

CS 577 - F22 - Assignment 5

Programming Portion

Due date: 11/22/2022

Anas Puthawala - A20416308

Professor Gady Agam

Sentiment Classification

Data Description

The data used was the IMDB reviews dataset in Keras which is a bunch of reviews and its corresponding sentiment (pos. or neg.). The data is well suited for a binary classification task, although sentiment classification doesn't necessary have to be a binary classification task, it could also be multi-class (pos., neg., neutral). Reviews have been preprocessed, and each review is encoded as a list of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data.

Problem Statement & Proposed Solution

We are dealing with a binary classification task. Given the preprocessed reviews, train different models of different layers (i.e. swap a fully connected layer for an LSTM) and also evaluate the performance of using pre-trained embeddings with fixed weights and trainable weights fine-tuned to our specific task. The proposed solution consists of applying the different changes in our model to follow along with the homework details (i.e. swapping layers around, using fixed weights for embedding layer, and fine-tuning embedding weights afterwards).

Implementation Details

Some feature details I set are that `max_features = 10000`, `embedding_dim = 100` and `maxlen = 100`. The steps of implementing are as follows:

1. Load in IMDB data using keras
2. We're using `max_features` of 10,000 so only 10,000 words
3. Split data into training, val, and test sets (70, 15, 15 split)
4. Create a sequential model with a trainable embedding layer with dimension of 100, vocab size of 10,000 and max len of 100. Evaluate the performance of this model
5. Acquire and utilize glove word embeddings
 - A. Evaluate performance of model with frozen weights
 - B. Evaluate performance of model with fine-tuning
6. Compare results of models above
7. Replace the fully connected layer with an LSTM layer, utilize fine-tuning on the gloVe embeddings, and re-train and evaluate.

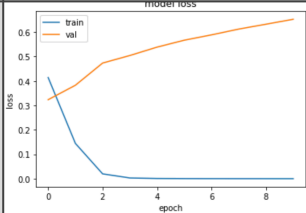
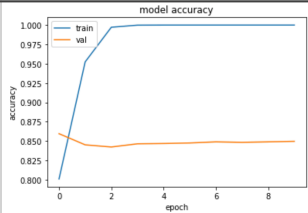
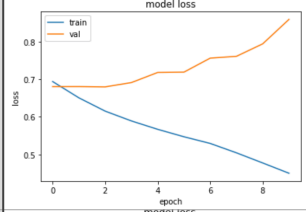
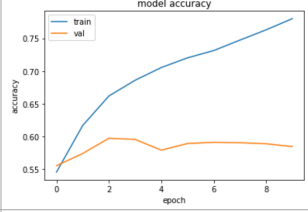
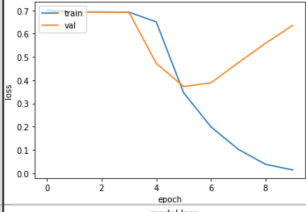
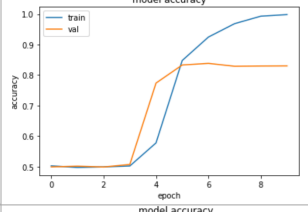
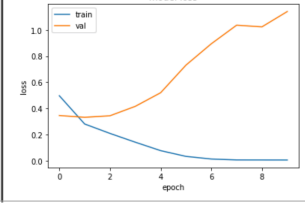
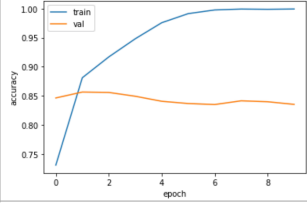
There were a few things that were adjusted in the process of hyper-parameter tuning such as the batch size and ideal epochs to train. The optimizer used was Adam with a loss function of binary crossentropy. In addition, the final output fully connected layer had an activation of 'sigmoid' and the one prior had an activation of 'relu' The metric used is accuracy. In future

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 100, 100)	1000000
flatten_3 (Flatten)	(None, 10000)	0
dense_6 (Dense)	(None, 8)	80008
dense_7 (Dense)	(None, 1)	9
=====		
Total params: 1,080,017		
Trainable params: 1,080,017		
Non-trainable params: 0		

Fig. 1

iterations, the only thing that was adjusted in the network shown in *Fig. 1* was the dense_6 layer in which it was swapped for an LSTM with 64 units. Lastly, note that final evaluations (best accuracy) were conducted after deciding optimal parameters, combining the training + validation data into one training set (using np.concatenate) and re-training for the ideal epochs then evaluating on the test set.

Results & Discussion

Model	Loss Curve	Accuracy Curve	Best Accuracy & Loss
2 fully connected layers, keras embedding layer			Acc = 85.1%, Loss = .359
2 fully connected layers, glove embedding layer, fixed weights			~58.5%, .688
2 fully connected layers, glove embedding layer, fine-tuned			~84.4%, .55
replacing 1 fc layer with LSTM, fine-tuning glove embedding layer			~86.9%, .311

Note that in all cases, the model was significantly overfitting. Since the assignment did not explicitly state to tune hyper-parameters as needed, I left it out and did not tune in such great details. Some aspects that may have helped in the tuning process would be adding dropout, weight decay, batch normalization, and tweaking the learning rate.

The best model performance came from when we swapped the fully connected layer with the LSTM and fine-tuned the GloVe embeddings. Although it's important to note that using Keras' learning embedding layer is not very far off itself. Not tuning the GloVe embedding layer (i.e. using the pre-trained weights) results in poor performance, close to 60%. Therefore, it's mostly good to fine-tune the embedding layer if we result to using a third-party one as opposed to just learning the weights for our model using Keras' embedding layer.

Topic Classification

Data Description

The data used was the Reuters newswire dataset from Keras. The data is well-suited for a multi-class classification task. The newswire have been pre-processed internally from Keras and each newswire is encoded as a list of word indexes (integers). There are a total of 46 topics for which we will try to develop a model to try and classify between them.

Problem Statement & Proposed Solution

We are dealing with a multi-class classification task. Given the preprocessed reviews, train different models of different layers (i.e. swap a fully connected layer for an LSTM) and also evaluate the performance of using pre-trained embeddings with fixed weights and trainable weights fine-tuned to our specific task. The proposed solution consists of applying the different changes in our model to follow along with the homework details (i.e. swapping layers around, using fixed weights for embedding layer, and fine-tuning embedding weights afterwards).

Implementation Details

Some feature details I set are that `max_features = 10000`, `embedding_dim = 100` and `maxlen = 100`. The steps of implementing are as follows:

1. Load in Reuters newswire data using Keras
2. We're using `max_features` of 10,000 so only 10,000 words
3. Split data into training, val, and test sets (70, 15, 15 split)
 - Additional care was taken here with the labels, we had to use the 'to_categorical' function in Keras to OHE the labels into a vector of length 46.
4. Create a sequential model with a trainable embedding layer with dimension of 100, vocab size of 10,000 and max len of 100. Evaluate the performance of this model
5. Acquire and utilize glove word embeddings
 - Evaluate performance of model with frozen weights
 - Evaluate performance of model with fine-tuning
6. Compare results of models above
7. Replace the fully connected layer with an LSTM layer, utilize fine-tuning on the glove embeddings, and re-train and evaluate.

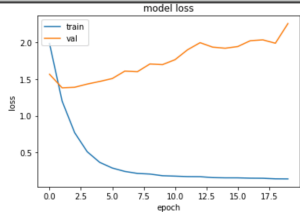
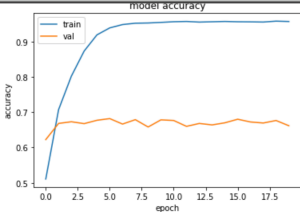
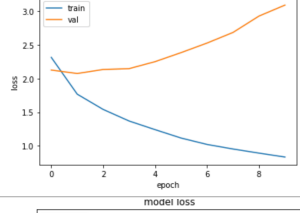
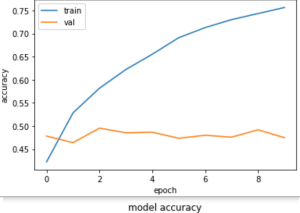
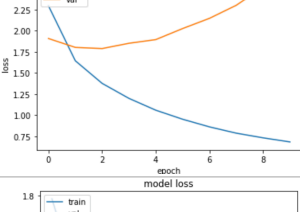
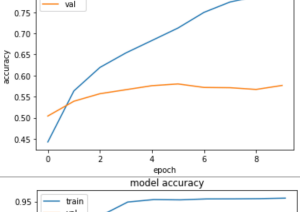
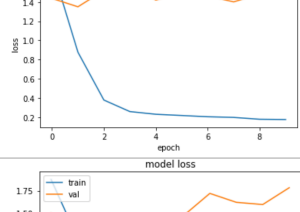
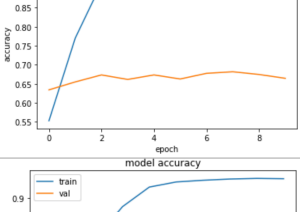
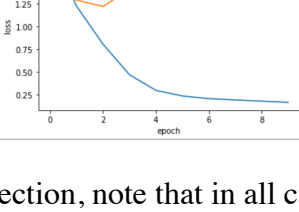
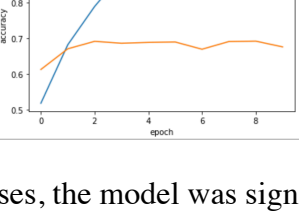
Since we're dealing with a multi-class classification problem, our final dense layer will consist of a softmax with 46 total units because there are 46 topics. There were a few things that were adjusted in the process of hyper-parameter tuning such as the batch size and ideal epochs to train. The optimizer used was Adam with a loss function of categorical crossentropy. The metric used is 'categorical_accuracy'. And in future iterations, the only thing that was adjusted in the network shown in *Fig. 2*

Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(None, 100, 100)	1000000
flatten_6 (Flatten)	(None, 10000)	0
dense_12 (Dense)	(None, 8)	80008
dense_13 (Dense)	(None, 46)	414
Total params: 1,080,422		
Trainable params: 1,080,422		
Non-trainable params: 0		

Fig. 2

was the dense_12 layer in which it was swapped for an LSTM with 64 units. Lastly, note that final evaluations (best categorical accuracy) were conducted after deciding optimal parameters, combining the training + validation data into one training set (using np.concatenate) and re-training for the ideal epochs then evaluating on the test set.

Results & Discussion

Model	Loss Curve	Accuracy Curve	Best Accuracy & Loss
2 fully connected layers, keras embedding layer			~70.1%, 1.563
2 fully connected layers, glove embedding layer, fixed weights			~46.4%, 2.304
2 fully connected layers, glove embedding layer, fine-tuned			~60.5%, 2.215
replacing 1 fc layer with LSTM (1 LSTM layer), fine-tuning glove embedding layer			~69.4%, 1.451
Adding on another LSTM layer (2 total LSTM), fine-tuning glove embedding layer			~70%, 1.461

Similar to the previous section, note that in all cases, the model was significantly overfitting. Since the assignment did not explicitly state to tune hyper-parameters as needed, I left it out and did not tune in such great details. Some aspects that may have helped in the tuning process would be adding dropout, weight decay, batch normalization, and tweaking the learning rate. The best model performance stemmed from using fully connected layers and Keras' embedding layer. Although it's important to note that using two LSTM layers and fine-tuning GloVe embeddings is not very far off itself. Not tuning the GloVe embedding layer (i.e. using the pre-

trained weights) results in poor performance. Therefore, it's mostly good to fine-tune the embedding layer if we result to using a third-party one as opposed to just learning the weights for our model using Keras' embedding layer.

References

[1] <https://keras.io/api/datasets/imdb/>

[2] <https://keras.io/api/datasets/reuters/>

[4] <https://stackoverflow.com/questions/50060241/how-to-use-glove-word-embeddings-file-on-google-colaboratory>