Neural-Machine-Translation

Project Overview

In this project, we are focusing on text language translation. The input is a sentence in German, and the output is an equivalent English sentence.

Data-Visualization

```
array([['Go.', 'Geh.'],
       ['Hi.', 'Hallo!'],
       ['Hi.', 'Grüß Gott!'],
       ...,
       ['I heard you did well.',
        'Ich habe gehört, Sie haben gut abgeschnitten.'],
       ['I heard you laughing.', 'Ich habe dich lachen gehört.'],
       ['I heard you laughing.', 'Ich habe euch lachen gehört.']],
      dtype='<U537')
```

1. We have a dictionary of pairs of strings where the first element is in English and the second in German.
2. As part of basic preprocessing, we drop any rows that are duplicates or contain null values.

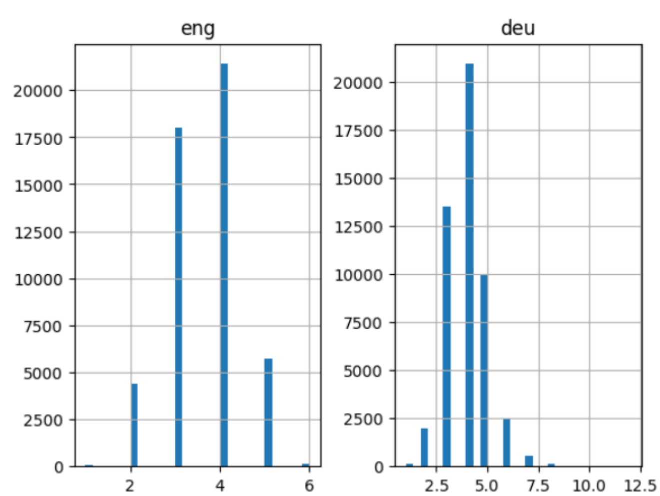
Pre-Processing

As part of the preprocessing steps, I have performed the following transformations:

1. Converted all text to lowercase and removed punctuations.

Feature-Engineering

1. Now, as the input length of the model can vary, but all inputs in a given batch must be of the same length, choosing an appropriate `max_length` based on the typical length distribution of your dataset helps retain the most relevant information.



```
English - Max Length: 6, Min Length: 1, Most Frequent Length: (4, 21394)
German - Max Length: 12, Min Length: 1, Most Frequent Length: (4, 20950)
```

2. We try to pad smaller sentences and truncate the longer ones. Hence, I choose an optimal `max_length` of 8 for English and 8 for German.

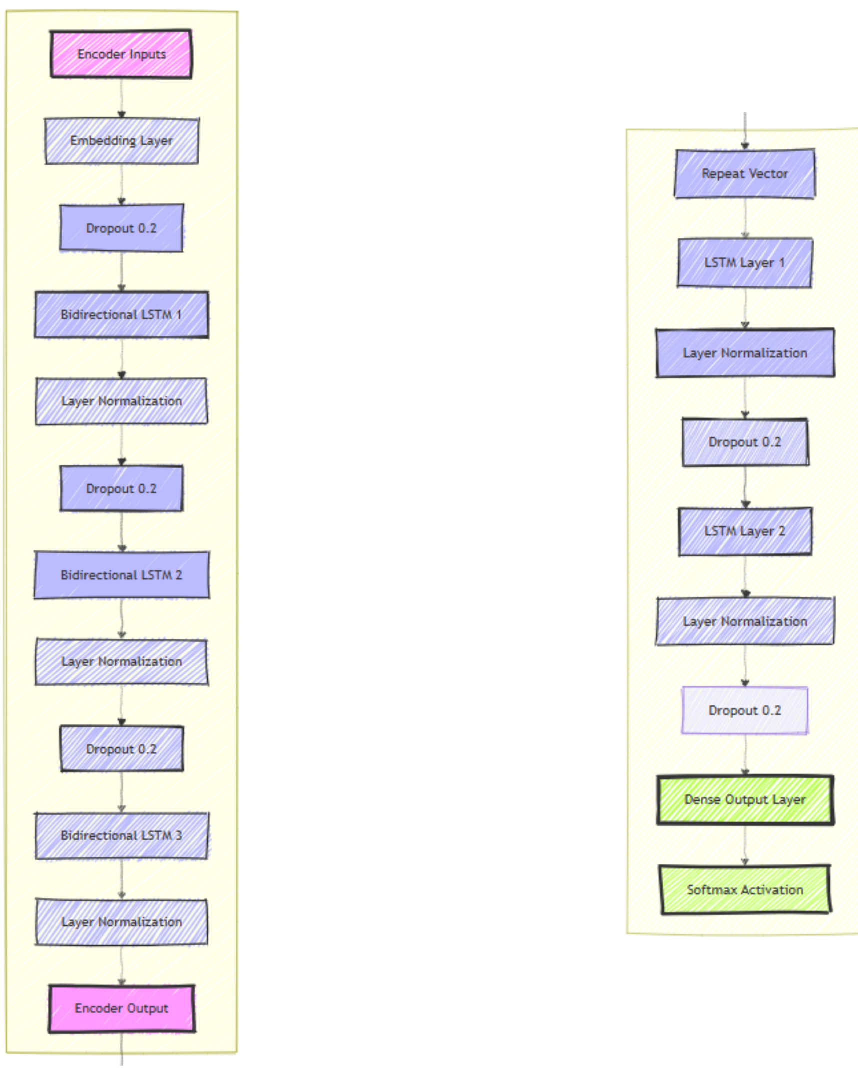
3. Initialize two tokenizers, one for each language, and feed the entire corresponding sentences as corpus.

4. The result is we have a vocabulary of size 6098 in English and 10071 in German.

5. Encode sentences with their corresponding tokenizers, then perform padding, truncation, and finally convert them to tensors.

```
[   2,   67, 4539,    0,    0,    0,    0,    0]
```

Model-Architecture



- **Inputs:**
  - `encoder_inputs`: Input tensor with shape (in_timesteps,).
- **Encoder:**
  - `Embedding` Layer:
    * Converts input integers to dense vectors of fixed size (units) with dropout ($p = 0.2$).
  - `Bidirectional LSTM` Layers:
    * Three stacked bidirectional LSTM layers, with the first two returning sequences.
    * Each LSTM layer is followed by Layer Normalization and Dropout ($p = 0.2$).
- **Decoder:**
  - `RepeatVector`: Repeats the output from the encoder to match the decoder's time steps.
  - `LSTM` Layers:
    * Two stacked LSTM layers (units * 2), returning sequences, with Layer Normalization and Dropout ($p = 0.2$).
- **Output Layer:**
  - `Dense` Layer: Produces outputs of size `out_vocab` with a softmax activation function.

Training Parameters:

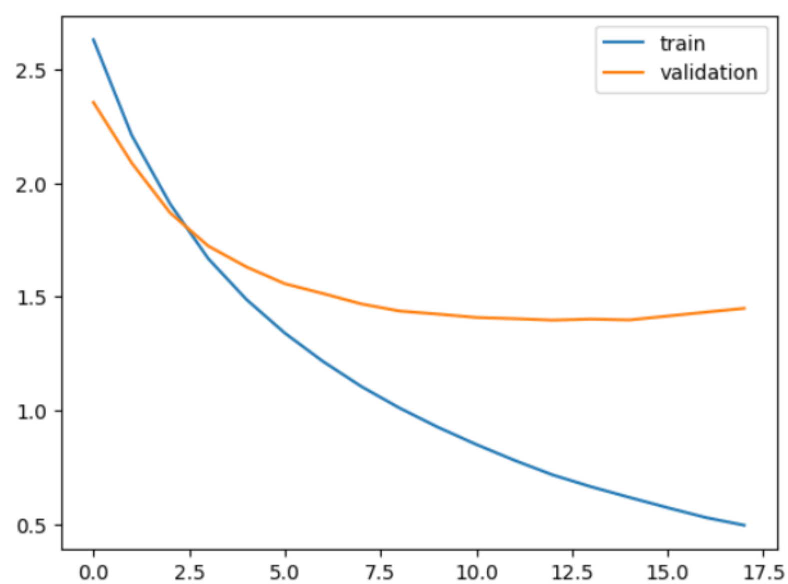
→ The optimizer used here is `Adam` with lr = 0.001.

→ The loss function is `sparse_categorical_crossentropy`.

→ The metric used is accuracy.

→ A scheduler was not used.

→ Early stopping was implemented with a patience of 5 epochs if the model doesn't learn anything new after a while.



Result

After converting numerical predictions back into readable text by mapping indices to their corresponding words.

actual	predicted
does tom eat grapes	did tom eat too
im not really sick	im not really sick
i cant stand it	i dont wear funerals
how deep is it here	how bad is it here
thats my cat	this is my cat
its urgent	its disappointing
i bet you know this	i know you do that
its time to go	we have to go
did tom lie to mary	did tom ask mary
im just watching	im not dead
the lovers kissed	they started
can i see too	can i pay something
tom wasnt prepared	tom wasnt prepared
tom was writing	tom had
toms dog bit me	toms dog bit come