CSA 250 : Deep Learning Project 3

Report

Dhaval Parmar

April 15, 2020

❖ Logistic Regression :

- I have implemented logistic regression based classifier using scikit learn to provide class of test data as inference.
- I am preprocessing data by converting to lower case so model does not consider same word with capital and small letter as different word, then removing punctuation mark and removing stop words which does not carry much information and using lemmatizer to convert words to its base words. I have tried using stemming but lemmatizer perform better because it convert to base word.
- For certain training data Gold Label is given '-' due to no clear conclusion from annotator label. For such data I am using annotator label and if more than one label has same highest frequency I am considering it as different data with those labels.
- I am using TfidfVectorizer to find tfidf for both sentence. Then I am concatenating them to use as feature for LR model.
- I am getting accuracy of 64.37% on test data. I am neglecting test data with Gold Label as '-' while measuring accuracy to avoid ambiguity.
- Accuracy is some what less because tfidf is related to frequency of words in sentence rather than relation between words meaning so for natural language inference tfidf feature is not a good choice.

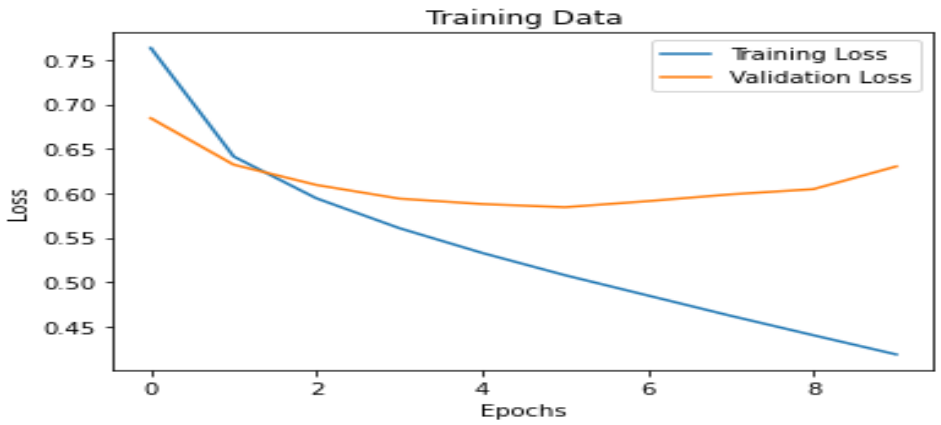
❖ LSTM Deep Neural Network:

- I have tried implementing deep network with LSTM with sentence fusion for NLI .
- Preprocessing of train and test data is done in the same way as for LR.
- I am using pre-trained Glove data to convert word to vector. I was initially using 100d vectors to represent words. Then I tried using 300d vectors because due to higher dimensional space vector can be distributed in better way according to relation between words. As per expectation 300d vectors perform better than 100d vectors.
- I am using tokenizer to convert word to sequence. So that this sequence can be used to generate embedding matrix using glove vector. So this generated sequence is given as input to embedding layer which convert it to 300d vector according to embedding matrix.
- Then I am giving embedded sequence of 300d vector corresponding to sentences to very important LSTM layer which is responsible for extracting important information from sequential data.
- I have tried using both unidirectional LSTM and also bidirectional LSTM. As bidirectional LSTM consider both past and future words gives better performance. So due to better accuracy I am using Bidirectional LSTM. I am using two separate LSTM network for both sentences.
- Then I have tried applying attention mechanism according to time step. I was using TimeDistributed wrapper with Dense(1) layer to get learnt weights for every time step. Then using sigmoid activation function to convert weights in the range of [0,1]. Then multiplying this weights with LSTM output and summing them to get final output vector. But attention model is not helping here.

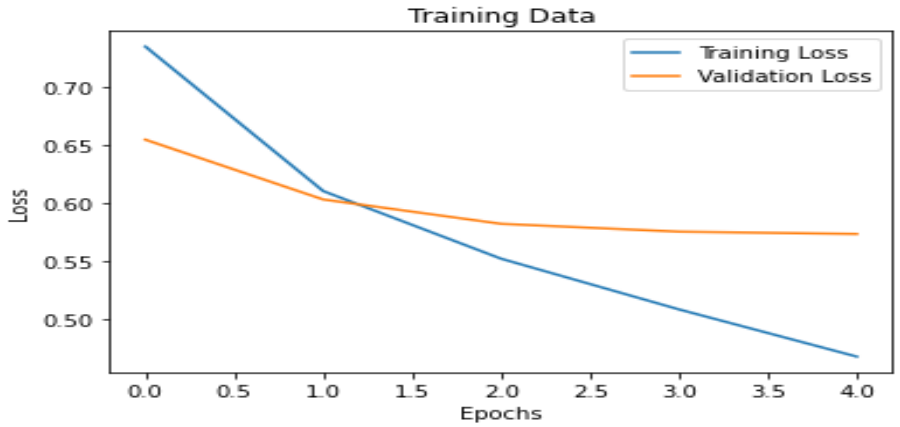
- I have tried different fusion of this two output vectors of LSTM. I have started with just concatenating two out put of two LSTM. Then I tried concatenating subtraction and multiplication of two sentences with those two sentences to extract relation between them because glove embedding is in such a way that arithmetic operation on vectors gives interesting relation between words. Then I tried concatenating this two output with subtraction, multiplication, subtraction with reverse of second tensor and multiplication with reverse of second tensor to get relation in better way and as expected it is performing better than other two fusion.
- Then I am applying this concatenated tensor to dense layers so that model can learn parameter to give best inference. I have tried different architecture of this dense layers. I have started with just 1 Dense layer and went up to 3 with different number of units. Finally I am using 3 dense layers with 1800, 512 and 128 units respectively. I am using dropout layers in between dense layer with 0.5 dropout rate.
- At the end using soft max layer to get probability distribution of three labels.
- I am using adam optimizer to get benefit of adaptive learning rate. I have also tried with RMSprop that is without momentum but does not perform well. I am using accuracy to measure performance of model.
- I have used validation split of 80 and 20 percentage of snli train data.
- I am using model checkpoint to save best model.
- Regarding data in both train and test JSON data set many gold labels are '-' that is due to no clear conclusion from annotator labels. So I tried using label smoothing i.e. if 2 neutral, 2 entailment and 1 contradiction is given as annotator label it can be represented as [0.4 0.4 0.2] according to probability distribution. So I am using smoothed label for all the data point according to annotator label. For using vector as output label I am using 'categorical_crossentropy' as loss function.
- There where many changes I have done parallelly regarding model architecture, glove vector dimension, loss function, optimizer and other things. Many of the time accuracy was not changing significantly. So I am not presenting output of every model but here I am presenting result of some of the models.

- Some of the trial models are given below.

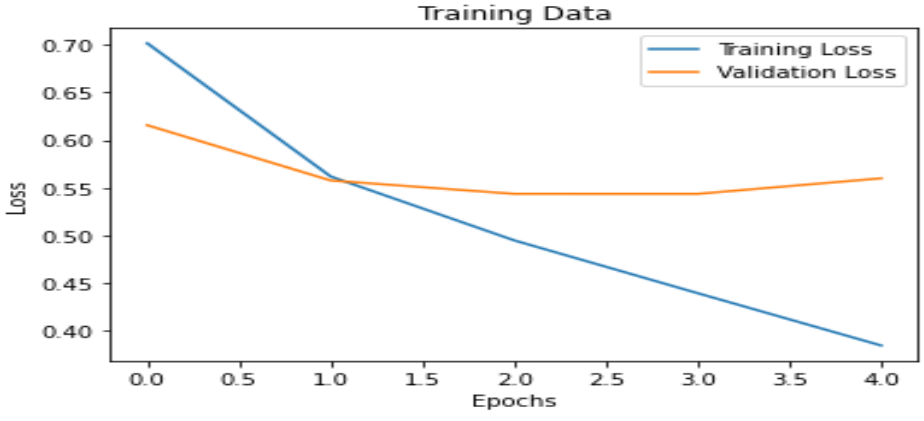
LSTM : Unidirectional	Feature dimension : 100d	Training Accuracy : 80.61
Layer : 600,128,3	optimizer : adam	Test Accuracy : 77.89



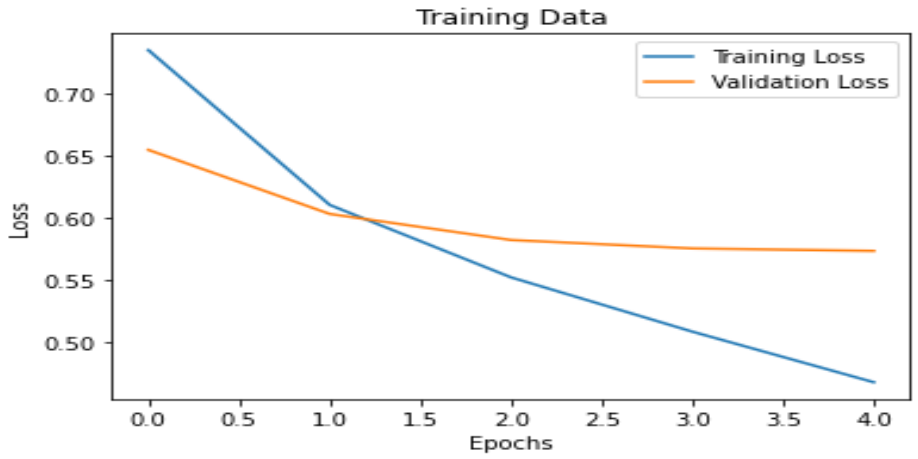
LSTM : Bidirectional	Feature dimension : 100d	Training Accuracy : 81.48
Layer : 600,256,128,3	optimizer : adam	Test Accuracy : 78.53



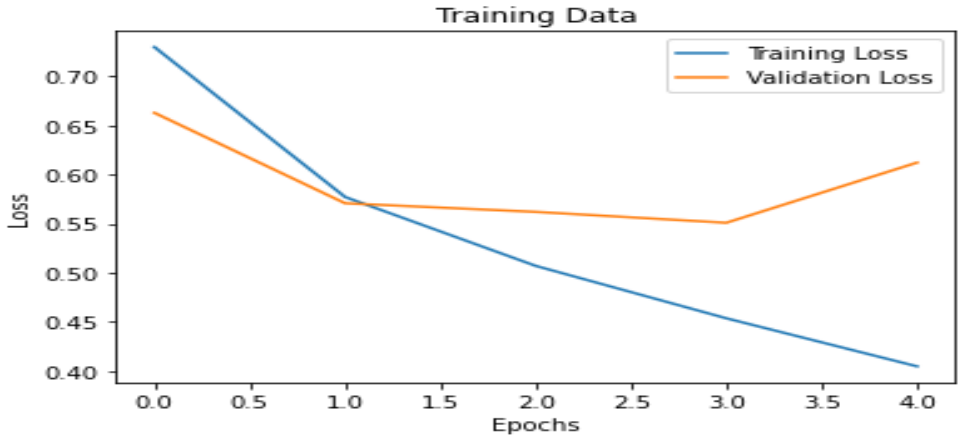
LSTM : Unidirectional	Feature dimension : 300d	Training Accuracy : 84.63
Layer : 1800,1800,128,3	optimizer : adam	Test Accuracy : 79.83



LSTM : Unidirectional	Feature dimension : 300d	Training Accuracy : 85.15
Layer : 1800,512,128,3	optimizer : adam	Test Accuracy : 80.01

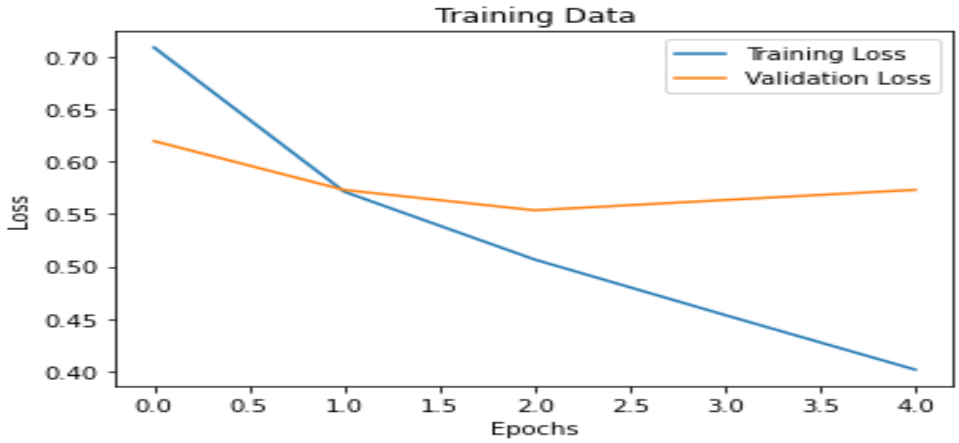


LSTM : Unidirectional	Feature dimension : 300d	Training Accuracy : 84.47
Layer : 1800,512,128,3	optimizer : RMSprop	Test Accuracy : 79.72

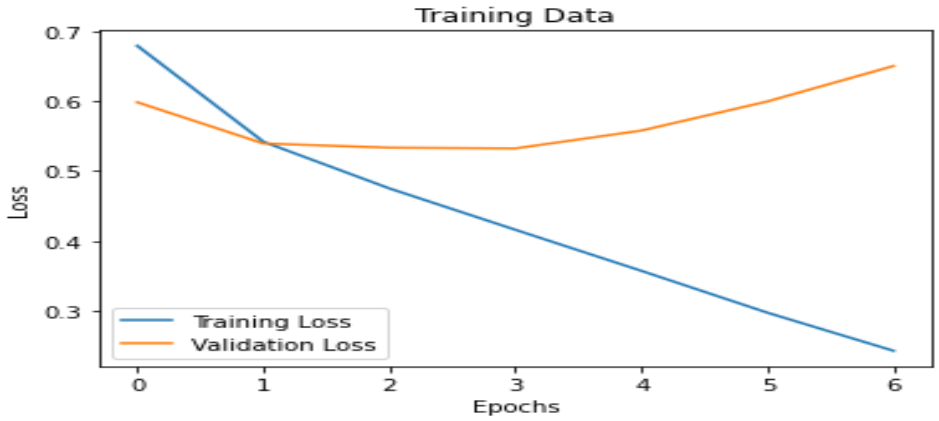


- Now onwards using annotator label and label smoothing.

LSTM : Unidirectional	Feature dimension : 300d	Training Accuracy : 85.17
Layer : 1800,512,128,3	optimizer : adam	Test Accuracy : 80.25

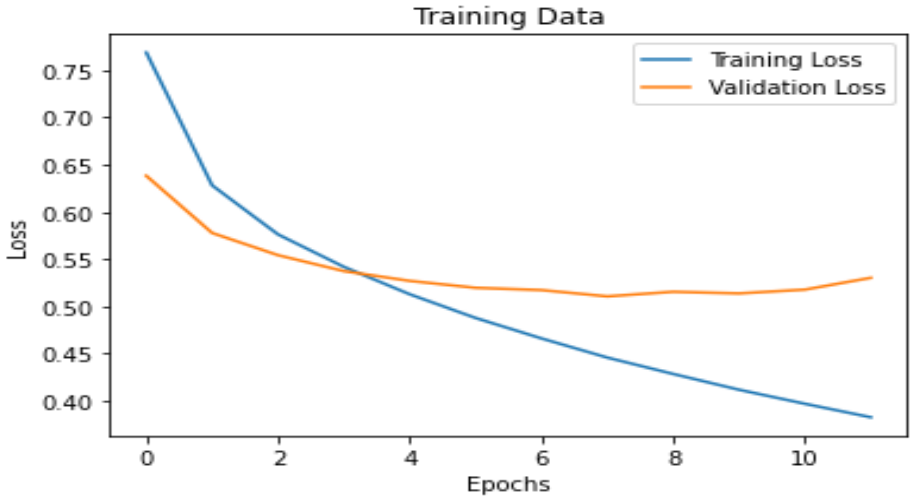


LSTM : Bidirectional	Feature dimension : 300d	Training Accuracy : 84.58
Layer : 1800,512,128,3	optimizer : adam	Test Accuracy : 80.77



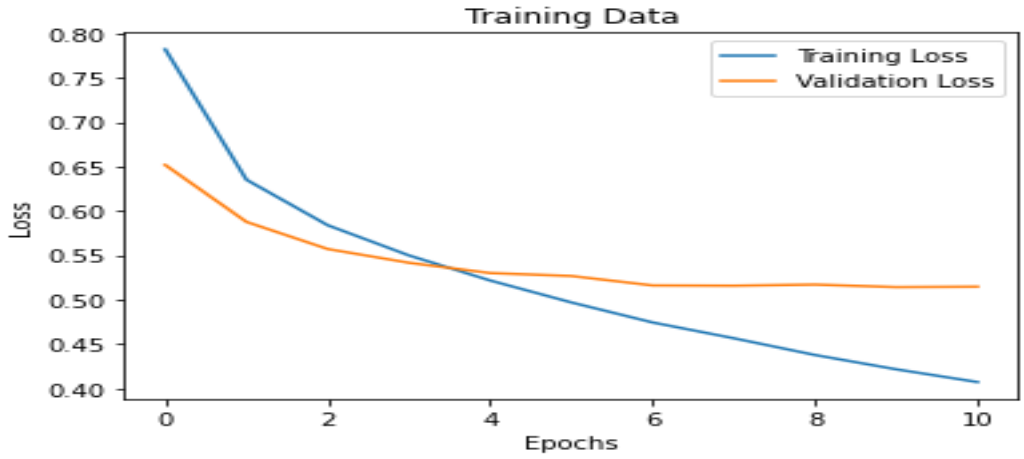
- Using Dropout Layers

LSTM : Bidirectional	Feature dimension : 300d	Training Accuracy : 84.89
Layer : 1800,512,128,3	optimizer : adam	Test Accuracy : **82.17**



- Using Attention model

LSTM : Bidirectional	Feature dimension : 300d	Training Accuracy : 84.78
Layer : 1800,512,128,3	optimizer : adam	Test Accuracy : 81.98



- Surprisingly model performance with attention is quite near to model without attention even poor than that. So I am submitting model trained without attention. In both model validation accuracy start decreasing after 9th epoch. I have kept commented attention model I have used for reference in 'Train_LSTM' file.
- Model summary for final model without attention is given below.

```
_____________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
==============================================================================================
input_3 (InputLayer)         [(None, None)]            0
_____________________________________________________________________________________________
input_4 (InputLayer)         [(None, None)]            0
_____________________________________________________________________________________________
embedding_1 (Embedding)      (None, None, 300)         9368100     input_3[0][0]
                                                                   input_4[0][0]
_____________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 600)            1442400     embedding_1[0][0]
                                                                   embedding_1[1][0]
_____________________________________________________________________________________________
lambda_7 (Lambda)            (None, 600)               0           bidirectional_1[1][0]
_____________________________________________________________________________________________
lambda_5 (Lambda)            (None, 600)               0           bidirectional_1[0][0]
                                                                   bidirectional_1[1][0]
_____________________________________________________________________________________________
lambda_6 (Lambda)            (None, 600)               0           bidirectional_1[0][0]
                                                                   bidirectional_1[1][0]
_____________________________________________________________________________________________
lambda_8 (Lambda)            (None, 600)               0           bidirectional_1[0][0]
                                                                   lambda_7[0][0]
_____________________________________________________________________________________________
lambda_9 (Lambda)            (None, 600)               0           bidirectional_1[0][0]
                                                                   lambda_7[0][0]
_____________________________________________________________________________________________
```

concatenate_1 (Concatenate) (None, 3600) 0 bidirectional_1[0][0]
lambda_5[0][0]
lambda_6[0][0]
lambda_8[0][0]
lambda_9[0][0]
bidirectional_1[1][0]

---

dense_4 (Dense) (None, 1800) 6481800 concatenate_1[0][0]

---

dropout_2 (Dropout) (None, 1800) 0 dense_4[0][0]

---

dense_5 (Dense) (None, 512) 922112 dropout_2[0][0]

---

dropout_3 (Dropout) (None, 512) 0 dense_5[0][0]

---

dense_6 (Dense) (None, 128) 65664 dropout_3[0][0]

---

dense_7 (Dense) (None, 3) 387 dense_6[0][0]

==========

Total params: 18,280,463

Trainable params: 8,912,363

Non-trainable params: 9,368,100

Train Loss : 84.89

Test Loss : **82.17**

NOTE : Due to maximum limit of 100mb for file on Github I am uploading everything other than LSTM model on Github while model can be accessed using drive link https://drive.google.com/open?id=1EIlx0RBCOog5YpTIamTGguxUzVQ_m5Cb. I have used masking to ignore padded value which was giving error initially on colab while running on GPU and suggesting to use tensorflow 1.14. So it is requested If you are trying to train LSTM model with GPU and face same problem please use tensorflow 1.14.