Sentiment Analysis of iMDB Review Dataset using Tensorflow Core CSE 143

Tyler Hoang - Assignment 2

February 19, 2020

The purpose of this assignment was to implement sentiment analysis on the iMDB reviews data set. Utilization of Google Colab allowed for use of a higher level GPU for model training and fitting. The data is initially split into 3 different sets: the training set, validation set, and the test set. The training set and the validation set are split 80% and 20%.

1 SimpleRNN Text Classification

The following hyperparameter tuning was performed on an RNN-based text classifier. My Keras model embeds the input text as a sequence of vectors, transforms the sequence into a vector with a SimpleRNN layer, then finally applies a feed-forward layer on the vector to obtain a label.
The batch size hyperparameter was not tuned due to the fact that smaller batch sizes yielded better validation accuracies. These batch sizes caused the model to run for an abysmal amount of time (especially when the value is less than 16). While a batch size of 16 yielded the highest validation accuracy, a batch size of 32 yielded an accuracy that was very close. The default batch size of 32 was used for the following tuning procedure so that runtimes wouldn't exceed 25 minutes.

The first hyperparameter tuned was the epoch number.

Epoch Number	Training Accuracy	Validation Accuracy
5	0.6956	0.5962
3	0.8372	0.5262
20	0.9710	0.5450
10	0.9203	**0.6320**

10 epochs yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the embed size.

Embed Size	Training Accuracy	Validation Accuracy
64	0.9779	0.5474
256	0.9656	0.6092
512	0.9105	0.5472
128	0.9203	**0.6320**

An embed size of 128 yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the choice of nonlinearity.

Activation	Training Accuracy	Validation Accuracy
`relu`	.4985	0.5062
`tanh`	0.6801	0.5292
`softmax`	0.4985	0.5062
softplus	0.6640	**0.5576**

The activation **softplus** yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the choice of optimization.

Optimizer	Training Accuracy	Validation Accuracy
`sgd`	0.6175	0.4936
`nadam`	0.7436	0.5142
`RMSProp`	0.8059	0.5820
adagrad	0.9214	**0.7354**

The optimizer **adagrad** yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the dropout rate.

Dropout Rate	Training Accuracy	Validation Accuracy
`0.5`	0.7394	0.6398
`0.8`	0.6151	0.5160
`0.35`	0.8357	0.6276
0.2	0.8375	**0.7250**

The dropout rate of 0.2 yielded the highest validation accuracy. The following are the final, tuned hyperparameters to be used on the test set.

Hyperparameter	Value
`Epoch Number`	10
`Embed Size`	128
`Choice of Nonlinearity`	SoftPlus
`Choice of Optimization Method`	Adagrad
`Dropout Rate`	0.2

The test set yielded an accuracy of **0.7388**.

2 LSTM-based Model

The SimpleRNN model presents two problems: vanishing gradients and large fluctuations between each iteration. Long short-term memories as recurrent units presents better validation accuracies. The following hyperparameter tuning was performed on an LSTM-based model.

The first hyperparameter tuned was the epoch number.

Epoch Number	Training Accuracy	Validation Accuracy
5	0.9609	0.7364
4	0.9454	0.7468
10	0.9898	0.7526
3	0.9168	**0.7762**

3 epochs (wierdly) yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the embed size.

Embed Size	Training Accuracy	Validation Accuracy
64	0.9085	0.7760
256	0.9276	0.7640
96	0.9093	0.7680
128	0.9168	**0.7762**

An embed size of 128 yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the choice of nonlinearity.

Activation	Training Accuracy	Validation Accuracy
relu	0.8102	0.7614
tanh	0.8396	0.7664
softmax	0.4985	0.5062
sigmoid	0.9085	0.7670
softplus	0.8765	**0.7676**

The activation **softplus** yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the choice of optimization.

Optimizer	Training Accuracy	Validation Accuracy
adam	0.8765	0.7676
sgd	0.4979	0.5094
adagrad	0.7872	0.7596
RMSProp	0.8388	0.7412
nadam	0.8674	**0.7758**

The optimizer **nadam** yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the dropout rate.

Dropout Rate	Training Accuracy	Validation Accuracy
0.2	0.8346	0.7674
0.8	0.8000	0.7776
0.35	0.8521	0.7720
0.5	0.8421	**0.7792**

The dropout rate of 0.5 yielded the highest validation accuracy. The following are the final, tuned hyperparameters to be used on the test set.

With the following tuned hyper-parameters:

Hyperparameter	Value
Epoch Number	3
Embed Size	128
Choice of Nonlinearity	SoftPlus
Choice of Optimization Method	Nadam
Dropout Rate	0.5

The test set yielded an accuracy of **0.75616**.

2.1 Longer Sequences

The LSTM-based model should have better accuracy than the SimpleRNN-based model on longer sequences. Altering the preprocess definition, with the sequence length changed from 300 to 400, the model yielded a test accuracy of **0.7775**, which is the highest accuracy so far.

3 Pretrained Word Embeddings

Learned word embeddings were used for Parts 1 and 2. We will now transition to using pre-trained word embeddings that are already constructed in advance. Ideally, this should simplify the learning of the model since the embedding parameters are fixed beforehand. With utilization of the GloVe 100d pretrained embeddings, the following hyperparameter tuning procedure was as follows:

The first hyperparameter tuned was the epoch number.

Epoch Number	Training Accuracy	Validation Accuracy
5	0.6425	0.6248
4	0.5252	0.5322
10	0.8816	0.5906
7	0.7650	**0.6316**

An epoch number of 7 yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the choice of nonlinearity. Contrasting from Part 1 and Part 2, the embed size cannot be tuned and must be fixed at 100.

Activation	Training Accuracy	Validation Accuracy
`relu`	0.5016	0.4938
`tanh`	0.5016	0.4938
`softmax`	0.4985	0.5062
softplus	0.7257	**0.6668**

The activation choice **softplus** yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the choice of optimization.

Optimizer	Training Accuracy	Validation Accuracy
`adam`	0.7257	0.6668
`sgd`	0.5064	0.5214
`adagrad`	0.5429	0.5298
`RMSProp`	0.7343	0.6604
nadam	0.7495	**0.6708**

The optimizer **nadam** yielded the highest validation accuracy. This value is used for the remaining duration of the tuning.

The next hyperparameter tuned was the dropout rate.

Dropout Rate	Training Accuracy	Validation Accuracy
`0.5`	0.6187	0.6172
`0.8`	0.5519	0.5440
`0.35`	0.6471	`0.6336`
0.2	0.7003	**0.6560**

A dropout rate of 0.2 yielded the highest validation accuracy. The following are the final, tuned hyperparameters to be used on the test set.

Hyperparameter	Value
`Epoch Number`	7
`Embed Size`	100
`Choice of Nonlinearity`	SoftPlus
`Choice of Optimization Method`	Nadam
`Dropout Rate`	0.2

The test set yielded an accuracy of **0.6572**.

3.1 Analysis

Oddly enough, the test accuracy on the model with pretrained word embeddings was worse than the models with learned word embeddings. It would make sense for things to be the other way around, but those are the results I received. Pretrained word embeddings should produce a better accuracy since it reduces the number of parameters needed when compared to a model learning from scratch. The following is parameter information for the pretrained word embedding model:

Total params: 1,588,449
Trainable params: 88,449
Non-trainable params: 1,500,000

The model used for Part 1 and Part 2 has 0 non-trainable parameters. I am going to assume that the pretrained embeddings that I used were bad for my task.

3.2 Synonyms and Antonyms

A very known issue with word embeddings is that antonyms and synonyms have similar embeddings. Distinguishing between antonyms and synonyms is a common challenge for models centered around sentiment analysis. The issue with antonyms and synonyms is that they can be used interchangeably. They are considered indistinguishable.

Let $\boldsymbol{w}$ be the word 'increase' and $\boldsymbol{w'}$ be the word 'decrease', which is the antonym of $\boldsymbol{w}$. Using cosine similiarity, these are the following embeddings for both words:

$\boldsymbol{w}$: 0.40740427
$\boldsymbol{w'}$: 0.42761517

It is evident that the word embeddings are very similar. We can also use the following sentences to analyze the behavior of the model when encountering synonyms and antonyms.
Let $\boldsymbol{w}$ be the sentence 'the movie will increase my happiness' and $\boldsymbol{w'}$ be the sentence 'the movie will decrease my happiness'. The predictions obtained are as follows:

$\boldsymbol{w}$: 0.9415511
$\boldsymbol{w'}$: 0.95250815

where we can definitely see a similar result aswell. Additionally, the sentences $\boldsymbol{w}$: 'increase the brightness', and $\boldsymbol{w'}$: 'decrease the brightness' yield the predictions:

$\boldsymbol{w}$: 0.31889683
$\boldsymbol{w'}$: 0.35120764

which are also very similar. Swapping synonyms with their antonyms (and vise versa) has a very little effect on the model's prediction.