Project Report

Natural Language Inference

(Introduction to NLP)

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Submitted by:

Team 21

Radhika Garg - 2023201030

Shalu Kumari - 2023201031

Shoaib Ahmed – 2023201080

May 8, 2023

Contents

1. Project Objective

Natural language understanding presents a fundamental challenge due to the inherent variability in semantic expression within text. One prominent task in this domain is textual entailment recognition, which involves determining whether the meaning of one text fragment can be inferred from another. This task is essential for various natural language processing applications, including question answering, information retrieval, and sentiment analysis, including question answering, information retrieval, and sentiment analysis.
Textual entailment recognition is a task that involves determining whether the meaning of one text can be inferred from another text. The task involves two text fragments, named text (t) and hypothesis (h), respectively. The objective is to identify if the hypothesis can be inferred from the text. The classification problem involved in TE is a balanced three-class problem over sentence pairs, which includes ***contradiction***, ***entailment***, and ***neutral.***

• ***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

There are many benchmark datasets for Natural Language Inference, but we are focused on a couple of famous datasets below:

e-SNLI:

The e-SNLI dataset is a comprehensive collection of sentence pairs annotated for Natural Language Inference (NLI). Similar to the original SNLI dataset, e-SNLI focuses on determining the logical relationship between two given sentences: a premise and a hypothesis.

The e-SNLI dataset, comprising 569,033 sentence pairs, is an extension of the widely used SNLI dataset. Each sentence pair in e-SNLI consists of a premise sentence and a hypothesis sentence, similar to SNLI.

Each pair is annotated with one of three possible relationships: "contradiction," "entailment," or "neutral." The dataset encompasses a diverse range of scenarios and contexts, ensuring broad coverage in terms of language styles and situations.

Here is what a sample from the e-SNLI dataset looks like:

Premise: A person in a blue shirt and tan shorts getting ready to roll a bowling ball down the alley.
Hypothesis: A person is napping on the couch.
Predicted label: contradiction
Explanation: A person cannot be napping and getting ready to roll a bowling ball at the same time.

e-SNLI serves as a benchmark dataset for evaluating the performance of natural language inference models. Overall, the e-SNLI dataset provides a valuable resource for advancing research in natural language processing, particularly in the domain of Natural Language Inference. Its comprehensive annotations, diverse scenarios, and rigorous quality control measures make it a cornerstone for developing and evaluating NLI models.

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

– Berlitz travel guide

In addition to above genres, premises from some other genres are also present in dev and
test datasets like

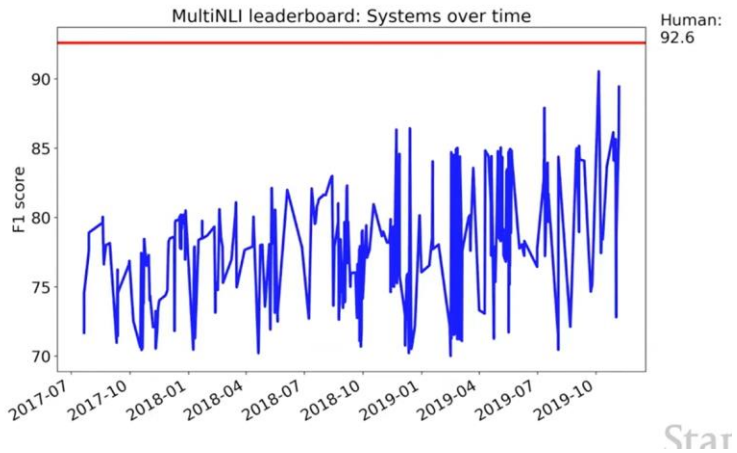
– From 9/11 reports

– 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 authors labels.
- Test dataset of MultiNLI is only available on Kaggle and in the form of competition.



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

SNLI:

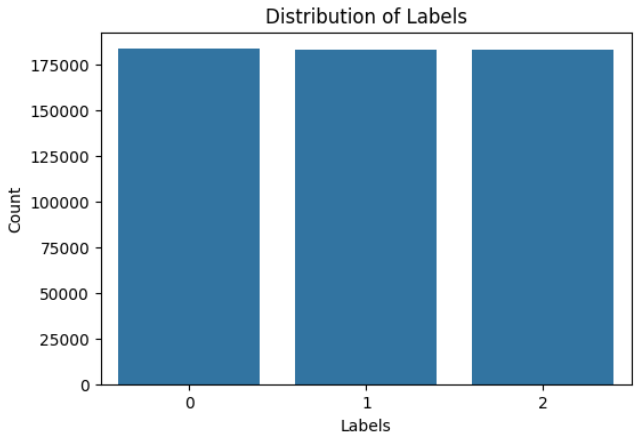
SNLI stands for Stanford Natural Language Inference. It is a benchmark dataset for natural
language inference tasks.

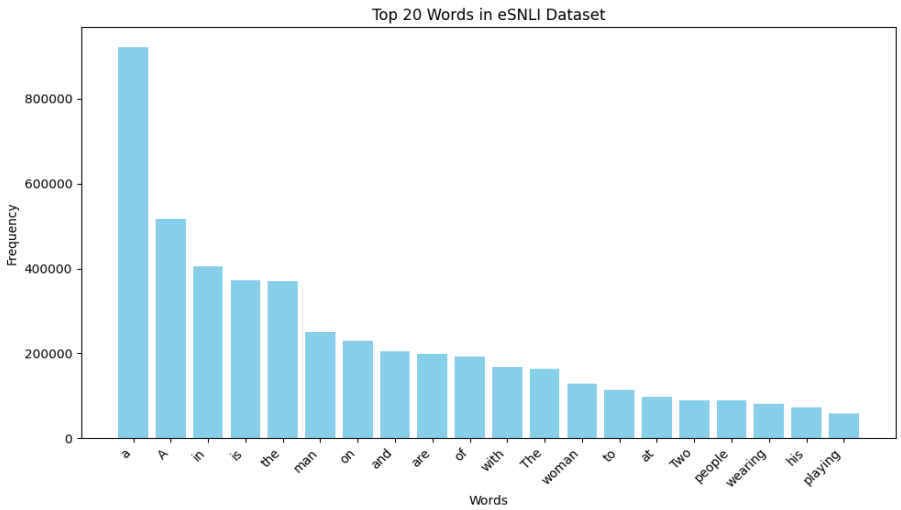
- All the premises are the image captions from Flicker30 dataset and hence it makes SNLI somewhat genre restricted.
- All the hypotheses are written by Crowd-workers i.e., corresponding to a premise, crowd workers will write 3 sentences one for each class.
- Total of 550152 train examples with 10,000 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 authors labels.
- For the SNLI dataset the overall Fleiss' kappa is 0.70, which is defined as the degree of agreement.

3. Exploratory Data Analysis

1. e-SNLI

Dataset info:
DatasetDict({
  train: Dataset({
    features: ['premise', 'hypothesis', 'label', 'explanation_1', 'explanation_2', 'explanation_3'],
    num_rows: 549367
  })
  validation: Dataset({
    features: ['premise', 'hypothesis', 'label', 'explanation_1', 'explanation_2', 'explanation_3'],
    num_rows: 9842
  })
  test: Dataset({
    features: ['premise', 'hypothesis', 'label', 'explanation_1', 'explanation_2', 'explanation_3'],
    num_rows: 9824
  })
})





2. SNLI

Below are the EDA graphs For SNLI dataset (for train , validation and test set datasets). Similarly the graphs for all the datasets are included in the jupyter notebook submitted along with the report.

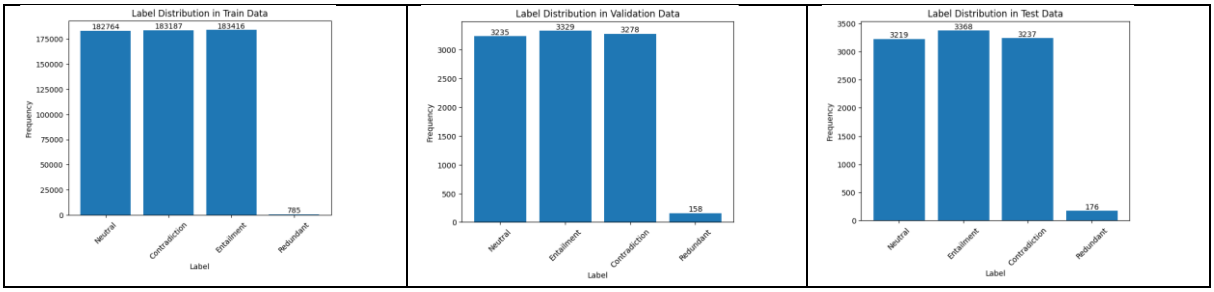
Number of samples in train split: 550152
Number of samples in validation split: 10000
Number of samples in test split: 10000

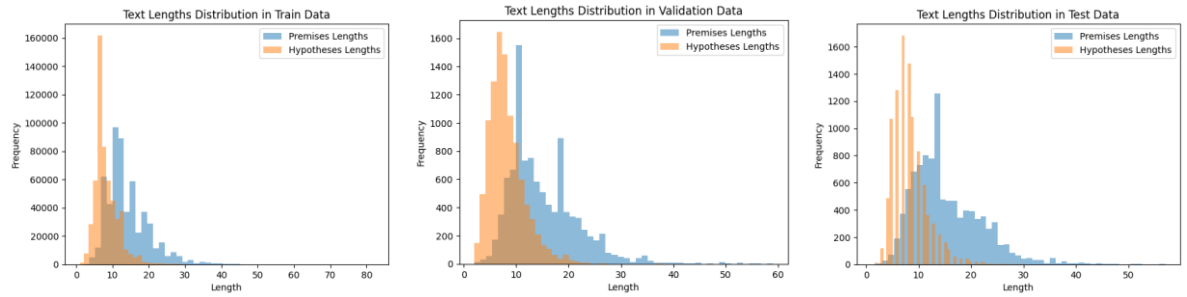
Sample of a data point in train split:
{'premise': 'A person on a horse jumps over a broken down airplane.', 'hypothesis': 'A person is training his horse for a competition.', 'label': 1}

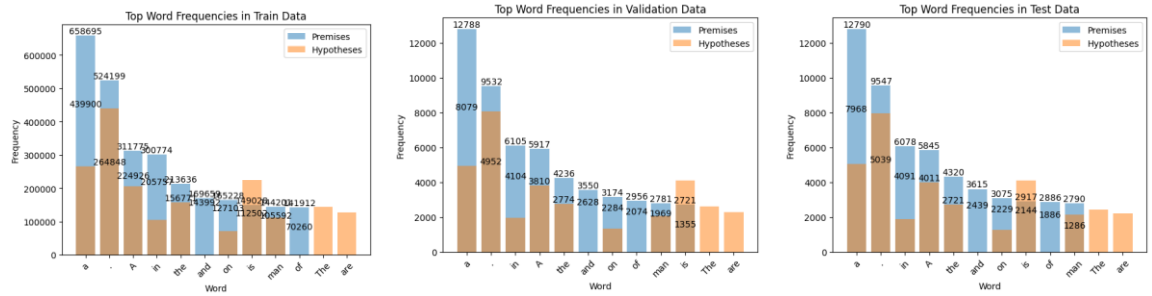
Unique labels: {0, 1, 2, -1}

Null values in train split: []
Null values in validation split: []
Null values in test split: []

Train, validation and test dataset (for SNLI)







4. Techniques Used

We have utilized 5 techniques to address the challenge of Natural Language Inference, namely:

- Logistic Regression
- Bi-directional LSTM
- Bi-directional GRU
- BERT
- T5 Transformer

4.1 Logistic Regression:

Logistic Regression has been applied on e-SNLI, MultiNLI as well as the SNLI dataset. The steps utilized are as follows :

Data Loading and Preprocessing:
Data preprocessing functions are defined to convert text to lowercase and extract premises, hypotheses, and labels.
Data-preprocessing step includes removing all the unnecessary columns that are not useful. Also, there are some sentence pair for which 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

Data Preparation:
The dataset is split into training, validation, and test sets.
Premises and hypotheses are concatenated to form input data.
Text data is vectorized using the *Bag-of-Words* approach with a maximum of 5000 features.

Model Definition and Training:
A logistic regression model is defined.
The model is trained using the training features and labels.

Model Evaluation:
The trained model is evaluated on the training, validation, and test sets.
Accuracy scores are calculated and printed for each set.
A classification report is printed for the validation set.

Visualization:
A confusion matrix is computed for the validation set predictions.
The confusion matrix is visualized using a heatmap, showing the distribution of true and predicted labels.

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

1.eSNLI

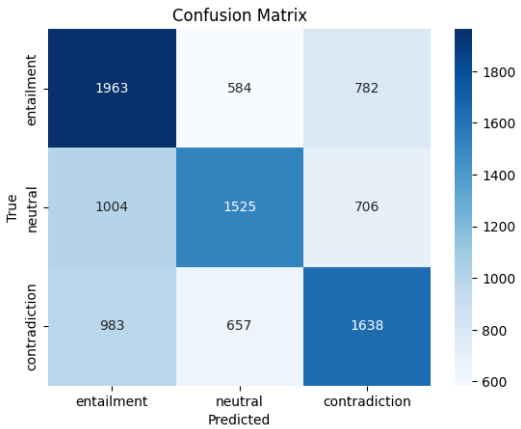
Train Accuracy: 0.53

Validation Accuracy: 0.52

Test Accuracy: 0.52

Classification Report:

	precision	recall	f1-score	support
0	0.50	0.59	0.54	3329
1	0.55	0.47	0.51	3235
2	0.52	0.50	0.51	3278
accuracy			0.52	9842
macro avg	0.52	0.52	0.52	9842
weighted avg	0.52	0.52	0.52	9842



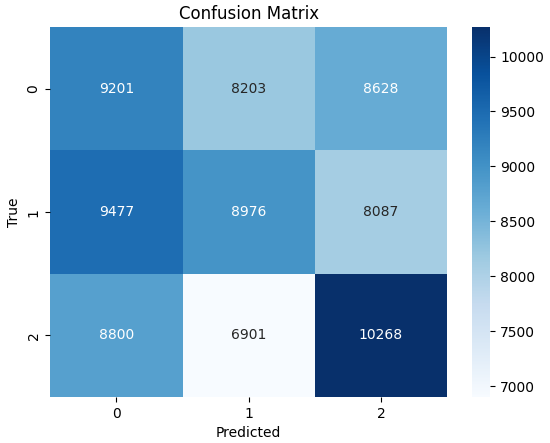
Classification Report and Confusion Matrix for Logistic Regression Model on e-SNLI Dataset

2.MultiNLI

Train Accuracy: 0.42
Test Accuracy: 0.36

Classification Report:

	precision	recall	f1-score	support
0	0.33	0.35	0.34	26032
1	0.37	0.34	0.35	26540
2	0.38	0.40	0.39	25969
accuracy			0.36	78541
macro avg	0.36	0.36	0.36	78541
weighted avg	0.36	0.36	0.36	78541



3.SNLI

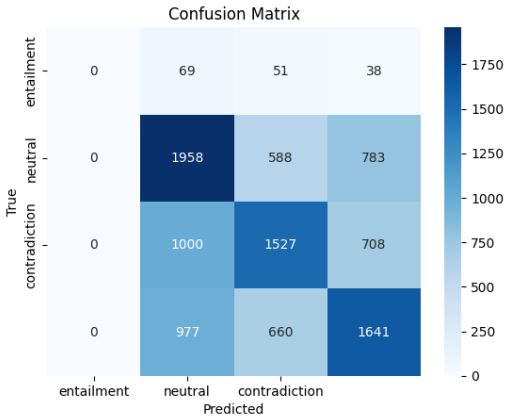
Train Accuracy: 0.53

Validation Accuracy: 0.51

Test Accuracy: 0.52

Classification Report:

	precision	recall	f1-score	support
-1	0.00	0.00	0.00	158
0	0.49	0.59	0.53	3329
1	0.54	0.47	0.50	3235
2	0.52	0.50	0.51	3278
accuracy			0.51	10000
macro avg	0.39	0.39	0.39	10000
weighted avg	0.51	0.51	0.51	10000



4.2 Bidirectional LSTM:

Bi-LSTM also has been applied on e-SNLI, MultiNLI as well as the SNLI dataset. The steps utilized are as follows :

Dataset Loading and Preprocessing: We loads the appropriate dataset using the load_dataset function from the datasets library. It then preprocesses the dataset by converting text to lowercase and extracting premises, hypotheses, and labels.

As done in Logistic Regression also, 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.

Text Tokenization and Indexing: Text data is tokenized using the basic English tokenizer provided by torchtext. The tokens are then converted into indices based on a vocabulary built from the training data.

Model Definition: The BiLSTM model is defined using PyTorch, which consists of an embedding layer, a bidirectional LSTM layer, and a linear layer for classification.

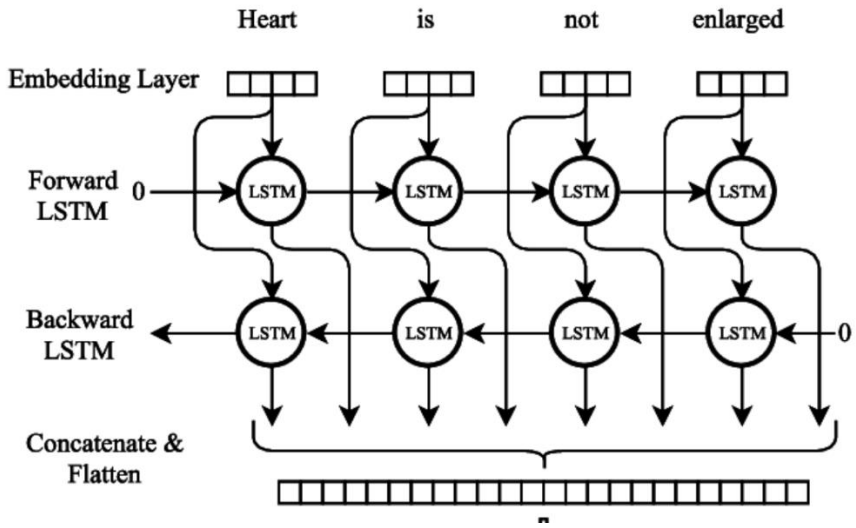
Model Training and Evaluation: The model is trained and evaluated using a training loop and evaluation functions. Training is performed for multiple epochs with early stopping based on validation loss.

Loss and Optimization: Cross-entropy loss is used as the loss function, and Adam optimizer is used for optimization.

Evaluation Metrics: The code computes and prints various evaluation metrics such as accuracy, classification report, and confusion matrix.

Plotting: The code also includes plotting functionality to visualize the training and validation loss vs. epoch and the confusion matrix.

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-LSTM layer one at a time. Afterward, the output vectors are separately normalized using the Batch Normalization 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 Batch Normalization 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.

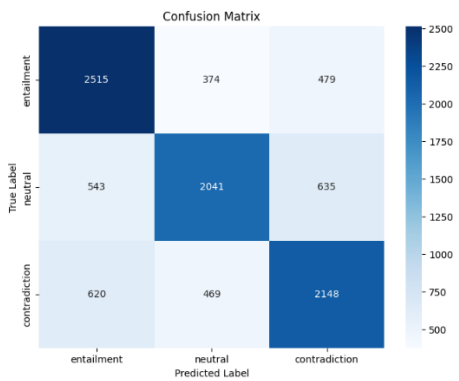
Adam optimizer and Categorical Cross Entropy Loss is used. A callback is implemented for better training of the model, that is Early Stopping.
Early Stopping monitors a specified quantity (here validation loss) and stops training when the quantity stops improving.

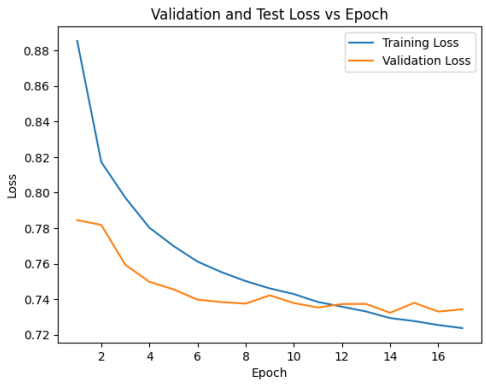
Classification report, Confusion Matrix aand Test Loss / Validation Loss vs Epoch graphs are generated from this model is shown below –

1.eSNLI

Test Loss: 0.740, Test Accuracy: 68.24%

	precision	recall	f1-score	support
entailment	0.68	0.75	0.71	3368
neutral	0.71	0.63	0.67	3219
contradiction	0.66	0.66	0.66	3237
accuracy			0.68	9824
macro avg	0.68	0.68	0.68	9824
weighted avg	0.68	0.68	0.68	9824

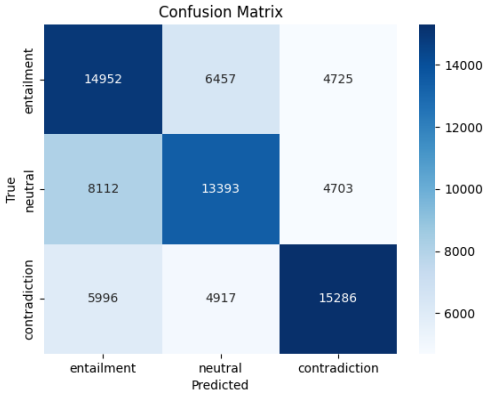


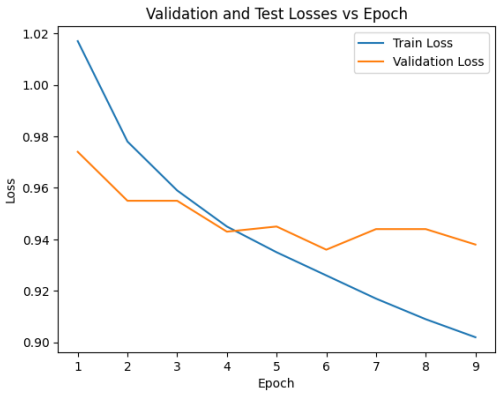


2.multiNLI

Test Loss: 0.949, Test Accuracy: 55.55%

	precision	recall	f1-score	support
entailment	0.51	0.57	0.54	26134
neutral	0.54	0.51	0.53	26208
contradiction	0.62	0.58	0.60	26199
accuracy			0.56	78541
macro avg	0.56	0.56	0.56	78541
weighted avg	0.56	0.56	0.56	78541

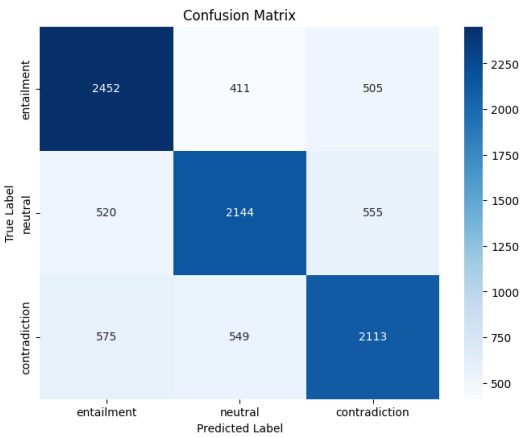


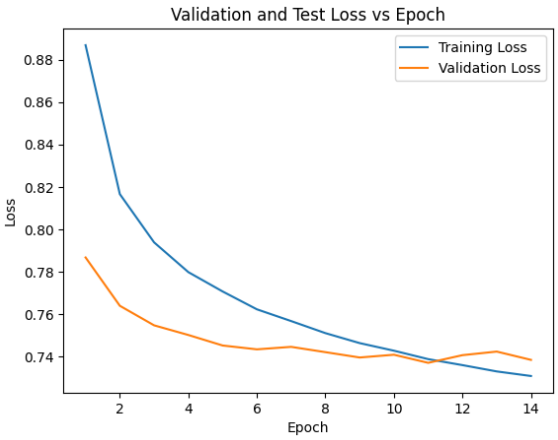


3. SNLI

Test Loss: 0.743, Test Accuracy: 68.29%

	precision	recall	f1-score	support
entailment	0.69	0.73	0.71	3368
neutral	0.69	0.67	0.68	3219
contradiction	0.67	0.65	0.66	3237
accuracy			0.68	9824
macro avg	0.68	0.68	0.68	9824
weighted avg	0.68	0.68	0.68	9824





4.3 Bidirectional GRU:

Bi-directional GRU (Gated Recurrent Unit) architecture-based model is also trained on the e-SNLI, 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.
The steps utilized are as follows:

Data Loading and Preprocessing: The code loads the eSNLI dataset using the load_dataset function and preprocesses it by converting text to lowercase.
Text Tokenization and Indexing: Text data is tokenized using the basic English tokenizer from torchtext. Tokens are then converted into indices based on a vocabulary built from the training data. 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. Furthermore, any Unicode characters have been eliminated from the premise and hypothesis

Model Definition (BiGRU): The BiGRU model is defined using PyTorch, consisting of an embedding layer, a bidirectional GRU layer, and a linear layer for classification.

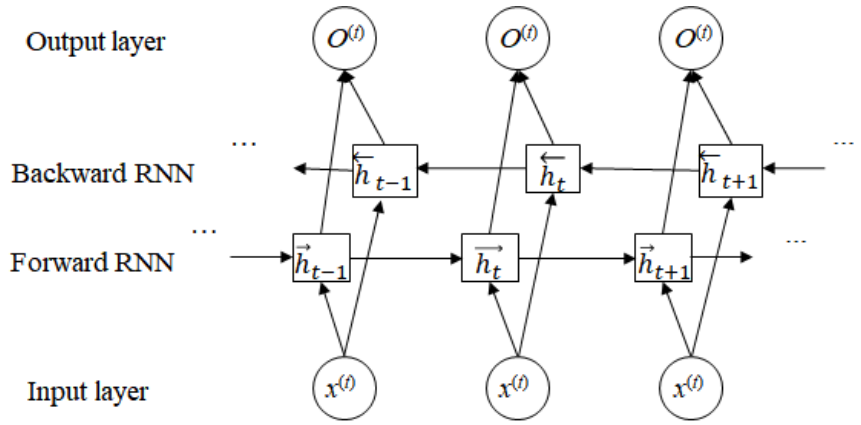
Training and Evaluation: The model is trained and evaluated using a training loop and evaluation functions. Training is performed for multiple epochs with early stopping based on validation loss.

Loss and Optimization: Cross-entropy loss is used as the loss function, and Adam optimizer is used for optimization.

Evaluation Metrics: We compute and prints various evaluation metrics such as accuracy, classification report, and confusion matrix.

Plotting: The code also includes plotting functionality to visualize the training and validation loss vs. epoch and the confusion matrix.

Architecture:



- Similar to BiLSTM, the input to the BiGRU network consists of sequences of tokens or words represented as word embeddings. The core of the BiGRU architecture comprises two GRU layers: one processes the input sequence in the forward direction, and the other processes it in the backward direction.

- Each GRU layer consists of a series of GRU cells, each of which contains an update gate and a reset gate. The GRU cells, similar to LSTM, capture dependencies within the input sequence over both forward and backward directions, enabling the network to capture long-range dependencies effectively.

- Similar to BiLSTM, the hidden states from the forward and backward GRUs are concatenated element-wise at each time step, effectively doubling the dimensionality of the hidden states.

- The concatenated hidden states are fed into an output layer, typically consisting of one or more fully connected layers followed by an activation function.
  The output layer predicts the target variable for classification tasks, such as sentiment analysis or named entity recognition.The final output of the BiGRU network is a prediction or classification score for each token in the input sequence.

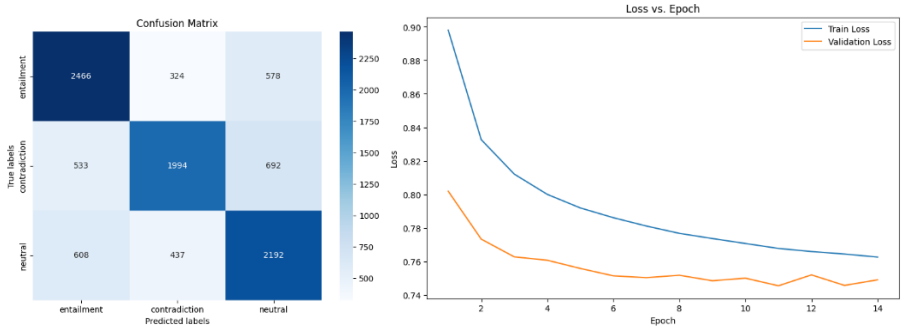
GRU is different from the LSTM in the sense that GRU has a smaller number of training parameters as compared to LSTM. There is no separate forget and update gate in GRU. They both are combined and thus the number of parameters reduced.

Classification report, Confusion Matrix and Test Loss / Validation Loss vs Epoch graphs are generated from this model is shown below –

1.eSNLI

Test Loss: 0.749, Test Accuracy: 67.71%

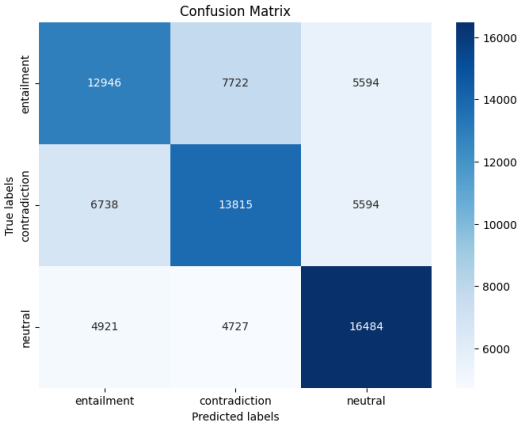
	precision	recall	f1-score	support
0	0.68	0.73	0.71	3368
1	0.72	0.62	0.67	3219
2	0.63	0.68	0.65	3237
accuracy			0.68	9824
macro avg	0.68	0.68	0.68	9824
weighted avg	0.68	0.68	0.68	9824

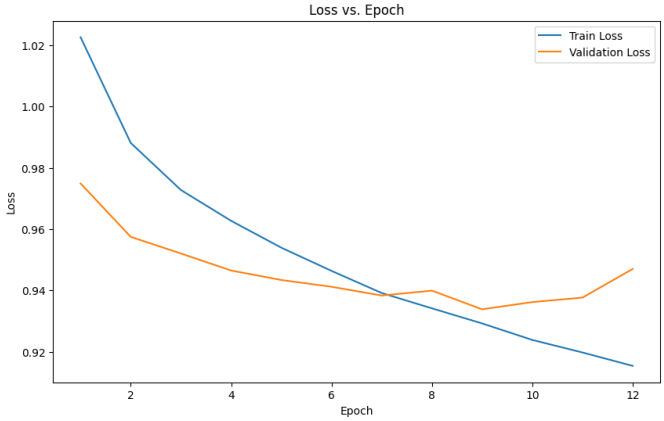


2.MultiNLI

Test Loss: 0.949, Test Accuracy: 55.06%

	precision	recall	f1-score	support
0	0.53	0.49	0.51	26262
1	0.53	0.53	0.53	26147
2	0.60	0.63	0.61	26132
accuracy			0.55	78541
macro avg	0.55	0.55	0.55	78541
weighted avg	0.55	0.55	0.55	78541

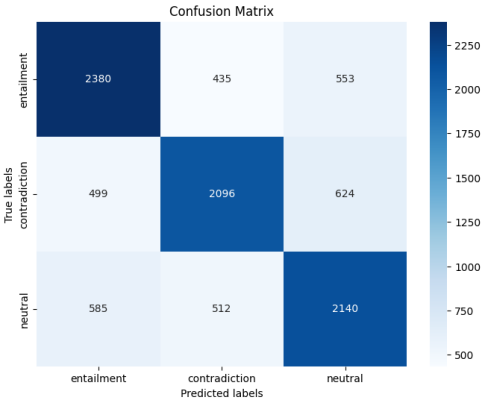


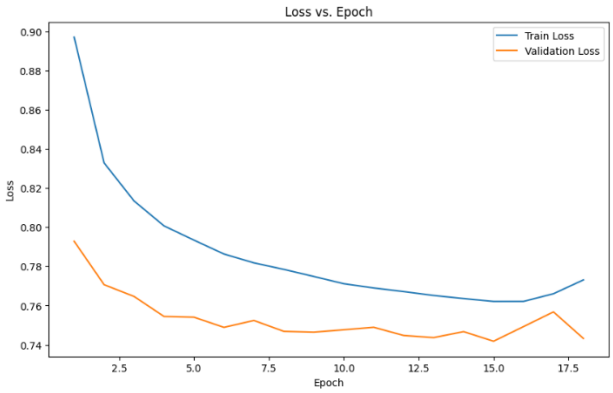


3.SNLI

Test Loss: 0.753, Test Accuracy: 67.35%

	precision	recall	f1-score	support
0	0.69	0.71	0.70	3368
1	0.69	0.65	0.67	3219
2	0.65	0.66	0.65	3237
accuracy			0.67	9824
macro avg	0.67	0.67	0.67	9824
weighted avg	0.67	0.67	0.67	9824



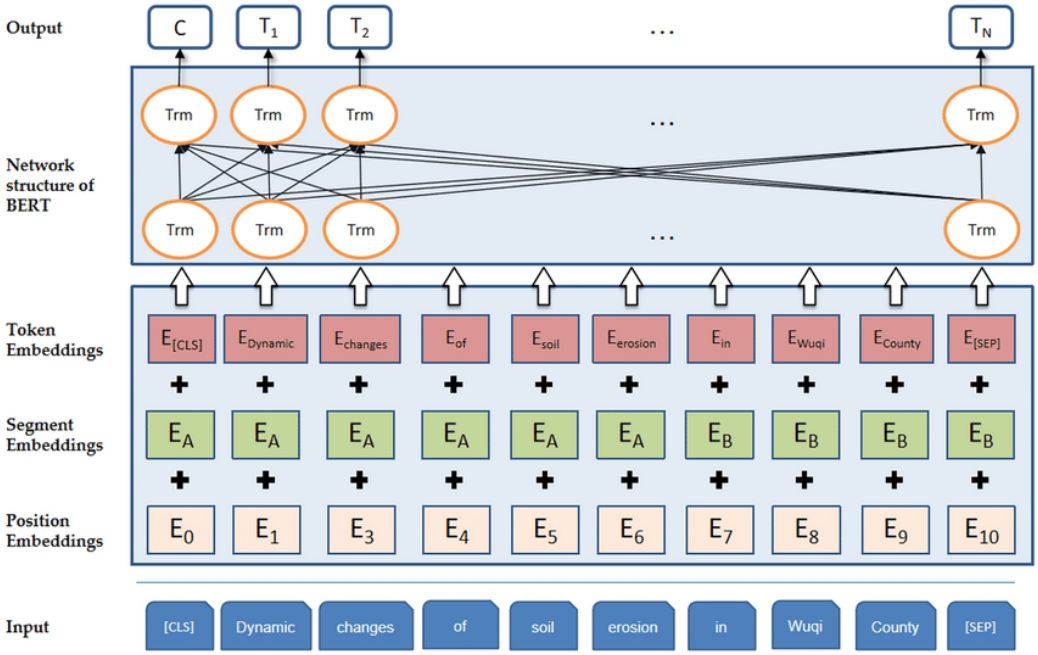


4.4 BERT:

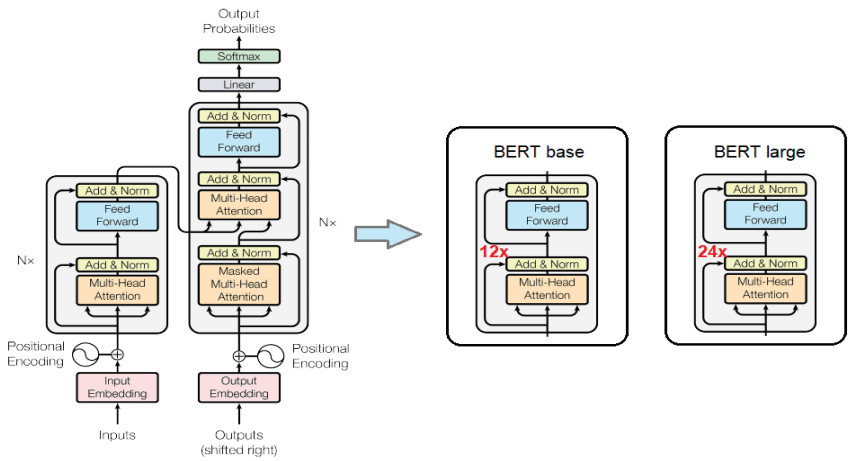
BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained natural language processing (NLP) model developed by Google. It is based on the transformer architecture and is trained using a large corpus of unlabelled text data. BERT is a deep neural network that can be fine-tuned for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

BERT model for sequence classification (BertForSequenceClassification) is loaded from the transformers library.

Architecture:



The pretrained model used :



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.

How the code works:

Importing Libraries: The code starts by importing necessary libraries such as PyTorch, transformers, tqdm, matplotlib, seaborn, etc.

Preprocessing the Dataset: The eSNLI dataset is loaded using the load_dataset function from the datasets library.The dataset is preprocessed by converting the text to lowercase and separating premises, hypotheses, and labels.

Tokenization: The BERT tokenizer is used to tokenize the text data.

Data Loading: The preprocessed data is converted into PyTorch tensors and loaded into data loaders for training, validation, and testing.

Model Definition: BERT model for sequence classification ***(BertForSequenceClassification)*** is loaded from the transformers library.

Optimizer and Scheduler: AdamW optimizer and linear scheduler with warmup are defined.

Train Loss: 0.414 Train Accuracy: 83.90%,
Val Loss: 0.310 Val Accuracy: 88.54%

As seen, an accuracy of 89% has been achieved through BERT based model.
Note:- Due to the resource scarcity, the BERT based model is trained only for 3 epochs.

---

5. Applications of NLI

Natural Language Inference (NLI) has various applications in Natural Language Processing (NLP) due to its ability to understand and reason about relationships between pieces of text. Some applications of NLI include:

Question Answering Systems: NLI can be used to determine whether a given statement or passage logically entails, contradicts, or is neutral with respect to a given question. This is useful in building question answering systems where the system needs to reason about whether the answer to a question is supported by the provided information.

Textual Entailment: NLI models can be used to determine whether one piece of text logically entails another. This is useful in tasks like paraphrase detection, where determining if two sentences convey the same meaning is important.

Information Retrieval: NLI can aid in improving the precision of information retrieval systems by identifying documents or passages that entail the user's query, even if they do not contain the exact same wording.

Automated Reasoning: NLI models can be used in automated reasoning systems to verify logical relationships between statements, which is useful in various domains such as theorem proving, semantic verification, and logical reasoning tasks.

Machine Translation: NLI can be utilized in machine translation systems to improve the fluency and coherence of translated text by ensuring that the translated sentences logically entail the original ones.

Text Summarization: NLI can assist in generating coherent and logically consistent summaries by ensuring that the summarized text logically entails the important information from the original text.

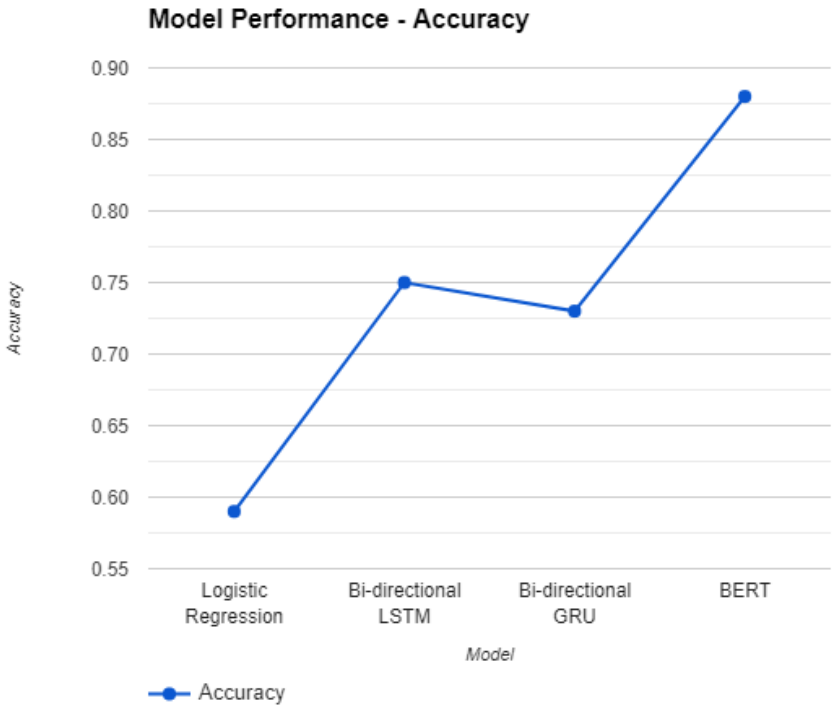
Semantic textual similarity (STS) : Comparison of sentence pairs. We may want to identify patterns in datasets.

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) and BERT.

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

Model	Precision			Recall			F1-Score			Accuracy
	L0	L1	L2	L0	L1	L2	L0	L1	L2	
Logistic Regression	0.50	0.55	0.52	0.59	0.47	0.50	0.54	0.51	0.51	52.31%
Bi-directional LSTM	0.68	0.61	0.66	0.75	0.63	0.66	0.71	0.67	0.66	68.24%
Bi-directional GRU	0.68	0.72	0.63	0.73	0.62	0.68	0.71	0.67	0.65	67.71%
BERT	0.91	0.90	0.84	0.89	0.91	0.85	0.89	0.90	0.84	88.54%



7. Qualitative Analysis

1. Logistic Regression:

- Precision: The precision values across all labels (L0, L1, L2) are relatively low, indicating that when this model predicts a class, it is correct about half the times.
- Recall: The recall values vary, with higher recall for L1 compared to L0 and L2. This suggests that the model is better at capturing instances of L1, but it misses a significant number of instances of L0 and L2.
- F1-Score: The F1-scores are generally low, indicating a balance between precision and recall. However, they are not particularly high for any level.
- Accuracy: The overall accuracy is 52.31%, which is low compared to the other models.

2. Bi-directional LSTM:

- Precision: The precision values are higher compared to Logistic Regression, indicating that this model makes fewer false positive predictions.
- Recall: The recall values are also higher, indicating that this model captures more instances of the classes, especially for L1 and L2.
- F1-Score: The F1-scores are the highest among the models listed, indicating a good balance between precision and recall.
- Accuracy: The accuracy of 68.24% is significantly higher compared to Logistic Regression.

3. Bi-directional GRU:

- Precision: Similar to the LSTM model, the precision values are higher compared to Logistic Regression.
- Recall: The recall values are also higher, especially for L1, indicating that this model captures more instances of the classes.
- F1-Score: The F1-scores are slightly lower compared to the LSTM model but still higher than Logistic Regression.
- Accuracy: The accuracy of 67.71% is comparable to the LSTM model.

4. BERT:

- Accuracy: BERT achieves the highest accuracy of 88.54%, which is significantly higher than the other models listed. However, since precision, recall, and F1-score values are not provided, it's hard to make a detailed qualitative analysis based solely on accuracy.

In summary, the deep learning models (Bi-directional LSTM and Bi-directional GRU) outperform the traditional Logistic Regression model in terms of precision, recall, F1-score, and accuracy. Among the deep learning models, Bi-directional LSTM performs slightly better than Bi-directional GRU. BERT achieves the highest accuracy, but without precision, recall, and F1-score values, it's challenging to compare its performance in detail with the other models.

8. 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

[2] Narang, S., Raffel, C., Lee, K., Roberts, A., Fiedel, N., & Malkan, K. (2020). WT5?! Training Text-to-Text Models to Explain their Predictions. arXiv preprint arXiv:2004.14546

[3] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805

[4] Lee, K., Lin, H., & Chen, Y. (2019). NCUEE at MEDIQA 2019: Medical Text Inference Using Ensemble BERT-BiLSTM-Attention Model. In Proceedings of the ACL-BioNLP Workshop 2019 (pp. 50-58). Association for Computational Linguistics. ACL Anthology