Assignment 4 Deep Learning

Filippo Casari

December 2021

1 1 Problem Setups and Preliminaries

1.1

The number of sentences:

- Training:

  $total_{sentences} = (31250-1)\ batches* \ 64(per\ batch)+62(last\ batch) = 1'999'998$

- Validation:

  $total_{sentences} = (157-1)\ batches* \ 64(per\ batch)+16(last\ batch) = 10'000$

Number of chars:

- Training:
  number of chars=

$$1'999'998 * \ 50 = 99'999'900\ chars$$

- Validation:
  number of chars=

$$10'000 * \ 48 = 480'000\ chars$$

The average for the training set is about 38 chars for question/sentence with a standard deviation of 5 chars.
The average for the validation set is about 39.37 chars per sentence with a standard deviation of 3.2.

2 Dataloader

2.1

We got 2 vocabularies. Indeed, we do not have the same vocabulary, the source one is composed by 33 items while the other (target) is composed by 14 items.

2.2

The purpose of the unknown char is explained in the method "get index" because while we read the source or target file to load data, if we encounter a new char (labeled as unknown) we add it into the vocabulary.

3 Model

I create my class for Transformer called "MyTransformer". I implemented 3 methods:

- **__init__** : when I create my model I initialize the objects Embedder (one for the target, one for the source), Positional Embedding ((one for the target, one for the source), the d_model to store the dimension of the model, the memory (tensor to save my memory after encoding to access to it outside the class for the greedy algorithm), the Transformers of pytorch, and the Linear Layer. This method takes as args all the parameters that the assignment suggested to get.
- **forward**: forward my model according to Transformer's step
- **greedy**: implements the greedy search

4 Greedy search

1.1, 1.3, 1.4

I call my model.greedy() each 300 batches of the training set and I calculate the accuracy on a validation batch from validation set. My greedy algorithm exits when the last predicted char is "eos" or when the length of my prediction exceeds target sequence. I forward the encoder just one time and then I forward multiple times the decoder to predict the output. My idea is to begin with "sos" and insert it into the decoder just for the first decoder step. Moreover, I concatenate my prediction with the output of the decoder as the assignment suggested.

1.2

With nn.Transformer I cannot get an output length as I want because the "len" of my output must match the "len" of the input of the decoder. I have to build a custom method to iterate multiple times the decoder forward step to implement the greedy search, and as the assignment said I have to concatenate the previous input with my output each time.

5 Accuracy

I compute the accuracy as:

$$Accuracy = \frac{number\ of\ correct\ answers\ in\ a\ batch}{batch\ size}$$

In evaluation, training and greedy search.

6 Training

I implemented my training with the following policy:

- each batch: computing training loss
- each 10 batches: implementing the gradient accumulation
- each 200 batches: computing training accuracy, validation loss (average) and validation accuracy
- each 300 batches: computing the greedy search with a validation batch. Metrics showed: 3 examples, greedy accuracy.

I used the Cross Entropy Loss because it has already the Softmax function internally. I do not need to apply a Softmax after the linear layer (except for greedy).

7 Experiments

I reached the validation accuracy over 90.0% after about 10 minutes with google colab that allows me to get about 10 iterations (batches) per second during my training. Sometimes google colab allows me to extract about 30 batches for second, so I got the validation accuracy (over 90.0) after 4 minutes. Indeed, the model performs really well with these suggested parameters.
Note: for the first image, the x axis batches must be multiply by 200.

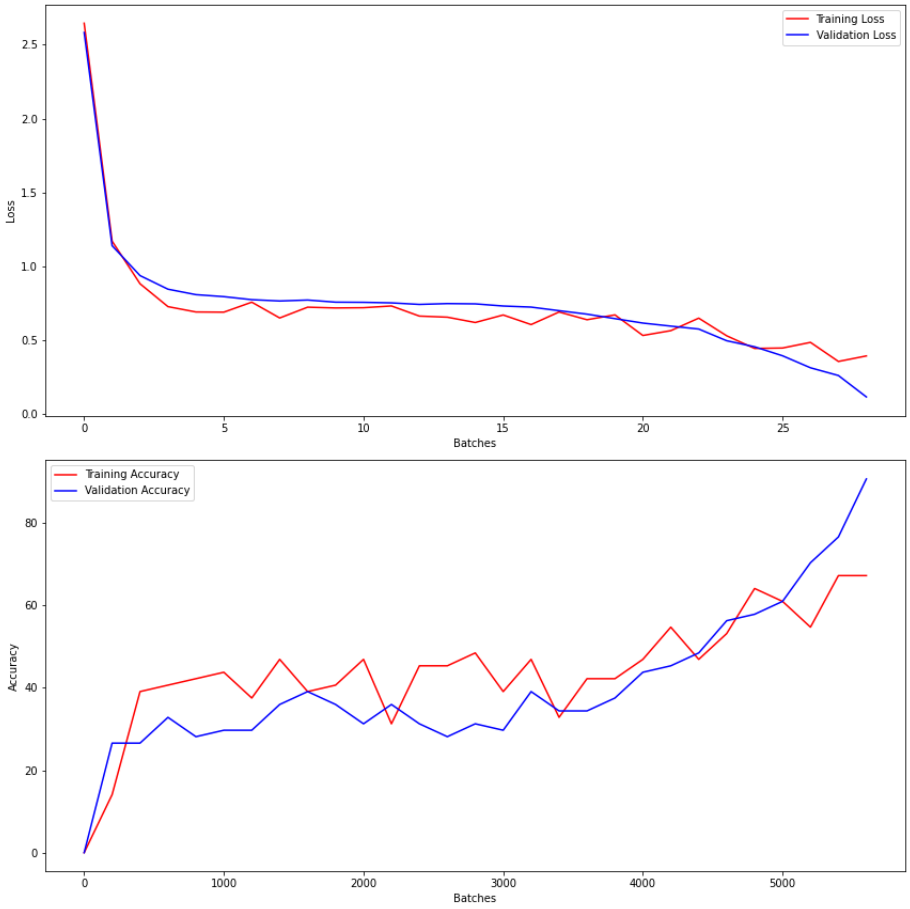

Figure 1: Results of point 2

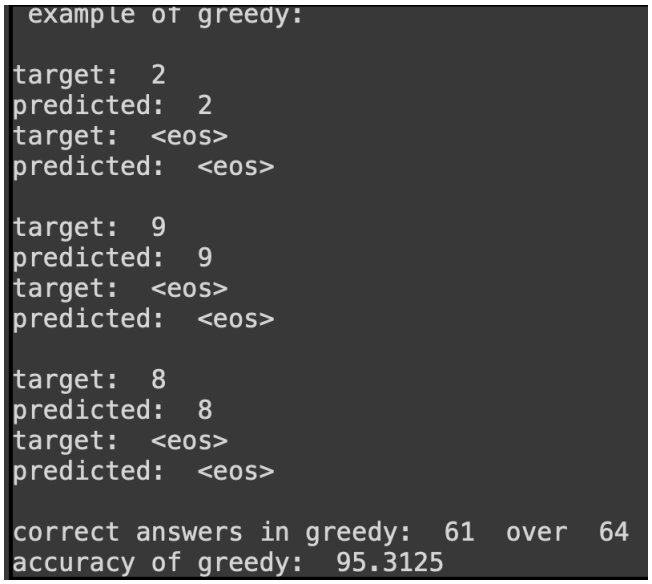

Figure 2: Example of greedy search

7.1 Hyper-parameter tuning

First changing

I changed the encoder and decoder size to **encoder size: 1** and **decoder size: 1** I show the results below. As you can see, it performs worst: after 10'000 iterations I got validation accuracy greater than 90.0%.
Note: for the first image, the x axis batches must be multiply by 200.

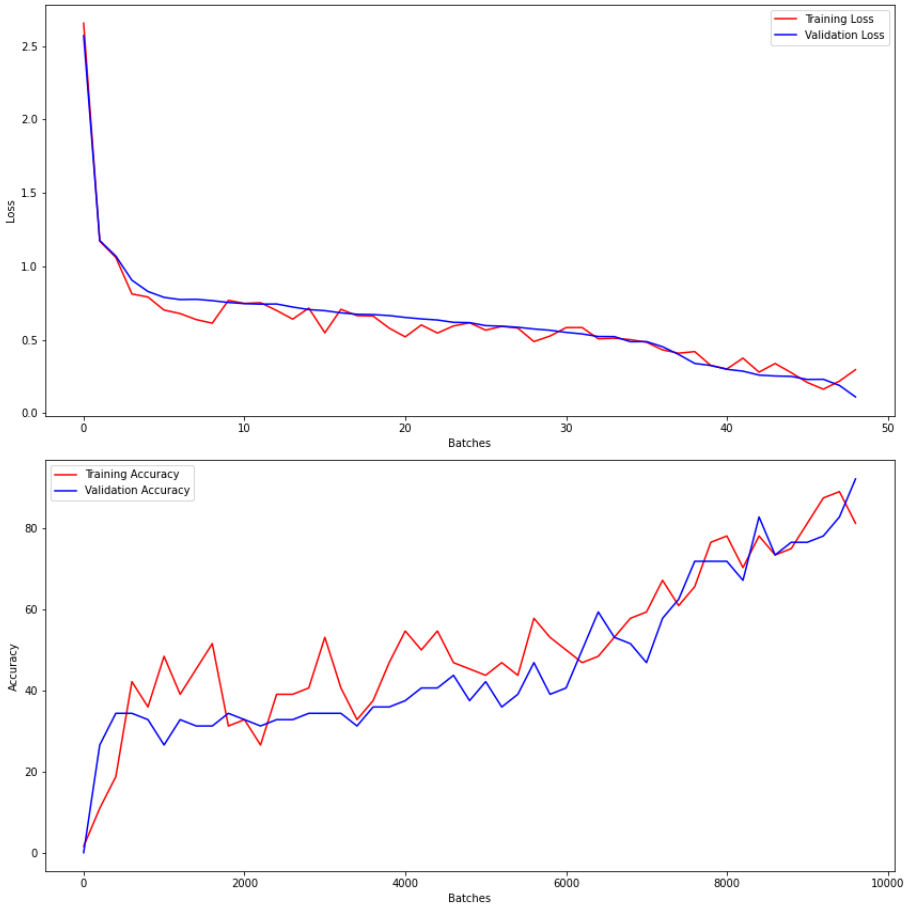

Figure 3: Changing hyperparameters, first change

Second changing

I changed the feed forward hidden layer from 1024 to 512.

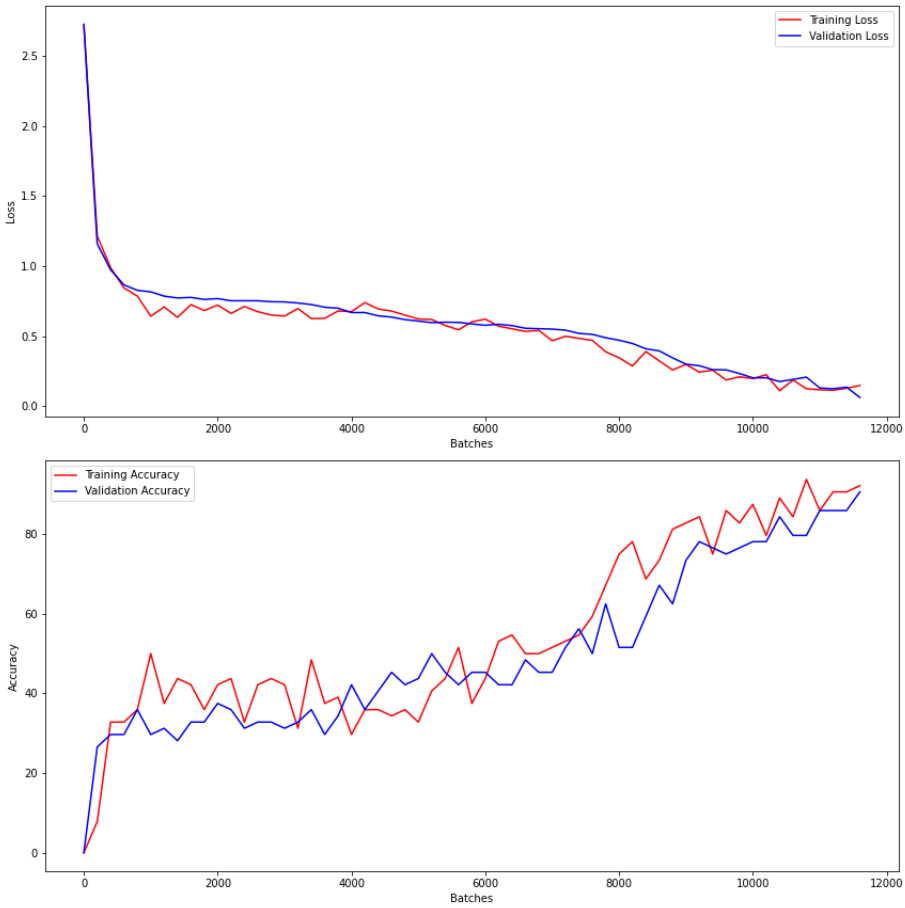

Figure 4: Changing hyperparameters, second change

After this change, my model performed really worst. It took about 20 minutes to reach the target validation accuracy. I had to run more than 10'000 batches to get this results.

7.2 compare - sort Dataset

With same parameters as before the changes, I tried my model also with the second dataset and I got these results as showed in the figure below.
I tried my model just for few iterations because google colab didn't let me to run it completely. This is an intermediate result.

The accuracy of training and validation reached just 4.6 at most%.

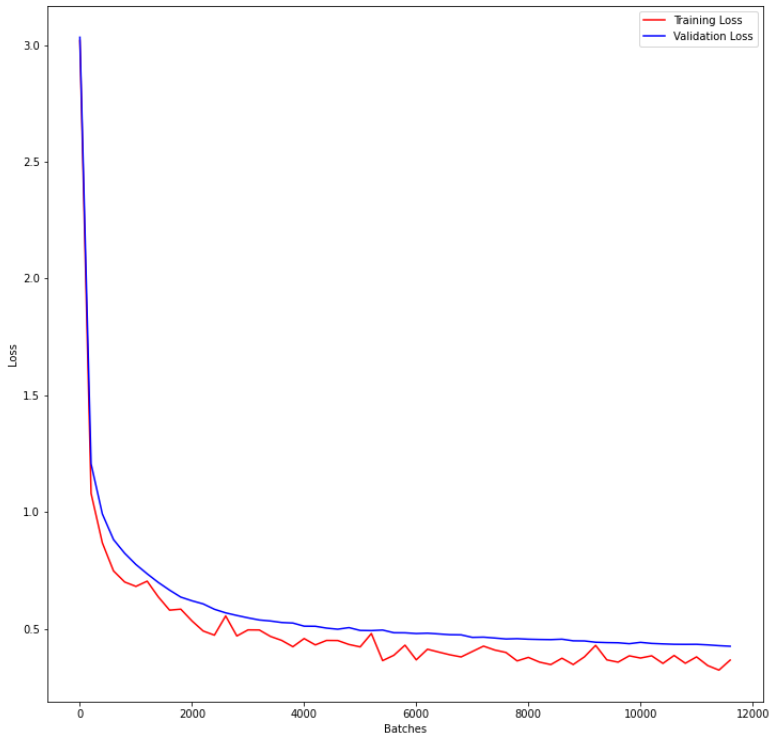

Figure 5: Second Dataset, Losses

8 References

- https://stackoverflow.com/questions/67926524/pytorchs-crossentropyloss-how-to-deal-with-the-sequence-length-dimension-with
- https://pytorch.org/tutorials/beginner/translation_transformer.html
- https://github.com/dreamgonfly/transformer-pytorch