Natural Language Processing on SNLI dataset

Name: Arun kumar S

Department: UG

S.R.No: 15656

SNLI dataset : The SNLI corpus (version 1.0) is a collection of 570k human-written English sentence pairs manually labelled for balanced classification with the labels *entailment*, *contradiction*, and *neutral*, supporting the task of natural language inference (NLI), also known as recognizing textual entailment (RTE). Training set, validation set and test set have sizes respectively 550152, 10000, 10000.

Text cleaning:

In the get_sentences_and_labels(data_corpus) function, we are cleaning the given corpus to obtain the premise and hypothesis sentences along with labels. So now we have cleaned premise and hypothesis sentences along with their labels.

Text Preprocessing:

In the first model, we were supposed to use tf-idf features and implement the logistic regression using scikit learn. I have obtained the same using TfidfVectorizer of sklearn. I have also manually implemented Logistic regression without using sklearn and will discuss the results of it in the following sections.

For the second deep model specific for text, to represent the words as vectors so that we can get the semantics of the words (like similar words being together in the vector space, unrelated words being separated by large distance), I have used Word embeddings , pretrained glove model. So, I have considered all the prominent ways of text preprocessing. Now, for each of these methods of feature extraction, I need to create various Deep models specific for text like RNN, LSTM, GRU, etc.

1)Logistic Regression Classifier using tf-idf features.

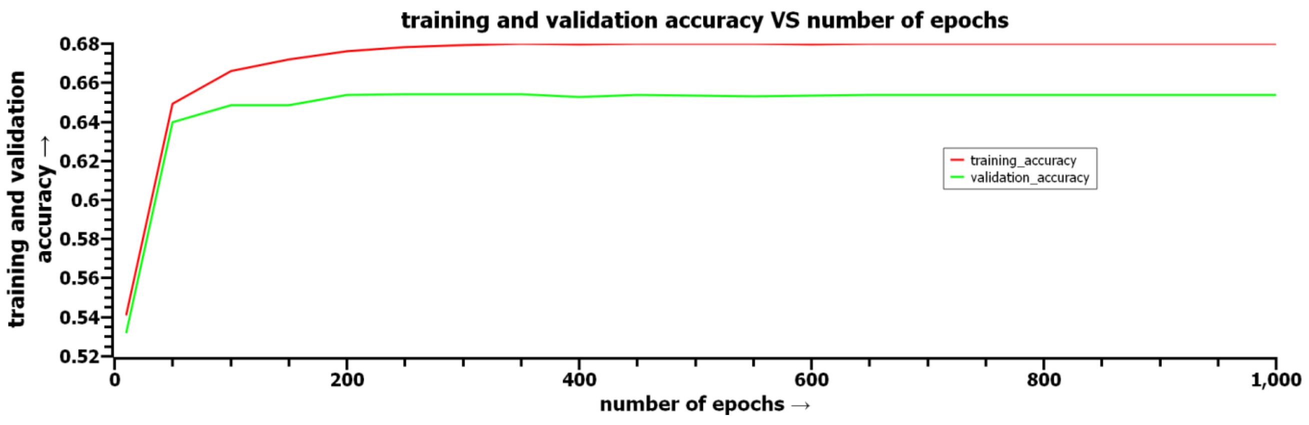
We instantiated TfidfVectorizer. This vectorizer was fitted on the training text (i.e, both the premise and the hypothesis). Then using the vectorizer we have obtained the premise and hypothesis sentences in numbers which we have concatenated to get the final input for training, validation and testing on the model. Logistic regression was implemented from scikit learn.

The hyperparameters that I could tune were very few in this model. I could find only three prominent hyperparameters that I could tune- stop_words, max_features and max_iterations(number of epochs). The following table has the accuracies that I obtained when I used different combination of hyperparameters. The accuracies were better when stop_words were set to None compared to the case when it was set to 'english'. So, I decided to keep stop_words as

None. With increase in max_iterations or the number of epochs, all the accuracies increased though the amount of increase was less as epochs went to higher and higher numbers. Since the loss function is convex, the accuracies always tend to improve with iterations in this case. So, I need to tune max_features and then use as many epochs as possible with that model. I have tried a variety of numbers for max_features and it turned out that with 25000 max_features I could get the best accuracies for the given number of epochs. Note that max_features correspond to the maximum number of words that will be considered in the dictionary by the vectorizer. Thus, with stop_words being None, max_features being 25000 and with 1000 epochs I could get the best accuracies and I have saved the same model.

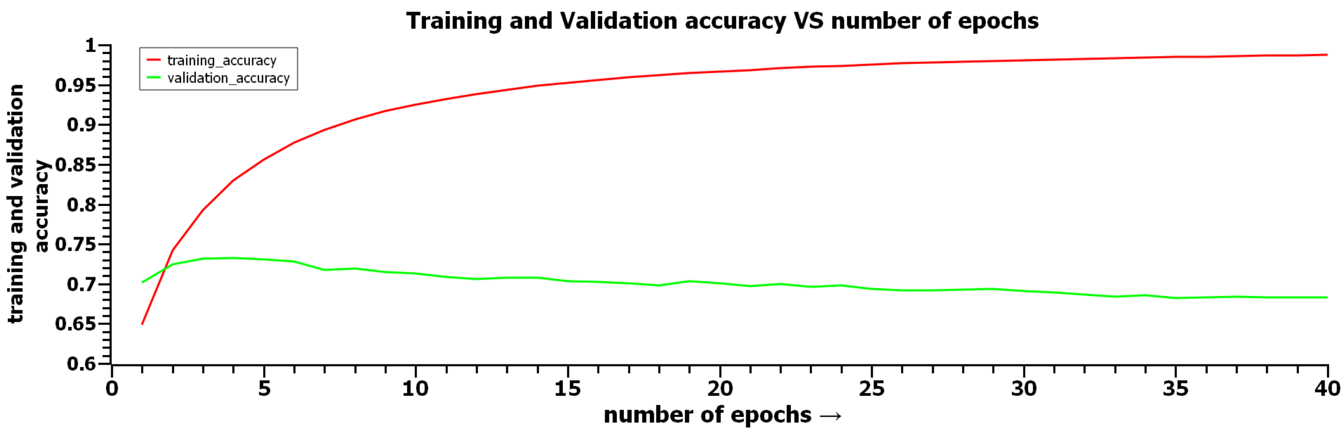
I have also plotted the training and validation accuracies with number of epochs below. It is not possible to obtain the loss when sklearn logistic regression is used. So, I could not plot loss.

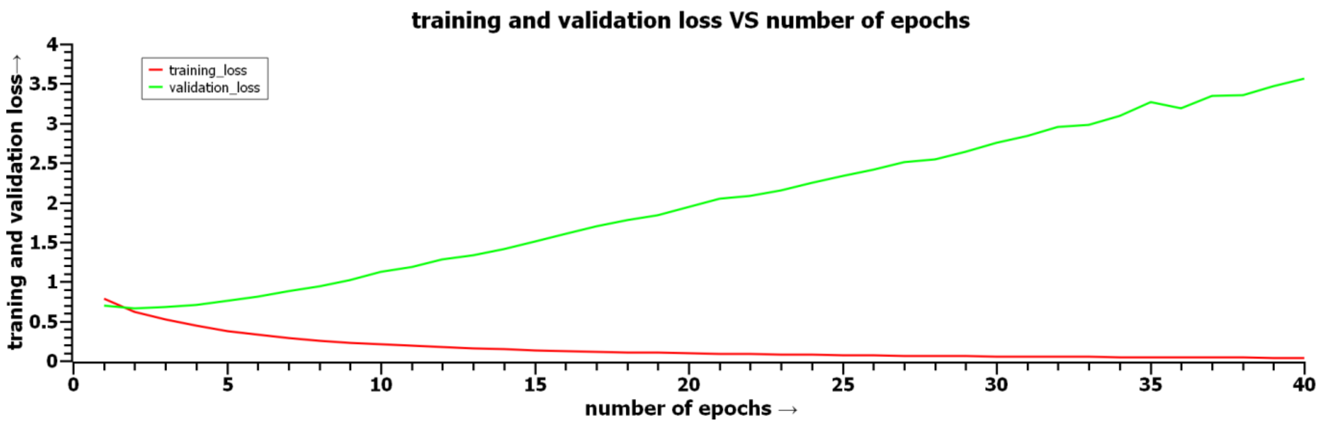
Hyperparameters	training_acc	validation_acc	test_acc
stop_words = None, max_features = None max_iterations = 100	0.6609	0.6443	0.6437
stop_words = None, max_features = None, max_iterations = 500	0.6818	0.6537	0.6512
stop_words = 'english' , max_features = None, max_iterations = 300	0.6498	0.6279	0.623
stop_words = 'english', max_features =None ,max_iterations = 500	0.652	0.6254	0.6227
stop_words = None, max_features = 28000, max_iterations = 100	0.66	0.6449	0.643
stop_words = None, max_features = 30000, max_iterations = 100	0.6633	0.6455	0.648
stop_words = None, max_features = 25000, max_iterations = 100	0.666	0.6485	0.6482
stop_words = None, max_features = 31000, max_iterations = 100	0.6603	0.6443	0.6446
stop_words = None, max_features = 30500, max_iterations = 100	0.6616	0.6443	0.6433
stop_words = None, max_features = 22000, max_iterations = 100	0.6599	0.6444	0.6471
stop_words = None, max_features = 27000, max_iterations = 100	0.6651	0.6465	0.6462
stop_words = None, max_features = 26000, max_iterations = 100	0.6619	0.6449	0.6454
stop_words = None, max_features = 25000, max_iterations = 300	0.6793	0.654	0.6507
stop_words = None, max_features = 25000, max_iterations = 500	0.6799	0.6535	0.6518
stop_words = None, max_features = 25000, max_iterations = 1000	0.6798	0.6537	0.6522



2) Manual Implementation of Logistic Regression.

There are few differences compared to the Logistic Regression using sklearn case. I have used the labels function to represent the golden_labels in numbers 0, 1 and 2(for entailment, neutral and contradiction). In the model, number of Dense layers does not make much difference as totally it is just a linear classifier and the final activation function would be softmax obviously to do the multi class classification. The loss function is sparse_categorical_crossentropy and optimizer is adam. As expected, this model also has the same set of parameters. The following table contains the accuracies I obtained with different set of hyperparameters. With similar experiments, again stop_words = None gives better result. Also, with increase in epochs, I observed that training accuracy raised high, but validation accuracy was not reducing much (comparatively it was almost stagnant). Now for max_features I again tried variety of numbers and again 25000 was giving the best result this time. So, I just had to stop at that epoch which gave the best validation accuracy. I found that with 3 epochs, validation and test accuracies were highest. So, I saved that model itself. Training accuracy always improved with epochs and I found that with 40 epochs training accuracy reached around 0.9922. This was surprising to me as the logistic regression model could so well fit to the training data (though it is overfitting) but hereafter training accuracy was obviously increasing at a much lower pace. With 100 epochs, the training accuracy was around 0.9987 and it was quite astonishing for me. Below I have also plotted the training and validation accuracies. Also, since I manually implemented the logistic regression here, I could obtain the training and validation losses as well and I have plotted them as well.

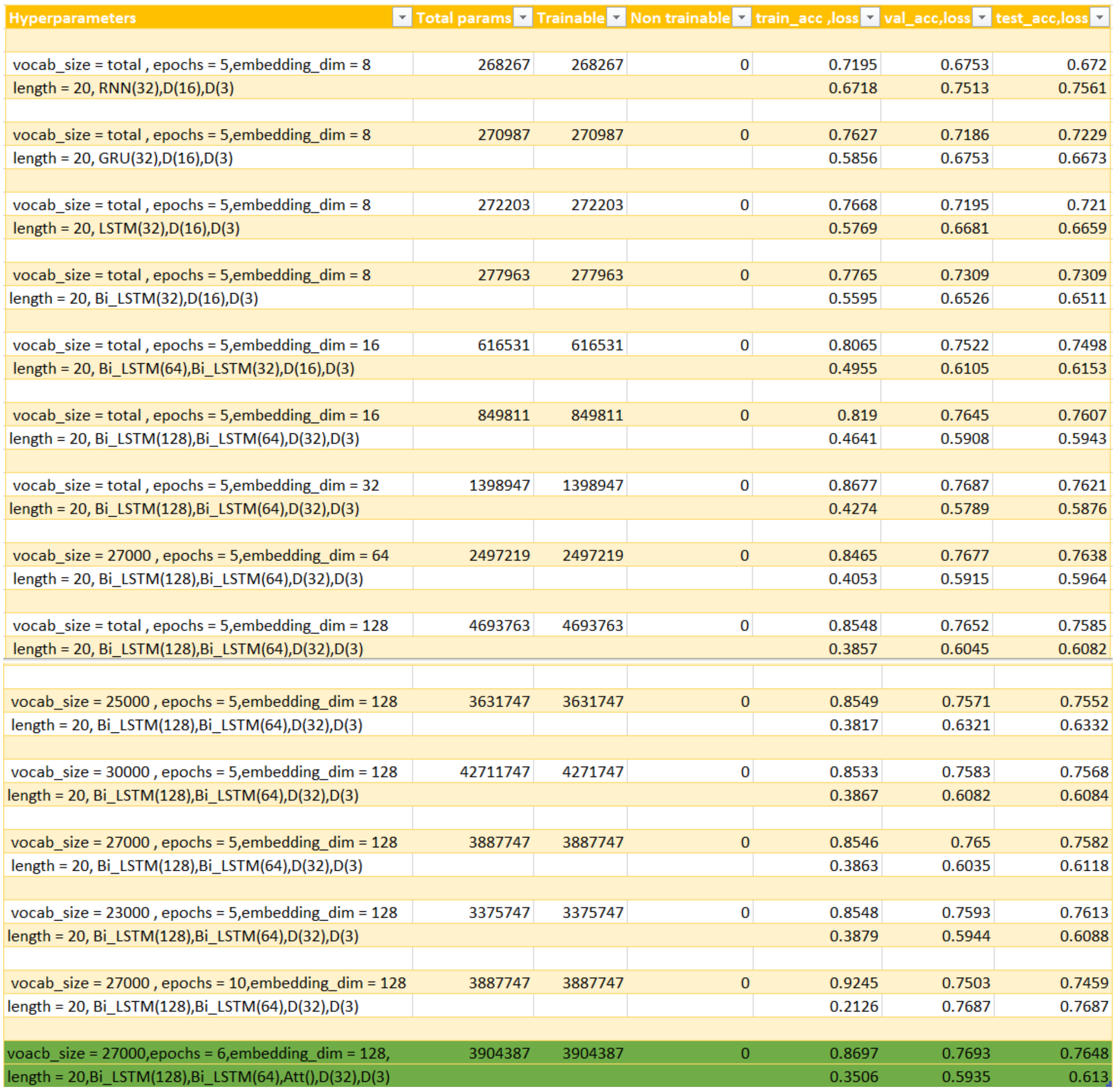




Hyperparameters	train_acc,loss	val_acc,loss	test_acc,loss
stop_words = None, max_features =None,epochs = 10	0.8377	0.7188	0.7218
D(16),D(3)	0.4324	0.7632	0.76
stop_words = None , max_features =None, epochs = 10	0.9508	0.7175	0.7198
D(64),D(16),D(3)	0.1449	1.1201	1.0969
stop_words = None , max_features =28000, epochs = 10	0.9504	0.7181	0.7188
D(64),D(16),D(3)	0.1441	1.1187	1.1098
stop_words = None , max_features =25000, epochs = 10	0.9563	0.7035	0.7074
D(64),D(16),D(3)	0.1322	1.2006	1.2422
stop_words = None , max_features =25000, epochs = 3	0.8523	0.7306	0.7283
D(64),D(16),D(3)	0.4072	0.6708	0.676
stop_words = None , max_features =30000, epochs = 10	0.9518	0.7164	0.7128
D(64),D(16),D(3)	0.1429	1.1036	1.1153
stop_words = None , max_features =23000, epochs = 10	0.9514	0.7221	0.7153
D(64),D(16),D(3)	0.1452	1.1072	1.1189
stop_words = None, max_features =25000, epochs = 3	0.8494	0.7344	0.7397
D(64),D(16),D(3)	0.4063	0.6757	0.6625
stop_words = 'english', max_features =25000, epochs = 10	0.932	0.6895	0.6842
D(64),D(16),D(3)	0.1987	1.0909	1.0895
stop_words = None, max_features = 25000,epochs = 40	0.9922	0.6813	0.6727
D(64),D(16),D(3)	0.0243	3.5003	3.5686
stop_words = None, max_features = 25000,epochs = 70	0.9976	0.6634	0.6693
D(64),D(16),D(3)	0.0081	5.6064	5.2209
stop_words = None, max_features = 25000,epochs = 100	0.9987	0.6589	0.6658
D(64),D(16),D(3)	0.0046	6.9345	6.4501

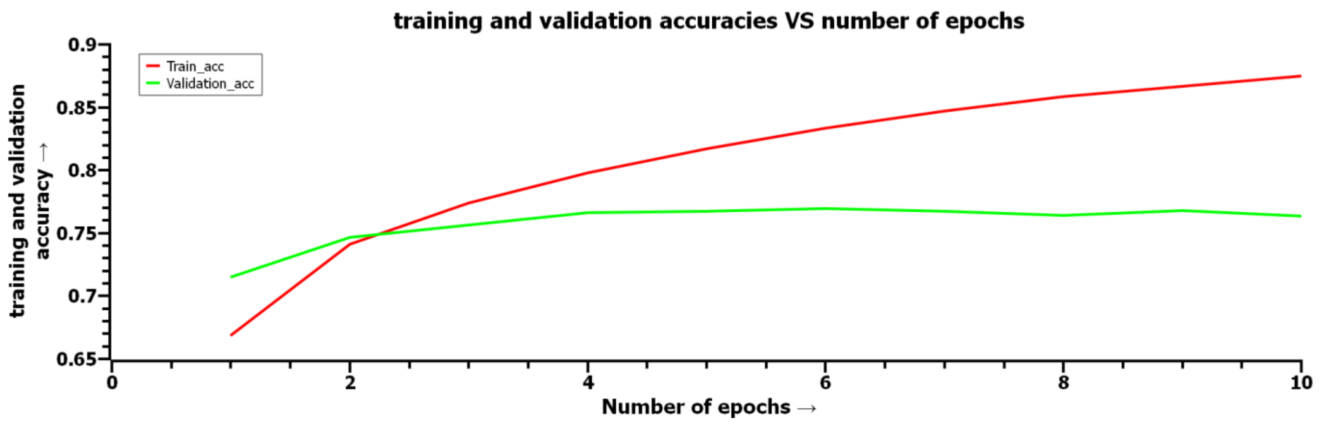
Note: D(x) is a dense layer with x units. In every architecture, above is accuracy and below is the loss.

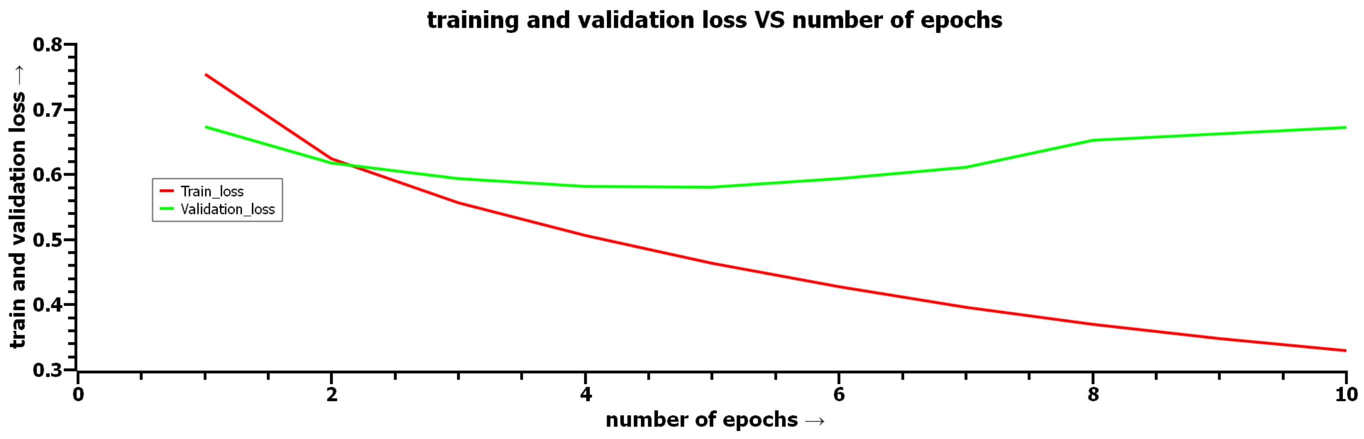
Word_embeddings_model: This is the model in which word embeddings are also being learnt by the model. The accuracies most of the time improved with embedding dimension. Because of computational issues, I could go up to 128 and so I have many of my models using 128 dimensions. Again, I have recorded most of my architectures along with results in the below table. Many of the times, a single epoch was taking around 20 minutes. Thus, I have run on just 5 epochs in many of my models. I have done several experiments many of the time focusing only one hyperparameter and decided about the hyperparameter. These experiments can be observed in the below table. By similar experiments, I found that truncated length of 20 for the sentences is giving the best result. Then, I have tried RNNs, GRUs, LSTMs, etc. I found that Bidirectional LSTMs layers are giving the best results. I also have varied the size of the vocabulary and I have obtained that vocabulary size of 27000 gave best result. Again, with many experiments with several of them being shown below, I obtained the model with embedding dimension of 128, length = 20, vocab_size =27000, two bidirectional LSTM layers, result was the best. I have also added the attention layer in the final model to obtain the highest validation accuracy of 0.7693. This is the model that I have saved though this is not the model that will be evaluated when main.py is run. With glove embeddings the results are better as explained in the next section, thus this model is in my repository. So, model with glove embeddings will be the second model for this Project.



Hyperparameters	Total params	Trainable	Non trainable	train_acc ,loss	val_acc,loss	test_acc,loss
vocab_size = total , epochs = 5,embedding_dim = 8	268267	268267	0	0.7195	0.6753	0.672
length = 20, RNN(32),D(16),D(3)				0.6718	0.7513	0.7561
vocab_size = total , epochs = 5,embedding_dim = 8	270987	270987	0	0.7627	0.7186	0.7229
length = 20, GRU(32),D(16),D(3)				0.5856	0.6753	0.6673
vocab_size = total , epochs = 5,embedding_dim = 8	272203	272203	0	0.7668	0.7195	0.721
length = 20, LSTM(32),D(16),D(3)				0.5769	0.6681	0.6659
vocab_size = total , epochs = 5,embedding_dim = 8	277963	277963	0	0.7765	0.7309	0.7309
length = 20, Bi_LSTM(32),D(16),D(3)				0.5595	0.6526	0.6511
vocab_size = total , epochs = 5,embedding_dim = 16	616531	616531	0	0.8065	0.7522	0.7498
length = 20, Bi_LSTM(64),Bi_LSTM(32),D(16),D(3)				0.4955	0.6105	0.6153
vocab_size = total , epochs = 5,embedding_dim = 16	849811	849811	0	0.819	0.7645	0.7607
length = 20, Bi_LSTM(128),Bi_LSTM(64),D(32),D(3)				0.4641	0.5908	0.5943
vocab_size = total , epochs = 5,embedding_dim = 32	1398947	1398947	0	0.8677	0.7687	0.7621
length = 20, Bi_LSTM(128),Bi_LSTM(64),D(32),D(3)				0.4274	0.5789	0.5876
vocab_size = 27000 , epochs = 5,embedding_dim = 64	2497219	2497219	0	0.8465	0.7677	0.7638
length = 20, Bi_LSTM(128),Bi_LSTM(64),D(32),D(3)				0.4053	0.5915	0.5964
vocab_size = total , epochs = 5,embedding_dim = 128	4693763	4693763	0	0.8548	0.7652	0.7585
length = 20, Bi_LSTM(128),Bi_LSTM(64),D(32),D(3)				0.3857	0.6045	0.6082
vocab_size = 25000 , epochs = 5,embedding_dim = 128	3631747	3631747	0	0.8549	0.7571	0.7552
length = 20, Bi_LSTM(128),Bi_LSTM(64),D(32),D(3)				0.3817	0.6321	0.6332
vocab_size = 30000 , epochs = 5,embedding_dim = 128	42711747	4271747	0	0.8533	0.7583	0.7568
length = 20, Bi_LSTM(128),Bi_LSTM(64),D(32),D(3)				0.3867	0.6082	0.6084
vocab_size = 27000 , epochs = 5,embedding_dim = 128	3887747	3887747	0	0.8546	0.765	0.7582
length = 20, Bi_LSTM(128),Bi_LSTM(64),D(32),D(3)				0.3863	0.6035	0.6118
vocab_size = 23000 , epochs = 5,embedding_dim = 128	3375747	3375747	0	0.8548	0.7593	0.7613
length = 20, Bi_LSTM(128),Bi_LSTM(64),D(32),D(3)				0.3879	0.5944	0.6088
vocab_size = 27000 , epochs = 10,embedding_dim = 128	3887747	3887747	0	0.9245	0.7503	0.7459
length = 20, Bi_LSTM(128),Bi_LSTM(64),D(32),D(3)				0.2126	0.7687	0.7687
voacb_size = 27000,epochs = 6,embedding_dim = 128,	3904387	3904387	0	0.8697	0.7693	0.7648
length = 20,Bi_LSTM(128),Bi_LSTM(64),Att(),D(32),D(3)				0.3506	0.5935	0.613

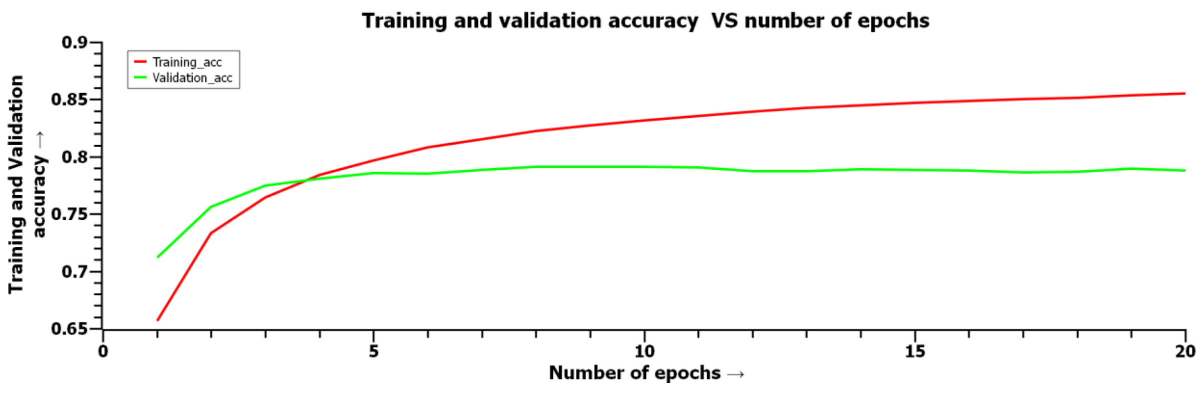
Below are the plots of training, validation accuracies and training, validation losses for the word embeddings model.

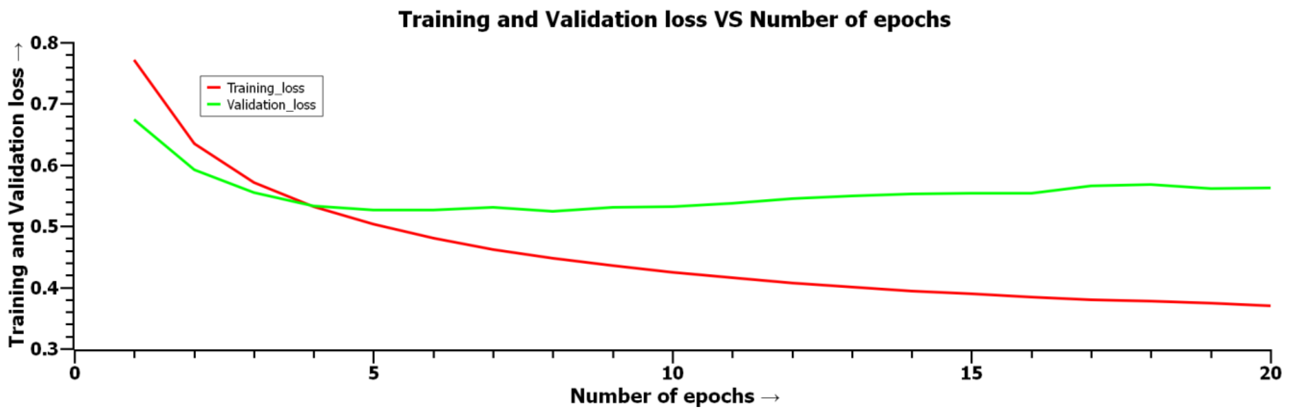




Glove_embeddings_model:

In this model, I am using the Glove embeddings which is trained on huge corpus and consists of vector representation for 400K words nearly with 6B tokens. Due to internet issues, I am using these glove embeddings (smallest in size). So, I have set training = False for the embedding layer. Text preprocessing is done in the same way as done before. We are using the labels 0,1 and 2 for entailment, contradiction and neutral, respectively. In many architectures, I have used 100D embeddings. Later, I have used 200D embeddings. With increase in the glove dimension, the results improved. This is expected as greater number of features and more information is available with larger dimension. I could not use 300D embeddings due to internet issues. I have used the truncated length of 20 for sentences in many architectures as this was computationally much feasible. I have tried RNNs, GRUs, LSTMs, etc. I found that Bidirectional LSTMs were producing the best result. Many of the different architectures that I have tried along with the results obtained are attached in the below table. So now I have tried different numbers of bidirectional LSTM layers. I could see that bidirectional LSTM layers were giving the best results. With a specific architecture by only varying the truncated length of sentences, I found that 25 was the best. Also, using attention layer improved the accuracies. By reducing the vocabulary size, I found that results rather got worse. So I kept num_words = None. Combining all the best hyper parameters that I could get; I have my model with 3 Bidirectional LSTMs (128,64 and 32 units) followed by attention layer with truncated length of sentences being 25 and using 200D glove embeddings. This model gave the highest validation accuracy of 0.7912. This is the model that I have saved. It gives a test accuracy of 0.79 and training accuracy of 0.8645. Compared to the word_embeddings model, this model has better results. Glove embeddings are obtained after training on a huge corpus. So, we can expect that they are better than the word embeddings that we can learn while training using this comparatively smaller dataset. So, the model using glove embeddings is my second model for this project. I have plotted the graphs of training and validation accuracies, training and validation losses.





Hyperparameters	Total parameters	Trainable	Non trainable	train_acc,loss	val_acc,loss	test_acc,loss
G_D = 100, length = 20,No_attention ,num_words = None	3342919	13219	3329700	0.6245	0.6206	0.6215
epochs = 10, Dr(0.1),RNN(64),D(32),D(16),D(3)				0.8327	0.8408	0.8446
G_D = 100, length = 20,num_words = None	3344015	14315	3329700	0.647	0.6438	0.6476
epochs = 10, Dr(0.1),RNN(64),RNN(32),Att(),D(16),D(3)				0.7902	0.7966	0.7545
G_D = 100, length = 20, epochs = 15,num_words = None	1675765	10915	1664850	0.4695	0.4904	0.4958
Dr(0.1),B_LSTM(128),B_LSTM(64),Att(),D(32),D(16),D(3)				1.0341	1.0189	1.0177
G_D = 100, length = 20,No_attention ,num_words = None	3371559	41589	3329700	0.7742	0.7629	0.7635
epochs = 15, Dr(0.1),GRU(64),GRU(32),D(16),D(3)				0.5537	0.5749	0.5846
G_D = 100, length = 20,num_words = None	3461671	131971	3329700	0.8	0.7724	0.77
epochs = 15, Dr(0.1),GRU(128),GRU(64),Att(),D(32),D(3)				0.5003	0.5566	0.5649
G_D = 100, length = 20, epochs = 15,num_words = None	3749895	420195	3329700	0.8662	0.7857	0.7828
Dr(0.1),B_LSTM(128),B_LSTM(64),Att(),D(32),D(16),D(3)				0.3539	0.5427	0.5625
G_D = 100, length = 20, epochs = 8,num_words = None	3733255	403555	3329700	0.8381	0.784	0.7819
Dr(0.1),B_LSTM(128),B_LSTM(64),D(32),D(16),D(3)				0.4168	0.5436	0.5553
G_D = 100, length = 20 epochs = 10,num_words = None	4800807	1471107	3329700	0.7801	0.7718	0.888
Dr(0.1),B_LSTM(256),B_LSTM(128),Att(),D(64),D(16),D(3)				0.5799	0.599	0.3011
G_D = 100, length = 20 epochs = 10,num_words = None	3775079	445379	3329700	0.8082	0.7622	0.7655
Dr(0.1),B_LSTM(128),Att(),D(64),D(16),D(3)				0.48	0.5892	0.592
G_D = 100, length = 20, epochs = 10,num_words = None	3362663	32963	3329700	0.824	0.7874	0.784
Dr(0.1),B_LSTM(128),B_LSTM(64),B_LSTM(32),Att(),D(16),D(3)				0.4487	0.5358	0.5485
G_D = 100, length = 20, epochs = 10,num_words = None	3460711	131011	3329700	0.804	0.7758	0.7687
Dr(0.1),B_LSTM(64),B_LSTM(32),Att(),D(16),D(3)				0.4092	0.5513	0.5632
G_D = 100, length = 15, epochs = 10,num_words = None	3749895	420195	3329700	0.8564	0.7694	0.7678
Dr(0.1),B_LSTM(128),B_LSTM(64),Att(),D(16),D(3)				0.3724	0.5823	0.5892
G_D = 100, length = 15, epochs = 10,num_words = None	3749895	420195	3329700	0.8825	0.7698	0.766
Dr(0.1),B_LSTM(128),B_LSTM(64),Att(),D(16),D(3)				0.3166	0.64	0.6441
G_D = 100, length = 30, epochs = 8,num_words = None	3749895	420195	3329700	0.8541	0.7636	0.7574
Dr(0.1),B_LSTM(128),B_LSTM(64),Att(),D(16),D(3)				0.3828	0.6163	0.6228
G_D = 100, length = 17, epochs = 10,num_words = None	3749895	420195	3329700	0.8974	0.7493	0.7435
B_LSTM(128),B_LSTM(64),Att(),D(16),D(3)				0.2773	0.7532	0.7817
G_D = 100, length = 23, epochs = 15,num_words = None	3749895	420195	3329700	0.8968	0.7593	0.756
Dr(0.1),B_LSTM(128),B_LSTM(64),Att(),D(16),D(3)				0.2773	0.6897	0.7064
G_D = 100, length = 25 epochs = 10,num_words = None	3749895	420195	3329700	0.887	0.7663	0.7626
Dr(0.1),B_LSTM(128),B_LSTM(64),Att(),D(16),D(3)				0.3024	0.6782	0.6854
G_D = 200, length = 20,epochs = 10,num_words = None	7181995	522595	6659400	0.8558	0.7773	0.7757
Dr(0.1),B_LSTM(128),B_LSTM(64),Att(),D(16),D(3)				0.3792	0.562	0.5774
G_D = 200, length = 25, epochs = 10,num_words = None	7208747	549347	6659400	0.8645	0.7912	0.79
Dr(0.1),B_LSTM(128),B_LSTM(64),Bi_LSTM(32),Att(),D(16),D(3)				0.3571	0.5314	0.5375
G_D = 200, length = 25, epochs = 10,num_words = 27000	5922595	522595	5400000	0.8617	0.7792	0.78
Dr(0.1),B_LSTM(128),B_LSTM(64),Att(),D(16),D(3)				0.3625	0.5594	0.5734

Note: G_D = Glove dimension, length = truncated length of sentences, num_words = size of vocabulary, Dr – Dropout, Att() – Attention layer. Some architectures have been run on small number of epochs as they were very time consuming.

Note: I tried to implement BERT using ktrain. I found that a single epoch would take around 200 hours even when GPU is used. So, I could not try the BERT model for this project.

Some observations:

Sklearn did not improve in the training accuracy in the way manually implemented Logistic regression improved. In fact, it was almost saturated after small number of epochs. But, as said could reach 0.9987 training accuracy with 100 epochs. This was surprising to me as a linear classifier could overfit so well.

Comparing the best word embeddings model and the best glove embeddings model, I could see that bidirectional LSTMs are the best layers to have for this dataset. Also, the difference in the best accuracies obtained in the 2 models is not largely different. Time to run word embeddings model was very large compared to the glove embeddings model. I expected that glove embeddings will give much better results compared to word embeddings model which did not happen.

Many times, I have treated a hyperparameter individually to find its best value by keeping the remaining architecture same. This sometimes failed as sometimes best value of the hyperparameter is varying with the architecture. So, we cannot treat everything independently. But with some observations, I have come with the models that I have saved in this project.

There were 5998 words for which there was no embedding in the glove embeddings. I printed these words and they were all wrongly spelled words most of the time. The total vocabulary size was around 33000 for the whole training set and so 5998 is a significant number. So, this will affect the model. If we had corrected the spellings, we might have had a better training set and better results.

Adam was the best optimizer in all the cases. I tried sometimes with SGD, but Adam always dominated. I have used Sparse categorical cross entropy in embedding models as the embeddings themselves would be very sparse.