Automated Essay Grading Using Neural Networks

Revathi Bhuvaneswari
rbhuvaneswari@fordham.edu

Tianying Luo
tluo9@fordham.edu

Abstract

Our paper presents a way to automatically grade essay papers through the mean of different Neural Network architectures. Different structures such as Multi-Layer, Convolutional-Layer and Recurrent-Layer are explored in accurately predicting essay scores using regression approach. Special emphasis is placed on pre-processing and corpus handling techniques from Natural Language Processing domain. Several representations of essay vectors in the form of word embeddings are explored. The Mean Quadratic Cohen Kappa score is used as the primary metric to evaluate the performance of our models.

1 Introduction

Essays have been considered an effective medium for measuring academic metrics compared to generalized multiple-choice tests. However, the cost and effort associated with scoring the essays have made instructors prefer the multiple-choice path. Automating the essay grading process would not only help overcome these obstacles but would also aid with removing subjectivity and biases that might sometimes occur with human graders.

Automated Essay Scoring (AES) has been a continuous domain of research that began as early as the 1960s when the first essay scoring system called Project Essay Grade (PEG) was published by Ellise Batten Page [9]. True developments began to resurface in the early 2010s, a time-period when Machine Learning algorithms were beginning to gain popularity. In 2012, the Hewlett Foundation sponsored a Kaggle Competition called the 'Automated Student Assessment Price' (ASAP) for the grading of high school students' essays. Our paper utilizes this dataset for models.

Several works have been published on this very dataset the last few years using a few different modeling approaches. While the majority of the early papers experimented with Machine Learning algorithms such as Support Vector Machines and Linear Regression, like [7], the recent years have seen a rise in the usage of Neural Networks for the task. As presented in papers such as [5], [6] and [8], Convolutional and Recurrent based Neural Networks produce significantly better results compared to Machine Learning approaches. The papers also experimented with different embedding models such as word2vec, doc2vec and C&W embeddings [6].

In our project, we aim to utilize NLP techniques along with different types of Neural Networks to develop a model that would effectively predict the score of a given essay. We employ a regression approach for the prediction of these scores, and experiment with different pre-processing methodologies to produce optimal results. We believe that this application can be further expanded for automated essay generation, and even into other domains where there is a need for distinguishing between a 'good' and 'bad' document or entity.

2 Dataset

We will be using the Automated Essay Scoring dataset developed by the Hewlett Foundation for the Kaggle competition. The dataset contains 8 different essay sets, with each set containing essays written by students belonging to Grade 7 through Grade 10. Each of these sets also has a unique domain/topic and utilizes different scoring systems. The main feature set consists of the essay corpus and their corresponding scores provided by a number of human graders. The dataset provides training (~13K essays), validation (~4K essays) and testing (~4K essays) split. The validation and test split lacked true essay scores, and the training set was further split into train/validation/test splits for our model training. For our project, we used 10% for validation and 10% training of the provided training dataset. The statistics of the train split is described below.

Table 1. Statistics of Dataset

Essay Set	Grade Level	Scores Range	Avg Words	Avg Sentences	Total Essays
1	8	2 - 12	366	22	1783
2	10	1 - 6	381	21	1800
3	10	0 - 3	109	7	1726
4	10	0 - 3	94	6	1772
5	8	0 - 4	122	8	1805
6	10	0 - 4	153	9	1800
7	7	2 - 24	168	12	1569
8	10	10 - 60	605	35	723

3 Approach

While analyzing the distribution of essay scores, it was noted that the number of available essays for a few score values was less, because of which a regression-based approach was taken, as a classification method would have led to problems with class imbalance. The preprocessing of essay corpus along with scores as well as representation of essays and evaluation metrics are discussed below. Four main types of Neural Network models were experimented with for this project, whose implementation details and results are included in the next section.

3.1 Preprocessing

Similar to many other NLP tasks, we started the cleaning of our essay corpus by converting everything into lowercase characters. Additionally, the original dataset employed Stanford's Named Entity Recognition to replace identifying information such as Name, Location, Date etc. with tags such as @NAME1, @LOCATION3, @DATE5 etc. The tags were replaced with *label_name*, *label_location* and *label_date* for example in order to remove duplicate information along with number information. Each essay in the dataset was then tokenized into a single list with words belonging to the particular essay. This was done in two ways - retaining stop words and removing all stop words. All punctuation marks along with any numeric characters were replaced with an empty character for both list types of tokenized essays.

As mentioned in the above section, each essay set has a unique essay score distribution. Due to the regression approach we've taken, it is ideal to scale the scores within a certain range, as it'll help with our model training, especially when choosing activation functions. All scores were scaled to a range [0, 1] using the MinMaxScaler, which will help with Sigmoid activation function that will be used in the output layer of our models. It should be noted that this scaling is done per essay set rather than the entire dataset as a whole, as a particular score value might carry different weightage for different sets. For example, a score of 11 in essay set 1 represents a 'high' score while the same value in essay set 8 represents a 'low' score. The scaled scores are used as target during model training and the predictions are rescaled back to original set range for evaluation.

3.2 Essay Corpus Representation

The pre-processed and tokenized essay corpus for each essay is used to create embedding representation for model training. These embeddings are created through custom word2vec training on the essay corpus using the gensim package. Models of {50, 100, 200, 300, 500} dimensions were created, with both stopwords present and absent, however during model training it was noted that the embeddings without stopwords performed better. This embedding model would help in spite of grammatical and spelling errors present in the corpus as well since it is customized to our dataset. The embeddings of essays were used in three ways during the training.

The first method was taking an average of the embedding values for each word in the essay corpus to produce a vector of desired dimension. For example, if an essay has 100 words in total, and we are using a 200 dimension embedding model, for each word of the essay the 200 dimensioned embedding vector is first found. Later, this embedding matrix of size (100, 200) is averaged for each column, producing a single vector of 200 dimension for the essay. This method helps produce same dimensioned vector for all essays in the corpus.

The next two methods utilize the Keras Tokenizer API and the Embedding Layer. Each token in the essay corpus is tokenized to an integer value, and the essay sequences are padded with zeros to resize all vectors to the maximum essay's length. This is then passed to an Embedding Layer in our models to produce appropriate vector embeddings for training. The Embedding Layer will be initialized with the custom embedding matrix created from our essay corpus. Two different paths are taken - the Embedding layer is either trained along with the model weights to improve our own custom embeddings, or the outputs are directly used as input for our model without training.

3.3 Loss Function and Evaluation Metrics

As we've taken a regression approach for this task, we used the L2 or MSE loss function during the training of our model, which calculates the squared differences between truth and predictions.

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The y_true values refer to the scaled essay scores while y_pred values refer to sigmoid prediction of the models. We train the model to optimize this loss and keep it as low as possible.

In order to evaluate the performance of our models, we used the Mean Quadratic Cohen Kappa Score that was originally used in the Kaggle competition. This metric helps measure the level of 'agreement' between two raters for the essay scores, with a value ranging between [-1, 1].

If two raters agree completing in scoring the essays, producing the same scores, then the Kappa score would be 1. If metric produces a value of 0, then there is random agreement between the raters. The metric may result in a value less than 0 if there is less agreement between the raters than expected by change. In the below equation, *i* refers to rater_1 and *j* refer to rater_2. The quadratic weights are calculated based on *k*, which is the number of values possible. For our models, this score is calculated between true essay scores and rescaled whole number prediction essay scores, which can be roughly translated into individual 'class' of scores for this metric.

$$\kappa = 1 - \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} w_{ij}x_{ij}}{\sum_{i=1}^{k}\sum_{j=1}^{k} w_{ij}m_{ij}} \qquad \text{Quadratic: } w_i = 1 - \frac{i^2}{(k-1)^2}$$

While the top leaderboard score was 0.81407, it was calculated on the original test dataset that has not been released, and the models were trained on the entire train dataset and validated on provided validation set which has also not been released. As we don't have the same data available for evaluation, we decided to calculate the Kappa score between two human raters from the dataset and use that as our baseline/goal. This value turned out to be 0.7544, which will be used as a performance goal for all our models.

4 Experiments

Automatic scoring is thought of a regression problem in our experiments. While it can also be regarded as a classification problem, for which case the target variable should be one-hot encoded to the range of unique grades with the same dimension of output layer in a neural network, it performs better with regression model. For classification, the results will be approximated through softmax activation and then again being min-max reversed; however, for regression, the results are only approximated or rounded after scaling.

4.1 Multi-Layer Perceptron

Multi-Layer Perceptron model is first utilized to perform the autograding task. It takes the feature embedding as the input layer and then takes a scalar value as the output layer. The architecture of our MLP model is described below. It has one convolutional layer, activated by relu function, and weights initialized by he_normal; after that, dropout layer will be added, then the embeddings will be fed into the dense layer and produce the predicted score.

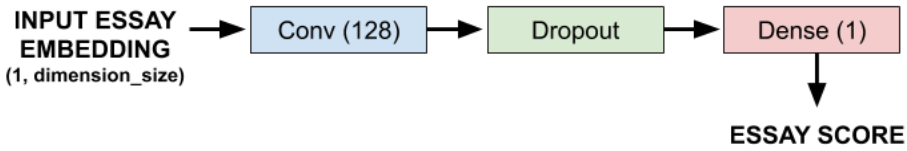

Figure 1. Architecture of MLP Model

Formally, an MLP is comprised of three layers, an input layer, a hidden layer and an output layer. The hidden layer can be defined as follows:

$$h_i = s(x_i * W_x + b_x)$$

s is a nonlinear function such as sigmoid or tanh. *W* is the weight parameter matrix for the input and *b* is the bias for the hidden layer. Through the model training, the weight matrices will be continually optimized and fed into the output layer. The output layer can be defined as follows:

$$o_i = G(h_i * W_o + b_o)$$

G is also a nonlinear function that can be in the form of a softmax for the case of multi-class classification. Since we regard regression as part of our experiment, instead we'll be implementing sigmoid activation function to generate the value in the range of [0, 1]. In order to train MLP, we used different optimization algorithm like Stochastic Gradient Descent, RMSprop and Adam. Different dimensions of feature vectors per essay are experimented and dimension of 300 produced the highest kappa scores.

Table 2. Performance Statistics of MLP Models

Parameters: Batch Size = 128, Epochs = 100, Dropout = 0.2, Dimension = 300	
Optimization Algorithm	**Test Kappa Score**
Stochastic Gradient Descent	0.4338
RMSprop	0.6268
Adam	0.6388

For feedforward neural networks, the learning process doesn't consider the sequences or the order in time. In essay grading, the sequence of words provide substantially relevant information. Thus, we improve the model by implementing Recurrent Neural Network architectures, which are good at capturing a series of events.

4.2 Convolutional Neural Network (CNN)

While CNN's have been extensively used for image data in the past, recently there has been a rise in using this model for text-related problems. CNN can help preserve spatial information of textual data by applying filters in all possible 'areas' of the input sequence, which in turn helps learn the sequence information in an effective manner. This is especially useful for our essay scoring problem as the relationship between the words and the overall structure plays a crucial role in determining the optimal score.

Our CNN structure uses Convolutional 1D layers (since the data is 2 dimensional) along with MaxPooling layers to capture spatial and hierarchical information found in the input sequence. In the end, a Global MaxPooling is performed in order to feed to dense layers and get the predicted essay score. The model also has Dropout layers to help prevent overfitting and better generalize for unseen data. ReLU activation along with He weights initialization were used, which the exception of the output layer that used the Sigmoid activation function. The architecture of our CNN model is described below.

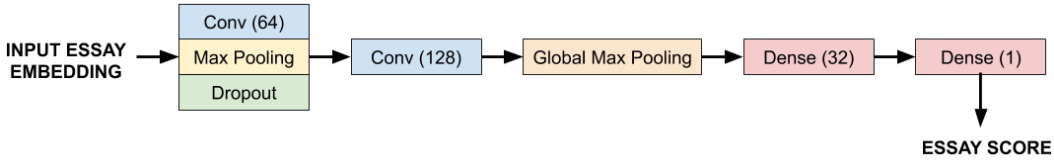

Figure 2. Architecture of CNN Model

The CNN model was experimented with three different essay embedding formats:

(1) Original Word2Vec essay embedding averaged per word fed as input directly
(2) Integer tokenized essays by Tokenizer API fed into Embedding Layer before Conv Layer (Embedding Layer's weights initialized with original Word2Vec essay embeddings):
    (a) Not Trained along with CNN model weights
    (b) Trained along with CNN model weights

The embedding dimensions or sequence lengths were also experimented with from {50, 100, 200, 300, 500} along with different dropout rates. All experiments used Adam optimizer with a default learning rate of 0.001 and a batch size of 256 for 100 epochs. It was found that a dropout of 0.2 gave better results overall for all three embedding formats with an embedding dimension of 100.

The embeddings trained along with CNN model weights gave the least favorable Quadratic Cohen Kappa score while the same integer tokenized essay words retaining original Word2Vec essay weights gave the best results. The approach of averaging word2vec essay embedding values per word and feeding it directly to the Conv. layer without an Embedding layer gave moderate scores.

Table 3. Performance Statistics of CNN Models

Parameters: **Batch Size = 256, Epochs = 100, Adam(lr = 0.001), Dropout = 0.2, Dimension = 100**	
Embedding Format	**Test Kappa Score**
Embedding Layer (Trained Weights)	0.5882
Averaged Word2Vec Essay Vectors per Word	0.6237
Embedding Layer (Word2Vec Essay Weights)	0.6808

4.3 Long - Short Term Memory (LSTM)

Each essay will be regarded as a sequence of tokens and a Long-Short Term Memory network (LSTM) will be utilized to map essays into lower-dimensional embeddings, which will be then fed into a dense layer to generate the predicted score. Unlike vanilla Recurrent Neural Network, LSTM has been proven to be effectively used for embedding long sequences without dealing with gradient descent problem. The encoding process and internal mechanism are described as below:

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot s_t + U_i \cdot h_{t-1} + b_i) \\
f_t &= \sigma(W_f \cdot s_t + U_f \cdot h_{t-1} + b_f) \\
\tilde{c}_t &= \sigma(W_c \cdot s_t + U_c \cdot h_{t-1} + b_c) \\
c_t &= i_t \circ \tilde{c}_t + f_t \circ c_{t-1} \\
o_t &= \sigma(W_o \cdot s_t + U_o \cdot h_{t-1} + b_o) \\
h_t &= o_t \circ tanh(c_t)
\end{aligned}
$$

The internal mechanisms are called gates in LSTM which can regulate the flow of information. All different gates can learn which data in the sequence should be kept or aborted. By doing this, LSTM can pass on only the relevant information down the sequence to make predictions. In detail, for each essay, $e = \{s^1, s^2, ..., s^m\}$, where s indicates the t^{th} sentence embedding in the essay. In the above equations, ht means the hidden state of sentence st. W and U indicate the weight matrices for the input gate, forget gate, candidate state, and output gate respectively. b are the bias vectors for all different gates. σ denotes the sigmoid function and ◦ stands for element-wise multiplication. Thus, for the essay e, we will get the hidden state set $H = \{h^1, h^2, ..., h^m\}$. The hidden state of essay representation will then be fed into a dense layer to convert the vector into a scalar value. The output of the dense layer will be projected into scaled value within the range between 0 and 1. Sigmoid activation function will be utilized and defined as below:

$$S_e = sigmoid(w_s \cdot h_m + b_s)$$

Se indicates the semantic score of essay e. w means the weighted matrix of the dense layer and b stands for the bias. The objective for loss function is the Mean Squared Error (MSE) loss function.

The architecture of LSTM is comprised of two layers: the input layer takes the feature vectors for each sentence tokens in the essay by fixed number of dimensions, then the LSTM layer will be added to map the feature vectors into lower-dimensional embeddings, after that a dense layer is utilized to turn the embeddings into scalar value in the range of (0, 1).

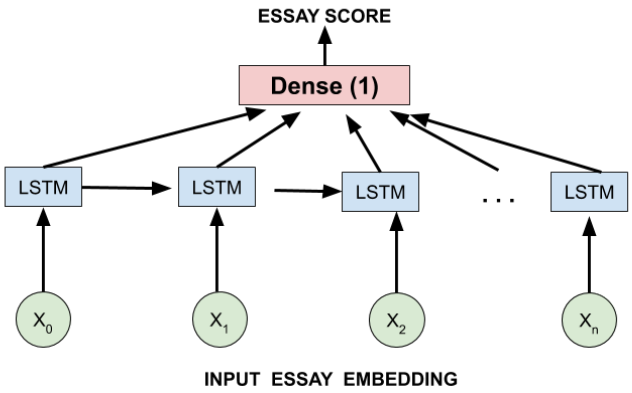

Figure 3. Architecture of LSTM Model

The hyperparameters of the LSTM model are the size and the weights of the LSTM layer, and the dropout rate. Different dimensions of the word vectors were also experimented with.

Table 4. Performance Statistics of LSTM Models

Parameters: Batch Size = 128, Epochs = 100, Dimension = 100	
Dropout Rate	**Test Kappa Score**
0.1	0.5992
0.2	0.6053
0.3	0.5881
0.4	0.5976
0.5	0.5871

4.4 Bidirectional Long Short Term Memory (Bi-LSTM)

The Bidirectional LSTM model helps better preserve information during training and improve the quality of learned information compared to the traditional LSTM models. In traditional LSTM models, the sequence relations are learned in a front-to-back approach where the sequence is traversed only in one direction during the training process. As a result, the model would only learn the 'past' relationships, which could become problematic if the sequence length is large.

In contrast, the Bidirectional LSTM reads the input sequence in both front-to-back and back-to-front directions, thus preserving both 'past' and 'future' relationships of the sequence. This also aids during each timestep by preventing loss of information towards the end of a sequence iteration. The model mainly achieves this by implementing 2 LSTM layers, with one traversing the sequence in the forward direction, and the other traversing it in the backward direction. The output of a particular Bidirectional layer is the merged results of each individual LSTM layer. This merging can be done by either concatenating the results (which doubles the output size) or by taking a sum or average of each LSTM results together.

The architecture of our Bidirectional LSTM model comprises of a single LSTM hidden layer with the Bidirectional component wrapped around it, which in turn produces 2 LSTM layers' results concatenated together. This concatenated output is then passed through a dense layer which predicts our essay's score. The architecture of our model is depicted in the below figure.

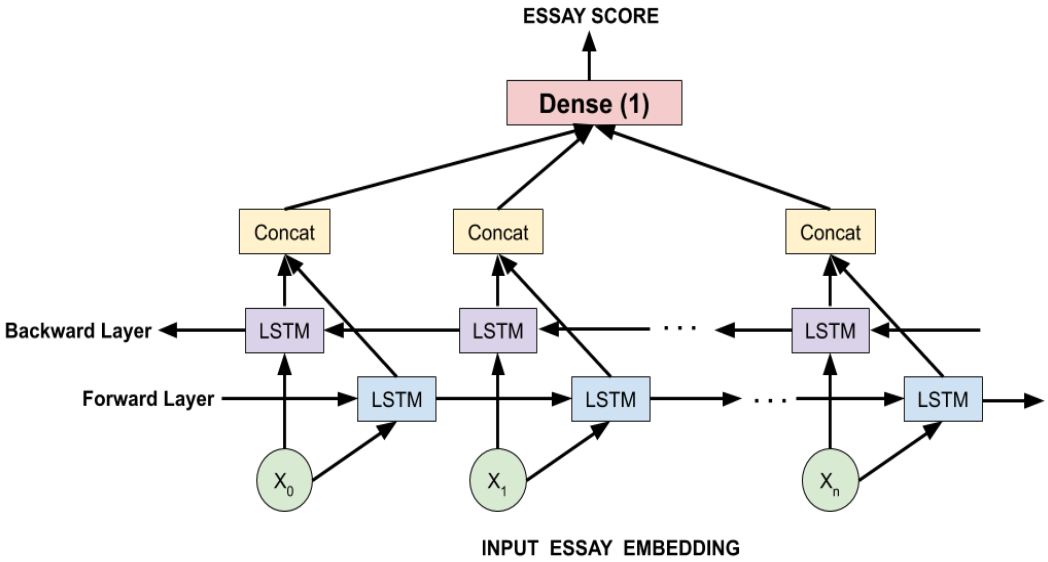

Figure 4. Architecture of Bi-LSTM Model

Similar to the previous models, combinations of different hyperparameters were tuned in order to achieve optimal Quadratic Cohen Kappa score. The embedding formats were first experimented with a dimension of 100 to measure its performance. It was found that in contrast to CNN, here the Averaged essays vectors performed better than using Embedding Layer initialized with our custom essay embeddings. We believe this is because of the increase in the sequence length found in the Embedding layer method to accommodate the longest essay length without losing information. Additionally, padding the vectors with zero led to sparse vectors, which could have hindered with the model training. The below table shows the difference in Kappa scores.

Table 5. Bi-LSTM Embedding Formats Performance Comparison

Parameters: **Batch Size = 256, Epochs = 100, Adam(lr = 0.001),** **Dimension = 100, # Hidden Units = 64**			
Embedding Format	**Train Loss**	**Val Loss**	**Test Kappa Score**
Embedding Layer (Word2Vec Essay weights)	0.0081	0.0268	0.5299
Averaged Word2Vec Essay Vectors per Word	0.0309	0.036	0.5466

It was also noted that the Embedding Layer led to severe overfitting compared to the Average vector method as shown in the below figure. Although there are some fluctuations in the validation loss performance for the Average vector method, the decreasing trend of training and validation loss shows promise of improvement.

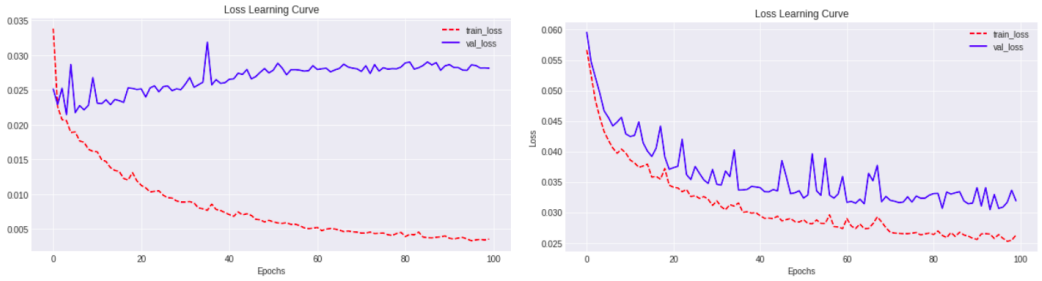

Figure 5. Learning Curve: Left - Embedding Layer method, Right - Average Vector method

After determining the best essay representation format, the next step was to determine optimal parameters for the model. This parameter tuning was done through a 5-Fold Cross Validation technique in order ensure best combination of parameters. After different parameter combinations, it was found that a batch size of 32 with 100 epochs produced the best Kappa score for a dimension vector of 200 with 128 hidden layer units per LSTM layer.

Out of different learning rates that were tried, the default learning rate for Adam optimizer of 0.001 gave the highest Test Kappa score. Additionally, L2 regularization was also tried, however, it didn't give good scores. Though TanH activation is original used in LSTM cells, ReLU activation with He-Normal weight initialization gave better results. The combination of best possible parameters produced a total Test Kappa score of 0.6226, which was a bit disappointing compared to the performance of the CNN. In order to further explore the issue, the model was trained and evaluated on increasing subsets of the essay sets such as {1}, {1, 2}, … {1, …, 8}, whose results are displayed in the table below.

Table 6. Bi-LSTM Performance on Essay Set Combinations

Parameters: **Batch Size = 32, Epochs = 100, Adam(lr = 0.001), Dimension = 200, # Hidden Units = 128**				
Essay Set	**Train Loss**	**Val Loss**	**Test Kappa**	**Rater Kappa**
1	0.0107	0.0148	0.5934	0.721
1, 2	0.0115	0.0144	0.597	0.765
1, 2, 3	0.0179	0.0243	0.5885	0.7681
1, 2, 3, 4	0.0204	0.0279	0.5795	0.7888
1, 2, 3, 4, 5	0.02	0.0304	0.6007	0.7816
1, 2, 3, 4, 5, 6	0.0193	0.0299	0.6658	0.7807
1, 2, 3, 4, 5, 6, 7	0.0191	0.0297	**0.7043**	0.7723
1, 2, 3, 4, 5, 6, 7, 8	0.0185	0.0283	0.6226	0.7544

It can be noted from the above table that the Test Kappa score has fluctuations in results as each new essay subset is added to the training. While it starts off slow, the measure of Test Kappa score starts to increase with essay subsets of [1, 5]. This trend continues till essay subsets of [1, 7], after which there is a sudden huge drop in the Kappa score performance. It can also be seen that the

same trend follows in the actual human raters' Kappa score as well. We believe this difference in performance is mainly because of the different grading scale used in the essay sets. The essay set 8 has the least amount of essays with a huge scale difference of [10, 60] compared to other sets. Using only a subset of the provided essay sets could have been an approach worth trying during our parameter tuning for achieving optimal results in our models.

5 Results

The below table depicts the comprehensive best performances of all our models for the entire essay training set utilizing all subsets together. It can be seen that while the CNN produces the best Mean Quadratic Cohen Kappa score of 0.6808, which is the closest to the human raters' Kappa score of 0.7544, the Recurrent Neural Networks such as LSTM and Bi-LSTM didn't produce optimal results. One of the main challenges was feeding 'appropriate' information to the model during training time. The LSTM models, which has a complex architecture compared to CNN and MLP models, requires properly scaled score distributions along with extensive parameter tuning to produce desired Kappa score on par which human raters, and it shows promise of improvement.

Table 7. Performance Statistics of All Models (Best for All Essay Sets Together)

Model	Test Kappa Score
Multi-Layer Perceptron (MLP)	0.6388
Convolutional Neural Network (CNN)	0.6808
Long-Short Term Memory (LSTM)	0.6053
Bidirectional Long-Short Term Memory (Bi-LSTM)	0.6226

6 Conclusion and Future Work

In this paper, we introduced Multi-Layer Perceptron, CNN, LSTM and Bi-LSTM neural network models to tackle the automatic essay grading task. Each model's internal architecture has been explored and utilized to prove that neural network architectures have great potential in solving natural language processing problems. All the models outperformed the baseline models after parameter tuning by a large margin and our best model achieved score almost closer to human rater's Kappa scores, whereas our work still need lots of improvement.

More complex models like two-layer LSTM and Bi-LSTM are worthwhile being experimented with, and score-specific word embeddings which not only are able to capture contextual meaning but also reflect each word's contribution to the whole essay, would help improve the model performance. Also, in addition to Cohen Kappa Score, other effective correlation coefficients like Spearman's ρ and Pearson's r could be used for helpful reference.

The essay corpus contained numerous spelling and grammatical errors that produced the same words with different spellings in our embedding models. Correcting these errors and applying stemming/lemmatization could be a worthwhile area to explore. Additionally, custom features such as word count, word to sentence ratio, number of mistakes etc. could be engineered and combined with our word embedding features to produce more accurate results with essay scoring.

In the world of natural language process, exploring the network's internal scoring and rating criteria is one of the main worthwhile topics to contribute more effort in. Our work can be further expanded to automated essay generation task once the model learns to accurately distinguish between 'good' and 'bad' essay structures. Additionally, our work can be implemented beyond just the academic domain. There are several real-world problems in the Finance, Commerce,

Marketing and other industries where there is a need to solve the problem of distinguishing between a ‘good’ and ‘bad’ entity while also determining the properties of a ‘good’ entity. The models presented in our paper can be further exploited to give interpretable and useful results to the information receivers in various such industries.

References

[1] The Hewlett Foundation: Automated Essay Scoring. https://www.kaggle.com/c/asap-aes/data

[2] Brownlee, J. (2019, August 14). How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras.

https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

[3] Chawla, R. (2018, July 23). Tuning a LSTM to reduce variance on a Yelp Dataset for Sentiment Classification.
https://medium.com/ml2vec/training-and-tuning-a-lstm-to-classify-yelp-reviews-4d37b8aa2e91

[4] LSTM Neural Network Training - Few Useful Techniques for Tuning Hyperparameters and Saving Time. (2018, April 20).
http://intelligentonlinetools.com/blog/2018/04/17/lstm-neural-network-training-techniques-tuning-hyperparameters/

[5] Liang, G., On, B.-W., Jeong, D., Kim, H.-C., & Choi, G. S. (2018, December 1). Automated Essay Scoring: A Siamese Bidirectional LSTM Neural Network Architecture. https://doi.org/10.3390/sym10120682

[6] Alikaniotis, Dimitrios, Yannakoudakis, Helen, Rei, & Marek. (2016, June 16). Automatic Text Scoring Using Neural Networks. https://arxiv.org/abs/1606.04289v2

[7] Song, S., & Zhao, J. (2013). Automated Essay Scoring Using Machine Learning.
https://nlp.stanford.edu/courses/cs224n/2013/reports/song.pdf

[8] Taghipour, K., & Ng, H. T. (n.d.). A Neural Approach to Automated Essay Scoring.
https://www.aclweb.org/anthology/D16-1193.pdf

[9] Automated essay scoring. (2019, November 6). https://en.wikipedia.org/wiki/Automated_essay_scoring