A Neural Approach to Automated Essay Scoring

Kaveh Taghipour and **Hwee Tou Ng**
Department of Computer Science
National University of Singapore
13 Computing Drive
Singapore 117417
{kaveh, nght}@comp.nus.edu.sg

Abstract

Traditional automated essay scoring systems rely on carefully designed features to evaluate and score essays. The performance of such systems is tightly bound to the quality of the underlying features. However, it is laborious to manually design the most informative features for such a system. In this paper, we develop an approach based on recurrent neural networks to learn the relation between an essay and its assigned score, without any feature engineering. We explore several neural network models for the task of automated essay scoring and perform some analysis to get some insights of the models. The results show that our best system, which is based on long short-term memory networks, outperforms a strong baseline by 5.6% in terms of quadratic weighted Kappa, without requiring any feature engineering.

1 Introduction

There is a recent surge of interest in neural networks, which are based on continuous-space representation of the input and non-linear functions. Hence, neural networks are capable of modeling complex patterns in data. Moreover, since these methods do not depend on manual engineering of features, they can be applied to solve problems in an end-to-end fashion. SENNA (Collobert et al., 2011) and neural machine translation (Bahdanau et al., 2015) are two notable examples in natural language processing that operate without any external task-specific knowledge. In this paper, we report a system based on neural networks to take advantage of their modeling capacity and generalization power for the automated essay scoring (AES) task.

Essay writing is usually a part of the student assessment process. Several organizations, such as Educational Testing Service (ETS)[1], evaluate the writing skills of students in their examinations. Because of the large number of students participating in these exams, grading all essays is very time-consuming. Thus, some organizations have been using AES systems to reduce the time and cost of scoring essays.

Automated essay scoring refers to the process of grading student essays without human interference. An AES system takes as input an essay written for a given prompt, and then assigns a numeric score to the essay reflecting its quality, based on its content, grammar, and organization. Such AES systems are usually based on regression methods applied to a set of carefully designed features. The process of feature engineering is the most difficult part of building AES systems. Moreover, it is challenging for humans to consider all the factors that are involved in assigning a score to an essay.

Our AES system, on the other hand, *learns* the features and relation between an essay and its score automatically. Since the system is based on recurrent neural networks, it can effectively encode the information required for essay evaluation and learn the complex patterns in the data through non-linear neural layers. Our system is among the first AES systems based on neural networks designed without any hand-crafted features. Our results show that our system outperforms a strong baseline and

[1] https://www.ets.org

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1882–1891,
Austin, Texas, November 1-5, 2016.

achieves state-of-the-art performance in automated essay scoring. In order to make it easier for other researchers to replicate our results, we have made the source code of our system publicly available[2].

The rest of this paper is organized as follows. Section 2 gives an overview of related work in the literature. Section 3 describes the automated essay scoring task and the evaluation metric used in this paper. We provide the details of our approach in Section 4, and present and discuss the results of our experimental evaluation in Section 5. Finally, we conclude the paper in Section 6.

2 Related Work

There exist many automated essay scoring systems (Shermis and Burstein, 2013) and some of them are being used in high-stakes assessments. e-rater (Attali and Burstein, 2004) and Intelligent Essay Assessor (Foltz et al., 1999) are two notable examples of AES systems. In 2012, a competition on automated essay scoring called 'Automated Student Assessment Prize' (ASAP)[3] was organized by Kaggle and sponsored by the Hewlett Foundation. A comprehensive comparison of AES systems was made in the ASAP competition. Although many AES systems have been developed to date, they have been built with hand-crafted features and supervised machine learning algorithms.

Researchers have devoted a substantial amount of effort to design effective features for automated essay scoring. These features can be as simple as essay length (Chen and He, 2013) or more complicated such as lexical complexity, grammaticality of a text (Attali and Burstein, 2004), or syntactic features (Chen and He, 2013). Readability features (Zesch et al., 2015) have also been proposed in the literature as another source of information. Moreover, text coherence has also been exploited to assess the flow of information and argumentation of an essay (Chen and He, 2013). A detailed overview of the features used in AES systems can be found in (Zesch et al., 2015). Moreover, some attempts have been made to address different aspects of essay writing independently. For example, argument strength and organization of essays have been tackled by some researchers through designing task-specific features for each aspect (Persing et al., 2010; Persing and Ng, 2015).

Our system, however, accepts an essay text as input directly and learns the features automatically from the data. To do so, we have developed a method based on recurrent neural networks to score the essays in an end-to-end manner. We have explored a variety of neural network models in this paper to identify the most suitable model. Our best model is a long short-term memory neural network (Hochreiter and Schmidhuber, 1997) and is trained as a regression method. Similar recurrent neural network approaches have recently been used successfully in a number of other NLP tasks. For example, Bahdanau et al. (2015) have proposed an attentive neural approach to machine translation based on gated recurrent units (Cho et al., 2014). Neural approaches have also been used for syntactic parsing. In (Vinyals et al., 2015), long short-term memory networks have been used to obtain parse trees by using a sequence-to-sequence model and formulating the parsing task as a sequence generation problem. Apart from these examples, recurrent neural networks have also been used for opinion mining (Irsoy and Cardie, 2014), sequence labeling (Ma and Hovy, 2016), language modeling (Kim et al., 2016; Sundermeyer et al., 2015), etc.

3 Automated Essay Scoring

In this section, we define the automated essay scoring task and the evaluation metric used for assessing the quality of AES systems.

3.1 Task Description

Automated essay scoring systems are used in evaluating and scoring student essays written based on a given prompt. The performance of these systems is assessed by comparing their scores assigned to a set of essays to human-assigned gold-standard scores. Since the output of AES systems is usually a real-valued number, the task is often addressed as a supervised machine learning task (mostly by regression or preference ranking). Machine learning algorithms are used to learn the relationship between the essays and reference scores.

[2]https://github.com/nusnlp/nea

[3]https://www.kaggle.com/c/asap-aes

3.2 Evaluation Metric

The output of an AES system can be compared to the ratings assigned by human annotators using various measures of correlation or agreement (Yannakoudakis and Cummins, 2015). These measures include Pearson's correlation, Spearman's correlation, Kendall's Tau, and quadratic weighted Kappa (QWK). The ASAP competition adopted QWK as the official evaluation metric. Since we use the ASAP data set for evaluation in this paper, we also use QWK as the evaluation metric in our experiments.

Quadratic weighted Kappa is calculated as follows. First, a weight matrix $\mathbf{W}$ is constructed according to Equation 1:

$$\mathbf{W}_{i,j} = \frac{(i-j)^2}{(N-1)^2} \tag{1}$$

where i and j are the reference rating (assigned by a human annotator) and the hypothesis rating (assigned by an AES system), respectively, and N is the number of possible ratings. A matrix $\mathbf{O}$ is calculated such that $\mathbf{O}_{i,j}$ denotes the number of essays that receive a rating i by the human annotator and a rating j by the AES system. An expected count matrix $\mathbf{E}$ is calculated as the outer product of histogram vectors of the two (reference and hypothesis) ratings. The matrix $\mathbf{E}$ is then normalized such that the sum of elements in $\mathbf{E}$ and the sum of elements in $\mathbf{O}$ are the same. Finally, given the matrices $\mathbf{O}$ and $\mathbf{E}$, the QWK score is calculated according to Equation 2:

$$\kappa = 1 - \frac{\sum_{i,j} \mathbf{W}_{i,j}\mathbf{O}_{i,j}}{\sum_{i,j} \mathbf{W}_{i,j}\mathbf{E}_{i,j}} \tag{2}$$

In our experiments, we compare the QWK score of our system to well-established baselines. We also perform a one-tailed paired t-test to determine whether the obtained improvement is statistically significant.

4 A Recurrent Neural Network Approach

Recurrent neural networks are one of the most successful machine learning models and have attracted the attention of researchers from various fields. Compared to feed-forward neural networks, recurrent neural networks are theoretically more powerful and are capable of learning more complex patterns from data. Therefore, we have mainly focused on recurrent networks in this paper. This section gives a description of the recurrent neural network architecture that we have used for the essay scoring task and the training process.

4.1 Model Architecture

The neural network architecture that we have used in this paper is illustrated in Figure 1. We now describe each layer in our neural network in detail.

Lookup Table Layer: The first layer of our neural network projects each word into a d_{LT} dimensional space. Given a sequence of words $\mathcal{W}$ represented by their *one-hot* representations $(\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_M)$, the output of the lookup table layer is calculated by Equation 3:

$$LT(\mathcal{W}) = (\mathbf{E}.\mathbf{w}_1, \mathbf{E}.\mathbf{w}_2, \cdots, \mathbf{E}.\mathbf{w}_M) \tag{3}$$

where $\mathbf{E}$ is the word embeddings matrix and will be learnt during training.

Convolution Layer: Once the dense representation of the input sequence $\mathcal{W}$ is calculated, it is fed into the recurrent layer of the network. However, it might be beneficial for the network to extract *local features* from the sequence before applying the recurrent operation. This *optional* characteristic can be achieved by applying a convolution layer on the output of the lookup table layer. In order to extract local features from the sequence, the convolution layer applies a linear transformation to all M windows in the given sequence of vectors[4]. Given a window of dense word representations $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_l$, the convolution layer first concatenates these vectors to form a vector $\bar{\mathbf{x}}$ of length $l.d_{LT}$ and then uses Equation 4 to calculate the output vector of length d_c:

$$Conv(\bar{\mathbf{x}}) = \mathbf{W}.\bar{\mathbf{x}} + \mathbf{b} \tag{4}$$

In Equation 4, $\mathbf{W}$ and $\mathbf{b}$ are the parameters of the network and are *shared* across all windows in the sequence.

[4]The number of input vectors and the number of output vectors of the convolution layer are the same because we pad the sequence to avoid losing border windows.

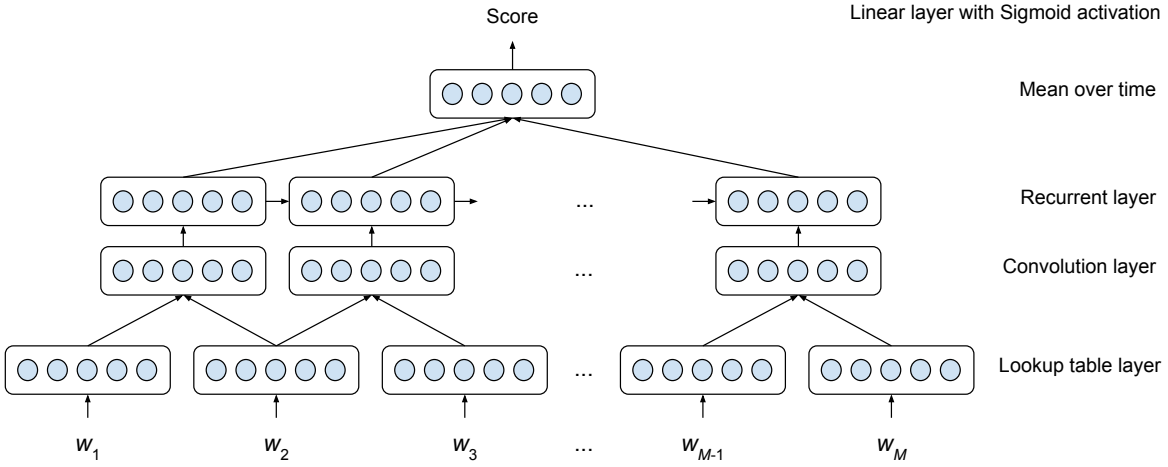

Figure 1: The convolutional recurrent neural network architecture.

The convolution layer can be seen as a function that extracts feature vectors from n-grams. Since this layer provides n-gram level information to the subsequent layers of the neural network, it can potentially capture local contextual dependencies in the essay and consequently improve the performance of the system.

Recurrent Layer: After generating embeddings (whether from the convolution layer or directly from the lookup table layer), the recurrent layer starts processing the input to generate a representation for the given essay. This representation should ideally encode all the information required for grading the essay. However, since the essays are usually long, consisting of hundreds of words, the learnt vector representation might not be sufficient for accurate scoring. For this reason, we preserve all the intermediate states of the recurrent layer to keep track of the important bits of information from processing the essay. We experimented with basic recurrent units (RNN) (Elman, 1990), gated recurrent units (GRU) (Cho et al., 2014), and long short-term memory units (LSTM) (Hochreiter and Schmidhuber, 1997) to identify the best choice for our task. Since LSTM outperforms the other two units, we only describe LSTM in this section.

Long short-term memory units are modified recurrent units that can cope with the problem of vanishing gradients more effectively (Pascanu et al., 2013). LSTMs can learn to preserve or forget the information required for the final representation. In order to control the flow of information during processing of the input sequence, LSTM units make use of three gates to discard (forget) or pass the information through time. The following equations formally describe the LSTM function:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i.\mathbf{x}_t + \mathbf{U}_i.\mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f.\mathbf{x}_t + \mathbf{U}_f.\mathbf{h}_{t-1} + \mathbf{b}_f) \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c.\mathbf{x}_t + \mathbf{U}_c.\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{i}_t \circ \tilde{\mathbf{c}}_t + \mathbf{f}_t \circ \mathbf{c}_{t-1} \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o.\mathbf{x}_t + \mathbf{U}_o.\mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{c}_t)
\end{aligned} \tag{5}$$

$\mathbf{x}_t$ and $\mathbf{h}_t$ are the input and output vectors at time t, respectively. $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{W}_c$, $\mathbf{W}_o$, $\mathbf{U}_i$, $\mathbf{U}_f$, $\mathbf{U}_c$, and $\mathbf{U}_o$ are weight matrices and $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_c$, and $\mathbf{b}_o$ are bias vectors. The symbol $\circ$ denotes element-wise multiplication and σ represents the sigmoid function.

Mean over Time: The outputs of the recurrent layer, $\mathcal{H} = (\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_M)$, are fed into the mean-over-time layer. This layer receives M vectors of length d_r as input and calculates an average vector of the same length. This layer's function is defined in Equation 6:

$$MoT(\mathcal{H}) = \frac{1}{M}\sum_{t=1}^{M}\mathbf{h}_t \tag{6}$$

The mean-over-time layer is responsible for aggregating the variable number of inputs into a fixed length vector. Once this vector is calculated, it is fed into the linear layer to be mapped into a score.

Instead of taking the mean of the intermediate recurrent layer states $\mathbf{h}_t$, we could use the last state vector $\mathbf{h}_M$ to compute the score and remove the mean-over-time layer. However, as we will show in Section 5.2, it is much more effective to use the mean-over-time layer and take all recurrent states into account.

Linear Layer with Sigmoid Activation: The linear layer maps its input vector generated by the mean-over-time layer to a scalar value. This mapping is simply a linear transformation of the input vector and therefore, the computed value is not bounded. Since we need a bounded value in the range of valid scores for each prompt, we apply a sigmoid function to limit the possible scores to the range of $(0, 1)$. The mapping of the linear layer after applying the sigmoid activation function is given by Equation 7:

$$s(\mathbf{x}) = \text{sigmoid}(\mathbf{w}.\mathbf{x} + b) \tag{7}$$

where $\mathbf{x}$ is the input vector ($MoT(\mathcal{H})$), $\mathbf{w}$ is the weight vector, and b is the bias value.

We normalize all gold-standard scores to $[0, 1]$ and use them to train the network. However, during testing, we rescale the output of the network to the original score range and use the rescaled scores to evaluate the system.

4.2 Training

We use the RMSProp optimization algorithm (Dauphin et al., 2015) to minimize the mean squared error (MSE) loss function over the training data. Given N training essays and their corresponding normalized gold-standard scores s_i^*, the model computes the predicted scores s_i for all training essays and then updates the network parameters such that the mean squared error is minimized. The loss function is shown in Equation 8:

$$MSE(\mathbf{s}, \mathbf{s}^*) = \frac{1}{N}\sum_{i=1}^{N}(s_i - s_i^*)^2 \tag{8}$$

Additionally, we make use of dropout regularization to avoid overfitting. We also clip the gradient if the norm of the gradient is larger than a threshold.

We do not use any early stopping methods. Instead, we train the neural network model for a fixed number of epochs and monitor the performance of the model on the development set after each epoch. Once training is finished, we select the model with the best QWK score on the development set.

5 Experiments

In this section, we describe our experimental setup and present the results. Moreover, an analysis of the results and some discussion are provided in this section.

5.1 Setup

The dataset that we have used in our experiments is the same dataset used in the ASAP competition run by Kaggle (see Table 1 for some statistics). We use quadratic weighted Kappa as the evaluation metric, following the ASAP competition. Since the test set used in the competition is not publicly available, we use 5-fold cross validation to evaluate our systems. In each fold, 60% of the data is used as our training set, 20% as the development set, and 20% as the test set. We train the model for a fixed number of epochs and then choose the best model based on the development set. We tokenize the essays using the NLTK[5] tokenizer, lowercase the text, and normalize the gold-standard scores to the range of $[0, 1]$. During testing, we rescale the system-generated normalized scores to the original range of scores and measure the performance.

Prompt	#Essays	Avg length	Scores
1	1,783	350	2–12
2	1,800	350	1–6
3	1,726	150	0–3
4	1,772	150	0–3
5	1,805	150	0–4
6	1,800	150	0–4
7	1,569	250	0–30
8	723	650	0–60

Table 1: Statistics of the ASAP dataset.

In order to evaluate the performance of our system, we compare it to a publicly available open-source[6] AES system called 'Enhanced AI Scor-

[5]http://www.nltk.org

[6]https://github.com/edx/ease

ing Engine' (EASE). This system is the best open-source system that participated in the ASAP competition, and was ranked third among all 154 participating teams. EASE is based on hand-crafted features and regression methods. The features that are extracted by EASE can be categorized into four classes:

- Length-based features
- Parts-of-Speech (POS)
- Word overlap with the prompt
- Bag of n-grams

After extracting the features, a regression algorithm is used to build a model based on the training data. The details of the features and the results of using support vector regression (SVR) and Bayesian linear ridge regression (BLRR) are reported in (Phandi et al., 2015). We use these two regression methods as our baseline systems.

Our system has several hyper-parameters that need to be set. We use the RMSProp optimizer with decay rate (ρ) set to 0.9 to train the network and we set the base learning rate to 0.001. The mini-batch size is 32 in our experiments[7] and we train the network for 50 epochs. The vocabulary is the 4,000 most frequent words in the training data and all other words are mapped to a special token that represents unknown words. We regularize the network by using dropout (Srivastava et al., 2014) and we set the dropout probability to 0.5. During training, the norm of the gradient is clipped to a maximum value of 10. We set the word embedding dimension (d_{LT}) to 50 and the output dimension of the recurrent layer (d_r) to 300. If a convolution layer is used, the window size (l) is set to 3 and the output dimension of this layer (d_c) is set to 50. Finally, we initialize the lookup table layer using pre-trained word embeddings[8] released by Zou et al. (2013). Moreover, the bias value of the linear layer is initialized such that the network's output before training is almost equal to the average score in the training data.

[7]To create mini-batches for training, we pad all essays in a mini-batch using a dummy token to make them have the same length. To eliminate the effect of padding tokens during training, we mask them to prevent the network from miscalculating the gradients.

[8]http://ai.stanford.edu/∼wzou/mt

We have performed several experiments to identify the best model architecture for our task. These architectural choices are summarized below:

- Convolutional vs. recurrent neural network
- RNN unit type (basic RNN, GRU, or LSTM)
- Using mean-over-time over all recurrent states vs. using only the last recurrent state
- Using mean-over-time vs. an attention mechanism
- Using a recurrent layer vs. a convolutional recurrent layer
- Unidirectional vs. bidirectional LSTM

We have used 8 Tesla K80 GPUs to perform our experiments in parallel.

5.2 Results and Discussion

In this section, we present the results of our evaluation by comparing our system to the above-mentioned baselines (SVR and BLRR). Table 2 (rows 1 to 4) shows the QWK scores of our systems on the eight prompts from the ASAP dataset[9]. This table also contains the results of our statistical significance tests. The baseline score that we have used for hypothesis testing is underlined and the statistically significant improvements ($p < 0.05$) over the baseline are marked with '*'. It should be noted that all neural network models in Table 2 are unidirectional and include the mean-over-time layer. Except for the CNN model, convolution layer is *not* included in the networks.

According to Table 2, all model variations are able to learn the task properly and perform competitively compared to the baselines. However, LSTM performs significantly better than all other systems and outperforms the baseline by a large margin (4.1%). However, basic RNN falls behind other models and does not perform as accurately as GRU or LSTM.

[9]To aggregate the QWK scores of all prompts, Fisher transformation was used in the ASAP competition before averaging QWK scores. However, we found that applying Fisher transformation only slightly changes the scores. (If we apply this method to aggregate QWK scores, our best ensemble system (row 7, Table 2) would obtain a QWK score of 0.768.) Therefore we simply take the average of QWK scores across prompts.

ID	Systems	Prompts								
		1	2	3	4	5	6	7	8	Avg QWK
1	CNN	0.797	0.634	0.646	0.767	0.746	0.757	0.746	**0.687**	0.722
2	RNN	0.687	0.633	0.552	0.744	0.732	0.757	0.743	0.553	0.675
3	GRU	0.616	0.591	0.668	0.787	0.795	0.800	0.752	0.573	0.698
4	LSTM	0.775	0.687	0.683	0.795	**0.818**	0.813	0.805	0.594	0.746*
5	CNN (10 runs)	0.804	0.656	0.637	0.762	0.752	0.765	0.750	0.680	0.726*
6	LSTM (10 runs)	0.808	**0.697**	0.689	**0.805**	**0.818**	**0.827**	**0.811**	0.598	0.756*
7	(5) + (6)	**0.821**	0.688	**0.694**	**0.805**	0.807	0.819	0.808	0.644	**0.761***
8	EASE (SVR)	0.781	0.621	0.630	0.749	0.782	0.771	0.727	0.534	0.699
9	EASE (BLRR)	0.761	0.606	0.621	0.742	0.784	0.775	0.730	0.617	0.705

Table 2: The QWK scores of the various neural network models and the baselines. The baseline for the statistical significance tests is underlined and statistically significant improvements ($p < 0.05$) are marked with '*'.

This behaviour is probably because of the relatively long sequences of words in essays. GRU and LSTM have been shown to 'remember' sequences and long-term dependencies much more effectively and therefore, we believe this is the reason behind RNN's relatively poor performance.

Additionally, we perform some experiments to evaluate ensembles of our systems. We create variants of our network by training with different random initializations of the parameters. To combine these models, we simply take the average of the scores predicted by these networks. This approach is shown to improve performance by reducing the variance of the model and therefore make the predictions more accurate. Table 2 (rows 5 and 6) shows the results of CNN and LSTM ensembles over 10 runs. Moreover, we combine CNN ensembles and LSTM ensembles together to make the predictions (row 7).

As shown in Table 2, ensembles of models always lead to improvements. We obtain 0.4% and 1.0% improvement from CNN and LSTM ensembles, respectively. However, our best model (row 7 in Table 2) is the ensemble of 10 instances of CNN models and 10 instances of LSTM models and outperforms the baseline BLRR system by 5.6%.

It is possible to use the last state of the recurrent layer to predict the score instead of taking the mean over all intermediate states. In order to observe the effects of this architectural choice, we test the network with and without the mean-over-time layer. The results of this experiment are presented in Table 3, clearly showing that the neural network fails to learn the task properly in the absence of the mean-over-time layer. When the mean-over-time layer is not used in the model, the network needs to efficiently encode the whole essay into a single state vector and then use it to predict the score. However, when the mean-over-time layer is included, the model has *direct* access to all intermediate states and can recall the required intermediate information much more effectively and therefore is able to predict the score more accurately.

Systems	Avg QWK
LSTM	**0.746***
LSTM w/o MoT	0.540
LSTM+attention	0.731*
CNN+LSTM	0.708
BLSTM	0.699
EASE (SVR)	0.699
EASE (BLRR)	0.705

Table 3: The QWK scores of LSTM neural network variants. The baseline for the statistical significance tests is underlined and statistically significant improvements ($p < 0.05$) are marked with '*'.

Additionally, we experiment with three other neural network architectures. Instead of using mean-over-time to average intermediate states, we use an attention mechanism (Bahdanau et al., 2015) to compute a weighted sum of the states. In this case, we calculate the dot product of the intermediate states and a vector trained by the neural network, and then apply a softmax operation to obtain the normalized weights. Another alternative is to add a convolution layer before feeding the embeddings to the recurrent LSTM layer (CNN+LSTM) and evaluate the model. We also use a bidirectional LSTM model (BLSTM), in which the sequence of words

is processed in both directions and the intermediate states generated by both LSTM layers are merged and then fed into the mean-over-time layer. The results of testing these architectures are summarized in Table 3.

The attention mechanism significantly improves the results compared to LSTM without mean-over-time, but it does not perform as well as LSTM with mean-over-time. The other two architectural choices do not lead to further improvements over the LSTM neural network. This observation is in line with the findings of some other researchers (Kadlec et al., 2015) and is probably because of the relatively small number of training examples compared to the capacity of the models.

We have also compared the accuracy of our best system (shown as 'AES') with human performance, presented in Table 4. To do so, we calculate the agreement (QWK scores) between our system and each of the two human annotators separately ('AES - H1' and 'AES - H2'), as well as the agreement between the two human annotators ('H1 - H2'). According to Table 4, the performance of our system on average is very close to human annotators. In fact, for some of the prompts, the agreement between our system and the human annotators is even higher than the agreement between human annotators. In general, we can conclude that our method is just below the upper limit and approaching human-level performance.

We also compare our system to a recently published automated essay scoring method based on neural networks (Alikaniotis et al., 2016). Instead of performing cross validation, Alikaniotis et al. (2016) partition the ASAP dataset into two parts by using 80% of the data for training and the remaining 20% for testing. For comparison, we also carry out an experiment on the same training and test data used in (Alikaniotis et al., 2016). Following how QWK scores are computed in Alikaniotis et al. (2016), instead of calculating QWK for each prompt separately and averaging them, we calculate the QWK score for the whole test set, by setting the minimum score to 0 and the maximum score to 60. Using this evaluation setup, our LSTM system achieves a QWK score of 0.987, higher than the QWK score of 0.96 of the best system in (Alikaniotis et al., 2016). In this way of calculating QWK scores, since

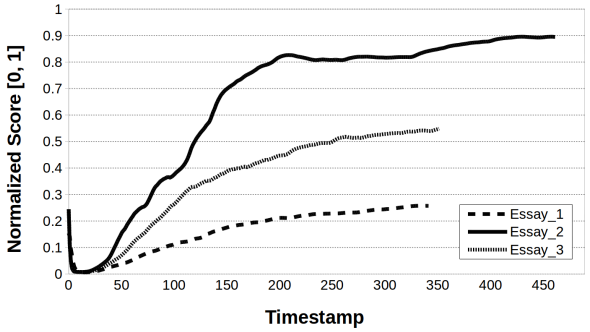

Figure 2: Score variations per timestamp. All scores are normalized to the range of $[0, 1]$.

the majority of the test essays have a much smaller score range (see Table 1) compared to $[0, 60]$, the differences between the system-predicted scores and the gold-standard scores will be small most of the time. For example, more than 55% of the essays in the test set have a score range of $[0, 3]$ or $[0, 4]$ and therefore, for these prompts, the differences between human-assigned gold-standard scores and the scores predicted by an AES system will be small in the range of $[0, 60]$. For this reason, in contrast to prompt-specific QWK calculation, the QWK scores are much higher in this evaluation setting and far exceed the QWK score for human agreement when computed in a prompt-specific way (see Table 4).

Interpreting neural network models and the interactions between nodes is not an easy task. However, it is possible to gain an insight of a network by analyzing the behavior of particular nodes. In order to understand how our neural network assigns the scores, we monitor the score variations while testing the model. Figure 2 displays the score variations for three essays after processing each word (at each timestamp) by the neural network. We have selected a poorly written essay, a well written essay, and an average essay with *normalized* gold-standard scores of 0.2, 0.8, and 0.6, respectively.

According to Figure 2, the network learns to take *essay length* into account and assigns a very low score to all short essays with fewer than 50 words, regardless of the content. This pattern recurs for all essays and is not specific to the three selected essays in Figure 2. However, if an essay is long enough, the content becomes more important and the AES system starts discriminating well written

Description	Prompts								
	1	2	3	4	5	6	7	8	Avg QWK
AES - H1	0.750	0.684	0.662	0.759	0.751	0.791	0.731	0.607	0.717
AES - H2	0.767	0.690	0.632	0.762	0.769	0.775	0.752	0.530	0.710
H1 - H2	0.721	0.812	0.769	0.851	0.753	0.776	0.720	0.627	0.754

Table 4: Comparison with human performance. H1 and H2 denote human rater 1 and human rater 2, respectively, and AES refers to our best system (ensemble of CNN and LSTM models).

essays from poorly written ones. As shown in Figure 2, the model properly assigns a higher score to the well written essay 2, while giving lower scores to the other essays. This observation confirms that the model successfully learns the required features for automated essay scoring. While it is difficult to associate different parts of the neural network model with specific features, it is clear that appropriate indicators of essay quality are being learnt, including essay length and essay content.

6 Conclusion

In this paper, we have proposed an approach based on recurrent neural networks to tackle the task of automated essay scoring. Our method does not rely on any feature engineering and automatically learns the representations required for the task. We have explored a variety of neural network model architectures for automated essay scoring and have achieved significant improvements over a strong open-source baseline. Our best system outperforms the baseline by 5.6% in terms of quadratic weighted Kappa. Furthermore, an analysis of the network has been performed to get an insight of the recurrent neural network model and we show that the method effectively utilizes essay content to extract the required information for scoring essays.

Acknowledgments

This research is supported by Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2013-T2-1-150. We are also grateful to the anonymous reviewers for their helpful comments.

References

Dimitrios Alikaniotis, Helen Yannakoudakis, and Marek Rei. 2016. Automatic text scoring using neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Yigal Attali and Jill Burstein. 2004. Automated essay scoring with e-rater® v. 2.0. Technical report, Educational Testing Service.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations*.

Hongbo Chen 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*.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Yann N. Dauphin, Harm de Vries, and Yoshua Bengio. 2015. Equilibrated adaptive learning rates for non-convex optimization. In *Advances in Neural Information Processing Systems 28*.

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

Peter W Foltz, Darrell Laham, and Thomas K Landauer. 1999. The Intelligent Essay Assessor: Applications to educational technology. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*.

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

Ozan Irsoy and Claire Cardie. 2014. Opinion mining with deep recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Rudolf Kadlec, Martin Schmid, and Jan Kleindienst. 2015. Improved deep learning baselines for Ubuntu corpus dialogs. In *Proceesings of the NIPS 2015*

Workshop on Machine Learning for Spoken Language Understanding and Interaction.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*.

Isaac Persing and Vincent Ng. 2015. Modeling argument strength in student essays. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*.

Isaac Persing, Alan Davis, and Vincent Ng. 2010. Modeling organization in student essays. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*.

Peter Phandi, Kian Ming A. Chai, and Hwee Tou Ng. 2015. Flexible domain adaptation for automated essay scoring using correlated linear regression. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Mark D. Shermis and Jill Burstein, editors. 2013. *Handbook of Automated Essay Evaluation: Current Applications and New Directions*. Routledge.

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

Martin Sundermeyer, Hermann Ney, and Ralf Schlüter. 2015. From feedforward to recurrent LSTM neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):517–529.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28*.

Helen Yannakoudakis and Ronan Cummins. 2015. Evaluating the performance of automated text scoring systems. In *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications*.

Torsten Zesch, Michael Wojatzki, and Dirk Scholten-Akoun. 2015. Task-independent features for automated essay grading. In *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications*.

Will Y. Zou, Richard Socher, Daniel Cer, and Christopher D. Manning. 2013. Bilingual word embeddings for phrase-based machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.