English-Italian Machine Translation Using Seq2Seq Model

Siyu Xiao
Technische Universität Berlin
Berlin, Deutschland
xiao.siyu@tu-berlin.de

1 Introduction

Machine Translation (MT), also known as automated translation, is the process where software translates text from one language to another without human intervention. Currently, machine translation is most effective when a text needs to be understood in another language. The use of deep neural networks has significantly improved the performance and accuracy of machine translation models in recent years.

2 Dataset

The data for this project is based on "European Parliament Proceedings Parallel Corpus 1996-2011". The Europarl parallel corpus is extracted from the proceedings of the European Parliament. This project focuses on English-Italian texts. Data are downloaded from `https://www.statmt.org/europarl/v7/it-en.tgz` and saved respectively in

```
./data/europarl-v7.it-en.en
./data/europarl-v7.it-en.it
```

The total length is `1909115`. To match the computational resource (my laptop), here I use 1% of the total dataset.

3 Implementation

3.1 Data Exploration

- Compare within one language set.
  The total number of sentences in the sampled data is 19091. The average length of English sentences is 157, the average length of Italian sentences (170), the average length difference between English and Italian sentences (14), the maximal length of English sentences (754) and the maximal length of Italian sentences (1320). The distribution of sentence length is shown below. For both language, **the data are subjected to a "long-tailed" data distribution, which means there are some "outliers" to be processed in the next step.**

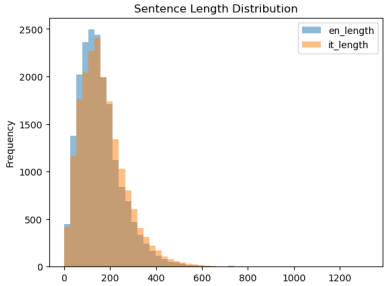

Figure 1: Sentence length distribution.

- Compare between language set.
  Similarly, to identify outliers, here I calculate the sentence-based difference. There is no huge length difference between 2 sentences in different language with the same meaning.

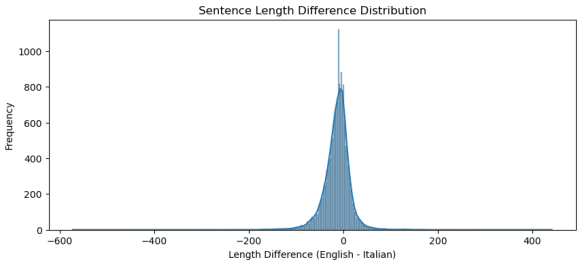

Figure 2: Sentence-based length difference.

3.2 Pre-processing

Here I performed the following pre-processing step:

(1) **First, ensure that all sentences are string and remove NaN values.** Since NaN could cause errors duing model and traning is particularly sensitive to the data with missing values.
(2) **Lowercase the text.** This is a standard pre-processing step in NLP, reducing the complexity of the words. And English as well as Italian are not too sensitive to the capital letters.
(3) **Remove xml label line.** This step removes the metadata or markup tags that are not part of the actual text content.
(4) **Remove repeated blanks**. Clean up excessive whitespace, improve the tokenization.
(5) **After the steps above, remove the empty lines.**

```python
def preprocess_text(text):
    text = [str(line) for line in text if pd.notna(line)]
    text = [line.lower() for line in text]
    text = [line for line in text if not line.startswith('<')]
    text = [re.sub(r'\s+', ' ', line).strip() for line in text]
    text = [line for line in text if line.strip()]
    return text
```

Tokenization is performed in later tasks. Stemming is not implemented here, given the fact that different "form" of words in language is significant for understanding the semantic information.

3.3 Neural Machine Translation

3.3.1 *Data Split & Tokenization*.

Split data into train, validation and test sets. Use 20% of data as the test set.
Create tokenizer for each language set and convert text into sequences. And creating individual tokenizers for each language is to ensure each language has a corresponding vocabulary. The actual implementation use the tokenizer from `tensorflow.keras.preprocessing.text`.

```
def create_tokenizer(texts):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(texts)
    return tokenizer
```

It is also necessary to convert text data into integer sequences and pads them to a specified length. Here I pad the sequences to the maximum length for each language.

```
def encode_sequences(tokenizer, length, lines):
    seq = tokenizer.texts_to_sequences(lines)
    seq = pad_sequences(seq, maxlen=length, padding='post')
    return seq
```

3.3.2 *Create RNN Based Seq2Seq Model & training*.

Develop an RNN based sequence to sequence model (encoder-decoder) to translate English input into Italian text. Architechture:

- *ENCODER - Input layer*. The length of input sequences varies.
- *ENCODER - Embedding layer*. Convert the input into dense vectors, dimension of `embedding_dim`.
- *ENCODER - LSTM Layer*. To implement RNN, here I use 1 layer of LSTM to process the embedded sequences. The output of `hidden state` and `cell state` are passed to the decoder.

- *DECODER - Input layer*.
- *DECODER - Embedding layer*. This time convert the target word indices into dense vectors.
- *DECODER - LSTM Layer*. Process the embedded target sequences.
- *DECODER - Dense Layer*. Output the probability distribution over the target vocabulary.

To match the computational capacity of my laptop, I tried several combinations based on intuitions and select the following hyperparameters providing the best "trade-off":

- *embedding_dim* = 150 - Emedding dimension (based the average sentence length).
- *units* = 32 - LSTM dimension, the largest I can use.
- *epochs* = 10
- *batch_size* = 16 - Generate the best training accuaracy.

English-to-Italian translation results analysis:

The model capacity is clearly not sufficient. Overall, It can be seen that both training and validation accuracy gradually increase with the number of epochs and stabilize within 10 epochs with no significant overfitting or underfitting.

Table 1: English-to-Italian translation scores

	accuracy	loss
train	0.8473	5.6443
validation	0.8496	5.8045
test	0.8492	5.8096

For a DL model, a loss of 5 is normally not acceptable.

- First, the number of LSTM units is not enough, and thus the model is not able to capture the complex relationships between input and output sequences. But unfortunately, increasing *units* didn't work out with Jupyter Notebook on my laptop.
- The pre-processing seems to be just fine. And a dataset of 19,000 also could be fitted if the model is powerful enough. Another problem might be that the embedding is randomly initialized. Therefore, the embedding didn't provide effective word vector representations and it's hard to learn effective translation rules.

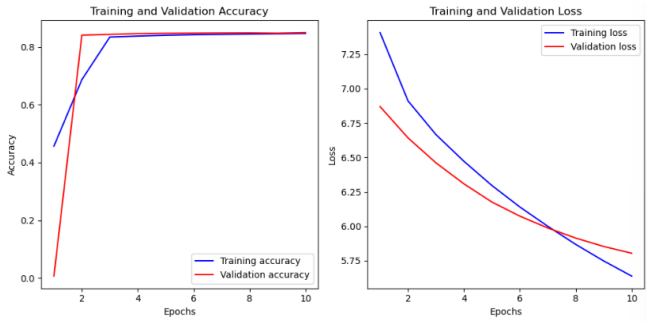

Figure 3: English-to-Italian task results.

3.3.3 *Translate Italian to English*.

Swap dataset. The training and validation accuracy fluctuate more with the number of epochs, possibly due to data imbalance, but still show an overall increasing trend. Results are shown below:

Table 2: Italian-to-English translation scores

	accuracy	loss
train	0.8421	5.2220
validation	0.8406	5.3447
test	0.8393	5.3557

Comparison:

The training and validation accuracy for both tasks are very close, indicating good generalization ability of the model on these tasks. However there are still differences in language structure and grammar between English and Italian. The loss in the Italian-to-English translation task is lower than in the English-to-Italian translation task. This may be because the model finds it easier to learn alignment rules when translating from Italian to English, and can be further improved by introducing an attention mechanism.

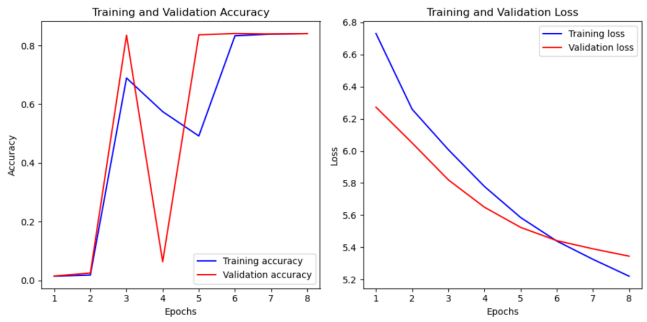

Figure 4: Italian-to-English task results.

3.3.4 *Character-based Model*.

Here I performed a character-based model trained on the English-to-Italian task. I reused the sequence encoding function and model above and modified the tokenizer.

```
tokenizer = Tokenizer(char_level=True)
```

Without changing the architecture of the model, like increasing the size of LSTM, it's very challenging to learn semantic information effectively from individual characters, because a character-level model needs to capture and combine characters over longer sequences to understand and generate meaningful words. Due to limited computational capacity, the accuracy is relatively low while the loss has already reduced to 2.

3.3.5 *Other Subtasks*.

For the interpretation of model results and investigation into the impacts of sentence characteristics, the evaluation is clarified in chapter 3.4, since the model using attention mechanism gives better results and provides more solid evidence. Moreover, I failed to perform the implementation of different embeddings.

3.4 Neural Machine Translation with Attention

In this part, I performed EN-to-IT task using attention mechanism, given that with the attention mechanism, the model can selectively pay attention to relevant parts of the input, improving the quality of translations, especially for long and complex sentences.

```
# implement attention
attention = Dot(axes=[2, 2])([decoder_outputs, encoder_outputs])
attention = Activation('softmax')(attention)
context = Dot(axes=[2, 1])([attention, encoder_outputs])
decoder_combined_context = Concatenate(axis=-1)([context, decoder_outputs])
```

Implementation: First calculate the attention weights (normalized using a softmax activation). Then the context vector is computed as a weighted sum of the encoder outputs, which is concatenated with the decoder outputs. Therefore the decoder can dynamically focus on relevant parts of the input sequence for each output time step. I used the same hyperparameters (*embedding_dim* = 150). The results are shown below.

Table 3: EN-to-IT with attention results.

	accuracy	loss
train	0.8500	0.9798
validation	0.8528	0.9664
test	0.8522	0.9729

Results: The attention mechanism provides better context and alignment between the input and output sequences. This improvement allows the model to make more confident predictions, which reduces the loss. The model doesn't significantly impact the overall accuracy.

Visualization of the attention weights: I used the heatmap with the input sequence of an English sample. The heatmap shows how the attention shifts dynamically across the input sequence and improves translation accuracy by considering the most relevant context for each word.
For instance, "parlamento" (parliament) and "europeo" (European) in the output sequence have high attention weights for specific words in the input sequence, indicating that the model is correctly focusing on the relevant context.

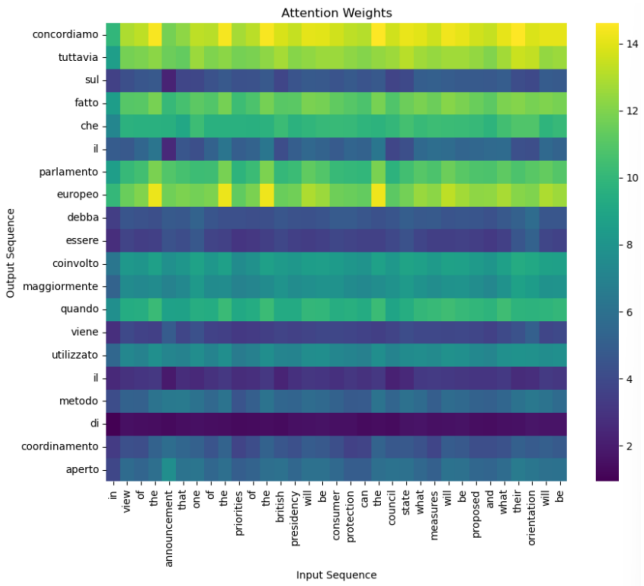

Figure 5: Heatmap of attention weights.

Interpretation of the model results (Length Impact):

As shown in the figure 6, when the sentence length increases, the loss rises and accuracy drops, which indicates that the model performs worse on longer sentences compared to shorter ones.
Additionally, there are significant fluctuations at different lengths, especially when the length is greater than 60. The lack of longer sentence samples could be the reason for that, which leads to unstable model performance on these samples. By contrast, for shorter sentences (between 20 and 60), the accuracy is relatively stable and high (around 0.8 to 0.9).
The reason for high individual accuracy (reaching 0.9) compared to the overall accuracy of only 0.85, is that Keras averaged the accuracy (as well as the loss) over all sentence samples.

Therefore, to improve the model performance, we could increase the number of longer sentence samples to balance the data distribution, helping the model better learn the characteristics of longer sequences. Data augmentation and other unsupervised methods could be implemented given this insight.

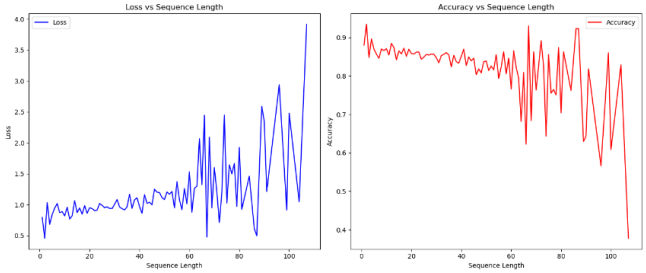

Figure 6: Interpretation of the model results.

Comparison:
The attention mechanism enhances the model's ability to process long and complex sentences, reducing loss and improving translation quality. The former model without attention weights converges quickly but hits a performance bottleneck, unable to improve further.

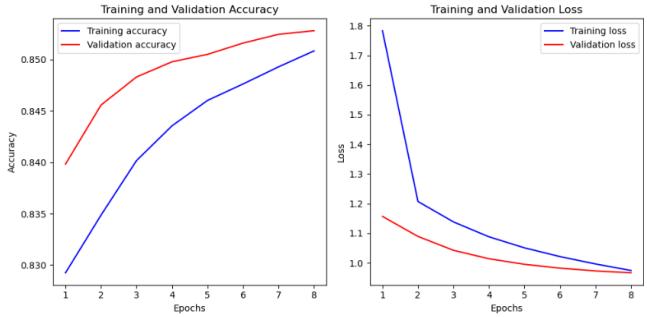

Figure 7: English-to-Italian task (using attention) results.

4 Conclusion

The Seq2Seq model's performance is highly dependent on the length of the input sequences, and smaller number of long sentence (sequence) samples leads to unstable model performance on these samples, indicating that imbalanced data distribution affects the overall performance of the model.
However, the attention mechanism significantly improves the model's performance, particularly for longer sequences. The attention mechanism allows the model to dynamically focus on relevant parts of the input sequence, providing better context and alignment between the input and output sequences.
To further improve the performance of the translation model:

- Increase Model Capacity: Increasing the number of LSTM units and layers could help the model capture more complex relationships between input and output sequences, although this requires more computational resources.
- Data Augmentation: Increasing the number of longer sentence samples can help balance the data distribution, allowing the model to better learn the characteristics of longer sequences.