Post Graduate Program *in*

Artificial Intelligence & Machine Learning

(Great Learning & UT Austin)

Capstone Project

Machine Translation

By:

NLP 2 Group 11

Bipin Kumar

Jeffry Jones

Rajat Agrawal

Rishi Barve

Soumit Kundu

Table of Contents

1. Introduction

Language is the primary means of human communication, relying on both grammar and vocabulary to convey meaning. In contrast, computers operate using Machine Language, which consists of numeric codes in the form of algorithms—strings of 0s and 1s known as bits—that enable them to execute operations directly. However, computers cannot understand human language in its natural form.

To bridge this gap between humans and machines, Natural Language Processing (NLP) was developed. NLP allows computers to process and interpret human speech, making it possible to analyze and respond to what users say. By combining computational linguistics with Artificial Intelligence (AI), NLP enables effective communication between humans and computers. This technology now powers everyday applications like Alexa and Siri.

For a computer program to engage in meaningful communication with humans, it must understand grammatical syntax, word meanings (semantics), correct tense usage (morphology), and conversational context (pragmatics). The complexity of these requirements has historically posed significant challenges for NLP, leading to the failure of earlier approaches. However, modern NLP has shifted from rule-based methods to pattern-learning-based programming, resulting in more effective language processing.

In this project, we explore the development of a Machine Translation model designed to translate text between languages. Specifically, we have focused on translating sentences between German and English. The goal of this project is to facilitate communication and idea exchange between people from different countries, breaking down language barriers and fostering global collaboration.

2. Problem Statement

The task is to design a Sequence-to-Sequence model for Machine Translation, that can be used to translate sentences from German to English language

The dataset comes from ACL2014 Ninth workshop on Statistical Machine Translation. This workshop mainly focused on language translation between European language pairs. The idea behind the workshop was to provide the ability for two parties to communicate and exchange the ideas from different countries.

Three datasets are used here –

- Common Crawl Dataset: General collection of parallel German-English sentences.
- Europarl Dataset: Collection of text from the European Parliament proceedings.
- News Commentary Dataset: Collection of text from news articles.

We will use sequence-to-sequence models like RNN and LSTM models, build different variants of these models, to find the algorithm best suited for the machine translation task.

3. Import and Read Files

We have 3 sets of files in English and German each.

- Common Crawl and contain about 2 million sentences.

```
Length of Common Crawl English Sentences File: 2399123
Length of Common Crawl German Sentences File: 2399123

English Sentence: iron cement is a ready for use paste which is laid as a fillet by putty

German Sentence: iron cement ist eine gebrauchs-fertige Paste, die mit einem Spachtel oder


Length of Europarl English Sentences File: 1920209
Length of Europarl German Sentences File: 1920209

English Sentence: Resumption of the session

German Sentence: Wiederaufnahme der Sitzungsperiode
```

- News Commentary file contain about 200 K sentences

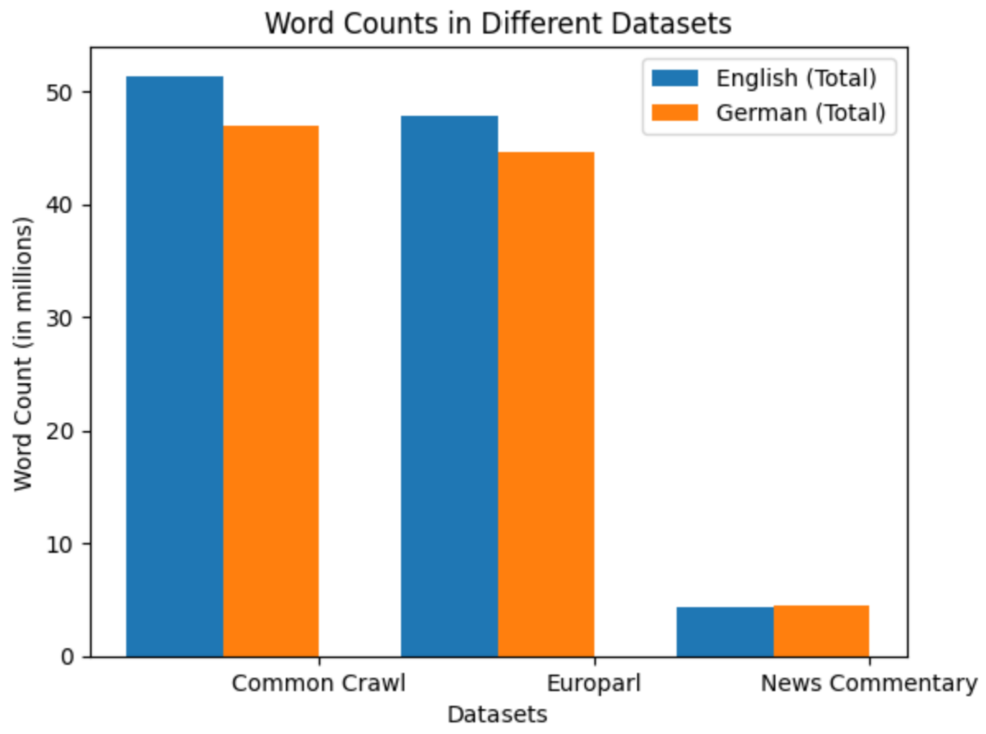
```
Length of News Commentary English Sentences File: 201995
Length of News Commentary German Sentences File: 201854

English Sentence: $10,000 Gold?

German Sentence: Steigt Gold auf 10.000 Dollar?
```

News Commentary English File has 141 more sentences than the German File

4. Exploratory Data Analysis

Check Vocabulary Size

Total Word Count



Observations

- Both Common Crawl and Europarl have a huge total word count in English and German (~50 million), which is expected considering the no of sentences in both.
- News Commentary has a much smaller total word count of ~5 million
- The total word count for English is slightly higher than German for both Common Crawl and Europarl, but almost equal across for News Commentary

Unique Word Count

```
Common Crawl:
English (Unique)- 1532712
German (Unique)- 2557806

Europarl:
English (Unique)- 271951
German (Unique)- 616947

News Commentary:
English (Unique)- 139001
German (Unique)- 139001
```

Observations

- Even though Common Crawl and Europarl have similar file and vocab size, the unique word count differs considerably, with Common Crawl having unique count of 1.5 – 2.5 million , while Europarl having unique count in 200-600 K.
- In both of these, Unique word count in German is much higher than that in English.
- Interestingly, in News Commentary, both English and German have same unique word count

Check and Remove Blank Lines

In the dataset, we see blank lines as well. It seems that in most cases, 2 lines seem to have been combined in one line, leaving the next line as blank.

To handle these blank lines, we

- Check for the blank lines in each dataset's German and English Language files and store the corresponding indices
- Combine indices of blank lines in both German and English Files
- Remove lines with these combined indices from both German and English files, so that same lines are removed from each file and rest of the lines remain aligned

Function to check for blank lines

```
# Define function to check for blank lines in given file

def check_blank(list_name):
    blank_lines_index = []
    for i in range(len(list_name)):
        if list_name[i] == '\n':
            blank_lines_index.append(i)

    return blank_lines_index
```

Results

```
No of blank lines in Common Crawl English File: 0
No of blank lines in Common Crawl German File: 0
No of blank lines in Europarl English File: 8366
No of blank lines in Europarl German File: 2923
No of blank lines in News Commentary English File: 322
No of blank lines in News Commentary German File: 224
```

```
Europarl English File Length, after removing blank lines: 1908920
Europarl German File Length: 1908920

News Commentary English File Length, after removing blank lines: 201449
News Commentary German File Length,after removing blank lines: 201309
```

There are no blanks in Common Crawl files.

After removing blank lines from News Commentary, we still have diff of 140 lines between English and German files

5. Merging Datasets

Because of the huge size of datasets, it is not feasible to use entire datasets for training and testing, due to amount of computing resource and training time which will be required.

So, for training, validation and testing, 10000 sentences from each of the 3 datasets are taken and merged to form common English and German datasets

```python
# Merge the three lists
en_sentences = common_crawl_en[:10000] + europarl_en[:10000] + news_commentary_en[:10000]
de_sentences = common_crawl_de[:10000] + europarl_de[:10000] + news_commentary_de[:10000]

# Print the lengths of the merged lists
print('Length of Merged English File:', len(en_sentences))
print('Length of Merged German File:', len(de_sentences))
```

```
Length of Merged English File: 30000
Length of Merged German File: 30000
```

For ease of viewing, data cleaning and visualization, we create a dataframe

```python
# Create a DataFrame from the merged lists
df = pd.DataFrame({'english': en_sentences, 'german': de_sentences})

df.head()
```

	english	german
0	iron cement is a ready for use paste which is ...	iron cement ist eine gebrauchs-fertige Paste, ...
1	iron cement protects the ingot against the hot...	Nach der Aushärtung schützt iron cement die Ko...
2	a fire restant repair cement for fire places, ...	feuerfester Reparaturkitt für Feuerungsanlagen...
3	Construction and repair of highways and...\n	Der Bau und die Reparatur der Autostraßen...\n
4	An announcement must be commercial character.\n	die Mitteilungen sollen den geschäftlichen kom...

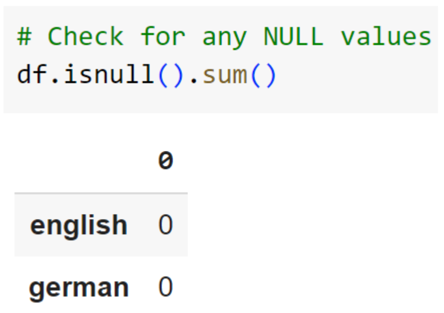
6. Data Cleaning and Pre-Processing

Previously, we have already checked and removed all blank lines from each of the files, before selecting the sample data from each file and merging them.

As part of data cleaning, we will now check for any NULL values, remove punctuation marks and covert the text to lowercase

Check for NULL values

As blank lines were removed, no NULL values are remaining

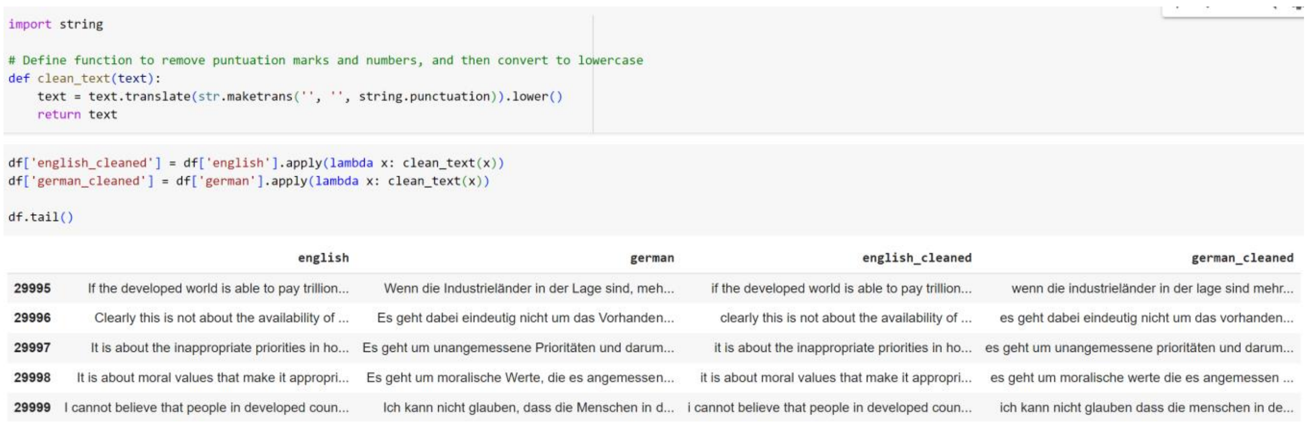


```
# Check for any NULL values
df.isnull().sum()
```

	0
english	0
german	0

Remove Punctuation Marks and Convert to Lowercase

We define a function to perform required operations on English and German Sentence columns in dataframe and add cleaned columns



```
import string

# Define function to remove puntuation marks and numbers, and then convert to lowercase
def clean_text(text):
    text = text.translate(str.maketrans('', '', string.punctuation)).lower()
    return text
```

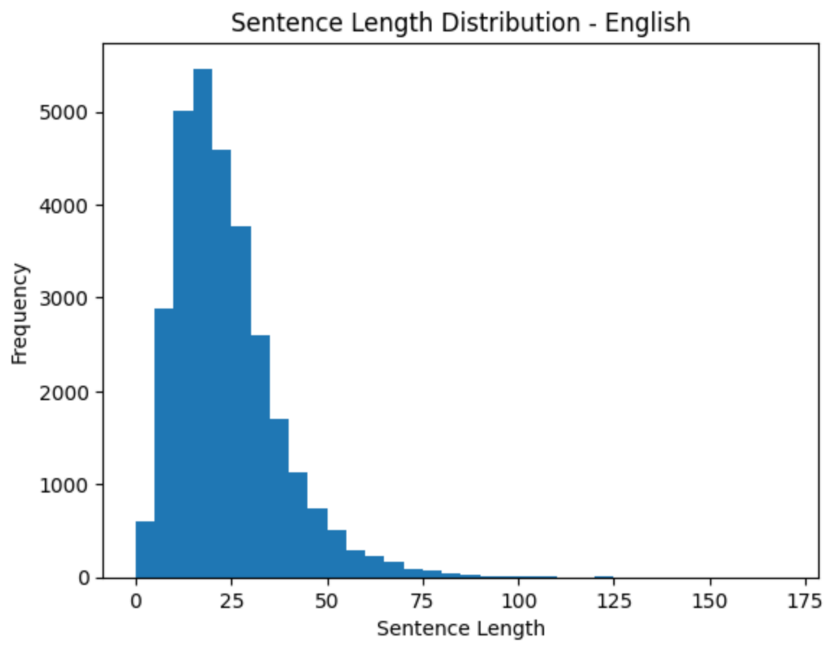
```
df['english_cleaned'] = df['english'].apply(lambda x: clean_text(x))
df['german_cleaned'] = df['german'].apply(lambda x: clean_text(x))

df.tail()
```

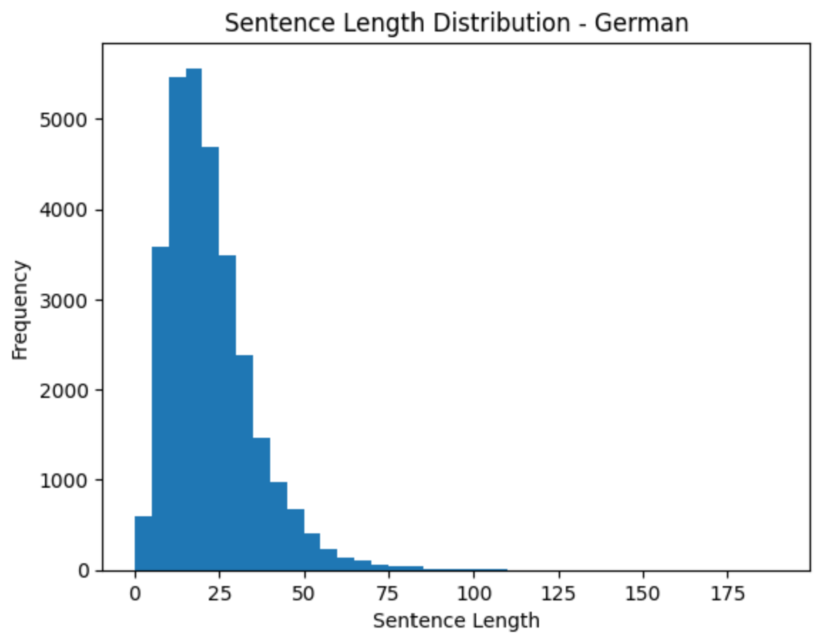
	english	german	english_cleaned	german_cleaned
29995	If the developed world is able to pay trillion...	Wenn die Industrieländer in der Lage sind, meh...	if the developed world is able to pay trillion...	wenn die industrieländer in der lage sind mehr...
29996	Clearly this is not about the availability of ...	Es geht dabei eindeutig nicht um das Vorhanden...	clearly this is not about the availability of ...	es geht dabei eindeutig nicht um das vorhanden...
29997	It is about the inappropriate priorities in ho...	Es geht um unangemessene Prioritäten und darum...	it is about the inappropriate priorities in ho...	es geht um unangemessene prioritäten und darum...
29998	It is about moral values that make it appropri...	Es geht um moralische Werte, die es angemessen...	it is about moral values that make it appropri...	es geht um moralische werte die es angemessen ...
29999	I cannot believe that people in developed coun...	Ich kann nicht glauben, dass die Menschen in d...	i cannot believe that people in developed coun...	ich kann nicht glauben dass die menschen in de...

We will now use these cleaned columns for further processing

Check Sentence Length Distribution and Max Sentence Length

We now check the sentence length distribution of cleaned English and German Sentences





Observations

- Distribution of English and German sentence lengths are similar most sentences made up of 20-25 words.
- There are a few outliers with very long sentences, in both English and German datasets, resulting in a long right skew.

- Long right skew indicates that there are more shorter sentences than longer ones.
- Both have maximum sentence lengths of > 100

Checking maximum length of cleaned English and German sentences in sample data

```
Maximum English Sentence length: 172
Maximum German Sentence length: 193
```

Tokenization and Padding

Tokenization is a process to assign a unique id to each word in the dataset vocabulary, which is then used to convert each sentence to a sequence of word IDs. This is done as we need to convert words to numerical values before using as input for deep learning model.

Also, when we feed these sequences of word IDs into the model, each sequence needs to be the same length. Since sentences are dynamic in length, we add padding to the end of the sequences to make them the same length.

Then we create a function to perform tokenization and padding, in which

- All the words of the sentence are tokenized. As we have sentences in 2 languages, two tokenizers are defined - an English Tokenizer and a German tokenizer.
- Then based on the calculated maximum length of sentences in sample data, we perform padding for English and German sequences.

```
# Define function to perform tokenization
def tokenization(sentences, max_length):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(sentences)
    sequences = tokenizer.texts_to_sequences(sentences)
    padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post')

    return tokenizer, padded_sequences
```

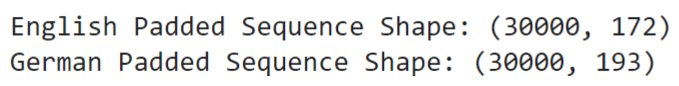
The function returns the tokenizer and padded sequences.

Using tokenizer we can find the sampled English and German vocabulary size

```
English Vocabulary size: 34856
German Vocabulary size: 59642
```

Even with just 10000 sentences, we get a huge vocabulary for both English and German.

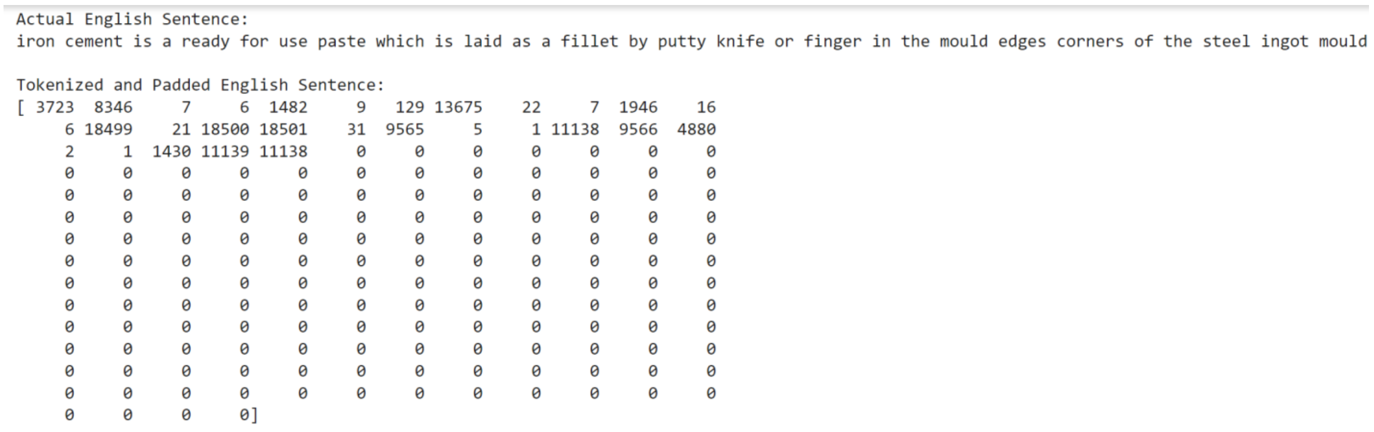
We also check the Padded sequence size



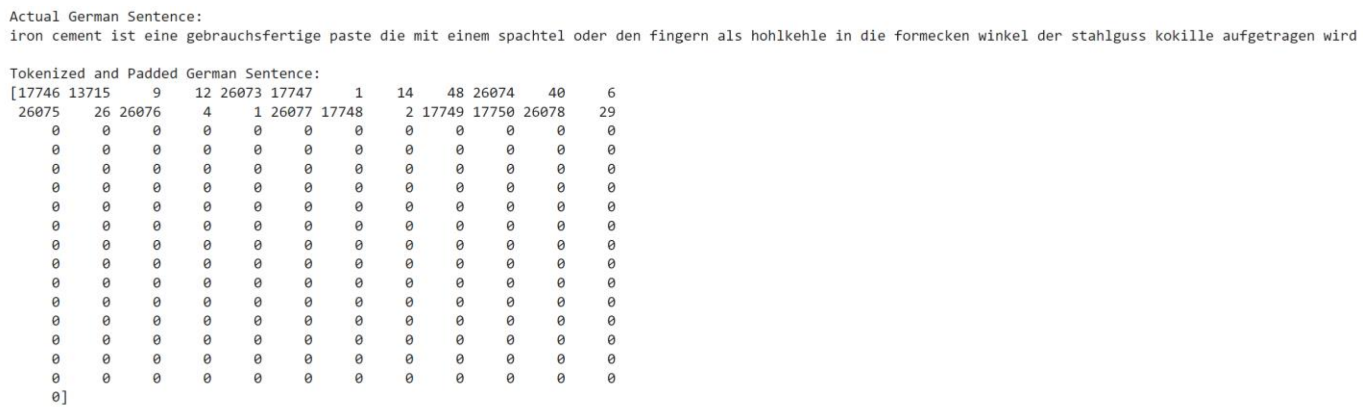
```
English Padded Sequence Shape: (30000, 172)
German Padded Sequence Shape: (30000, 193)
```

Both English and German sentence lengths are standardized.

Sample English and German sentences and their respective padded sequences



```
Actual English Sentence:
iron cement is a ready for use paste which is laid as a fillet by putty knife or finger in the mould edges corners of the steel ingot mould

Tokenized and Padded English Sentence:
[ 3723  8346     7     6  1482     9   129 13675    22     7  1946    16
     6 18499    21 18500 18501    31  9565     5     1 11138  9566  4880
     2     1  1430 11139 11138     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0]
```



```
Actual German Sentence:
iron cement ist eine gebrauchsfertige paste die mit einem spachtel oder den fingern als hohlkehle in die formecken winkel der stahlguss kokille aufgetragen wird

Tokenized and Padded German Sentence:
[17746 13715     9    12 26073 17747     1    14    48 26074    40     6
 26075    26 26076     4     1 26077 17748     2 17749 17750 26078    29
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0]
```

7. Model Selection and Architecture

Model Selection

Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) models are widely used in language translation tasks due to their ability to handle sequential data. Here are the key advantages of using these models:

- **RNN Model**: RNNs are designed to process sequences of data, making them well-suited for tasks like language translation where the order of words matters. They can capture temporal dependencies by maintaining a hidden state that evolves as the input sequence is processed.

  RNNs can retain information about previous words in a sequence, which helps in understanding the context necessary for translating sentences correctly.
- **LSTM Model**: LSTMs, a specialized form of RNNs, are particularly effective at handling long sequences. They address the vanishing gradient problem, which RNNs often face when dealing with long-term dependencies. Selectively remembering and forgetting information through their gating mechanisms (input, forget, and output gates), enables them to maintain context over longer sequences, improving the translation of complex sentences.

We will build, train and test below Models

- Basic RNN and LSTM Models without Embeddings
- RNN and LSTM Model with Embeddings
- Bi-Directional RNN and LSTM Models
- Encoder-Decoder RNN & LSTM Models

Model Architecture

We will build a Sequential Model with each Model Architecture having one of more layers from below

- Input Layer: which takes the English Sequences as input
- Reshape Layer: Reshapes the input sequence to be compatible with given model
- Model Layer: RNN or LSTM model. We will keep the no of units small to ensure that the model can work on the limited computing power.
- Dropout Layer: Dropout of X% to make model more robust
- Time Distributed Layer: Ensures that the model can predict the entire sequence of words in the target language, with each time step being independently processed but contributing to the overall sequence prediction.
- Dense Layer: Neural Network layer with the number of neurons corresponding to the size of German vocabulary. This allows the model to predict probabilities for each German word.

- Activation Layer: Softmax Activation is to ensure the output of the dense layer is a probability distribution, where each value represents the likelihood of a specific German word being the correct translation.

Model compilation

- Optimizer: Adam optimizer with a learning rate and clipvalue.

  Adam is a popular optimization algorithm for training neural networks. It adapts the learning rate for each parameter during training.

  A smaller learning rate leads to slower but potentially more stable convergence.

  Clip Value limits the magnitude of the gradients to prevent exploding gradients, which can cause instability during training
- Loss Function: sparse_categorical_crossentropy is used as we are dealing with a multi-class classification problem where each word in the target sentence is a class, and the labels are integers.
- Metrics: accuracy is a common metric to evaluate the performance of classification models. It measures the percentage of correctly predicted words.

8. Model Build, Training and Evaluation

Train-Test Split

We split the padded sequences into Train and Test data. 10% of sample is taken for testing

```
Shape of X_train: (27000, 172)
Shape of y_train: (27000, 193)
Shape of X_test: (3000, 172)
Shape of y_test: (3000, 193)
```

English sequences is the input, so X_train has length of max English sentence length from sample

German sequences is the output, so y_train has length of max German sentence length from sample

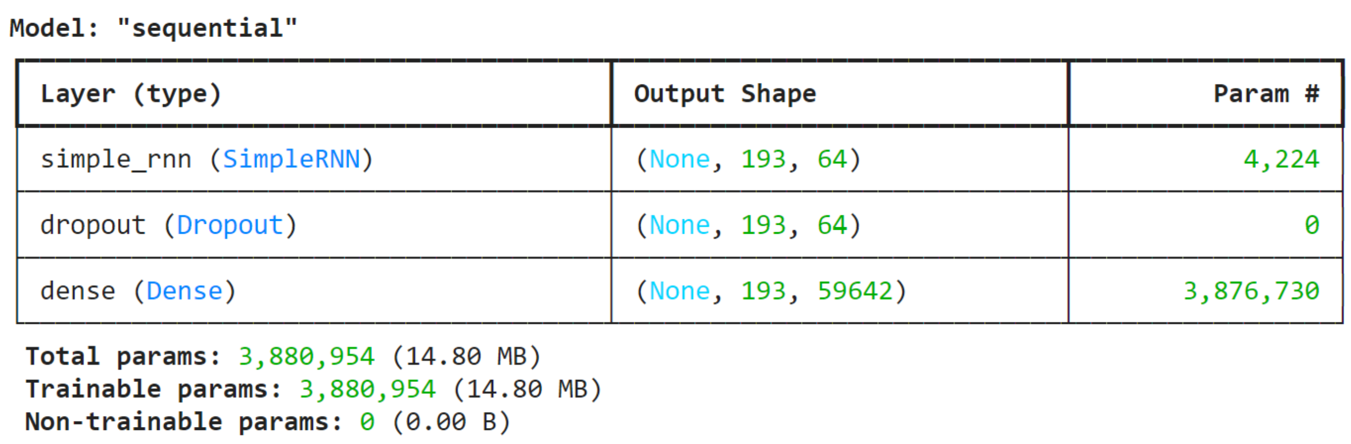
Simple RNN Model (without Embeddings)

First, we build a Base RNN Model without Embeddings, with

- Input Layer
- SimpleRNN layer with 64 units to process the input sequence.
- Dropout Layer with 20% dropout
- Dense Layer

We also reshape the input and output to add a 3rd dimension to make it compatible with RNN

Model Summary

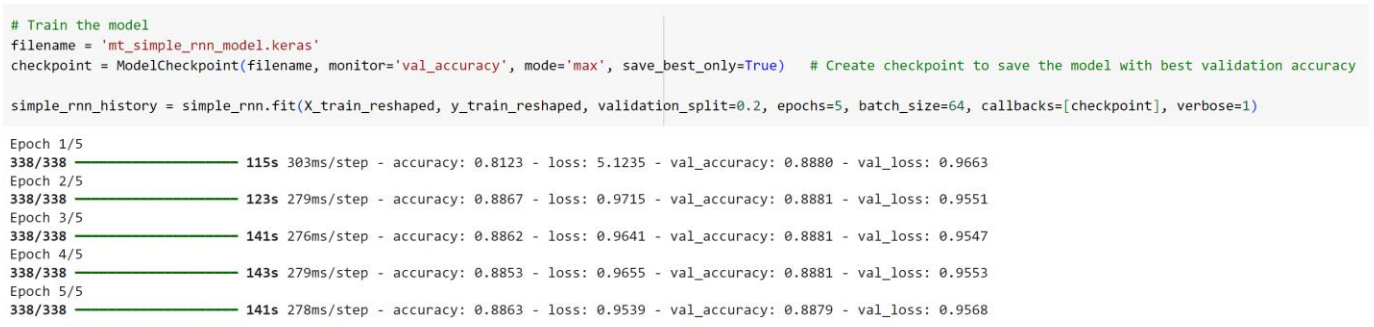


```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ simple_rnn (SimpleRNN)               │ (None, 193, 64)             │           4,224 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 193, 64)             │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 193, 59642)          │       3,876,730 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 3,880,954 (14.80 MB)
 Trainable params: 3,880,954 (14.80 MB)
 Non-trainable params: 0 (0.00 B)
```

Model Training

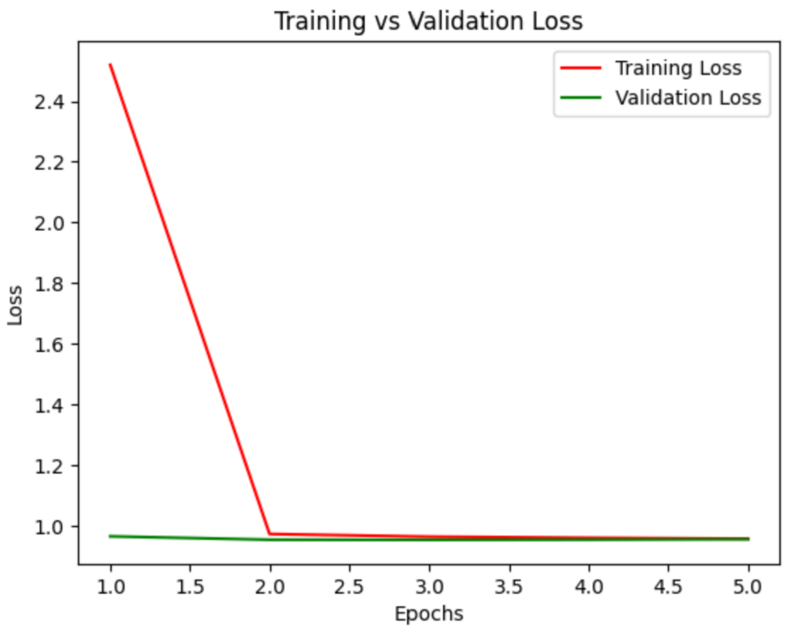
Model training is done with validation split of 20%, 5 epochs and a batch size of 64.

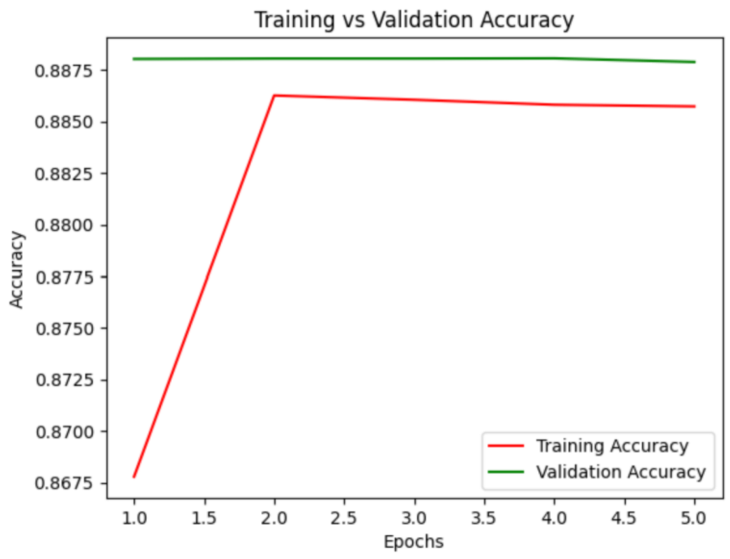
We also create a checkpoint to save the model with max validation accuracy



```
# Train the model
filename = 'mt_simple_rnn_model.keras'
checkpoint = ModelCheckpoint(filename, monitor='val_accuracy', mode='max', save_best_only=True)   # Create checkpoint to save the model with best validation accuracy

simple_rnn_history = simple_rnn.fit(X_train_reshaped, y_train_reshaped, validation_split=0.2, epochs=5, batch_size=64, callbacks=[checkpoint], verbose=1)
```

```
Epoch 1/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 115s 303ms/step - accuracy: 0.8123 - loss: 5.1235 - val_accuracy: 0.8880 - val_loss: 0.9663
Epoch 2/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 123s 279ms/step - accuracy: 0.8867 - loss: 0.9715 - val_accuracy: 0.8881 - val_loss: 0.9551
Epoch 3/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 141s 276ms/step - accuracy: 0.8862 - loss: 0.9641 - val_accuracy: 0.8881 - val_loss: 0.9547
Epoch 4/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 143s 279ms/step - accuracy: 0.8853 - loss: 0.9655 - val_accuracy: 0.8881 - val_loss: 0.9553
Epoch 5/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 141s 278ms/step - accuracy: 0.8863 - loss: 0.9539 - val_accuracy: 0.8879 - val_loss: 0.9568
```





Observations:

- In first Epoch, when weights are randomized, the loss is high and accuracy is lower.
- From 2nd Epoch onwards, loss comes down to ~0.97 and then remains pretty much same
- Same is seen for accuracy as well, from 2nd epoch onwards, accuracy remains at ~88%
- This RNN model shows stable performance on validation data throughout the epochs, as evident from straight line
- Maximum Training Accuracy is 88.67%, and Maximum Validaiton Accuracy is 88.81%
- Since the training and validation loss and accuracy are pretty close, which means model is not overfitting.

Model Evaluation

We can now evaluate the model on Test Data. To avoid crashing, we only use first 100 records from Test data for evaluation.

```
# Evaluate the model on 100 samples from Test data
loss, accuracy = simple_rnn_model.evaluate(X_test_reshaped[:100], y_test_reshaped[:100], verbose=0)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
```

```
Test Loss: 0.9517783522605896
Test Accuracy: 0.8900517821311951
```

Even with a Simple RNN model without Embedding, we have a very good accuracy of 89%

Simple LSTM Model (without Embeddings)

Now, we build a Base LSTM Model without Embeddings, with

- Input Layer
- LSTM layer with 64 units to process the input sequence.
- Dropout Layer with 20% dropout
- Dense Layer

Here also, we use the same reshaped input and output training data

Model Summary

```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ lstm (LSTM)                     │ (None, 193, 64)        │        16,896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 193, 64)        │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 193, 59642)     │     3,876,730 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 3,893,626 (14.85 MB)
 Trainable params: 3,893,626 (14.85 MB)
 Non-trainable params: 0 (0.00 B)
```

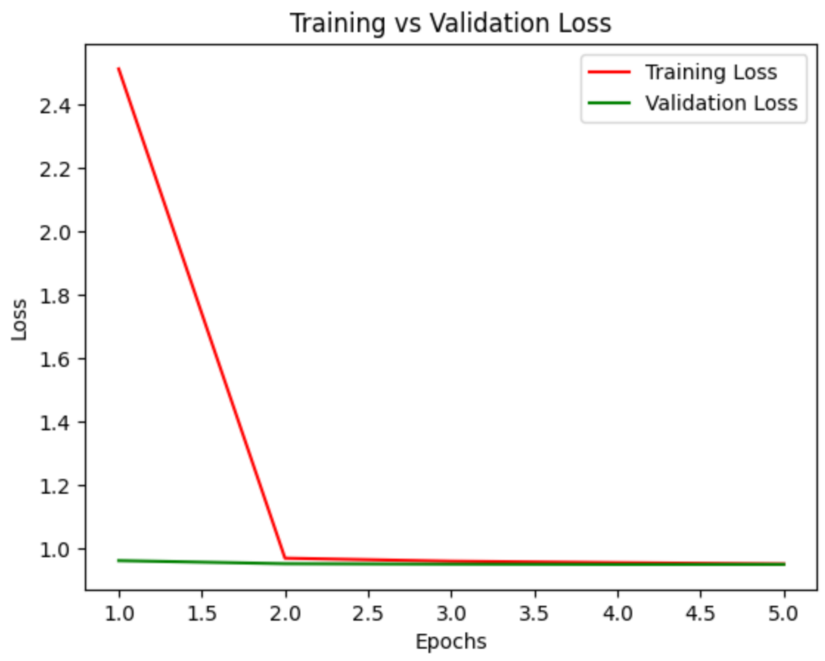
Model Training

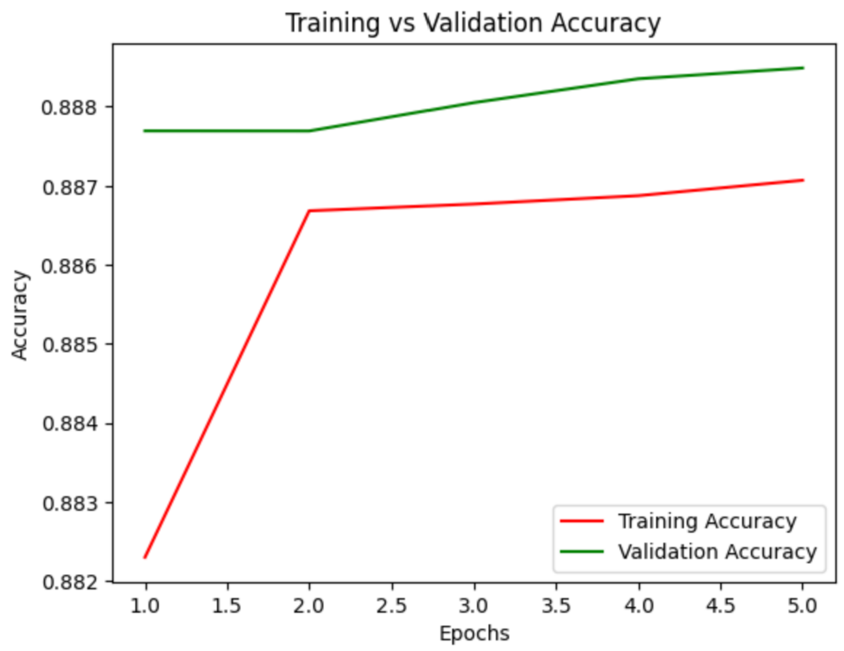
Model training is done with validation split of 20%, 5 epochs and a batch size of 64.

We also create a checkpoint to save the model with max validation accuracy

```
# Train the model
filename = 'mt_simple_lstm_model.keras'
checkpoint = ModelCheckpoint(filename, monitor='val_accuracy', mode='max', save_best_only=True)   # Create checkpoint to save the model with best validation accuracy

simple_lstm_history = simple_lstm.fit(X_train_reshaped, y_train_reshaped, validation_split=0.2, epochs=5, batch_size=64, callbacks=[checkpoint], verbose=1)
```

```
Epoch 1/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 203s 586ms/step - accuracy: 0.8665 - loss: 5.1829 - val_accuracy: 0.8877 - val_loss: 0.9630
Epoch 2/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 198s 584ms/step - accuracy: 0.8873 - loss: 0.9686 - val_accuracy: 0.8877 - val_loss: 0.9534
Epoch 3/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 203s 586ms/step - accuracy: 0.8864 - loss: 0.9620 - val_accuracy: 0.8881 - val_loss: 0.9520
Epoch 4/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 214s 622ms/step - accuracy: 0.8866 - loss: 0.9571 - val_accuracy: 0.8884 - val_loss: 0.9511
Epoch 5/5
338/338 ━━━━━━━━━━━━━━━━━━━━ 249s 585ms/step - accuracy: 0.8874 - loss: 0.9491 - val_accuracy: 0.8885 - val_loss: 0.9514
```





Observations:

- Both Training and Validation, Loss and Accuracy curves are very similar to what we saw earlier for RNN model
- Training and Validation Loss curves are almost same as that of RNN Model, but some difference is seen for Training vs Validation Accuracy.
- For Training Accuracy, in RNN we it reaching its high in 2nd epoch and then slighltly decreasing towars the end, while in LSTM it is slightly increases from 86.65% to 88.74%
- For Validation Accuracy also, in RNN we see a straight line and then a slight decrease towards the end, while in LSTM it increase slightly from 88.77% to 88.85%
- Maximum Training Accuracy is 88.74%, and Maximum Validaiton Accuracy is 88.85%.
- Since the training and validation loss and accuracy are pretty close, which means model is not overfitting.

Model Evaluation

We can now evaluate the model on Test Data. To avoid crashing, we only use first 100 records from Test data for evaluation.

```
# Evaluate the model on 100 samples from Test data
loss, accuracy = simple_lstm_model.evaluate(X_test_reshaped[:100], y_test_reshaped[:100], verbose=0)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
```

```
Test Loss: 0.8728080987930298
Test Accuracy: 0.9013472199440002
```

Simple LSTM model without Embeddings, results in a slightly higher accuracy of ~90%

9. Improving Model Performance

Here are some approaches to enhance the performance of the RNN and LSTM models:

Model Architecture Enhancements:

- Add Embeddings: Use pre-trained word embeddings (e.g., GloVe, Word2Vec) to capture semantic relationships between words. Embeddings can significantly improve model performance by providing richer word representations.
- Use Bidirectional Layers: Bidirectional RNNs and LSTMs process the sequence in both forward and backward directions, capturing context from both ends of the sequence.
- Use Encoder-Decoder Architecture: For machine translation tasks, encoder-decoder architectures with attention mechanisms can capture complex dependencies between input and output sequences by enabling the model to focus on relevant parts of the input sequence when generating each word in the output. This can improve translation accuracy.

Hyperparameter Tuning:

- Model Complexity: Tune the number of layers and units in each layer to balance model capacity and avoid overfitting.
- Adjust Learning Rate: Experiment with different learning rates and optimizers. A learning rate scheduler can help in finding the optimal rate.
- Batch Size and Epochs: Modify the batch size and number of epochs to find the best training setup. Sometimes, increasing batch size or adding more epochs can improve performance.

Regularization Techniques:

- Dropout: Experiment with dropout percentage to prevent overfitting by randomly dropping neurons during training.
- L2 Regularization: Apply L2 regularization to the weights to reduce overfitting by penalizing large weights.

Advanced Techniques:

Explore Pre-trained Models: Utilize pre-trained models and fine-tune them for the specific task. Transfer learning can leverage the knowledge from large-scale datasets.

In Summary, improving model performance involves a combination of better data cleaning and pre-processing, more sophisticated and complex model architectures, and careful tuning of hyperparameters. By implementing these strategies, we can enhance the accuracy and robustness of any sequence-to-sequence models.

10. Conclusion

For Milestone 1, we have focussed on setting up the foundational aspects of the project, including data acquisition, cleaning, preprocessing, and the initial base model development using RNN and LSTM architectures.

A Basic RNN model was implemented as a baseline to understand the sequential nature of the data. The model was trained and tested on the preprocessed data, with performance metrics recorded to establish a benchmark.

Following the RNN model, an LSTM model was also implemented to address the vanishing gradient problem often encountered with RNNs. The LSTM model's performance was evaluated and compared with the RNN model.

Conclusion: Both models show reasonable performance, with the LSTM model slightly outperforming the RNN model, indicating better performance in capturing complex patterns and long-range dependencies in the sequences.

Moving forward, **Milestone 2** will focus on experimenting with advanced variants of both RNN and LSTM architectures. The goal will be to refine the translation accuracy and address any remaining challenges in the translation task