Automated Essay Feedback

Noah Arthurs
B.S. Candidate: Computer Science
narthurs@cs.stanford.edu

Sawyer Birnbaum
B.A.S. Candidate: Computer and Political Science
sawyerb@cs.stanford.edu

Abstract

We have set out to create a system that gives essay feedback by identifying stronger and weaker parts of a given essay. In order to do this, we have created a two step model: the first step uses an LSTM to turn a chunk of an essay into an encoding vector. The second step uses a feed forward network to assign a score to an encoding. We believe that if the feed forward network is able to predict the score of an essay based on the average of the encodings of the chunks, then it should be able to distinguish between the stronger and weaker chunks as well. We have achieved an RMSE of 0.17 on the task of predicting the score of an essay. While this is not comparable to human grading, it is good enough to be able to determine when there is significant variation in quality between different parts of the same essay.

1 Introduction

Automated essay grading is a well explored problem with practical applications for standardized testing companies. However, the task of giving more fine-grained feedback to an essay than just a grade is much less explored. A system that can give targeted feedback to different parts of an essay would be able to help students improve their writing and prepare for these standardized tests. We have taken first steps towards creating such a system.

We are using a dataset of 13,000 graded student essays provided by the Hewlett Foundation on Kaggle [1]. The essays are between 150 and 550 words, written by 7th-10th grade students across 8 different prompts. This is a standard dataset for the automated essay grading problem, but because it only gives the final grade for each essay, it is a challenge to turn this information into more fine-grained feedback.

We know that the essay grading problem is approachable on this dataset, so in order to ensure that our task would be feasible, we have defined our problem so that it would not be too far off from essay grading: instead of giving a grade to the whole essay, we divide the essay into chunks and give a grade to each chunk. This allows us to give feedback in the form of identifying the weaker and stronger parts of an essay.

2 Background

Our dataset was originally published on Kaggle in 2012 for an automated essay scoring competition, but it continues to be used as a standard dataset for the problem. Recent efforts such as Alikaniotis et al. [2] and Taghipour et al. [3] have shown that multilayer bidirectional LSTM's are quite capable of grading the essays. In order to bootstrap our hyperparameter search, we started with an LSTM structure based on these past results.

Liu et al. [4] used SVM's to develop an essay feedback system that assigns scores to an essay like "Spelling", "Supporting Ideas", "Spelling", etc. While this is more specific than a grade, it does not give feedback to specific parts of an essay.

3 Approach

3.1 Preprocessing

First we downloaded 300 dimensional GloVe vectors [5] and used the set of tokens associated with those vectors as our corpus.

We normalized all essay scores to be between 0 and 1. We then preprocessed the essays themselves in four steps:

1. Using NLTK [6] we parse each essay into sentences.
2. Using the Stanford Parser [7] we tokenize each sentence.
3. Using PyEnchant [8] we correct each word that is not in our corpus into the closest word that is in our corpus.

   NOTE: despite the fact that spelling errors could be indicative of essay quality, we decided to correct them so that not all semantic meaning would be lost when a word is misspelled.
4. Finally we translate the tokens into indexes in our embeddings matrix. In order to avoid overfitting, we assigned each word that appeared in fewer than three essays to an UNKNOWN vector that we initialized to 0's. Furthermore, because the dataset masks off with keywords proper nouns and a few other types of words, we added vectors of 0's for these keywords.

At first our chunks were individual sentences, but in later tests we experimented with different methods of chunking essays (see section 4.4).

3.2 Encoding

We used TensorFlow [9] to implement our model, the first step of which involves encoding the chunks of an essay into vectors.

In order to encode a chunk, we look up its word vectors and feed them into a multilayer bidirectional LSTM [10]. In order to capture both the forward input and the backward input, we concatenate the first and last outputs of the BLSTM to create our chunk encoding.

3.3 Predicting

In order to turn a set of chunk encodings into a predicted essay score, we first average the encodings to get an encoding for the whole essay. We then push this average encoding through a fully connected feed-forward neural network. The hidden layers of the network use ReLU activation functions, and the network ends with a sigmoid layer in order to output a predicted score between 0 and 1.

For example, if we are using a single hidden layer, this is how we would make our predictions for an essay with encodings E_1 through E_n:

$$E = \frac{1}{n}\sum_{i=1}^{n} E_i \tag{1}$$

$$H = ReLU(EW_h + b_h) \tag{2}$$

$$Pred = \sigma(HW_o + b_o) \tag{3}$$

Where W_h is $encoding\_size \times hidden\_size$ and W_o is $hidden\_size \times 1$.

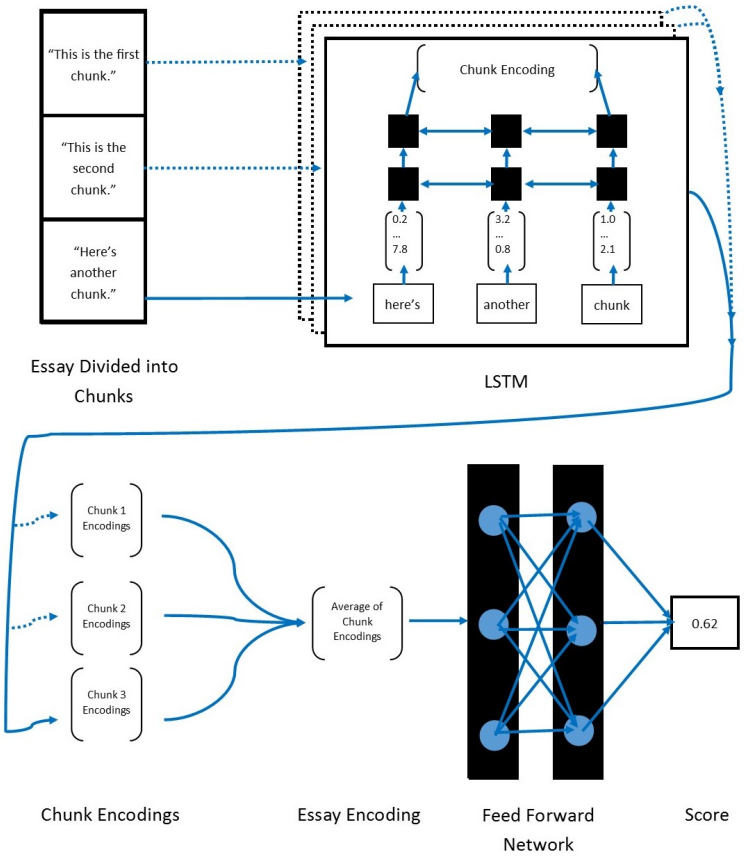

Figure 1:
This figure demonstrates how the model takes in the chunks of an essay, feeds them through the LSTM in order to create chunk encodings, and predicts a score based on the average of those encodings. Once trained, the feed-forward network is used to assign scores to the individual chunk encodings (see section 3.5).

3.4 Training

Our main error metric is the mean squared error between the predicted scores and the actual scores:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(Pred[i] - Score[i]) \tag{4}$$

Where m is the number of essays in the set. We train using the sum of MSE and a regularization term as our loss, using TensorFlow's built in Adam Optimizer [11] to propagate the error signal all the way to the word embeddings. The regularization term is the product of a regularization constant and the sum of the L2 norms of all of the non-bias variables (including the word embeddings).

We train in batches of 100 essays, pausing to evaluate our weights after every 30 batches.

3.5 Feedback

Because we create the essay encoding by averaging the chunk encodings, the prediction network does not take the relative positions of the chunks into account. This means that we should be able to assign a score to each chunk by pushing it through the same prediction network. By doing this, we are having the network answer the question, "How good would this essay be if every part of it were like this chunk?"

During training, we only have error signals coming from the overall essay score, but we make the assumption that the network trained to predict the essay score based on the average of the encodings will be able to tell us the relative strengths of the individual chunks. This way, we can train a model that assigns scores to chunks using only the total score of each essay.

4 Experiments

4.1 The hyperparameters

Here are our hyperparameters along with their initial values for the hyperparameter search. The ones below the dashed line were tuned during the hyperparameter search. The others could use more exploration:

Name	Description	Initial Value
Vec Size	Size of GloVe vectors used	300
Learning Rate	Learning rate for the Adam Optimizer	0.001
Reg Constant	Multiplier for the L2 Norms during regularization	10^{-6}
Keep Prob	Probability of keeping a unit during dropout	0.5
Min Frequency	# of essays a word must be in to have its own word vector	3
State Size	Size of the LSTM hidden state	200
Bidirectional	Whether or not the LSTM is bidirectional	False
FF Layers	Number of layers in the feed-forward network	1
Chunk Size	Number of sentences per chunk	1
LSTM Layers	Number of layers in the LSTM	2
Num Hidden	Number of hidden units per feed forward layer	200

4.2 Initial Tests

Initially our model was overfitting substantially despite using dropout, at which point we added in regularization and the minimum frequency constraint during preprocessing (see 3.1). After adding these precautions, there was still some overfitting, but the dev RMSE improved significantly. Below is a plot of the training vs. dev RMSE for training with the initial parameters specified in section 4.1 and the parameters of our final model. Note that training RMSE was calculated by sampling randomly from the training set, so the measurements may be noisy:

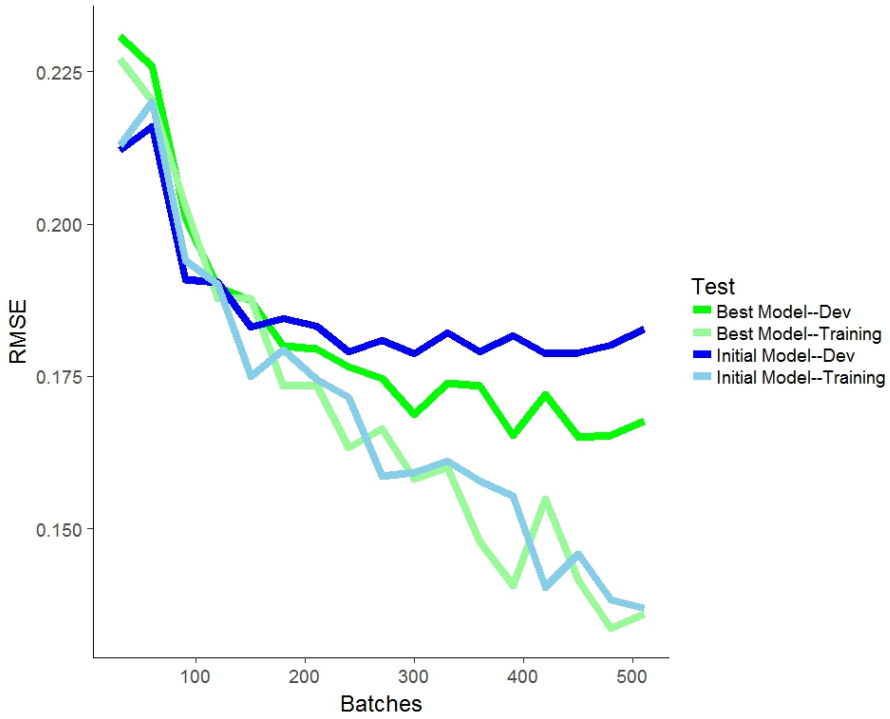

Figure 2: Compares training and dev RMSE on the initial model and the best model found (see 4.3).

4.3 Hyperparameter Search

We performed a hyperparameter search by scoring each set of hyperparameters with the best dev RMSE achieved. The parameters that are being tested are in bold:

Bidirectional	FF Layers	Chunk Size	State Size	LSTM Layers	Num Hidden	RMSE
False	1	1	200	2	200	0.1815
True	1	1	200	2	200	0.1772
True	**3**	1	200	2	200	0.1767
True	3	**2**	200	2	200	0.1740
True	3	2	**400**	2	200	0.1845
True	3	**4**	200	2	200	**0.1650**
True	3	**8**	200	2	200	0.1786
True	3	4	**400**	**3**	**400**	0.1662

Surprisingly, ramping up the power of the network did not make a big difference. The two most important parameters to change were Bidirectional and Chunk Size. It is worth noting that the larger chunk size means that our feedback is less fine-grained, but we believe that this is a sacrifice worth the lower RMSE. The test third from the bottom is still our best model, with a test set RMSE of 0.171.

4.4 Chunking by Wordcount

Because chunk size made such a big difference in the previous test, we decided to explore chunking further. One thing we noticed was that our sentence parser was relatively inconsistent, which meant that the chunks of four sentences had wildly different lengths. In addition, our feedback model was giving much lower scores to shorter chunks than it was giving to longer chunks.

As a result, we investigated chunking based on a fixed number of words rather than numbers of sentences, with the hope that chunks of consistent lengths would be easier for the model to learn from. Here are our results with the rest of the hyperparameters the same as in our best test from the hyperparameter search:

Words per Chunk	Best Dev RMSE
32	0.1713
64	0.1698
100	0.1735

The average chunk of 4 sentences was 64 words long, so it makes sense that the 64 word test performed the best. However, this method of chunking is clearly not as effective. There are two reasons this might be the case:

1. Chunking based on word counts disregards semantic boundaries, so the model may be losing the ability to take sentence and phrase structure into account.
2. When chunking by sentences, the model was deriving information from the sentence lengths (giving lower scores to shorter chunks), which is now lost.

Although this experiment failed, we are still a little bit suspicious of our sentence parser, and we believe that a better parser might improve our model's ability to learn from the data.

4.5 Further Experiments

Another potential problem is the fact that we may not be working with a large enough dataset to effectively train the word embeddings, especially since we are training a handful of the embeddings from scratch. We experimented with training our best model while keeping the embeddings constant, but this resulted in an RMSE of 0.1788, so clearly training the embeddings is helpful. Again, this still may be an issue in our model, and it might be worth exploring other solutions.

We also wondered whether a different optimizer would make a difference. Alikaniotis et al. used an RMSProp [12] Optimizer. Using this optimizer, our model learned too slowly given the time constraints that we have, even with a much larger learning rate than Alikaniotis's model, which

required 55 hours to train. It is possible that we would have achieved better performance given more time with this optimizer.

5 Results

5.1 Compared to the State of the Art

In order to compare our best model directly to Alakiniotos et al.'s best model, we measured our Spearman's ρ and Pearson r on the test set:

Model	Spearman's ρ	Pearson r
Arthurs et Birnbaum	0.6719	0.6931
Alakiniotos et al.	0.91	0.96

We perform about as well at the task of essay grading as Alakiniotos's less powerful models. However, we expect to do worse because of the fact that we are feeding chunks of essays into our LSTM rather than whole essays.

5.2 Giving Feedback

In order to get a sense for the capacity of our model to provide feedback, we calculated the difference between the highest and lowest scoring chunks in each essay. Here is a histogram of those differences:

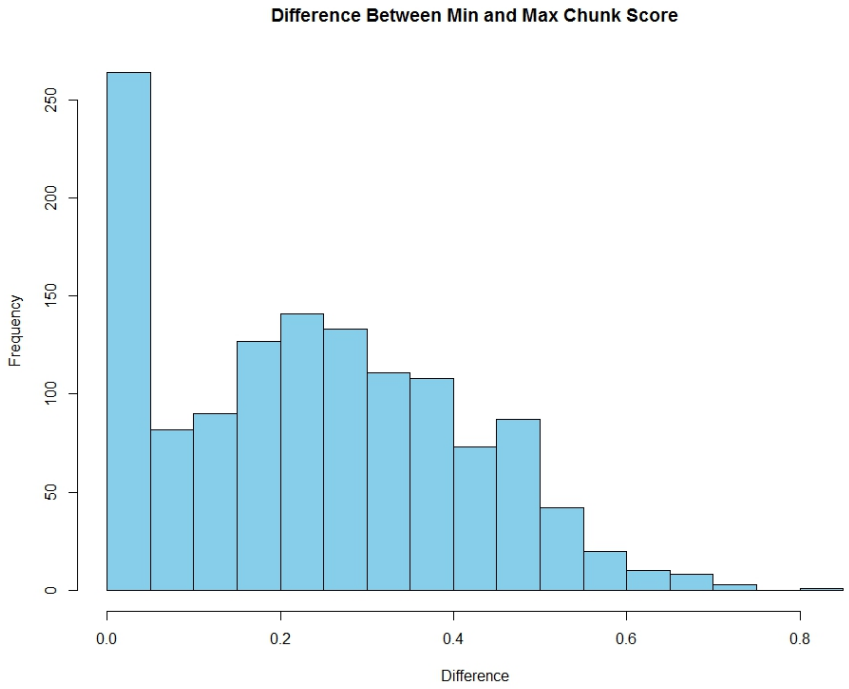

Figure 3: The distribution of differences between the highest and lowest scoring chunks in each essay.

In order to give feedback, we make the assumption that chunks with very different scores are likely to be of very different quality. Since our RMSE is 0.17, we would like two chunks to have scores more than 0.34 apart in order to feel confident that one is better than the other. This means that we can only give what we consider reliable feedback to a little under half of the essays.

5.3 Feedback Examples

In the examples we have looked at where there is a big (around 0.7) difference between the scores of two chunks, there is usually a clear difference in quality between the chunks.

In one essay by an 8th grade student analyzing the mood of a short story, the following chunk was given a score of 0.92:

"The family are migrants and they came looking for a better lifestyle in a foreign country just for the benefit of the children. From my point of view, only loving parents would give up the country they all love in order to do something like that because they had to leave behind families, friends, and careers. now they're facing cultural hardships over just one sedition. In this memoir by Narciso Rodriguez the mood is very warm and inviting."

while the following chunk from later in the essay receives a score of 0.25 (note that symbols starting with @ are the dataset's keywords discussed in section 3.1):

"In the author's memoir the author creates a mood and that mood in the story as happyness enjoy. It is happyness and @caps1 because he was happy for his home and how his parents were cooks. He also enjoyed the music and things he does with his family and relatives."

Here we can see that the thoughts presented in the higher scoring chunk are much more clear. In the lower scoring sentence, it is possible that the spell checking has hurt our young writer, but overall what is being said is repetitive and poorly articulated.

However, we can still find examples where it is not so clear why a chunk was given the score that it was. Here is a chunk from another 8th grade student writing about whether computers are good for society. Our model gave it a score of 3.93:

"My experience with commuters is good because I can play games or do my homework. I can install my own music on my commuter and do my homework at the same time. I can also go on the internet with commuters. On commuters, I can also do things quicker and easier."

while the following chunk from later in the essay receives a score of 8.00:

"Dial up connection is what I can use to get free internet by putting a person's phone number in the place were the phone number goes the laptop will call up the mote if the phone line is connected to the mote. On my laptop I also have a place to put a floppy disc in and a place for a @caps1 - @caps2 to go in too. I also have a calculator on my laptop. Some laptops also have a mouse that looks like a little pad like mine does."

Here the second chunk has a score that is much too high, given how rambling and incoherent it is. So our model clearly could be improved.

6 Conclusions

Our approach shows promise when it comes to distinguishing between stronger and weaker parts of essays. We believe that driving the RMSE down further would allow us to provide more accurate feedback.

Our current model is limited by the fact that it must average the encodings of the chunks in order to create the essay encoding, which loses structural information about the essay. Perhaps a model that incorporated that information would perform better, but there would then be the question about how to score parts of the essay.

Alakiotos et al. used a method that they called "Score Specific Word Embeddings" in order to pay more attention to certain words in the input. This method greatly improved their model, and perhaps implementing something similar could improve ours.

Because the results of our hyperparameter search were so close together, we believe that the next steps should be improving our preprocessing step. Better spell-checking and division into sentences

could give our model more accurate information to learn from. Beyond that, we would want to experiment with the hyperparameters that we did not have time to tune rigorously: namely, the optimizer, the learning rate, the regularization constant, and the regularization method.

7 References

[1] Kaggle Dataset: https://www.kaggle.com/c/asap-aes

[2] Alikaniotis, Dimitrios, Helen Yannakoudakis, and Marek Rei. "Automatic text scoring using neural networks." arXiv preprint arXiv:1606.04289 (2016).

[3] Taghipour, Kaveh, and Hwee Tou Ng. "A Neural Approach to Automated Essay Scoring."

[4] Liu, Ming, et al. "Automated Essay Feedback Generation and Its Impact in the Revision." IEEE Transactions on Learning Technologies (2016).

[5] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." EMNLP. Vol. 14. 2014.

[6] Bird, Steven. "NLTK: the natural language toolkit." Proceedings of the COLING/ACL on Interactive presentation sessions. Association for Computational Linguistics, 2006.

[7] Stanford Parser: https://nlp.stanford.edu/software/lex-parser.shtml

[8] PyEnchant: http://www.pythonhosted.org/pyenchant/

[9] TensorFlow: https://www.tensorflow.org/

[10] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[11] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[12] Dauphin, Yann, Harm de Vries, and Yoshua Bengio. "Equilibrated adaptive learning rates for non-convex optimization." Advances in Neural Information Processing Systems. 2015.

8 Contributions

Both teammates were present and consulted one another for all of the substantial work done on the project. Noah wrote the code for training/running the model and the text for the report. Sawyer wrote all of the preprocessing/postprocessing code and created the graphs/figures for the report.