Deep Learning Lab
Assignment 4

Pietro Miotti

January 6, 2022

1 Problem Setups and Preliminaries

1.1 Report the number of sentences and characters in the training and validation sets.

Training Set
Total Number of lines: 1999998
Total Number of words: 14531838
Total Number of Characters: 78429947
Average Question length: 38
Average Answer length: 1

Validation Set
Total Number of lines: 10000
Total Number of words: 73641
Total Number of Characters: 413219
Average Question length: 40
Average Answer length: 1

2 Dataloader

2.1

No, it creates a separate vocabulary for the source and the target sets.

2.2

All the vocabularies contain the *unk* token as the token with id=1. It's role is similar to a wildcard: when training/validating the model encounter a character

that is not present in vocabulary is encountered, the unknown token is considered. For the creation of the vocabulary instead, every unknown character is added in the vocabulary and the 'unk' token is not considered.

3 Model

Please consider the implementation of the model in the *Model* class reported in the *main.py* file from line 251.

4 Greedy search

4.1

Please you can find the implementation of the greedy search in the *main.py* file, as the *greedy_choice_batch* function from line 332.

4.2

The main problem that I see in the implementation of nn.Transformer is that it necessary to create two ad-hoc functions for calling the encoder and the decoder while creating each time manually new masks. Since it is necessary for every new letter predicted to create new masks and import them into the device, the performance of the search decrease exponentially with the length of the sequence to predict.

4.3

Please check the code in the *greedy_choice_batch* function.
In order to check for the *eos* id, I've created a mask that I use to check if all the elements in the batch have terminated the string (i.e. a <eos>token is predicted), for the elements in the batch that have achieved the <eos>id, every new letter after <eos>is replaced with the <pad>token.

4.4

Please check the code in the *greedy_choice_batch* function.

5 Accuracy computation

Please check the code in the *accuracy* function from line 384.

6 Training

6.1 Which loss function do you use? Why? (1 sentence)

I used the Cross Entropy Loss since it is a class-matching problem where classes are the ids of the target vocabulary.

6.2

Please you can find the training code after line 420 of the *main.py* file.

6.3 Gradient accumulation

As requested, in the training I perform Gradient Accumulation as you can see in the implementation reported after line 420 of the *main.py* file.

7 Experiments

7.1

As requested, by using the hyperparameters reported in the paper I have managed to obtained the desired accuracy (as also reported in the following subsection), please you can find the code after line 420.

7.2

Please check the training loss vs the validation loss in the plot reported in Fig: 1 and the accuracy of both validation and training in Fig: 2

- QUESTION: What is the ten millions digit of 10729338?<pad><pad><pad><pad><pad>
  ANSWER: <sos>1 <eos>
- QUESTION: What is the ten millions digit of 71864338?<pad><pad><pad><pad><pad>
  ANSWER: <sos >7 <eos>
- What is the thousands digit of 6970835?<pad><pad><pad><pad><pad><pad><pad><pad><pad>
  ANSWER: <sos>0 <eos>

7.3 Hyperparameter Tuning

7.3.1 1 layer encoder, 1 layer decoder

By considering 1 layer for the encoder and 1 layer for the decoder, as you can see from the plots reported in Fig 3 I still managed to obtain $\approx 100\%$ accuracy in the validation set (even if it has required more iterations)

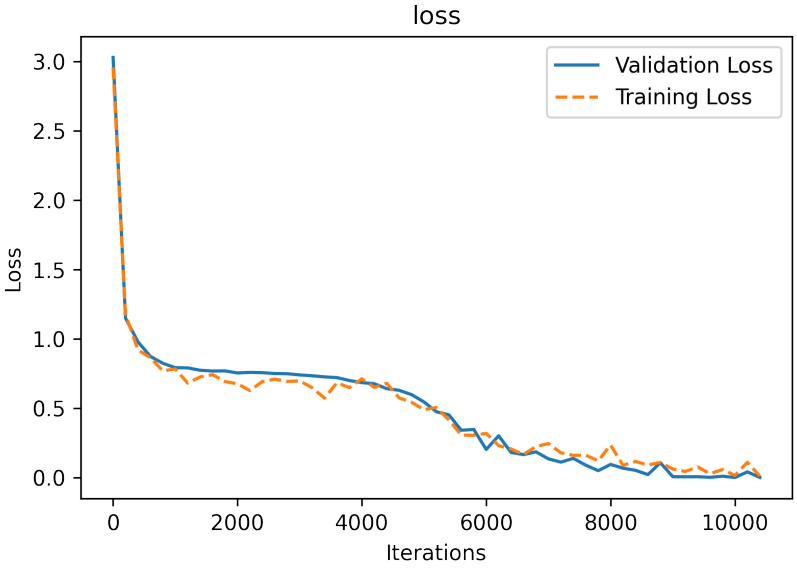

Figure 1: Loss for numbers - place value task

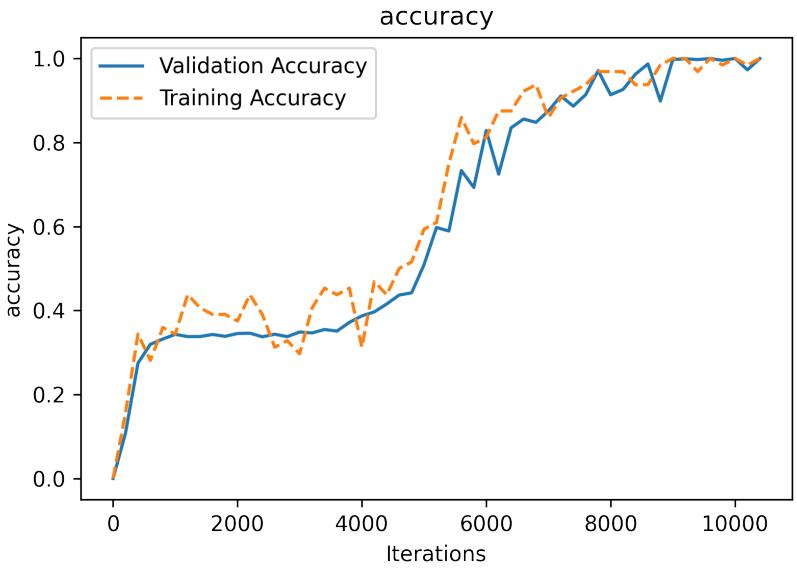

Figure 2: Accuracy for numbers - place value task

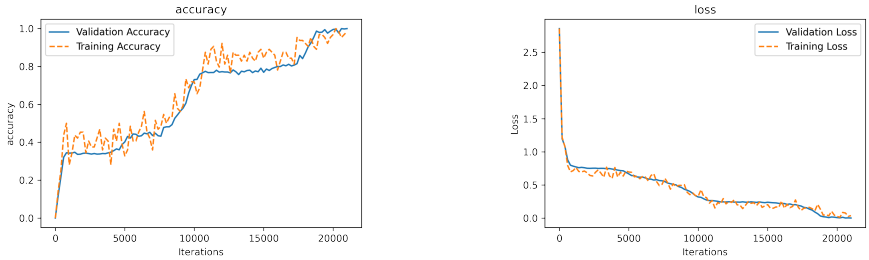

Figure 3: Accuracy and Loss for the Transformer with 1 layer encoder and 1 layer decoder

7.3.2 embedding_size=32 instead of 256

Transformer architecture is very stable also if we reduce the embedding size from 256 to 32, from the plot reported in Fig:4 we can see that also in this case the model converges slowly to 100% accuracy.

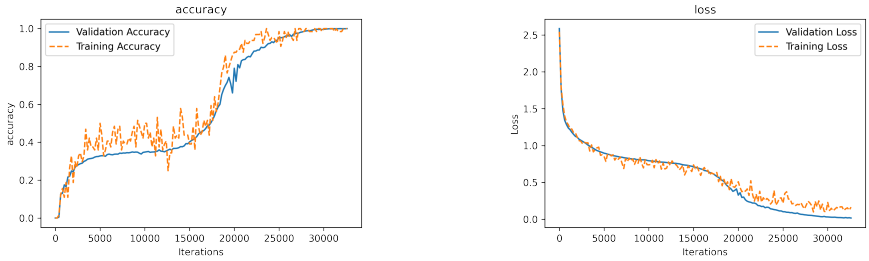

Figure 4: Accuracy and Loss for the Transformer with emb_size=32

7.4 compare - sort

By training the model for the compare-sort dataset I have managed to achieve the 90% performance in $\approx$ 3 hours of training, as expected. Please find in Fig: 5 the learning curves.
This task is extremely more difficult than the previous one since the model does not only need to learn the semantic of positioning (as the previous task) but also the concept of ordering and hence the concept of greater and lower and associate them to the position concept.

7.5 linalg

I have also tried to run the model for the *linalg* dataset for $\approx$ 4/5 hours of training, unfortunately I didn't managed to achieve high accuracy (just 30% for the validation set) but I am reporting in Fig: 6 some intermediate results

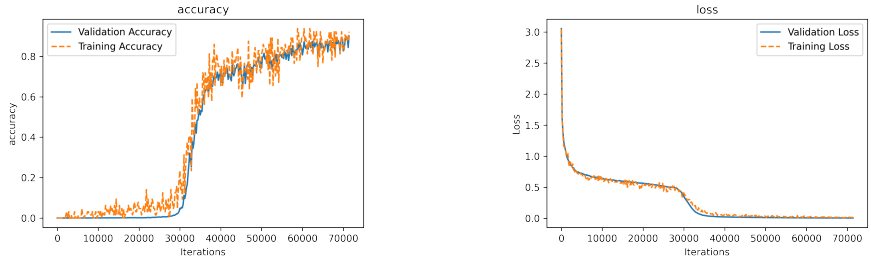

Figure 5: Accuracy and Loss for the compare dataset

that let us imagine that with longer training the model should have managed to achieve higher accuracy

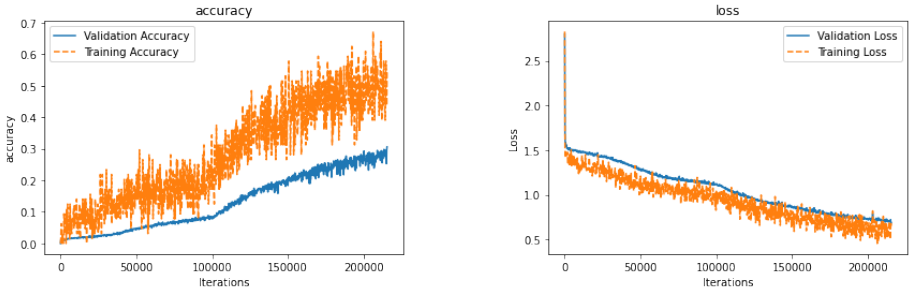

Figure 6: Accuracy and Loss for the linalg dataset