LU TP 19-35
May 2019

On the Effectiveness of Handcrafted and Learned Features in Automated Essay Scoring

Edvin Jakobsson

Department of Astronomy and Theoretical Physics, Lund University

Master thesis supervised by Mattias Ohlsson

Abstract

The task of Automated Essay Scoring (AES) has been active for more than half a century, starting with handcrafting statistical features used for linear regression, and currently being improved by the latest advancements in machine learning and natural language processing. Most current research uses some form of character or word embeddings to represent the essays rather than statistical features, enabling the models to analyze the text in full and automatically learn what to look for. Handcrafted features have possibly reached their maximum potential, and have been shown to be outperformed by more complex representations of textual data. However the fundamental differences between handcrafted and learned features have not been properly documented, nor their fundamental strengths and weaknesses compared.

In this paper we compare two different kinds of models for automated essay scoring, a Multilayer Perceptron (MLP) using handcrafted features and a standard Convolutional Neural Network (CNN) using word embeddings. The models are trained and tested and their strengths and weaknesses are discussed. We show that a simple CNN outperforms the MLP using handcrafted features, but that the MLP is a viable method to use for small tasks because of the easier implementation and shorter training time. We also provide some tips and suggestions when constructing a CNN for AES, and we discuss a potential downside of the quadratic weighted kappa score that is sometimes a suggested validation metric for AES-systems.

Populärvetenskaplig sammanfattning

När man talar om framtiden finns det ett begrepp som dyker upp i nästan varje diskussion: ”artificiell intelligens”. Vår fascination kring framtidens potentiellt självtänkande robotar speglas i hur mycket det diskuteras i alla yrkesgrupper just nu, oavsett om ämnet är koldioxidutsläpp, självkörande bilar eller automatiska inköpslistor. Men hur långt har egentligen maskininlärning kommit idag? Hur intelligent är dagens artificiella ”intelligens”?

En viktig aspekt inom maskininlärning är att kunna få en dator att förstå mänskligt tal och skrift. Många yrken idag hade betydligt underlättats om datorer kunde hjälpa till med monotona uppgifter som till exempel att tolka läkarjournaler eller betygsätta labbrapporter. Lärare lägger otaliga timmar på att rätta prov och rapporter, tid som annars hade kunnat spenderas på undervisning och utveckling. Uppgiften är enformig och sällan särskilt givande, men oundviklig och otroligt viktig att få korrekt och rättvis. Intelligenta program, kapabla till att läsa en text och kunna ge insiktsfulla kommenterar och poäng hade varit ovärderliga för lärare, och det hade kunnat spara samhället ofantliga resurser.

Utvecklingen av provrättande program påbörjades redan i samband med datoriseringen av samhället, och mycket har hänt i deras struktur och effektivitet sedan dess. De tidigaste modellerna gjorde kvalificerade gissningar om texten baserat på statistiska värden så som textens längd eller antalet stavfel. Dagens modeller baseras på en helt annan typ av intelligens som påminner om hur ett barn först lär sig att läsa. Programmen exponeras för ofantliga mängder text och får själva lära sig vad som skiljer en halvfärdig inköpslista från ett litterärt mästerverk.

Så hur väl fungerar de här nya programmen? Hur väl förstår de vad en riktig människa tänker eller skriver? Och hur användbara är de i dagens samhälle?

Contents

1 Introduction

Grading student essays is an unavoidable part of the current education system, yet the process is both time-consuming and expensive. The task is performed by people who often work full time as professional teachers and the task takes up a lot of time that could otherwise been spent on teaching. This is an even greater problem in Sweden, that is currently moving towards a crises because of a massive lack of teachers, (Lärarförbundet, 2018). For many teachers, the task of grading papers, essays or lab reports is often considered to be monotonic and tiresome, and it is, if possible, often passed to someone else. There is also the aspect of subjective bias grading, as many factors such as personal opinion or the state of mind of the grader my affect the outcome. It is quite common that the person doing the grading is employed at the same institute as where the students took the exam. All these factors are problematic when considering that the system is supposed to evaluate students competing for future gains and opportunities.

Automated Essay Scoring (AES) is an attempt to introduce computer software for precise, fair and automatic grading of human written texts. The system is often not supposed to be a complete replacement of the human grader, but rather a tool to heavily speed up the process and to impose guidelines to the task. Historically, the origin of AES goes back to 1960's and the works of Ellis B Page (1968), who is often considered to be the founding father of AES. Page made use of handcrafted features, statistics chosen to be extracted from each essay in order to represent it, and then trained a regression model on the data. Features were for example the total number of words, number of sentences, average length of words, number of adjectives, and so on. However useful, common critique to these handcrafted features is that it mainly captures the surface structure of an essay, with little to no insight as to the deeper semantic meaning of the text, (Chung and O'Neil Jr, 1997). Another drawback is feature engineering, as it can be quite time-consuming to decide on which features to use, as well as to extract them. A lot of prepossessing of the text is needed for certain features, such as syntactic analysis in order to find verbs, adjectives and so on.

In more recent years the subject of Natural Language Processing (NLP) has made major advancements when creating language models, especially with the help of artificial neural networks. It is possible to completely avoid the drawbacks of handcrafted features using neural networks, as they have been shown to be capable of finding semantic and syntactic features in a text automatically. Recent approaches has consisted of creating networks such as Hierarchical Convolutional Neural Networks, (Dong and Zhang, 2016), Long-Short Term Memory networks, (Alikaniotis et al., 2016), and Recurrent Convolutional Neural Networks, (Dong et al., 2017). These have shown to surpass the manually designed features on many tasks, as well as being more robust across different domains, (Dong and Zhang, 2016). They also hold some of the benchmark results on the data set used in this research, and will hence be used when comparing the effectiveness of the networks created here.

The purpose of this paper is not to compete with these complex and perfected networks for new results, as this would be far out of reach for the level and time frame of this

research. Instead the goal is to compare the effectiveness of the two types of features, handcrafted and automated, on somewhat simple networks. This is done by implementing shallow networks that rely mostly on the fundamental building blocks of the two methods. This will serve as an indication of how these methods compare when time, manpower and computer processing power is limited in a project, and could be viewed as guidelines as to which method to choose depending on the limits of those factors.

2 Theory

2.1 Analyzing an Essay

The task of analyzing an essay is for a computer quite demanding. Computers inability to read, understand, or to think for themselves makes this task problematic, and computers are only created to interpret one kind of input: numbers. A major part of creating a software with the ability to analyze a text is therefore to first decide on a procedure in which to create a string of numbers that can accurately represent the text. A fundamental question is of course whether this is possible without losing any of the information that a human written text contains. The answer to this question is "yes", as the text is in itself only a series of symbols extracted from a restricted set: the alphabet. It should be noted, however, that a series of numbers to the computer is not read in the same way as a human is reading a sentence. A human derives meaning and actions from the words, while the computer derives only mathematical values that can be applied to mathematical formulas.

A large part of Natural Language Processing comes down to how to best represent the text we want to analyze as a series of numbers in order to get the best performance out of the software. This can be done in many different ways and they are not all equal in adequacy, which brings us to one of the main points of this paper: handcrafted features.

2.2 Handcrafted Features

A common approach when trying to automatically classify written text is to decide on a series of specific characteristics of the text and then make a decision based on only those statistics. For example, it is quite effective to flag spam email by counting the use of specific words such as "free", "cash" or "viagra". This approach is not uncommon in the pursuit of Automated Essay Scoring (AES) either, and a lot of research has been done on what features to extract from the essay in order to analyze it, (Page, 1968). A few of the most useful features to use are the number of words in the essay, their average length, the number of commas, or the number of spelling errors and so on. These values can simply be extracted from the essay and together they create a basic representation of the text.

While this method is rather simple it can still be extremely effective when used in AES. As shown in this paper, representing each essay as only four different extracted values could be enough information for a network to reach similar accuracy as human expert graders on

certain basic tasks. However, simple networks comes with some severe weaknesses. Grading real essays is a serious task and if an automatic approach is to be implemented in real schools then they have to be extremely accurate, fair, and difficult to cheat. By knowing exactly what information is fed to the network and what is discarded it would be easy to write a nonsense-essay that would still be awarded a very high mark. Therefore, accuracy on real attempts to write an essay is not the only factor when evaluating an AES-system, and the complexity of the task of grading an essay must be reflected in the software. Nevertheless, most research on AES does not take this into consideration, this paper included, and the results are only measured in the accuracy of the automated grading. This is by no means a product ready for commercial use, and general automated essay scoring still needs more work before it could be implemented in real situations effectively. However, with more complex networks more of the information in the text can be analyzed properly, which should increase the performance of the network as well as diminish the possibility of cheating it. But in order to achieve this we need to move away from handcrafted features.

2.3 One-hot Word Vectors

If an AES-system is to truly reflect the work of an expert human grader then it must not discard any of the information that the essay contains. The two most common ways of transforming textual data into vectors is by either the use of *one-hot vectors* or by *word embeddings*, (Goldberg, 2017). One-hot encoding is the process of mapping items in a list of categorical data to a set of binary vectors of the same dimension as the list. Each item is represented by a dimension in the vector, setting that entry to a non-zero value, while all other entries are zero. For example, encoding a list of four words would result in four unique one-hot vectors, each being non-zero in only one dimension, see figure 1.

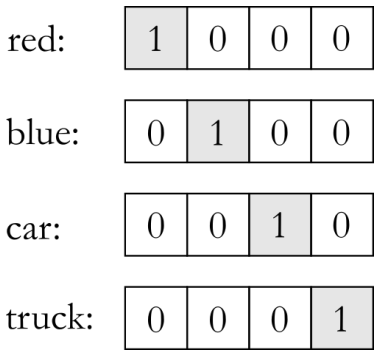

Figure 1: Example of a one-hot encoding of a set of four items.

Textual data can be encoded at different levels, the lowest one being at character level. In this case each possible character is assigned a one-hot vector, which in turn makes it possible to represent any text as a matrix. Since there is more meaning in a word than in individual characters however, it is more common to use words as the fundamental building block of textual data. By representing a text as a series of words rather than characters the

task of analyzing it becomes much more manageable. A word representation does however affect the dimensionality of the problem, as the sparsity of the one-hot vectors scale with the vocabulary. If the vocabulary encoded is 40.000 words long then each word vector will contain that same number of dimensions. This becomes a computational problem when creating an AES-system, as neural networks are not optimized for sparse, high-dimensional vectors. Another drawback of the one-hot encoding is that the connection between similar words is non-existent, since all word vectors are independent from each other. The word "green" is as different from the word "blue" as it is from the word "idea". This becomes clear when looking at the dot product of any two words, as it is always zero. Before creating a classifier for the textual data it is therefore advantageous to first implement *word embeddings*.

2.4 Word Embeddings

Instead of representing each word as a unique dimension, the multidimensional vector space can be optimized by enabling each word vector to stretch in every dimension, utilizing all the vector space, see figure 2. The issue discussed above referring to dissimilarity between correlated words is then resolved, as similar words can have similar yet distinctive word vectors. The drawback of sparse vectors can also be addressed using this technique as word vectors are allowed to be linearly dependent. The number of dimensions in the vector space can then be lowered without making any of the word vectors parallel to each other. These dense word vectors are often referred to as word embeddings, arguably first mentioned by Bengio et al. (2003). The number of dimensions is decided on by the researcher and is always lower than the length of the vocabulary. Common values range between 50 and 300 dimensions, and for the research regarding this paper a vector space of 100 dimensions was used. The values for each word vector could either stay fixed or be included in the training of a network, and they can either be initially randomized or pre-trained on textual data.

red:	0.23	0.21	0.07	0.13
blue:	0.20	0.25	0.16	0.05
car:	0.04	0.07	0.31	0.26
truck:	0.01	0.02	0.22	0.30

Figure 2: Example of four-dimensional word embeddings for four words.

2.5 Pre-trained Word Embeddings

By first creating one-hot encoding for the vocabulary used it is possible to let a network create and train its own word embeddings. This is achieved by letting the input of the network be the one-hot word vectors and feeding them directly into an embedding matrix, where each row represents the word embedding of a word. Multiplying the one-hot vector with the matrix results in a vector representing that word's embedding which can then be fed into the next layer of the network, see figure 3. This way the input stays the same while the values in the embedding matrix can be trained along with the rest of the network. This does however require supervised training, and the word embeddings will be trained to enhance the networks performance on the training data used. Since the data set in a research is always limited and, at least in AES, never quite large enough, training the word embeddings on the same data as is being evaluated does quickly become a problem of over-fitting. The word embeddings will learn characteristics not from the general meaning of the words, but rather from how they are being used in these particular essays. Fortunately there are several methods for learning word embeddings using unsupervised learning, which makes it possible to train the word embeddings using gigantic amounts of text. This creates embeddings that represents a more general meaning of each word, similar to how a human learns to read by first reading many other texts before taking on for example a paper in advanced physics. An early approach to unsupervised training of word embeddings came from Collobert and Weston (2008), where they trained a neural network to predict a word in a text by looking at the words immediately preceding and exceeding it. Using this technique it is possible to train on huge unlabeled text corpora, creating word embeddings that can then be used in any number of tasks. The trust in these embeddings is based on the *distributional hypothesis*, (Harris, 1954), stating that *words are similar if they appear in the same context*. This is clear when taking the example of synonyms, which are usually interchangeable in a sentence.

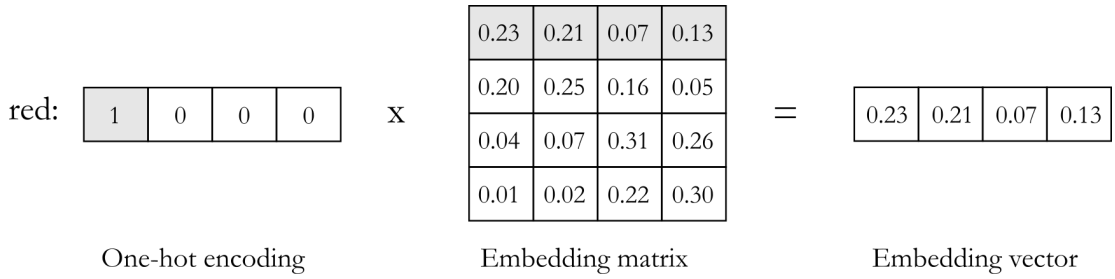

Figure 3: Example of how an embedding matrix can be a trainable part of a network without changing the one-hot encoding input.

2.6 Word2Vec and GloVe

In 2013, a new family of algorithms for word embeddings called Word2Vec was created by a team at Google, (Mikolov et al., 2013a,b). The Word2Vec model utilizes either

of two architectures: Continuous Bag-Of-Words (CBOW), or continuous skip-gram. In the CBOW model, a neural network tries to predict a word in a text from a window of surrounding context words, while the skip-gram model does the opposite, and tries to predict the surrounding words by looking at the center word. According to the authors' note, the CBOW model is faster to implement and train while the skip-gram model is slower but does a better job for infrequent words(Mikolov et al., 2013a). Pre-trained word embeddings from Word2Vec are available for download and use by Google.

Mikolov et al. (2013a) also found that the word embeddings were able to find syntactic and semantic patterns, which could be proved using vector arithmetic. Patterns between words such as "Man is to Woman as King is to Queen" could be generated by algebraic operations on the vector representation of these words. By taking the word vectors for "King", subtracting the word vector for Man and adding the word vector for Woman a new word vector was created. When comparing which word vector was closest to this newly created vector in the vector space it was found to be the vector for the word "Queen". Similar relationships for many other words were also possible, such as semantic relations like Country-Capital as well as syntactic relations such as do-did-done and eat-ate-eaten.

Another common pre-trained word embedding database for use is that of GloVe (Global Vectors), (Pennington et al., 2014). The GloVe model looks at a global word-to-word co-occurrence matrix of the words in the corpus in order to create the embeddings. It is essentially a log-bilinear model with a weighted least-squares objective. This is a costly method when dealing with such large corpora, but it is a one-time cost. GloVe is developed as an open-source project at Stanford University, and pre-trained word embeddings with dimensions of 25, 50, 100, 200 or 300 are all available for download at their website[1]. For the research in this paper, the word embeddings used for the convolutional neural networks was that of the GloVe pre-trained embeddings with 100 dimensions, (Pennington et al., 2014).

2.7 The Multilayer Perceptron

The Multilayer Perceptron (MLP) is among the simplest of neural networks. It is essentially a mathematical function mapping from one vector space to another, where the structure of those vector spaces as well as what they represent always mirrors the training data used for the network. If a data point and the desired output can both be described as a vector then pre-labeled data can be used to train the network to optimize the mathematical function transforming the input vector to the desired output vector. For example, a multilayer perceptron for grading essays could look as follows:

A vector representation of each essay is created by extracting some statistical features. The output of the network is also a vector, so its shape will be designed to represent the desired conclusion, for example each value may represent the probability of an essay deserving a certain score. The network itself consists of a series of nodes and weights, creating a mathematical formula for how to transform the input vector to the output vector.

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

By feeding the network pre-labeled essays the mathematical formula can be updated to better map the inputs to the desired outputs. Afterwards the training the network can be used to make qualified predictions on new unlabeled data. See figure 4 for a structure of the network.

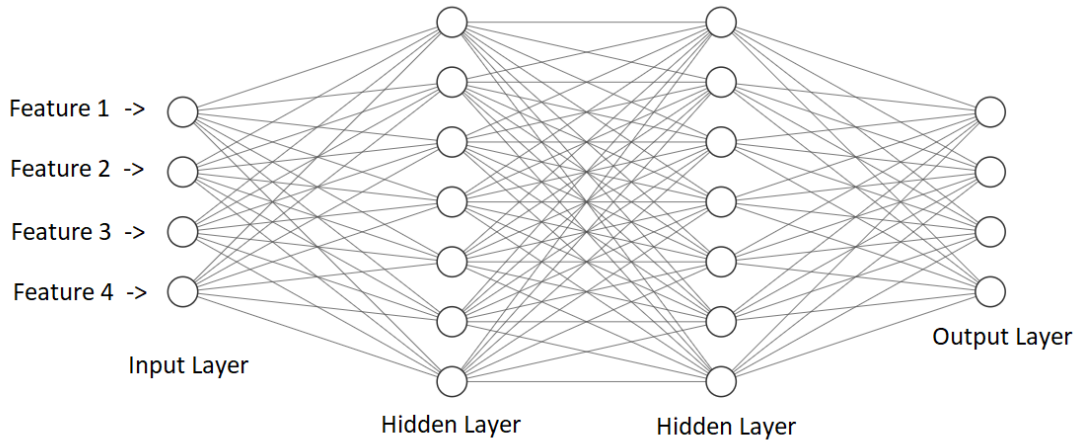

Figure 4: Example of a Multilayer Perceptron used for Automated Essay Scoring. The figure was created using a software by Alexander Lenail, `http://alexlenail.me/NN-SVG/`.

There are many sources for a more in-depth explanation of neural networks, for example Haykin (1994). Both the strength and the weakness of the multilayer perceptron come from its simplicity. It is rather easy and fast to both implement and train, however it is limited in its ability to find correlations between input and output when dealing with complex structures of data. It is considered a good candidate when analyzing an essay using handcrafted features, but not when attempting to analyze an essay represented as a huge matrix of word embeddings. For this complex structure of input it is more suitable to apply for example a convolutional neural network, famously proficient when dealing with large matrices as input, (LeCun et al., 1995).

2.8 The Convolutional Neural Network

The Convolutional Neural Network (CNN) is similar to the MLP in its fundamental mechanics, but quite different in its structure. While the MLP prefers a structure where all nodes in a layer are connected to all nodes in the next one, the CNN uses more complex patterns to allow parts of the network to extract different features in the data. This makes them useful when trying to detect patterns in a matrix, such as a face in an image. The idea of artificial convolutional neural networks were inspired by biological processes, and the connectivity pattern between the neurons resembles the organization of an animal visual cortex, (Fukushima, 1980). The CNN uses several filters or kernels to scan the input matrix looking for a pattern, each kernel trained by the network to find patterns useful

for whichever task it was trained on. The kernels are simply small matrices, and when multiplied by different parts of the image matrix they produce a high output only if they find the specific pattern they are designed to look for. Several convolutional layers can be used in series in the network, taking advantage of potential hierarchical patterns in the data. Between convolutional layers it is common to use pooling layers, which is a sort of non-linear down-sizing. A common pooling technique is the max-pooling, which divides a matrix into a set of non-overlapping matrices, and outputs the maximum value for each sub-region. These values are then used to reconstruct a smaller version of the initial matrix, which progressively reduce the spatial size of the representation and reduces the number of parameters used in the network. The trust in max-pooling is based on the idea that the exact location of a feature is less important than its relative position to other features. After a set of convolutional and max-pooling layers the network becomes fully connected in a layer of neurons that are then used to create an output for the network, see figure 5 for an example of the structure.

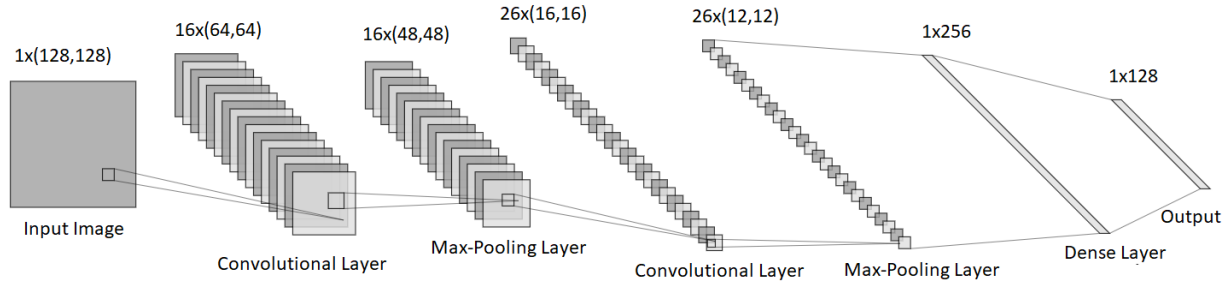

Figure 5: Example of the structure of a convolutional neural network. The figure was created using a software by Alexander Lenail, `http://alexlenail.me/NN-SVG/`.

Pattern recognition can be exceptionally effective in image analysis because it enables the network to detect an object regardless of its position in the matrix. This ability is important in natural language processing as well, since the meaning of a sentence is roughly the same no matter where in the text it is located. CNN's are therefore a realistic tool to apply when creating an AES-system that uses matrices as representation of text, (Zhang et al., 2015).

For a two-dimensional image the kernels needs to look at patterns in both dimensions, and each kernel matrix only covers a small portion of the image at a time. While a text represented by word embeddings also is a two-dimensional matrix, there is no need to look for patterns along the length of the word embeddings. A text essentially stretches in only one dimension, which is one word after another. It is therefore reasonable to make the kernel matrices as large as the word embeddings, so that they always see the complete word embeddings. The kernels can then scan the text, moving in only one dimension and looking for patterns a few words at a time, see figure 6. Each kernel creates a a vector of length:

$$\text{kernel output length} = \text{text length} - \text{kernel size} + 1$$

.

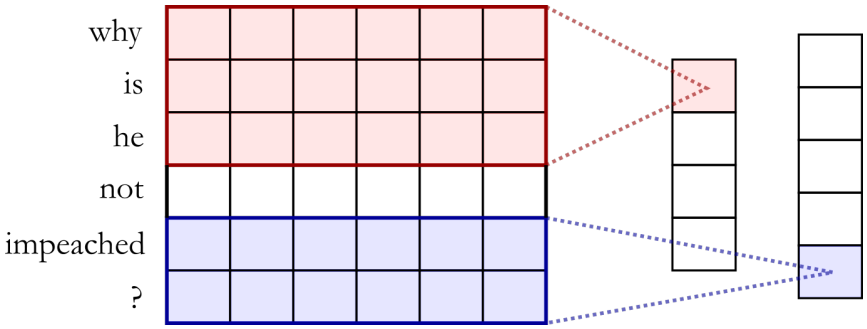

Figure 6: Example of two CNN kernels scanning textual data in only one dimension. The red (top) kernel looks at three words at a time and creates an output, resulting in a vector of length 4 for the sentence. The blue (bottom) kernel looks at two words at a time and outputs a vector of length 5.

2.9 Regression and Classification

Automated essay scoring can be treated as either of two problems, a classification problem or a regression problem. By viewing the possible grades for the essays as different classes the output of the network becomes categorical, and a softmax output model can be used. The softmax output is simply a value for every possible class, each representing the probability of the essay belonging to that class. Softmax output layers are very common in classification problems, for example when categorizing text genres or spam email. The drawback with softmax is that it treats all classes as independent and unique, while when a range of grades is an incremental scale. Using a softmax output layer the network does not utilize the fact that a score of 6 is closer to a score of 5 than to a score of 0. It can therefore be advantageous to treat AES as a regression problem instead. Using a singe output node the output can represent the exact score of the essay, and when a prediction needs to be made the output can be rounded off to the closest allowed grade. This allows the network to take advantage of the incremental scale that is the possible grades.

3 Data

3.1 Automated Student Assessment Prize

One of the most important criteria when training a neural network is having enough data, and in this line of research large data bases are very time-consuming to create. Thousands of essays are needed to create any sort of useful results in Automated Essay Scoring (AES), and they all have to be digitally stored along with a reliable score assigned to each essay. If this project were to gather its own data then there would had been no time left to perform any kind of experiments.

Instead, the data set used in this paper is that of the Automated Student Assessment Prize (ASAP)[2], provided by The Hewlett Foundation for a competition in AES launched on Kaggle in 2012. Even though the competition is over the data set is still available for anyone to use, which is why it is still being utilized by many researchers on this subject today. The data set contains almost 13.000 human written essays from students ranging from 7th to 10th grade. The essays are collected from 8 different tasks or prompts given to the students that they followed when writing the essays. For example the first of the prompts looked as follows:

"More and more people use computers, but not everyone agrees that this benefits society. Those who support advances in technology believe that computers have a positive effect on people. They teach hand-eye coordination, give people the ability to learn about faraway places and people, and even allow people to talk online with other people. Others have different ideas. Some experts are concerned that people are spending too much time on their computers and less time exercising, enjoying nature, and interacting with family and friends.

Write a letter to your local newspaper in which you state your opinion on the effects computers have on people. Persuade the readers to agree with you."

Each essay is graded by two independent human expert graders and in each prompt the rubric guidelines for scoring differs slightly. Because of the major differences between these 8 essay sets there is no straight-forward way of combining them into a single data set, so instead each model has to be evaluated on each of the sets separately. This is mainly because the essays are graded based on different factors and with different score ranges, and also because the students are of different ages and so their requirements differ. Even if the score ranges were the same it would not be fair to hold a 7th grade student to the same standard as a 10th grade student, and even the exact same essay should be rewarded different scores depending on which essay set it belongs to. However, the size, quality and diversity of this data set makes it a great choice for the research of this paper, and since the data is commonly used by other researchers the results found here can easily be compared to those of many others, as well as to the results of the competition held in 2012. For each essay set 30% of the essays were not used throughout the research regarding this paper, it was instead saved as a test set in order to validate the final models at the end of the project, see table 1.

In order to visualize the distribution of classes (grades) in the essay sets a series of histograms were created, see figure 7. These histograms also serves to give an understanding as to the level of difficulty within each set. The percent of the most frequent grade within a set acts as a minimum requirement of accuracy for any set of predictions, as a model that always guesses for the most common grade would reach this accuracy. The distribution of essays over the possible grades were similar in in both training- and test data.

[2] `https://www.kaggle.com/c/asap-aes/data`

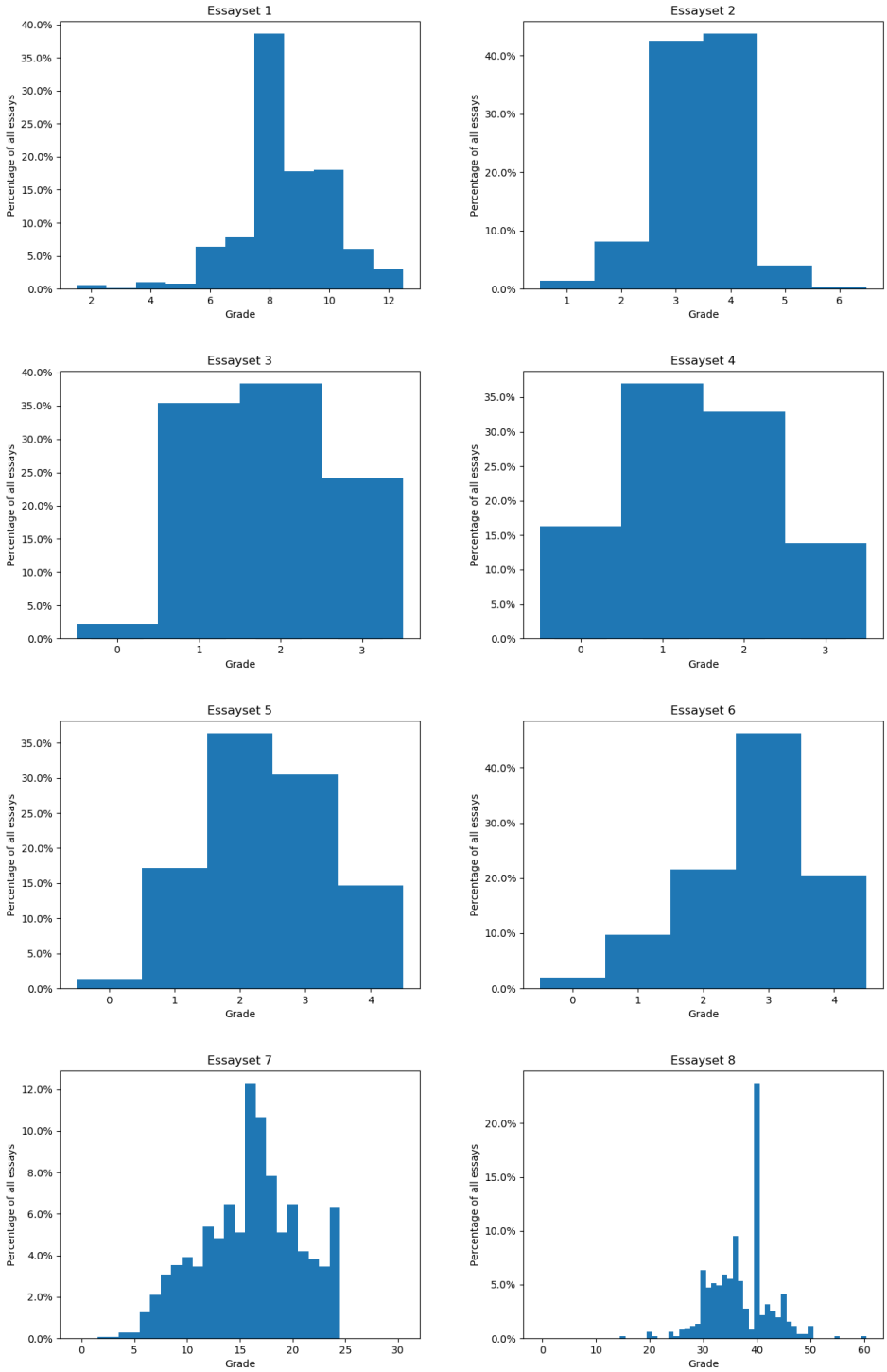

Figure 7: Histograms over the 8 different essay sets. Each histogram displays the distribution of the essays over the different grades.

ASAP Data Set

Set	**Avg Len.**	**Score Range**		**Number of essays**	
		Human	Resolved	Tr. and Val.	Test Data
1	350	1-6	2-12	1248	535
2	350	1-6	1-6	1260	540
3	150	0-3	0-3	1208	518
4	150	0-3	0-3	1239	531
5	150	0-4	0-4	1263	542
6	150	0-4	0-4	1260	540
7	250	0-15	0-30	1098	471
8	650	0-30	0-60	506	217

Table 1: Details on the ASAP data set. The human score is the score range given by each human expert grader, and the resolved score is the final score based on both graders. In set 1,7 and 8 the resolved score is the adjunction of the two human scores, hence the score range is doubled. For the research regarding this paper 30% of each essay set was left as a test set, and only used to evaluate the final versions of each model.

3.2 Evaluation

When deciding on the quality of an essay grader, human as well as machine, it is sub-optimal to look only at the percentages of correctly labeled essays. Since the grades cover a range from good to bad, a wrong prediction of a grade can still be either a good guess or a bad one. For example, if the target grade for an essay is "5" on a scale from 1 trough 5, it is better for a classifier to predict a "4" rather than a "1". Accuracy alone is hence not enough when determining the usefulness of the classifier, a stronger metric of quality is required.

The Automated Student Assessment Prize (ASAP) competition was constructed with an official criteria for evaluation metric, the Quadratic Weighted Kappa error function (QWK). The function compares two sets of classification outputs and produces a number between -1 and 1, 1 being total agreement, 0 being no more agreement than expected by chance, and -1 being complete disagreement between the two outputs. The QWK score is based on the distance between unmatching predictions, solving the problem described above. It does also adjust to agreement by chance, taking into consideration the difficulty of the classification based on the distribution of the target predictions over the different grades. For example, if a network is always only predicting the most frequent grade in an essay set with no regard to the actual essay the QWK will, due to its structure of compensating for agreement by chance, output a kappa score of zero. This means that any kappa score of above 0 is a direct implication that the network is basing its predictions on information contained within the essays themselves rather than some clever interpretation of the data set as a whole.

The Quadratic Weighted Kappa error function looks as follows:

The possible grades for a set of essays are 1,2...,N, and the essays are graded by two raters, Rater A and Rater B. An $N-by-N$ matrix O is constructed over the essay ratings, such that O_{ij} corresponds to the number of essays that received a rating i by Rater A and a rating j by Rater B. A second $N-by-N$ matrix of expected ratings, E, is calculated, assuming that there is no correlation between rating scores. This is calculated as the outer product between each rater's histogram vector of ratings, normalized such that E and O have the same sum. A third $N-by-N$ matrix of weights, w, is then calculated as follows:

$$w_{ij} = \frac{(i-j)^2}{(N-1)^2}$$

Using these three matrices the Quadratic Weighted Kappa can be calculated:

$$\kappa = 1 - \frac{\sum_{i,j} w_{ij} O_{ij}}{\sum_{i,j} w_{ij} E_{ij}}$$

This kappa value between -1 and 1 will be the standard metric for evaluating the networks created in this research. Using this method it is possible to evaluate the consistency between any two raters, for example comparing the output grades of a network with that of expert human graders. It is however also possible to compare the two human experts that provided their scores for the data set with each other, and hence get a better understanding of to what extent two expert human graders agree with one another when grading essays. This gives a good indication as to how well the network is performing compared to humans.

It should be noted that the QWK is not used by the network during training, it is only used to evaluate the predictions made by the network. During training the network tries to minimize a loss function, calculated using the difference between the desired and the actual output for each essay. The kappa score is only based on the rounded predictions made by the network, it is an enhanced version of the accuracy as it is affected by the incremental scale of the grades. The QWK is therefore a discrete function, as its output only changes if the rounded prediction of the network changes. This makes the QWK function unfit to be used directly as a loss function for the network.

4 Method

4.1 Implementation

This research was done in Python, with the use of Tensorflow, (Abadi et al., 2015), Keras, (Chollet et al., 2015), the Natural Language ToolKit (NLTK), (Loper and Bird, 2002), and Scikit-Learn, (Pedregosa et al., 2011). Apart for following a few examples from the Keras blog, (Chollet, 2016), only two more sections of code were implemented from other authors than myself. When creating the confusion matrix plots, the code used was that

of Scikit-learn's examples, (Scikit-Learn, 2011). For evaluating the networks, the Kaggle-competition at which the data was extracted from provided both a suggestion for a metric, Quadratic Weighted Kappa, and a complete function for implementation, (Hamner, 2012). The rest of the code in this research was written by the author of this paper.

4.2 Extracting Features

Already in the 1960's the foundation was laid of handcrafted features as a tool for AES, and the effectiveness of different features was documented, (Page, 1968). Based on Page's research, a list of features were chosen to be used that are expected to have a great correlation with the target grade for an essay in general. Page used 30 different features, and the initial idea in this research was to implement more and more features as results were reached. The accuracy of the network in this research however was already quite high with only a single feature, the number of words, and it plateaued soon thereafter. The implementation of more features were therefore soon abandoned so that the time could be focused on further research instead. In the end the features used were:

- Length of essay in number of words
- Average length of each word
- Standard deviation of length of each word
- Dale-Chall readability score

The Dale-Chall readability score is a measurement of the difficulty of a text. It is based on a list of 3000 words that groups of fourth-grade American students could reliably understand. Word not on the list are considered "difficult", and a score is calculated as:

$$\text{Score} = 15.79 \left(\frac{\text{difficult words}}{\text{words}} \right) + 0.0496 \left(\frac{\text{words}}{\text{sentences}} \right)$$

For each essay the four features were extracted and put into a vector representation of that essay. After normalization, these vectors were the only input for the multilayer perceptron that was used to grade the essays.

4.3 The Multilayer Perceptron

The multilayer perceptron networks created for this research were made using the Keras library. The final network consisted of three hidden layers with 50 neurons each, and a batch size of 32. Two output structures were compared, that of a softmax output layer and that of a single output node with a linear activation function, see figure 8. The network consisted of approximately 5400 trainable parameters. The input of the network was a four dimensional vector consisting of the four handcrafted features extracted from

each essay, and the vectors were normalized before training. The classification-model used the sparse-categorical-cross-entropy as loss function, while the regression model used the mean squared error loss function. A K-fold cross validation of 5 splits of the data was implemented to further justify the results of the network. For regularization dropout was added and tuned.

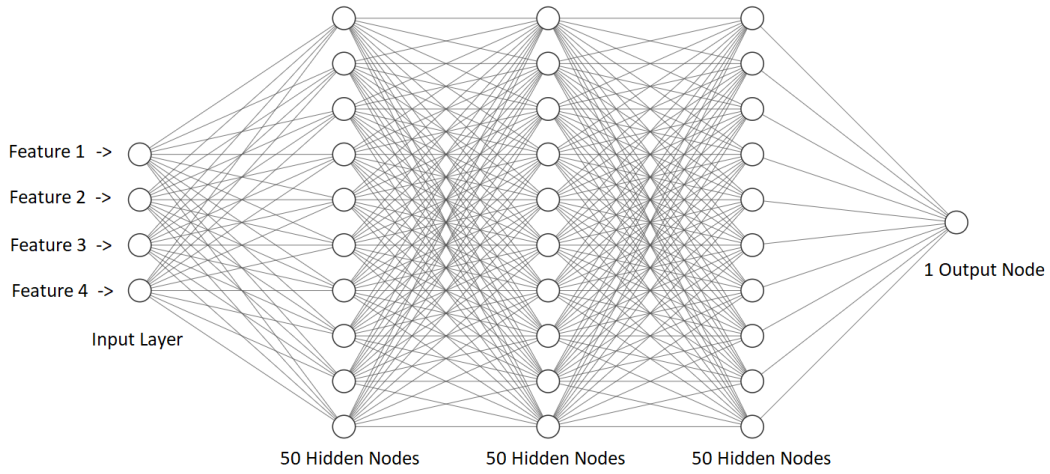

Figure 8: The structure of the multilayer perceptron used. For visualization not all 50 nodes are visible in the hidden layers. The figure displays the MLP regression-model, which uses a single output node with a linear activation function. The figure was created using a software by Alexander Lenail, `http://alexlenail.me/NN-SVG/`.

4.4 The Convolutional Neural Network

Most of the time given to this research had to be spent implementing and improving the Convolutional Neural Network (CNN) and its parts. Transforming a set of texts into a one-hot encoding of their library ready to be fed into a pre-trained word embedding matrix is in itself a demanding first time experience, as well as implementing and training the CNN.

The CNN was implemented and many of its hyper-parameters tuned, such as the number of convolutional layers, the number of kernels, the size of the kernels and the size of the max-pooling window. A few regression techniques were tested including dropout, batch normalization and the L2-regularizer. Different structures of the CNN were also compared, such as global max-pooling or adding a fully connected dense hidden layer after the convolutional layers. The final models used two convolutional layers, each with 100 kernels with a word size of 3, max-pooling layers with a window size of 5 and a final fully connected dense layer with 100 hidden nodes. The size of the network was also based on the longest text in each essay set, which was 783 words for essay set 1. See figure 9 for a structure of the CNN used for set 1. These models used the L2-regularizer, without dropout or batch normalization, and batch size of 40.

Different approaches to word embeddings were implemented, such as fixed or train-

able embeddings as well as randomized or pre-trained GloVe embeddings. The network consisted of approximately 360 000 parameters and the embedding layer of 1.3 million parameters, making a total of 1.66 million trainable parameters when the embeddings were trained as well. Finally, all models were tested using either a softmax output layer or a single output node with a linear activation function. Because of the time needed to train each model, no K-fold cross validation was implemented for the CNN.

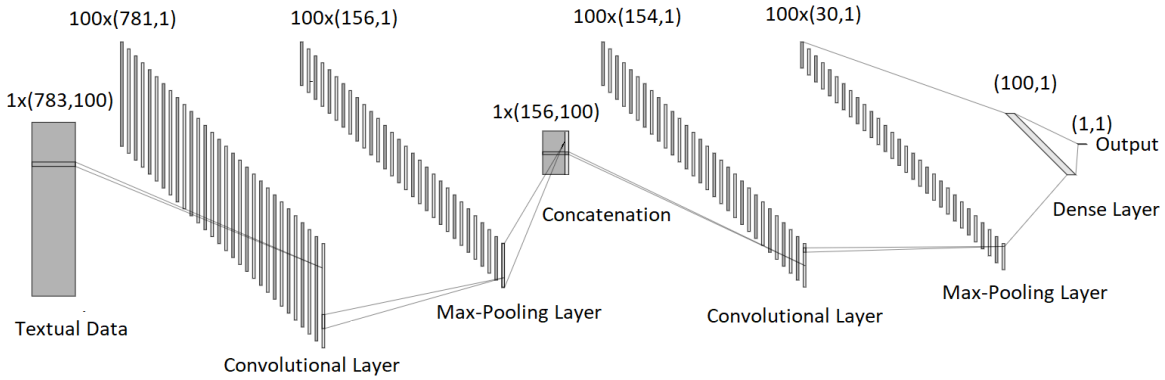

Figure 9: The structure of the CNN used for essay set 1, using an input length of 783 words. The network structure is the same for the other essay sets, except that the input size changes, and hence the length of each kernel output vector. The figure was created using a software by Alexander Lenail, `http://alexlenail.me/NN-SVG/`.

5 Results

5.1 The Multilayer Perceptron

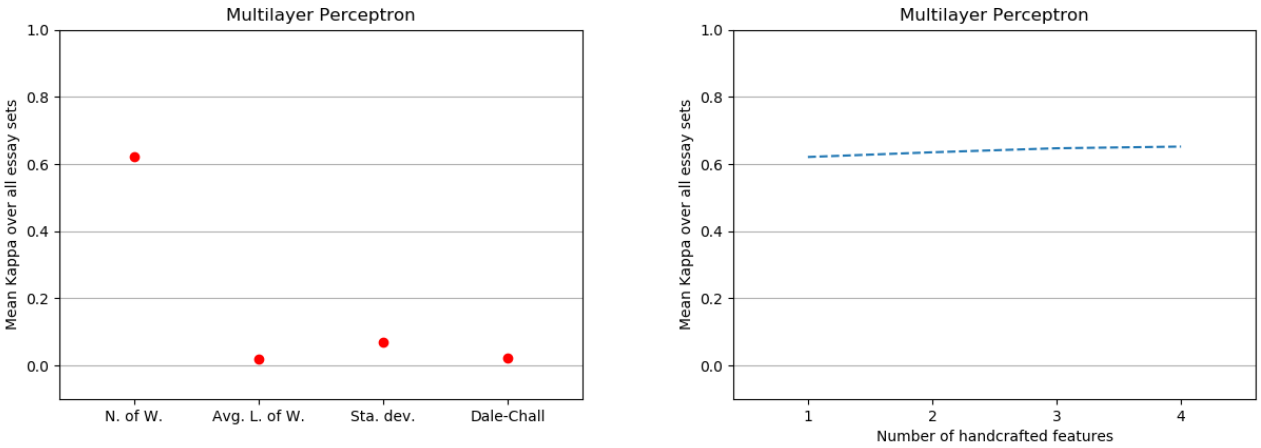

Figure 10: The performance of the multilayer perceptron. The left graph displays the performance of four models, each using only one feature as input. The right graph displays the mean quadratic kappa score as a function of the number of features used to represent an essay. The order of the features are: 1: Total number of words. 2: Average length of each word. 3: Standard deviation of the length of the words. 4: Dale-Chall readability score.

Extracted Features

When comparing the four handcrafted features chosen based on the work of Page (1968), it became clear that the most important feature by far was the total number of words in the essay. Using only this feature an overall quadratic weighted kappa score of 0.621 was reached, and adding the other three only manage to improve it to a total of 0.648, see figure 10. This is the mean score over all the essay sets, but similar results echoed for each individual set as well. These results were reached using a MLP with three hidden layers, each with 50 nodes, and a single linear output node.

Regularization and Network Structure

It may seem excessive to use three layers each with 50 nodes when the input vector only contains four values, but it was found that such a large network was needed to push the MLP to its maximal potential, see figure 11. Though surprisingly large, the network structures preferred by this investigation were used to obtain the results presented in this paper. However, with such a large network and a rather small data set, the network is in danger of over-training and over-fitting to the training data. To test this a network

was trained with an excessive amount of epochs, yet no divergence between training and validation was found, see figure 12. A regularized version was also tested using dropout, however this did not enhance the performance of any kind, and it only slowed down the training process.

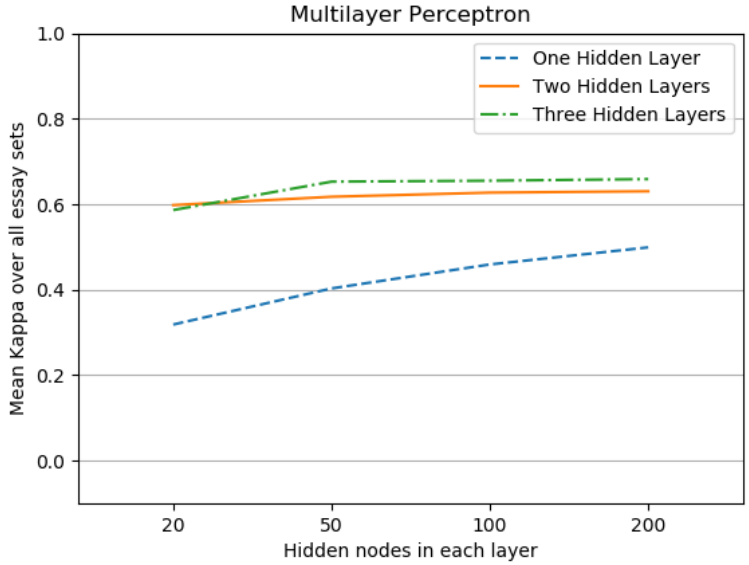

Figure 11: The performance of the MultiLayer Perceptron as a function of its size. All four handcrafted features are used as input for the models.

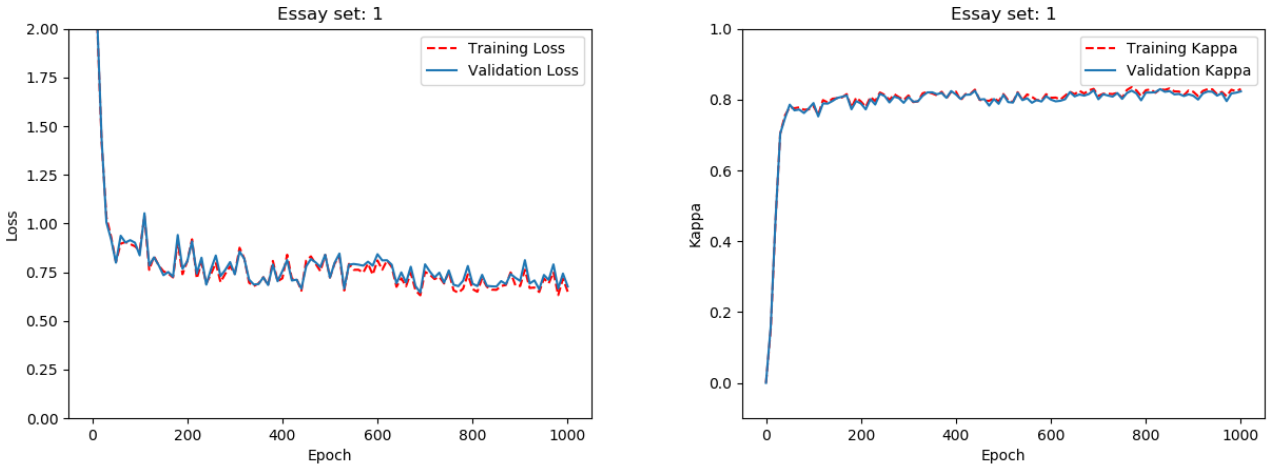

Figure 12: The training of the multilayer perceptron. The left graphs displays the sparse categorical cross-entropy loss during training, and the right graphs displays the quadratic weighted kappa score during the same period.

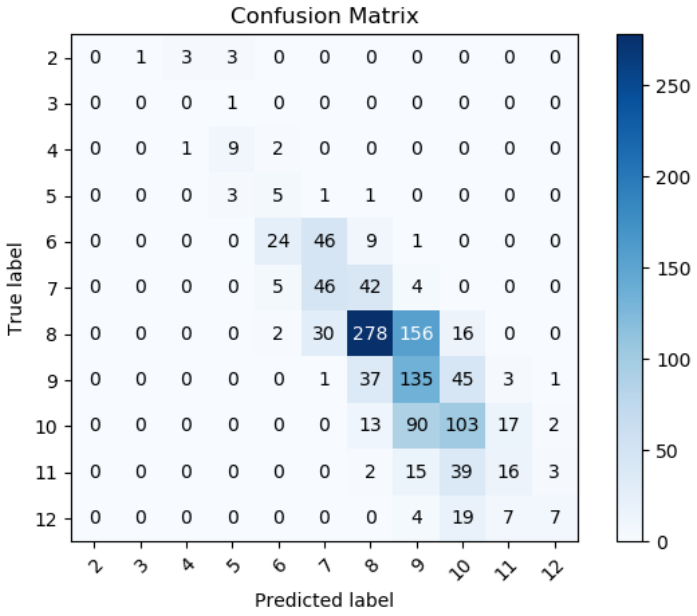

Figure 13: A visualization of the difference between two classifiers. The true label is the resolved score of the two human experts and the predicted label is that of the multilayer perceptron. The values in the matrix describes how many essays that were classified as that particular grade. The results are found using the MLP regression-model on essay set 1.

Confusion Matrix

While the quadratic weighted kappa score is a useful metric for the accuracy of a classification problem with dependent classes, it does not give a satisfactory visualization of what the network is doing, or how it is performing. In order to visualize the predictions made by each network model its output was used to create a confusion matrix, displaying how many essays were labeled as each grade. The rows in the matrix represent the true grade of the essays while the columns represent the predicted grades by the network, see figure 13. The numbers within each row and column can then represent the total number of essays that were classified as such by the human experts and the network accordingly. The diagonal of the matrix becomes the correctly labeled essays, and a well performing network is expected to classify most essays in or close to this diagonal. Using this visualization the performance of the MLP becomes clear, and most essays do indeed end up along the diagonal of the matrix, see figure 13. The matrix also reveals that for essay set 1 the network is better at classifying essays that belong to the highly populated grades of 7 through 10, and struggles with essays where the total amount of essays is low, such as grades 2 through 5.

Different Embeddings

Model		Essay set								Avg. QWK
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
GloVe	**Tr. Emb.**	0.65	0.56	0.58	0.68	0.78	0.74	0.75	0.47	**0.65**
	Fixed Emb.	0.61	0.49	0.62	0.67	0.77	0.73	0.66	0.44	0.63
No Glove	**Tr. Emb.**	0.62	0.46	0.59	0.61	0.77	0.70	0.71	0.38	0.61
	Fixed Emb.	0.62	0.46	0.58	0.61	0.76	0.63	0.68	0.33	0.58

Table 2: Quadratic Weighted Kappa performance of a Convolutional Neural Network model, while four different input embeddings are compared. The top two models used GloVe's pre-trained word embeddings for all words recognized by GloVe and randomized the rest, while the bottom two models randomized all word embeddings. The difference between fixed embeddings throughout the training of the network and trainable embeddings is also displayed in the table.

5.2 The Convolutional Neural Network

The Embeddings

A major component when training a network to automatically learn features from text is how the textual data is represented. For this, word embeddings with 100 dimensions were used, however their initialization and training varied. The basic approach to randomize all embeddings was tried, as well as using the pre-trained GloVe embeddings for all recognized words and only randomizing the rest. In both cases the options of either leaving the embedding fixed throughout the training of the network or to have it trained to better match the training data were also implemented.

As expected, using pre-trained GloVe embeddings did improve the overall performance of the network, see table 2. The CNN model was also improved by allowing the embedding layer to adapt to the training data, which is in general a more unconventional method. The danger of not using a fixed word embedding matrix throughout training is that the embeddings are very likely to over-fit to the training data, becoming representations of how the words are used in this particular text corpus rather than that of how the word is used in general. As seen in figure 14 however, over-fitting to the training data does not disrupt the validation results for this particular data set. A reason as to why the training of the embedding even improves the results may be that roughly half of all the words found did not exist in the pre-trained Glove embeddings and had to be randomly initialized, mainly because of the many spelling errors in the essays. It should be noted however, that while initializing the Glove word-embeddings took about 30 seconds and only had to be done once, training the embeddings rather than having them fixed did increase the training time for the model by roughly 50%.

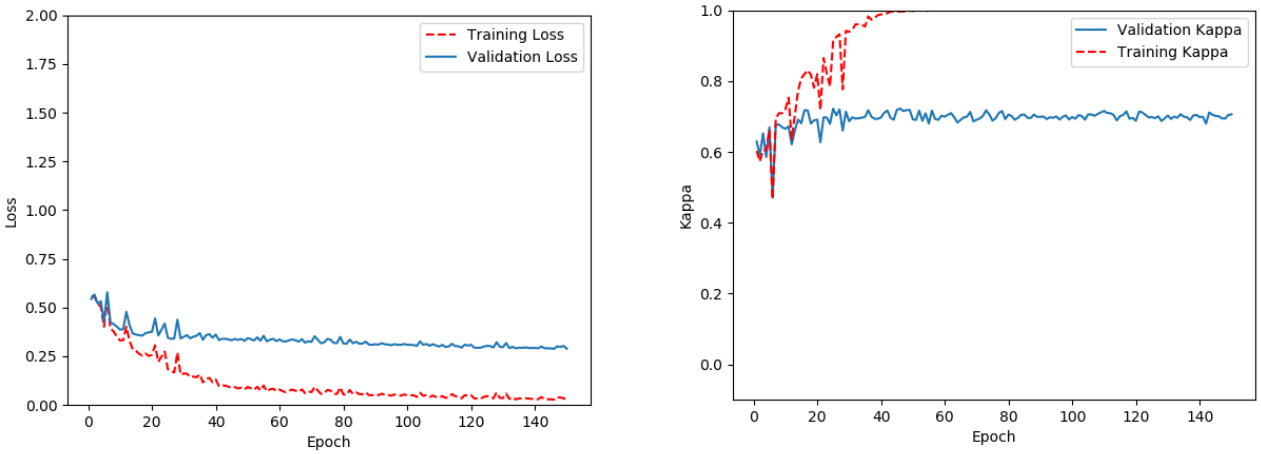

Figure 14: The training of the convolutional neural network. The left graphs displays the mean squared error loss during training, and the right graphs displays the quadratic weighted kappa score during the same period.

Regularization

Similar to the multilayer perceptron, the convolutional neural network showed to be insusceptible to over-fitting. Even though the linear output-model was able to learn all of the data points, pushing the mean squared error loss to 0 and the accuracy of the model to 100% on the training data, no consequences were seen in the validation loss or kappa, see figure 14. Over-fitting and learning all the essays in the training set rather seemed to improve the validation results, and this behavior was found for all 8 essay sets. Models using dropout and batch normalization were still tested, but this only slowed down the training. The L2 regularizer however was able to slightly improve the overall performance and was therefore implemented in the final version of the model.

Human Comparison

For three of the essay sets, 1, 7 and 8, the resolved score provided in the data set is constructed as the adjunction of the two human grades. This means that the resolved score range is doubled, as it contains twice the amount of possible scores for each essay. The task of classifying the essays then naturally becomes more difficult, and the number of correctly labeled essays should therefore diminish. When comparing a model trained on the score of a single expert with that of the resolved score the accuracy of the model did indeed improve from about 40% to 60% on essay set 1, however the quadratic weighted kappa score actually dropped from 0.75 to 0.58, see table 3. Similar large drops in kappa were also observed in the other two sets where the resolved score range differed from the human score range. To better visualize the difference between the model predictions and the human predictions their confusion matrices were studied, see figure 15. A clear difference in performance between the two can easily be observed.

Human Comparison

Model		Essay set								Avg. QWK
		1	2	3	4	5	6	**7**	**8**	
Human Agreement		0.73	0.80	0.76	0.85	0.75	0.78	0.73	0.60	0.750
CNN	**1 Human grader**	**0.70**	0.66	0.60	0.69	0.77	0.76	**0.65**	**0.35**	0.648
	Resolved Score	**0.80**	0.67	0.64	0.74	0.78	0.77	**0.76**	**0.45**	0.701

Table 3: Quadratic Mean Kappa score for each essay set when comparing models. The top row is the result when comparing the two human expert graders with each other. The second row is from a CNN trained on and predicting the score given by one of the graders, while the final row is from a CNN trained on and predicting the resolved score of both human graders. In set 1,7 and 8 the resolved score is the combined score of the two graders, resulting in twice as many possible classes for each essay.

It could be argued that the resolved score of two experts better represent the true deserved score of the essay, and that the deviations in the score from a single human grader could act as noise in the data, further complicating the problem. If so then the performance of the network is expected to drop for all essay sets when trained and evaluated only on the score of a single human expert. A small drop in kappa is observed in all of the sets, see table 3, however the large drop in kappa in essay set 1,7 and 8 can not be accounted for by this theory. To get a visualization of the difference between the two models their confusion matrices were studied, but no clear performance difference between the two was observed, see figure 16. It is difficult to argue that the predictions in the left confusion matrix is fundamentally better than the ones in the right by only observing the matrices. These results seem to suggest that the quadratic weighted kappa score is not enough to determine the effectiveness of the networks.

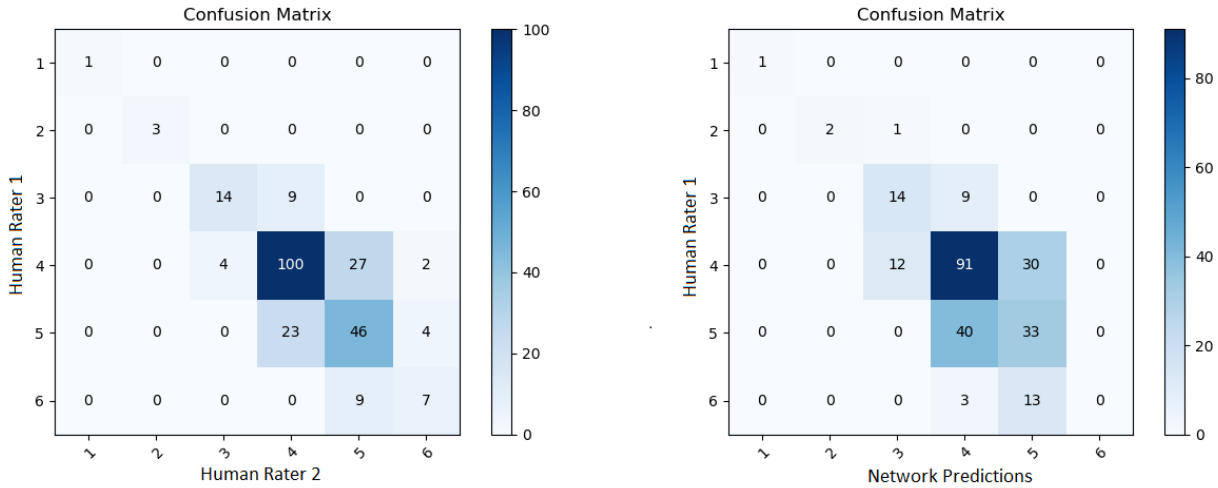

Figure 15: Visualization of the difference between classifiers. The left matrix displays the agreement between the two human graders with a kappa score of 0.74, and the right matrix displays a CNN predictions on the same essays, corresponding to a kappa score of 0.58.

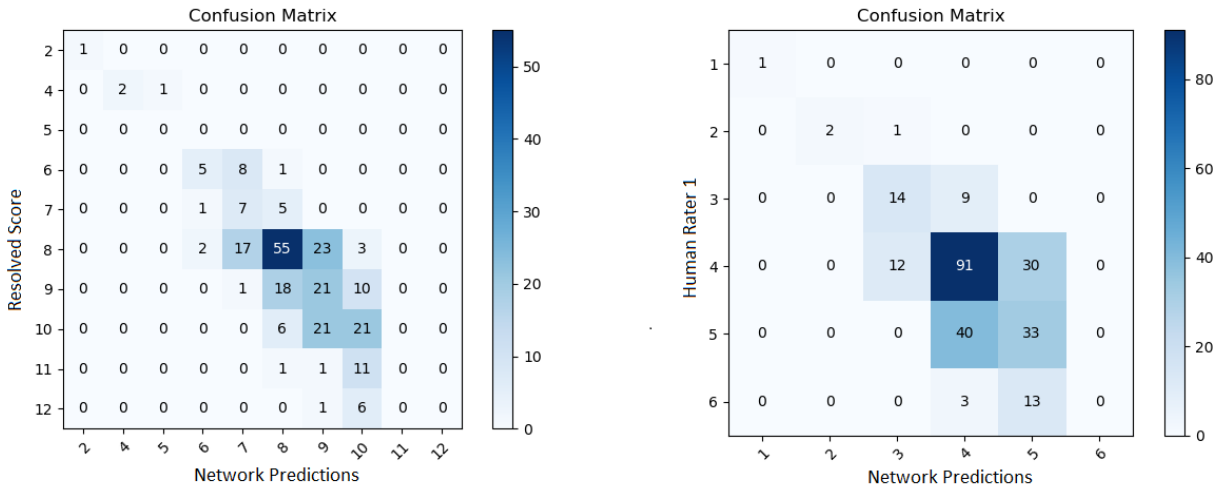

Figure 16: A visualization of the difference between classifiers. The same CNN model is trained to either predict the resolved score of the two human expert graders (left) or the score awarded by only one of the two humans (right). For the left matrix the amount of correctly labeled essays is 45% and the corresponding kappa score is 0.75. The right matrix results in a kappa of 0.58 and has a 57% accuracy.

Test and Validation Comparison

Model		Essay set								Avg. QWK
		1	2	3	4	5	6	7	8	
MLP Classification	**Validation**	0.73	0.63	0.65	0.66	0.77	0.60	0.66	0.49	0.648
	Test data	0.75	0.65	0.69	0.70	0.78	0.64	0.65	0.55	0.676
CNN Classification	**Validation**	0.58	0.56	0.62	0.67	0.77	0.72	0.70	0.37	0.624
	Test data	0.75	0.64	0.69	0.70	0.78	0.65	0.64	0.55	0.677

Table 4: Quadratic Weighted Kappa results comparing validation and test results. These models used a softmax output layer, treating AES as a classification problem. Similar results were found for the regression models.

Test and Validation Comparison

In order to see if the hyper-parameters tuned were too well fitted for the validation data compared to new data, a comparison between validation results and test results was made. It was found, however, that all models fared better on the test data, see table 4. This could mean that the test data happened to be an easier data set by chance, but a more probable reason is the increase in size of the training set used, as the complete training-validation data set could be used for training only. If this is indeed the reason then it hints at the importance of large data sets in AES, and that neural networks trained on the ASAP data set could potentially still be improved if the data set was expanded further.

Test Data

The best result of any model was a kappa of 0.701, reached by a convolutional neural network with a single node output using a linear activation function, see table 5. For comparison, the average human agreement kappa score was 0.750, and benchmark results by Dong et al. (2017) reached 0.764. Dong et al. used a combination of Long Short Term Memory (LSTM) units and a CNN and were able to surpass even the consistency of two human expert graders. This shows that there is still much more that could be done to the CNN model in order to improve the results beyond those of this paper.

While the regression-CNN model outperformed the handcrafted features of the MLP, the classification-CNN model did not. Using a much simpler method and very little information from each essay the MLP still reached a kappa score of 0.676, very similar to that of the classification-CNN model. It is possible that more handcrafted features could have raised this score further, but it is difficult to improve the structure of the MLP without turning it into something much more complex, defeating its purpose.

Final Test Results

Model		Essay set								Avg. QWK
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
Human Agreement		0.73	0.80	0.76	0.85	0.75	0.78	0.73	0.60	0.750
LSTM-CNN-attent		0.822	0.682	0.672	0.814	0.803	0.811	0.801	0.705	0.764
MLP	**Regression**	0.77	0.61	0.65	0.68	0.76	0.63	0.64	0.50	0.654
	Classification	0.75	0.65	**0.69**	0.70	**0.78**	0.64	0.65	**0.55**	0.676
CNN	**Regression**	**0.80**	**0.67**	0.64	**0.74**	**0.78**	**0.77**	**0.76**	0.45	**0.701**
	Classification	0.75	0.64	**0.69**	0.70	**0.78**	0.65	0.64	**0.55**	0.677

Table 5: Final Quadratic Weighted Kappa results when testing the main models on a previously completely unseen test data set. On top is the agreement between the two human expert graders, along with benchmark results found by using a combination of LSTM and CNN, (Dong et al., 2017). The top results found for each essay set are highlighted.

6 Discussion and Conclusion

6.1 The Multilayer Perceptron

It is difficult to measure the difference in time needed to implement handcrafted and learned features, as it depends a lot on previous knowledge and experience. In this research approximately two to three months of programming was spent on handcrafted features and the MLP and four to five months on learned features and the CNN. The CNN was also a lot more difficult to tune in order to find an optimal model, and the early results were much worse than those of the MLP. The training time needed for each model is also not exact, as neither of the models were implemented or tuned for optimal computation time, and the final models were trained using different computers. Overall it is however safe to say that the training time of the CNN was a lot longer, resulting in no K-fold cross validation being implemented. While the training time does not always scale linearly with the number of trainable parameters, the MLP used only about 5400 parameters and the CNN with a trainable embedding layer used 1.7 million parameters.

The four handcrafted features used was chosen based on the findings of Page (1968), but, unlike Page's work, in all these essay sets the only truly important factor of the four was the number of words in each essay, which managed to quite accurately represent the level of the text. It would seem reasonable to assume that it is foremost the shortest of essays that can accurately be graded based on their length, as they may be too short to even qualify for a higher mark. However, at least for the performance of the network this was not the case. Of the seven essays awarded the lowest resolved score (True label) the network classified none of them correctly, and most essays with a score of 5 or lower were also incorrectly classified, see figure 13. Instead it was clear that the network thrives when distinguishing between essays with a resolved score of 8, 9 or 10, which is where most of the essays are at. This is a hint at that the most important factor for Automated Essay Scoring

is the amount of available data to train on, and this hypothesis is further strengthened when analyzing the difference between the test data and the validation data, see table 4. Even though all hyper-parameters are tuned to maximize the results of the validation data, the overall performance of the model still improves when tested on the previously unseen test data, probably mostly because the size of the training data ha increased.

It is also interesting that even though the input was as small as a vector of four values, a very large network was needed to accurately predict the grades, see figure 11. Even more surprising is the lack of need for regularization of the network, as there seem to be little to no difference between the training values and the validation values, see figure 12.

Finally, grading essays is not a classification problem where all the classes are equally different from each other, but rather arranged on an incremental scale. In theory, a linear output-model would be able to take advantage of this where as a softmax output-model would not. It is therefore expected for a linear output-model to perform equally well or better than its softmax counterpart, and this is indeed seen when comparing the two for the CNN-models. However, for the MLP no such advantage was found, instead for the final results on the validation data it was the softmax-model that outperformed the linear-model. This may hint at that the data set is in total too small to draw major conclusions based on the small discrepancies between the models, but it also shows that the MLP models using handcrafted features are not obsolete when compared to a standard CNN using word embeddings. A big difference is of course that the MLP along with handcrafted features may be close to its peak performance while the CNN could still be remodelled and its complexity increased to improve it further, as seen by the benchmark results today, see figure 5. However, when considering the time spent on implementing and training the two approaches then the MLP is definitely a viable choice for small tasks.

6.2 The Convolutional Neural Network

At least for the ASAP data set, over-fitting to the training data does not seem to pose an issue for AES. Instead, the overall performance of the network seemed to only increase by pushing the accuracy of the network to 100% on the training data. This may be because of the small sample of roughly a thousand training essays spread over a huge representative vector space, as each essay was described as a matrix of an approximate size of 100x1000. An increase in size of the training set could also improve the networks further, as suggested when comparing the validation results with the test results, see table 4.

Almost half of the words used in the essays were not recognized by the GloVe word embeddings, and their embeddings had to be randomized instead. However, using GloVe still improved the results of the network, demonstrating the effectiveness of pre-trained embeddings. A possible further improvement is to pre-train the embeddings using only the essays as corpus, applying either the Word2Vec or GloVe technique. Another possibility is to apply some sort of spelling correction on the essays, as many of the words not recognized by the GloVe embeddings were just incorrect spellings of words that GloVe did in fact include. This would potentially have made the task of understanding the essays easier for the CNN, but it also would had changed to input and the essay. This might be viable, but

it might also require a count of misspelled and corrected words as another input somewhere in the network, further complicating its structure.

When comparing the models with the human experts it was seen that while the kappa score of the models did not reach that of the experts, there are issues with using the quadratic weighted kappa score as the only metric of performance in AES. When using the same CNN model to train on and predict either the score given by one human expert grader or the resolved score of both graders a large difference in kappa is found on the essay sets where the score range differs in the two situations. A large drop in kappa score is seen when using the smaller score range of a single human expert, even though the accuracy of the predictions improve. By observing the confusion matrices of the predictions it is also not clear which of the two networks does better. There may be a slight difference in the difficulties of predicting either the score of one expert or the resolved score of both, but using the same neural network model should not result in such a large difference as seen for essay set 1,7 and 8, see table 3. This argument is further strengthened by the very small discrepancies when comparing the other 5 essay sets. The conclusion is that the quadratic weighted kappa score is heavily influenced by the number of classes for the grades, and that the consistency of the kappa score may not scale well with respect to the number of classes.

Finally, the best model found was as expected a convolutional neural network with a single output node using a linear activation function. The standard CNN seems to be able to learn what features to look for in the essays without any prior instructions. This is one of the main advantages compared to the handcrafted features, as deciding, tuning and extracting features can be quite time-consuming. The use of word embeddings also better mimics how humans complete the task, reading the whole text rather than looking at statistics about it. It potentially makes the AES-system more robust and difficult to cheat, as it is not limited in its factors of how to view the text. These are fundamental flaws for handcrafted features, and so the limits of the approach are already set. That being said, the MLP did reach kappa scores only a little short of that of the standard CNN, and it was both simpler to implement and much faster to train. Both of the models fell short of the average human agreement on the data set with a kappa score of 0.750, however with scores of 0.701 and 0.676 they are both potentially applicable when attempting to speed up the process of grading essays. The conclusion is therefore that handcrafted features are a viable option for low-stake tasks, but if an AES-system is supposed to surpass human levels in consistency then more complex networks are required, as well as larger data sets than those of around 1000 essays.

7 Further Research and Improvements

Word embeddings

As shown, the word embeddings used for representing the essays are important, and pre-trained embeddings does help the network. It could therefore be advantageous to pre-

train the embeddings on the essays using the GloVe or the Word2Vec technique. This would remove the need of using randomized word embeddings for uncommon or misspelled words. Furthermore, many misspelled words could be replaced by their intended version by implementing some sort of spell correction, further strengthening the use of pre-trained embeddings. The number of misspelled words could also be used as an extra input to the dense layers at the end of the CNN model.

Hybrid Model

The two models, the MLP and the CNN, could be merged into a hybrid version of the two. The last hidden layer in both models could be connected with a final hidden layer, constructing an output prediction based on both models. This would create a fused model using the features of both approaches to make a prediction, which could be better than either model on itself.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Alikaniotis, D., Yannakoudakis, H., and Rei, M. (2016). Automatic text scoring using neural networks. *arXiv preprint arXiv:1606.04289*.

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

Chollet, F. (2016). Using pre-trained word embeddings in a keras model. `https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html`.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Chung, G. K. and O'Neil Jr, H. F. (1997). Methodological approaches to online scoring of essays.

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

Dong, F. and Zhang, Y. (2016). Automatic features for essay scoring–an empirical study. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1072–1077.

Dong, F., Zhang, Y., and Yang, J. (2017). Attention-based recurrent convolutional neural network for automatic essay scoring. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 153–162.

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

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

Hamner, B. (2012). Quadratic weighted kappa. `https://github.com/benhamner/Metrics/blob/master/Python/ml_metrics/quadratic_weighted_kappa.py`.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

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

Loper, E. and Bird, S. (2002). Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*.

Lärarförbundet (2018). Sverige behöver fler lärare. `https://www.lararforbundet.se/artiklar/lararbrist-sverige-behover-fler-larare`. Accessed on: April 27, 2019.

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

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

Page, E. B. (1968). The use of the computer in analyzing student essays. *International review of education*, 14(2):210–225.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Scikit-Learn (2011). Confsuion matrix. `https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html`.

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