Deep Learning Project - 3

Sentence pre-processing:

Stop words removal:

1. Stop words (like the, a, an, is) are not important features for classification because these words are present in all the sentences. So, these words must be removed.
2. I have used nltk pre-processing module to remove these words.

Lemmatization:

1. Words present in the sentence are not in its root form. They are present in the sentence based on its tense or plural form. These words are unable recognize and hence unable to convert into vector by using embedding matrix.
2. I have tried stemming and lemmatizer using porter Stemmer and Wordnet Lemmatizer of nltk library. In Stemming it truncates the words only (like leaves to leav, planning to plann), but Lemmatizer returns to its root form.
3. I tried both these approaches in logistic regression and achieved accuracy of 62.85% in Porter Stemmer and 64.44% using Wordnet Lemmatizer. So, in Deep Learning I have used only Wordnet Lemmatizer.

Remove punctuation:

1. Sentence contains punctuation characters (like: "!#$%&'()*+,-./:;?@[\]^_`{|}~)). Which must be removed because these punctuation characters along with words create problem in word to vector conversion using pretrained model in recognizing words.
2. I have used RegexpTokenizer of nltk module which removes punctuation characters.

a) Logistic regression model:

1. Target labels pre-processing:

In this model I have simply converted labels into number zero, one and two, here zero for 'neutral', one for 'contradiction' and two for 'entailment'.

1. Tf-idf features:

- In this model pre-processed sentences are converted using Tfidfvectorizer of scikit learn library. These converted vectorized sentences are then joined using scipy sparse hstack function because these sentence one and two vectors are sparse vectors.
- Now these sentences are in the form of TfIdf feature vector, along with target labels, which are directly used in LogisticRegression of scikit learn.
- Sentences of testing these models are also transformed using same vectorizer which are used for training sentences to convert into tfidf feature vector.

	Accuracy
With Porter stemmer	62.85%
With Wordnet Lemmatizer	64.44%

- TfIdf is not better feature for entailment because it gives less accuracy in test data as well as intuitively it does not perform well because words frequency features are not related to entailment.

b) Deep Learning Model:

Tokenization:

- Tokenizer is used to convert sentences into sequence of integers which allows to convert word into vector using embedding matrix in the embedding layer.
- Tokenizer assigns integer by turning each text into a sequence of integers (each integer being the index of a token in a dictionary) and then converts sentences into sequence of integer. So, both premise and hypothesis are tokenized using Tokenizer of tensorflow keras.

Padding:

- Each sentence in the dataset is not of same length. Now to pass the sentence into the layer one must make each sentence of same length because input layer cannot be of variable size.
- Solution to this problem is to add zeros at the end of the sentences which are having less no of words based on the sentence of maximum length. So, for that padding function is used on tokenized sentence 1 and sentence 2.

Embedding Matrix:

- Tokenizer has assigned numbers to the words based on the words present in the whole corpus. Here these integers are not any features which can capture statistics of cooccurrence of words which can be related to its entailment.
- Now in order to capture these statistics and give more dimension so that model can learn in more dimension regarding sentence entailment, Word2Vec or GloVe can be used.
- Word2Vec captures local statistics but in this problem, we require global statistics because in entailment problem test set and validation set will not contain similar sentences. Hence, I have used GloVe of 300 dimensions pretrained model in Embedding matrix which will be used in Embedding layer.

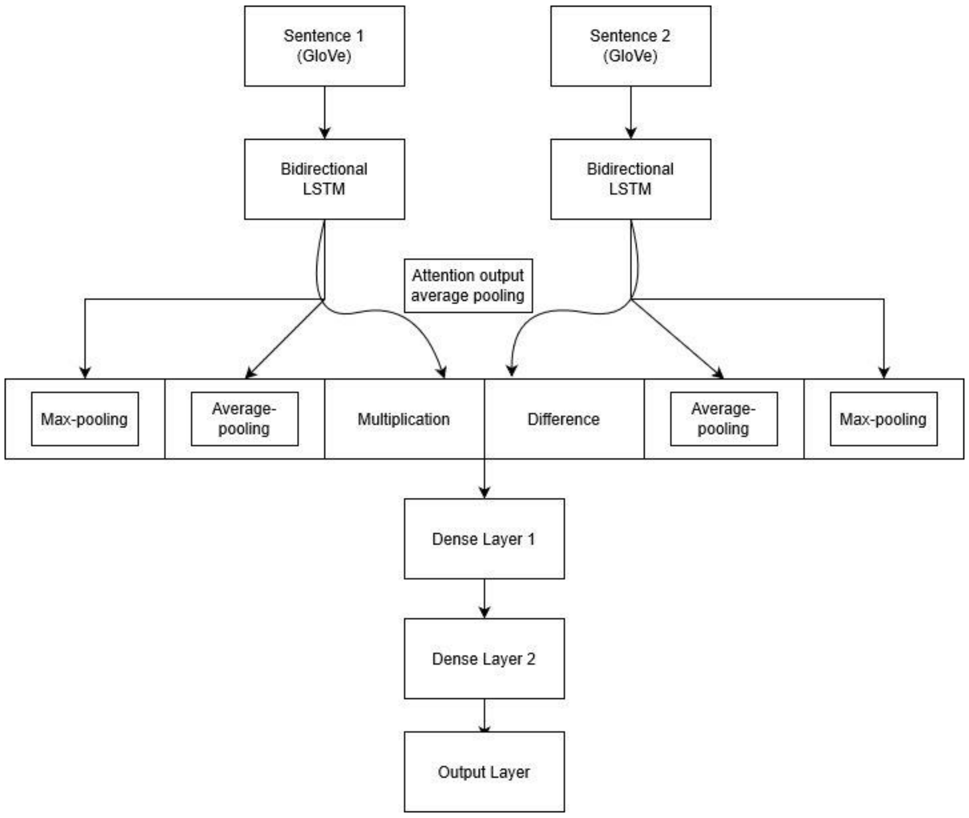
Masking:

- Zero padded sentences are directly passed to the model then model will train these zeros as features which is wrong. Masking layer can be used in this case, which will mask zeros as false and vector information as true which can be decoded by LSTM.

Target labels pre-processing:

- In this problem first I have used encoder and following which keras to categorical function which converts target labels into one hot vector. This one hot vector is used as target label.

Model:



- I have used Embedding layer to convert tokenizer output into Global Vector (GloVe). I have used Glove 6B.300D in this model. Here output of attention layer is passed through average pooling layer and then its multiplication and difference are taken as feature. Unk tokens generated randomly between [-0.05,0.05].
- Activation function for output layer: Softmax
- Loss function: categorical_crossentropy
- Optimizer: Adam (learning rate = default)

Hyper-parameter Tuning:

- No of layers:

By varying no of dense layer change in accuracy are as shown in the table below. Here I have shown maximum accuracy by varying no of nodes in each layer and dropout. Feature used are output of bidirectional LSTM context vector multiplication and subtraction (without return sequence or return state).

No of dense layers	Maximum Accuracy (%)
Zero	77.81%
One	78.17%
Two	80.85%

- One dense layer accuracy:

I have checked test accuracy by varying no of nodes in one dense layer. I have checked maximum 500 nodes because output of bidirectional LSTM is of 600 dimension2. I have shown output only for dropout of 0.2 because at 0.2 maximum accuracy achieved.

No of nodes	Dropout	Test Accuracy (%)
200	0.2	78.17
300	0.2	78.05
400	0.2	77.71
500	0.2	77.84

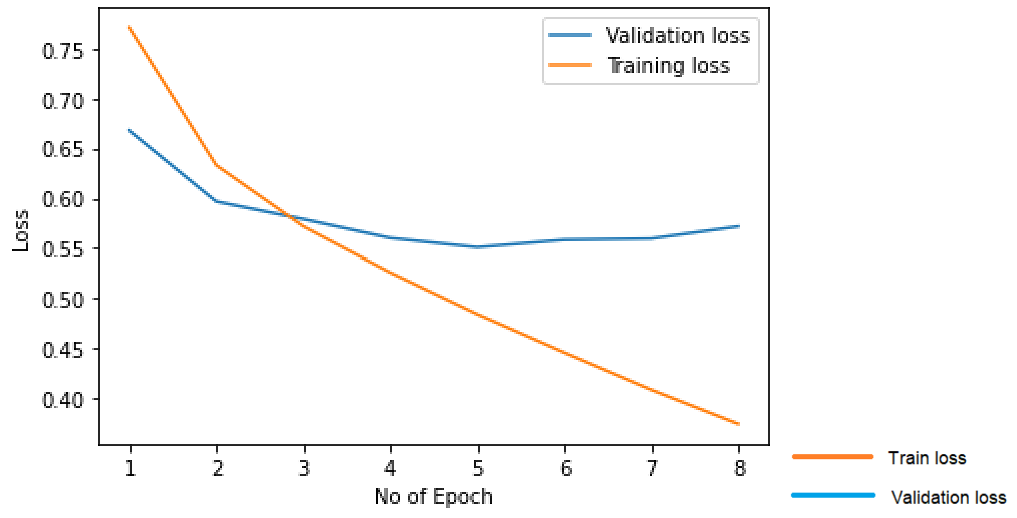
- Two dense layer accuracy:

First, I have checked maximum accuracy by taking 50 no of nodes in second layer and varying no of nodes in first layer with dropout and after that I have fixed 400 nodes in dense layer 1 and dropout 0.1 and no of nodes in second layer increases which shows maximum accuracy at no of nodes 400.

No of nodes			
Dense layer 1	Dense layer 2	Dropout (both layer)	Test Accuracy
200	50	0.1	78.76%
200	50	0.2	78.83%
200	50	0.3	78.82%
300	50	0.1	78.84%
300	50	0.2	78.70%
300	50	0.3	78.79%
400	50	0.1	79.24%
400	50	0.2	78.86%
400	50	0.3	79.16%
500	50	0.1	78.75%
500	50	0.2	79.21%
500	50	0.3	78.92%
400	200	0.1	79.60%
400	400	0.1	80.01%

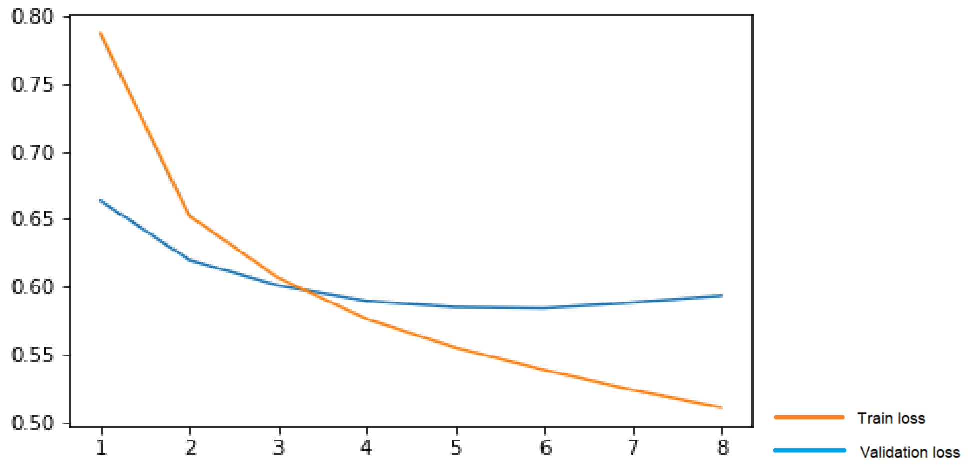
- I have used 300 units bidirectional LSTM and 0.25 dropout in both the bidirectional LSTM.

Training loss and validation loss vs no of epochs:



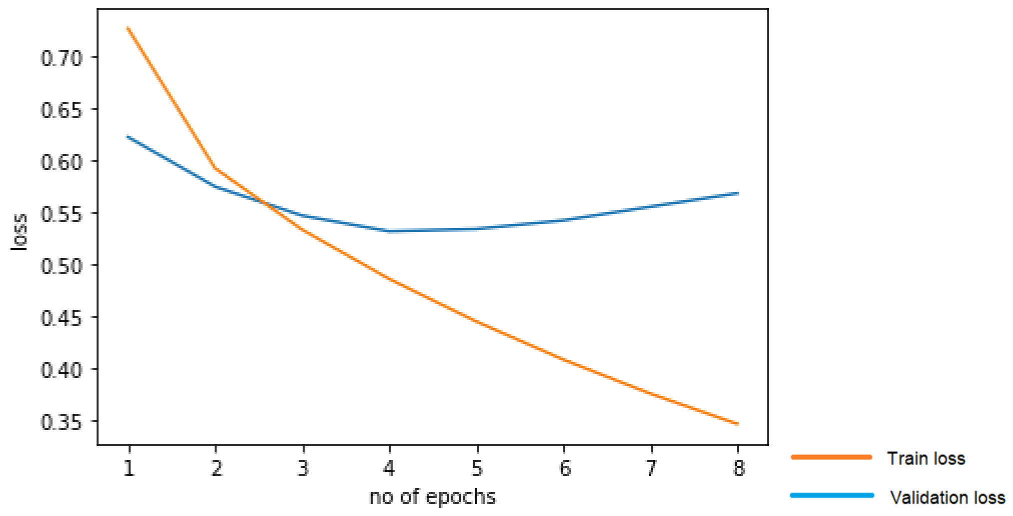
- After that I have used hyperas hyper parameter tuning library which uses random search algorithm has given parameters for dense layer1 no of nodes 300 and dropout 0.1, for dense layer2 no of nodes 300 and dropout 0.1.
- After that I have used attention mechanism with LSTM and output of attention layer of both sentence vectors are used as feature by taking multiplication and subtraction of these context vector generated by attention mechanism. Here both dense layers are used with same hyper parameter as obtained previously using hyperas. Accuracy obtained using this feature is 77.78%.

Training loss and validation loss vs no of epochs:



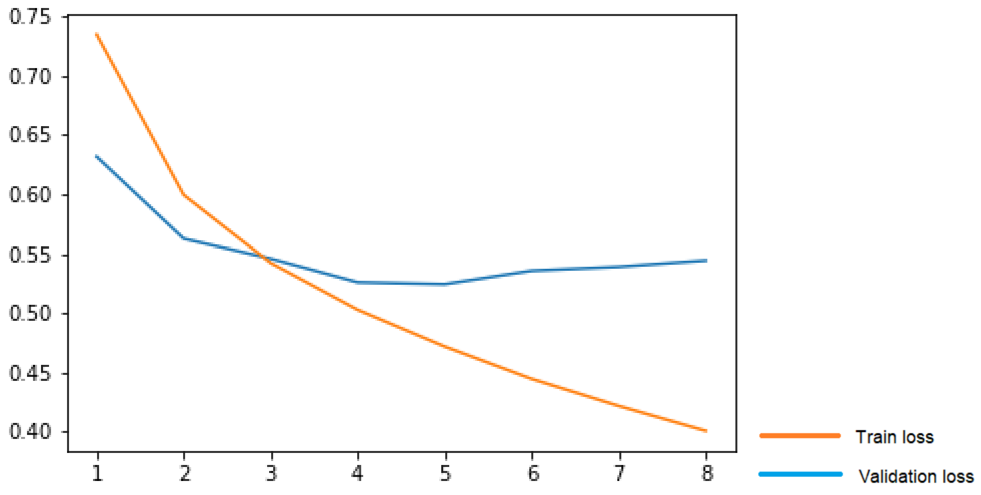
- After that I have used bidirectional LSTM with attention layer and features are multiplication and difference of sentence vector with two dense layers of 300 nodes and 0.1 dropout same as previous. Accuracy obtained using this feature is 80.45%

Training loss and validation loss vs no of epochs:



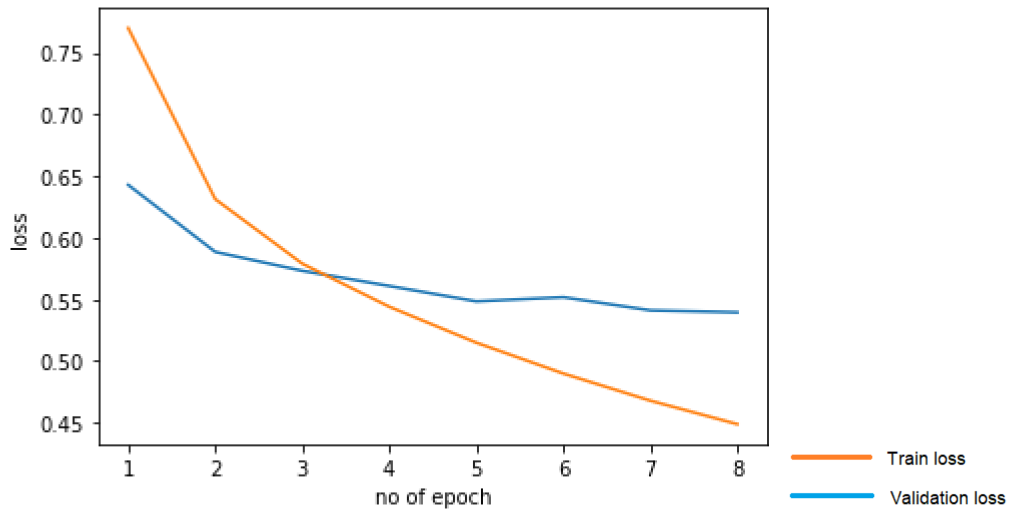
- After that I have used max-pooling and average pooling of each sentence sequence (output of bidirectional lstm) along with multiplication and difference of output of attention layer output (average pooling of attention layer) which has shown best feature representation and accuracy obtained using these features are 80.82% with ADAM optimizer.

  Training loss and validation loss vs no of epochs:

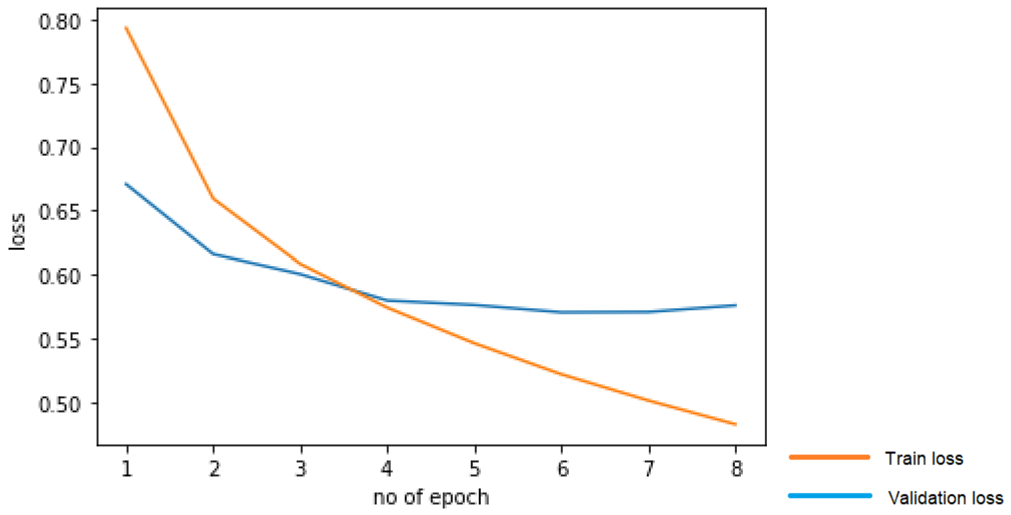


- After that I have used max-pooling of one sentence sequence (output of bidirectional LSTM) and average pooling of attention layer output and concatenated them and multiplication and difference of these features are taken as input which has shown accuracy 80.46% and in other experiment max-pooling of output of attention layers and its multiplication and difference are taken as feature, accuracy obtained was 78.26%.

Training loss and validation loss vs no of epochs:



Max-pooling of sentence sequence and average pooling of attention layer output



Max-pooling of attention layer output along with its multiplication and difference as feature

Features	Max accuracy
Multiplication and difference of output of bidirectional LSTM	80.01%
Multiplication and difference of output of attention layer with LSTM	80.45%
Multiplication and difference of output of attention layer with bidirectional LSTM concatenated with maxpooling of sentence sequences	80.46%
Multiplication and difference of output of attention layer with bidirectional LSTM concatenated with maxpooling of attention sequences	78.26%
Multiplication and difference of output of attention layer with bidirectional LSTM concatenated with max-pooling and average pooling of sentence sequences	80.82%

Note:

- I have used tensorflow 1.x for training deep learning model because tensorflow 2.x gives GPU error cuDNN Bad_Params but you can train deep learning model on cpu with tensorflow 2.x. If you want to train it on gpu then you must use tensorflow 1.x. Please check link below for the same error still have no solution:

  https://github.com/tensorflow/tensorflow/issues/33148

- I have shown accuracy only for those sentences which are not having sentence label as '-'.

- I have used GloVe.6B.300D for vector representation, categorical cross entropy as loss function in all experiments and in attention mechanism I have removed masking layer as well as UNK tokens were generated randomly between –0.05 to 0.05.