INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Introduction to NLP (CS7.401)

# Project Report
# Natural Language Inference

*Submitted by:*

Team 26

Bhanuj Gandhi - 2022201068

Ayush Lakhshkar - 2022201051

Aakash Tripathi - 2022201053

May 5, 2023

## Contents

## 1 Project Objective

The objective of project is to understand the task of Natural Language Inference in the field of Natural Language Processing. Natural Language Inference deals in understanding the relationship between two given sentences. It tries to identify whether a given sentence, called as "Hypothesis" can be derived/inferred/deduced from another given sentence, called "Premise" or not. The relationship can be classified into 3 different classes/labels –

- **Contradiction** – refers to a situation when both, premise and hypothesis, cannot be true at the same time.
- **Entailment** - refers to a situation where Hypotheses can be derived/inferred from given premise.
- **Neutral** – refers to a situation where there is not enough information in premise to infer or derive the given hypothesis.

## 2 About Datasets

For this task, we have explored the 2 famous available datasets – SNLI and Multi NLI.

### 2.1 SNLI

SNLI stands for Stanford Natural Language Inference. It is a benchmark dataset for natural language inference tasks.

- All the premises are the image captions from Flicker30 dataset and hence it makes SNLI somewhat genre restricted.
- All the hypotheses are written by Crowd-workers i.e., corresponding to a premise, crowd workers will write 3 sentences one for each class.
- Total of 550152 train examples with 10,000 as dev samples and 10,000 as test samples, balanced equally across 3 classes.
- Mean token length in SNLI dataset i.e., the average number of words per sentence
  - For premise – 12.9
  - For hypothesis – 7.4
- Approximately 56, 951 examples are validated by 4 additional annotators with 91.2% of gold labels matched with author's labels.
- For SNLI dataset the overall Fleiss' kappa is 0.70, which is defined as the degree of agreement between the annotators, based both categorical labels and similarity matrix.
- Progress on SNLI dataset for Natural Language Inference task
  - Red line indicates the human performance on SNLI dataset
  - The main fundamental logic in SNLI is – relating to image dataset, if premise and hypothesis probably describe a different photo, then the label is contradiction.

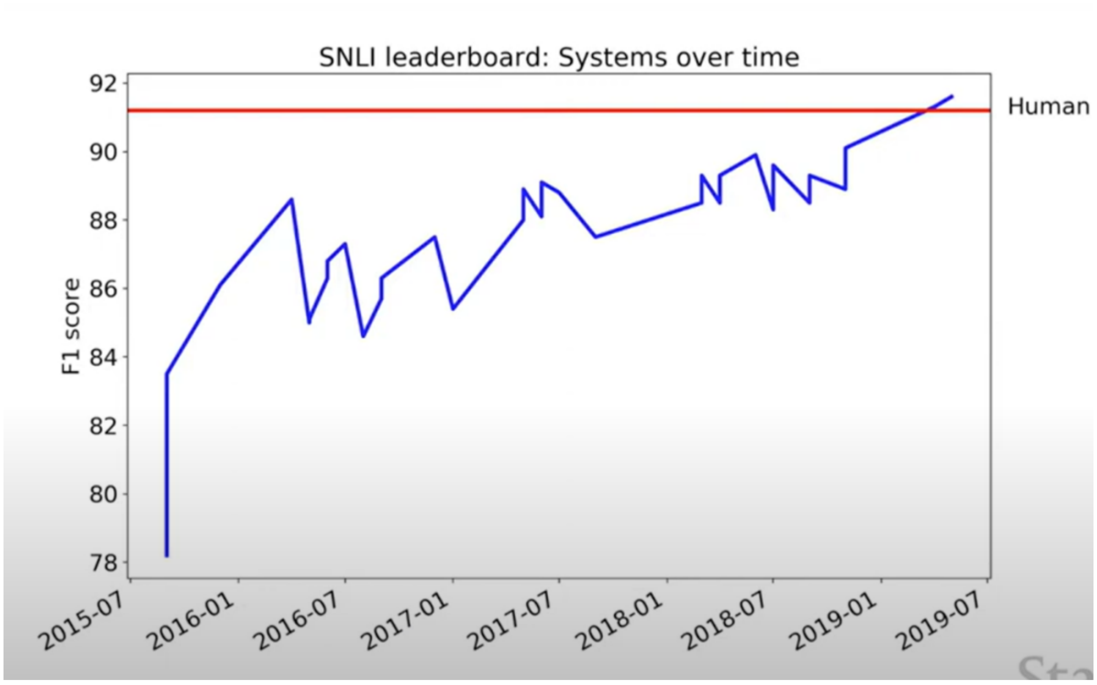


Figure 1: Source - SNLI, MultiNLI, and Adversarial NLI — Stanford CS224U Natural Language Understanding — Spring 2021

## 2.2 MultiNLI

MultiNLI stands for Multi-Genre Natural Language Inference. It is also another benchmark dataset for natural language inference tasks. It is an extension of the SNLI dataset, with a more diverse set of genres and sources and hence making it a more challenging dataset to train and evaluate the NLI models.

- Train Premises in MultiNLI are contributed from 5 genres –
  - Fiction: works from 1912 – 2010 : spanned across many genres.
  - Govt. information – available public reports, letters, speeches, Govt. websites etc.
  - The Slate website
  - The Switchboard corpus – Telephonic conversation
  - Bertlitz travel guide
- In addition to above genres, premises from some other genres are also present in dev and test datasets like
  - From 9/11 reports
  - From fundraising letters
  - Nonfiction from Oxford University Press
  - Verbatim – articles about linguistics.
- Total of approx. 3,92,702 train examples with 20,000 as dev samples and 20,000 as test samples.
- Approx. 19,647 samples are validated by 4 additional annotators with 92.6% of gold labels matched with author's labels.

- Test dataset of MultiNLI is only available on Kaggle and in the form of competition.
- Progress on SNLI dataset for Natural Language Inference task –
  - Red line indicates the human performance on Multi NLI dataset
  - There is so much variance in MultiNLI progress as compared to SNLI dataset because MultiNLI datasert is available on Kaggle and so there are too much data points/model to compare. But in SNLI, the progress is reported only from the implemented research papers only and not some from public domain, which is the case in MultiNLI.
  - MultiNLI dataset is filled with distributed annotations that helps to perform out of the box error analysis.

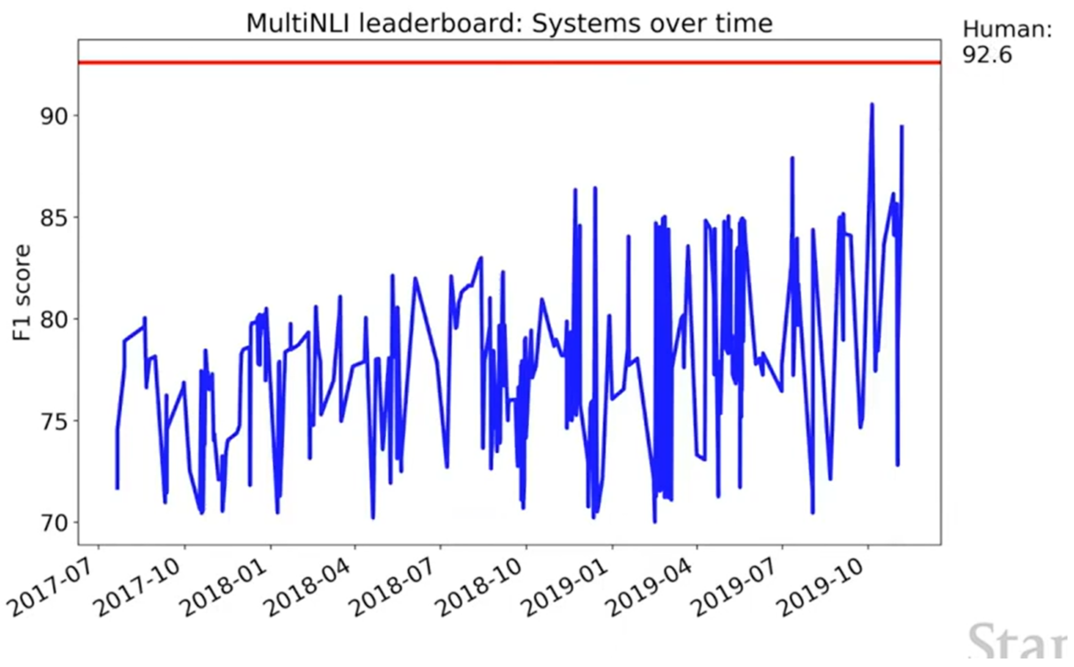


Figure 2: Source - SNLI, MultiNLI, and Adversarial NLI — Stanford CS224U Natural Language Understanding — Spring 2021

## 3 Exploratory Data Analysis

To get the look and feel of the data, some basic data analysis is done. It helps us to understand and gain insights into our data before starting any model or making decisions based on it.

Visualization of the distribution of each gold label (Contradiction, Neutral, Entailment) has been done to know the distribution of each gold label in the dataset o to know whether there is some bias towards any label is present or not.

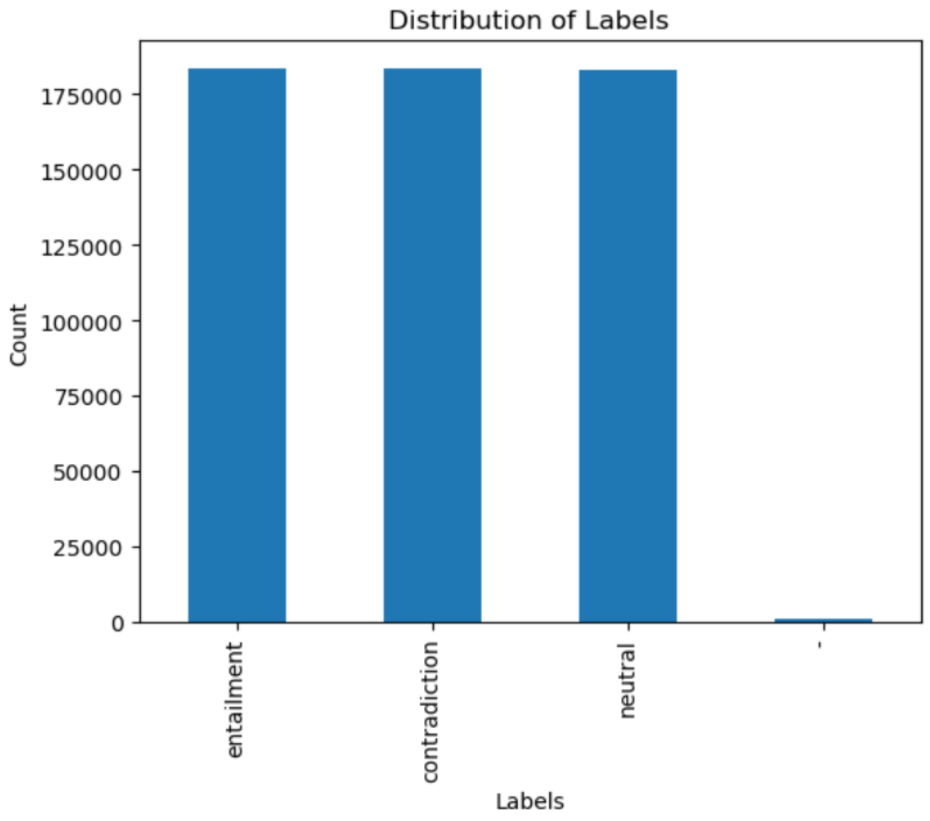


Figure 3: Distribution of Gold Labels in SNLI dataset

Clearly it can be seen that dataset contains almost equal proportion of Entailment, Contradiction and Neutral statement.

We have also analysed the various statistical parameters of premise and hypothesis that helped to better understand SNLI & MultniNLI dataset.

Also, the most frequently occurred 20 words in premise sentences are identified and viewed as shown below: -

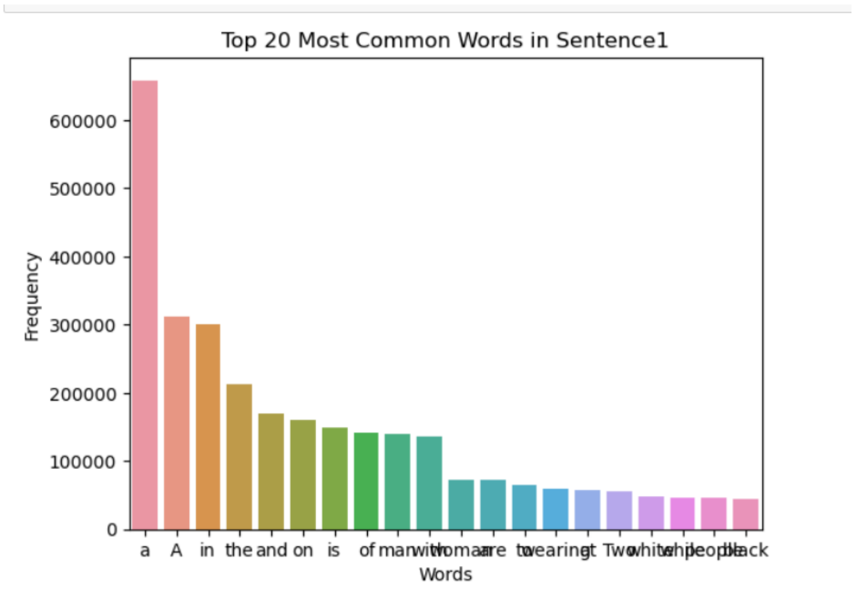


Figure 4: Top 20 words in SNLI dataset

## 4 Techniques Used

We have utilized 4 techniques to address the challenge of Natural Language Inference, namely: -

- Logistic Regression
- Bi-directional LSTM
- Bi-directional GRU
- ELMo
- BERT

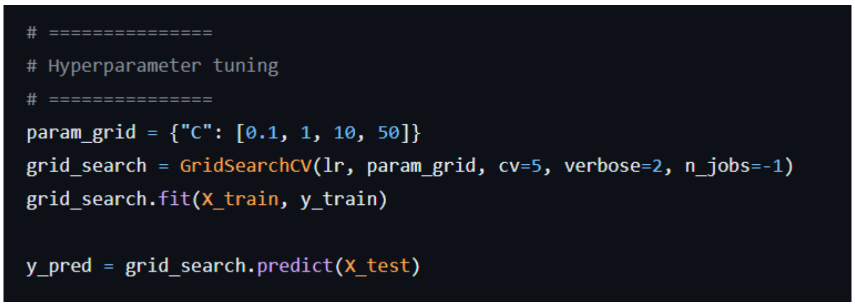
### 4.1 Logistic Regression –

First things first, Logistic Regression has been applied on both SNLI and MultiNLI dataset.

Implementing Logistic Regression on dataset first includes data pre-processing. Data-pre-processing step includes removing all the unnecessary columns that are not useful. Also, there are some sentence pair for which gold-label is '- ', as they do not add any information. Hence, such sentence pair are also removed from the dataset before training. All the sentence pairs having NULL premise or hypothesis were removed. Gold labels are also encoded to categories before feeding the input to Logistic Regression model. Pickle file for gold labels is also saved for future purposes.

Cleaned Premise and Hypothesis of training data are then concatenated and passed to TF-IDF (Term Frequency and Inverse Document Frequency) vectorizer with stop-word removal. Vectorizer pickle file is then saved as the same file will be used to vectorize test data while evaluating accuracy and other evaluation metrics. Premise and Hypothesis sentences are vectorized using the same TF-IDF vectorizer and concatenated (horizontally) to form the training data (X_train) to pass to Logistic Regression model with max_iter = 10000.

Since, we are using the scikit learn implementation of Logistic Regression, it uses some kind of regularization and amount of regularization is a hyperparameter. Optimal Value of hyperparameter 'C' is determined using cross-validation. For Cross Validation, we have used GridSearchCV.



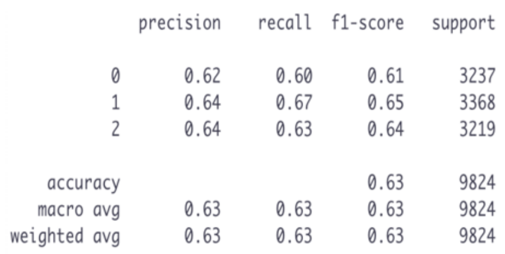
```
# ==============
# Hyperparameter tuning
# ==============
param_grid = {"C": [0.1, 1, 10, 50]}
grid_search = GridSearchCV(lr, param_grid, cv=5, verbose=2, n_jobs=-1)
grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)
```

Figure 5: Cross Validation in Logistic Regression

The parameter 'cv' specified the number of cross validation folds, 'n_jobs' specifies the number of CPU cores to be used in parallel, and 'verbose' control the output that will be printed on screen.

The trained best logistic regression model is then saved for future purposes.

The Classification report, confusion metrics calculated over test data using this Hyperparameter tuned Logistic Regression model are shown below –



```
              precision    recall  f1-score   support

           0       0.62      0.60      0.61      3237
           1       0.64      0.67      0.65      3368
           2       0.64      0.63      0.64      3219

    accuracy                           0.63      9824
   macro avg       0.63      0.63      0.63      9824
weighted avg       0.63      0.63      0.63      9824
```

Figure: Classification Report (SNLI Dataset)

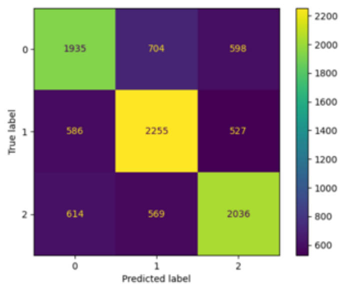


Figure: Confusion Matrix (SNLI Dataset)

Figure 6: Classification Report and Confusion Matrix for Logistic Regression Model on SNLI Dataset

**As seen, an accuracy of 63% has been achieved through Logistic Regression model.**

## 4.2 Bi-directional LSTM

Bi-directional LSTM is also implemented on both SNLI and MultiNLI dataset. Training, Validation, and test set for a particular dataset has been loaded and only the premise, hypothesis and gold labels are being used for training. As done in Logistic Regression also, gold labels with value '-' has been removed as they do add any information for taking decision. Also, the sentences where either premise or hypothesis is null are removed. All the sentence pair are converted to lowercase. Also, all the Unicode characters has been removed from each of the premise and hypothesis (if any). Gold labels are encoded to categorical labels just like we did in Logistic Regression case.

Here we have used GloVe embedding that convert each word into real values vectors. Embedding that we have used is 6 billion tokens trained over the entire Wikipedia corpus and has 300 dimensions for each word token. This embedding has been loaded in dictionary with key-value pair so that we can access it quickly without reading the file again and again.

Premise and Hypothesis are then concatenated using space as delimiter and passed to Keras Inbuilt tokenizer that will assign a unique number to each word. We have capped the number of words in our corpus by 20k. Weight matrix is then created such that for at *ith* position (row), it contains the embedding vector for the word that has been assigned number $i$ by the tokenizer.

Each pair of premise and hypothesis is converted to assigned number by tokenizer and padded with 0 to make each of them of size 42. Gold labels are also converted to one-hot vectors before passing to the model for training.

**Architecture -**

Model architecture can be very well explained in below diagram –

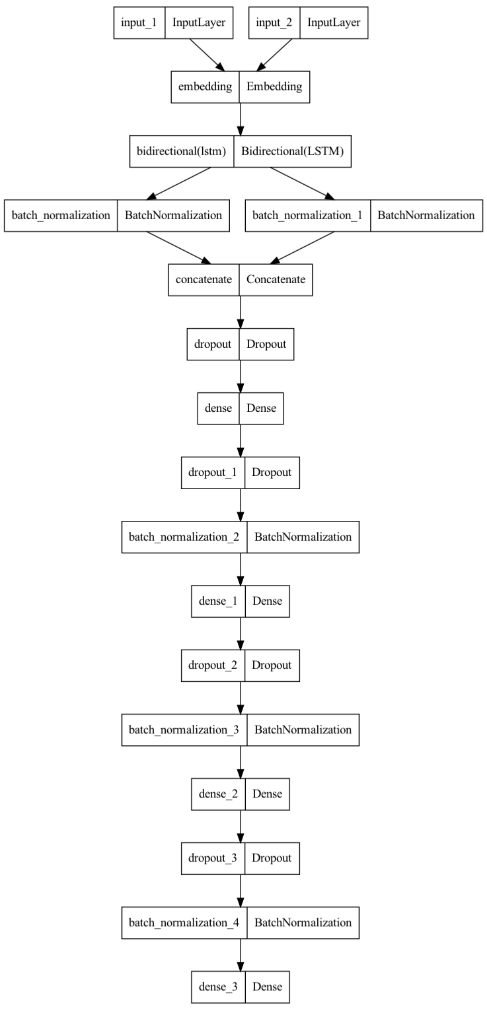

input_1 | InputLayer
input_2 | InputLayer
embedding | Embedding
bidirectional(lstm) | Bidirectional(LSTM)
batch_normalization | BatchNormalization
batch_normalization_1 | BatchNormalization
concatenate | Concatenate
dropout | Dropout
dense | Dense
dropout_1 | Dropout
batch_normalization_2 | BatchNormalization
dense_1 | Dense
dropout_2 | Dropout
batch_normalization_3 | BatchNormalization
dense_2 | Dense
dropout_3 | Dropout
batch_normalization_4 | BatchNormalization
dense_3 | Dense

Figure 7: Architecture of Bi-LSTM Model

- The model consists of two input layers, one for the premise and the other for the hypothesis, followed by an embedding layer that converts the sequence of integers to a sequence of embedding vectors. These output embedding vectors are then passed through a shared Bi-LSTM layer one at a time. Afterward, the output vectors are separately normalized using the BatchNormalization layer to reduce the internal covariate shift that occurs during network training.

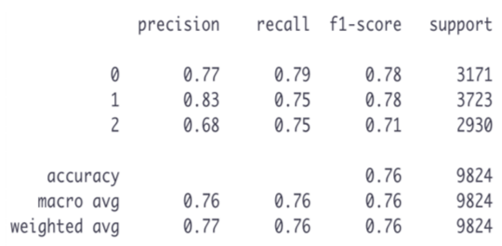
- Next, the normalized premise and hypothesis vectors are concatenated to create the train_input, which is then passed through a Dropout layer with a dropout rate of 20%. The output is then passed through three consecutive Dense layers, each followed by a Dropout and BatchNormalization layer with ReLU activation function.

- Finally, the output is passed through a Dense layer with Softmax activation function to obtain the final probability distribution among the three categories - Contradiction, Entailment, and Neutral. The category with the highest probability will be selected as the class or final label for that sentence pair.

Adam optimizer and **Categorical CrossEntropy Loss** is used. Two callbacks were implemented for better training of the model, that are **EarlyStopping** and **ModelCheckpoint**. EarlyStopping monitors a specified quantity (here validation loss) and stops training when the quantity stops improving. ModelCheckpoint keeps track and keeps saving the best model during training.

Saved model is then used to predict the accuracy on the test data. Classification report and Confusion Matrix generated from this model is shown below –

**As seen, an accuracy of 76% has been achieved through Bi-directional LSTM model.**



```
              precision    recall  f1-score   support

           0       0.77      0.79      0.78      3171
           1       0.83      0.75      0.78      3723
           2       0.68      0.75      0.71      2930

    accuracy                           0.76      9824
   macro avg       0.76      0.76      0.76      9824
weighted avg       0.77      0.76      0.76      9824
```

Figure: Classification Report (SNLI Dataset)

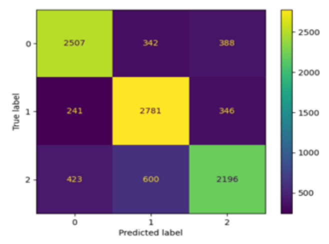
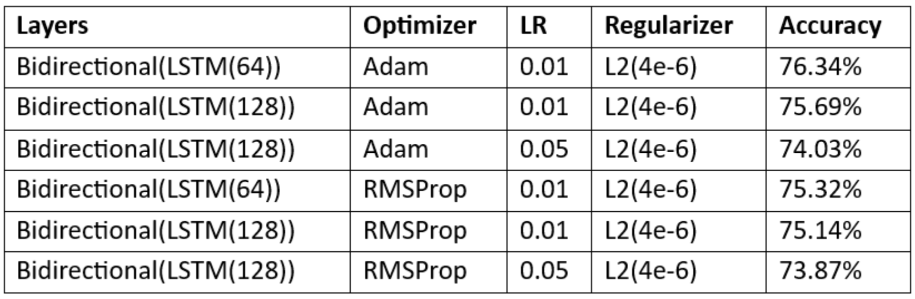


Figure: Confusion Matrix (SNLI Dataset)

Figure 8: Classification Report and Confusion Matrix for Bi-LSTM Model on SNLI Dataset

**Empirical Findings with Bi-directional LSTM-**



| Layers | Optimizer | LR | Regularizer | Accuracy |
|---|---|---|---|---|
| Bidirectional(LSTM(64)) | Adam | 0.01 | L2(4e-6) | 76.34% |
| Bidirectional(LSTM(128)) | Adam | 0.01 | L2(4e-6) | 75.69% |
| Bidirectional(LSTM(128)) | Adam | 0.05 | L2(4e-6) | 74.03% |
| Bidirectional(LSTM(64)) | RMSProp | 0.01 | L2(4e-6) | 75.32% |
| Bidirectional(LSTM(128)) | RMSProp | 0.01 | L2(4e-6) | 75.14% |
| Bidirectional(LSTM(128)) | RMSProp | 0.05 | L2(4e-6) | 73.87% |

Figure 9: Experiments with Bi-LSTM

## 4.3 Bi-directional GRU

Bi-directional GRU (Gated Recurrent Unit) architecture-based model is also trained on both the SNLI and MultiNLI datasets. The training, validation, and test sets for each dataset have been loaded and only the premise, hypothesis, and gold labels are being utilized for training purposes. As with the logistic regression and bi-directional LSTM models, gold labels with a value of ”-” have been removed, as they do not contribute to the decision-making process. Additionally, sentences where either the premise or hypothesis are null have been removed, and all sentence pairs have been converted to lowercase. Furthermore, any Unicode characters have been eliminated from the premise and hypothesis. The gold labels have been encoded into categorical labels as in the other two cases.

The GloVe embedding method has been utilized, which converts each word token into a real-valued vector. The embedding utilized is a 6 billion token model that has been trained on the entire Wikipedia corpus, with 300 dimensions for each word token. This embedding has been loaded into a dictionary for quick access without the need to read the file repeatedly.

The premise and hypothesis are concatenated using a space as a delimiter and passed through Keras' inbuilt tokenizer, which assigns a unique number to each word. The number of words in the corpus has been capped at 20k, and a weight matrix is then generated such that each row represents the embedding vector for the word assigned to the corresponding position by the tokenizer.

Each premise-hypothesis pair is then converted to the assigned number by the tokenizer and padded with 0s to create a length of 42. Finally, the gold labels are also converted to one-hot vectors before being passed to the model for final training.

**GRU is different from the LSTM in the sense that GRU has a smaller number of training parameters as compared to LSTM. There is no separate forget and update gate in GRU. They both are combined and thus the number of parameters reduced.**

**Architecture –**

- The model consists of two input layers, one for the premise and the other for the hypothesis, followed by an embedding layer that converts the sequence of integers to a sequence of embedding vectors. These output embedding vectors are then passed through a shared Bi-GRU layer one at a time. Afterward, the output vectors are separately normalized using the BatchNormalization layer to reduce the internal covariate shift that occurs during network training.
- Next, the normalized premise and hypothesis vectors are concatenated to create the train_input, which is then passed through a Dropout layer with a dropout rate of 20%. The output is then passed through three consecutive Dense layers, each followed by a Dropout and BatchNormalization layer with ReLU activation function.
- Finally, the output is passed through a Dense layer with Softmax activation function to obtain the final probability distribution among the three categories - Contradiction, Entailment, and Neutral. The category with the highest probability will be selected as the class or final label for that sentence pair.

Model architecture can be very well seen in below diagram –

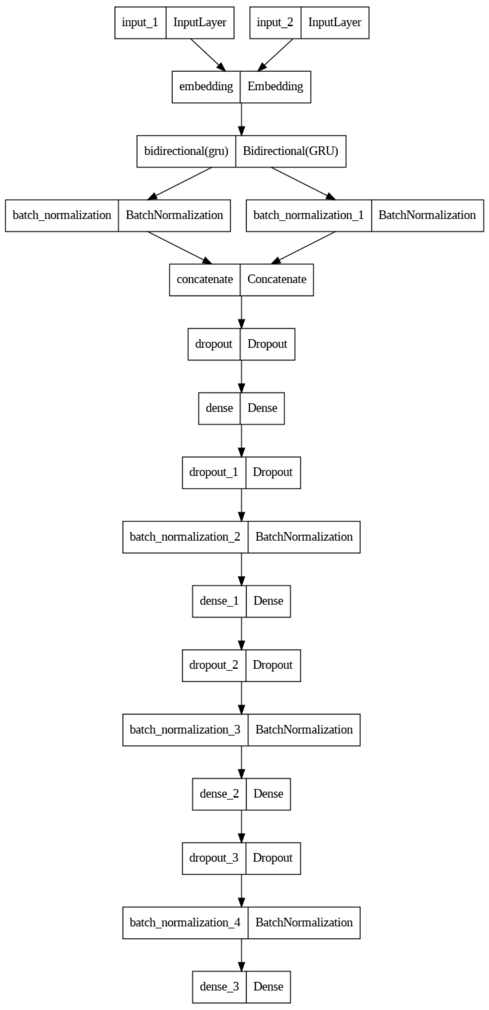

input_1 | InputLayer
input_2 | InputLayer
embedding | Embedding
bidirectional(gru) | Bidirectional(GRU)
batch_normalization | BatchNormalization
batch_normalization_1 | BatchNormalization
concatenate | Concatenate
dropout | Dropout
dense | Dense
dropout_1 | Dropout
batch_normalization_2 | BatchNormalization
dense_1 | Dense
dropout_2 | Dropout
batch_normalization_3 | BatchNormalization
dense_2 | Dense
dropout_3 | Dropout
batch_normalization_4 | BatchNormalization
dense_3 | Dense

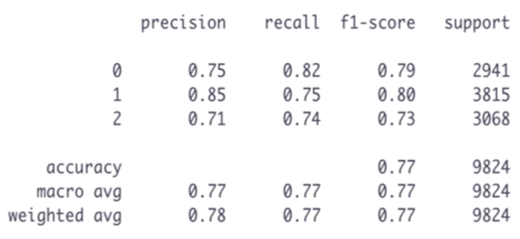
Figure 10: Architecture of Bi-GRU Model

**RMSProp** optimizer and **Categorical CrossEntropy Loss** is used. Two callbacks were implemented for better training of the model, that are **EarlyStopping** and **ModelCheckpoint**. EarlyStopping monitors a specified quantity (here validation loss) and stops training when the quantity stops improving. ModelCheckpoint keeps track and keeps saving the best model during training.

Saved model is then used to predict the accuracy on the test data. Classification report and Confusion Matrix generated from this model is shown below –

**As seen, an accuracy of 77% has been achieved through Bi-directional GRU model.**

**Empirical Findings with Bi-directional GRU-**



```
              precision    recall  f1-score   support

           0       0.75      0.82      0.79      2941
           1       0.85      0.75      0.80      3815
           2       0.71      0.74      0.73      3068

    accuracy                           0.77      9824
   macro avg       0.77      0.77      0.77      9824
weighted avg       0.78      0.77      0.77      9824
```

Figure: BiGRU (SNLI Dataset)

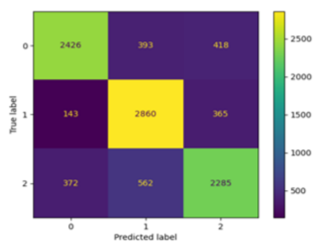


Figure: BiGRU (SNLI Dataset)

Figure 11: Classification Report and Confusion Matrix for Bi-GRU Model ON SNLI dataset

| Layers | Optimizer | LR | Regularizer | Accuracy |
|---|---|---|---|---|
| Bidirectional(GRU(64)) | RMSProp | 0.01 | L2(4e-6) | 77.34% |
| Bidirectional(GRU (128)) | RMSProp | 0.01 | L2(4e-6) | 76.17% |
| Bidirectional(GRU (128)) | RMSProp | 0.05 | L2(4e-6) | 75.02% |
| Bidirectional(GRU (64)) | Adam | 0.01 | L2(4e-6) | 72.19% |
| Bidirectional(GRU (128)) | Adam | 0.01 | L2(4e-6) | 71.12% |
| Bidirectional(GRU (128)) | Adam | 0.05 | L2(4e-6) | 69.54% |

Figure 12: Experiments with Bi-GRU

## 4.4 ELMO

ELMo (Embeddings from Language Models) is a deep contextualized word embedding model. It is based on a bi-directional LSTM (Long Short-Term Memory) language model that is trained on a large corpus of text data, and it generates context-dependent word embeddings that take into account the meaning of the word in the context of the sentence.

Unlike traditional word embeddings such as word2vec or GloVe, which generate a single fixed vector representation for each word, ELMo generates multiple vector representations for each word, depending on the context in which the word appears. This allows ELMo to capture the nuances of meaning that can be missed by traditional word embeddings, which treat each occurrence of a word as independent.

Like BERT, Pre-trained ELMo can also be fine-tuned to adapt to downstream task like Natural Language Inference in our case.

**Architecture of ELMo –**

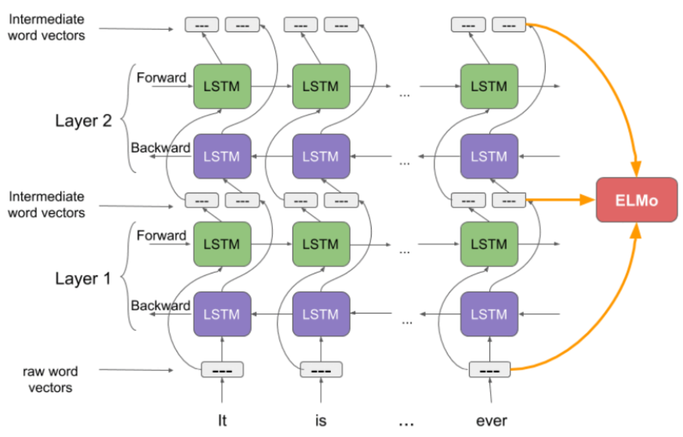
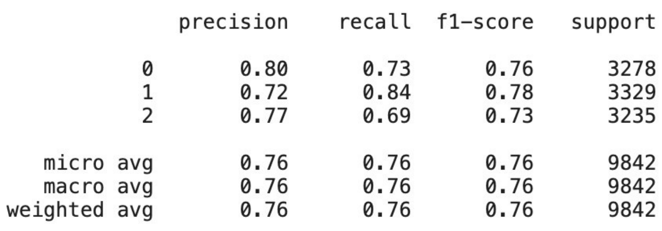


Figure 13: ELMo Architecture Source - https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/

To address the problem of Natural Inference, we are using the pre-trained ELMo available in Tensorflow Hub. Using Pre-trained ELMo, we extract the embedding vector for both premise and hypothesis. Both are of size 100 dimensions.

For NLI task, the obtained embedding vectors are concatenated (200 dim) and passed through a linear classifier (single fully connected layer) that gives the probability of all the 3 classes i.e. ***Entailment, Neutral and Contradiction***. The label for the max probability will be considered as the predicted label for that premise-hypothesis pair.

Saved model is then used to predict the accuracy on the test data. Classification report and Confusion Matrix generated from this model is shown below –



```
              precision    recall  f1-score   support

           0       0.80      0.73      0.76      3278
           1       0.72      0.84      0.78      3329
           2       0.77      0.69      0.73      3235

   micro avg       0.76      0.76      0.76      9842
   macro avg       0.76      0.76      0.76      9842
weighted avg       0.76      0.76      0.76      9842
```

Figure 14: Classification Report for ELMo Model ON SNLI dataset

Here we are fine tuning the pre-trained ELMo model. We also tried to implement ELMo from scratch. But the results are not good. The accuracy through our ELMo implementation comes out to be approx. 45%. This might be because of the reason that out ELMo model is trained on very limited data, both in quantity and diversity (nearly 50k sentences pairs). Also, due to the scarcity of resources, we are training them on a very few numbers of epochs.

## 4.5 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained natural language processing (NLP) model developed by Google. It is based on the transformer architecture and is trained using a large corpus of unlabelled text data. BERT is a deep neural network that can be fine-tuned for various downstream NLP tasks such as text classification, question answering, and named entity recognition, Natural Language Inference etc.

The unique feature of BERT is that it is trained using a bidirectional approach, meaning that it can take into account the entire context of a word or phrase, rather than just the surrounding words. This is accomplished through the use of a masked language modelling task, where a certain percentage of words in a sentence are randomly masked and the model must predict what those words are based on the surrounding context. In the original paper, the optimal masking of words is 15%.

BERT uses the concept of self-attention. It is a mechanism that allows model to focus on different parts of input sequence by assigning weights to each input element based on its relevance to current context.

In a typical self-attention mechanism, the input sequence is transformed into three vectors: the query vector(Q), the key vector (K), and the value vector (V). These vectors are then used to calculate a weight for each input element (so called attention), which is then used to compute a weighted sum of the value vectors. The resulting weights are passed through a **softmax** function to ensure that they sum to 1 and can be used as probabilities. The resulting output vector represents the attention-weighted sum of the input elements, which can be used for downstream tasks such as classification or translation.

BERT uses the concept of Multi-Headed Self Attention. Each head has its own query (Q), key (K) and value (V) vector. Attention from each of the head is calculated and will get head specific hidden state. A composition function is needed that will combine all head specific state to 1 hidden state. The main motivation behind multi headed self-attention is that each attention can focus on different aspects of linguistic property and together they will capture more complex patterns and dependencies. For example, one head might focus on subject of sentence, other might on object or some other part of speech.

**Transformers are different from the RNN in the sense that transformers use the concept of attention, while RNN uses recurrent connections. In RNN, hidden state from the previous state is fed into current state and thus making it sequential. While Transformers, all the words are processed parallelly and uses the concept of attention to calculate the importance/contribution of each word in determining the current word.**

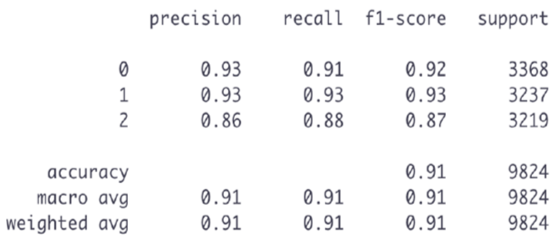
For Natural Language Inference task, in this project we have utilized BERT-base-uncased model. The base refers to the model's size and complexity. It has 12 transformer layers (Block), 110 million parameters, and a hidden size of 768, making it a relatively large and powerful model. The "uncased" in BERT-base-uncased refers to the fact that the model was trained on lowercased text, which means that it treats uppercase and lowercase letters as equivalent.

Here we have used the BertTokenizer provided by pre-trained BERT model. As the job of tokenizer, it converts raw text data to numerical data that can be fed to pretrained BERT model. It also adds the special tokens like [CLS] and [SEP].

Here we are fine-tuning the pre-trained BERT model by placing a linear layer on the top of it to cater the needs of the provided downstream task which is Natural Language Inference in our case. It might be possible that pre-trained BERT has already captured some of the properties needed for NLI task. By adding a linear layer on the top of it, we are trying to fine-tune and trying to extract classification power of pre-trained BERT to correctly classify between contradiction, entailment and neutral premise and hypothesis pair.

We have used **AdamW Optimizer** (Adam with weight decay regularization to avoid overfitting) along with **CrossEntropyLoss** and follows the same method to train as done in other methods mentioned above.

Saved model is then used to predict the accuracy on the test data. Classification report and Confusion Matrix generated from this model is shown below –



```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92      3368
           1       0.93      0.93      0.93      3237
           2       0.86      0.88      0.87      3219

    accuracy                           0.91      9824
   macro avg       0.91      0.91      0.91      9824
weighted avg       0.91      0.91      0.91      9824
```

Figure: Classification Report (SNLI Dataset)

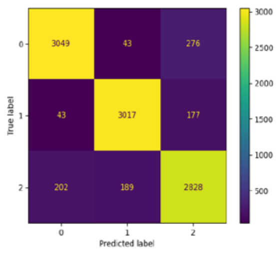


Figure: Confusion Matrix (SNLI Dataset)

Figure 15: Classification Report and Confusion Matrix for BERT Model ON SNLI dataset

**As seen, an accuracy of 91% has been achieved through BERT based model.**

*Note:- Due to the resource scarcity, the BERT based model is trained only for 3 epochs.*

## 5 Application of Natural Language Inference

**Sentence Transformer –**

Sentence transformer is a type of NLP technique that uses deep learning to map sentences into a high-dimensional space where semantically similar sentences are closer together and dissimilar sentences are far apart, making it easier to perform various downstream tasks such as machine translation, text classification, clustering, and retrieval.

Sentence Transformer are used to generate the rich sentence embeddings. These increasingly rich sentence embeddings can be used to quickly compare sentence similarity for various use cases. Such as:

- **Semantic textual similarity (STS)** — comparison of sentence pairs. We may want to identify patterns in datasets.
- **Semantic search** — information retrieval (IR) using semantic meaning. Given a set of sentences, we can search using a 'query' sentence and identify the most similar records. Enables search to be performed on concepts (rather than specific words).
- **Clustering** — we can cluster our sentences.

In machine translation encoder-decoder models, the original sentence is encoded into a context vector and decoder decodes that vector to target language.

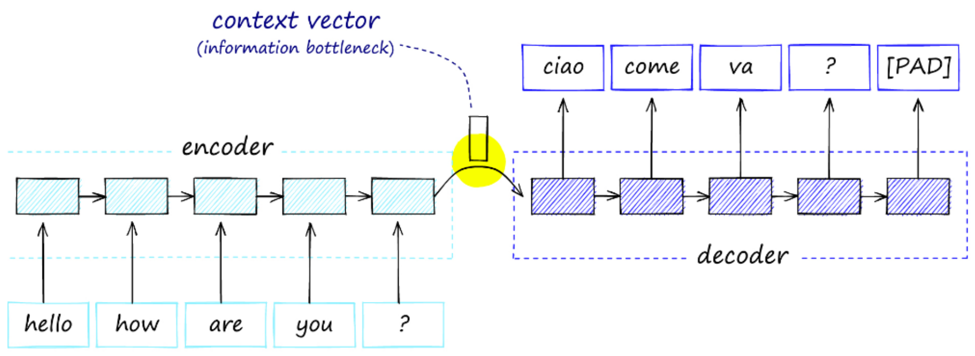


Figure 16: Information BottleNeck between models

During decoding, the model decodes one word/timestep at a time. Similarity between the word and all encoder annotations is calculated for each step.

The problem here is that we create an information bottleneck between the two models. We're creating a massive amount of information over multiple time steps and trying to squeeze it all through a single connection. This limits the encoder-decoder performance because much of the information produced by the encoder is lost before reaching the decoder.

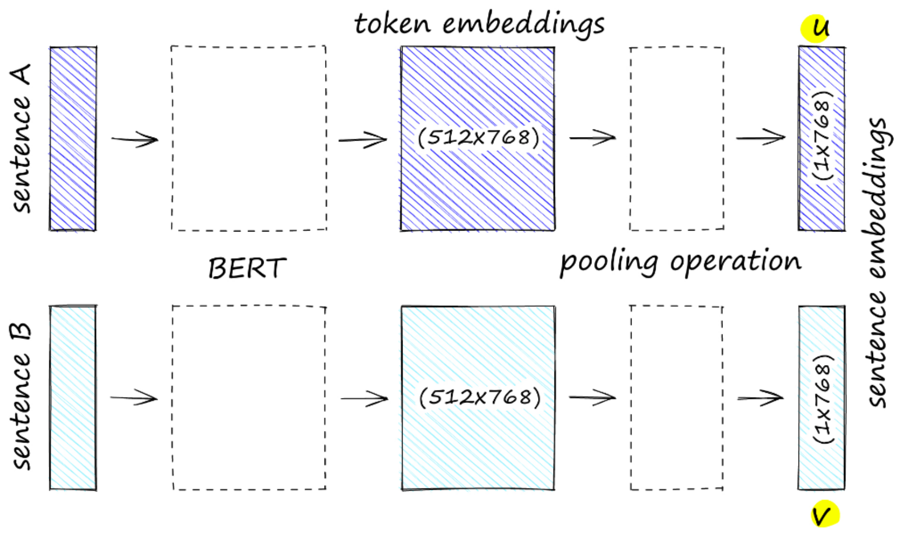
The attention mechanism provided a solution to cater bottleneck issue. It offered another route for information flow.

But all the transformers-based approach has 1 issue: - Transformers work using word or token-level embeddings, not sentence-level embeddings. Before sentence transformers, the approach to calculating accurate sentence similarity with BERT was to use a cross-encoder structure. This meant that we would pass two sentences to BERT, add a classification head to the top of BERT - and use this to output a similarity score.

The cross-encoder network does produce very accurate similarity scores (better than SBERT), but it's not scalable. If we wanted to perform a similarity search through a small 100K sentence dataset, we would need to complete the cross-encoder inference computation 100K times. To cluster sentences, we would need to compare all sentences in our 100K dataset, resulting in just under 500M comparisons - this is simply not realistic.

Ideally, we need to pre-compute sentence vectors that can be stored and then used whenever required. If these vector representations are good, all we need to do is calculate the cosine similarity between each. With the original BERT (and other transformers), we can build a sentence embedding by averaging the values across all token embeddings output by BERT (if we input 512 tokens, we output 512 embeddings). Alternatively, we can use the output of the first [CLS] token (a BERT-specific token whose output embedding is used in classification tasks). However, the accuracy obtained is not good, and is worse than using averaged GloVe embeddings.

The solution to this lack of an accurate model with reasonable latency is sentence-BERT (SBERT). Many more sentence transformers have been built using similar concept of SBERT. Using loss function like softmax loss and multiple negatives ranking loss or MSE margin loss, these models are optimized to produce similar embeddings for sentences, and dissimilar embedding otherwise.



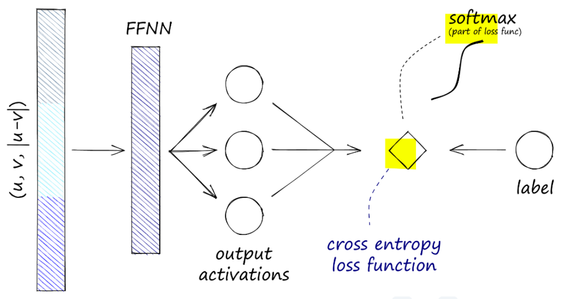
An SBERT model applied to a sentence pair *sentence A* and *sentence B*. Note that the BERT model outputs token embeddings (consisting of 512 768-dimensional vectors). We then compress that data into a single 768-dimensional sentence vector using a pooling function.

Figure 17: SBert Model

To train sentence transformer, the softmax-loss approach used the Siamese architecture fine-tuned on the Stanford Natural Language Inference (SNLI) and Multi-Genre NLI (MNLI) corpora

In the sentence transformer model, a premise sentence (A) and a hypothesis sentence (B) are fed into two separate siamese BERT models. These models output sentence embeddings using three different pooling methods: mean-pooling, max-pooling, and [CLS]-pooling. In the SBERT paper, it was found that the mean-pooling approach performed the best for both natural language inference (NLI) and semantic textual similarity benchmark (STSb) datasets. The embeddings generated are then concatenated. The concatenated embedding contains premise embeddings (U), hypothesis embeddings (V) and element wise difference between the 2 embeddings ($U - V$) (all in the same order).

The concatenated embeddings are then passed through Feed forward Neural Network that produces 3 outputs as labels 0,1,2. These three outputs align to our NLI similarity labels 0, 1, and 2. We need to calculate the softmax from our FFNN, which is done within the cross-entropy loss function. The softmax and labels are used to optimize on this softmax-loss.



The operations were performed during training on two sentence embeddings, u and v. Note that *softmax-loss* refers cross-entropy loss (which contains a softmax function by default).

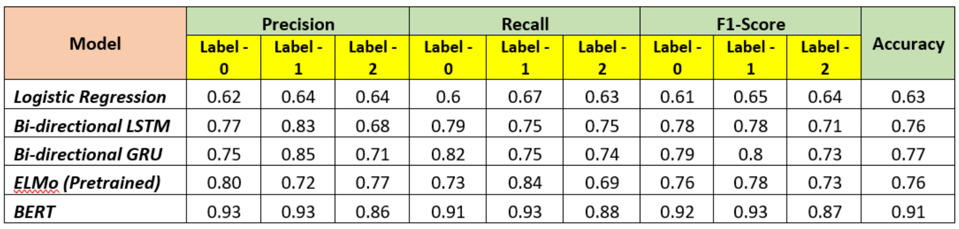
Figure 18: FFNN in SBERT

This results in our pooled sentence embeddings for similar sentences (label 0) becoming more similar, and embeddings for dissimilar sentences (label 2) becoming less similar. Here we are using the siamese BERTs not dual BERTs. Meaning we don't use two independent BERT models but a single BERT that processes sentence A followed by sentence B. This means that when optimizing the model weights, they are pushed in a direction that allows the model to output more similar vectors where we see an entailment label and more dissimilar vectors where we see a contradiction label.

## 6 Conclusion

To address the task of Natural Language Inference, which involves determining whether a given sentence (hypothesis) can be inferred from another sentence (premise) or not. The Machine Learning model utilized are Logistic Regression, Bi-directional LSTM (Bi-LSTM), Bi-directional Gated Recurrent Unit (Bi-GRU), BERT base uncased and ELMo.

Accuracy achieved with all the above-mentioned models are as follows -



| Model | Precision | | | Recall | | | F1-Score | | | Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|
| | Label - 0 | Label - 1 | Label - 2 | Label - 0 | Label - 1 | Label - 2 | Label - 0 | Label - 1 | Label - 2 | |
| ***Logistic Regression*** | 0.62 | 0.64 | 0.64 | 0.6 | 0.67 | 0.63 | 0.61 | 0.65 | 0.64 | 0.63 |
| ***Bi-directional LSTM*** | 0.77 | 0.83 | 0.68 | 0.79 | 0.75 | 0.75 | 0.78 | 0.78 | 0.71 | 0.76 |
| ***Bi-directional GRU*** | 0.75 | 0.85 | 0.71 | 0.82 | 0.75 | 0.74 | 0.79 | 0.8 | 0.73 | 0.77 |
| ***ELMo (Pretrained)*** | 0.80 | 0.72 | 0.77 | 0.73 | 0.84 | 0.69 | 0.76 | 0.78 | 0.73 | 0.76 |
| ***BERT*** | 0.93 | 0.93 | 0.86 | 0.91 | 0.93 | 0.88 | 0.92 | 0.93 | 0.87 | 0.91 |

Figure 19: Cumulative result archived

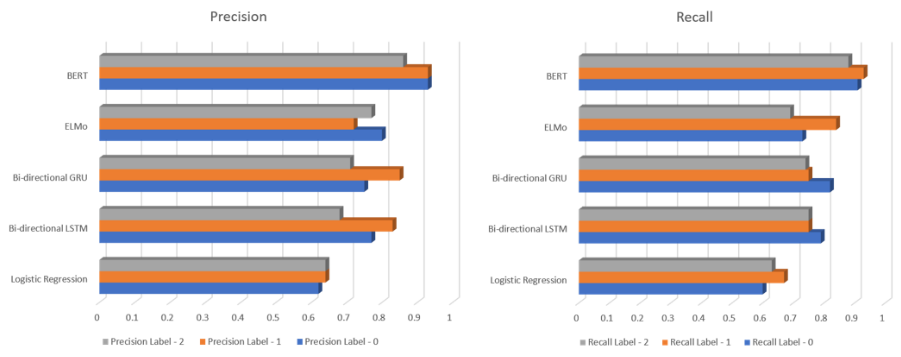


Figure 20: Precision and Recall Comparison

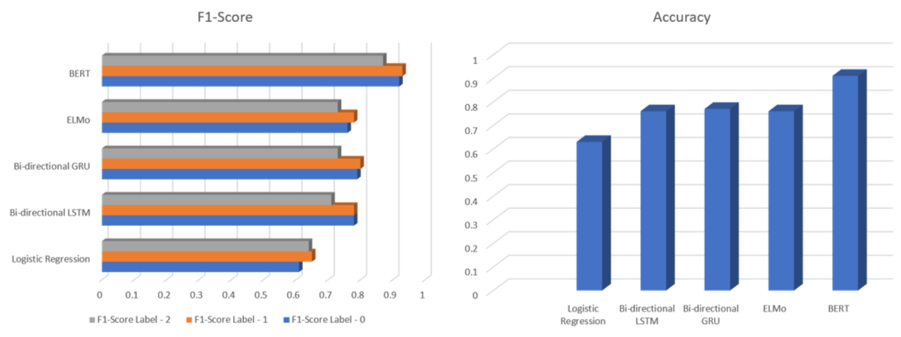


Figure 21: F1 Score and Accuracy

## 7 Quantitative Analysis

We have implemented 5 different models to capture Natural Language Inference task and reported accuracy and other evaluation metric in all of the methods. The graph for the accuracy is plotted as above. Clearly it can be seen that BERT surpasses all other techniques implemented.

So, for NLI task we can say empirically that Transformer based techniques works better than RNN based techniques. Also, in BERT we trained the model only for 3 epochs due to the scarcity of computation and still it gives ¿90% accuracy.

Also, we have used BERT tokenizer for tokenizing the sentences which empirically proofed to work better than any other traditional tokenizing techniques. Data Pre-processing also played a major role to attain a desirable accuracy.

We have used pretrained glove embeddings as initial embedding in BERT model that also contributes to produce better accuracy.

At the end, during the course of this project, we also analysed that choosing correct hyperparameters also played a major role to pass a threshold of evaluation metric.

## 8 Limitations

We have only explored few of the Language Modelling techniques. There may be some other State of the Art methods also that we haven't explored that may give better results. Also, we have fine-tuned BERT model only for 3 epochs due to GPU limitations. Better accuracy can be achieved if we fine-tuned it for more epochs.

Here we have used only 1 data set to train the model. However, the obtained results may not be generalizable to other data set.

## References

[1] Bowman, S. R.; Angeli, G.; Potts, C. Manning, C. D. (2015), A large annotated corpus for learning natural language inference, in 'Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)' , Association for Computational Linguistics, .

[2] Williams, A., Nangia, N. and Bowman, S.R., 2017. A broad-coverage challenge corpus for sentence understanding through inference. arXiv preprint arXiv:1704.05426.

[3] Parikh, A.P., Täckström, O., Das, D. and Uszkoreit, J., 2016. A decomposable attention model for natural language inference. arXiv preprint arXiv:1606.01933.

[4] Conneau, A., Kiela, D., Schwenk, H., Barrault, L. and Bordes, A., 2017. Supervised learning of universal sentence representations from natural language inference data. arXiv preprint arXiv:1705.02364.

[5] Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K. Zettlemoyer, L. (2018), 'Deep contextualized word representations' , cite arxiv:1802.05365Comment: NAACL 2018. Originally posted to openreview 27 Oct 2017. v2 updated for NAACL camera ready.

[6] Devlin, J.; Chang, M.-W.; Lee, K. Toutanova, K. (2019), BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, in 'Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)' , Association for Computational Linguistics, Minneapolis, Minnesota , pp. 4171–4186.

[7] Reimers, N. and Gurevych, I., 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084.