WHAT I HAVE DONE:

- Implemented a Logistic Regression Classifier using TF-IDF features using scikit in python.
- Implemented a BILSTM neural network using glove embeddings using pytorch in colab.

IMPLEMENTATION OF LOGISTIC REGRESSION CLASSIFIER:

- Written code to download the SNLI dataset and separate the premise, hypothesis and labels individually with respect to train, valid and test data from the dataset.
- Combined the premise and hypothesis of train and test individually and cleaned the data using NLTK toolkit, however we can combine or do dot product of premise and hypothesis after vectorization also.
- The combined train data given to the TFIDF vectorizer, fit the logistic regression model and predicted the accuracy by applying test data.
- Created a tfidf_text file with the predicted labels.

RESULTS:

- Varied the Inverse of Regularization strength parameter(C) in sk-learn model with and without cleaning the data.

	Without cleaning data	With cleaning data
C = 1	57.78	56.35
C = 10	58.42	56.66
C = 15	58.44	56.89

WHAT I OBSERVED:

- As we see that, we are getting better accuracies without cleaning the data.
- When I used only tokenization and lowercase alphabets, means without stemming and removal of stopwords in cleaning the data, I got the 58.45 accuracy (same as without cleaning) for C=15.

IMPLEMENTATION OF BILSTM NEURAL NETWORK:

- I used torchtext library to download the dataset and preprocess it.
- I converted the words to vectors using pretrained glove embeddings instead of bag of words because to reduce the dimension of the vectors to be processed.
- Implemented Bidirectional Long Short Term Memory (BILSTM) neural network because it can handle long sentences compared to RNN's, however we have transformers, BERT for better performances.
- Concatenated premise and hypothesis after applying to Lstm network individually.
- After the initial setup trained the model with train data and validated with valid data by tuning hyper parameters for higher validation accuracy.
- Finally tested with the test data to generalize the trained network.

RESULTS:

Choosing the optimizer: noted accuracies for different optimizers with 10 epochs.

a) ADAM – 79.79%
b) SGD – 33%
c) RMSprop – 33%

Choosing learning rate: when I used default learning rate (10^-3) model is generalizing well if I took lesser than this learning rate loss is increasing.

Choosing no. of layers and units:

a) No of Layers: 3
   No of units/layer: 600
   No of fc layers: 4
   Valid_accu: 78.74%

b) No of Layers: 2
   No of units/layer: 300
   No of fc layers: 3
   Valid_accu: 79.79

- when I increase only layers or units, model is taking too much time to train.
- When I increase both layers and units accordingly model is taking considerable time.

Choosing Regularization methods:

- I purposefully overfit the model by increasing the number of epochs and used weight decay and dropout.

a) Weight_decay = 0

Dropout = 0.25

Accuracy = 79.79

b) Weight_decay = learning rate/(epoch+1)

Dropout = 0.25

Accuracy = 80.96

Generalized model:

- Adam optimizer with weight_decay and learning rate as 0.001.
- 2 LSTM layers with 300 units in each layer and 3 fully connected linear layers with RELU activation.
- Used dropout of 0.25
- This model embeds the tokens in each sentence into 300-dimensional GloVe embeddings (which are frozen) and then creates an embedding for the entire sentence by simply summing the tokens of all embeddings within the sentence.
- These are then fed into 3 linear layers which output a prediction.

WHAT I OBSERVED:

- Adam optimizer is giving better accuracy for large datasets and its improving the model performance when I used weight decay.
- Bilstm is not giving better results when i increase the model complexity moreover its taking too much time to train.
- When I change pretrained glove embeddings from glove.6B.300d to glove.840B.300d model is improving the performance.
- Finally, I got the accuracy of around 81-82%.

Finally, I shown how to use the model for inference, allowing us to perform natural language inference on any sentence.