Project Report – NLP

Written by 209948728 & 931188684

Work repartition

For this project, Yonathan mostly worked on the 'basic part' (getting at least a *sacrebleu* score of 30% on file *val.labeled*). And Raphael mostly worked on the competition part. Then in this report, Yonathan will explain our training, inference, and test process for the 'basic part.' Then, Raphael will elaborate on how we extend the 1our model to get better results for the test & comp part (using the given roots and modifiers).

Training

Let's outline the most important steps of this section:

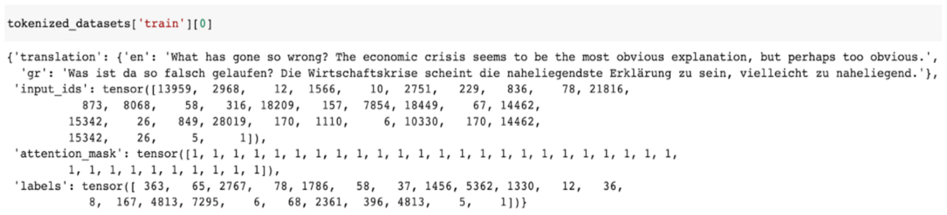
1- Create dataset & pre-process function: The train and validation data are converted into a dictionary with English and German translations. Each dictionary in the list contains two keys: "en", which represents the English source text, and "gr", which represents the German target text. Then this dataset is taken as a parameter to create two lists: inputs and targets. The input list is created by concatenating a prefix (with the English source text from each translation dictionary. The target list is created by taking the German target text from each translation dictionary.

2- Model: We used a pre-trained model from Hugging Face. After several training tests, we finally kept the most efficient one in our case (based on the *sacrebleu* score) which was T5-Base[1]. Then we will fine-tune this model in the tokenization step.

3- Tokenization: the tokenizer function is responsible for tokenizing the input and output texts, converting them into numerical representations that can be fed into the NLP model. For this task, we used the function AutoTokenizers which was using our chosen model (T5-Base). For the tokenization we mostly worked on:
   a. The *max_length* argument specifies the maximum length of the tokenized inputs and outputs, and the truncation argument specifies whether to truncate texts that exceed the maximum length.
   b. The *preprocessing function*, tokenizing the inputs and outputs and creating our final *tokenized_datasets* variable. When mapping us *tokenized_datasets* we also used the "batched=True" argument specifies that the preprocessing should be applied to batches of items rather than individual items, which speeds up the process (allows the function to take advantage of parallel processing capabilities).

[1] More info: https://huggingface.co/t5-base

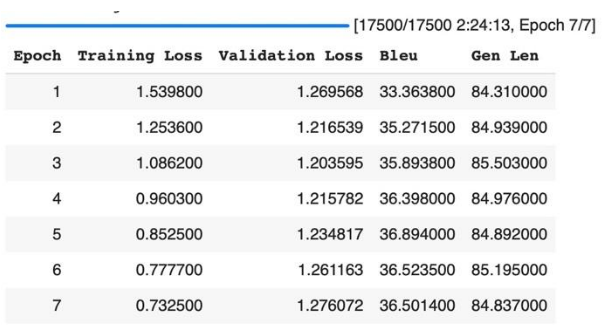
Let's see an example for the first sentence in the tokenized_datasets[2]:



```
tokenized_datasets['train'][0]
```

```
{'translation': {'en': 'What has gone so wrong? The economic crisis seems to be the most obvious explanation, but perhaps too obvious.',
  'gr': 'Was ist da so falsch gelaufen? Die Wirtschaftskrise scheint die naheliegendste Erklärung zu sein, vielleicht zu naheliegend.'},
 'input_ids': tensor([13959,  2968,    12,  1566,    10,  2751,   229,   836,    78, 21816,
           873,  8068,    58,   316, 18209,   157,  7854, 18449,    67, 14462,
         15342,    26,   849, 28019,   170,  1110,     6, 10330,   170, 14462,
         15342,    26,     5,     1]),
 'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
 'labels': tensor([ 363,   65, 2767,   78, 1786,   58,   37, 1456, 5362, 1330,   12,   36,
            8,  167, 4813, 7295,    6,   68, 2361,  396, 4813,    5,    1])}
```

4- Compute metrics: for the training process. We used the *sacrebleu* calculation has it appears in the Hugging Face documentation.
5- Seq2Seq trainer: We then define our hyper-parameters for our training step which was using the Seq2Seq model[3]. During this step we tried several hyper-parameters and several models. Our lower result obtained were 0.71 (using t5-small/ batch size = 16 / 5 epochs) and the best results obtained were with the following hyper-parameters:
   - evaluation_strategy="epoch",
   - learning_rate=2e-4,
   - batch size = 4,
   - weight_decay=0.01,
   - total epochs = 7,
   - logging_steps=1000,
   - generation_max_length=500[4]
   - model: T5-Base

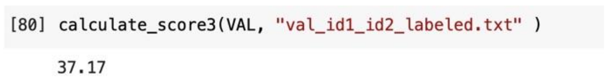


[17500/17500 2:24:13, Epoch 7/7]

Epoch	Training Loss	Validation Loss	Bleu	Gen Len
1	1.539800	1.269568	33.363800	84.310000
2	1.253600	1.216539	35.271500	84.939000
3	1.086200	1.203595	35.893800	85.503000
4	0.960300	1.215782	36.398000	84.976000
5	0.852500	1.234817	36.894000	84.892000
6	0.777700	1.261163	36.523500	85.195000
7	0.732500	1.276072	36.501400	84.837000

Inference

After saving our model, we translated every German sentence to English and created the val_id1_id2.labeled as required. We then compute the sacrebeu score using the given functions and obtains the following score[5]:



```
[80] calculate_score3(VAL, "val_id1_id2_labeled.txt" )

     37.17
```

[2] The tokenized_dataset is a dictionary with four keys: *translation, input_ids, attention_mask,* and *labels:*
- *Translation:* self-explanatory
- *input_ids:* a tensor that contains the tokenized input text in the form of numerical IDs. Each ID represents a specific token in the vocabulary.
- *attention_mask:* a tensor used to tell the model which tokens in the *input_ids* tensor are important and should be paid attention to during training. It has a length equal to the length of the *input_ids* tensor, with a value of 1 for each token that should be attended to, and a value of 0 for each token that should be ignored. By setting a correct *attention_mask,* we can help the model focus on the relevant parts of the input while ignoring irrelevant parts, which can improve its performance.
- *Labels:* a tensor that contains the tokenized output text in the form of numerical IDs.

[3] The seq2seq trainer takes in a sequence of words in one language and outputs a sequence of words in another language. The trainer works by training the model to minimize the difference between the predicted output and the actual output. During training, the model is fed pairs of sentences in the source and target languages, and it adjusts its parameters to make more accurate translations.

[4] Same max_lenght as we used before in our pre-processing.

[5] The score is a little bit higher than our train since we used a different calculation method for the *sacrebleu* score.

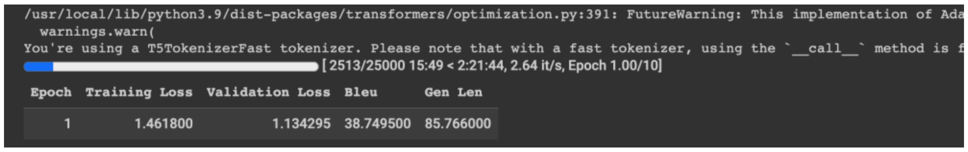
Test & Competition

To improve the NLP algorithm for translating German to English, several approaches were taken. One of these was the use of a new data set that included root and modifier information, which helped to provide a more accurate translation. Additionally, we tried to switch the positions of elements in the prompt of the T5 model, for example putting the roots and modifiers information before or after the sentences, which was another strategy that showed promise.

Another approach that was attempted was the addition of POS tags to the roots and modifiers in the dataset. While this did not initially yield positive results, the team believes that it could potentially improve accuracy with further debugging.

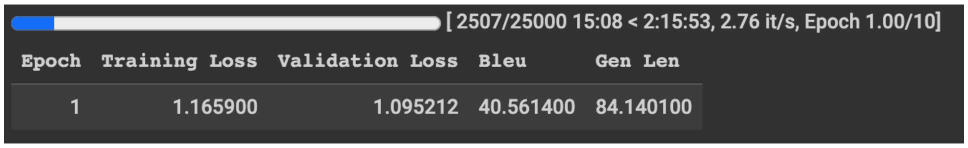
Here are our results:

After adding the Root/Modifiers:



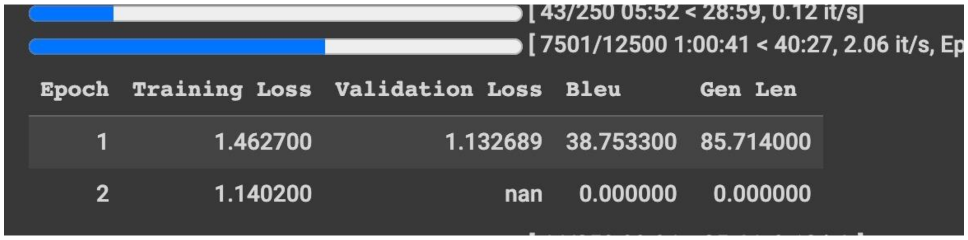
```
/usr/local/lib/python3.9/dist-packages/transformers/optimization.py:391: FutureWarning: This implementation of Ada
  warnings.warn(
You're using a T5TokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is f
[2513/25000 15:49 < 2:21:44, 2.64 it/s, Epoch 1.00/10]
Epoch  Training Loss  Validation Loss  Bleu       Gen Len
    1       1.461800         1.134295  38.749500  85.766000
```

Playing with the prompt :



```
[2507/25000 15:08 < 2:15:53, 2.76 it/s, Epoch 1.00/10]
Epoch  Training Loss  Validation Loss  Bleu       Gen Len
    1       1.165900         1.095212  40.561400  84.140100
```

After adding the POS tag for the roots :



```
[43/250 05:52 < 28:59, 0.12 it/s]
[7501/12500 1:00:41 < 40:27, 2.06 it/s, Ep
Epoch  Training Loss  Validation Loss  Bleu       Gen Len
    1       1.462700         1.132689  38.753300  85.714000
    2       1.140200              nan   0.000000   0.000000
```