Group 11: Language Translation Service

Vikas Manchikant, Priyanka Chakraborty, Bhanu Sahith Bonthula
Department of Computer Science
University at Buffalo
vikasman@buffalo.edu
priyank2@buffalo.edu
bhanusah@buffalo.edu

Abstract

A neural machine translation model is being developed as part of this project in order to overcome the difficulty of translating from English to French. Through the use of a sequence-to-sequence architecture, a bidirectional LSTM, and an attention mechanism, we have developed a system that is capable of efficiently capturing the intricate details of both languages. In order to evaluate the performance of the model, a comprehensive evaluation using BLEU metric, in addition to accuracy tests, has been carried out. According to our findings, the combination of a Bi-LSTM with Attention model, CrossEntropyLoss, and the Adam optimizer results in great translation quality, surpassing the quality of other combinations that were explored. Furthermore, a real-time translation front-end interface is used to show the actual deployment of this model. This interface helps bridge the gap between theoretical study and the use of this model in the real world. Not only do the findings demonstrate that the model is capable of producing translations of exceptional quality, but they also demonstrate that it has the potential to be easily integrated into software that is used for language translation.

1 Dataset

1.1 Description of the Dataset:

Our dataset has 2 columns one column has english words/sentences and the other one has french words/sentences and 175622 rows. We have taken this dataset with the intention of making it simpler to construct a language translation model. Due to the fact that it contains a large number of distinct words, the model is robust and is able to accommodate real-life variations in language usage. These variations range from simple common phrases to complex grammatical structures. The information was carefully put together to be fair and showing the features of both languages.

1.2 Data Engineering Used for the Dataset

We conducted a range of Exploratory Data Analysis (EDA) activities on the dataset, aimed at prepping it for text analysis and model training. This involved thorough cleaning and processing of the textual data to ensure its quality.

1.3 Exploratory Data Analysis

The EDA analysis which we performed are given in below steps:

1. **Sentence Length and Word Count Distributions**:

- **English Sentence Lengths**:
  The distribution indicates that most English sentences are of moderate length, with a balanced spread across different lengths
- **French Sentence Lengths**:
  Similar to English, French sentences also predominantly fall within a moderate length range, but there might be slight variations in the distribution pattern compared to English.
- **English Word Counts**:
  Most English sentences have a relatively low word count, suggesting a prevalence of shorter phrases or sentences.
- **French Word Counts**:
  The French sentences show a similar pattern to the English ones, with most sentences having fewer words, indicating a collection of brief phrases or sentences.

2. **Total and Unique Word Counts**:
   The analysis revealed a significant difference in the total and unique word counts between English and French. Both languages have a substantial number of total words, but French has a notably higher count of unique words. This suggests a richer diversity in vocabulary usage in the French part of the dataset.
3. **Most Frequent Words**:
   - **English**:
     The most frequent words are common pronouns, articles, and verbs, reflecting typical usage in everyday language. The name "Tom" appears frequently, which might indicate specific thematic content or source material.
   - **French**:
     Similar to English, common functional words dominate the list. However, the presence of frequent question marks indicates a higher number of questions in the French sentences.
4. **Sentence Length Categories**:
   When categorized as short, medium, and long, both English and French sentences predominantly fall into the 'short' category, further affirming the presence of concise sentences in the dataset.
5. **Start and End Words of Sentences**:
   The analysis of the words that commonly start and end sentences in both languages shows a clear pattern of sentence construction, with a focus on personal pronouns and simple structure in English, and a similar trend in French with a notable emphasis on questions.
6. **Character Distribution**:
   The character frequency analysis was not detailed in the provided information but typically includes an examination of the prevalence of different letters, punctuation marks, and other characters in the text.

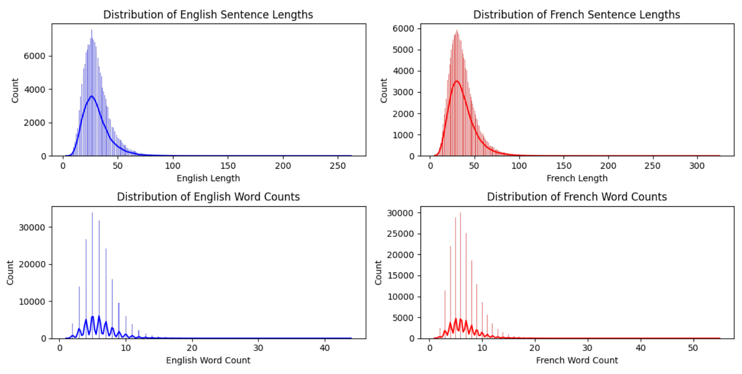


Figure 1: Sentence Length and Word Count Distributions

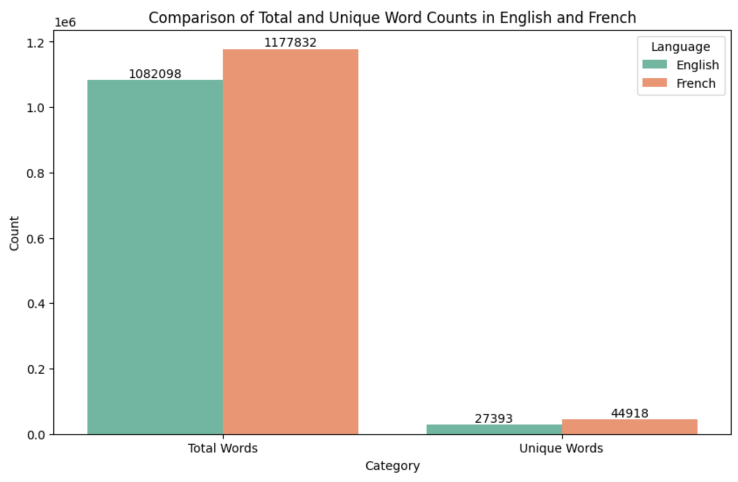


Figure 2: Total and Unique Word Counts

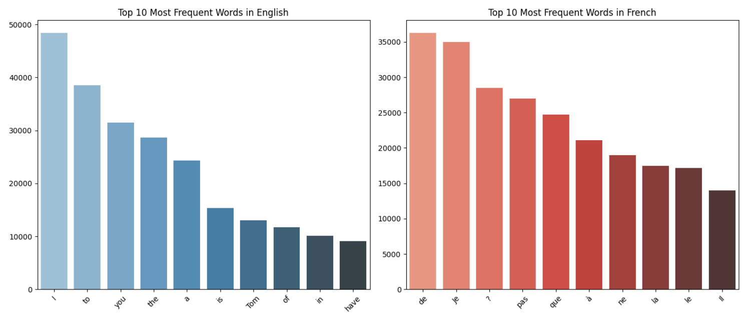


Figure 3: Figure: Most Frequent Words

1.3.1 Text Preprocessing

Text preprocessing is a critical step in natural language processing. For our dataset, this involved:

Tokenization: Splitting text into individual tokens (words and punctuation) using Spacy's encorewebsm and frcorenewssm models for English and French, respectively.

Cleaning: Removing unwanted characters and formatting to ensure consistency.

Normalization: Lowercasing all tokens to reduce the vocabulary size and streamline the learning process.

1.3.2 Numerical Encoding

After preprocessing, we converted the text data into numerical form through the following steps:

Vocabulary Building: Creating a vocabulary index for each language using a tokenizer function and aggregating all unique tokens from the dataset.

Text Transformation: The texttransform dictionary includes lambda functions for both the source and target text, which convert the text into a sequence of numerical indices, with special tokens for the start and end of sentences

Batch Preparation: In the collatebatch function, the source and target sentences are converted to tensors of indices, and then padding is applied using the padsequence function from torch.nn.utils.rnn. This ensures that all sequences in a batch have the same length, which is a requirement for batch processing in neural networks.

Token to Index Mapping: Assigning a unique integer to each token in the vocabulary, enabling the conversion of textual data into a numerical format that our models can interpret.

1.3.3 Model-Specific Data Formatting

Finally, to make the data amenable to our models, we formatted it according to the requirements of sequence-to-sequence learning:

Padding: Sequences were padded to a fixed length to ensure batch consistency during training.

Start/End Tokens: Special tokens were added to signal the beginning and end of sentences, aiding the model in recognizing sentence boundaries.

Batching: The dataset was divided into batches, each containing a set number of sentence pairs, to optimize the training process and resource utilization.

These data engineering processes were very important in the process of developing precise and uniform representations of the customer evaluations, which successfully prepared the dataset for the training of the language translation model. By making use of these processed data inputs, the models were able to acquire pertinent patterns and relationships from the textual feedback, which ultimately resulted in correct language translation and valid predictions.

2 Model Description

2.1 Seq2Seq Model with Bidirectional LSTM and Attention Mechanism

1. **Seq2Seq Model Overview**:
   A Seq2Seq model is typically used in natural language processing tasks like machine translation. It consists of two main components: an encoder and a decoder. The encoder processes the input sequence and compresses the information into a context vector (sometimes called the state). The decoder then uses this vector to generate the output sequence.
2. **Bidirectional LSTM (Encoder)**:
   In your model, the encoder uses a bidirectional LSTM. An LSTM is a type of recurrent neural network (RNN) capable of learning long-term dependencies in data. It achieves this by having a memory cell that can maintain information in memory for long periods.
   The bidirectional aspect means that your LSTM runs in two directions:
   - one for forward pass
   - one for backward pass

   This allows the network to have both past and future context. Each LSTM cell takes an input

   vector $\mathbf{x}_t$, and gives an output vector $\mathbf{h}_t$ and maintains a memory state $\mathbf{c}_t$. The LSTM cell updates are given by:
   - **Forget gate**:

     $$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

   - **Input gate**:

     $$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

   - **Cell state**:

     $$\widetilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C)$$

- **Final cell state**:

$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \widetilde{\mathbf{C}}_t$$

- **Output gate**:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

- **Hidden state**:

$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{C}_t)$$

Where σ is the sigmoid function, $\mathbf{W}$ and $\mathbf{b}$ are weights and biases, and $*$ denotes element-wise multiplication.

3. **Attention Mechanism**:
The attention mechanism allows the decoder to focus on different parts of the encoder's outputs for each step in the output sequence. Essentially, it computes a context vector by weighing the encoder's output states based on their relevance to the current decoder state.

Given a decoder hidden state $\mathbf{h}_t$ and a set of encoder outputs $\mathbf{h}^1, \mathbf{h}^2, \ldots, \mathbf{h}^N$, the attention mechanism computes the context vector $\mathbf{c}_t$ as follows:

- **Compute energy scores**:
$$e_{tj} = \mathbf{v}^T \tanh(\mathbf{W}[\mathbf{h}^j; \mathbf{h}_t])$$
- **Compute attention weights**:
$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_k \exp(e_{tk})}$$
- **Compute context vector**:
$$\mathbf{c}_t = \sum_j \alpha_{tj} \mathbf{h}^j$$

Here, $\mathbf{W}$ and $\mathbf{v}$ are trainable parameters of the attention mechanism.

2.1.1 OUR ARCHITECTURE FOR THE LSTM MODEL

- **Encoder:** Embeds input tokens and processes them through a bidirectional LSTM. The final state is passed through a fully connected layer to produce a context vector.
- **Attention:** Calculates attention weights based on the encoder output and the current state of the decoder, producing a weighted sum of encoder outputs (context vector).
- **Decoder:** Takes one token at a time, uses the attention mechanism to focus on specific parts of the input sequence, and generates the output sequence token by token.
- **Seq2Seq:** Orchestrates the encoder and decoder, managing the flow of data and the use of teacher forcing during training.

2.1.2 Training Process

- **Optimization and Monitoring Progress:** In the training process, optimization techniques are utilized to enhance model performance and monitor progress. These techniques help stabilize and expedite the learning process.
- **Teacher Forcing and Evaluation:** To facilitate learning, the model employs **teacher forcing**, which aids in guiding the training process. Additionally, the model evaluates its progress by measuring how well it aligns with the desired outcomes.
- **Gradient Clipping for Stability:** To ensure that the model's gradients do not become too large and lead to unstable training, **gradient clipping** is applied.
This technique scales down gradients that exceed a threshold θ by a factor of $g^*/\|g\|_\theta$.

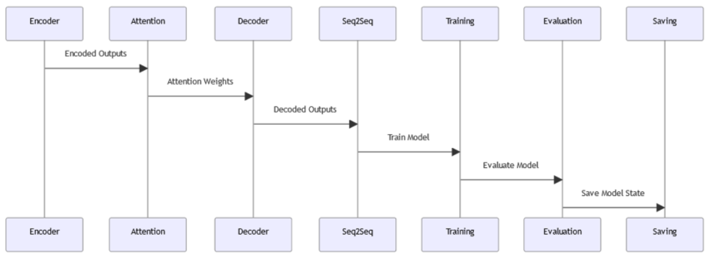


Figure 4: Seq2Seq Model with Bidirectional LSTM and Attention Mechanism

2.2 Seq2Seq Model with GRU

1. **Introduction**
   The Seq2Seq model with Gated Recurrent Units (GRU) is a neural network architecture primarily used for sequence-to-sequence tasks like machine translation. This model comprises two key components: an encoder and a decoder, both utilizing the GRU architecture. GRUs are a type of recurrent neural network (RNN) known for their efficiency in capturing long-term dependencies and mitigating the vanishing gradient problem common in standard RNNs.
2. **GRU in Encoder**
   The encoder's role is to process the input sequence and compress its information into a context vector, a representation of the input sequence's critical features. In this implementation, the encoder uses GRUs which are defined by the following formulas:
   - **Update Gate:**
     - $z_t = \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t])$
   - **Reset Gate:**
     - $r_t = \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t])$
   - **Candidate Hidden State:**
     - $\mathbf{h}_t^{\sim} = \tanh(\mathbf{W} \cdot [r_t * \mathbf{h}_{t-1}, \mathbf{x}_t])$
   - **Final Hidden State:**
     - $\mathbf{h}_t = (1 - z_t) * \mathbf{h}_{t-1} + z_t * \mathbf{h}_t^{\sim}$

   Here, σ denotes the sigmoid function,
   $\mathbf{W}$s are weight matrices,
   $\mathbf{h}$ is the hidden state, and
   $\mathbf{x}$ is the input at each time step.
3. **GRU in Decoder**
   The decoder's purpose is to generate the output sequence from the context vector. It also employs GRUs, but with additional inputs from the encoder's context vector and previously generated outputs. The decoder's GRU operates on similar principles to the encoder's but focuses on generating the next output in the sequence.

2.2.1 OUR ARCHITECTURE FOR THE GRU MODEL

In the Seq2Seq model, data flows from the encoder to the decoder. The encoder processes the input sequence and generates a context vector. This context vector is then passed to the decoder, which uses it, along with its previous states, to generate the output sequence. The key is that the decoder's output at each step feeds into its next step, creating a recursive generation process.

2.2.2 Training process

The training process involves optimizing both the encoder and decoder in tandem. The model is trained to minimize the difference between the predicted output sequence and the actual target sequence. This is achieved through backpropagation and gradient descent algorithms. We experimented with the model using two types of loss functions, three types of metrics, and two types of criteria. they play a crucial role in guiding the training process, affecting how the model learns to generate more accurate outputs. the Seq2Seq model with GRU is a powerful architecture for handling sequence-to-sequence

tasks, offering a balance between computational efficiency and the ability to capture complex patterns in data.

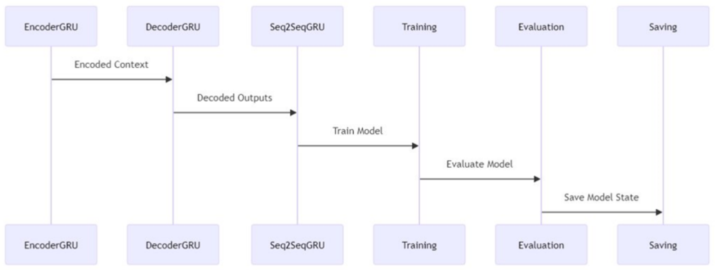


Figure 5: Seq2Seq Model with GRU

2.3 BERT

1. **Introduction** BERT (Bidirectional Encoder Representations from Transformers) represents a significant advance in natural language processing (NLP). It has transformed how machines understand and interpret human language by comprehensively analyzing the context of each word in a sentence.
   - **Bidirectional Context Analysis:** BERT is designed to understand the context of a word based on all of its surroundings (left and right of the word).
   - **Transformer Architecture:** Utilizes attention mechanisms, enabling the model to focus on different parts of the input sequence for better context understanding.
   - **Pre-training and Fine-Tuning:** BERT is first pre-trained on a large corpus of text and then fine-tuned for specific tasks, making it versatile and adaptable.

2.3.1 Overall Architecture and Data Flow of the Model

1. **Model Architecture**
   - **BERT as Encoder:** At the heart of the model, BERT encodes input text into a high-dimensional space, capturing the nuanced context of each word.
     Encoding formula: $\text{encoded} = \text{bert}(\text{input\_ids})[0]$
   - **Linear Decoder Layer:** This layer translates BERT's output into the target language's vocabulary.
     Decoder formula: $\text{output} = \text{decoder}(\text{encoded})$
2. **Data Flow Process**
   - **Input Tokenization:** Input sentences are tokenized into BERT-readable formats, transforming words into numerical representations.
     Tokenization formula: $\text{tokens} = \text{tokenizer.encode}(\text{text})$
   - **Contextual Encoding:** The BERT encoder processes these tokens to create contextual embeddings, representing both the word and its contextual relevance.
   - **Decoding and Output Generation:** The decoder then converts these embeddings into a sequence of tokens representing the translated text in the target language.

2.3.2 Training Loop Mechanics

1. **Batch Processing:** The model is trained in batches, allowing for efficient processing and optimization.

2. **Forward Pass:** Each batch of tokenized text undergoes the encoding-decoding process. **Forward pass formula:** $\text{output} = \text{model}(\text{input\_batch})$
3. **Backpropagation and Weight Adjustment:** The model updates its weights in response to the training data, refining its ability to translate accurately.

2.3.3 Model Fine-Tuning

Fine-tuning involves adjusting the BERT model on the specific translation task, which enhances its performance and accuracy for the given language pair.

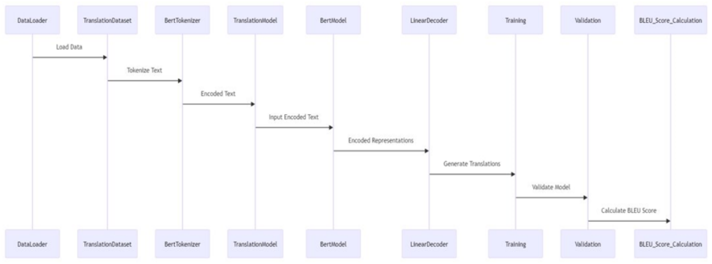


Figure 6: Bert Model

3 Loss Function

In the realm of machine learning and deep learning, loss functions play a critical role. They are fundamental in guiding algorithms towards accuracy and efficiency. Essentially, a loss function is a mathematical method used to measure how well a machine learning model performs. It quantifies the difference between the predicted outputs of the model and the actual target values.

3.1 CrossEntropyLoss

CrossEntropyLoss is a widely used loss function in classification tasks, especially in scenarios involving multiple classes. It measures the performance of a classification model whose output is a probability value between 0 and 1. CrossEntropyLoss increases as the predicted probability diverges from the actual label.

$\text{CrossEntropyLoss} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$

Where:

- M is the number of classes.
- y is a binary indicator (0 or 1) if class label c is the correct classification for observation o.
- p is the predicted probability of observation o being of class c.

In practice, CrossEntropyLoss combines LogSoftmax and NLLLoss (Negative Log Likelihood Loss) in a single class. It is equivalent to applying LogSoftmax to the last layer of the model followed by NLLLoss.

3.2 NLLLoss (Negative Log Likelihood Loss)

NLLLoss is another loss function commonly used in classification problems, particularly with classification models like neural networks. It is frequently used in conjunction with Softmax activation in the final layer of a neural network.

$\text{NLLLoss} = -\sum_{i=1}^{N} \log(p_i)$

Where:

- N is the number of classes.
- p_i is the probability predicted by the model for the correct class.

In simple terms, NLLLoss measures the likelihood of the correct class's probability, penalizing more when the model's output distribution is further away from the truth distribution.

4 Optimization Algorithm

In machine learning, optimizers are algorithms or methods used to change the attributes of the neural network, such as weights and learning rate, to reduce the losses. Optimizers guide the training process and are crucial for the model's convergence to a minimum loss.

4.1 Adam

Adam is an algorithm for gradient-based optimization of stochastic objective functions, combining ideas from RMSprop and SGD with momentum.

$\theta = \theta - \frac{\hat{m}}{\sqrt{\hat{v}}+\epsilon} \cdot \eta$

Where:

- $\hat{m}$ and $\hat{v}$ are estimates of the first and second moments of the gradients.
- η is the learning rate.
- ϵ is a small scalar (to prevent division by zero).

4.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$

Where:

- θ represents parameters (weights).
- η is the learning rate.
- $\nabla_\theta J$ is the gradient of the loss function J with respect to the parameters θ for a single data point $(x^{(i)}, y^{(i)})$.

4.3 RMSprop (Root Mean Square Propagation)

RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in his course.
$\theta = \theta - \frac{E[g^2]}{\sqrt{E[g^2]+\epsilon}} \cdot \eta \cdot g$

Where:

- $E[g^2]$ is the running average of the squared gradients.
- g is the gradient.
- η is the learning rate.
- ϵ is a small scalar.

4.4 AdamW

AdamW is a tweak to the original Adam optimizer that decouples the weight decay from the optimization step.

1. **Moment Updates**

- First Moment (mean) update:

$$\mathbf{m}_t = \beta_1 \cdot \mathbf{m}_{t-1} + (1 - \beta_1) \cdot \mathbf{g}_t$$

- Second Moment (uncentered variance) update:

$$\mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1 - \beta_2) \cdot (\mathbf{g}_t^2)$$

Where:

  - $\mathbf{m}_t$ is the first moment vector (moving average of the gradients).
  - $\mathbf{v}_t$ is the second moment vector (moving average of the squared gradients).
  - $\mathbf{g}_t$ is the gradient at time step t.
  - β_1 and β_2 are the exponential decay rates for these moment estimates.

2. **Bias Correction**

   - Corrected first moment:
     $\widehat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}$
   - Corrected second moment:
     $\widehat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}$

3. **Parameter Update**

   - The parameters are updated as follows:
     $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{\mathbf{v}}_t}+\epsilon} \cdot \widehat{\mathbf{m}}_t$
     Here, η is the learning rate and ϵ is a small constant added for numerical stability.

4. **Weight Decay Integration**

   - The parameters are updated as follows:
     $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{\mathbf{v}}_t}+\epsilon} \cdot \widehat{\mathbf{m}}_t - \eta \cdot \text{wd} \cdot \theta_t$
     Where wd represents the weight decay rate.

Each optimizer has its own strengths and is suited for specific kinds of problems. SGD is simple and effective for large datasets, Adam provides fast convergence with adaptive learning rates, RMSprop is beneficial for recurrent neural network tasks, and AdamW offers improved generalization by modifying the weight decay process. The choice of optimizer can significantly affect the performance of a neural network, and thus it is an important aspect of designing a machine learning model.

5 Metrics

1. **BLEU Score (Bilingual Evaluation Understudy)**
   BLEU is a popular metric in machine translation. It compares the machine-generated translation of text with reference translations to quantify the translation's quality.

   BLEU score is calculated based on n-gram precision, often combined with a brevity penalty to discourage overly short translations.

   $BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$

   Where:

   - p_n is the precision of n-grams
   - w_n is the weight for each n-gram
   - BP is the brevity penalty to penalize short translations

2. **Accuracy** Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.
   $Accuracy = \frac{TotalNumberofPredictions}{NumberofCorrectPredictions}$

6 Experimental Results

Based on the training of models using various optimizers and loss functions, the following insights have been gained

Sequence-to-Sequence (Seq2Seq) Model Utilizing Bidirectional LSTM and Attention Mechanism, Employing the Adam Optimizer and Cross-Entropy Loss Function achieve the highest bleu score of 35.6 here 7

Sequence-to-Sequence (Seq2Seq) Model Utilizing Bidirectional LSTM and Attention Mechanism, Employing the AdamW Optimizer and NLLLoss Function follows closely with an bleu score of 31.34 here 10

Sequence-to-Sequence (Seq2Seq) Model Utilizing Bidirectional LSTM and Attention Mechanism, Employing the RMSprop Optimizer and CrossEntropyLoss Function follows closely with an bleu score of 30.03 provided here 11

Seq2Seq Model with GRU, Employing the Adam Optimizer and CrossEntropyLoss Function follows closely with an bleu score of 30.23 here 8

Seq2Seq Model with GRU, Employing the SGD Optimizer and NLLLoss Function follows closely with an bleu score of 25.58 provided here 9

BERT model with AdamW optimizer and CrossEntropyLoss follows closely with an bleu score of 31.97 demonstrating competitive performance in sentiment classification. 12

Composite Performance Graph of Various Models: This composite graph illustrates the accuracy achieved by different models employing various configurations. On the graph, the x-axis corresponds to the number of epochs, while the y-axis depicts the loss for each model. The graph enables a visual comparison of model performance, facilitating an understanding of how different combinations of models, optimizers, and loss functions impact their accuracy in the assigned task. Refer to Figure 13 for visual details.

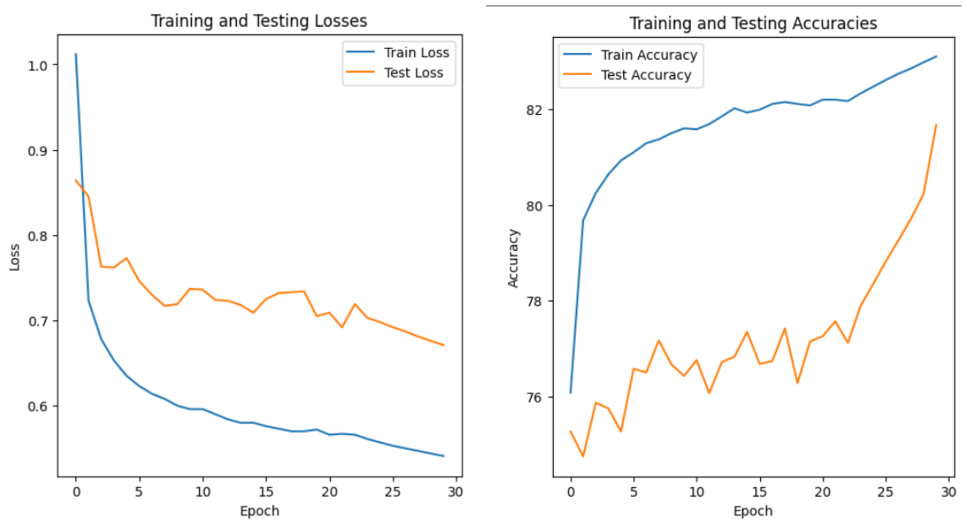


Figure 7: Seq2Seq Model with Bidirectional LSTM and Attention Mechanism, Employing the Adam Optimizer and Cross-Entropy Loss Function:- Left: Loss VS Epoch Right: Accuracy VS Epoch

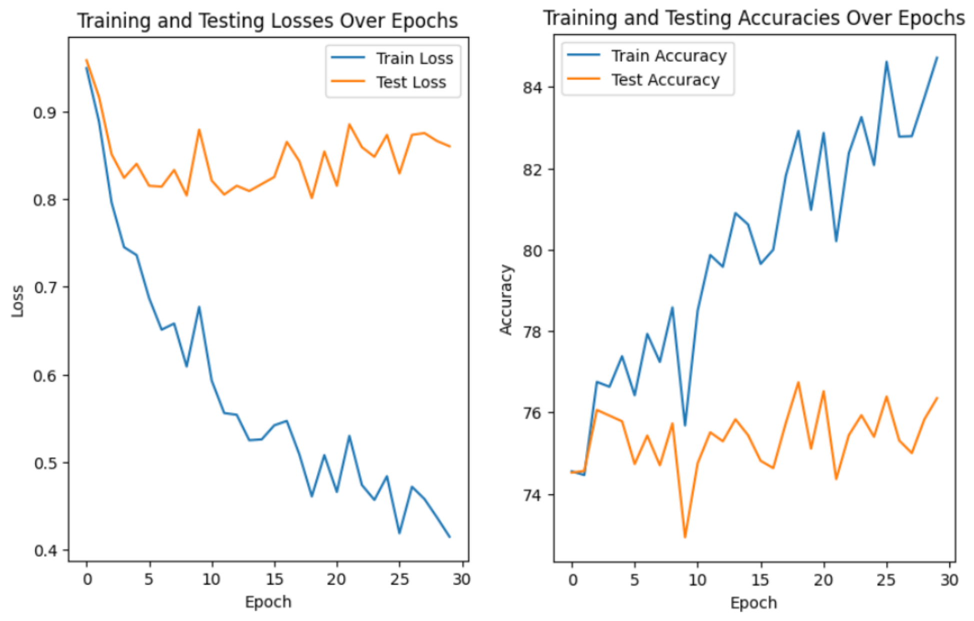


Figure 8: Seq2Seq Model with GRU, Employing the Adam Optimizer and CrossEntropyLoss Function:- Left: Loss VS Epoch Right: Accuracy VS Epoch

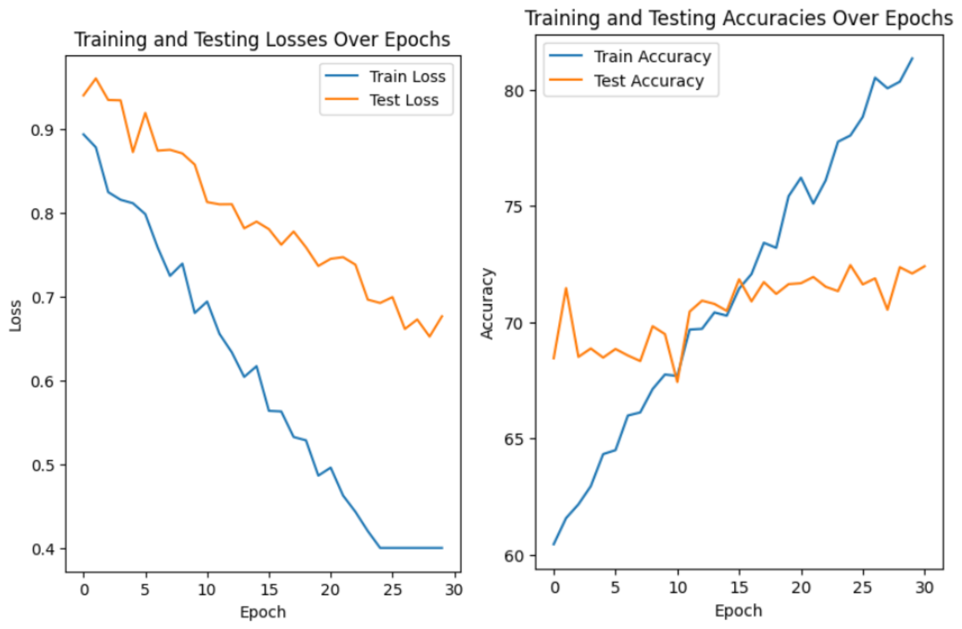


Figure 9: Seq2Seq Model with GRU, Employing the SGD Optimizer and NLLLoss Function:- Left: Loss VS Epoch Right: Accuracy VS Epoch

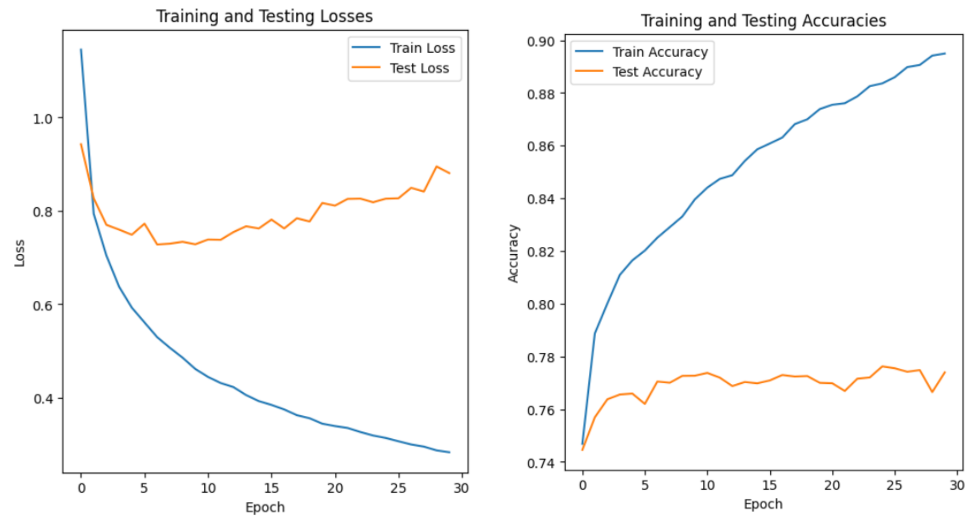


Figure 10: Sequence-to-Sequence (Seq2Seq) Model Utilizing Bidirectional LSTM and Attention Mechanism, Employing the AdamW Optimizer and NLLLoss Function:- Left: Loss VS Epoch Right: Accuracy VS Epoch

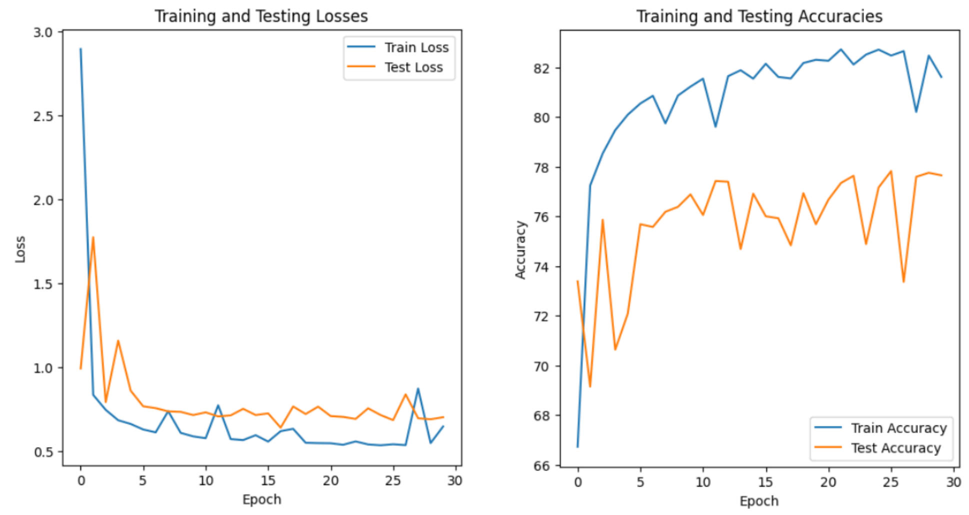


Figure 11: Sequence-to-Sequence (Seq2Seq) Model Utilizing Bidirectional LSTM and Attention Mechanism and Attention Mechanism, Employing the RMSprop Optimizer and CrossEntropyLoss Function:- Left: Loss VS Epoch Right: Accuracy VS Epoch

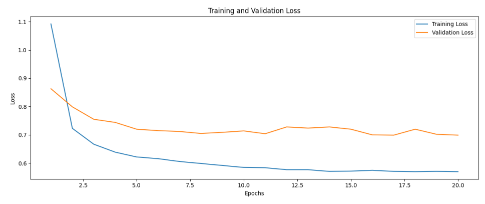


Figure 12: BERT

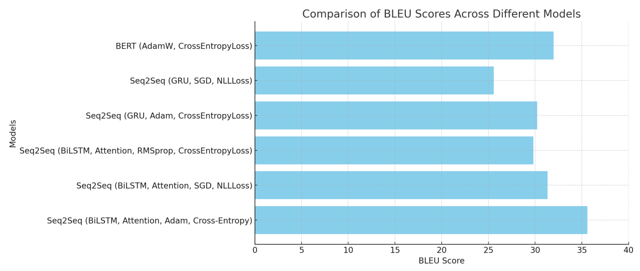


Figure 13: Bleu scores of all models

7 Demo

We deployed our model in Google Cloud Platform(GCP),
The URL for accessing the demo is **http://35.245.152.69:5000/**.
We designed our web interface with Flask, HTML, CSS.

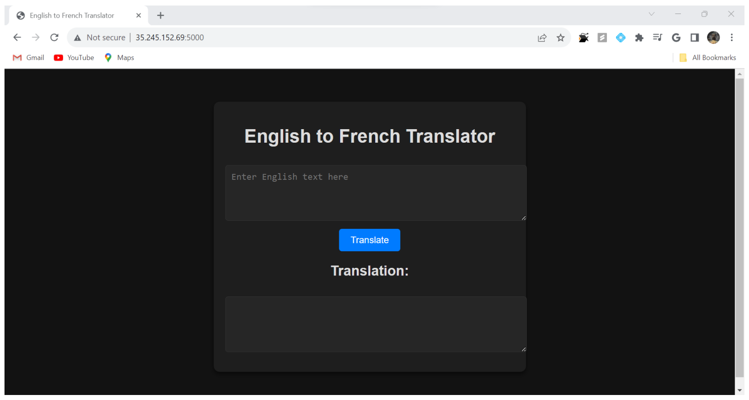
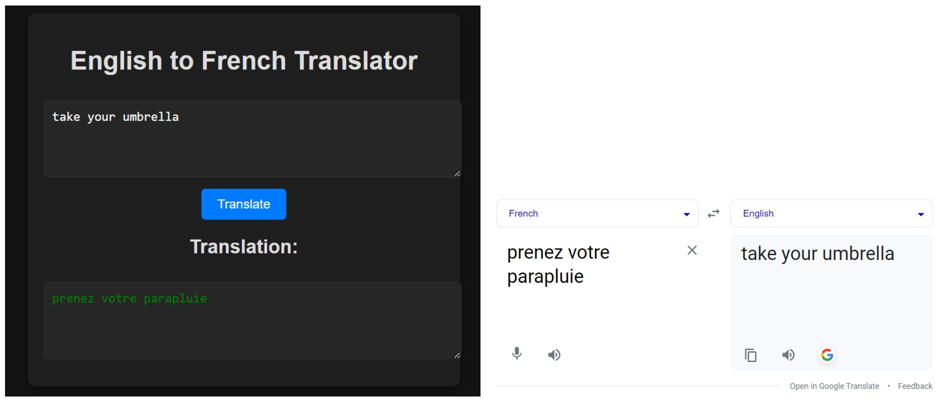

English to French Translator
Not secure | 35.245.152.69:5000
Gmail
YouTube
Maps
All Bookmarks
English to French Translator
Enter English text here
Translate
Translation:

Figure 14: Figure: Model Frontend



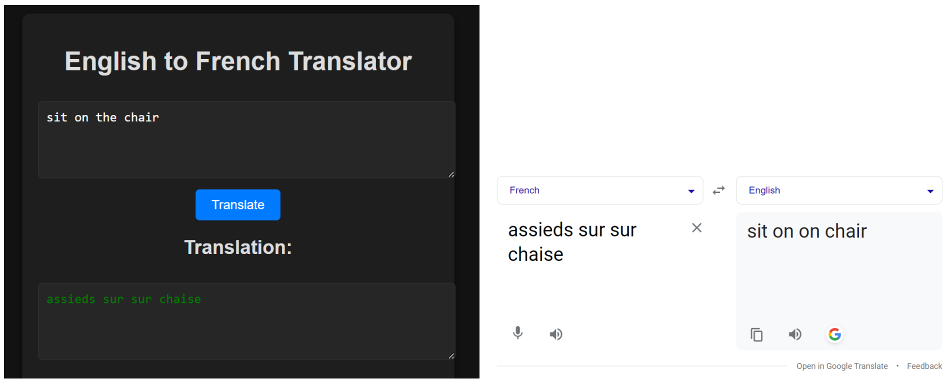
take your umbrella

prenez votre parapluie

prenez votre parapluie

take your umbrella

Figure 15: Demo 1


English to French Translator
Translate
Translation:
French
English
Open in Google Translate • Feedback

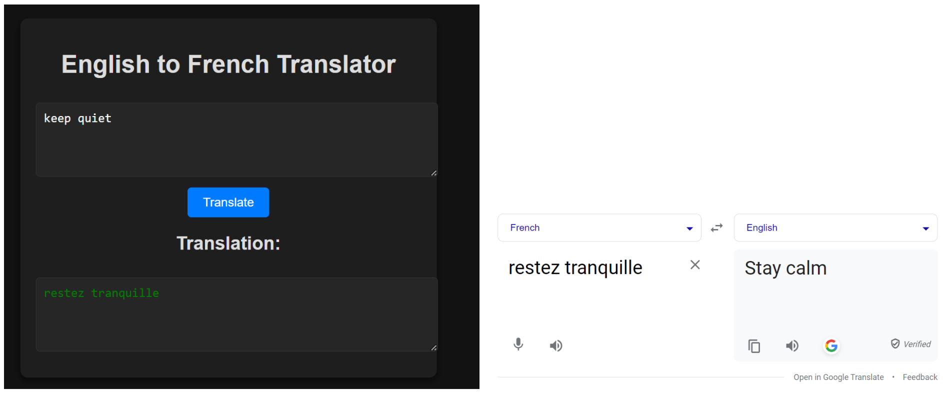
```
sit on the chair
```

```
assieds sur sur chaise
```

assieds sur sur chaise

sit on on chair

Figure 16: Demo 2


English to French Translator
Translate
Translation:
French
English
Verified
Open in Google Translate • Feedback

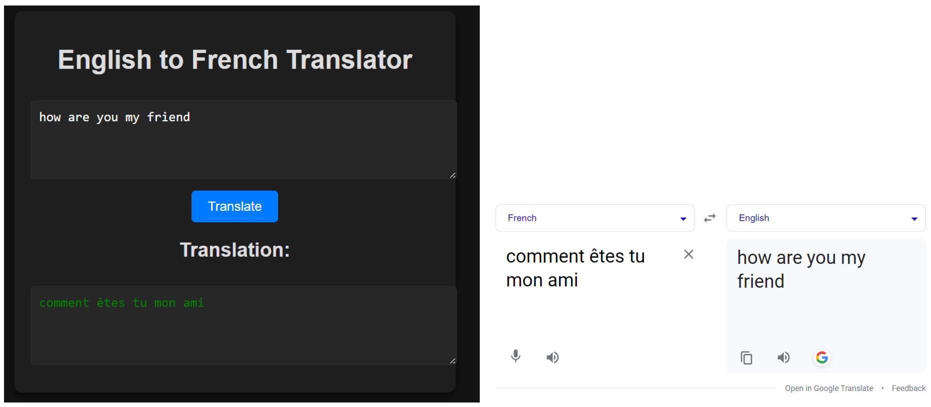
```
keep quiet
```

```
restez tranquille
```

restez tranquille

Stay calm

Figure 17: Demo 3


English to French Translator
Translate
Translation:
French
English
Open in Google Translate • Feedback

```
how are you my friend
```

```
comment êtes tu mon ami
```

comment êtes tu mon ami

how are you my friend

Figure 18: Demo 4

8 Conclusion

Different architectures were evaluated using various optimizers and loss functions. Sequence-to-Sequence (Seq2Seq) Model Utilizing Bidirectional LSTM and Attention Mechanism, Employing the Adam Optimizer and Cross-Entropy Loss Function.

8.1 Reasons for Variation

- **Optimizer**: Adam generally performed better.

- **Loss Function**: Cross-Entropy performed better.
- **Model Complexity**: Sequence-to-Sequence (Seq2Seq) Model Utilizing Bidirectional LSTM and Attention Mechanism

Contributions and GitHub

We collaborated in a Zoom meeting, jointly dedicating our efforts towards developing the code, compiling the report, and preparing the presentation for all the models.

Name	**Percentage**
Vikas Manchikanti	33.33%
Priyanka Chakraborty	33.33%
Bhanu Sahith Bonthula	33.33%

article

Click here to visit the GitHub Repository.

References

[1] DEVICHARITH, 020, Language Translation (English-French) , Kaggle dataset Available at: https://www.kaggle.com/datasets/devicharith/language-translation-englishfrench

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio,014, Neural Machine Translation by Jointly Learning to Align and Translate, Available at: https://arxiv.org/abs/1409.0473

[3] KSepp Hochreiter, Jürgen Schmidhuber, 2014, "Long Short-Term Memory " Available at: https://direct.mit.edu/neco/article-abstract/9/8/1735/6109/Long-Short-Term-Memory?redirectedFrom=fulltext

[4] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, 2014, "Sequence-to-Sequence Learning with Neural Networks" Available at: https://arxiv.org/abs/1409.3215

[5] Oriol Vinyals, Quoc Le 2015, "A Neural Conversational Model" Available at: https://arxiv.org/abs/1506.05869

[6] Kishore Papineni, Salim Roukos, Todd Ward, Wei-Jing Zhu 2002, "BLEU: a Method for Automatic Evaluation of Machine Translation" Available at: https://aclanthology.org/P02-1040/

[7] PyTorch Documentation - GRU updated periodically, "PyTorch Team" Available at: https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

[8] PyTorch NLP Tutorial: Sequence Models last updated on 2021, "PyTorch Team" Available at: https://pytorch.org/tutorials/beginner/nlp/sequence$_models_tutorial.html$

[9] Suraj Pattar 2021, "BERT for Natural Language Inference Simplified in PyTorch" Available at: https://www.analyticsvidhya.com/blog/2021/05/bert-for-natural-language-inference-simplified-in-pytorch/