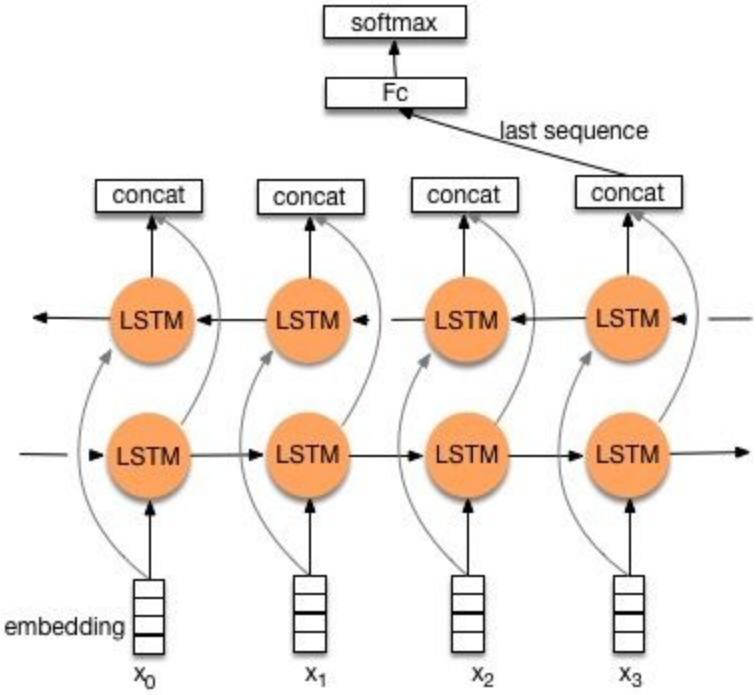
Artificial Intelligence : Lab 1

Sentiment analysis using LSTM



INTRODUCTION

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They work tremendously well on a large variety of problems, and are now widely used.
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

Sentiment Analysis: the process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral.

THE PROBLEM STATEMENT

- The provided dataset comprised of the positive and negative reviews for the mobile phones in amazon website, the reviews were in two parts: text and summary .
- Using the summary as the sequence input for our model, we were suppose to build an LSTM based model to classify the reviews as positive or negative.
- Using 3000 as sample size in the initial phase and increment the size for further verification and enhancement.
- Use pretrained GloVe embeddings.

PRE-PROCESSING THE DATASET

We considered the summary and the label.

We used Keras itself for processing the summary(text preprocessing component of the library).

Processing:

1. Tokenized the summaries.
    a. Removed all special characters.
2. Created a vocabulary.
    a. Assigned unique values to each word in the vocabulary to create a sequence of word to integer mapping.
3. Converted Every sample of both train set and test set to a sequence.
4. Made sure each sequence is of same length by padding 0's.
5. Converted the labels to one hot encoding.

THE MODEL

We used a simple LSTM model having:

1. An Embedding layer
2. LSTM layer having 256 units
3. An output dense sigmoid layer calculating the probability of whether the summary is positive or negative.

```
model = Sequential()
model.add(Embedding(len(vocab) + 1, 100, weights=[embedding_matrix],input_length=pad_length, trainable=False))
model.add(LSTM(256,dropout=0.5,recurrent_dropout=0.2))
#model.add(LSTM(128,dropout=0.2,return_sequences=False))
model.add(Dense(2,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy', precision, recall])
```

Analysis

After training the model for 3000 samples as was asked we ended up with the following metrics :

```
Accuracy: 87.45%
Precision: 83.05%
Recall: 89.91%
F-score: 86.34%
```

Example Predictions and their outcome:

review = "Not that great. Don't buy"

```
[[ 0.99462402  0.00451099]]
Class :  0
Negative
```

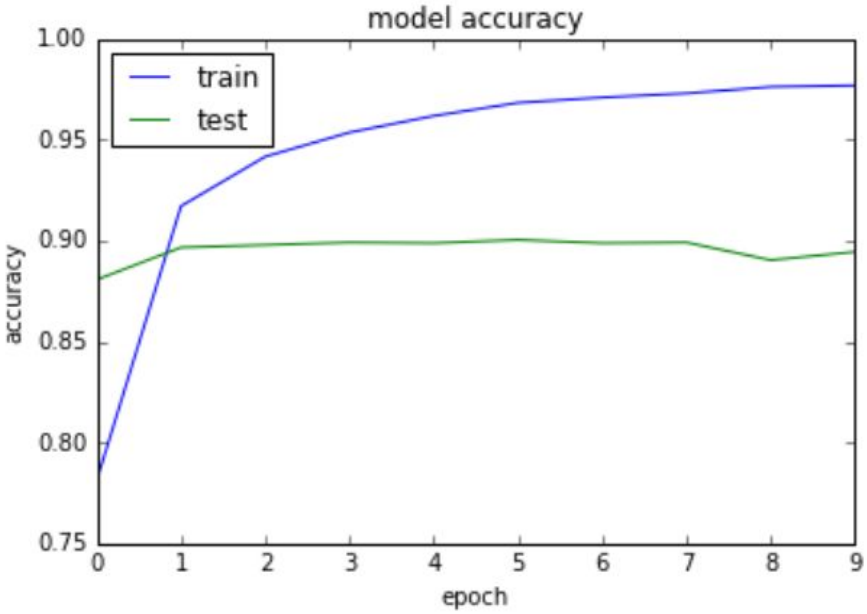
review = 'good one'

```
[[ 0.02560367  0.97549033]]
Class :  1
Positive
```

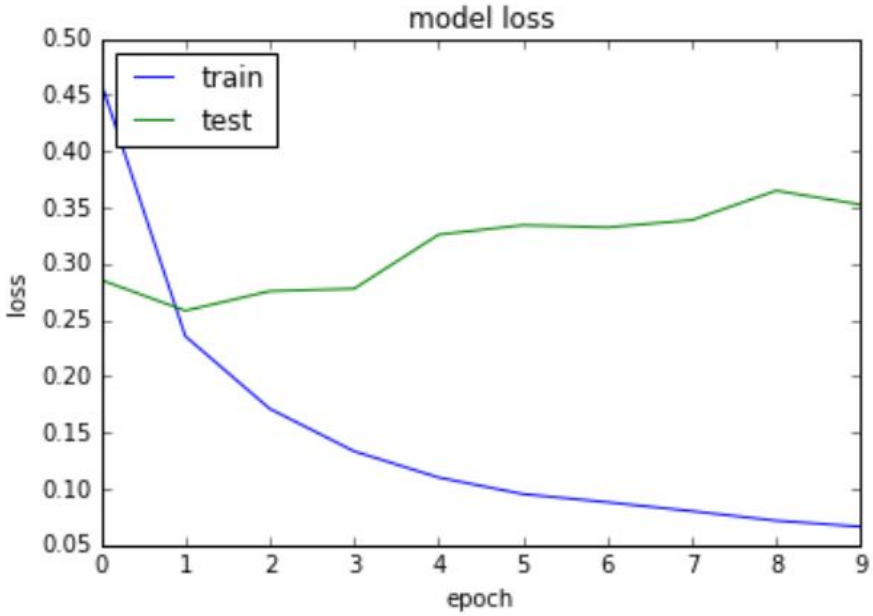
Summary of the model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 31, 100)           577400
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               365568
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 514
=================================================================
Total params: 943,482
Trainable params: 366,082
Non-trainable params: 577,400
_________________________________________________________________
```

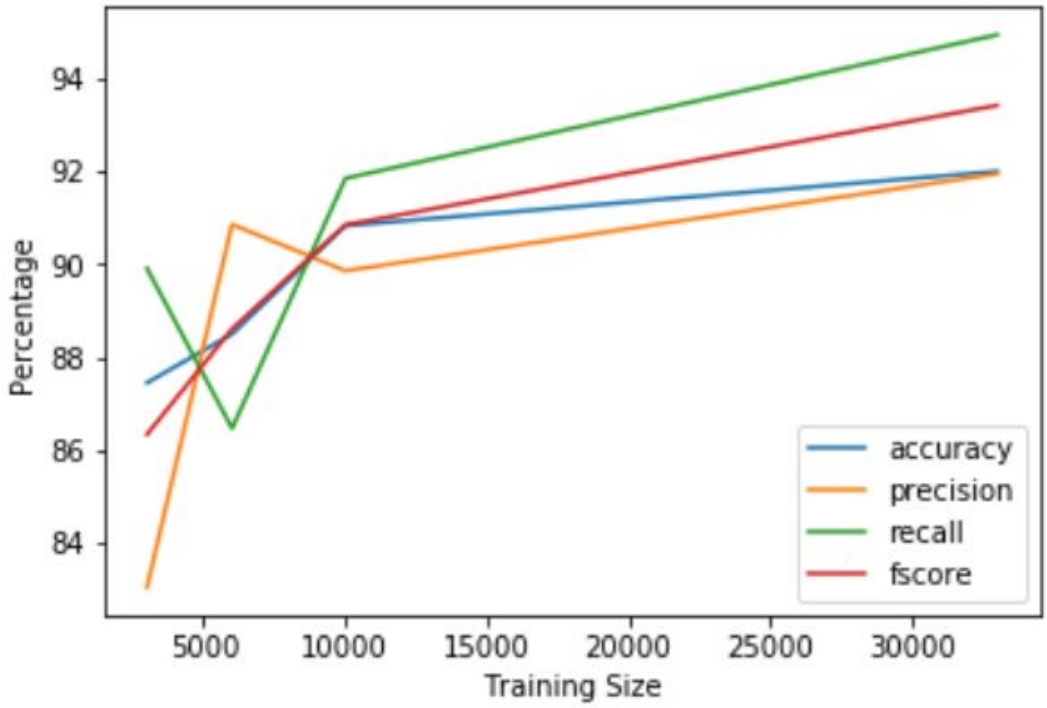
VISUALISATIONS



Here we can see that the test accuracy is fairly constant



Here we can see although the loss for training decreases over time, the test loss sees a slight upside hence showing that the model is overfitting a bit,

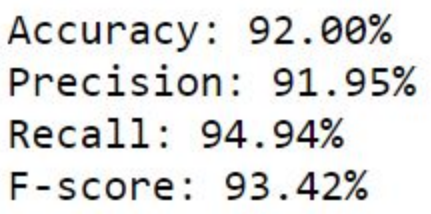


This plot shows how the percentages of accuracy, precision, recall and fscore vary as we vary the size of the training data used for training the model.

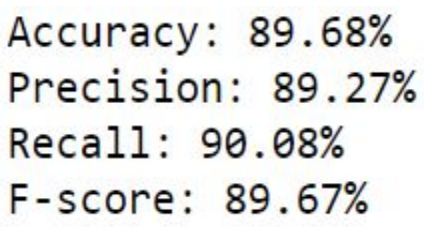
NOTE: For the above model we didn't use pre-trained word embeddings, Instead we trained and obtained the weights of the embedding layer.

OBSERVATIONS

- The pretrained GloVe embeddings gave a poorer accuracy than training the embedding layer specifically for our dataset through the Embedding layer of the Sequential model API provided by Keras.



Without GloVe



With Glove

[The highest metrics recorded is the shown above(left side) using around 20k positive examples and 13k negative examples....(i.e., a total of 33k samples) using Embedding layer of the Sequential model API provided by Keras.]

- Increasing the training dataset clearly results in better performance, but the increase in accuracy gradually flattens as we go on increasing the size as is evident from the above graph.
- As expected, from the first graph, the model is overfitting(slightly). Increasing the dropout and reducing the number of neurons did not help that much.

THE SQUAD

★ Sanketh Rangreji ---- 01FB15ECS267
★ Shahid Ikram ---- 01FB15ECS274
★ Sondhi Nishant ---- 01FB15ECS298
★ Sumanth Rao ---- 01FB15ECS314