaluation examples with a 0.48019323 or **48.02%** Accuracy and the Confusion matrix following.

PREDICTED GRADE/LABEL							
ACTUAL GRADE/LABEL		**F**	**E**	**D**	**C**	**B**	**A**
	F	5	2	0	0	0	0
	E	1	65	24	3	0	0
	D	1	77	275	74	98	0
	C	0	5	47	32	95	0
	B	0	1	19	21	108	9
	A	0	0	1	6	54	12

So the **Accuracy±1** of the model in version 5.0, is 0.870531401 or **87.05%**.

ACCURACY AND ACCURACY±1 FOR EACH CLASS OF VERSION 6.0

Actual Grade/Label	Accuracy	Accuracy±1
F	71.43	100
E	69.89	96.77
D	52.38	81.14
C	17.88	97.21
B	68.35	87.34
A	16.44	90.41
Average of Classes	**49.4**	**92.15**

LOSS TO GLOBAL STEPS GRAPH OF VERSION 6.0

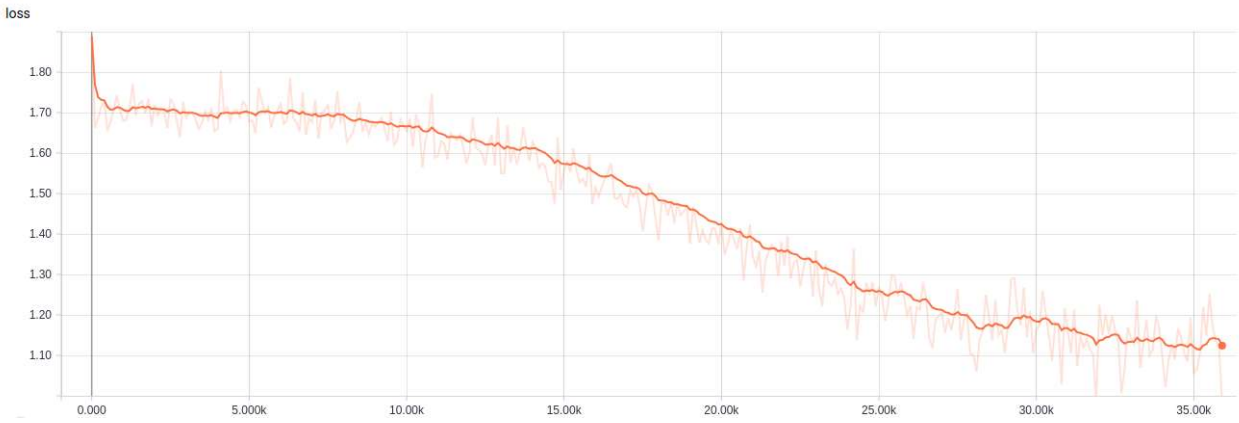

Figure 63: Results and Evaluation - Version 6.0 - Loss to Global Steps Graph

ANALYSIS OF THE RESULTS OF VERSION 6.0

The results of Version 6.0 are very encouraging. Starting from the Accuracy, more than 48% of the testing examples were classified correctly. Taking into consideration, the Confusion Matrix and the Average Accuracies per class, the model seems to capture both the general pattern as well as the pattern of essays of the same grade. In comparison to the previous versions using the second word-to-vector representation (Versions 3.0 to 5.0), the model classifies essays in all grade classes as can be seen from the Confusion Matrix. This can be also seen from the Average Accuracies per class table, where both the Accuracy and Accuracy±1 in all classes does not have a significant difference from the Average Accuracy and Accuracy±1 per class respectively. Last but not least, the Loss to Global Steps graph also reflects the success of the model. Loss seems to steadily decline and at specific points even reaching below 1 (unsmoothed curve).

ACCUMULATED RESULTS OF VERSION 6.0

Metric	Value
Accuracy	48.02
Accuracy±1	87.05
Average Accuracy of Classes	49.40
Average Accuracy±1 of Classes	92.15

5.3. ACCUMULATED RESULTS OF ALL VERSIONS

VERSION	ACCU	ACCU±1	AVG ACCU	AVG ACCU±1
1.0	50.05	95.46	59.05	96.22
2.0	50.82	96.03	59.05	96.22
3.0	24.15	90.43	30.51	80.09
4.0	53.43	85.22	33.78	89.56
5.0	29.76	89.66	20.09	64.40
6.0	48.02	87.05	49.40	92.15

5.4. EVALUATION OF THE DIFFERENT FACTORS CONSIDERED

The investigation emphasized in three important factors of the classification. The first factor was the word-to-vector representation as the essays are inserted into the Convolutional Neural Network. The second factor was the essay representation as the essays are inserted into the Convolutional Neural Network and the third factor was the depth or size of the Convolutional Neural Network.

5.4.1. WORD TO VECTOR REPRESENTATION

Versions 1.0 and 2.0 implemented the first word-to-vector representation where each word is represented by a 2-dimensional vector where the first dimension is the words' dictionary index and the second dimension is the frequency of the word in the example. Versions 3.0 to 6.0 implemented the second word-to-vector representation where each word is represented by a 300-dimensional vector extracted from the pre-trained Google model using the Continuous-Bag-of-Words word2vec model architecture. The versions which implemented the first representation prevailed those the versions which implemented the second representation in all evaluation metrics. However, this certainly does not reflect that the first word-to-vector representation itself is more suitable as the word-to-vector representation for Automated Essay Marking or text classification is general. On the contrary, the model implemented for the first word-to-vector representation may be smaller, but taking into consideration the size of the input in the first and second word-to-vector representation, they are relatively larger. Unfortunately, this assumption

cannot be validated by a model (such as Version 7.0) as the memory capabilities at my disposal during the project did not allow training a model of this size.

5.4.2. ESSAY REPRESENTATION

Version 1.0 and 2.0 both implemented the first word-to-vector representation. As a result, their inputs were exactly the same and their parameters were set in order their extraction feature part to produce the same output. Their classification part structure was, again, exactly the same and taking into consideration that the examples are flattened to 1-dimensional representation before entering the classification part, their classification as a whole was exactly the same. Their only difference is that Version 1.0 converted the 363-vectors essay/input to a 2-dimensional form of height 33 and width 11, whereas Version 2.0 kept the input in 1-dimensional form with size 363. Both models were structured this way in order to compare their performance in a "ceteris paribus[1]" environment. Their results were very much alike with Version 2.0 results being slightly better and thus the conclusion of this comparison is that the essay representation, does not affect the feature extraction and thus the classification.

5.4.3. ARTIFICIAL NEURAL NETWORK DEPTH

There is no correct answer for the Artificial Neural Network's depth to construct an accurate Automated Essay Marking system. The reason is that it depends on the size of the input. The same deep Artificial Neural Network is very likely to be overfitted if it is trained on a simple word-to-vector training set and at the same time it is very likely to be inadequate and underfitted if it is trained on a complex word-to-vector training set.

[1] Other things equal

6. FUTURE WORK

The project was completed with success. However, there are is variety of improvements that can be and have been planned to be made in the future.

6.1. VERSION 7.0

The "Version-Specific Approach" section described the approaches taken in each version and the reason for taking them. The approach in Version 7.0 described an Artificial Neural Network structure, which could result to a high accuracy classification using the second word-to-vector representation. Although Version 7.0 was implemented, the limited computing power in disposal made its training impossible. Training Version 7.0 could produce very important outcomes since, compared to the rest of the versions using the second word-to-vector (Versions 3.0 to 6.0), it is the only Artificial Neural Network, which is deep enough in order to extract the features from the high dimensional input vectors of the second word-to-vector representation and deep enough to then classify the essays using them. Training Version 7.0 could take weeks, but it is the first priority of the future work.

6.2. LINGUISTIC ANALYSIS

Text classification of this kind requires a highly qualified team of researchers from various disciplines such as linguistics. Linguistics could be very helpful in determining words that should be dropped before the essays are converted into word vectors and they are inserted into the Artificial Neural Network. Words like "a", "the", "of", "that" are most likely to appear in all essays, regardless their class/grade. Such words and other that do not or should not affect the classification of an essay is better to be removed from the essays. Removing unnecessary words will also reduce the size of the essays keeping only important local indicators that can lead to a class membership.

6.3. RECURRENT NEURAL NETWORKS

Recurrent Neural Networks have also been used to perform complex text classifications. The assumption we made and led us to using Convolutional Neural Networks is that the order of the

words in the essay is insignificant and that local indicators will lead to the correct class regardless of their position in the text. The assumption of Recurrent Neural Networks is exactly the opposite. Recurrent Neural Networks have been proven to outperform Convolutional Neural Networks when the order of the words in the text affects their class membership, whereas Convolutional Neural Networks have been proven to outperform Convolutional Neural Networks when it does not (Yin, et al., 2017). Although, there are no indicators that our hypothesis is invalid, there is no proof that it is not until using Recurrent Neural Networks and comparing their performance to the existing models.

6.4. ARTIFICIAL NEURAL NETWORK OPTIMIZATION

As mentioned before, the choice of Stochastic Gradient Descent was mainly because of its computational efficiency since it changes the weights after each training example in comparison to Batch Gradient Descent or Minibatch Gradient Descent, which change the weights after an entire batch or part of the batch respectively. Stochastic Gradient Descent was forced by the limited memory capabilities of the machines used (or available to be used). However, Stochastic Gradient Descent is normally used with very large datasets (unlike the dataset used). A larger dataset would need a larger batch size so that each batch can adequately represent the dataset. So, since the Batch Gradient Descent's run time increases exponentially in relation to the batch size, the Stochastic Gradient Descent is used instead. Since the dataset used for the text classification is limited, it is very likely that a Batch Gradient Descent will perform better, given that we have the memory capabilities to utilize it. Decreasing the learning rate and increasing the global steps, so that the model learns more slowly for a larger number of steps, was tried repeatedly but in vain for the same reason. Another option for the optimization algorithm is to use a cyclical learning rate, starting from a high learning rate and monotonically decreasing. The cyclical learning rate has been implemented in various classification models with great results (Smith, 2017).

7. CONCLUSIONS

After completing an investigation, it is vital to evaluate the general outcomes. The aim of this report was to investigate the application of Machine Learning in Automated Essay Marking. Although there were many obstacles, this aim was completed with success. Automated Essay Marking is long from being widely implemented in the Education System. However, the results of this project proved that an accurate Automated Essay Marking system is not only possible but "inevitable" if the existing software is improved by implementing the future work described. The existing software may not be a solution to the problem but it is a significant attempt towards a solution. However, the most important accomplishment of this project is that the report in combination with the source code is a thorough guide to someone that wants to get involved with Automated Essay Marking with Artificial Neural Networks, or even Artificial Neural Networks in general.

The models designed, implemented, trained, tested and evaluated, produced a number of very important outcomes in regard to the significance of the Artificial Neural Network architecture, the essay 1-dimensional and 2-dimensional representation and the word-to-vector representation. In summary, after investigation, the outcome was reached that the essay representation is unimportant, in comparison to the word-to-vector representation, where a complex word-to-vector representation can produce better results as long as the model's depth can support its size. If the model's depth cannot support the size of the complex word-to-vector representation, then a simple word-to-vector representation will produce far better results and perform a more accurate classification.

The biggest challenge about Artificial Neural Networks is that the evaluation metrics help assessing the performance about the Artificial Neural Networks, but they provide limited information about what went wrong and what the next step should be. No matter how experienced one is with Artificial Neural Network, the only thing you can do is speculate and test. On the other hand, there is a limited number of speculations I could make taking into consideration the short time frame and limited memory capabilities. The development of a fine-tuned Deep Neural Network that can carry out demanding tasks such as text classification require a combination of time and computing power of which I had none.

Probably the major drawback of the software developed for the purposes of this project is that it must be both trained and evaluated on a set of essays with the same topic. Thus, it is unlikely that it will be able to successfully work and predict a grade with some success and accuracy for a different set of essays with a different topic. The reason lies on the way the models work, which is finding the local indicators/word that are able to point to a specific class membership. Considering, however, the task for which the software was made makes this drawback absolutely normal and was expected from the beginning.

The investigation of the underlying technology required for an Automated Essay Marking system also gave me a lot of food for thought. When I started this project, I was puzzled by the fact that education and automated essay marking systems have not been affected by the technological improvement. However, after carefully studying the field of Machine Learning and Artificial Neural Networks, I reconsidered. Undoubtedly, a successful Automated Essay Marking system could save both time and resources by automating the marking process. Essays could be converted to bag of words and inserted into a model, which could classify the essays based on local indicator. But are essays just bag of words? Are local indicators enough to classify an essay? Are we willing to sacrifice creativity to achieve efficiency? The objective of this project was to investigate whether Machine Learning and Artificial Neural Networks could be used to make an Automated Essay Marking which can classify essays with high accuracy. This objective has been, in my opinion, completed. The software developed with some modifications and improvement which were described in the “Future Work” section could and probably will lead to an Automated Essay Marking with great accuracy, which can be used in schools. But should it?

8. REFLECTION ON LEARNING

The final year dissertation is without a doubt a lot different to any of the other projects undertaken. It requires a lot of commitment and dedication to conduct a successful investigation and a thorough report. Thus, the learning outcomes are also different to the learning outcomes of any other project.

First of all, I chose this dissertation topic because I am intrigued by Machine Learning and planning to continue my studies in that field. When I started the project, my knowledge in Machine Learning was limited and I had no idea what Artificial Neural Networks are. Less than four months later, I achieved studying and understanding Machine Learning and Artificial Neural Networks down to the detail, becoming very familiar with two different libraries, Gensim and Tensorflow which are used for Machine Learning applications, constructing close to 100 models using the libraries, training many of them, evaluating and changing them, and finally constructing a thorough report of the entire investigation. Thus, I acquired technical knowledge that I hope to use in my later career.

The dissertation is the first project without a safety net. In contrast to the rest of the projects, which are made to be doable, the investigation is very likely to fail. Until the approach and implementation part of the project is over and the results are evaluated, there are no indications about the success of the project. In that respect, the dissertation is very similar to the real professional projects that we will undertake in our future careers and thus provide us with important learning outcomes and prepare us for our professional career.

The design of a successful Artificial Neural Network requires tremendous patience and concentration. In the projects I undertook before the dissertation I was able to make mistakes without a "cost". I was able to see my mistakes and run the program again. The time required to train an Artificial Neural Network, which can be days or an entire week, multiplies the "cost" of every mistake. So, working on a Machine Learning project requires cautiousness and diligence, which I have gained after this project.

Last but not least, this was the first project, which other than time management also required resources management. The limited memory capabilities of my laptop and the machines of the Linux labs forced me to look for alternatives, such as the insista@cs.cf.ac.uk server provided by

my supervisor. Due to my lack of experience in working on a remote server, I faced a lot of difficulties working with the ssh protocol necessary. After a lot of struggle, I overcame them, which helped me perform the trainings faster, but also helped acquire useful technical knowledge in working on remote servers. However, even the remote server had limited capabilities. So the models designed and implemented had to be big enough to perform the required task but at the same time relatively small to be trained by the server. This added an extra constraint to the developing process, which made the process more difficult, but also provided me with important learning outcomes for my later career.

REFERENCES

(PEG), P. E. G., n.d. *Measurement Incorporated.* [Online]
Available at: http://www.measurementinc.com/products-services/automated-essay-scoring

Beck, K., Beedle, M., Bennekum, A. v. & al, e., 2001. *Manifesto for Agile Software Development,* s.l.: s.n.

Bhande, A., 2018. *Medium.* [Online]
Available at: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

Bishop, C. M., 2006. *Pattern Recognition and Machine Learning.* s.l.:Springer.

D.T. Pham, X. L., 1995. *Neural Networks for Identification, Prediction and Control.,* Springer.

Deep Learning Course Wiki, n.d. *Log Loss.* [Online]
Available at: http://wiki.fast.ai/index.php/Log_Loss

defaultdict, 2018. *Python: defaultdict from collections.* [Online]
Available at: https://docs.python.org/3.6/library/collections.html#collections.defaultdict

eRater, n.d. *eRater.* [Online]
Available at: https://www.ets.org/erater/about

Fauske, K. M., 2006. *Neural Network,*

Goldberg, Y., 2016. *A Primer on Neural Network Models,*

Goodfellow, I., Bengio, Y. & Courville, A., 2016. *Deep Learning.* s.l.:Massachusetts Institute of Technology.

Graupe, D., 2013. *Principles of Artificial Neural Networks.* 3rd ed. s.l.:World Scientific.

Grefenstette, E., Blunsom, P., Freitas, N. d. & Hermann, K. M., 2014. *A Deep Architecture for Semantic Parsing,* s.l.: Department of Computer Science, University of Oxford.

Intellimetric, n.d. *Intellimetric.* [Online]
Available at: http://www.intellimetric.com/direct/

James, G., Witten, D., Hastie, T. & Tibshirani, R., 2013. *An Introduction to Statistical Learning.* 8th (2017) ed. s.l.:Springer.

Kaggle, 2012. *Kaggle: The Hewlett Foundation: Automated Essay Scoring.* [Online]
Available at: https://www.kaggle.com/c/asap-aes

Kalchbrenner, N., Grefenstette, E. & Blunsom, P., 2014. *A Convolutional Neural Network for Modelling Sentences,*

Kim, Y., 2014. *Convolutional Neural Networks for Sentence Classification,*

Manning, C. D., Raghavan, P. & Schütze, H., 2009. *An Introduction to Information Retrieval.* Online Edition ed. s.l.:Cambridge University Press.

Mcculloch, W. S. & Pitts, W., 1943. *A logical calculus of the ideas immanent in nervous activity*

Mihált z, M., 2016. *Github.* [Online]
Available at: https://github.com/mmihaltz/word2vec-GoogleNews-vectors

Mikolov, T., Chen, K., Corrado, G. & Dean, J., 2013. *Efficient Estimation of Word Representations in Vector Space*

Mitchell, T. M., 1997. *Machine Learning.* s.l.:McGraw-Hill International Editions.

Mitra, N. et al., 2018. *Machine Learning Introduction,* s.l.: s.n.

Nigrin, A., 1993. *Neural Networks for Pattern Recognition.* s.l.:Massachusetts Institute of Technology.

Nilson, N. J., 1998. *Introduction To Machine Learning.* s.l.:Stanford University, Department of Computer Science.

Nir, S. M., 2011. 7000 Private School Applicants Got Incorrect Scores, Company Says. *The New York Times*, 11 4.

NumPy, 2018. *NumPy: Datatypes.* [Online]
Available at: https://docs.scipy.org/doc/numpy-1.13.0/user/basics.types.html

NumPy, 2018. *NumPy: General.* [Online]
Available at: http://www.numpy.org/

Page, E. B., 1967. *Statistical and linguistic strategies in the computer grading of essays*.

PaperRater, n.d. *PaperRater.* [Online]
Available at: https://www.paperrater.com/

Řehůřek, R., 2009. *Gensim: About.* [Online]
Available at: https://radimrehurek.com/gensim/about.html#

Řehůřek, R., 2009. *Gensim: Corpora and Vector Spaces.* [Online]
Available at: https://radimrehurek.com/gensim/tut1.html#from-strings-to-vectors

Renals, S., 2016. *Stochastic Gradient Descent; Classification*

Rumelhart, D. E., Hinton, G. E. & Williams, R. J., 1986. *Learning Representations by back-propagating errors*

Schalkoff, R. J., 1997. *Artificial Neural Networks.* s.l.:McGraw-Hill.

Silvescu, A., 2000. *Fourier Neural Networks*

Smith, L. N., 2017. *Cyclical Learning Rates for Training Neural Networks*

TensorFlow, 2015. *TensorFlow: Homepage.* [Online]
Available at: https://www.tensorflow.org/

TensorFlow, 2018. *TensorFlow: A Guide to TF Layers.* [Online]
Available at: https://www.tensorflow.org/tutorials/layers

Yin, W., Kann, K., Yu, M. & Schütze, H., 2017. *Comparative Study of CNN and RNN for Natural Language Processing,* s.l.: s.n.

Yin, W. & Schütze, H., 2016. *Multichannel Variable-Size Convolution for Sentence Classification,* s.l.: s.n.

Zhang, Y. & Wallace, B., 2015. *A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification,* s.l.: s.n.