UPPSALA
UNIVERSITET

Automatic Essay Scoring of Swedish Essays using Neural Networks

By Mathias Lilja

Department of Statistics
Uppsala University

Supervisor: Patrik Andersson

2018

Contents

Abstract

We propose a neural network-based system for automatically grading essays written in Swedish. Previous system either relies on laboriously crafted features extracted by human experts or are limited to essays written in English. By using different variations of Long Short-Term Memory (LSTM) networks, our system automatically learns the relation between Swedish high-school essays and their assigned score. Using all of the intermediate states from the LSTM network proved to be crucial in order to understand the essays. Furthermore, we evaluate different ways of representing words as dense vectors which ultimately have a substantial effect on the overall performance. We compare our results to the ones achieved by the first and previously only automatic essay scoring system designed for the Swedish language. Although no state-of-the-art performance is reached, indication of the potential from a neural based grading system is found.

1 Introduction

Grading student essays is an important yet potentially hard and slow process. Traditionally, the grading is carried out by a human rater which is more often than not also working as a professional teacher. Every hour spent on grading is time perhaps better spent on teaching and interacting with students. The rater is also vulnerable to subjective bias, generated by factors such as economic incentives, personal opinion and general state of mind. Swedish national exams have traditionally been graded by teachers employed at the same school as the students taking the exam. Furthermore, the exams are not anonymous, exposing the human rater to even more sources of bias. The grade from the national exam may not necessarily be used as the final course grade, but it could certainly have an substantial influence on it. Fortunately, recent decisions made by the Swedish government together with the Swedish National Agency for Education have initiated an effort towards the digitization of the national exams (Skolverket 2018). Along with it, there is also an intention of making the exams anonymous as well as switching to external raters.

Automated Essay Scoring (AES) refers to the statistical and *natural language processing* (NLP) techniques used to grade student essays without the direct involvement of humans. Most existing AES system are constructed and used on English texts. To our knowledge, only one previous study has been published where an AES system is implemented on Swedish essays (Östling et al. 2013). This study uses a linear classifier with manual feature engineering. Since then, recent advancements within neural networks have allowed for the creation of new state-of-the-art AES systems, pushing the precision of the AES closer to perfection.

All things considered, there is lot of potential gain from using AES for the grading of the Swedish national exams, especially at a time when the Swedish school system is being modernized. Therefore, the intention of this study is to make use of the latest findings within the AES research area and create as well as evaluate, a neural network based AES system on high school essays written in Swedish. Hopefully, this will reveal some of the advantages as well as difficulties to the potential implementation of an AES system into the Swedish school system.

2 Background

2.1 Automated Essay Scoring

Automated Essay Scoring (AES) is defined as *the computer technology that evaluate and score written prose* (Shermis and Barrera 2002). AES systems are mainly developed to assist teachers in low-stakes classroom assessment and authorities as well as testing companies in high-stakes assessments, such as the Test of English as a Foreign Language (TOEFL iBT) and Graduate Record Examination (GRE) tests. The main purpose of AES is to help overcome time, cost

and reliability as well as generalizability issues present in writing assessments (Dikli 2006). Most AES systems have implicitly or explicitly treated AES as a supervised text classification problem, utilizing a number of techniques within the field of natural language processing and machine learning.

Usually, an AES system works as follows: A collection of student texts in their plain text form (also known as a *text corpus*) are inputted into the AES software. The first stage of the AES is to preprocess the texts, that is making the texts readable by the computer and useful for further analysis. Basic examples of this includes stripping the texts of whitespace and removing certain characters such as punctuation, both of which are examples of *tokenization*, which is the task of splitting text sequences into pieces, referred to as *tokens*. Other common methods employed in the preprocessing stage are *normalization*, *stemming* and *lemmatization* (Schütze, Manning, and Raghavan 2008). The second stage typically involves *feature extraction* which is the process that maps the text sequences to a vector of measurable quantities, the most common example is the occurrences (frequency) of each unique word in the texts (Goldberg 2017). In most existing AES models, the feature extraction is a manual process performed by the researcher. It is considered as the most difficult part of the construction of an AES system and it is a real challenge for humans to take into account all the factors affecting the grade. Furthermore, the effectiveness of the AES system is constrained by the manually chosen features (Chen and He 2013). Lastly, a machine learning algorithm is implemented, using the predefined features as input. The type of models used in this last step is usually divided into three categories: regression, classification and preference ranking (Dong, Zhang, and Yang 2017). In the regression approach, often applied when the score range of the essays are wide, each score is seen as continuous values and regression models such as linear regression and Bayesian linear ridge regression are used (Attali and Burstein 2004; Phandi, Chai, and Ng 2015). In the classification approach, each score is instead seen as a separate class and common classification techniques such as *support vector machine* (SVM) and *naive bayes* (NB) are applied (Östling et al. 2013; Larkey 1998). Preference ranking has recently been proposed, where the scoring task is seen as a ranking problem (Yannakoudakis, Briscoe, and Medlock 2011).

Formally, an AES system is trained to minimize the deviation between the system-generated scores and human scores, given a set of training data. This can be formulated as:

$$\min_{\theta} \sum_{i=1}^{N} f(y_i^*, y_i), \quad s.t.\ y_i = g_\theta(t_i),\ i = 1, 2, ..., N, \tag{1}$$

where N is the number of texts used in the training set, y_i^* is the human score and y_i is the predicted AES system score of the i-th essay, respectively. The function f measures the difference between y_i^* and y_i, t_i is the feature representation of the i-th essay and g maps the feature t_i to score y_i (Dong, Zhang, and Yang 2017).

2.2 Existing Automatic Essay Scoring

The first AES system is commonly credited to the Project Essay Grader (PEG) developed by Ellis Page in 1966 (Page 1967; Page 1968). In similarity to most AES systems, PEG first uses a training sample to form its weights and then inputs new and unseen data in the testing stage. Its primary method is to use the correlation between different intrinsic variables such as fluency, grammar and punctuation. Criticism of the PEG have pointed out that the system mainly captures the surface structure of the texts and neglects the deeper semantic aspect (Chung and O'Neil Jr 1997). It was also possible to trick the PEG system by writing longer essays (Dikli 2006).

Another AES system which is also in use today, is the Intelligent Essay Assessor (IEA). This system is produced by the Pearson Knowledge Analysis Technologies. The IEA scoring system is implemented and used in the high-stake exam Pearson Test of English. Unlike other AES systems, the IEA is not used in combination with human raters but is instead the sole rater (Williamson, Xi, and Breyer 2012). In comparison to the PEG, the IEA uses a semantic text-analysis technique referred to as *latent semantic analysis* (Lemaire and Dessus 2001).

Perhaps the most known AES system is the electronic essay rater (e-rater; Burstein and Marcu 2000; Burstein 2003; Attali and Burstein 2004) which is developed and maintained by the Educational Testing Service (ETS). The scoring system of the e-rater uses regression in combination with predefined features representing different aspects of writing- and content quality (Williamson, Xi, and Breyer 2012). The number and type of extracted features have changed during the lifetime of the e-rater. However, ETS assures that these features are not only predictive of raters' score, but also have a meaningful correspondence to the features that raters are instructed to consider when they award scores (ETS 2018a). The e-rater system is used together with human raters to score the writing sections of the TOEFL iBT and GRE tests (ETS 2018b).

2.3 Evaluation of Automatic Essay Scoring

The accuracy of the automatic scoring system in relation to the human scores is commonly used as the primary performance metric of the AES system. In the literature, the similarity of scores from one rater to another (machine or human), is commonly referred to as the level of *interrater agreement* (Burstein et al. 1998). Interrater agreement has been utilized throughout the history of AES systems. It has been common practice to report the exact agreement (when scores from the human and the machine are exactly the same), as well as exact-plus-adjacent agreement (when the scores differ by not more than a certain threshold e.g. one point). However, using the level of agreement to evaluate an AES system can result in some issues regarding the reliability of the system. As an example, the scoring scale being used can have a large impact on the percentage of agreement since a coarser scale would imply a higher probability of an agreement by pure chance. Thus, it would overestimate

the performance of systems implemented on such coarser scoring scale tasks. Therefore, level of exact or exact-adjacent agreement are often limited as a convenience measurement for laypersons (Williamson, Xi, and Breyer 2012).

There are several alternative measurements for the level of interrater agreement, including Pearson's correlation, Spearman's rank correlation, Kendall's Tau and kappa, in particular the quadratic-weighted kappa (QWK) introduced by Jacob Cohen in 1968 (Cohen 1960; Cohen 1968; Dong, Zhang, and Yang 2017). The QWK was the official criteria of evaluation for the Kaggle competition Automated Student Assessment Price (ASAP) and is also used as a benchmark metric by the ETS and in several recent papers within AES (Williamson, Xi, and Breyer 2012; Taghipour and Ng 2016; Alikaniotis, Yannakoudakis, and Rei 2016; Dong and Zhang 2016; Zhao et al. 2017). The QWK is computed as follows: First, a weight matrix W is defined as:

$$W_{i,j} = \frac{(i-j)^2}{(N-1)^2},\ i,j \in \{1,...,N\}, \tag{2}$$

where i and j are the reference score (assigned by the human rater) and the system score (assigned by an AES system), respectively, and N is the number of possible scores. An observed score matrix O is computed such that $O_{i,j}$ denotes the number of essays that receive a score i by the human rater and a score j by the AES system. An expected count matrix E is calculated as the outer product of the two (reference and system) scores. The matrix E is then normalized such that the sum of the elements is the same in both the observed O and the expected E matrix. Lastly, given the matrices O and E, the QWK score is calculated according to:

$$\kappa = 1 - \frac{\sum_{i,j} W_{i,j} O_{i,j}}{\sum_{i,j} W_{i,j} E_{i,j}}. \tag{3}$$

The upper limit of the QWK is +1.00, which would imply perfect interrater agreement. A QWK value of 0 would on the other hand suggest that the there is no agreement, the result is not better than pure randomness (Cohen 1968). ETS uses a QWK value of 0.70 as an established baseline for the assessment of the level of agreement between a human and an AES system. Anything less than this threshold is considered as a bad agreement and anything above it is considered as a good agreement.

2.4 Automatic Essay Scoring in Swedish

Although there is extensive research and existing AES systems tailored for the English language, little can be said about the use of AES system texts written in Swedish. Possibly the first (and perhaps the only) AES system to be trained and tested using Swedish essays is published by Östling et al. (2013). This work is to some extent based on the B.A thesis by Smolentzov (2013).

The work of Östling et al. (2013) uses 1702 student essays from the essay writing part of the Swedish high school national exams in Swedish. These

essays are written during two different test takes, fall 2005 and spring 2006. Each text is graded by both the student's teacher and an external blind grader. The collection and digitization were originally done by Hinnerich, Höglin, and Johannesson (2011).

There are four grades: IG (fail), G (pass), VG (pass with distinction) and MVG (excellent). These are used as the dependent variable in a supervised machine learning approach, utilizing a *linear discriminant analysis* (LDA) classifier. Each essay is represented as a set of features, which in part uses the correlation between the features and the grades as the input of the model. The selected features include, among other, the number of words in each text, average word length and spelling error as well as more linguistically complex features.

Instead of using the QWK as a measure of the interrater agreement (as done in Smolentzov (2013)), the authors use a linearly weighted kappa value in conjunction with exact agreement classification accuracy. The result varies depending on which type of grade that is used. Using the average of the grades from the student's teacher and the blind grader, the authors achieve the best result with a kappa of 0.399 (62.2% exact agreement). Only using the grades from the blind graders led to a kappa of 0.369 (57.6%), and 0.345 (53.6%) using the grades from the student's teacher.

In Smolentzov (2013), the author uses four types of linear classifiers with manual feature engineering. The method based on the linear discriminant analysis classifier achieves the best result, with an average QWK value of 0.475 and an average exact agreement accuracy of 57.3%.

2.5 A neural approach to Automatic Essay Scoring

There are several areas within natural language processing which have seen recent advancements by utilizing the power of deep learning and neural networks (Young et al. 2017). The area of AES is not an exception. Introduced by Alikaniotis, Yannakoudakis, and Rei (2016) as well as Taghipour and Ng (2016), these new models surpassed the previous baseline in terms of interrater agreement; thus, outperforming the linear classifiers. Moreover, neural AES system do not rely on manual feature engineering but instead only uses numerical vectors which have been automatically converted from the text data. This means that one of the tougher parts of building an AES system can be skipped.

Alikaniotis, Yannakoudakis, and Rei (2016) treats each essay as a sequence of words and utilizes the power of *recurrent neural networks* on the Kaggle ASAP data. This dataset consists of 12.976 essays marked by two raters with a QWK for the interrater agreement of 0.86. Splitting the data into two parts with 80% for training and 20% for validation, the authors achieve a QWK of 0.96. The authors also train an SVM model using manual feature extraction for the purposes of comparison. This model achieved a QWK of 0.75.

Taghipour and Ng (2016) explores a wide variety of neural network models, including *convolutional-* and recurrent neural networks. They experiment with using these layers as standalone features, as well as combing them. Running a 5-fold *cross-validation* on the ASAP dataset, the best model achieves a QWK

score of 0.761. This outperforms the best open-source system that participated in ASAP competition by 5.6% in terms of QWK. This baseline system used bayesian linear ridge regression. Using the same data and training-validation approach as Alikaniotis, Yannakoudakis, and Rei 2016, the authors claim that their model achieves a QWK score of 0.987.

Dong and Zhang (2016) uses a hierarchical two-layer convolutional neural network. In this model, one of the convolutional layer is used to extract sentence representations, and the other is stacked above these representations in order to learn essay representations. In similarity to the above, Dong and Zhang also uses the ASAP dataset for training and validation. The hierarchical model achieved an average QWK of 0.734, which was slightly better than their feature-engineered linear model baseline of 0.725.

3 Method

The purpose of this study is to grade Swedish essays by constructing an automatic essay scoring model. This system will be based on a neural network architecture, which will enable the AES to almost autonomously generate a grade for each essay. We define our problem to be: *given a dataset of n student essays* $\boldsymbol{x_{1:n}}$ *and their respective grade* $\boldsymbol{y_{1:n}}$*, the target is to create a function* $f(\boldsymbol{x})$ *that correctly maps essays* $\boldsymbol{x}$ *to the AES outputs* $\hat{\boldsymbol{y}}$*, in terms of the similarity between* $\hat{\boldsymbol{y}}$ *and* $\boldsymbol{y}$. Ultimately, the AES system should be able to learn from graded essays in order to generalize this results and score ungraded essays. The system will be developed by using a training- and testing stage. During training, the AES model will be optimized to increase its performance which then will be evaluated during the testing stage.

In order to construct this AES system, several components are needed: First of, a translation component is required, which take each essay and transforms it into machine-readable information. Since our AES will be dealing with limited and relative complex data it need to do this in an effective way. Secondly, the AES system needs to somehow understand each essay in order to avoid generating random guesses when grading. We want the AES to some extent mimic the behavior of when a human is reading an essay. That is, our AES must be able to identify useful features within each essay which it later can relate to a specific grade. This also relates to the last component of the AES system which concerns the task of optimizing the model. In the beginning the AES is certain to make mistakes. In order to learn from these mistakes, the AES needs to understand how large the mistake was and what it can do to improve. However, since the ultimate goal of the AES is to grade unseen essays, the AES must be able to generalize its results. This also means preventing the system from getting overly attached to the training data.

These components are highly intertwined within the AES system. Which is why it is hard to introduce them separately and in the above presented order. Instead, the dataset will be presented in Section 3.1 and a brief explanation of cross-validation in Section 3.2. This is followed by a general introduction

to the neural network framework as well as specific solutions implemented in our AES system in Section 3.3. Lastly, Section 3.4 will discuss different ways of transforming each essay into some quantitative information which can be processed by the AES system.

3.1 Data

Two samples from the national test in writing given in the Swedish upper secondary school is used as the dataset. The texts are administered as part of the courses *Svenska B* as well as *Svenska som andra språk B* and are formulated and written completely in Swedish.

The first sample (sample I) contains 1736 student essays collected by Hinnerich, Höglin, and Johannesson (2011) and are the same texts which was used in the creation of the first AES for Swedish introduced in Section 2.4. All texts in this sample are from the 1994 national curricula, in which there are between eight and ten topics to choose from per test (Östlund-Stjärnegårdh 2014). For each test there is a common theme and each topic can require a specific writing style within that theme, such as an argumentative essay, article or a chronicle. There is one test per semester i.e. two tests per year. Sample I contains texts from two different test takes: fall of 2005 and spring of 2006. There is in total 19 different topics in this sample. Moreover, sample I contains two grades for each text; one grade is the one originally assigned by the examining teacher, whereas the other grade is given by an independent blind rater. The interrater agreement between the teacher and the blind rater is found to have a rather low QWK of 0.38 and an exact-agreement accuracy of 45.1%.

The second sample (sample II) contains 1.495 student texts and is collected as part of the Uppsala Corpus of Student Writings (Megyesi, Näsman, and Palmér 2016). These essays are also based on the 1994 national curricula. Sample II is made up of texts from each semester for the years 1996 to 2013. This fact opens up for a lot of variability within this sample since there is a potential range of 288 — 360 unique topics in the dataset.

There are four possible grades for each text: *IG* (failed), *G* (passed), *VG* (pass with distinction) and *MVG* (excellent). These are mapped to integer values of $0, 1, 2, 3$, respectively. Each text is first preprocessed by tokenizing (splitting it up into separate words), transforming into lowercase and removing different stopwords as well as special symbols such as *!#"-?*.

The occurrence of each grades as well as the average and standard deviation are displayed in Table 1. The distinct number of tokens for the sample set and average number of words for each text is also found in Table 1. There are two grade sets available for sample I, meanwhile only one grade set for sample II. The average for all three grade sets lies between *G* and *VG*. Notably, the mean of both grades assigned by the teacher are both closer to each other as well as higher than the grade set assigned by the blind rater. There is around 62% more unique tokens present in sample II than there are in sample I; another indication that sample II contains more variation in terms of topics and word use.

	Frequency		
Grades	Blind rater I	Teacher I	Teacher II
IG (=0)	283	155	137
G (=1)	866	746	558
VG (=2)	458	620	548
MVG (=3)	129	215	252
Total	1736	1736	1495
Avg. score	1.249	1.516	1.612
(Std. dev)	(0.814)	(0.822)	(0.871)
Unique tokens	36 401	36 401	58 937
Avg. num of words	550	550	597
(Std. dev)	(257)	(257)	(242)

Table 1: Frequency table over grades. *Teacher I* and *Blind rater I* corresponds to the grades assigned by the teacher and blind rater in sample I, respectively. *Teacher II* corresponds to the teacher grades in sample II.

From Figure 1, it is evident that the grades are approximately normally distributed, especially for the grade sets created by the teachers. This also means that the data is quite unbalanced in its classes with for example 50% *G*s and only 7% *MVG*s for the blind rater.

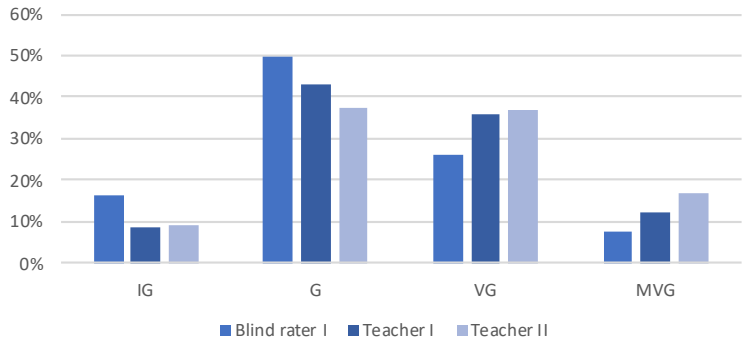

Figure 1: Bar plot over the relative frequency of grades for each grade set.

The AES model will first be trained and evaluated using the essays from sample I, more specifically using the blind rater's grades. This will allow for comparisons to be made between our AES system and the AES created by Östling et al. (2013). In addition, the performance of a neural network model is often directly related to the number of training samples used, especially when generalizing the results (Goldberg 2017). For this reason, the AES will also

train using essays from both samples. When this combined dataset is used, the network will use the blind rater's grades from sample I and the teacher's grades from sample II. Hopefully, combing these datasets will reveal if more data leads to a better result.

3.2 Cross-validation

It is evident from Section 3.1 that there is a lot of variability in the data. In addition, preliminary testing showed that the result is dependent on the specific training/test split. This is likely also a result of the relatively small sample size. For this reason, a cross-validation (CV) approach is used in order to train and validate the models. Furthermore, because of the relatively imbalanced data set, *stratification* (explained below) is also implemented in the CV method.

CV is a simple yet widely used method for assessing the performance of a classifier. The first stage of the CV method is to divide the whole dataset into K roughly equal-sized parts (also known as *folds*); for example, when $K = 3$ the scenario looks like Figure 2.

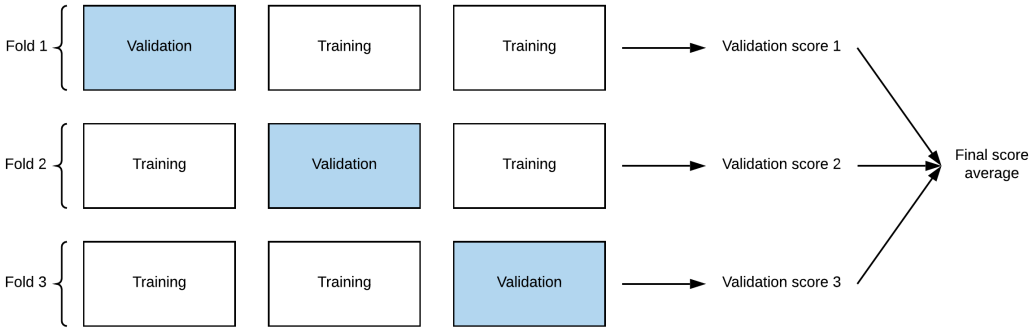

Figure 2: Three-fold cross validation.

Secondly, for each part k, train a model on the remaining $K - 1$ parts and evaluate it on part k. Repeat this process for every $k = 1, 2, ..., K$ part. Finally, average the validation score over all the K parts to create a final score of the model.

Adding stratification to the CV method means that the proportion of classes (grades) are kept approximately equal in each fold. This means that even though the grades G and VG are the most frequent, we do not risk training on a part only containing these grades. If this was not the case, it could potentially have a negative effect for the validation score to train on a dataset which is not adequately represented by each class.

The choice of K is often based upon the computational burden of running the model on each fold, as well as the *bias-variance trade-off* (see e.g. Friedman,

Hastie, and Tibshirani 2001). In this paper, we will employ a 5-fold CV method, mainly due to the large computational burden associated with training the AES system.

3.3 Neural Networks

3.3.1 Feedforward Network

Despite the mystery surrounding the name neural networks, it is just a special case of nonlinear statistical methods. This section introduces the most simple version of the neural network models, the *feedforward* network. Neural network models are typically illustrated using graphs as seen in Figure 3, this is a representation of a feedforward network. This model is represented by three layers, where the circles in each layer are the *neurons* of that layer. Each neuron can have incoming- and outgoing arrows, depending on the type of layer. Each arrow represents a weight, corresponding to the importance of that connection. The bottom layer in Figure 3 has no incoming arrows and is referred to as the input layer. Similarly, the top layer has no outgoing arrows and is the output layer of the network. The middle layer has both incoming- and outgoing arrows. This is the hidden layer of the network. In Figure 3, there is only one hidden layer. Furthermore, in Figure 3 every neuron is connected to the ones above, making it a *fully connected layer*. Each layer can vary in size (number of neurons), but the size of the input layer corresponds to the size of the input and the size of the output layer is determined by the target of the model. Our AES model should be able to classify each student essay into one out of four possible grades. In order to make this possible, the output layer of the AES system will have four neurons, where each neuron corresponds to a specific grade. Note that the arrows of the network in Figure 3 are all going upwards, i.e. there are no connections between the neurons of the same layer; hence the name feedforward.

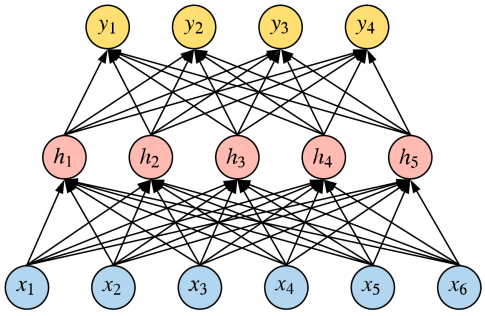

Figure 3: Graphical representation of a feedforward neural network with one hidden-layer.

The values of each row of neurons in the network can be thought of as a vector

with the same length as the number of neurons in that layer. For example, in Figure 3 the input layer is a 6-dimensional vector $\boldsymbol{x}$, the hidden layer above is a 5-dimensional vector $\boldsymbol{h}$. Thus, the fully connected layer can be thought of as a linear transformation from six- to five dimensions. The neurons in the hidden layer $\boldsymbol{h}$, are created from the linear combinations of the inputs $\boldsymbol{x}$, and the target $\boldsymbol{y}$ are then derived as a function of linear combinations of $\boldsymbol{h}$.

This section will give a relatively basic mathematical introduction to neural networks by defining it as vector-matrix operations. The simplest neural network, the so called *perceptron* is just a linear model:

$$NN_{Perceptron}(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}, \tag{4}$$
$$\boldsymbol{x} \in \mathbb{R}^{d_{in}},\ \boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}},\ \boldsymbol{b} \in \mathbb{R}^{d_{out}},$$

where $\boldsymbol{W}$ is the weight matrix and $\boldsymbol{b}$ is the bias term. In order to make the linear into nonlinear, a hidden layer is used (it could of course also be several hidden layers). This means that the model is now a multi-layer perceptron with one hidden-layer (MLP1), which also is the type of network illustrated in Figure 3. This feedforward network with one hidden-layer is defined as:

$$NN_{MLP1}(\boldsymbol{x}) = \boldsymbol{y} \tag{5}$$
$$\boldsymbol{y} = \boldsymbol{h^1}\boldsymbol{W^2} + \boldsymbol{b^2}, \tag{6}$$
$$\boldsymbol{h^1} = g(\boldsymbol{x}\boldsymbol{W^1} + \boldsymbol{b^1}). \tag{7}$$

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}},\ \boldsymbol{W^1} \in \mathbb{R}^{d_{in} \times d_1},\ \boldsymbol{b^1} \in \mathbb{R}^{1},\ \boldsymbol{W^2} \in \mathbb{R}^{d_1 \times d_2},\ \boldsymbol{b^2} \in \mathbb{R}^{d_2},$$

where $\boldsymbol{W^1}$ and $\boldsymbol{b^1}$ are the weight matrix and bias for the first linear transformation applied to the input. g is a nonlinear function which is applied element-wise, commonly known in the neural network community as an *activation function*. Finally, $\boldsymbol{W^2}$ and $\boldsymbol{b^2}$ are the weight matrix and bias for the second linear transformation. In combination with the other weights and biases, these make up the parameters of the model and the entire collection of these parameters will be referred to as Θ.

Our AES system will primarily rely on another type of neural network architecture referred to as recurrent neural network (this model is introduced in Section 3.3.5). However, the final output layer will be the same as the one introduced in this section. As previously mentioned, the goal is to map each essay into one out of four possible grade categories, which is partly done by having four neurons in the output layer. Furthermore, for each essay the AES system needs to create some probability distribution over the four grades where the highest class probability determines which grade that should be assigned to that specific essay. In order to do this, the output layer will use a so called *softmax* activation function.

The softmax is a generalization of the logistic function and creates a probability distribution over class assignment, where the output vector entries sums to one. The softmax is defined as:

$$softmax(\boldsymbol{x})_i = \frac{e^{\boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{x}_j}}. \tag{8}$$

Ultimately, implementing a four neuron output layer in combination with a softmax activation function means that the AES will take each essay as input, process the information and in the end generate an output vector of length four. Each element within this vector will be the probability that the essay should be assigned to the corresponding grade. The grade that corresponds to the maximum value in the output vector will then be chosen and assigned as the predicted grade.

3.3.2 Loss function

Recall, the objective of our AES system is to create a function that accurately maps a set of essays to a set of grades. That is, creating a function $f(\cdot)$ such that the predictions $\hat{\boldsymbol{y}} = f(\boldsymbol{x})$ based on the training set are accurate. The term accuracy is quantified by the use of a *loss function* L, which measures the loss suffered when comparing the predicted grades $\hat{\boldsymbol{y}}$ to the human assigned grades $\boldsymbol{y}$. Thus, in order to accurately map the inputs to the desired labels, the goal becomes to minimize this loss function $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$.

The training of the AES model then becomes a process of minimizing the loss L by updating the values of the model parameters Θ, this can be referred to as optimizing the network. The optimization process is introduced in Section 3.3.3. Also, minimizing the loss at all cost may not necessarily produce the best generalizations on new data. This is referred to as *overfitting* and will be discussed briefly in Section 3.3.4. The remainder of this section will introduce the specific type of loss function used to create our AES system.

Each student essay can belong to one out of four possible categories, which are defined as the different grades. As previously mentioned, in order for the AES to classify each input text into one of these grade classes a softmax function is used as the final output layer; generating an output vector of length four with a probability distribution over the class assignments. During training, the AES needs to measure how well it is doing in order to further tune the network i.e. it needs to calculate the loss. Our AES will do this over the essays $1, ..., n$ by comparing the generated grades $\hat{\boldsymbol{y}} = (\hat{\boldsymbol{y}}_1, ..., \hat{\boldsymbol{y}}_n)$ which was passed through the softmax layer, with the human grades $\boldsymbol{y} = (\boldsymbol{y}_1, ..., \boldsymbol{y}_n)$. This is referred to as calculating the *categorical cross-entropy* loss (Goldberg 2017), which is in this case is given by:

$$L_{cross-entropy}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_i \boldsymbol{y}_i \log(\hat{\boldsymbol{y}}_i). \tag{9}$$

It follows that the goal of the AES then becomes to minimize this cross-entropy loss. From Equation 9, it is evident that the cross-entropy is in fact a negative log likelihood; making the goal of minimizing this equal to maximizing the (log)likelihood of this model.

3.3.3 Neural network training

In machine learning, the way to minimize the loss function $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ can often be solved by implementing some gradient-based method such as *gradient descent*. To put it simply, this method functions by iteratively computing an estimate of the loss over the training data, calculating the gradients of the collection of parameters Θ with respect to the loss estimate, and moving the parameters in the opposite directions of the gradient (Goldberg 2017). However, the major difference between more traditional linear models such as SVM and neural networks, is that the nonlinearity of the neural network makes most loss-functions *non-convex*. In contrast to convex optimization, non-convex loss functions no longer implies a guaranteed convergence to the global optimum and it is also sensitive to the initial values of the parameters (Goodfellow et al. 2016).

A persistent problem in machine learning is that large amount of training data is often necessary in order to achieve a good generalization of the results. However, the downside of using a larger training set is the added computational time required for training. Using the regular gradient descent method which calculates the gradient based on the entire training set, will in combination with a moderately sized training data, render the required computational time far too long. Instead, an effective method for optimizing non-convex loss functions and thereby training the model is the *stochastic gradient descent* (SGD) algorithm, and other methods based on it (see e.g. Goodfellow et al. (2016) for more information on SGD).

The SGD works very similar to what was described above for the standard gradient descent method. However, the insight added with this approach is to consider the gradient as an expectation. Thus, making it possible to approximately estimate this expectation using a small set of samples. Specifically, on each step of the algorithm, a so called *minibatch* of m examples $(\boldsymbol{x_1}, \boldsymbol{y_1}), ..., (\boldsymbol{x_m}, \boldsymbol{y_m})$ are sampled uniformly from the training set. Typically, the size of the minibatch is a relatively small number, often ranging between 1 to a few hundred (Goodfellow et al. 2016). Once all the minibatches of the training data has been fed to the network, one so called *epoch* is completed. At that point and if specified, the model will start over with a new training epoch where the training data is reshuffled into new minibatches. The number of epochs is a hyperparameter and needs to be set by the researcher. It is difficult to know how many epochs to train the model beforehand, although some important factors are the size of the training set and the complexity of the problem (Goodfellow et al. 2016). For the SGD and similar methods, there is also the notion of *learning rate*. This is defined as a positive scalar determining the size of the update to the parameters Θ.

In order to calculate the gradient of a neural network, an algorithm referred to as *backpropagation* is used. In its most fundamental state, this method repeatedly applies the chain rule of derivatives through all the possible paths of the network, starting with the final layer of weights and progressing backwards layer by layer. However, there is a lot more to it than this. In contrast to the ineffective way of calculating the gradient for each layer separately, the back-

propagation reuses partial computations from one layer in the calculation of the gradient of the next layer. For an in-depth explanation of backpropagation see e.g. Goodfellow et al. (2016).

Finally, newer and more advanced versions of the SGD algorithm can be used and could potentially both speed up the computation time as well as produce a better result. One popular example of such an algorithm is the *adam* (Kingma and Ba 2014) optimizer. This is a so called adaptive learning rate algorithm, which is setting the learning rate for each minibatch separately. This study will implement the adam optimizer into the AES system.

In order to decide the most appropriate number of epochs, minibatch size and initial learning rate, a *grid search* approach will be implemented in the construction of the AES model. This basically means that the AES is trained on all possible combinations of a pre-defined set of hyperparameter values. This is also combined with a cross-validation approach (which is explained in Section 2). We use insights from previous research as well as knowledge about the specific characteristics of our datasets as the basis when deciding the hyperparameter grid. The best combination of hyperparameters will then be implemented in the final training and testing stage of the AES system.

3.3.4 Overfitting

The primary strength of multi-layer neural networks with many parameters is the ability to learn from the training data. For this reason, neural networks can be applied to numerous different tasks such as image classification, sentiment classification and in our case AES systems. On the other hand, a large model with many parameters may learn the training data well, but it could potentially fail to make generalizations of the result. This is what is known as overfitting and in simple terms happens when the model gets too attached to the training data. In fact, overfitting is one of the major issues with neural networks and a lot of research is being invested into this research area.

Perhaps the biggest concern for the success of our AES system is the limited number of training examples (in comparison to other neural network models). This will probably mean that the AES model will have to struggle with a substantial level of overfitting. This can also happen since a small set of training examples could restrict the AES from properly learning what features that are important when assigning grades. Instead, it could end up with just mimicking the essays used during training and as a result have little or no indications about what grades to assign to unseen essays.

To remedy the problem of overfitting, we employ a method referred to as *dropout*. Dropout (Hinton et al. 2012; Srivastava et al. 2014) is a relatively new and intuitive technique for countering overfitting on the training data in neural networks. It works by randomly dropping (setting to 0) some of the weights in the network or in a specific layer. Thus, forcing the network to not rely too much on certain weights. It is effectively a binary mask drawn from a *Bernoulli* distribution being applied to each minibatch of examples. The mask is independent for each unit and the probability of sampling a mask

value of 1 (thereby including the unit) is determined before the training process (Goodfellow et al. 2016). For the AES system, the probability of being included will be set to 0.5. This is a common value in many neural network models and have proven to work in previous studies regarding AES systems (Dong, Zhang, and Yang 2017).

3.3.5 Recurrent Neural Network

Processing textual data such as student texts with a regular feedforward network actually means restricting the analysis to a single data point. That is, each unique text is being transformed into a single large vector. This is turn means that although we are looking at word frequency, we are neglecting the ordering of words. In the neural network community, such a mechanism of the feedforward network can be characterized as having no memory. That is, each input is processed independently and with no state kept between inputs (Chollet 2017).

Remember, we want the AES system to understand each essay, preferably by realizing what the human grader is doing when grading an essay. If the AES system could somehow learn important features in the texts, it could increase the accuracy of the classifier as well as reduce the amount of overfitting. However, this will certainly not be achieved by limiting the analysis to the word frequency, which is done when implementing a feedforward neural network. Instead, we want the AES system to some extent mimic the behavior of when a human is reading and grading an essay. Thus, in order for the AES system to perform adequately, we need it to start consider the ordering of words as well as have the ability to remember what it already have read.

Fortunately, there is type of neural network architecture that is suitable for this task. A recurrent neural network (RNN) is an adaptation to the neural network toolbox which behaves somewhat like a human when processing text. It does this for each essay by iterating through each word whilst also considering information relative to what it already has seen. In practice, we are still considering each text as a single data point, however it is no longer processed in a single step. Instead it implements an internal loop, feeding the output of the previous element into the next. Thus, an RNN can be viewed as multiple feedforward networks, passing previous information forward (Chollet 2017). Figure 4 is a graphical representation of a so called unrolled and fixed-length RNN, where each step (so called timestep) is visible.

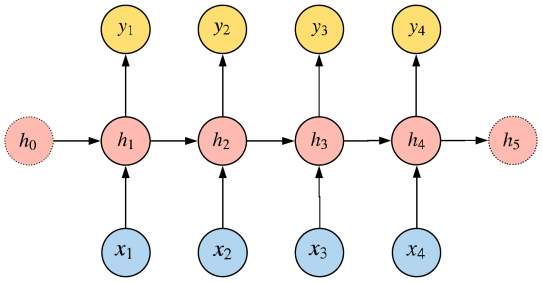

Figure 4: Graphical representation of a simple recurrent neural network (unrolled).

The RNN in Figure 4, is shown to input four words: where $x_1, ..., x_4$ represents each input word, $y_1, ..., y_4$ represent the predicted next word and $h_0, ..., h_5$ are the state of the network, holding information for the previous input words. Note that each value of h_i (and thus y_i) is based on the entire input $x_1, ..., x_i$, which also means that the last output of the network (y_4 in Figure 4) contains information of all input words. When implementing the RNN into the AES system, we can determine if the network should return the full sequences of successive outputs for each timestep or limit it to return only the last output for each text. In the case of the latter, Figure 4 would only show output y_4.

In similarity to the feedforward network presented in Section 3.3.1, RNN also passes the information through a nonlinear activation function, commonly a *tanh* or a $ReLU$ function (see e.g. Goodfellow et al. (2016) for details regarding these functions). The first hidden state h_0, is typically set to all zeros and referred to as the *initial state*. Finally, it is important to note that RNN does not function on its own, but should instead be used as a component of a larger network (Goldberg 2017).

As previously mentioned, in theory every value of h_i is based on the information of what happened in previous timesteps. However, in practice the simple RNN is having a difficulty capturing information from timesteps farther away. Our AES model will handle relatively long essays (we will set the limit to 600 words for computational purposes), which will result in such difficulties connecting words too far away. This problem is widely known in the neural network literature and is referred to as the *vanishing gradient problem*. This is a common occurrence in deep neural network models, such as RNN where the sentence length equals the depth of the network (recall that the RNN could be viewed as multiple feedforward layers). Basically, this problem occurs since gradients propagated over many timesteps tend to vanish (get smaller) and smaller gradients makes it harder for the model to learn properly (Goodfellow et al. 2016). This problem can occur even for a relatively small sequence length of 10 to 20 words which makes it a real problem for our AES with a sequence length of 600 words.

Fortunately, there are other types of RNN models more suitable to deal with vanishing gradients, such as the so called *long short-term memory* (LSTM) network which was first introduced by Hochreiter and Schmidhuber (1997). The LSTM is a special case of the simple RNN and still uses current input and previous state to compute the new state. However, the new state is computed in a more complex way using a *gated* architecture. For an illustration how this is done see Figure 5, which illustrates the steps taken to create a new state using the old state and the current input.

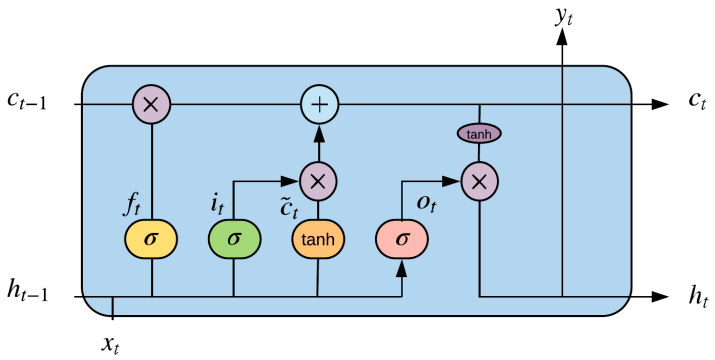

Figure 5: Graphical representation of the inner architecture of a LSTM. Symbol $\times$ represents the element-wise multiplication and σ represents the sigmoid function.

In contrast to the simple RNN, the LSTM splits the state vector into two components, the memory component c and the state component h. Previous state h_{t-1} and current input x_t is passed through a series of gates referred to as the input- (i_t), forget- (f_t) and output (o_t) gates. The forget gate is used to decide how much of the previous memory cell c_{t-1} that can be used in the computation of the next memory cell c_t by looking at current input x_t and the previous state h_{t-1}. The input gate determines how much of x_t and h_{t-1} that should be preserved. An update candidate $\tilde{c}_t$ is produced by passing x_t and h_{t-1} through a tanh activation function. The memory new cell c_t is produced by using the results stored in f_t, i_t and $\tilde{c}_t$. Finally, the new state h_t (which is also equal to the output y_t) is produced by c_t and o_t, where the latter is created by x_t and $h_t - 1$.

Implementing an LSTM layer into the AES will hopefully produce better predictions by utilizing the sequential information embedded within each essay as well as mitigate the problem of a vanishing gradient.

Lastly, when a human is reading a text he or she is constantly changing the interpretation of previous words as he or she are progressing forward through the text. This can sometimes be crucial for the interpretation of a specific word or a sentence. On the other hand, when a regular RNN model is processing text

it does not change the interpretation of previous words given a new word. Since our goal is to make the AES system capable of understanding the essays, much like a human would, we will experiment with adding another time direction to the RNN model. That is, the RNN network will process each essay both in a forward manner starting from the first word of the text and ending with the last as well as in a backwards manner starting with the last word and ending with the first. This is referred to as a *bidirectional* RNN (as opposed to a unidirectional RNN).

To summarize, in order for the AES system to accurately map the student essays to a set of predicted grades, an LSTM layer will be used. This will act as the component task at understanding the information embedded within each essay. Hopefully, by processing the text in a sequential manner, the AES can identify important features which it can relate to the human assigned grades. Consequently, generating accurate predictions with a high interrater agreement to the human grades. We will also experiment with restricting the output to the last output only as well as returning all the outputs. Both uni- and bidirectional LSTM will also be evaluated.

3.3.6 Pooling

In order to make full use of the power of the LSTM layer, a *pooling* layer will be experimented with. Pooling layers are perhaps most commonly used in convolutional neural networks, typically within the area of image recognition (Goodfellow et al. 2016). In this case a *mean-over-time* (MoT) pooling layer will be used as it have shown some success in previous work within the NLP area (Taghipour and Ng 2016). MoT works by taking the output vectors of the LSTM layer, $\boldsymbol{h} = (\boldsymbol{h_1}, \boldsymbol{h_2}, ...\boldsymbol{h_M})$ and calculate an average vector of the same length. This is defined as:

$$MoT(\boldsymbol{h}) = \frac{1}{M}\sum_{t=1}^{M}\boldsymbol{h}_t \tag{10}$$

Once the vectors have been averaged in the MoT layer, they will be fed into the softmax layer and mapped to one of the four possible grade classes. An alternative to this approach would be to skip the MoT layer and just feed the last output of the LSTM layer $\boldsymbol{h_M}$ into the softmax classifier. By doing this, there is a risk of losing valuable information by not utilizing the intermediate states of the LSTM layer. We will implement both methods and thereby find out which one leads to a better result for the AES system.

3.4 Working with textual data

Up until now, the AES system consists of two major components: one which is tasked with understanding each essay and another one which will quantify the performance and optimize the network. The former consists of the LSTM layer, MoT layer and the output layer with the softmax activation function. The latter

is composed of all the things that enables training of the AES system: the cross entropy loss function, stochastic gradient decent, minibatches, epochs and the dropout function. However, to make the AES system capable of processing the student essays, another component is needed to quantify the text data. This final component will be discussed in the two upcoming sections.

3.4.1 One-hot word vectors and word embeddings

When working within the context of natural language processing, one major concern is how to transform textual features, such as words and characters, into readable input for the classifier. Historically, the two most popular ways of doing this is by using *one-hot vectors* or *embedding vectors* (Goldberg 2017). This section will briefly introduce the technique of turning text into one-hot vectors (so called one-hot encoding), mainly for the purpose of later introducing the more useful embedding vector. Throughout this section, textual data will only refer to the word-level representation, simply because we want the AES system to work on this level. Meaning, every unique word such as *apple* and *pear* will be the most basic and fundamental component of the textual data. Also, our AES system will be used on Swedish essays but the text examples will be provided in English. For the purpose of quantifying text data, no distinction has to be made between these two languages.

One-hot encoding is essentially a way of incorporating the use of indicator functions as a representation of textual data. For example, consider a situation where the number of unique words in our data is equal to 10, this is referred to as having a *vocabulary* of size 10. One of the possible words in the vocabulary is *apple*. In the context of this specific vocabulary, the word *apple* will be represented by a sparse binary vector of length 10 where the entry corresponding to the vocabulary word *apple* is non-zero, and all other entries are zero. Similarly, consider two other words included in the vocabulary, *pear* and *car*. These are also each represented by a separate vector of length 10, with a non-zero entry corresponding to the location of each word in the vocabulary (see Figure 6). Note that in Figure 6, the word *apple* is represented by a 1 in the first column and 0 in all the rest, i.e. the first column corresponds to the word *apple*. In another example, *apple* could have a non-zero entry in column 5 and zeros elsewhere. In addition, if there are more than one of the same unique word in a corpus, the value of the non-zero entry in the vocabulary is equal to the number of occurrences of that particular word. If for example there are two instances of the word *apple* appearing in the data, then the value of the corresponding non-zero entry would be equal to 2.

The way to index columns is determined by the researcher and could for example correspond to the order of encounter or the most frequent words in descending order. The location of the word within the vector is often of no great importance, but using a consistent ordering throughout the entire corpus is.

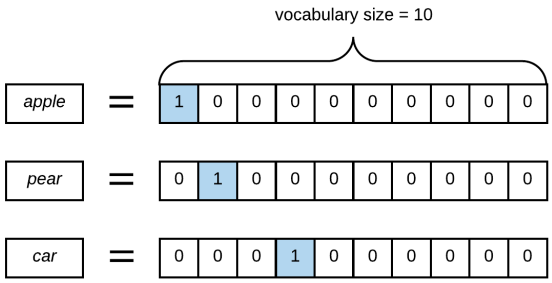

Figure 6: Example of one-hot vectors for the words *apple*, *pear* and *car* in a 10 word vocabulary.

The sparsity of the one-hot vectors grows with the size of the vocabulary. For example, if the vocabulary consists of $10,000$ words then a one-hot vector of one word would be a vector of length $10,000$ with only one non-zero entry. There can be several drawbacks with using one-hot encoding to represent textual data (Goldberg 2017). The high-dimensionality of the data is one issue, which can lead to high computational time since a majority of the neural network toolkit is not optimized for high-dimensional, sparse vectors. However, this may be just a technical issue. The major drawback of using one-hot vectors is that the generalization power which originates from word context and similarity is lost. When using one-hot vectors to represent textual data, all the words become independent of each other. The word *apple* is as dissimilar to the word *pear*, as it is to the word *car*. This can be seen by taking the dot product of any of the above one-hot vectors, it will always be zero.

The use of one-hot vector is common for machine learning problems which use a linear model with a lot of feature engineering, such as an SVM. When moving to deeper nonlinear models, as is the case for our AES system, an alternative approach is to instead represent each word as a dense vector. That is, each word is embedded into a d dimensional space, and represented as a vector in that space (Goldberg 2017). Hence, the dense word vectors are often referred to as *word embeddings* (arguably first mentioned by Bengio et al. 2003). The dimension d is chosen by the researcher and is usually much smaller than the number of words in the vocabulary. In contrast to one-hot vectors, word embeddings are learned from data. Moreover, the use of word embeddings allows for the concept of word similarity. For example, if the words *apple* and *pear* appear in similar context (which they probably do since they are both fruits), their respective word embedding should also be similar.

As mentioned above, word embeddings are learned from text data. There are numerous methods to do this, some more successful than others. Usually, it all revolves around a so called *embedding layer*. The input to this embedding layer is the one-hot vectors representing the words in the text data. The output of

this layer will then be the word embeddings, making the embeddings a function of the one-hot vector.

In similarity to the approach with one-hot vectors, consider a vocabulary of W words, each represented (embedded) as a d dimensional vector. The combination of these word vectors then forms the $W \times d$ embedding matrix $\boldsymbol{E}$, where each row corresponds to an embedded word. Each element in matrix $\boldsymbol{E}$ could be randomly initialized with small numbers or use pre-tuned weights from an auxiliary task (explained in next section). Let $\boldsymbol{w}_i$ be the one-hot vector of word w_i, which is a W dimensional vector with all zeros except for one index, corresponding to the value of the ith word, where the value is 1 (similar to the ones in Figure 6). Taking the multiplication $\boldsymbol{w}_i\boldsymbol{E}$ will then extract the corresponding row of $\boldsymbol{E}$, creating the word embedding $v(w_i)$. This is done for every unique word, and together these word embeddings form the first layer of the AES model, the embedding layer (see Figure 7 for an illustration). The randomly initialized word embedding will then be updated the same way as the other parts of the AES system in order to minimize the loss function. This layer will then be followed by the recurrent layer, where each word embedding will be taken as one input in a sequence (as illustrated in Figure 4 from Section 3.3.5). The training of this embedding layer may require a lot of data and time to achieve an adequate result. On the other hand, building an embedding layer from scratch, i.e. using randomly initialized weights and updating these, makes it specifically tailored to the student essays and to the topics which they have been written in.

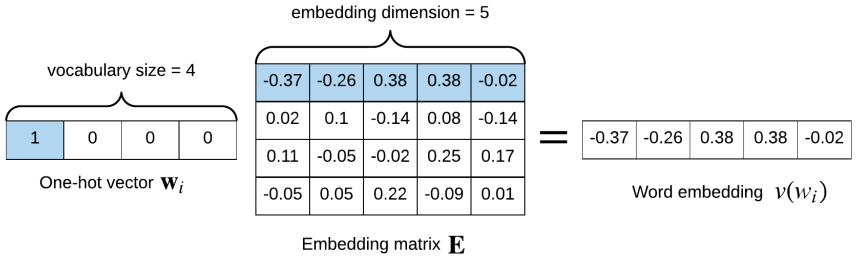

Figure 7: Turning a one-hot vectors into a word embedding.

3.4.2 Pre-trained word embeddings

If the researcher is blessed with large amount of text data, training an embedding layer from scratch could potentially be very beneficial. However, since the embedding layer is trained in combination with the NLP task, it requires supervised training data (i.e. labeled). In contrast to unsupervised data (i.e. unlabeled), supervised data exist in a much more limited quantity. Thus, not uncommonly, making the usefulness of the randomly initialized embedding layer in practical applications less viable. This could be the case for our AES system

since it has to train on relatively few essays.

Fortunately, there are several successful methods for learning word embeddings which do not require labeled data. Collobert and Weston (2008) proposed to learn such word embeddings by implementing a feedforward NN in order to predict a word based on the two words next to it. Thus, creating a supervised training problem from potentially unlabeled data. These word embeddings could later be used as the weights for the embedding matrix $\boldsymbol{E}$ in the embedding layer of the AES system. This concept is often referred to as using *pre-trained word embeddings*. The weights of matrix $\boldsymbol{E}$ may either be treated as fixed during the training process or treated as the randomly initialized word vectors and further optimized as part of the AES system.

The fundamental idea of using unsupervised training in order to learn word embeddings is that words with similar meaning should also have similar vectors. While word similarity being a somewhat vague concept, current methods relies on the *distributional hypothesis* (Harris 1954), stating that *words are similar if they appear in the same context*. As seen in Collobert and Weston (2008), this creates a supervised learning problem of either predicting a word from its context, or predicting the context from the word.

A breakthrough for this approach came with the creation of the *Word2Vec*-family of algorithms (Mikolov, Chen, et al. 2013; Mikolov, Sutskever, et al. 2013). These alogorithms consists of two primary methods for the creation of word embeddings: *Continuous Bag of Words* (CBOW) and *Skip-Gram*. In short, the CBOW approach tries to predict a word from its context, while the skip-gram does the opposite and tries to predict the context from the current word. Unfortunately, the Word2Vec algorithm only officially provide pre-trained word embeddings in English. However, other unsupervised methods have recently been trained on multi-lingual sources, thus enabling access to multi-lingual pre-trained word embeddings. Perhaps most notable of such algorithms, is the one provided by *fastText* (an NLP library provided by Facebook).

FastText most recent efforts provide access to pre-trained word embeddings in 157 languages, including Swedish (Grave et al. 2018). These word vectors are trained on two large and publicly available textual sources: the online encyclopedia Wikipedia and the common crawl, a nonprofit organization which scrapes the web and makes the resulting data publicly available. Wikipedia provides high-quality and peer reviewed data, although for many languages limited in size and coverage. However, this is compensated by the nosier but much larger corpus provided by the common crawl.

The pre-trained word embeddings provided by fastText is learned by implementing a modified version of the skip-gram approach in Word2Vec. As previously mentioned, the goal of the skip-gram model is to predict the context surrounding a given word. It starts by using one word as the input word and then randomly choosing another word from the surroundings of the input word. The output of the model will then give the probability for every unique word in the vocabulary of being the nearby word. In that sense, the higher the probability one word has to be randomly chosen given the input word, the higher similarity exist between the words. For example, if the input word is *apple*, then

there will probably be a higher output probability for the word *pear*, than for the word *car*.

The training of the skip-gram model is performed by inputting word pairs. The basic idea of how this is done is illustrated in Figure 8 (inspired by McCormick 2016). First, the number of surrounding words needs to be determined. This is commonly referred to as the *window size*. In Figure 8, the window size is set to two, meaning two words to the left of the input word, and two words to the right of the input word. Given a text corpus, represented as a sequence of words, for example *green apples are the tastiest*, word pairs in the form of *(input word, nearby word)* are extracted. For example, if the current word is *apples*, then there are three word pairs that are extracted *(apples, green)*,*(apples, are)*, *(apples, the)*. Similarly, if the input word is *are*, then four word pairs are extracted: *(are, green)*, *(are, apples)*, *(are, the)*, *(are, tastiest)*.

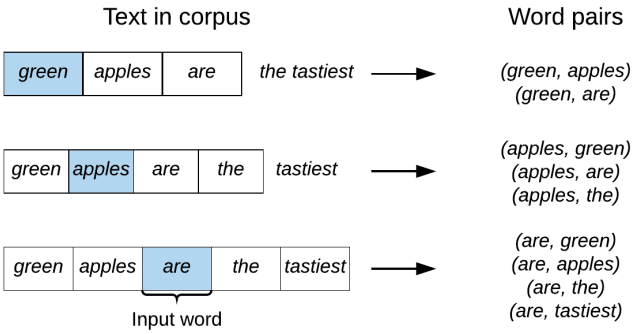

Figure 8: Skip-gram method generating training samples for the creation of pre-trained word embeddings.

Word pairs are extracted for every word in the sequence, and are then fed as training samples into a neural network model, where the task is to predict the nearby word given the input word. That is, the network will learn by using the number of times each word pair appears. When the training stage is done, inputting the word *apple*, will hopefully output a higher probability for *pear* than for the word *car*. The hidden layer which is learned and updated during the training stage is going to be represented by a weight matrix. For example, if there are $10,000$ word in the vocabulary, and the goal is to build word embeddings with size 300, then the weight matrix would have $10,000$ rows (one for each word) and 300 columns (this corresponds to the number of hidden neurons in this layer). Because the weight matrix has been learned from the data and represents every word in the vocabulary, each row of the weight matrix can now be used as a word embedding. This is the ultimate goal of the skip-gram model.

Before implementing the pre-trained embeddings into the AES system, it is

necessary to map the embeddings to the words in the student essays. This was explained in the introduction to the embedding layer. Usually, pre-trained embeddings such as the ones provided by fastText contains a lot of embeddings and often covers a majority of the words in the data. However, there are instances when the textual data contains a very specific language which is not necessarily covered by the pre-trained embeddings. It could for example occur when the texts are from a specific subject with a distinctive terminology, or when the population creating the texts are somehow distinctive. In our case, the latter is certainly the case since we are using texts made by students in upper-secondary school during a time restricted high-stake exam. Arguably, this leads to some of the words in our data not being assigned a learned word embedding. Usually, these words are referred to as *out-of-vocabulary* words (OOV). Another source of OOV words is the prevalence of misspelled words in the text data. In similarity to the case just explained, misspelled words will also fail to be assigned a learned word embedding.

Typically, the preferred method of handling the OOV words have been to treat them uniformly by reserving a special symbol, such as *UnK*, and randomly initialize the weights of this word (Goldberg 2017). When the number of OOV words is relatively small, this method may not entail any problem. However, it could be more harmful to the AES system's performance if there are many words assigned as OOV, which we believe is a likely scenario when dealing with student essays. Fortunately, a plausible solution to this problem is presented in a recent paper by Bojanowski et al. (2016). In this paper, the authors create pre-trained word embeddings by representing each distinct word as the sum of the vector representation of its character-level *n-grams*, which they refer to as subword enriched word embeddings. Character-level n-grams is basically all combinations of adjacent letters of length n that can be found in the input text. For example, given the word *apple*, an n-gram of length 2 (known as 2-gram or *bigram*) are *ap, pp, pl, le*. Now compare the similarity between these bigrams and the bigram of the misspelled word *aple*: *ap, pl, le*. Three out of four bigrams are the same and thus the probability that the network would recognize these as similar is substantially increased in comparison to not using the subword approach. The number of OOV words is also a function of the vocabulary size. If the vocabulary size is small, then the number of OOV words would probably also be very small since only the most used words are included in the vocabulary. On the other hand, if the vocabulary size is relatively large, then the number of OOV words would probably be even larger since the vocabulary contains more infrequent words which have a larger risk of being a misspelled- or a very content specific word.

So, to make the AES system able to understand and optimize to the task of generating grades, an embedding layer will be used as the first component of the network. The limited number of essays as well as the specific vocabulary within these texts will demand specific solutions to be implemented in the system. To enhance the learning capability of the recurrent layer and the rest of the network we will experiment with randomly initialized embeddings as well as pre-trained embeddings. Given that the essays are written within a specific topic

and a certain writing style, we believe that randomly initialized embeddings will be a good choice. However, the fact that our AES will be restricted to far less data than any other neural based AES system we have seen, makes it especially important to create an effective network. This could be achieved by using pre-trained embeddings that are created using subword information. To our knowledge, this type of embedding has not been used in any previous AES model, which makes it particularly interesting to evaluate its performance in this study.

3.5 Setup

This part will bring together the models and methods presented in previous sections in order to give a comprehensive understanding of the layout of the AES system. Details of each respective component is found in the methodology section.

An overview of our AES system is given in Figure 9. It illustrates how one of the sample essays are transformed from a collection of words $w_1, w_2, ..., w_M$ into an output probability, which is then classified into one of the four potential grade categories.

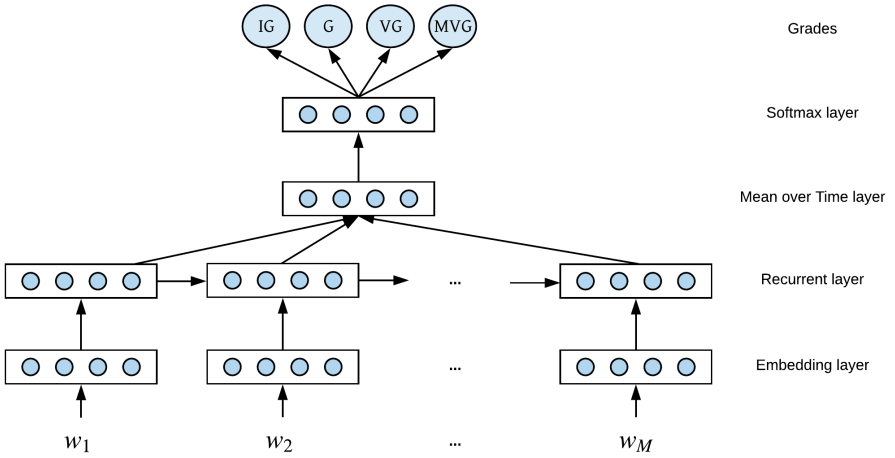

Figure 9: A generalization of our AES system architecture.

The first layer of the AES system is the embedding layer. As discussed in Section 3.4, this layer can create word embeddings by randomly initializing weights and learning them as part of minimizing the loss function of the network. Another way of doing this is to utilize pre-trained word embeddings from the fastText library. For the latter, the word embeddings can be further learned from data which is similar to learning them from scratch. Another option is

to freeze the embedding layer in order to preserve the context created by the auxiliary skip-gram method. We will explore all three methods. The vocabulary size will also be restricted to the 10,000 most occurring words. This is done since a smaller vocabulary means less computational time. Also, initial testing revealed that the performance of the network was not improved with a larger vocabulary size.

The word embeddings are then fed into the recurrent layer. The LSTM unit will be the primary component of this layer. Modifications to this unit in form of a bidirectional LSTM (BLSTM) or a double LSTM/BLSTM layer will also be evaluated. Different number of hidden neurons of this layer(s) will also be tested. In order to prevent overfitting to the training data, dropout will be applied.

The output of the recurrent layer will either be passed through the mean-over-time pooling layer, or/and fed directly into the softmax output layer. The softmax layer will then generate a vector of length four, with values ranging from zero to one where the maximum value will decide which grade that will be assigned to the text. Categorical cross-entropy will be used as the loss function and the adam method will be used as the optimizer.

The hyperparameters of the model are presented in Table 2. When learning the embedding layer from scratch an embedding size of 300 will be used. The LSTM/BLSTM layer(s) will consist of 50 or 100 hidden units. Using more units can improve the generalization performance of the AES, but on the other hand it could also increase overfitting. It is also more computationally expensive to use more units which is why it is interesting to explore if there is any performance increase when using a wider (i.e more hidden units) network. The network will be trained for 20 epochs by using minibatches of size 10. A 5-fold cross validation will be used for each of the resulting 27 models.

Layer	Parameter Name	Parameter Value
Embedding	random initialization	300
	pre-trained	300
LSTM/BLSTM	hidden units	(50, 100)
Dropout	dropout rate	0.5
	epochs	20
	batch size	10
	initital learning rate	0.001
	vocabulary size	10,000

Table 2: Hyperparameters of the network.

The analysis and modeling is primarily done in Python 3.6 using the modules Keras (Chollet 2015) and Scikit-learn (Pedregosa et al. 2011). Keras is a high-level neural network API, which is written in Python and operates on the Tensorflow framework (other frameworks are also available). Scikit-learn is a

machine learning library for Python, which in this case is primarily used to implement the cross-validation as well as calculate the evaluation metric quadratic weighted kappa.

4 Results

This part will present the results from the different network architectures that has been explored. We first present the results from the AES system using the sample I blind rater's grades in order to compare to previous benchmark. We then compare this result to the AES system using data from both sample I and sample II to gain a better insight to the potential performance achievable by using more training data.

It is evident from Table 3 that the most crucial component to the relative success of the AES system is the MoT-layer. In fact, without the MoT-layer the AES system does not perform better than if using random assignment (indicated by the QWK around 0). The second largest factor to the performance is the type of embedding layer that is used. Embeddings created by random initialization and pre-trained embeddings which are learned has the greatest performance. The latter has a slightly better result in some models but due to a relatively high standard deviation no significant distinction can be made. Pre-trained fixed embeddings has a significantly lower performance which makes it evident that it is crucial to let the embeddings to be learned as part of the AES system. This is also justified by the fact that using the pre-trained fixed embeddings results in an AES model which is not able to learn the training data (not shown).

The width and depth of the network has no significant effect on the result. It does not matter if the LSTM uses 50 or 100 neurons, neither does it matter if the system is comprised of one or two LSTM layers. Likewise, there is no indication that a bidirectional LSTM performs better than a unidirectional LSTM, or vice versa. The best models managed to achieve an average QWK of 0.36. The highest QWK score in any of the folds is 0.43, which is achieved by the 2-LSTM MoT (50) using pre-trained learned embeddings.

Model	Embeddings					
	Random		Pre-trained learned		Pre-trained fixed	
	Avg. QWK (Std. dev)	Avg. Acc (Std. dev)	Avg. QWK (Std. dev)	Avg. Acc (Std. dev)	Avg. QWK (Std. dev)	Avg. Acc (Std. dev)
LSTM (100)	0.02 (0.03)	37.73% (1.88%)	0.06 (0.03)	38.07% (2.40%)	0.01 (0.01)	49.42% (0.66%)
LSTM MoT (50)	0.33 (0.04)	52.31% (2.08%)	0.33 (0.04)	51.61% (1.98%)	0.14 (0.07)	52.42% (1.41%)
LSTM MoT (100)	0.35 (0.04)	51.85% (1.68%)	0.35 (0.03)	52.94% (1.99%)	0.18 (0.07)	52.02% (1.41%)
BLSTM MoT (50)	**0.36 (0.02)**	52.41% (2.43%)	0.33 (0.03)	51.04% (1.93%)	0.19 (0.05)	52.25% (1.15%)
BLSTM MoT (100)	0.31 (0.04)	50.18% (3.43%)	**0.36 (0.03)**	52.08% (1.24%)	0.24 (0.07)	52.19% (1.38%)
2-LSTM MoT (50)	0.33 (0.03)	51.04% (2.91%)	**0.36 (0.04)**	51.44% (2.03%)	0.11 (0.05)	52.36% (1.13%)
2-LSTM MoT (100)	0.33 (0.04)	49.88% (1.86%)	0.33 (0.02)	50.4% (2.53%)	0.13 (0.06)	52.13% (1.16%)
2-BLSTM MoT (50)	0.33 (0.02)	49.37% (1.08%)	0.35 (0.01)	51.85% (1.95%)	0.13 (0.04)	52.31% (0.04%)
2-BLSTM MoT (100)	0.32 (0.04)	48.1% (2.12%)	0.32 (0.04)	51.26% (1.76%)	0.18 (0.11)	52.13% (0.7%)

Table 3: Results from AES on sample I with blind rater's grades using a 5-fold cross-validation. Columns are split according to the specifics of the embedding layer. *Random* corresponds to embeddings with random initialization, *pre-trained learned* corresponds to Swedish fastText embeddings and training these, *pre-trained fixed* corresponds to fastText embeddings and not training these.

The classification accuracy (which is the measure of exact interrater agreement) is around 50% — 52% for most models with the MoT-layer. It is on average slightly higher for the models using the pre-trained fixed embeddings. This pattern does not hold for the QWK scores and it is evident that a slightly higher classification accuracy does not necessarily imply a higher average QWK.

A comparison between sample I grades assigned by the blind rater and the AES is shown in Figure 10 and in the confusion matrix in Table 4. The grades from the AES system are created using a LSTM (100) MoT model with random embeddings, which had a QWK score of 0.42 with the blind rater. In similarity to the grades assigned by the blind rater, the AES grades are also favored towards grade *G* and *VG*. However, the AES is clearly having a problem to distinguish *VG*-texts from other grades, resulting in almost doubling the number of *VG*s assigned.

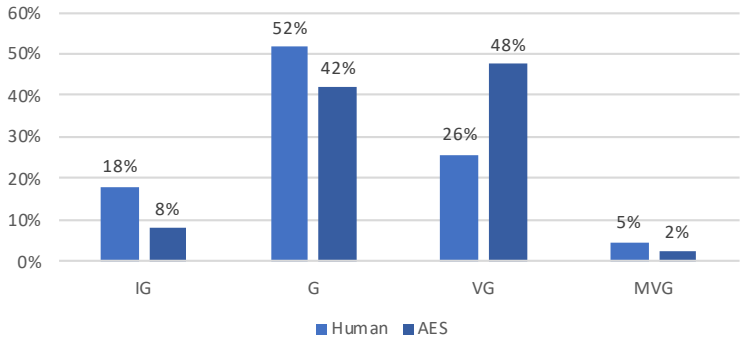

Figure 10: Bar plot over the relative frequency of grades assigned by the blind rater (sample I) versus the AES system. Classification accuracy is 52% and QWK is 0.42.

		AES				
		IG	G	VG	MVG	Sum
Human	IG	10	26	11	0	47
	G	10	71	50	3	134
	VG	1	12	53	1	67
	MVG	0	0	10	2	12
	Sum	21	109	124	6	260

Table 4: Confusion matrix for the grades assigned by the AES system, versus the grades from the blind rater (sample I). Classification accuracy is 52% and QWK is 0.42.

Using essays from both sample I and sample II results in an increase in the performance of the AES , which is seen in Table 5. The overall pattern for the results is quite similar to those found when only using data from sample I. Returning all the intermediate states of the LSTM layer and passing them through the MoT-layer is still the most crucial aspect to the performance of the AES system. There is however a slight increase in QWK and accuracy for the models without the Mot-layer, they are slightly better than random assignment. Similarly to previous results, learning the embeddings as part of minimizing the network's loss yields the best interrater agreement in terms of QWK. Other than the MoT-and the embedding-layer, no specification show significantly higher performance than the others. The highest average QWK is 0.44, which was achieved by two models. The highest QWK score in any of the folds is 0.49, which is achieved by the 2-LSTM MoT (50) using random embeddings.

Model	Embeddings					
	Random		Pre-trained learned		Pre-trained fixed	
	Avg. QWK (Std. dev)	Avg. Acc (Std. dev)	Avg. QWK (Std. dev)	Avg. Acc (Std. dev)	Avg. QWK (Std. dev)	Avg. Acc (Std. dev)
LSTM (100)	0.09 (0.02)	35.44% (0.99%)	0.07 (0.03)	35.37% (1.39%)	0.1 (0.04)	40.36% (1.81%)
LSTM MoT (50)	0.42 (0.02)	50.3.% (0.98%)	0.42 (0.05)	49.47% (2.3%)	0.36 (0.04)	50.46 (2.53%)
LSTM MoT (100)	0.43 (0.02)	49.22% (2.39%)	**0.44 (0.03)**	49.01% (2.54%)	0.36 (0.03)	50.95% (1.89%)
BLSTM MoT (50)	0.43 (0.04)	49.38% (1.71%)	0.42 (0.03)	48.85% (2.42%)	0.35 (0.03)	49.97% (1.97%)
BLSTM MoT (100)	0.43 (0.04)	48.42% (3.34%)	0.43 (0.03)	50.34% (2.08%)	0.36 (0.03)	50.27% (2.27%)
2-LSTM MoT (50)	**0.44 (0.03)**	50.12% (2.03%)	0.42 (0.03)	51.14% (1.74%)	0.33 (0.02)	48.66% (1.61%)
2-LSTM MoT (100)	0.43 (0.03)	49.66% (2.27%)	0.41 (0.03)	49.34% (3.43%)	0.33 (0.03)	48.44% (2.15%)
2-BLSTM MoT (50)	0.39 (0.03)	48.72% (1.57%)	0.42 (0.02)	48.24% (3.94%)	0.37 (0.04)	49.52% (2.49%)
2-BLSTM MoT (100)	0.39 (0.02)	49.07% (1.44%)	0.42 (0.03)	48.63% (1.33%)	0.36 (0.02)	50.4% (1.54%)

Table 5: Results from AES using blind rater's grade from sample I and teacher's grade from sample II using a 5-fold cross-validation. Columns are split according to the specifics of the embedding layer. *Random* corresponds to embeddings with random initialization, *pre-trained learned* corresponds to Swedish fastText embeddings and training these, *pre-trained fixed* corresponds to fastText embeddings and not training these.

A comparison between the grades assigned by the blind rater from sample I and the teacher from sample II versus the AES system is presented in Figure 11 and Table 6. The grades from the AES system are created by a LSTM (100) MoT model with random embeddings, which had a QWK score of 0.46 with the human grades. The difference between the AES and the human grades are somewhat more balanced in this comparison, compared to previous findings. However, it is evident that the AES sytem is having a difficulty separating essays with adjacent grades. Resulting in the overestimation of the two middle grades *G* and *VG*.

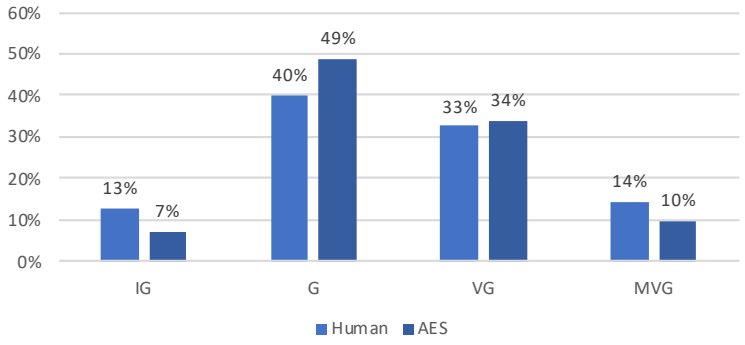

Figure 11: Bar plot over the relative frequency of grades assigned by the blind rater (sample I) and teacher (sample II) versus the AES system. Classification accuracy is 52% with a QWK of 0.46.

		AES				
		IG	G	VG	MVG	Sum
Human	IG	18	36	6	3	63
	G	12	132	40	10	194
	VG	3	57	82	16	158
	MVG	2	12	37	19	70
	Sum	35	237	165	48	485

Table 6: Confusion matrix for the grades assigned by the AES system, versus the grades from the blind rater (sample I) and teacher (sample II). Classification accuracy is 52% with a QWK of 0.46.

5 Discussion

The most crucial component to the relative success of the AES is found to be the mean-over-time pooling layer. Implementing this layer can mean the difference between having an AES system that is to some extent able to make generalizations or not. Without the MoT-layer, the network needs to accurately map the entire essay into one single state vector in order to make a prediction. In contrast, when the MoT-layer is applied, less information is lost since the network can utilize all the intermediate states as well. Perhaps it is the relatively long sequence length of each essay that makes accessing the intermediate states much more effective and produce more accurate predictions.

Learned embeddings showed to give a better result than fixed. Consequently, this implies that the word embeddings created from the Swedish Wikipedia and the common crawl are not adequately tailored to the specific vocabulary and

language use that is present in these essays. Furthermore, this would potentially mean that using more essays in order to build better embeddings could have a positive effect on the performance of the AES system.

The neural based AES models created and evaluated in this study does not provide an adequately high interrater agreement in order to pass the limit of 0.70 QWK set by the ETS. This may however not be a fair comparison since the English AES models often have access to a lot more data. The AES models in this study does come quite close to the Swedish AES designed by Östling et al. (2013), especially for specific folds. These authors also pointed out in their study that the text corpus (corresponding to sample I) is of low quality, which is seen by the low QWK between teacher and blind rater. There is also large variability in both sample sets with sample II having up to 360 unique topics. These factors are likely affecting the performance of our AES. While the result of this study may not directly lead to a reliable and high-performing neural based AES system for Swedish essays, it does say something about the potential of such a system.

In contrast to the AES developed by Östling et al. (2013), our AES model did not rely heavily on manual feature engineering. For the relatively small sample used in this study this resulted in a performance decrease. Perhaps human insight of important linguistic features is crucial when dealing with small sample sizes. However, building a more autonomous AES system with neural networks could be very beneficial in large-scale implementations since it could to a higher degree decide for itself how to successfully grade an essay. Moreover, as seen in this study, neural networks does tend to have increasing returns to scale when it comes to the quantity of the training data. By using more coherent training data, ideally from only one or a few different exams, we could see a great performance increase of our neural network based AES system; potentially leading to a new Swedish benchmark.

As a final note, the Swedish government has tasked the Swedish National Agency for Education with digitizing the national exams (Skolverket 2018). This will mean that students are required to write their short essay in Swedish using a computer, starting from fall 2018. Complete digitization of the exams (all subjects) should take place in 2022. Undoubtedly, these efforts will lead to an excellent opportunity to further test the use of automatic grading systems for Swedish essays. AES systems using neural networks have proven to achieve state-of-the-art results when implemented on English texts. With more data available, we see no reason to why this could not also be the case for a Swedish AES system.

6 Conclusion

In this study, we proposed a recurrent neural network model to handle the task of automatic essay scoring for Swedish upper-secondary essays. We based our model on uni- and bidirectional LSTM layers in order to capture the sequential information present in the student essays. This concept was further explored

by introducing a mean-over-time pooling layer, taking into account all the sequential steps given by the LSTM layer. Furthermore, dense word embeddings was used to represent textual data in the network. These embeddings were constructed either by random initialization and learning them as part of the primary AES task, or by using pre-trained embeddings learned from Swedish Wikipedia and common crawl data.

The AES models used in this study was on average neither able to beat the current Swedish baseline set by Östling et al. (2013) nor the 0.70 QWK acceptance limit set by the ETS. The AES models did however come quite close to the Swedish baseline (especially in specific folds) almost without any manual feature engineering. Implementing the mean-over-time layer and extracting the intermediate steps of the recurrent layer showed a significant increase in performance. The specific design of the embedding layer also resulted in a substantially better AES system as learned embeddings proved to outperform fixed pre-trained embeddings.

Adding more training data also resulted in a performance increase. Highest average QWK when using the 1.736 essays from sample I was 0.36. However, when using all the 3.231 essays from both samples a higher average QWK of 0.44 was achieved.

Neural network based AES systems has shown great result when used on large amount of English text data. The availability of such data has lead to the implementation of English based AES systems in several low- and high stakes assessments such as TOEFL iBT and GRE. In order for this to be a feasible reality in Sweden, much more textual data is needed; preferably also graded by an external rater in order to mitigate any subjective bias. Fortunately, recent and forthcoming efforts made by the Swedish National Agency for Education will hopefully provide such data in an unprecedented magnitude. This will surely be an excellent opportunity for further research within the application of neural AES systems for the Swedish language.

7 References

References

Alikaniotis, Dimitrios, Helen Yannakoudakis, and Marek Rei. 2016. "Automatic text scoring using neural networks." *arXiv preprint arXiv:1606.04289.*

Attali, Yigal, and Jill Burstein. 2004. "AUTOMATED ESSAY SCORING WITH E-RATER® V. 2.0." *ETS Research Report Series* 2004 (2).

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. "A neural probabilistic language model." *Journal of machine learning research* 3 (Feb): 1137–1155.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. "Enriching Word Vectors with Subword Information." *arXiv preprint:1607.04606.*

Burstein, Jill. 2003. "The E-rater® scoring engine: Automated essay scoring with natural language processing."

Burstein, Jill, Karen Kukich, Susanne Wolff, Chi Lu, and Martin Chodorow. 1998. "Computer analysis of essays." In *NCME Symposium on Automated Scoring.*

Burstein, Jill, and Daniel Marcu. 2000. "Benefits of modularity in an automated essay scoring system." In *Proceedings of the COLING-2000 Workshop on using toolsets and architectures to build NLP systems,* 44–50. Association for Computational Linguistics.

Chen, Hongbo, and Ben He. 2013. "Automated essay scoring by maximizing human-machine agreement." In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing,* 1741–1752.

Chollet, François. 2015. *Keras.* `https://github.com/fchollet/keras`. Accessed: 2018-05-15.

———. 2017. *Deep learning with python.* Manning Publications Co.

Chung, Gregory KWK, and Harold F O'Neil Jr. 1997. "Methodological Approaches to Online Scoring of Essays."

Cohen, Jacob. 1960. "A coefficient of agreement for nominal scales." *Educational and psychological measurement* 20 (1): 37–46.

———. 1968. "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit." *Psychological bulletin* 70 (4): 213.

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

Dikli, Semire. 2006. "An overview of automated scoring of essays." *The Journal of Technology, Learning and Assessment* 5 (1).

Dong, Fei, and Yue Zhang. 2016. "Automatic features for essay scoring–an empirical study." In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing,* 1072–1077.

Dong, Fei, Yue Zhang, and Jie Yang. 2017. "Attention-based Recurrent Convolutional Neural Network for Automatic Essay Scoring." In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017),* 153–162.

ETS, Educational Testing Service. 2018a. *Automated Scoring of Writing Quality.* Accessed March 5, 2018. `https://www.ets.org/research/topics/as%7B%5C_%7Dnlp/writing%7B%5C_%7Dquality/`.

———. 2018b. *How the e-rater® Engine Works.* Accessed March 5, 2018. `https://www.ets.org/erater/how/`.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning.* Vol. 1. Springer series in statistics New York.

Goldberg, Yoav. 2017. *Neural network methods for natural language processing.* 10:1–309. 1. Morgan & Claypool Publishers.

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning.* Vol. 1. MIT press Cambridge.

Grave, Edouard, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. "Learning Word Vectors for 157 Languages."

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

Hinnerich, Björn Tyrefors, Erik Höglin, and Magnus Johannesson. 2011. "Are boys discriminated in Swedish high schools?" *Economics of Education review* 30 (4): 682–690.

Hinton, Geoffrey E, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580.*

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

Kingma, Diederik P, and Jimmy Ba. 2014. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980.*

Larkey, Leah S. 1998. "Automatic essay grading using text categorization techniques." In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval,* 90–95. ACM.

Lemaire, Benoit, and Philippe Dessus. 2001. "A system to assess the semantic content of student essays." *Journal of Educational Computing Research* 24 (3): 305–320.

McCormick, Chris. 2016. *Word2Vec Tutorial - The Skip-Gram Model.* Accessed March 19, 2018. `http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/`.

Megyesi, Beáta, Jesper Näsman, and Anne Palmér. 2016. "The Uppsala Corpus of Student Writings: Corpus Creation, Annotation, and Analysis." In *LREC.*

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781.*

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. "Distributed representations of words and phrases and their compositionality." In *Advances in neural information processing systems,* 3111–3119.

Östling, Robert, Andre Smolentzov, Björn Tyrefors Hinnerich, and Erik Höglin. 2013. "Automated essay scoring for swedish." In *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications,* 42–47.

Östlund-Stjärnegårdh, Eva. 2014. *Det nationella kursprovet i Svenska B och Svenska som andraspråk B.* Technical report. Instiutionen för nordiska språk, Uppsala Universitet. `http://www.natprov.nordiska.uu.se/digitalAssets/216/a%7B%5C_%7D216007-f%7B%5C_%7Dslutrapport-svb-96-13.pdf`.

Page, Ellis B. 1967. "Grading essays by computer: Progress report."

———. 1968. "The use of the computer in analyzing student essays." *International review of education* 14 (2): 210–225.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. "Scikit-learn: Machine learning in Python." *Journal of machine learning research* 12 (Oct): 2825–2830.

Phandi, Peter, Kian Ming A Chai, and Hwee Tou Ng. 2015. "Flexible domain adaptation for automated essay scoring using correlated linear regression": 431–439.

Schütze, Hinrich, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval.* Vol. 39. Cambridge University Press.

Shermis, Mark D, and Felicia D Barrera. 2002. "Exit Assessments: Evaluating Writing Ability through Automated Essay Scoring."

Skolverket. 2018. *Digitalisering av nationella prov.* Accessed April 23, 2018. `https://www.skolverket.se/bedomning/nationella-prov/fragor-och-svar/digitalisering-av-nationella-prov-1.265681`.

Smolentzov, André. 2013. "Automated Essay Scoring: Scoring Essays in Swedish."

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15 (1): 1929–1958.

Taghipour, Kaveh, and Hwee Tou Ng. 2016. "A neural approach to automated essay scoring." In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing,* 1882–1891.

Williamson, David M, Xiaoming Xi, and F Jay Breyer. 2012. "A framework for evaluation and use of automated scoring." *Educational measurement: issues and practice* 31 (1): 2–13.

Yannakoudakis, Helen, Ted Briscoe, and Ben Medlock. 2011. "A new dataset and method for automatically grading ESOL texts": 180–189.

Young, Tom, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2017. "Recent trends in deep learning based natural language processing." *arXiv preprint arXiv:1708.02709.*

Zhao, Siyuan, Yaqiong Zhang, Xiaolu Xiong, Anthony Botelho, and Neil Heffernan. 2017. "A Memory-Augmented Neural Model for Automated Grading": 189–192.