

## Shreya Roy, SR-14996, RTP Credit

1. We have a total of 550152 sentence pairs in the training dataset. Each sentence is treated as a document. I calculated the TF/IDF for every unique word and document (Sentence) pair. So, the vector corresponding to each sentence contains the TF/IDF value (for the sentence/document) for every unique word present in the documents set. Initially, I checked the result after converting every sentence pair in the training data into a **3-dimensional vector which is obtained computing the Cosine similarity, Euclidean distance and Manhattan distance between the two vectors (obtained from TF/IDF value) corresponding to the respective sentence pairs. This gives the accuracy of .4536** in the validation set. As it is a 3 class classification problem, this model is performing better than any random classifier [random classifier will have accuracy around  $100/3$  percent = 33.333%]. So this simple logistic regressor which uses this 3 dimensional extracted feature [Cosine similarity, Euclidean distance and Manhattan distance between two vectors representing the respective sentences] is doing better than a random classifier.

Next, I tried using the 2 vectors for a pair of sentences (**concatenated the two vectors**) to train my logistic regressor [Scikit-learn library I used for logistic regression]. As the number of unique words is too many ( **Around 32000 in this set of documents after eliminating the stop words and converting to lower cases** ), this time I used only the useful words with very high document frequency or extremely low document frequency. I checked the performance of the model using different document frequency to filter out the unimportant words. I present my observation here:

Document frequency	Validation Accuracy	Number of unique words (Say n) after filtering out the unimportant words based on document frequency [each document is a n dimensional vector]
min_df=.01,max_df=.8	0.43	88
min_df=.005,max_df=.9	0.47	185
min_df=.002,max_df=.95	0.5168	427
min_df=.002,max_df=.97	0.5168	427
min_df=.001,max_df=.98	0.544	769
<b>min_df=.0005,max_df=.98</b>	<b>0.5787</b>	<b>1307</b>
min_df=.0007,max_df=.98	0.557	998
min_df=.0007,max_df=.99	0.557	998
min_df=.0006,max_df=.99	0.5612	1133
min_df=.00055,max_df=.99	0.5664	1221

So, here we see there is a tradeoff between the number of unique words used in training and the validation accuracy. But, with the increase in the number of words used for training the validation accuracy improves. At some point in time, the accuracy does not increase much even if we keep using more numbers of unique words. So, here I **chose min\_df=.0005,max\_df=.98** in my final model which gives validation accuracy of **0.5787** which involved 1307 unique words. So each sentence is a vector of length 1307. **Also, on the test data it gave .5732 accuracy**

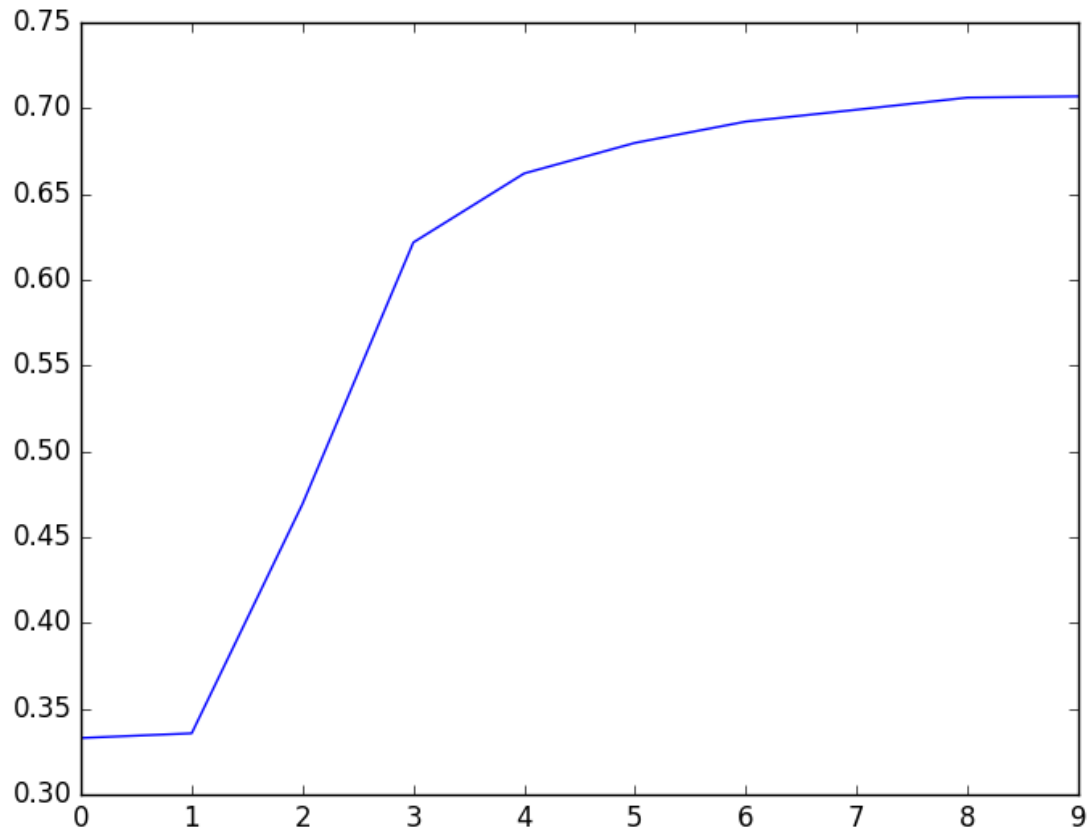
**The 3-dimensional vector which is obtained computing the Cosine, L2 and L1 distance between the two vectors (obtained from TF/IDF value) corresponding to the respective sentence pairs also give better accuracy (.4536) than the accuracy we get after filtering out the words with greater documents frequency than .01 and lesser document frequency than .8, representing each document as an 88-dimensional vector during training. This is because this 3-dimensional vector captures better features than choosing the top 88 important words using document frequency based filtration.**

2.As the problem is a multiclass classification problem, in the very beginning I checked if the training data is well class balanced or not and I found 183187 number of sentence pairs labeled as contradiction, 182764 number of sentence pairs as neutral and 183416 number of sentence pairs as entailment. So, the training data is well class balanced, hence there is no need for doing class balancing during training.

Model I. In deep models I tried with a simple architecure where I embeded the sentence as a vector by taking the average of its word embedding (300 dimensional Glove word embedding I used here) and the passed it through a simple 3 layer feed forward neural network which gave me the validation accuracy =.62 . Total number of learnable parameters were very less and the validation accuracy was also slightly better than the TF/IDF feature engineering model. In the feature engineering method [as used in the Ist problem] Logistic Regressor acts as a single layer Neural Network where the input dimension vector in the problem was used 1307\*2 (as concatenated two sentences). Here the input dimension was 600[as I used 300 dimensional word to vec embedding for each word and take the average of them to represent a sentence and then concatenated two sentences]. So, here number of features used to represent a pair of sentences is 600 dimensional. Although, the feature space dimension [to represent a pair of sentence is less than the earlier case, but still the model performed better as it is a 3 layer neural network where the logistic regressor works as a single layer neural network. So, this 3 layer neural network has more capacity to represent a function than the logistic regressor.

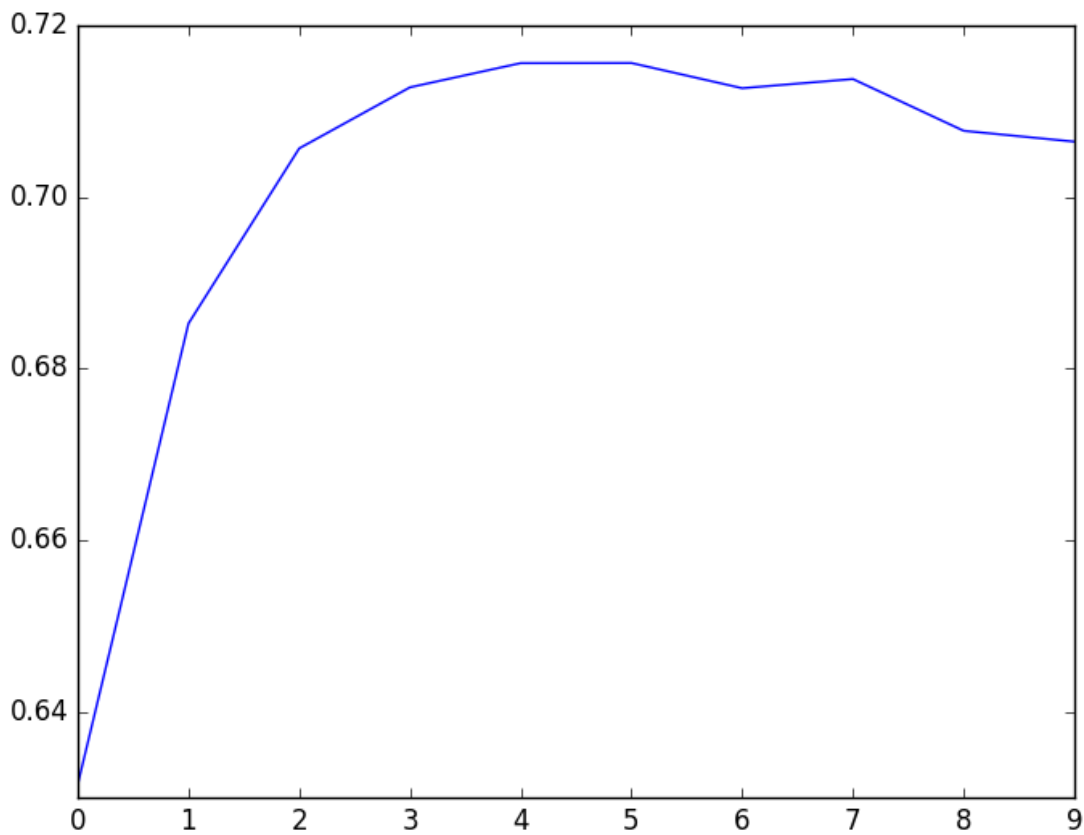
Model II. Next I tried with 100 dimensional Glove word embedding, then passing the embedding to a single layer LSTM of output dimension 512, and again adding a Softmax layer for 3 class classification. This gave me the validation accuracy =.69. Total number of learnable parameters were **1,256,963**.

Model III. Next I tried with 100 dimensional Glove word embedding, then passing the embedding to a single layer Bidirectional LSTM of output dimension 512, and again adding a Softmax layer for 3 class classification. This gave me the validation accuracy =.71 after 10 iterations Total number of learnable parameters were **2,513,923**.



*Illustration 1: Model III. Validation accuracy vs number of epochs*

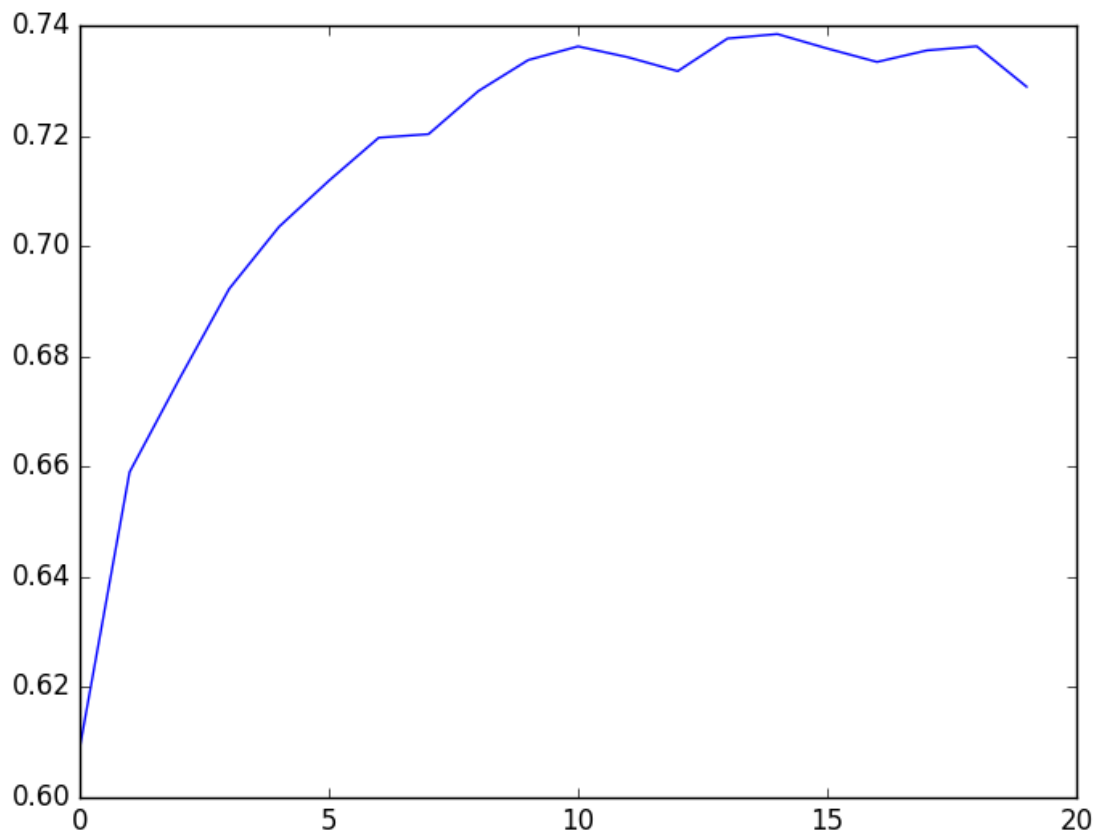
Model IV. Next I tried with 100 dimensional Glove word embedding, then passing the embedding to a single layer LSTM of output dimension 512, and again adding a Softmax layer for 3 class classification. But, this time I trained the Word embedding too, after initializing the embedding matrix with the pretrained Glove weight matrix. This gave me the validation accuracy =.716 after 4 iterations and then started overfitting. Total number of learnable parameters were **5,802,823**. So, for single layer BiLSTM model training the embedding layer did not give any improvement in the validation accuracy over using the pretrained 100 dimensional word vector, rather number of trainable parameters increased unnecessarily. The reason could be the pretrained glove word vectors is good enough for the vocabulary in this snli dataset.



*Illustration 2: Model IV. validation accuracy vs number of epochs*

Model V. Next I tried with 100 dimensional Glove word embedding, then passing the embedding to a double layer Bidirectional LSTM of output dimension 512, and then a Dropout of .2 followed by a 512 dimensional Dense layer and another dropout followed by another dense layer of same dimension and the Dropout of .2 and, then finally adding the Softmax layer for 3 class classification. I used pretrained 100 dimensional Glove weight vectors. This gave me the validation accuracy = .7205 and then it started overfitting, hence I stopped training after 10 iterations. Total number of learnable parameters were **8,809,475**.

Model VI. Next I tried with 200 dimensional Glove word embedding, then passing the embedding to a double layer Bidirectional LSTM of output dimension 512, and then a Dropout of .2 followed by a 512 dimensional Dense layer and another dropout followed by another dense layer of same dimension and the Dropout of .2 and, then finally adding the Softmax layer for 3 class classification. I used pretrained 200 dimensional Glove weight vectors. This gave me the validation accuracy = .738 after 10 iterations and then it started overfitting, hence I saved the best model received after 10<sup>th</sup> iteration. Total number of learnable parameters were **10,004,995**.

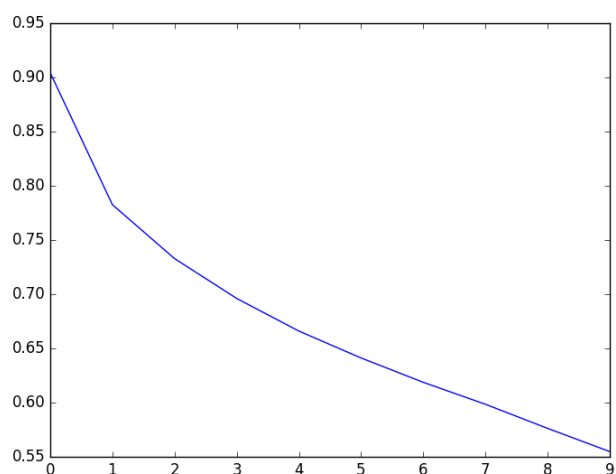


*Illustration 3: Model VI. Validation Accuracy vs Number of epochs*

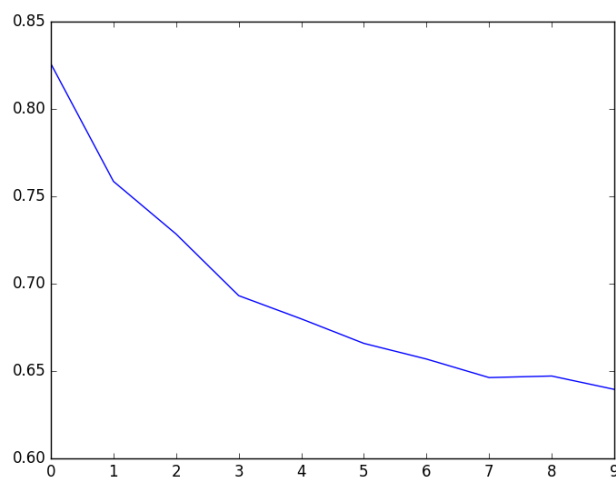
Model VII. Also, I trained 300 dimensional Glove word embedding, then passing the embedding to a double layer Bidirectional LSTM of output dimension 512, and then a Dropout of .2 followed by a 512 dimensional Dense layer and another dropout followed by another dense layer of same dimension and the Dropout of .2 and, then finally adding the Softmax layer for 3 class classification. I used pretrained 300 dimensional Glove weight vectors. This gave me the validation accuracy = .74 and then it started overfitting, hence I stopped training after 10 iterations. Total number of learnable parameters were **10,414,595**.

Also, another important observation is with the increase in model capacity/complexity I had to decrease the learning rate from .001 to .0001. [for model V, VI]

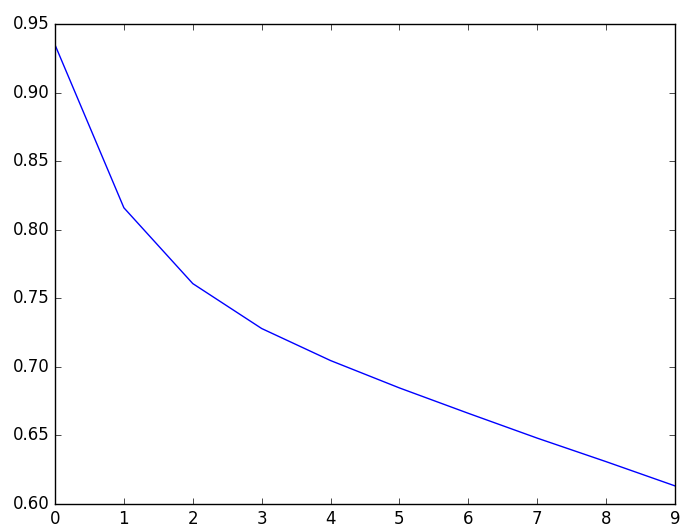
**Model V and model VII Comparision in respect to training and validation loss: As we see with**



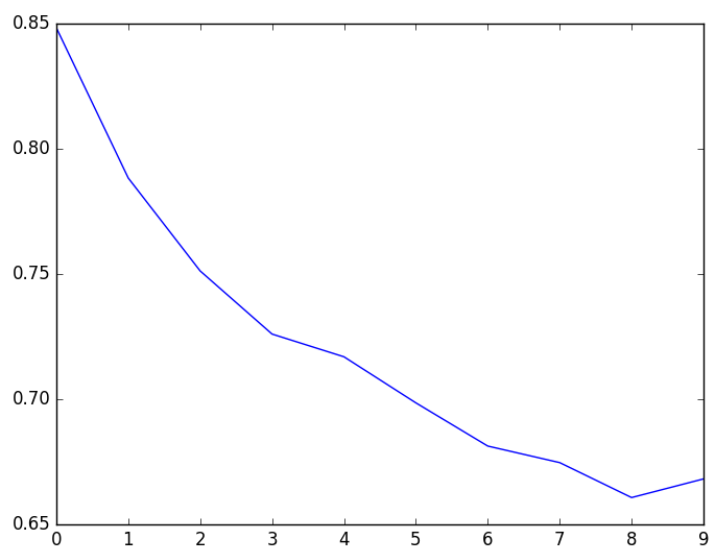
*Illustration 1: Model VII. Training loss vs number of epoch with 300 dimensional word embedding model*



*Illustration 2: Model VII. Val loss vs number of epoch with 300 di ensional embedding*



*Illustration 3: Model V. Training loss vs number of epochs with 100 dimensional embedding*



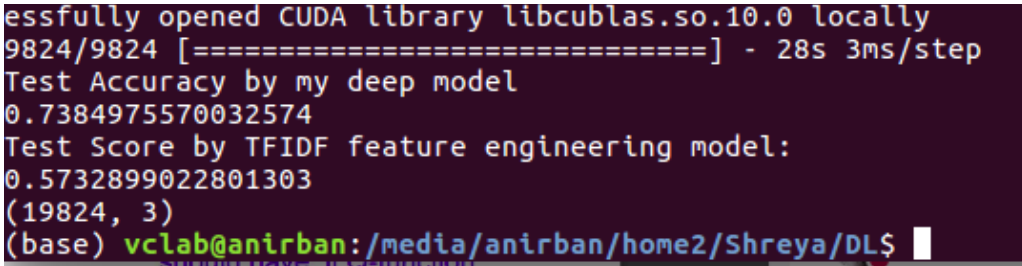
*Illustration 4: Model V. val loss vs number of epoch with 100 dimensional embedding*

**300 dimensional embedding the model performs slightly better than the embedding being used of dimension 100. Except for the word embedding dimension rest of the architectures are similar in model V and VII.**

ModelVIII. Next I tried with 300 dimensional Glove word embedding, then passing the embedding to a double layer Bidirectional LSTM of output dimension 512, and then a Dropout of .2 followed by a 512 dimensional Dense layer and another dropout followed by another dense layer of same dimension and the Dropout of .2 and, then finally adding the Softmax layer for 3 class classification. But, this time I trained the Word embedding too, after initializing the embedding eight matrix with the pretrained Glove weight matrix. This gave me the validation accuracy =.76 after 10 iterations, and then it started overfitting, hence I stopped training after 10 iterations. Total number of learnable parameters were **20,281,295** which is very high(almost double in terms of number of learnable parameters) as compared to the previous two models which gave .74 (approx ) validation accuracy which is not very less than this model.

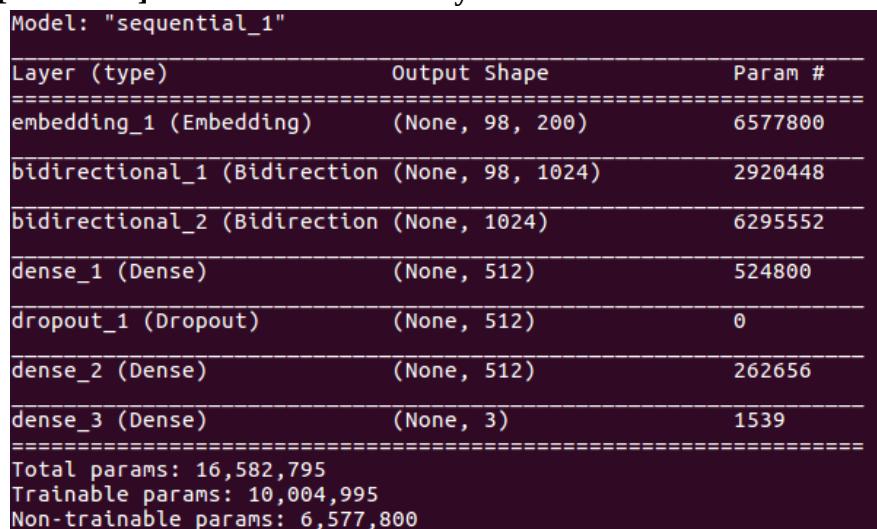
As, we can see there is a tradeoff between model complexity and model's performance, hence we preferred the previous model over this.

**So, Model V [100 dimensional pre trained embedding ],Model VI [200 dimensional pre trained embedding ] ,model VII[300 dimensional pretrained embedding , and model VIII[300 dimensional pretrained embedding with trainable= True] give validation accuracy 0.7205, 0.739 ,0.74 and 0.76 respectively. The number of learnable parameters for these 4 models are respectively 8,809,475 ; 10,004,995 ; 10,414,595 and 20,281,295 . So, as the model complexity/number of learnable parameters increase the validation accuracy increases. More complex models lead to more training time. So, keeping the time of training in mind I decided to use model VI [200 dimensional pre trained embedding ] as my final deep model which gave validation accuracy 73.8% and on the test set it gave 73.85% accuracy**

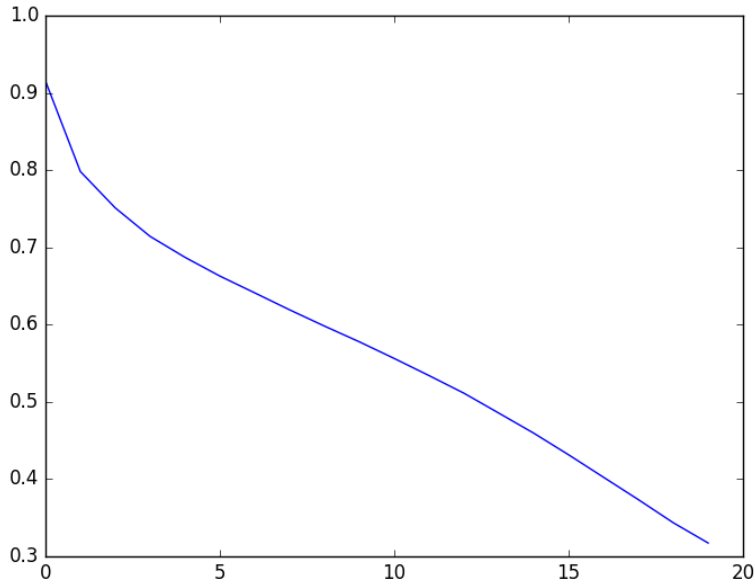
Test Accuracy: The image is a terminal window screenshot with a dark background and light-colored text. It shows the output of a test run. The first line says 'Successfully opened CUDA library libcublas.so.10.0 locally'. The second line shows '9824/9824 [=====] - 28s 3ms/step'. The third line is 'Test Accuracy by my deep model' followed by the value '0.7384975570032574'. The fourth line is 'Test Score by TFIDF feature engineering model:' followed by '0.5732899022801303'. The fifth line shows '(19824, 3)'. The sixth line shows '(base) vclab@anirban: /media/anirban/home2/Shreya/DL\$'.

**Test Accuracy by my final TF/IDF feature engineering model and deep model: .5733 and .7384 respectively.**

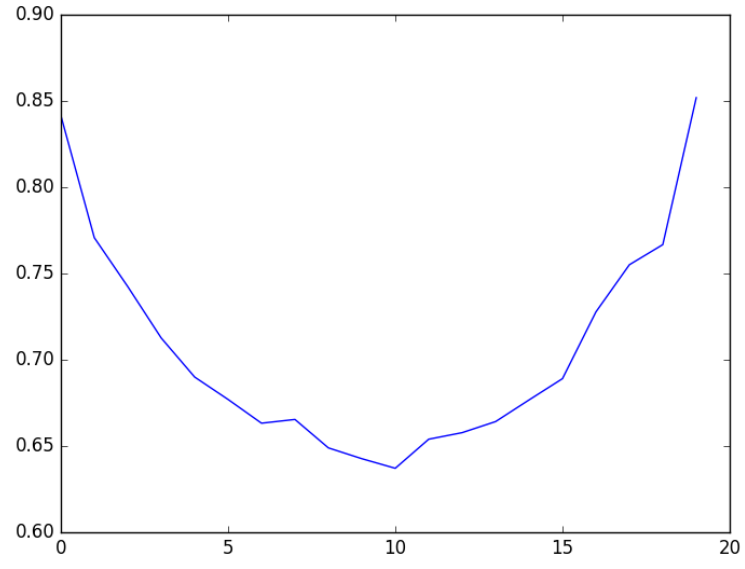
**My Final Model[Model VI] Architecture Summary:**

The image shows a terminal window with a dark background and light-colored text. It displays the architecture summary of a model named 'sequential\_1'. The table has three columns: 'Layer (type)', 'Output Shape', and 'Param #'. The layers listed are: embedding\_1 (Embedding), bidirectional\_1 (Bidirectional), bidirectional\_2 (Bidirectional), dense\_1 (Dense), dropout\_1 (Dropout), dense\_2 (Dense), dense\_3 (Dense), and a final layer with 3 units. Below the table, it shows the total number of parameters (16,582,795), trainable parameters (10,004,995), and non-trainable parameters (6,577,800).

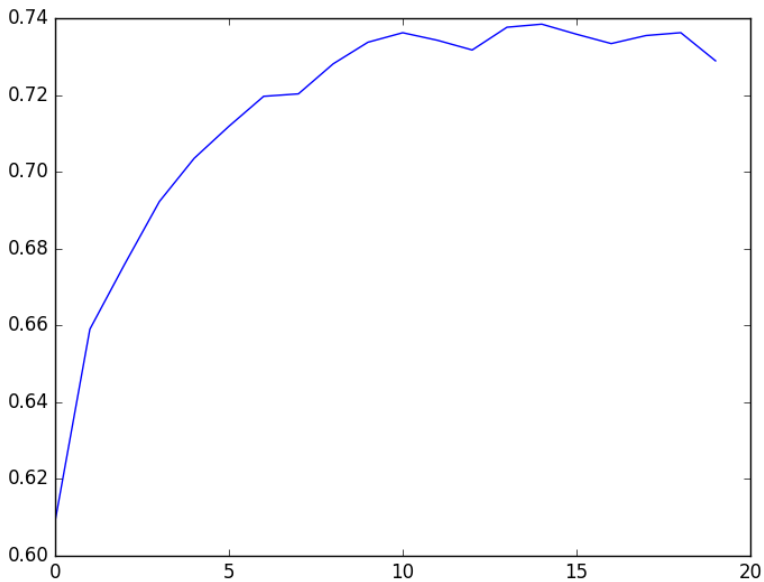
Model: "sequential_1"		
Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 98, 200)	6577800
bidirectional_1 (Bidirectional)	(None, 98, 1024)	2920448
bidirectional_2 (Bidirectional)	(None, 1024)	6295552
dense_1 (Dense)	(None, 512)	524800
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dense_3 (Dense)	(None, 3)	1539
Total params: 16,582,795		
Trainable params: 10,004,995		
Non-trainable params: 6,577,800		



*Illustration 4: Training Loss vs number of iteration for final deep model [Model VI]*



*Illustration 5: Validation loss vs number of epoch for the deep model [Model VI]. I saved and used the best model after 10<sup>th</sup> iteration for testing, I just continued the training till 20 iterations to show how the validation loss increases during overfitting but the training loss on the left picture still decreases.*



*Illustration 6: Validation Accuracy vs number of epochs for model VI. so we can see after 10 iterations overfitting starts.*