Automated Essay Grader

Shivesh Ganju
Courant Institute Of Mathematical Sciences
New York University
sg6148@nyu.edu

Ravi Shankar
Courant Institute of Mathematical Sciences,
New York University
rs6980@nyu.edu

1. Introduction

In this modern era of revolution, the education sector continues to thrive on the Natural Language Processing Techniques for automating manual human labour tasks. One such application is automatically scoring essays without human intervention.
Essays continue to be one of the key aspects for testing students from different backgrounds and ages. Several examinations such as Graduate Record Examination (GRE) , Scholastic Aptitude Test (SAT) use essay writing for the evaluation process. Since a large number of students appear for such tests grading all of them manually requires humongous effort and time, thus several testing organizations such as Educational Testing Services (ETS) have adopted the *Automated Essay Scoring* (AES) so as to reduce their costs and time.
An AES system takes as an input an essay from a given topic and then assigns a score proportional to grammar, content, organizational structure correctness etc. The process for feature engineering for AES systems is a challenging task which is why a neural network approach can be tried as neural networks are capable of modelling complex patterns in the data. Our approach is based out of Recurrent Neural Networks (RNN) as information from essays can be encoded in an effective manner and the system can learn the patterns effectively through non-linear layers.

2. Related Works

A lot of organisations have tried building an AES system after extending one of the first models which was the Project Essay Guide (PEG). AES can essentially be considered as a regression or a classification problem. In regression based methods essay score is considered to be a dependent variable and the features are considered to be independent variables. The features help learn the appropriate weights using which a regression equation is devised via which essays are graded. In a classification based approach the essay scores are divided into different categories and using these labels, future essays are classified and scored. In fact one of the bigger organizations for conducting exams : educational testing services (ETS) have used stepwise regression techniques on vast varieties of features. AES systems have been tackled using intensive hand crafted features and several supervised machine learning algorithms. A lot of effort has been given to manual feature engineering for automated essay scoring. *Chen et al.*[6] used several features with them being as simple as

number of unique words or complex features such as lexical features (spelling errors, word length, word level), syntactical features (sentence length, subclauses, mode, prepositions, commas), grammar and fluency features (word and POS bigrams). The authors have also taken into account text coherence under the content and prompt specific features so as to assess the argumentation and flow of information of an essay. *Zesch et al.*[7] have also incorporated readability features to extract more information. The author also takes into consideration another important dimension in essay writing which is how appropriate is the style by taking into account the relative ratios of POS tags to determine the style used by the writer. The authors have derived a numerically engineered feature for this where they utilise the type-token ratios to check if the vocabulary is rich or not. Incorporating such features enhanced the accuracy of their model.

Taghipour et al.[11] have developed one of the first neural network based approaches to solve AES systems. The authors evaluate several architecture choices such as RNN unit type which could be basic RNN,GRU or LSTM, using a recurrent layer vs a convolutional recurrent layer or using a bidirectional vs unidirectional LSTM. It was found out that the best performing model was using LSTM whereas basic RNN models don't perform as well as a GRU or LSTM. The reason for the same attributed by authors is GRU and LSTM have been shown to remember sequences and long term dependencies effectively when compared to RNN.

In our case we have utilised several neural network architectures which capture the complex features and can solve the problem in an end to end fashion. Several recurrent neural network approaches have been used to a great extent for solving various other problems in NLP. *Cho et al.*[8] have used an attentive neural network approach for machine translation which uses Gated Recurrent Unit developed by *Cho et al.*[9]. The authors have made an extension to the original encoder-decoder model which can learn to align and translate jointly. The authors used 2 types of models: RNN encoder-decoder and RNNsearch where RNN search outperforms the RNNencdec model. Neural network approaches have also been utilised in syntactic parsing. *Vinyals et al.*[10], have used long short-term memory (LSTM) networks for getting parse trees by using a sequence-to-sequence model and constructing the parsing job as a sequence generation problem. As compared to LSTM, GRU uses less training parameters and executes faster as it requires lesser memory. In turn, LSTM gives more accuracy on datasets using longer sequences. We compare and contrast the results of all such models in our work

3. Dataset

The data was provided by William and Flora Hewlett Foundation (Hewlett) while sponsoring the Automated Student Assessment Prize (ASAP). The data contains ASCII formatted text for each essay followed by one or more human scores, and (where necessary) a final resolved human score. Essays have on an average length between 150 to 500 words, mostly written by students between grade 7 to grade 10. In order to preserve anonymity and protect privacy, the original dataset was included with Name Entity Recognizers from

Stanford's NLP group. The dataset was split into 80% training data and 20% validation data. There are 8 sets of essays corresponding to different topics where each training example contains an essay_id which serves as a unique identifier for each student, essay - which contains the ASCII text response of the student, essay_set denoting id for an essay set, rater1_score, raater2_score, rater3_score and a resolved domain1_score showing a consolidating score of the essay. There are domain2_scores as well but they are available for a few data points thus are ignored for all purposes.

4. System Design

The following section explains the system design with which we have experimented and also explains our best performing model

4.1 Feature Extraction

Since Neural networks are unable to understand words or strings and require a numeric representation, we had to convert each word into a numeric vector. The test set is then transformed using the same representations with unknown words getting ignored in calculating the final document vector. However for dynamic embedding technique, we replace the unknown words with a common OOV token. The word vector models which we tried are as follows

4.1.1 TF-IDF

TF-IDF (Term frequency - inverse document frequency) is a numerical means of representing a word based on its importance in a document in a corpora. The tf-idf algorithm is the product of the term frequency and the inverse document frequency of a word.

We have calculated the term frequency by the below formulae

$$tf_{t,d} = log_{10}(count(t,d) + 1)$$

1 has been added in the count to account for the terms which do not occur in a document. Inverse document frequency has been calculated as follows

$$idf_t = log(N/df_t)$$

where N is the collection of documents and df is the number of documents in thich the term t occurs. The tf-idf weight of a word is then calculated as follows

$$tf - idf_{t,d} = tf_{t,d} * idf_t$$

After calculating the tf-idf vectors for all words occurring in an essay, we then represent the essay itself as a document vector by computing the centroid of all the vectors

$$d = \frac{w_1 + w_2 + ... + w_n}{n}$$

Where d is the document vector for an essay and w1,w2...wn are the words occurring in an essay.

TF-IDF vector has been computed using sklearn's TFIDFVectorizer which takes the essay corpora as input and tokenizes it using CountVectorizer and generates the weight vectors.

4.1.2 Pre-trained Word2Vec (GloVe)

GloVe is an unsupervised learning algorithm for obtaining word vector representation of words. We have used the pre-trained word vectors which are provided by Stanford. These word vectors have been trained on the Wikipedia dataset and have around 400,000 Vocabulary words with dimensions ranging from 50 to 300. In our

experiments we have trained our model using 300 dimension vectors for the words as they gave the best results. The document vector has been then calculated in a similar fashion by taking an average of all the word vectors. This document vector has been then used in our model. The pretrained vectors are fetched and converted to a proper representation using Gensim library.

4.1.3 Trained Word2Vec

We have also used the word2vec representation of the words by training the CBOW algorithm on the training dataset. The Word2Vec implementation is provided by Gensim which implements the Word2Vec family of algorithms using highly optimized C routines. The model has been trained by using 10 words as the context window and includes all the words that occur in the training corpora. The final document vector is then calculated by averaging out the word2vec representations of the words occurring in the essay. This document vector is then fed into our model

4.1.4 Dynamic Word embeddings

In this approach we will basically train the word embeddings as a part of our neural network architecture. Suppose each word vector is of the form

$$w = WE$$

Where E is the embedding matrix and W is the one hot encoding version (Bag of words) representation of the word. One hot encoding of the vector is done by sklearn's tokenizer. After that using Keras pad_sequences we would need to make all the document vectors of the same length where the document vector would be represented by the one hot encoding of each word vector.

The embedding matrix will be trained and updated while we are training our neural network model using backpropagation.

We initialize the embedding matrix with the word vectors we had obtained when we were training our own Word2Vec model using the essays as the training corpora. By doing this we are giving our network a better starting point than assigning it as a sparse matrix or a matrix of random integers. Embedding layer provided by Keras has been used for this purpose. These word embeddings will be updated as the model gets trained

4.2 Experiments

This section would explain the experiments conducted using the word vector representations mentioned in the above section along with different neural network models. The best performing model has been explained in detail in the next section.

4.2.1 TF-IDF Vector representation with 2 - LSTM layers

In this model we used the TF-IDF model for representing words as vectors as described in section 4.1.1. The first layer of the neural network model consisted of a Long Short Term Memory layer (LSTM) which used the TF-IDF vectors as input. The second layer consisted of another LSTM layer which used the outputs from the previous layer as inputs. The outputs generated from this layer were then fed into a dense neural network which was the final layer. An additional dropout layer of Keras was added

for preventing overfitting. This dense network used Relu (Rectified Linear unit) as the activation function. Tensorflow's Keras library was used to implement the neural network architecture.

4.2.2 Pre trained Word2Vec model with 2 - LSTM layers

This is the same architecture as above with word vectors being represented as the ones described in section 4.1.2.

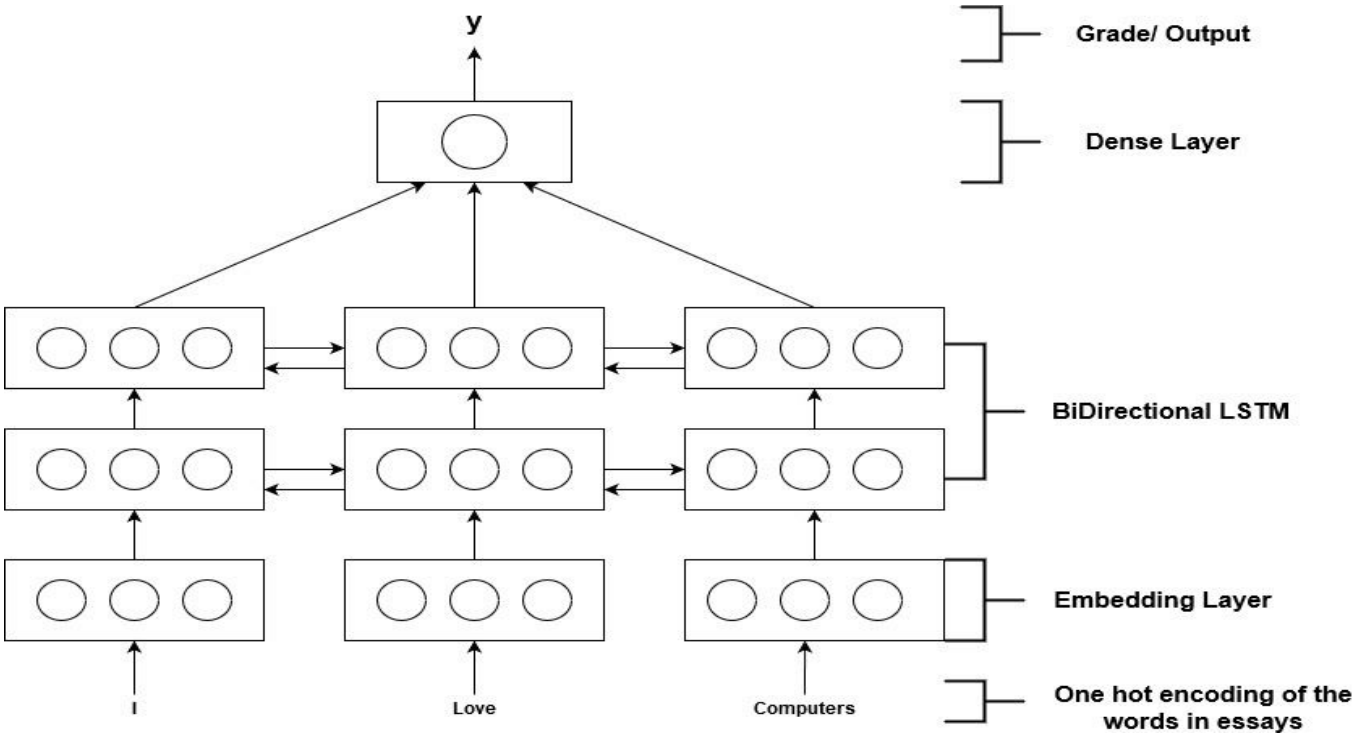


Figure 1 : Final Neural Network Model implemented Bi-LSTM with Self trained embeddings

4.2.3 Pre trained Word2Vec model with 2 - Bidirectional Simple RNN layers

The architecture of the model is the same as defined in section 4.2.1. Instead of using LSTM we have used simple recurrent neural networks(RNN) which uses the pre-trained Word2Vec vectors as input. There is however one more change in this model. We have used the Bidirectional property which is provided by Keras so that for a word, we can also use the results/outputs obtained from the future words as well as past words for prediction. The final layer is a dense neural network which uses the outputs generated by the Bidirectional RNN layers for generating the output by using Relu as the activation function.

4.2.4 Pre trained Word2Vec model with 2 - Bidirectional GRU layers

In this architecture we have used pre-trained word2vec vectors along with Gated Recurrent Unit (GRU). The architecture is implemented in the same way as 4.2.3 where we have introduced bidirectionality in GRU layers as well.

4.2.5 Pre-trained Word2Vec model with 2- Bidirectional LSTM layers

In this architecture we have used pre-trained word2vec vectors along with LSTM. The architecture is implemented in the same way as 4.2.3 where we have introduced bidirectionality in LSTM layers as well.

4.2.6 Trained Word2Vec model with 2- Bidirectional LSTM layers

In this architecture we have used trained word2vec vectors as described in section 4.1.3 along with LSTM. The architecture is implemented in the same way as 4.2.5 where we have introduced bidirectionality in LSTM layers as well.

4.2.7 Dynamic word embedding model with 2- Bidirectional LSTM layers

In this architecture we have used a dynamic word embedding model as described in section 4.1.4 along with LSTM. The architecture is implemented in the same way as 4.2.5 where we have introduced bidirectionality in LSTM layers as well. This was our best performing model and will be explained in detail in the next section

4.3 Model Architecture

Figure 1 shows the neural network architecture which performed best with the training dataset and gave the best score according to the evaluation metric that we chose.

The bottom most layer is the word embedding layer which was implemented as described in section 4.1.4. 300 was chosen as the embedding dimension so that we could initialize the embedding matrix with our vectors obtained by training the Word2Vec model.

The embedding layer provided inputs to the Bidirectional LSTM layer. This layer was configured with 300 neurons for the input,update and forget gates. The recurrent dropout which is used to mask the inputs for adding some regularization to prevent overfitting was set to 0.4.

The output from the previous layer was then used as an input to the next Bidirectional LSTM layer which was configured with 64 neurons and 0.4 as the recurrent dropout layer.

A Dropout layer was then added which was initialized to 0.3 so as to prevent overfitting on the training data.

The final layer was a dense neural network layer which had just the one neuron and had the ReLU as the activation function since we are performing a regression task.

Squared loss was used as the loss function and Adam optimization technique was used. Adam optimization technique performed better than RMSEProp in our case.

Hyper parameter tuning was performed by wrapping the entire model using KerasClassifier and then by using GridSearchCV of sklearn to tune the model. The configurations used are listed below.

Hyper parameter	Hidden Layer 1	Hidden Layer 2	Output Layer
Neurons	300	64	1
Recurrent dropout	0.3	0.3	-

5. Evaluation

5.1 Evaluation Metrics

The Evaluation metric chosen was quadratic weighted Kappa score. The quadratic weighted kappa's score gives the measure of agreement between two raters ranging from 0 (Complete disagreement between two graders) and 1(Complete agreement). This evaluation metric was chosen since it was the metric chosen by Kaggle to measure the accuracy of the model.

5.2 Evaluation results

The evaluation was conducted by splitting the training set with 80% as the training data and 20% as the test data. This data set was then shuffled randomly and then an average was taken for all the QWK values obtained by running the model on each set of data. Basically a 5 fold validation testing was done to measure the performance. The results on the official test data could not be calculated since the human labeled data was not available. However really bad essays were manually graded to check whether they matched the model's grading

Model	Kappa Score
TF-IDF + 2 - LSTM	0.2301
Pre trained Word2Vec model with 2 - LSTM layers	0.96807
Pretrained Word2Vec model with 2 - Bidirectional Simple RNN layers	0.9536
Pre trained Word2Vec model with 2 - Bidirectional GRU layers	0.96402
Pre-trained Word2Vec model with 2- Bidirectional LSTM layers	0.96831
Trained Word2Vec model with 2- Bidirectional LSTM layers	0.9631
Dynamic word embedding model with 2- Bidirectional LSTM layers	0.9726

5.3 Visualisation Dashboard

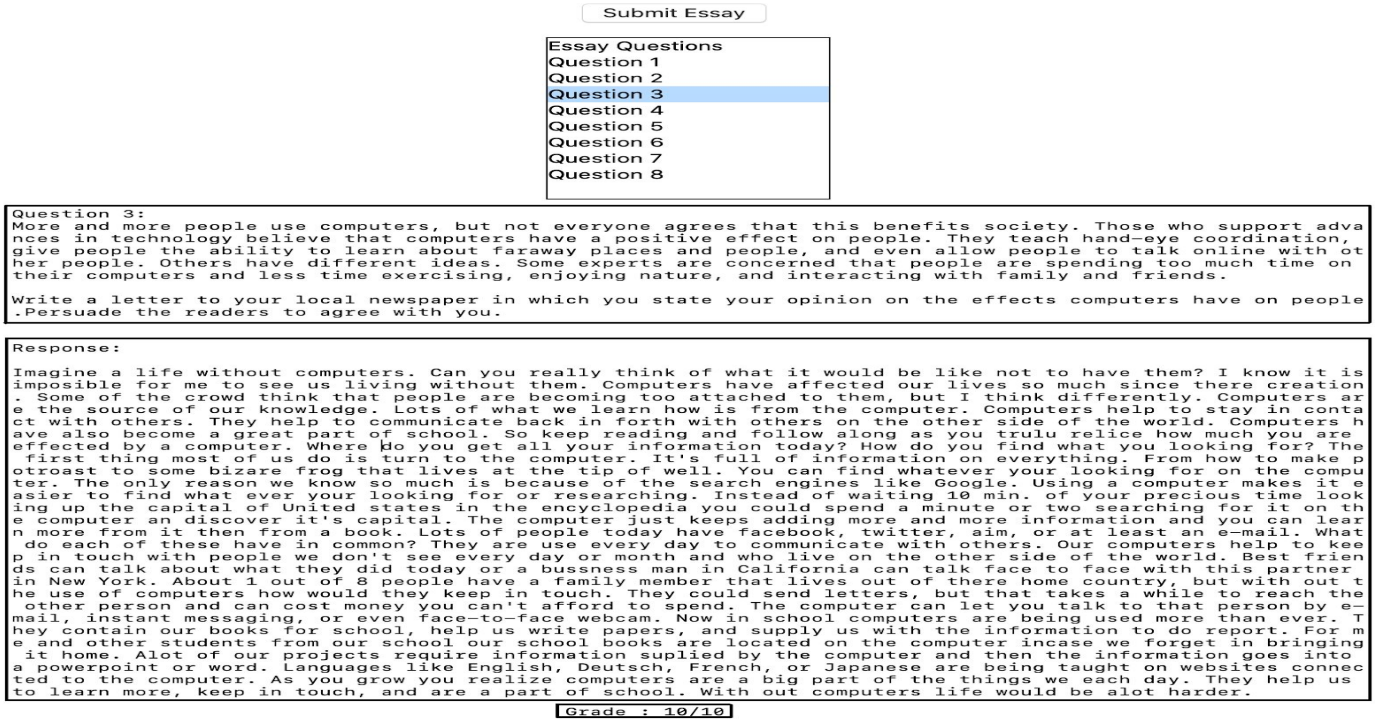
A visualisation dashboard has been created where the user can pick the question set and submit the essay. When the user clicks the submit button, our model will evaluate the essay. Below there are two examples of a strongly graded essay and a weakly graded essay. Clearly the second essay

written has been graded as poor one as it lacks several rich features which one might incorrectly spelled words, very less unique words, has a short length, a weak set of vocabulary with several repetitions of phrases and sentiments. As compared to this, the first essay seems cogent and well

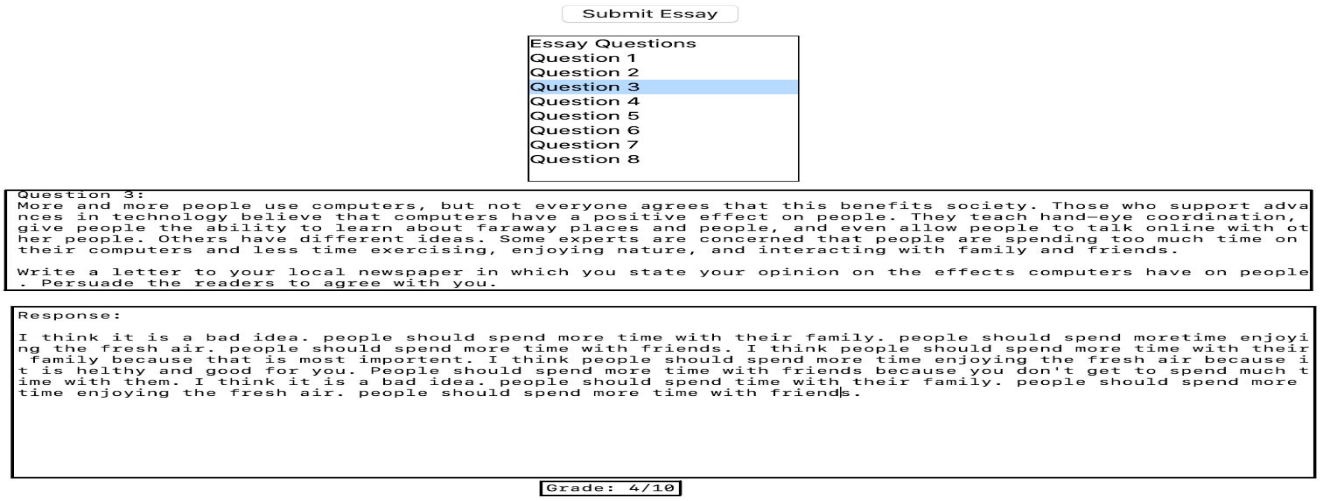
look at while grading an essay. The essay includes

written with a clear direction of the topic in mind. Both the essays here have been recorded for the same question which asks if computers are boon or bane to the society.

Response 1: High graded essay



Response2: Weakly graded Essay



6. Conclusion

From the evaluation results we can see that we got the best performing model by using a Bidirectional LSTM layer with dynamic word embeddings. Overall we can see that by using a bidirectional layer we get a higher score from the unidirectional layers because linguistically some sentences tend to have dependence on future words rather than past words and it is not possible to capture this dependence by using a unidirectional layer which captures the dependence on the past results. Moreover, we can see that the LSTM layer performs better because of its property of retaining useful information from previous results. Dynamic word embeddings perform better because instead of having a static representation, the word vectors have a dynamic representation where they are getting updated through back propagation. The embedding matrix keeps on adjusting its weight based in order to minimize the overall loss. By keeping the initial Embedding matrix as the trained Word2Vec representation we are giving the network a good starting point.

7. Future Work

In order to test the accuracy of the model, this system can be extended in form of a web portal where students could write an essay and it would be graded by multiple professors. The accuracy of the system could then be tested properly. This can further be completely automated where our model would be responsible for grading the essays.

We can also use BERT instead of using Word2Vec as the initial starting point for our self trained word embeddings.

In order to improve the model's performance, we can try incorporating a Convolutional layer in our network. We can also further experiment using Gated Recurrent Neural Network [8.5] and compare it with other models.

8. Challenges

The foremost challenge we faced was that since training a neural network requires a high GPU support it couldn't be done on our local machine so we used Google Collab's free version but that had its own limitation such as low RAM and low GPU which resulted in hyperparameter tuning over a smaller set and longer running times. If we had more computing power we could have done better hyperparameter tuning and increased the number of K fold cross validations.

The test set for this competition isn't publicly available hence we performed a 5 fold cross validation to gauge our results. Had the test set been available we could have had a better reflection as to how well our model is working.

9. References

[1] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation

[2] Cho, Kyunghyun; van Merrienboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua (2014). "Learning Phrase Representations using RNN

Encoder-Decoder for Statistical Machine Translation

[3] Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". *Neural Computation*.

[4]Mikolov, Tomas; et al. (2013). "Efficient Estimation of Word Representations in Vector Space"

[5] Wang, J., & Hu, X. (2017). Gated Recurrent Convolutional Neural Network for OCR. In *Advances in Neural Information Processing Systems* (pp. 334–343).

[6] 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.*

[7] 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.*

[8] 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.*

[9] 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.*

[10] 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.*

[11] Kaveh Taghipour, Hwee Tou Ng. 2016 A Neural Approach to Automated Essay Scoring *Conference on Empirical Methods in Natural Language Processing (EMNLP 2016)*