DeepLearning Lab **2022/2023**

Student: LUCA PERNIGO StudentID:19-993-658

Assignment 4

Due date: 15 January 2023, 10:00 PM

1. Problem Setups and Preliminaries

The number of sentences and characters for each file are easily obtained by typing **wc count train.x** inside the terminal. Additionally, the average length of questions and answers in terms of characters is obtained by diving the numbers of total characters by the number of sentences.

Numbers Place Value	Sentences	Characters	Avg. sentence length
train.x (train questions)	1999998	78429947	39.21501272
train.y (train answers)	1999998	3999996	2
interpolate.x (validation questions)	10000	413219	41.3219
interpolate.y (validation answers)	10000	20000	2

Comparison sort	Sentences	Characters	Avg. sentence length
train.x (train questions)	1999998	81588795	40.79443829
train.y (train answers)	1999998	37413762	18.70689971
interpolate.x (validation questions)	10000	434938	43.4938
interpolate.y (validation answers)	10000	213802	21.3802

Algebra linear 1d	Sentences	Characters	Avg. sentence length
train.x (train questions)	1999998	73368093	36.68408318
train.y (train answers)	1999998	5968004	2.984004984
interpolate.x (validation questions)	10000	419669	41.9669
interpolate.y (validation answers)	10000	32978	3.2978

2. Dataloader

2.1.

The provided implementation uses the same vocabulary for the input (source) and output (target) of the model by default since in the default case both **src_vocab** and **tgt_vocab** are given the full vocabulary **Vocabulary()**.

2.2.

When a particular string is not within the **string_to_id** the function **get_idx** cannot return any index; the unknown token is used in order to return an index for all the tokens that are not inside our vocabulary.

3. Model

3.1.

In this section the function **forward separated** was implemented. This function uses self.encoder and self.decoder instead of directly using self.transformer in the forward computation of the model.

4. Greedy search

4.1.

Then, the **greedy_search** function was implemented. To do so, the encoder was forwarded one time while the decoder was forwarded multiple times. Additionally, at each decoding step, the maximum between all output tokens was taken and appended to the decoder input sequence.

4.2.

The problem with the standard implementation of nn.Transformer when using it in a search algorithm is that nn.Transormer encodes the input sentence at each time step.

4.3.

A stopping criteria for the greedy search algorithm was specified. In this implementation when the model outputs end of sentence the greedy search is stopped. That is before entering a new decoding iteration, it is first checked that the selected token is not eos.

4.4.

In order to carry out a quicker evaluation, the greedy search has to terminate when the stopping criteria are all met for the sequences in the batch.

5. Accuracy computation

5.1.

The function **accuracy** computes the percentage of correct answers. A model prediction is counted as correct, only when the entire output sequence (all characters) matches that of the correct answer; to do so, in this implementation it is computed the number of entries in the predicted sequence that are equal to the entries in the target sequence, if this number is equal to the size of the target sequence then it means that the entire sequence matches the correct answer.

6. Training

6.1.

Since the transformer outputs a vector of probabilities for each digit from 0 to 9 we have to use the cross-entropy loss function.

6.2.

The training loop over the epochs and the batches has been implemented. Furthermore, in order to keep track of losses and accuracies on both the validation and the training datasets, these values are outputted to the user at the end of every epoch.

6.3.

In addition, gradient were accumulated for a couple of steps. Doing this enable us to simulate a large batch size while working with limited GPU memory. In this case the updates are done every 10 batches with a batch size of 64; this results in an effective size of 640.

7. Experiments

7.1.

Using the training pipeline of the previous section, the model was trained on the **numbers__place_value** dataset with the following hyper-parameters:

- learning rate 0.0001
- batch size 64
- accumulating gradient for 10 steps
- effective batch size 640
- 3 encoder layers
- 2 decoder layers
- hidden dimension of 256
- feed-forward dimension of 1024
- 8 attention heads
- Adam optimizer
- gradient clipping with a clipping rate of 0.1

7.2.

Hereafter, two plots are reported. In Figure 1 it is visualized how the training and validation losses behave as the number of epochs increase. Figure 2 shows how training and validation accuracy evolve over epochs.

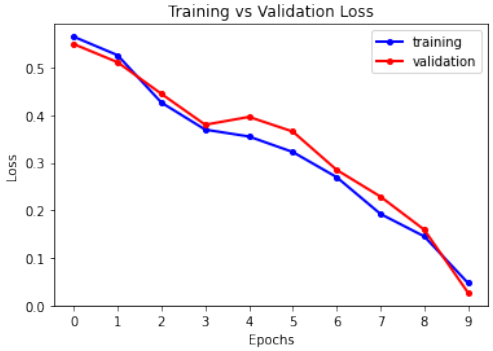

Figure 1: Training and Validation Loss over epochs

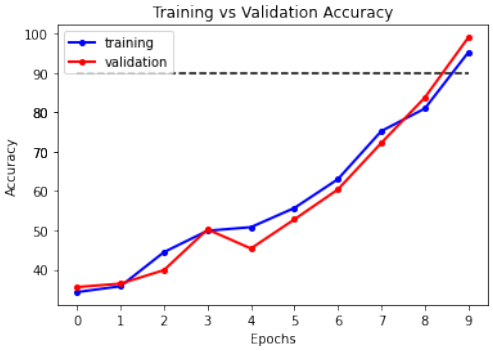

Figure 2: Training and Validation accuracy over epochs

Furthermore, model predictions of 3 example questions from the validation set are reported.

Question 1: What is the ten millions digit of 82446846?
$< pad >< pad >< pad >< pad >< pad >$

Predicted answer:
$< sos > 8$
$< eos >$

Correct answer:
$< sos > 8$
$< eos >$

Question 2: What is the millions digit of 24377448?
$< pad >< pad >< pad >< pad >< pad >$

Predicted anwser:
$< sos > 4$
$< eos >$

Correct answser:
$< sos > 4$
$< eos >$

Question 3: What is the tens digit of 50096231?
$< pad >< pad >< pad >< pad >< pad >$

Predicted answer:
$< sos > 3$
$< eos >$

Correct answer:
$< sos > 3$
$< eos >$

7.3.

Then an analysis on how the accuracy changed when reducing the model size was carried out.

First case: Here both the hidden and the feed-forward dimension were halved

- hidden dimension of 128
- feed-forward dimension of 512

Then, the loss and accuracy curves for this model are reported. In Figure 3 are visualized the training and validation loss while in Figure 4 are visualized the training and validation accuracy.

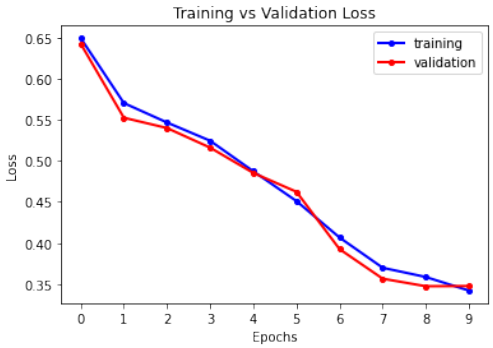

Figure 3: Training and Validation Loss over epochs

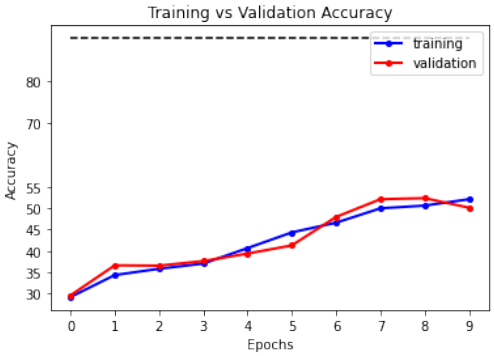

Figure 4: Training and Validation accuracy over epochs

Second case: Here the number of encoder layers was reduced from 3 to 2.

- hidden dimension of 128
- feed-forward dimension of 512

Then, the loss and accuracy curves for this model are reported. In Figure 5 are visualized the training and validation loss while in Figure 6 are visualized the training and validation accuracy.

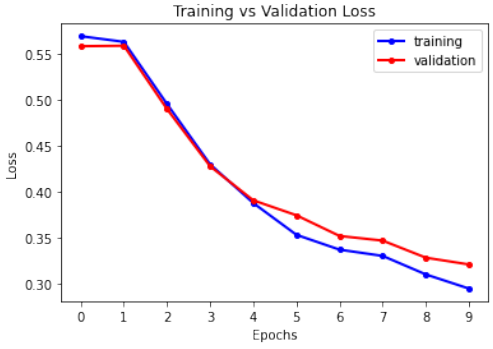

Figure 5: Training and Validation Loss over epochs

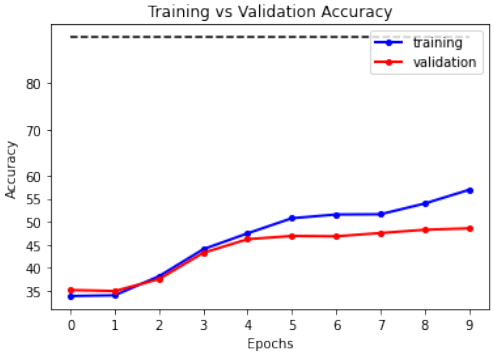

Figure 6: Training and Validation accuracy over epochs

Bottom line is that reducing the size of the model results in a degradation in terms of loss and accuracy.

7.4.

Finally the module **compare_sort** was considered.
This task compared to the previous one is significantly harder. This follows from the fact that the two tasks are fundamentally different in nature. In the former the model has to learn what is the units, the tens,..., the billions digits of the input number; the only difference between the questions in the **numbers__place_value** dataset are the numbers but the structure of the task is always the same. In the latter the model has to learn the relation in terms of magnitude (both decreasing and decreasing, depending what the question asks) between different values; moreover, it has to learn how to put in the correct order fractions and integer. Additionally, in the first we knew our output to be in the range of digits from 0 to 9. On the other hand, in the comparison task there a lot of ways in which our input numbers can be organized (that is the range of output is bigger) but only one correspond to the right sorted sequence (for example with 5 numbers to be sorted, there are 20 possible combinations).

Finally, it is reported the performance in terms of validation and training loss Figure 7 of the **compare_sort** module, note that only the validation vs the training loss plot is reported since the training was taking a lot and the intermediate prediction were not meaningful.

Figure 7: Training and Validation Loss over epochs