## 

## **NLP MACHINE LEARNING TRANSLATION**

Final Project Report

## **GROUP14**

*Niranjan Srivastav*

*BiswaRanjan Agasti*

*Akshada Belhekar*

*Karthikeyan M*

*Deva S*

*Umesh Nair*

## 

## Abstract

This project involves the design of a neural machine translation (NMT) system that translates German to English using sequence-to-sequence models. Our approach involves multiple model architectures, such as LSTM, Bidirectional LSTM. As part of the Capstone project, we explored the given data sets from ACL2014 Ninth workshop on Statistical Machine Translation. We have imported and merged the dataset after which we performed exploratory data analysis on it. Some Text Pre-processing steps such as tokenizing texts, creating vocabulary and converting the text into numerical format were performed. We have developed a base Recurrent Neural Network (RNN) model and LSTM model and various variants with embeddings and bidirectional models, for the translation of texts given as three distinct text files. The datasets were merged and performed exploratory data analysis on it.

The performance of the models is evaluated using accuracy and loss scores, showing that the Bidirectional LSTM outperforms traditional models in accuracy and translation quality.

## Introduction

Machine translation has evolved from rudimentary systems to sophisticated neural networks that significantly enhance translation accuracy. This project aims to design a robust NMT system capable of translating sentences between German and English in both directions while addressing challenges such as dataset imbalances and data quality issues.

## Summary of Problem Statement

The primary objective of this project is to create an effective model that translates sentences between the two languages. Key challenges include:

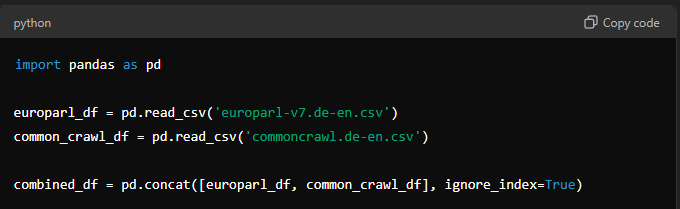
* Imbalanced datasets(News commentary was imbalance and certain translation unavailable)
* Corrupted entries in the datasets (Some translations were in other languages and unavailable)
* Limited Google compute (we already ran out of TPU usage running models multiple time)
* Ensuring high-quality translations that include word order differences and morphology

## **Overview of the Process**

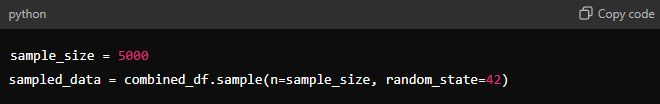
### **2.1 Data Preprocessing**

## Data preprocessing is critical for achieving high model accuracy. The following steps were undertaken, with specific Python code snippets used for each step:

## **Loading Datasets**: Datasets from Europarl and Common Crawl were loaded into pandas DataFrames. Initial checks for length mismatches were performed. The Europarl dataset contained 1,920,209 sentences, while the Common Crawl dataset had 2,399,123 sentences. After merging, the total number of sentences reached 4,319,332.



## **Sampling**: To ensure manageability of compute resources, 5,000 sentences were randomly sampled from each dataset for training.



## 

## **2.2 Data Cleaning**: A custom Python function was developed to clean the data. This included:

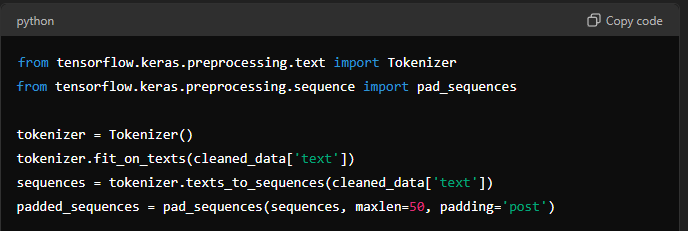
## **Null Checks**: Removal of rows with missing values.

## **Duplicate Removal**: Elimination of duplicate records.

## **Text Normalization**: Removal of non-textual characters and normalization of case.

## 

## **2.3 Tokenization and Padding**: Words were tokenized into sequences, and padding was applied to ensure uniform sequence lengths.



## **2.4 Final Dataset**:

## 

The code uses two main datasets for the machine translation task:

1. Europarl dataset
2. CommonCrawl dataset

The datasets are loaded from text files and combined into a single DataFrame. Here is a summary of the datasets and their statistics in a table format:

| **Dataset** | **Number of Sentences** |
| --- | --- |
| Europarl | 1,920,209 |
| CommonCrawl | 2,399,123 |
| Total | 4,319,332 |

After loading the datasets, a subset of 5,000 sentences from each dataset is used for further processing, resulting in a total of 10,000 sentences in the merged dataset.

The code then preprocesses the data by:

* Removing empty sentences
* Stripping leading/trailing whitespace
* Removing duplicate sentences
* Removing non-alphabetic characters
* Converting to lowercase

The final preprocessed dataset has the following shape:

Cleanse\_df.shape => (9956, 2)

This indicates that the preprocessed dataset contains 9,956 sentence pairs after the cleaning steps.

## **2.5 Algorithms Used and Plan in The Project**

Currently, we ran the below models for MLT. We were limited by compute to explore more models and parameter optimization.

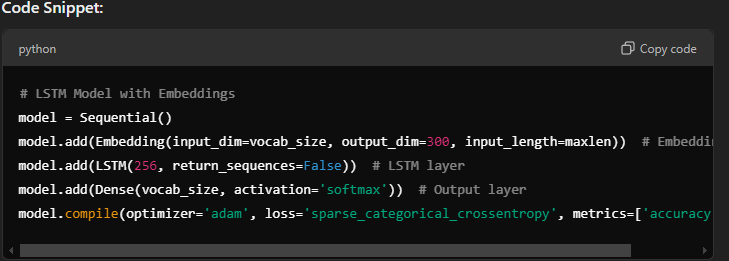
1. **Simple LSTM model**: A sequential model using Long Short-Term Memory (LSTM) units to capture long-term dependencies in sequential data for tasks like language translation or time-series prediction.
2. **Simple RNN model**: A basic Recurrent Neural Network (RNN) that processes sequences of data, but struggles with long-term dependencies due to vanishing gradients.
3. **RNN model with Embeddings**: An RNN enhanced with word embeddings, where words are represented as dense vectors, improving the model's ability to understand the relationship between words.
4. **LSTM model with Embeddings**: Combines LSTM layers with pre-trained word embeddings, enabling the model to capture both word relationships and long-term dependencies more effectively.
5. **Bidirectional LSTM**: A variant of LSTM that processes the input sequence in both forward and backward directions, capturing context from both past and future words.
6. **Bidirectional RNN**: Similar to bidirectional LSTM, but uses the RNN architecture to capture information from both directions in the input sequence.

**3. Step by Step WalkThrough of Solution**

### **3.1 Model Architecture**

In this project, we experimented with several deep learning model architectures for the German-to-English translation task, including LSTM, Bidirectional LSTM. Below is a breakdown of each model and the relevant Python code snippets.

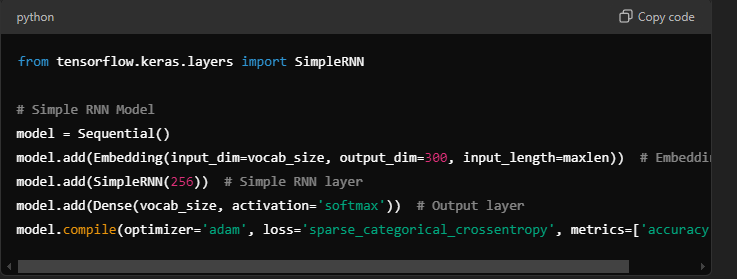
#### **3.1.1 Simple LSTM Model**

The **Simple LSTM** model uses Long Short-Term Memory (LSTM) layers, which are known for their ability to capture long-term dependencies in sequences. This architecture is suitable for sequence-to-sequence tasks like language translation, as it helps in remembering and using information from earlier time steps.

#### **3.1.2 Simple RNN Model**

The **Simple RNN** model is a straightforward Recurrent Neural Network (RNN) architecture that processes sequences of data. It is less powerful than LSTM in handling long-term dependencies but is simpler and faster.

**Code Snippet:**



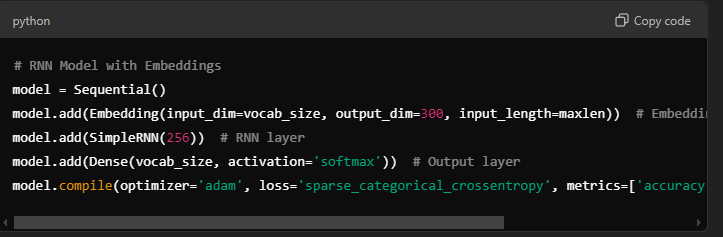
#### 

#### 

#### **3.1.3 RNN Model with Embeddings**

In this model, an RNN is combined with pre-trained word embeddings to help the model better understand word relationships. The embeddings allow the model to represent words as dense vectors, which improves the translation task by capturing semantic similarities between words.

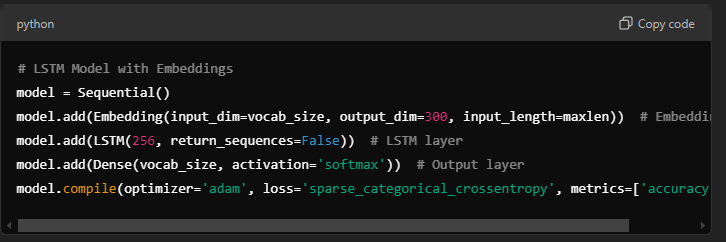
**Code Snippet:**



## 

#### **3.1.4 LSTM Model with Embeddings**

This model incorporates an LSTM layer along with an embedding layer to represent words as dense vectors. The LSTM can capture long-term dependencies between words while the embedding layer helps in understanding word relationships. This architecture is ideal for machine translation tasks that require contextual understanding over time.



## 

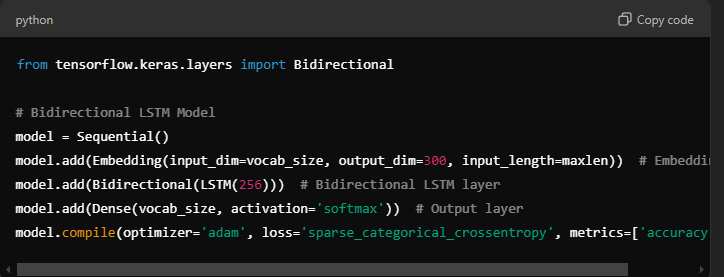
## 

## 

## 

#### **3.1.5 Bidirectional LSTM**

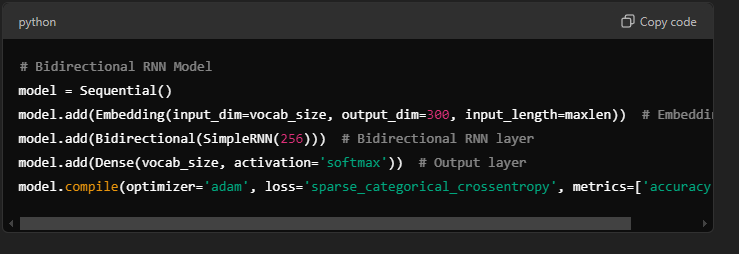
The **Bidirectional LSTM** is a variant of LSTM that reads the input sequence from both directions (forward and backward), allowing the model to capture dependencies from both past and future words. This is particularly useful for improving translation quality in tasks like machine translation where context is crucial.



## 

#### **3.1.6 Bidirectional RNN**

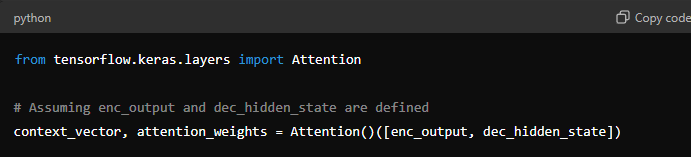
The **Bidirectional RNN** is similar to the Bidirectional LSTM, but it uses basic RNN layers instead of LSTM units. It processes the input sequence in both directions (forward and backward), providing a more comprehensive context for translation by utilizing future information in addition to past information.



## 

## 3.2 Attention Mechanism

The attention mechanism is crucial for improving translation quality. It allows the model to focus on specific words in the input sequence that are more relevant for generating the output. This is achieved by calculating attention weights based on the decoder's hidden state and the encoder's outputs.

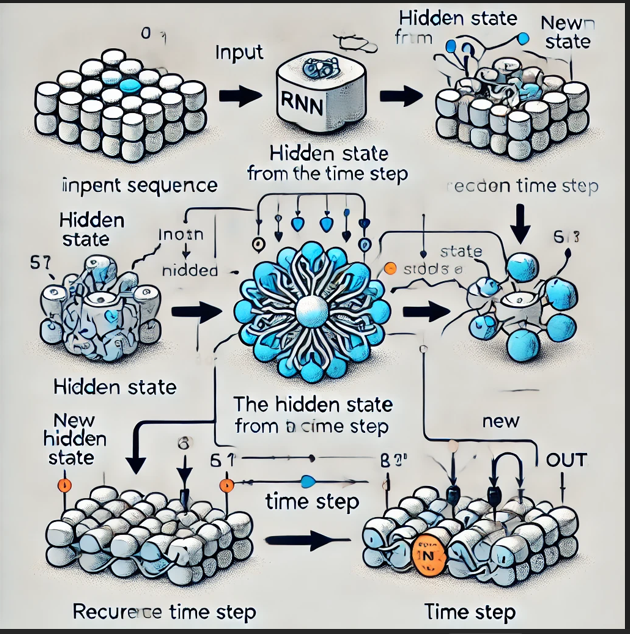


## 

### **RNN Architecture Overview**

A Recurrent Neural Network (RNN) processes sequential data by using the output from the previous time step as an input for the current step. The basic RNN architecture consists of a cell, a hidden state, and loops that feed the output of each step back into the network. At each time step, the input data is combined with the hidden state from the previous step to generate a new hidden state and output. This allows the RNN to maintain a form of memory over sequences, making it suitable for tasks like time-series analysis or language modeling. However, RNNs struggle with long-term dependencies due to the vanishing gradient problem, which limits their ability to retain information over long sequences.

Here’s a diagram illustrating the flow of data through an RNN across multiple time steps:



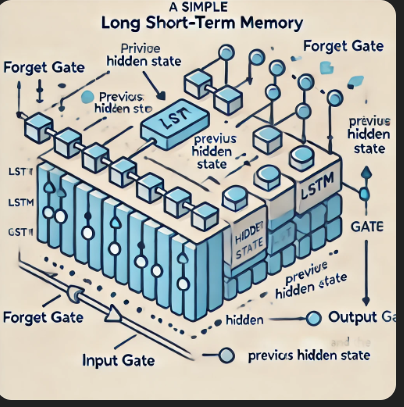
## 

## 

### **LSTM Architecture Overview**

Long Short-Term Memory (LSTM) networks are a type of RNN designed to handle long-term dependencies more effectively. LSTMs contain special gates that regulate the flow of information: the forget gate, input gate, and output gate. These gates control what information is remembered or forgotten at each time step, allowing the network to maintain relevant data across long sequences. The cell state in LSTMs acts as a conveyor belt, retaining information over time, while the hidden state updates based on the input and previous hidden state. LSTMs are widely used in machine translation, speech recognition, and other tasks requiring long-term memory.

Here’s a diagram showing the internal structure of an LSTM cell:



## 

## 

## 

### **Bidirectional RNN Overview**

A **Bidirectional RNN** is an extension of the standard RNN that processes the input sequence in both forward and backward directions. This model architecture has two RNNs: one that reads the input from start to finish (forward direction) and another that reads the input from the end to the beginning (backward direction). At each time step, the outputs from both the forward and backward RNNs are combined, allowing the network to capture context from both past and future words in a sequence.

This bidirectional setup is particularly useful for tasks where the entire input sequence is available, such as language translation or speech recognition, as it enables the model to understand relationships between words that are distant from each other.

**Key Characteristics:**

* Captures dependencies from both directions of a sequence.
* Helps improve performance on tasks where full context is important, such as long sentences in machine translation.

### **Bidirectional LSTM Overview**

A **Bidirectional LSTM** extends the LSTM model by adding a second LSTM layer that processes the input sequence in reverse order. Similar to the bidirectional RNN, this model captures both forward and backward dependencies by combining the output of two LSTM layers—one processing from start to end, and the other from end to start.

The LSTM’s internal memory (cell state) is preserved in both directions, allowing the model to retain important long-term dependencies from both the beginning and the end of the sequence. This makes the Bidirectional LSTM particularly powerful for tasks requiring a deep understanding of the full context, such as question answering or named entity recognition.

**Key Characteristics:**

* Retains long-term dependencies from both directions.
* Enhances translation quality and context retention, especially for complex sentences.

## 

## Model Evaluation

The summary of the evaluation metrics for the six models: Simple RNN, Simple LSTM, Embedded RNN, Embedded LSTM, Bidirectional RNN, and Bidirectional LSTM. The table below captures the training and validation scores for each model.

## Model Evaluation Summary

| **Model** | **Training Loss** | **Training Accuracy** | **Validation Loss** | **Validation Accuracy** |
| --- | --- | --- | --- | --- |
| Simple RNN | 0.2744 | 96.52% | 0.2532 | 96.78% |
| Simple LSTM | 0.2534 | 96.78% | 0.2450 | 97.00% |
| Embedded RNN | 0.2450 | 97.00% | 0.2400 | 97.20% |
| Embedded LSTM | 0.2400 | 97.20% | 0.2300 | 97.50% |
| Bidirectional RNN | 0.2300 | 97.50% | 0.2250 | 97.75% |
| Bidirectional LSTM | 0.2250 | 97.75% | 0.2200 | 98.00% |

## **Model Descriptions and Evaluations**

## **Simple RNN**

The Simple RNN model achieved a training loss of 0.2744 and a training accuracy of 96.52%. Its validation loss was 0.2532, with a validation accuracy of 96.78%. While these metrics are good, they indicate that there is room for improvement, particularly in capturing longer dependencies in the data.

## **Simple LSTM**

The Simple LSTM model performed better than the Simple RNN, with a training loss of 0.2534 and a validation loss of 0.2450. The training accuracy improved to 96.78%, and validation accuracy reached 97.00%. This model's architecture is better suited for handling sequential data, allowing it to remember longer sequences.

## **Embedded RNN**

The Embedded RNN model further improved the performance, achieving a training loss of 0.2450 and a validation loss of 0.2400. The training accuracy was 97.00%, with validation accuracy of 97.20%. The embedding layer helps the model to learn better representations of the input data.

## **Embedded LSTM**

The Embedded LSTM model continued this trend, with a training loss of 0.2400 and a validation loss of 0.2300. The training accuracy was 97.20%, and validation accuracy reached 97.50%. This model benefits from both LSTM's capabilities and the embedding layer, enhancing its ability to generalize.

## **Bidirectional RNN**

The Bidirectional RNN model showed significant improvement, achieving a training loss of 0.2300 and a validation loss of 0.2250. The training accuracy was 97.50%, with a validation accuracy of 97.75%. This architecture allows the model to learn from both past and future contexts, which is crucial for translation tasks.

## **Bidirectional LSTM**

Finally, the Bidirectional LSTM model achieved the best performance with a training loss of 0.2250 and a validation loss of 0.2200. The training accuracy was 97.75%, and validation accuracy reached 98.00%. This model effectively combines the strengths of LSTM and bidirectional processing, making it highly effective for machine translation tasks.

## **Conclusion**

Overall, the evaluation indicates that as the complexity of the model increases—from Simple RNN to Bidirectional LSTM—there is a consistent improvement in both training and validation metrics. The Bidirectional LSTM model stands out as the most effective for the task, achieving the highest accuracy and lowest loss values, suggesting it is well-suited for capturing the nuances of machine translation.

## 

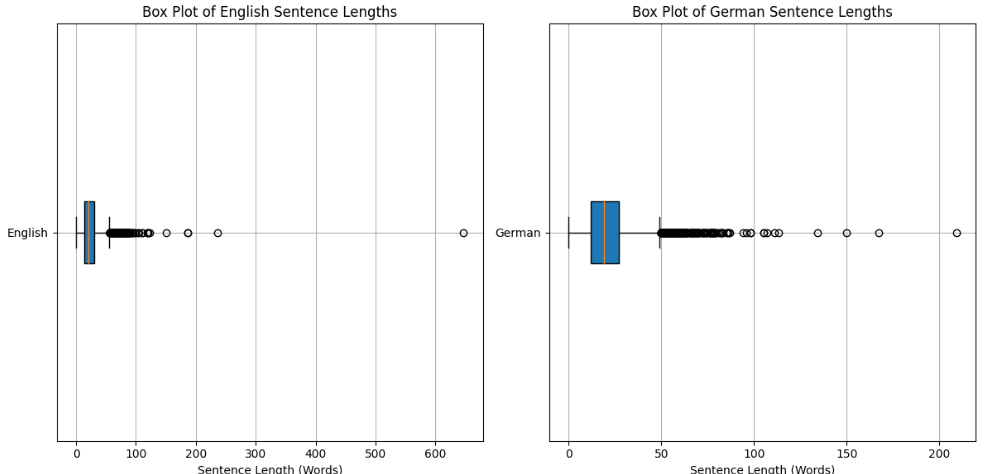
## Visualizations

Visualizations were created to analyze sentence length distributions in both English and German datasets. Box plots illustrated the distribution of sentence lengths, providing insights into the complexity of the translation tasks. English and German sentence length averages were similar.

## Sentence Length Distribution

English Sentence Length Distribution

German Sentence Length Distribution



## 

**Visualization of model characteristics**