CSE 143: Assignment 2

Jacob Doar jjbagins@ucsc.edu
Daniel Xiong dxiong5@ucsc.edu
Edward Christenson edfchris@ucsc.edu

Due Feb. 19, 2020

1 Introduction

We explored the field of sentiment analysis using text classification techniques such as RNNs and GRUs. Each model was trained on the IMDB review database provided by Keras and tested with a variety of hyper-parameters.

2 Tools Used

We used Keras 2.3.0 with TensorFlow 2.1.0 on Python 3.7

3 Data Processing

We loaded three data sets from the IMDB review database: training, validation, and testing sets. We then create a vocabulary of the 10000 most frequent words and batch them with a batch size of 64. These are then encoded into integers to be fed into our model.

4 Text Classification with Simple RNNs

For this part of the assignment, we built a RNN-based text classifier using a Keras SimpleRNN layer. Our model is made up of an Embedding layer, followed by a SimpleRNN layer, with a final Dense (Fully-Connected) layer acting as the output layer.

We tested four different sets of hyper-parameters, where each set is labeled as $\theta_{1\rightarrow 4}$. Each test had a batch size of 64 and used the Adam optimizer.

	Epochs	Activation Function	Dropout	Training Acc.	Validation/Dev Acc.	Test Acc.
θ_1	10	ReLU	0.50	0.9063	0.5023	0.5000
θ_2	15	ReLU	0.25	0.9833	0.5048	0.4942
θ_3	15	tanh	0.25	0.9911	0.4951	0.5002
θ_4	10	tanh	0.75	0.5074	0.5022	0.4971

Hyper-parameters $\theta_{1\to3}$ resulted in a high training accuracy and a very low test accuracy. One probable reason θ_4 had a low training accuracy was because it had a relatively high dropout rate. The best performing hyper-parameter set on the training data was θ_3, with an accuracy of 0.9911. Each θ performed about the same on the test data, albeit θ_3 wins by technicality; θ_3 had a test accuracy of 0.5002.

5 Text Classification using LSTM/GRUs

5.1 GRU Model

We chose to use a GRU for this part of the assignment. In our model, we replaced the SimpleRNN layer with a single GRU layer. Overall the GRU model performed better than the simple-RNN model did, with θ_1 having the best test accuracy of 0.5261.

	Epochs	Activation Function	Dropout	Training Acc.	Validation/Dev Acc.	Test Acc.
θ_1	10	ReLU	0.50	0.9824	0.5094	0.5261
θ_2	15	ReLU	0.25	0.9958	0.5047	0.5206
θ_3	15	tanh	0.25	0.9955	0.5134	0.5255
θ_4	10	tanh	0.75	0.9417	0.5097	0.5234

With using a GRU model, we expected to have better accuracy over using the simple-RNN model. Our test results show that overall the test accuracy is slightly better using the GRU model. In all instances, using the same hyper-parameters as in the simple-RNN model tests, we have better training accuracy as well. The only time the simple-RNN model outperformed the GRU model in validation/Development accuracy was for θ_3, where the GRU scored .5047 and the simple-RNN scored .5048. Overall the GRU model had better accuracy when compared to the simple-RNN using the same hyper-parameters.

5.2 GRU Experimentation

By using the same hyper-parameters as we did with the simple-RNN model, we were able to compare the two models performances. While the GRU model we reported consisted of only a single layer GRU, we did experiment with more than one GRU layer to see how the accuracy would differ. For our best run with the single GRU layer model, θ_1 returned a test accuracy of 0.5261. When we ran these same hyper-parameters on a double GRU layered model the test accuracy was 0.5253. Not only was the test accuracy worse for this model, but both training and development accuracy were also lower. Similar results were observed by a three layer GRU model. Thus we decided that the single layer GRU model was the best to report as it gave us the best accuracy from our experimentation.

	Training Acc.	Validation/Dev Acc.	Test Acc.
2 GRU layers	0.9711	0.5075	0.5253
3 GRU layers	0.9779	0.5082	0.5208

6 Pretrained Word Embeddings

6.1 Text Classification using Pretrained Word Embeddings

For previous models, a 300-dimensional word2vec embedding layer was learned during training. In order to make use of pre-trained word embeddings we chose to use a model provided by TensorFlow Hub. This model is a token based text encoder that has been trained on English Google News with a corpus of 200B and can be found at https://tfhub.dev/google/tf2-preview/nnlm-en-dim128/1.

Once again, we tested four different set of hyper-parameters. This model was made up of the the pretrained embedding layer, two GRU layers and a single dense layer. Each test had a batch size of 64 and used the Adam optimizer.

	Epochs	Activation Function	Dropout	Training Acc.	Validation Acc.	Test Acc.
θ_1	10	Sigmoid	0.25	0.7008	0.6984	0.7008
θ_2	10	ReLU	0.25	0.6068	0.6902	0.6902
θ_3	15	ReLU	0.5	0.6986	0.7012	0.6968
θ_4	10	Sigmoid	0.1	0.7008	0.6984	0.7008

Universally, the use of pretrained embeddings resulted in higher test accuracy than any of the previous models, but the training accuracy was lower than the majority of the previous models. We predict that this is the case because our previous models were able to train the word embeddings on the given vocabulary which led to overfitting on the training data and poor results on the actual test data. Since the embeddings used were the same for all data sets this gave more consistent results across the training, validation, and test sets.

6.2 Word Embedding Experimentation

A common occurrence in pretrained embeddings is that words that are known to be antonyms have similar embeddings. This occurs because similar embeddings between two words does not necessarily mean that the words are similar, it might simply mean that there is some type of relationship between them(in this case them being antonyms is their relationship).

We chose the word "good", because that would be expected to be commonly found in sentiment texts, and compared its embedding to a synonym, an antonym, and a randomly chosen word from the vocabulary. These comparisons were made using the cosine similarity metric. For our tests we chose the words "great", "bad", and "dash" which had cosine similarity scores of 0.7894, 0.6647, and 0.0074 respectively. Our conclusion is that the synonym and antonym both have relationships to "good" so they are expected to exist in similar areas of vector space and thus have scores that are closer to 1(signifying a smaller angle between the vectors). Whereas the randomly chosen word has no obvious relationship to "good" and is expected to exist in a different region of the vector space, this is shown by the fact that its cosine similarity score is farther away from 1(signifying a larger angle between the vectors).

When our best performing model, θ_1, was run on the test review, "This movie was really good! I enjoyed it a lot and I want to see it again." and a copy of the review, but with the word "good" replaced by its antonym, "bad", it gave prediction scores of 0.5604 and 0.8304 respectively(a positive prediction for both reviews). So swapping the words did not change the prediction, in fact the classifier gave an even more confident score to the review containing the antonym.