MA 721 Project 2

Derek Jones

December 9, 2017

1 Problem and Dataset Description

For this project the task is to classify the sentiment (positive/negative) of a movie viewer given a tokenized version of their review. Each word in the set of all possible words contained in the dataset is encoded as a natural number. Each sentence is represented by an array of these natural numbers. For each sentence, a sentiment score is assigned based upon the particular sequence of words, thresholds are given that segment these scores between positive, negative, and neutral sentiment. All reviews with neutral sentiment are excluded.

2 Methods

I chose to implement a basic convolutional neural network (CNN) and recurrent neural network (RNN). The specific implementations are detailed in the following section.

2.1 Networks & Implementation Details

All implementations were developed in python 3.6.1 using the Keras neural network API for tensorflow. For each of the networks I chose to use an embedding layer as the input layer in order to allow the networks to learn lower dimensional representations of the input sequences. This layer accounted for most of the parameters in each of the networks. When training each of the models I use early stopping in order to stop training when the model begins to overfit wrt to the validation data performance. I use a 10% split of the training data for validation in order to assess generalization performance during training. The Adam optimizer with default settings in keras is chosen as the optimizer. The initialization of each of the networks is random.

2.1.1 Recurrent Neural Network (RNN)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 250, 128)          2560000
_________________________________________________________________
dropout_1 (Dropout)          (None, 250, 128)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                35800
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
```

```
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 2,595,851
Trainable params: 2,595,851
Non-trainable params: 0
_________________________________________________________________
```

2.1.2 Convolutional Neural Network (CNN)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 250, 128)          2560000
_________________________________________________________________
batch_normalization_1 (Batch (None, 250, 128)          512
_________________________________________________________________
dropout_1 (Dropout)          (None, 250, 128)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 195, 25)           179225
_________________________________________________________________
batch_normalization_2 (Batch (None, 195, 25)           100
_________________________________________________________________
dropout_2 (Dropout)          (None, 195, 25)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 97, 25)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2425)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2426
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 2
=================================================================
Total params: 2,742,265
Trainable params: 2,741,959
Non-trainable params: 306
_________________________________________________________________
```

3 Results

Network	**minibatch size**	**# Hidden Layers**	**#Hidden Units**	**Total # parameters**
RNN	1000	1	2,595,851	2,595,851
CNN	1000	1	2,741,959	2,742,265

Figure 1: Network Descriptions

Network	optimizer	lr	initialization	epochs
RNN	Adam	1e-3	random	5
CNN	Adam	1e-3	random	33

Figure 2: Optimization details

Dataset	1 - drop prob	Accuracy	Ye score
RNN	0.3	0.853	0.195
CNN	0.3	0.859	0.195

Figure 3: Test Set Performance

3.1 RNN Figures

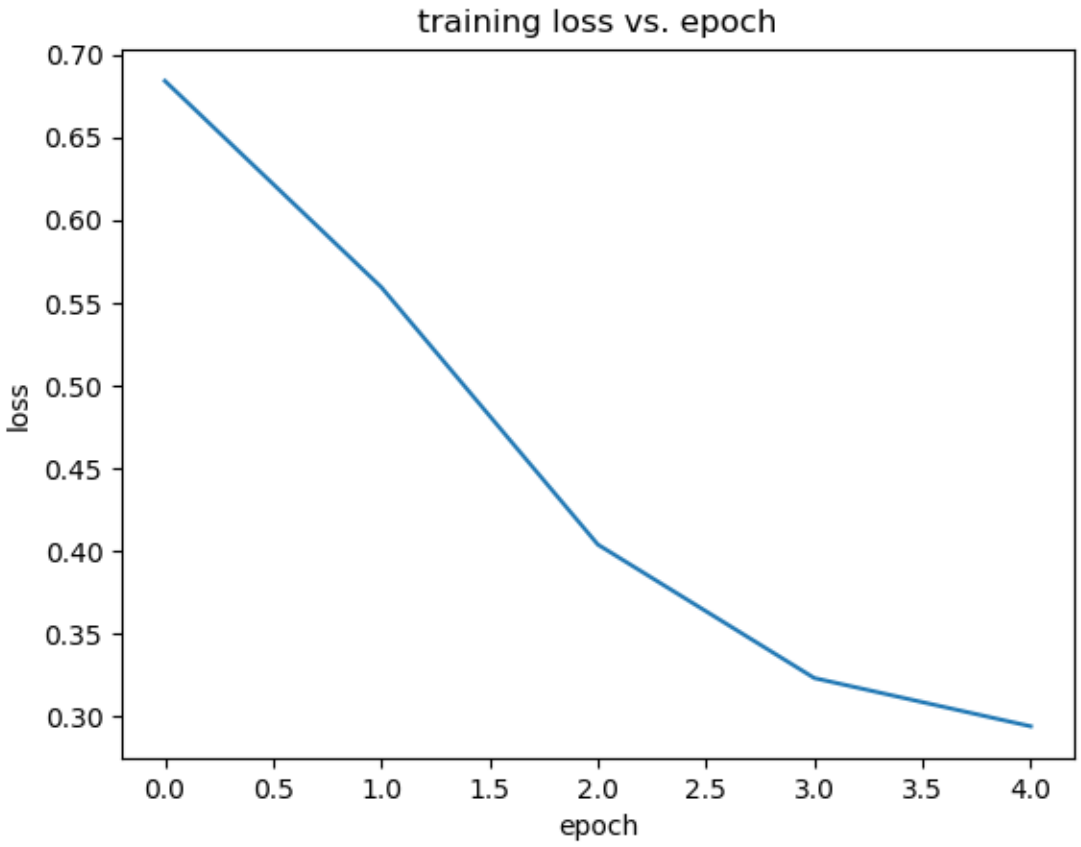

Figure 4: CNN training loss

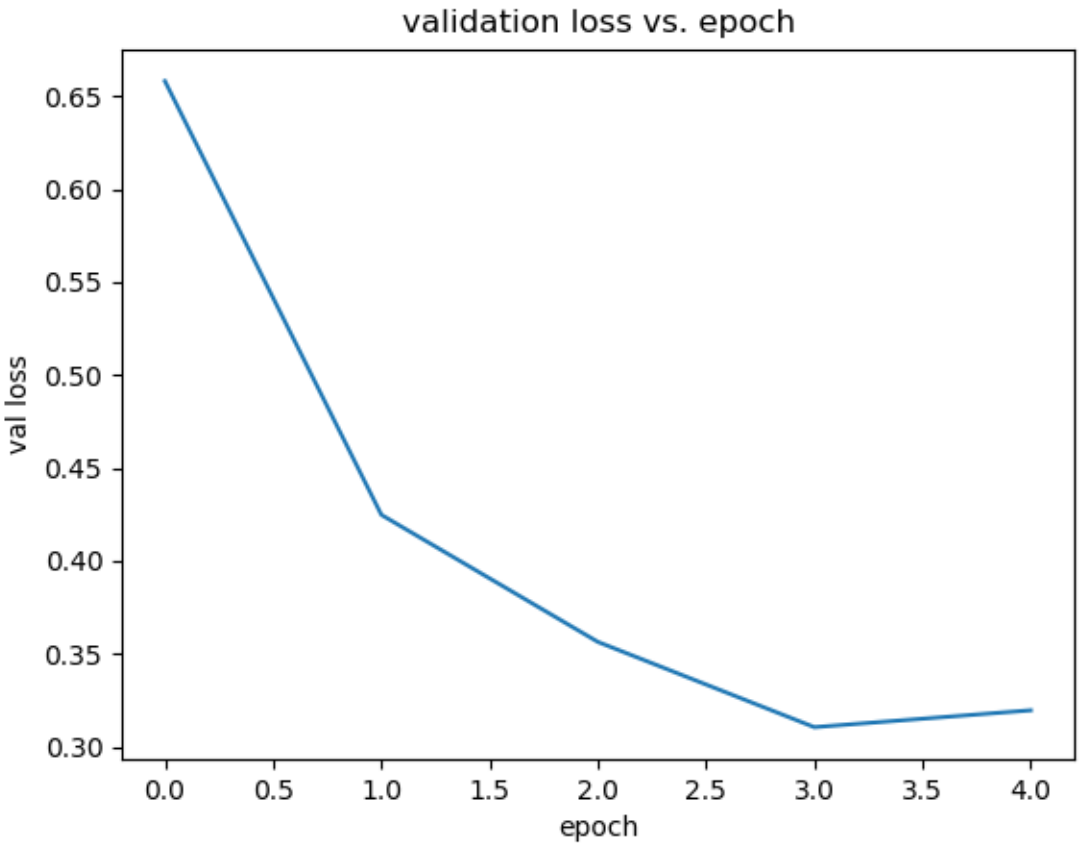

Figure 5: CNN validation loss

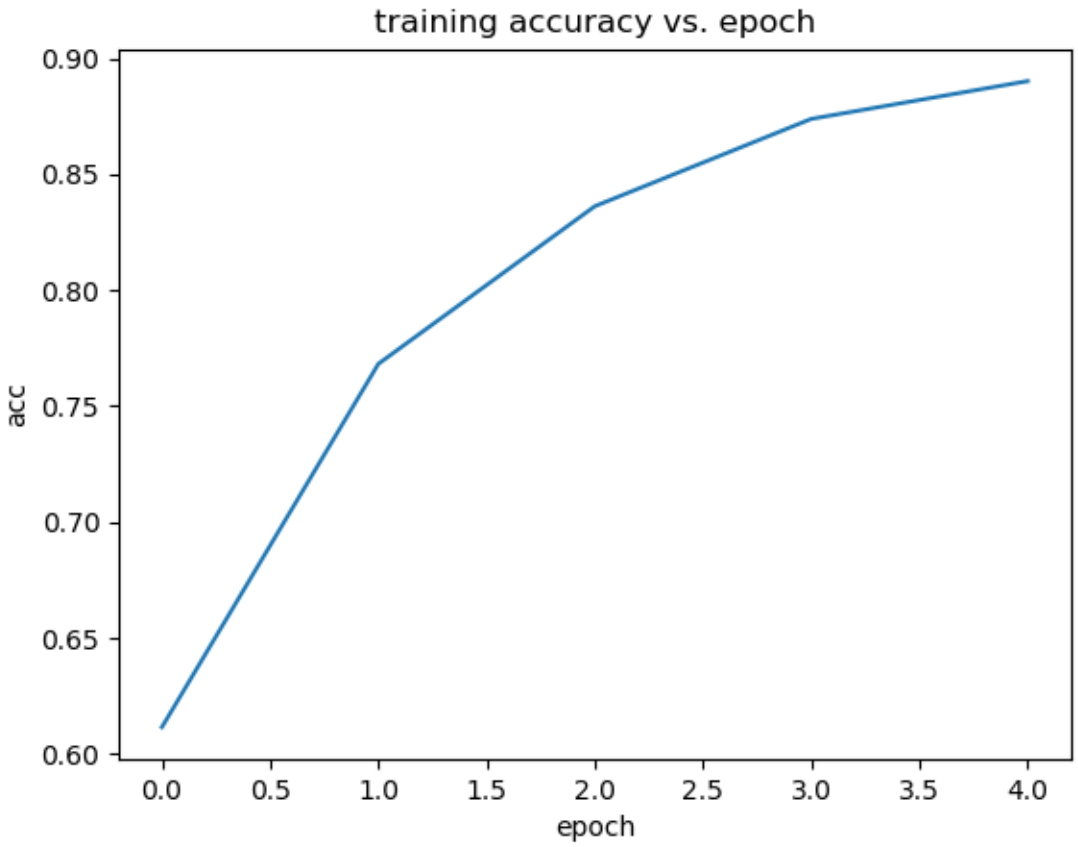

Figure 6: CNN training accuracy

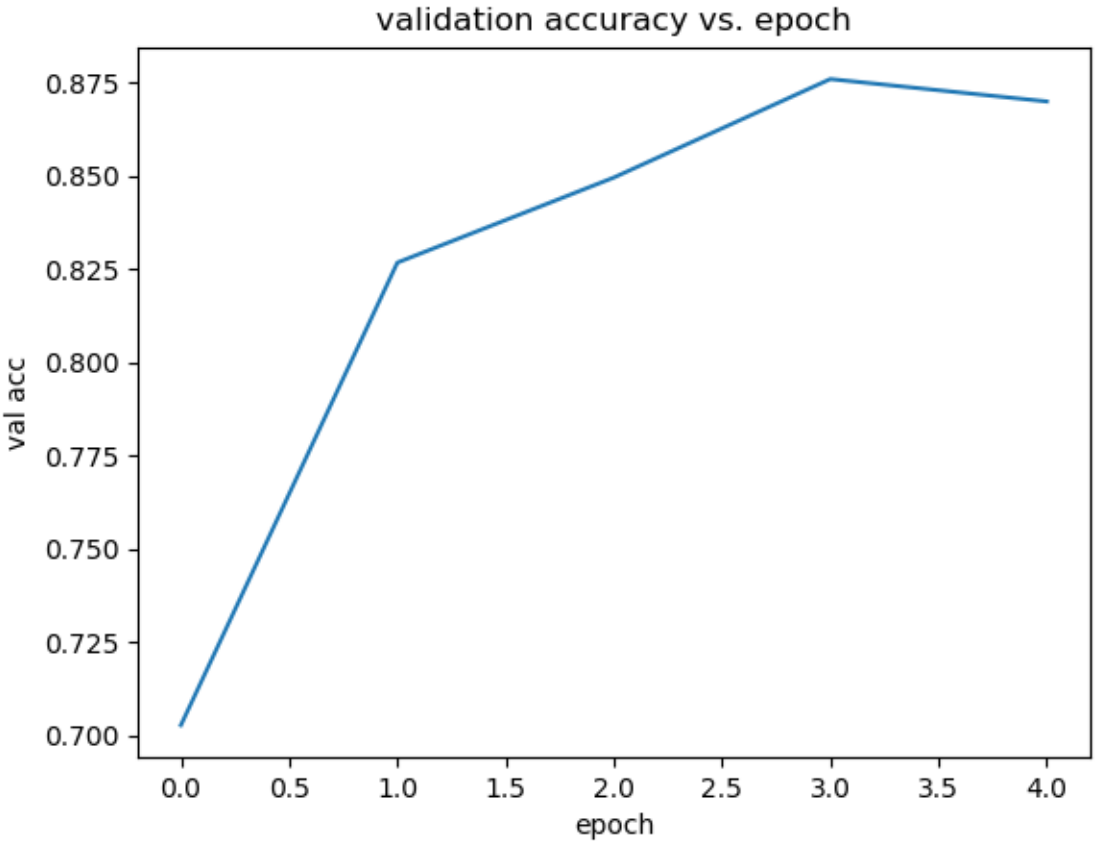

Figure 7: CNN validation accuracy

3.2 CNN Figures

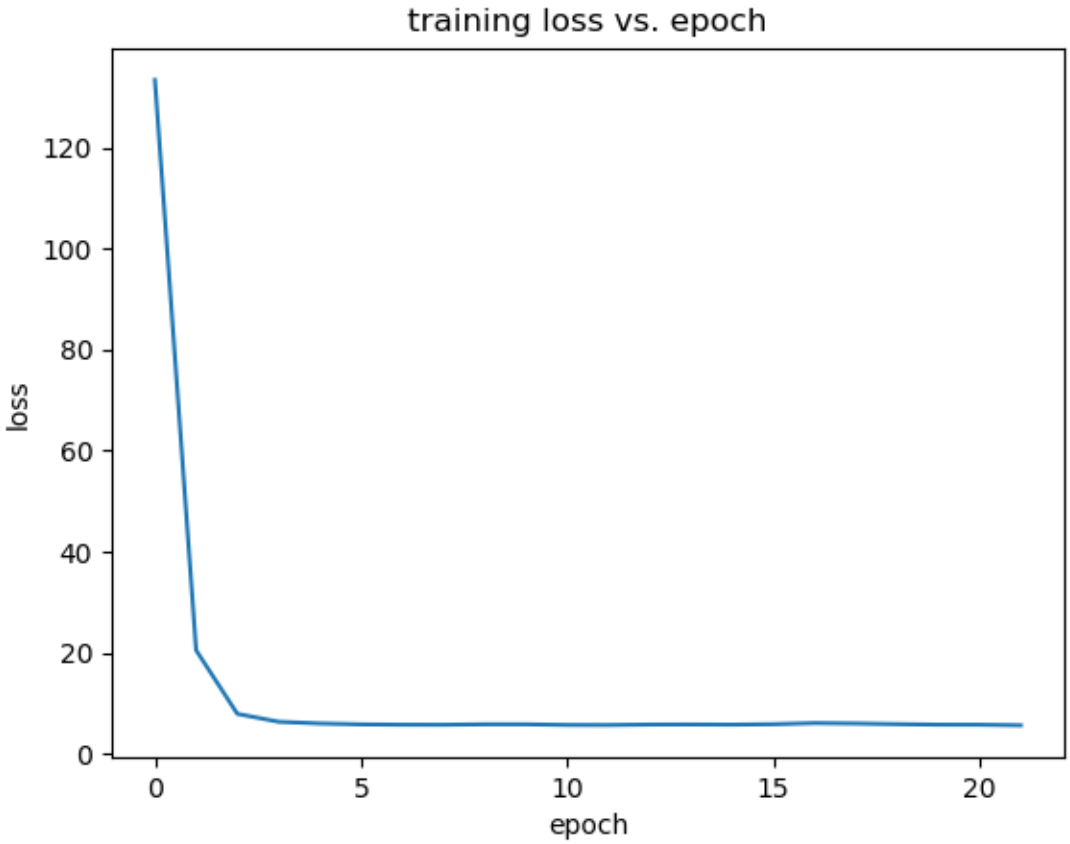

Figure 8: CNN training loss

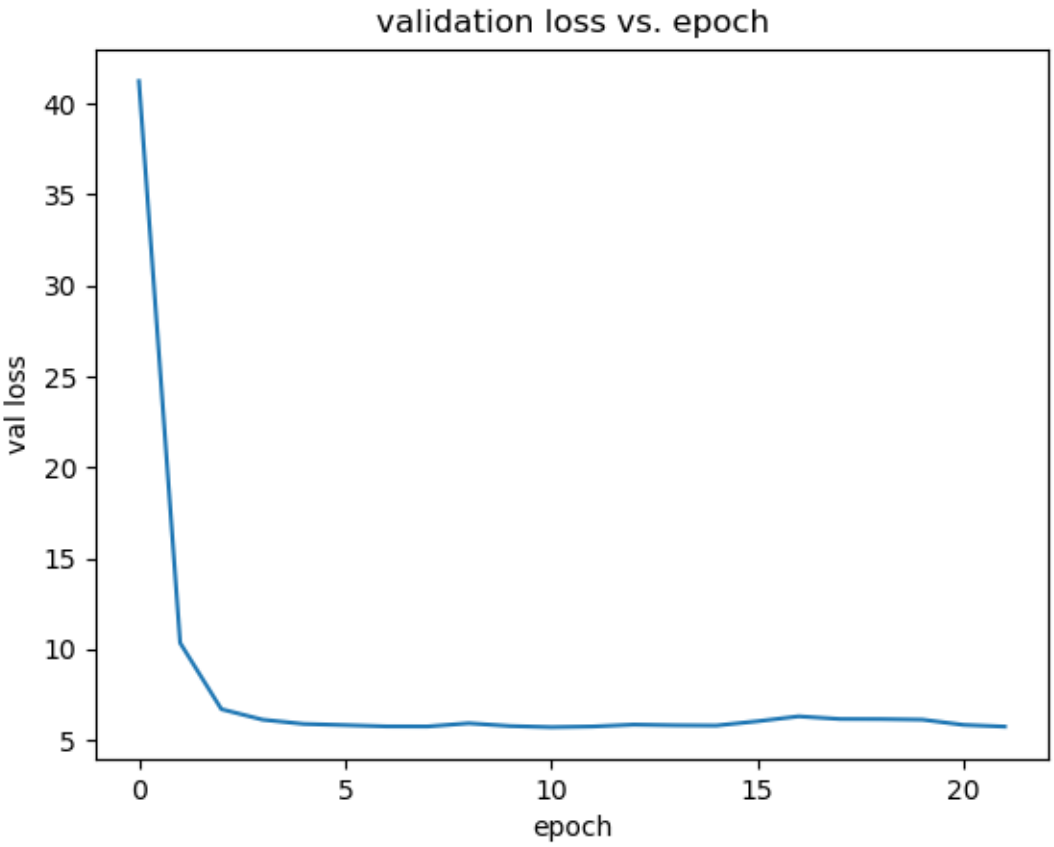

Figure 9: CNN validation loss

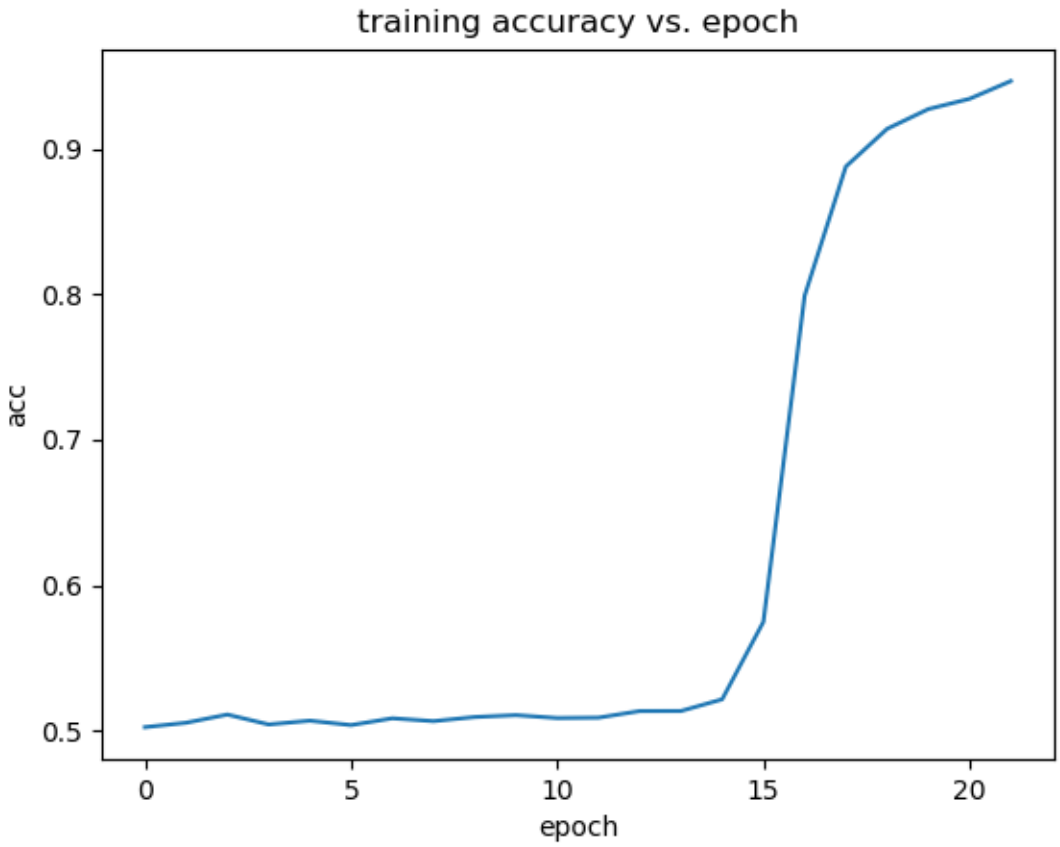

Figure 10: CNN training accuracy

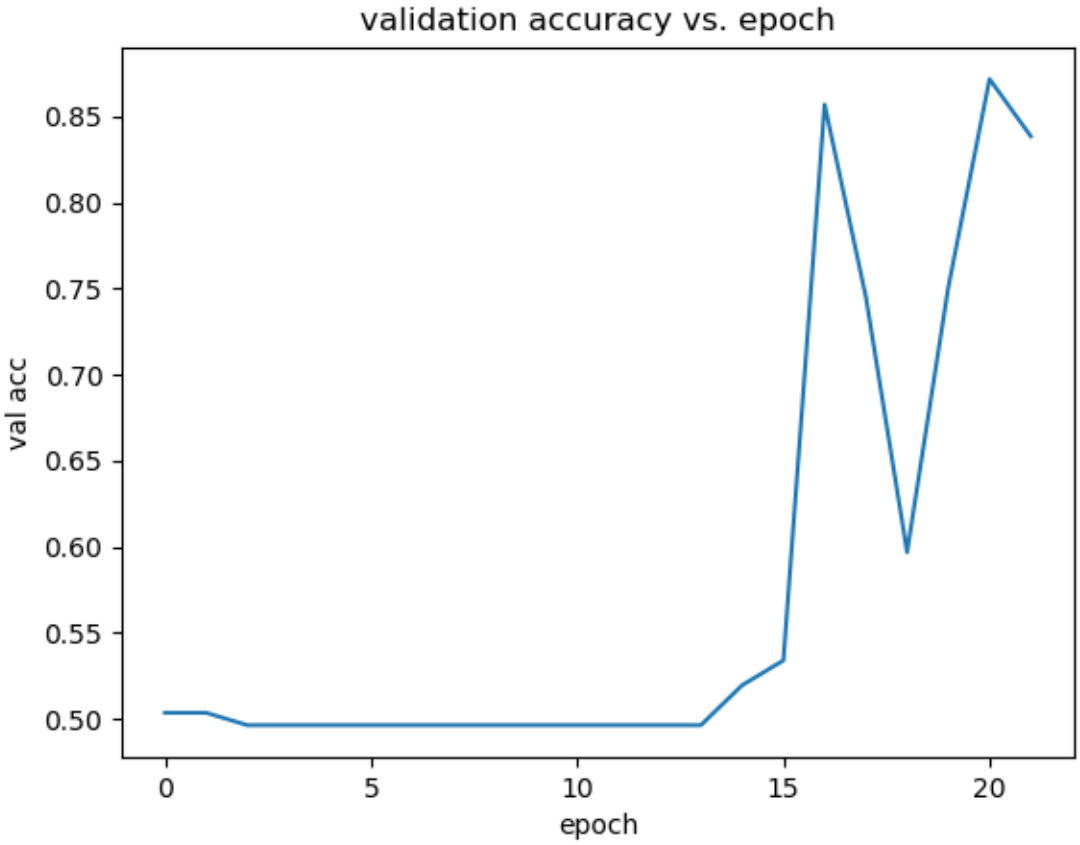

Figure 11: CNN validation accuracy

4 Conclusions

I found both of the networks to achieve about the same performance with the LSTM learning much faster than the CNN. For the CNN, specifying a kernel size less than the size of the input dimension as a whole resulted in much slower and or degraded learning. Defining the convolution to be the entire size of the input worked much better. Overall, the LSTM would be the better method in my opinion as it trains much faster than the CNN. This gives credibility to the argument that the RNN is superior at sequence processing tasks. The CNN is able to adapt quite successfully, however it does require all inputs to be of the same length. This can cause problems when few sentences are quite long w.r.t. the lengths of the remaining sentences.