Assignment 2 CSE 143: Natural Language Processing

Henry Nguyen

hnguye87

Abstract

The assignment is to build text classifiers for the goal of sentiment analysis. The sentiment analysis is going to see if the text has a positive or negative sentiment. The dataset used in this assignment is going to be the IMDb reviews dataset (IMBb; Maas et al., 2011). The assignment will be built on python language and there will be provided code from the book [Geron, 2019].

Problem

The assignment is broken down into three required parts with an optional fourth. The first part is to implement a Keras model which performs the following:

1. Embed the input text as a sequence of vectors.
2. Transform the sequence of embeddings into a vector using a single-layer, simple RNN.
3. Apply a feed-forward layer on that vector to obtain a label.

The code provided from the textbook satisfies one and three, but has no simple RNN layer. The starter code also had no development set to optimize hyperparameters, so I split 20% of the training set to be the development set. The hyperparameters I chose to work with were optimization method, learning rate, embed size, dropout rate, and number of epochs. I tested it in that specific order.

The second part is to hopefully improve the model by replacing RNN with long short-term memories (LSTMs) or gated recurrent units (GRUs). Based on the reading, LSTMs would perform better on longer sequences that are found in text classifiers like the one for this assignment due to having more training parameters. The downside of more training parameters is that it will take longer to train though, so for ease of testing I chose GRUs.

The last part is to work with pre trained word embeddings as opposed to preprocessing the data manually. For this part I chose to make a seperate file to run the pre trained data for ease of running and reading the program. The two pre-trained sets I used were both from TensorFlow Hub, but varying in size dramatically.

Part 1 Programming: Text Classification with RNNs (35%)

1.1

Parameter 1: Optimization Method

Optimization Method	SGD	RMSProp	Adagrad	Adam	Adamax
Accuracy (Training)	0.5012	0.6152	0.5120	0.5021	0.5569
Accuracy (dev)	0.5002	0.6181	0.4986	0.5001	0.5480

Conclusion: Chose RMSProp because it was highest in both Training and Dev.

Parameter 2: Learning Rate

Learning Rate	0.0005	0.001	0.002	0.004	0.008
Accuracy (Training)	0.9449	0.6152	0.5614	0.5016	0.5062
Accuracy (Dev)	0.6165	0.6181	0.5027	0.5017	0.5031

Conclusion: Picked 0.0005 since the accuracy in the training set was much higher than the others.

Parameter 3: Embed Size

Batch Size	128	256	512	1024	2048
Accuracy (Training)	0.9011	0.9449	0.9482	0.9484	0.9486
Accuracy (Dev)	0.5049	0.6165	0.5518	0.6132	0.6023

Conclusion: Picked 256 because it is the best in the dev set while stilling having a high accuracy in the training set.

Parameter 4: Dropout Rate

Dropout Rate	0.02	0.05	0.1	0.2	0.4
Accuracy (Training)	0.9084	0.9144	0.9419	0.9039	0.8578
Accuracy (Dev)	0.5417	0.5475	0.5756	0.5974	0.5943

Conclusion: Picked 0.2 because it performed best in both dev without dropping training too much.

Parameter 5: Number of Training Epochs

# of Epochs	2	5	10	20
Accuracy (Training)	0.8530	0.9419	0.9926	0.9993
Accuracy (Dev)	0.4934	0.6156	0.5933	0.5734

Conclusion: Choose 5 Epochs since it had the best results in dev.

1.2 Final Hyperparameter values:

Epochs = 20, optimization method = RMSProp,
learning rate = 0.0005, dropout rate = 0.2, embed size = 256

Training Data Accuracy: 0.9993
Dev Data Accuracy: 0.6156
Test Data Accuracy: 0.6031

Part 2 Experimentation: Recurrent Units (30%)

The program implemented GRUs in place in simpleRNN layers.

Parameter 1: Optimization Method

Optimization Method	SGD	RMSProp	Adagrad	Adam	Adamax
Accuracy (Training)	0.7511	0.8527	0.8034	0.8227	0.8323
Accuracy (dev)	0.6023	0.7875	0.7145	0.7275	0.7254

Conclusion: Chose RMSProp because it was highest in both Training and Dev.

Parameter 2: Learning Rate

Learning Rate	0.0005	0.001	0.002	0.004
Accuracy (Training)	0.8527	0.8656	0.9063	0.9387
Accuracy (Dev)	0.7875	0.7650	0.7608	0.7321

Conclusion: Picked 0.0005 since the accuracy in the training set was highest in dev.

Parameter 3: Embed Size

Batch Size	128	256	512	1024

Accuracy (Training)	0.8522	0.8527	0.8668	0.8692
Accuracy (Dev)	0.7804	0.7875	0.7818	0.7820

Conclusion: Picked 256 because it is the best in the dev set while stilling having a high accuracy in the training set.

Parameter 4: Dropout Rate

Dropout Rate	0.05	0.1	0.2	0.4
Accuracy (Training)	0.8523	0.8685	0.8527	0.8483
Accuracy (Dev)	0.7853	0.7863	0.7875	0.7848

Conclusion: Picked 0.2 because it performed best in dev without dropping training too much.

Parameter 5: Number of Training Epochs

# of Epochs	2	5	10	20
Accuracy (Training)	0.8034	0.8527	0.9034	0.9634
Accuracy (Dev)	0.6934	0.7875	0.7534	0.7023

Conclusion: Picked 5 epochs since it performed best on dev set.

I tested the hyperparameters again with the GRUs. The values of the hyperparameters chosen in part 1 still seem to be the most optimal with the GRU layers too, so they stayed the same.

2.1 Final Accuracies with GRU layers:

Training Data Accuracy: 0.8527
Dev Data Accuracy: 0.7875
Test Data Accuracy: 0.7529

2.2

	Training	Dev	Test
SimpleRNN	0.9993	0.6156	0.6031
GRU	0.8627	0.7875	0.7529

Looking at the differences in accuracy, the GRU did much better with longer sequences than simple RNN.

PART3 3 Experimentation: Pre-trained Word Embeddings (35%)

Set : `glove.6B.100d.txt`
https://nlp.stanford.edu/projects/glove/

3.1 : Used a set with a vocabulary containing 6B tokens and 400k vocab. It has 100 dimensions. This embedding versus the one used in past sections has much more vocabulary and tokens.

3.2

I chose to optimize hyperparameters based on set1 of pretrained words.

Parameter 1: Optimization Method

Optimization Method	SGD	RMSProp	Adagrad	Adam	Adamax
Accuracy (Training)	0.6842	0.8417	0.8123	0.7795	0.8045
Accuracy (dev)	0.7045	0.8910	0.8093	0.8255	0.8911

Conclusion: Chose RMSProp because it was highest in both Training and Dev.

Parameter 2: Learning Rate

Learning Rate	0.0005	0.001	0.002	0.004
Accuracy (Training)	0.8417	0.8817	0.9263	0.9587
Accuracy (Dev)	0.8910	0.8517	0.7642	0.7242

Conclusion: Picked 0.0005 since the accuracy in the dev set was highest.

Parameter 3: Embed Size

Batch Size	128	256	512	1024
Accuracy (Training)	0.8273	0.8417	0.8568	0.8630
Accuracy (Dev)	0.7304	0.8910	0.8823	0.8634

Conclusion: Picked 256 because it is the best in the dev set while stilling having a high accuracy in the training set.

Parameter 4: Dropout Rate

Dropout Rate	0.05	0.1	0.2	0.4

Accuracy (Training)	0.8045	0.8348	0.8417	0.8502
Accuracy (Dev)	0.8326	0.8723	0.8910	0.8834

Conclusion: Picked 0.2 because it performed best in dev without dropping training too much.

Parameter 5: Number of Training Epochs

# of Epochs	2	5	10	20
Accuracy (Training)	0.4923	0.6641	0.7349	0.8417
Accuracy (Dev)	0.5293	0.6605	0.7623	0.8910

Conclusion: Picked 20 epochs. Due to training time already being so long, I decided to not test past 20 epochs even though the model probably would have performed better with more.

I tested the hyperparameters again with the new embeddings. The values of the hyperparameters chosen in part 1 again still seem to be the most optimal, except the number of epochs. A larger number of epochs seem to work better with a larger set, so it went up to 20.

	Training	Dev	Test
Set 1	0.8417	0.8910	0.8705

The difference between using pretrained and the past embedding is huge. The improvement is very big as expected from a much bigger set of vocabulary.

3.3

Word embeddings transform the sequence of embeddings into a vector. The relationship of these vectors will be similar to the relationship of two words linguistically. Embeddings tend to give antonyms similar embeddings because antonyms are very often used in the same context as their counterpart. Take for example these two sentences. “The food is very heavy.” “The food is very light.” The embedding has difficulty distinguishing between light and heaving as they are both used in the same circumstances. For this reason it is also why the embeddings perform good on synonyms and analogies.

3.4

For the anonyms I chose the words “heavy” and “light”. The embeddings were very similar probably due to the similar context they are usually in. In example sentences, it seems the classifiers prediction is very very similar proving the fact that embeddings tend to give antonyms similar embeddings.