SENTIMENT ANALYSIS OF RATINGS ON *Amazon customer review dataset for video games Project Milestone-1 Report*

A PREPRINT

Amogh Vikram Sirohi
GCCIS, Rochester Institute of technology
Rochester, NY 14623
as4483@rit.edu

Dhrumil Mehta
GCCIS, Rochester Institute of technology
Rochester, NY 14623
dm9076@rit.edu

Sharjeel Ansari
GCCIS, Rochester Institute of technology
Rochester, NY 14623
sa2676@rit.edu

Vivek Lad
GCCIS, Rochester Institute of technology
Rochester, NY 14623
vl9244@rit.edu

July 13, 2020

1 Data Preprocessing

Initially, the database that we are using consists of 497577 rows and 12 features. We won't be using all the features, therefore we preprocess the data to change its shape and type in order to feed it to a machine learning model. In doing so, we go through following steps:

1. Feature Selection
2. Data cleaning
3. Incorporating polarity
4. Tokenizing and Sequence generation
5. Encoding

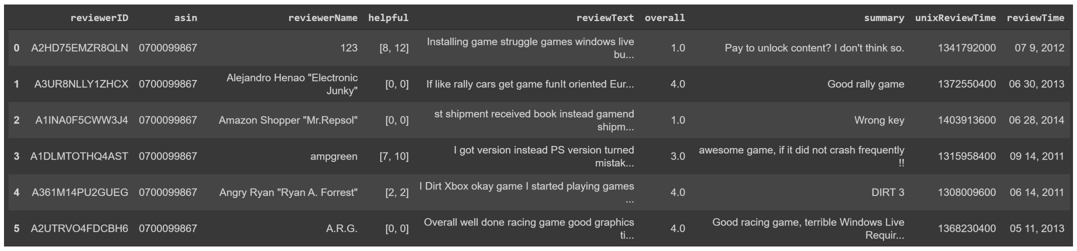
1.1 Feature Selection

As mentioned earlier, the dataset has 12 features. Out of which, we focus only on two, namely - *ReviewText* and *Overall*. ReviewText consists of comments written by customers for products, and Overall consists of the ratings provided by the customers for that product. We extract these two columns from the original dataset.

1.2 Data cleaning

In the process of cleaning we have largely performed 4 tasks,

1. Removal of non alphabetical characters from the text such as punctuation's and numerics
2. Removal of URL links from the text
3. Removal of stopwords from the text
4. Removal of rows with "overall" values not in range 1.0-5.0

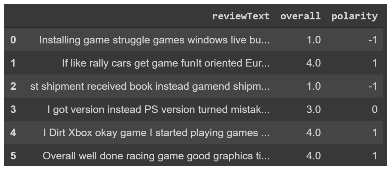


	reviewerID	asin	reviewerName	helpful	reviewText	overall	summary	unixReviewTime	reviewTime
0	A2HD75EMZR8QLN	0700099867	123	[8, 12]	Installing game struggle games windows live bu...	1.0	Pay to unlock content? I don't think so.	1341792000	07 9, 2012
1	A3UR8NLLY1ZHCX	0700099867	Alejandro Henao "Electronic Junky"	[0, 0]	If like rally cars get game funIt oriented Eur...	4.0	Good rally game	1372550400	06 30, 2013
2	A1INA0F5CWW3J4	0700099867	Amazon Shopper "Mr.Repsol"	[0, 0]	st shipment received book instead gamend shipm...	1.0	Wrong key	1403913600	06 28, 2014
3	A1DLMTOTHQ4AST	0700099867	ampgreen	[7, 10]	I got version instead PS version turned mistak...	3.0	awesome game, if it did not crash frequently !!	1315958400	09 14, 2011
4	A361M14PU2GUEG	0700099867	Angry Ryan "Ryan A. Forrest"	[2, 2]	I Dirt Xbox okay game I started playing games ...	4.0	DIRT 3	1308009600	06 14, 2011
5	A2UTRVO4FDCBH6	0700099867	A.R.G.	[0, 0]	Overall well done racing game good graphics ti...	4.0	Good racing game, terrible Windows Live Requir...	1368230400	05 11, 2013

Figure 1: After cleaning

1.3 Incorporating polarity

Polarity works as a threshold in our case and accounts for the fact that overall ratings can be subjective. Thus, they can simply be classified into negative(-1), neutral(0) and positive reviews(+1). Then furthermore, we only extract relevant features in order to obtain the tables as:



	reviewText	overall	polarity
0	Installing game struggle games windows live bu...	1.0	-1
1	If like rally cars get game funIt oriented Eur...	4.0	1
2	st shipment received book instead gamend shipm...	1.0	-1
3	I got version instead PS version turned mistak...	3.0	0
4	I Dirt Xbox okay game I started playing games ...	4.0	1
5	Overall well done racing game good graphics ti...	4.0	1

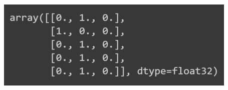
Figure 2: After Incorporating Polarity

1.4 Tokenizing and Sequence generation

We have made use of a preprocessing tool available in tensorflow known as *Tokenizer*. So, we created a Tokenizer object and gave the *vocabulary size* as 100 and used *<OOV_TOKEN>* to take care of new values which have not been seen before. Further, we assign a *maximum length* of a sentence to be 100 and take care of *padding* i.e sentences with length less than 100 and *truncating* i.e sentences with length greater than 100, both are of type "post" thus, they work from the end of the sentences. Using the Tokenizer's built-in function *texts_to_sequences* we convert the sentences to sequences and using *pad_sequences* we obtain padded sequences.

1.5 Encoding

We encode our data using tensorflows *to_categorical* library in order to numpy arrays of categorical data. This is used to define if a neutral, negative or positive response is present(1) or absent(0)



```
array([[0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.]], dtype=float32)
```

Figure 3: After Encoding

2 Sentiment Analysis using dense neural networks

2.1 Overview

We experiment by using dense(feed forward) neural networks in order to predict if the polarity of a given input of sentence is positive, negative or neutral.

2.2 Network architecture

- Input layer: Embedding layer with vocabulary size = 100 and embedding of 100 dimensions each.
- Layer 2: Global average pooling 1D layer

- Layer 3: Fully connected layer with 10 nodes with ReLU activation.
- Layer 4: Fully connected layer with 5 nodes with ReLU activation.
- Output layer: 3 nodes with Softmax activation.

2.3 Loss function and optimization

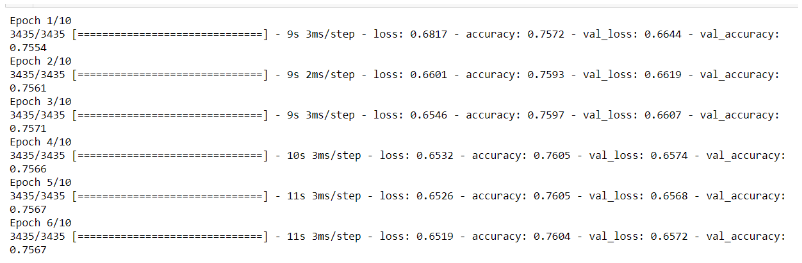
- Loss function: Categorical cross entropy
- Optimizer - Adam Optimizer

2.4 Training information

1. The network was trained for 10 epochs with a validation split = 0.2
2. In order to ensure that the model does not continue training when the loss curves overshoot, we have made use of early stopping criteria. The early stopping criteria monitor the loss curves and check if they are decreasing with every epochs. The patience value was set to 1 which means that the model will stop training if the loss increases instead of decreasing even once.

2.5 Results

The following image denotes the results of the training of above network:



```
Epoch 1/10
3435/3435 [==============================] - 9s 3ms/step - loss: 0.6817 - accuracy: 0.7572 - val_loss: 0.6644 - val_accuracy:
0.7554
Epoch 2/10
3435/3435 [==============================] - 9s 2ms/step - loss: 0.6601 - accuracy: 0.7593 - val_loss: 0.6619 - val_accuracy:
0.7561
Epoch 3/10
3435/3435 [==============================] - 9s 3ms/step - loss: 0.6546 - accuracy: 0.7597 - val_loss: 0.6607 - val_accuracy:
0.7571
Epoch 4/10
3435/3435 [==============================] - 10s 3ms/step - loss: 0.6532 - accuracy: 0.7605 - val_loss: 0.6574 - val_accuracy:
0.7566
Epoch 5/10
3435/3435 [==============================] - 11s 3ms/step - loss: 0.6526 - accuracy: 0.7605 - val_loss: 0.6568 - val_accuracy:
0.7567
Epoch 6/10
3435/3435 [==============================] - 11s 3ms/step - loss: 0.6519 - accuracy: 0.7604 - val_loss: 0.6572 - val_accuracy:
0.7567
```

Figure 4: Output of training a dense neural network.

Accuracy and loss curves:

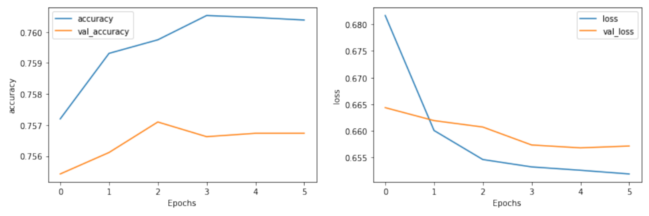


Figure 5: Learning curves of training a dense neural network.

2.6 Observations

1. The training was halted after training for 6th epoch because the early stopping criteria was met.
2. The validation and training accuracies decreases after Epoch 3 with accuracy about 75.7%

3 Hyperparameter tuning for the dense network

3.1 Overview

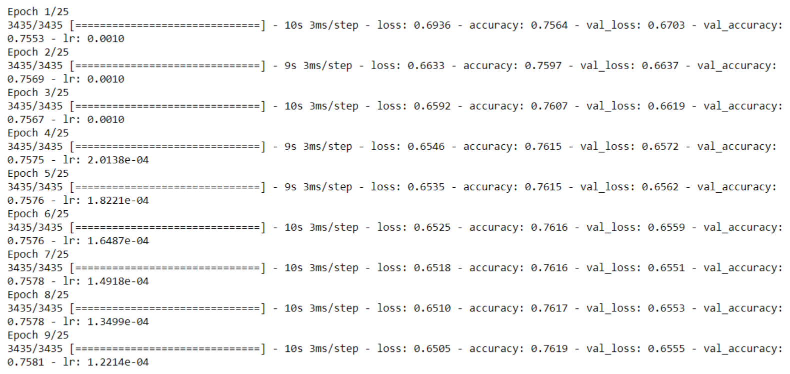
In order to solve some of the flaws of the above network, we perform some hyperparameter tuning on the same. The hyperparameter tuning performed involves only some minor changes in the configuration and hence the instructions for the same have been mentioned in the IPython Notebook.

3.2 Changes

1. In this new configuration, we change the number of epochs to be trained to 25 instead of 10.
2. We increased the value of patience in early stopping criteria from 1 to 2.
3. We added Learning Rate Scheduler which decays the learning rate after a specific epoch has been trained. We have included this in the form of exponential decay in learning rate after epoch 3.

3.3 Results

The following image denotes the results of the training of above network:



```
Epoch 1/25
3435/3435 [==============================] - 10s 3ms/step - loss: 0.6936 - accuracy: 0.7564 - val_loss: 0.6703 - val_accuracy:
0.7553 - lr: 0.0010
Epoch 2/25
3435/3435 [==============================] - 9s 3ms/step - loss: 0.6633 - accuracy: 0.7597 - val_loss: 0.6637 - val_accuracy:
0.7569 - lr: 0.0010
Epoch 3/25
3435/3435 [==============================] - 10s 3ms/step - loss: 0.6592 - accuracy: 0.7607 - val_loss: 0.6619 - val_accuracy:
0.7567 - lr: 0.0010
Epoch 4/25
3435/3435 [==============================] - 9s 3ms/step - loss: 0.6546 - accuracy: 0.7615 - val_loss: 0.6572 - val_accuracy:
0.7575 - lr: 2.0138e-04
Epoch 5/25
3435/3435 [==============================] - 9s 3ms/step - loss: 0.6535 - accuracy: 0.7615 - val_loss: 0.6562 - val_accuracy:
0.7576 - lr: 1.8221e-04
Epoch 6/25
3435/3435 [==============================] - 10s 3ms/step - loss: 0.6525 - accuracy: 0.7616 - val_loss: 0.6559 - val_accuracy:
0.7576 - lr: 1.6487e-04
Epoch 7/25
3435/3435 [==============================] - 10s 3ms/step - loss: 0.6518 - accuracy: 0.7616 - val_loss: 0.6551 - val_accuracy:
0.7578 - lr: 1.4918e-04
Epoch 8/25
3435/3435 [==============================] - 10s 3ms/step - loss: 0.6510 - accuracy: 0.7617 - val_loss: 0.6553 - val_accuracy:
0.7578 - lr: 1.3499e-04
Epoch 9/25
3435/3435 [==============================] - 10s 3ms/step - loss: 0.6505 - accuracy: 0.7619 - val_loss: 0.6555 - val_accuracy:
0.7581 - lr: 1.2214e-04
```

Figure 6: Output of training a tuned dense neural network.

Accuracy and loss curves:

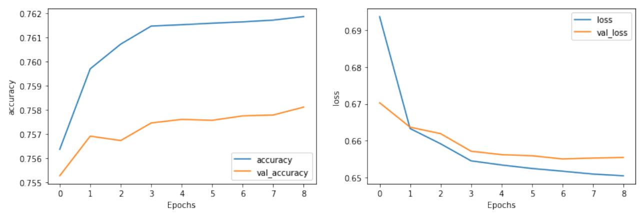


Figure 7: Learning curves of training a tuned dense neural network.

3.4 Observations

1. The learning curves are better than the previous configuration with accuracy slightly better = 75.8%
2. One can observe in the output image that the learning rate decays after Epoch 3.
3. Early stopping criteria is met after training for Epoch 9.

4 LSTM (Long Short Term Memory) networks Overview

Traditional neural Networks fail to classify the happenings of an event at any given point of time using any kind of previous knowledge about the event. Recurrent Neural Networks(RNNs) seem to address this issue. They can persist information with the help of loops in the networks. A recurrent Neural Network can be thought of as a chain of similar multiple copies repeating modules of same network where each network passes on information to the next network. However, these only work in situations where we need to look for recent information to perform a particular task in present.

Thus, LSTM networks were introduced. These are special kinds of RNNs and are known as Long-short Term Memory networks, capable of learning long-term dependencies. These are capable of storing information for long periods of time. The standard RNNs have a simple structure consisting of a single tanh layer in its repeating modules. However, the LSTMs have 4 different layers in the repeating module interacting in a special way to achieve its purpose.

5 Sentiment Analysis Using LSTM network

5.1 Overview

We experiment by using LSTM neural networks in order to predict if the polarity of a given input of sentence is positive, negative or neutral.

5.2 Network architecture

- Input layer: Embedding layer with vocabulary size = 100 and embedding of 100 dimensions each.
- Layer 2: LSTM layer with 18 nodes.
- Layer 3: Fully connected layer with 10 nodes with ReLU activation.
- Output layer: 3 nodes with Softmax activation.

5.3 Loss function and optimization

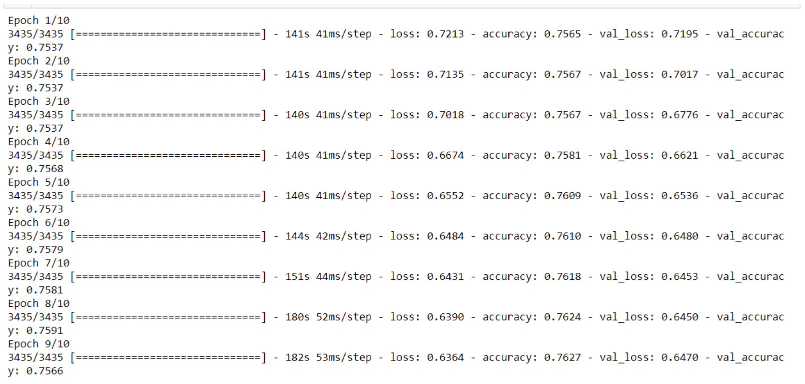
- Loss function: Categorical cross entropy
- Optimizer - Adam Optimizer

5.4 Training information

1. The network was trained for 10 epochs with a validation split = 0.2
2. In order to ensure that the model does not continue training when the loss curves overshoot, we have made use of early stopping criteria. The early stopping criteria monitor the loss curves and check if they are decreasing with every epochs. The patience value was set to 1 which means that the model will stop training if the loss increases instead of decreasing even once.

5.5 Results

The following image denotes the results of the training of above network:



```
Epoch 1/10
3435/3435 [==============================] - 141s 41ms/step - loss: 0.7213 - accuracy: 0.7565 - val_loss: 0.7195 - val_accurac
y: 0.7537
Epoch 2/10
3435/3435 [==============================] - 141s 41ms/step - loss: 0.7135 - accuracy: 0.7567 - val_loss: 0.7017 - val_accurac
y: 0.7537
Epoch 3/10
3435/3435 [==============================] - 140s 41ms/step - loss: 0.7018 - accuracy: 0.7567 - val_loss: 0.6776 - val_accurac
y: 0.7537
Epoch 4/10
3435/3435 [==============================] - 140s 41ms/step - loss: 0.6674 - accuracy: 0.7581 - val_loss: 0.6621 - val_accurac
y: 0.7568
Epoch 5/10
3435/3435 [==============================] - 140s 41ms/step - loss: 0.6552 - accuracy: 0.7609 - val_loss: 0.6536 - val_accurac
y: 0.7573
Epoch 6/10
3435/3435 [==============================] - 144s 42ms/step - loss: 0.6484 - accuracy: 0.7610 - val_loss: 0.6480 - val_accurac
y: 0.7579
Epoch 7/10
3435/3435 [==============================] - 151s 44ms/step - loss: 0.6431 - accuracy: 0.7618 - val_loss: 0.6453 - val_accurac
y: 0.7581
Epoch 8/10
3435/3435 [==============================] - 180s 52ms/step - loss: 0.6390 - accuracy: 0.7624 - val_loss: 0.6450 - val_accurac
y: 0.7591
Epoch 9/10
3435/3435 [==============================] - 182s 53ms/step - loss: 0.6364 - accuracy: 0.7627 - val_loss: 0.6470 - val_accurac
y: 0.7566
```

Figure 8: Output of training a LSTM neural network.

Accuracy and loss curves:

5.6 Observations

1. The training was halted after training for 9th epoch because the early stopping criteria was met.
2. The validation accuracy decreases during the 9th epoch training.

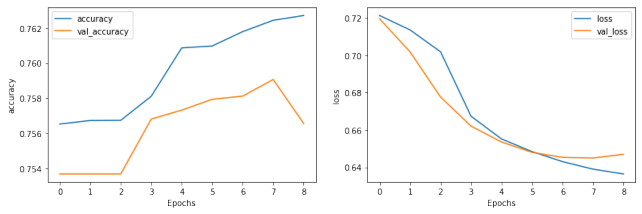


Figure 9: Learning curve of training a LSTM neural network.

6 Hyperparameter tuning for the LSTM network

6.1 Overview

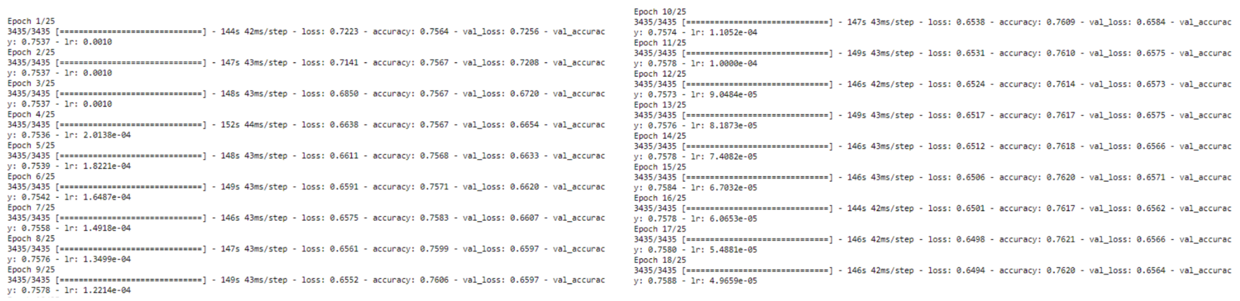
In order to solve some of the flaws of the above network, we perform some hyperparameter tuning on the same. The hyperparameter tuning performed involves only some minor changes in the configuration and hence the instructions for the same have been mentioned in the IPython Notebook.

6.2 Changes

The changes are similar to the hyperparameter tuning for the Dense Network.

6.3 Results

The following image denotes the results of the training of above network (**Patience = 2**):



```
Epoch 1/25
3435/3435 [==============================] - 144s 42ms/step - loss: 0.7223 - accuracy: 0.7564 - val_loss: 0.7256 - val_accurac
y: 0.7537 - lr: 0.0010
Epoch 2/25
3435/3435 [==============================] - 147s 43ms/step - loss: 0.7141 - accuracy: 0.7567 - val_loss: 0.7208 - val_accurac
y: 0.7537 - lr: 0.0010
Epoch 3/25
3435/3435 [==============================] - 148s 43ms/step - loss: 0.6850 - accuracy: 0.7567 - val_loss: 0.6720 - val_accurac
y: 0.7537 - lr: 0.0010
Epoch 4/25
3435/3435 [==============================] - 152s 44ms/step - loss: 0.6638 - accuracy: 0.7567 - val_loss: 0.6654 - val_accurac
y: 0.7536 - lr: 2.0138e-04
Epoch 5/25
3435/3435 [==============================] - 148s 43ms/step - loss: 0.6611 - accuracy: 0.7568 - val_loss: 0.6633 - val_accurac
y: 0.7539 - lr: 1.8221e-04
Epoch 6/25
3435/3435 [==============================] - 149s 43ms/step - loss: 0.6591 - accuracy: 0.7571 - val_loss: 0.6620 - val_accurac
y: 0.7542 - lr: 1.6487e-04
Epoch 7/25
3435/3435 [==============================] - 146s 43ms/step - loss: 0.6575 - accuracy: 0.7583 - val_loss: 0.6607 - val_accurac
y: 0.7558 - lr: 1.4918e-04
Epoch 8/25
3435/3435 [==============================] - 147s 43ms/step - loss: 0.6561 - accuracy: 0.7599 - val_loss: 0.6597 - val_accurac
y: 0.7576 - lr: 1.3499e-04
Epoch 9/25
3435/3435 [==============================] - 149s 43ms/step - loss: 0.6552 - accuracy: 0.7606 - val_loss: 0.6597 - val_accurac
y: 0.7578 - lr: 1.2214e-04
```

```
Epoch 10/25
3435/3435 [==============================] - 147s 43ms/step - loss: 0.6538 - accuracy: 0.7609 - val_loss: 0.6584 - val_accurac
y: 0.7574 - lr: 1.1052e-04
Epoch 11/25
3435/3435 [==============================] - 149s 43ms/step - loss: 0.6531 - accuracy: 0.7610 - val_loss: 0.6575 - val_accurac
y: 0.7578 - lr: 1.0000e-04
Epoch 12/25
3435/3435 [==============================] - 146s 42ms/step - loss: 0.6524 - accuracy: 0.7614 - val_loss: 0.6573 - val_accurac
y: 0.7573 - lr: 9.0484e-05
Epoch 13/25
3435/3435 [==============================] - 149s 43ms/step - loss: 0.6517 - accuracy: 0.7617 - val_loss: 0.6575 - val_accurac
y: 0.7576 - lr: 8.1873e-05
Epoch 14/25
3435/3435 [==============================] - 146s 43ms/step - loss: 0.6512 - accuracy: 0.7618 - val_loss: 0.6566 - val_accurac
y: 0.7578 - lr: 7.4082e-05
Epoch 15/25
3435/3435 [==============================] - 146s 43ms/step - loss: 0.6506 - accuracy: 0.7620 - val_loss: 0.6571 - val_accurac
y: 0.7584 - lr: 6.7032e-05
Epoch 16/25
3435/3435 [==============================] - 144s 42ms/step - loss: 0.6501 - accuracy: 0.7617 - val_loss: 0.6562 - val_accurac
y: 0.7578 - lr: 6.0653e-05
Epoch 17/25
3435/3435 [==============================] - 146s 42ms/step - loss: 0.6498 - accuracy: 0.7621 - val_loss: 0.6566 - val_accurac
y: 0.7580 - lr: 5.4881e-05
Epoch 18/25
3435/3435 [==============================] - 146s 42ms/step - loss: 0.6494 - accuracy: 0.7620 - val_loss: 0.6564 - val_accurac
y: 0.7588 - lr: 4.9659e-05
```

Figure 10: Output of training a tuned LSTM neural network.

Accuracy and loss curves:

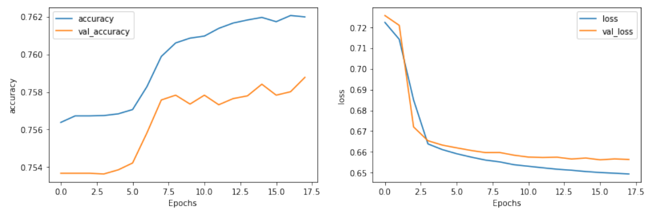


Figure 11: Learning curve of training a tuned LSTM neural network.

6.4 Observations

1. The learning curves are better than the previous configuration with validation accuracy = 75.9%
2. The validation accuracy is fluctuating over different epochs but increasing towards the last epoch.
3. Early stopping criteria is met after training for Epoch 18.

7 Conclusion and Future works

After implementing the LSTM and DNN models and playing around with the parameters in order to find the best fit, we have come to the conclusion that resulting accuracy for LSTM with higher patience and an added learning rate scheduler gives same accuracy as a DNN with similar tunining of the hyperparameters
Previous works include [1], where Bobby Nguy used 2-layer LSTM on bi-gram BoW to reach to an accuracy score of 65%. In [2], Nishit Shresthaand Fatma Nasoz achieved an accuracy score of 81.29% using deep learning RNNs on paragraph vectors. And in [3] Tan *et al...* used LSTM with Glove Vectors to reach a testing accuracy of 65.6%.
One thing that can be done is using a pre-trained embedding model (*like* gensim's word2vec) to achieve a better accuracy score, as such libraries are known to perform a better job at word correlation.

References

[1] Bobby Nguy. Evaluate Helpfulness in Amazon Reviews Using Deep Learning.

[2] Nishit Shresthaand Fatma Nasoz DEEP LEARNING SENTIMENT ANALYSIS OF AMAZON.COM REVIEWS AND RATINGS.

[3] Wanliang Tan, Xinyu Wang, Xinyu Xu Sentiment Analysis for Amazon Reviews