Alessandro De Grandi

Deep Learning Lab Assignment 4:

1 Problem Setups and Preliminaries

1.1

The training set consists of 31250 batches and the validation set 157 batches, the batch_size is 64, so the number of sentences is:
Training : 31250*64 = 2000000 (questions and answers)
Validation: 157*64=10048 (questions and answers)

Including the special characters such as <pad>, <sos>, <eos> the sentences are adjusted to be the same size, so:
Training sentence length = 50
Validation sentence length = 48
Answers length = 3

So the total number of characters:
Training : 2000000 *(50+3) = 106000000
Validation : 10048*(48+3) = 512448

2 Dataloader

2.1

No, different vocabularies.

2.2

Once the vocabulary is initialized (extend_vocab=False), unknown is used for tokens that are not already contained in it.

3 Model

3.1

Implemented the suggested model, using:
nn.Embeddings,
the provided PositionalEncoding,
nn.Transformer
nn.Linear

For the masks I ended up using only the `generate_square_subsequent_mask` already provided in the nn.Transformer implementation, as suggested in the slides.

4 Greedy search

4.1

Implemented the greedy decoding algorithm, it allows to process a batch in parallel by passing it to the nn modules, as specified:
The Encoder is forwarded once per batch passing it the source, saving the state in a “memory” variable.
Then the Decoder can be forwarded multiple times, passing it the target (initially it is a <sos> for all sequences) and the “memory”.
Then I used softmax and argmax to produce the output that is concatenated to the current decoder input, and fed as the new input to the decoder in the next decoding step.

4.3

As a stopping criteria I check that the model outputs the <eos> token for all sequences.
While doing the last exercise on compare-sort I modified this criteria to just output the same number of tokens as the length of target sequences.

4.4

As previously stated the algorithm processes a batch in parallel

4.2 (bonus)

5 Accuracy computation

5.1 The most convenient thing was to directly use the greedyDecode function to also compute the accuracy, by checking if the output sequences match the answers, counting the correct answers and dividing by the number of questions, times 100 to obtain a percentage.

I actually used two decoding functions. The first one takes as input the two tensors representing a specific batch(source, target) and performs the decoding and accuracy calculation on that one batch; it was useful during training to check the training accuracy on the last batch.
The second one is more general and takes as input a Dataloader and a max number of batches and performs the operation on a number of batches from the corresponding set, useful for validation accuracy.
Both functions include the option to print the sequences in text format, to check the correctness, also useful during training.

6 Training

6.1

I used nn.CrossEntropyLoss. It combines Softmax and Cross Entropy to calculate the uncertainty of the model efficiently.

6.2

Implemented the training loop, the model and loss targets are shifted as shown in the slides. It keeps track of the accuracy and losses every 500 batches, there is also the option to print some predictions.

6.3

Implemented gradient accumulation, optimizer.step() is called every 10 batches, and the gradients are reset, optimizer.zero_grad() after that.

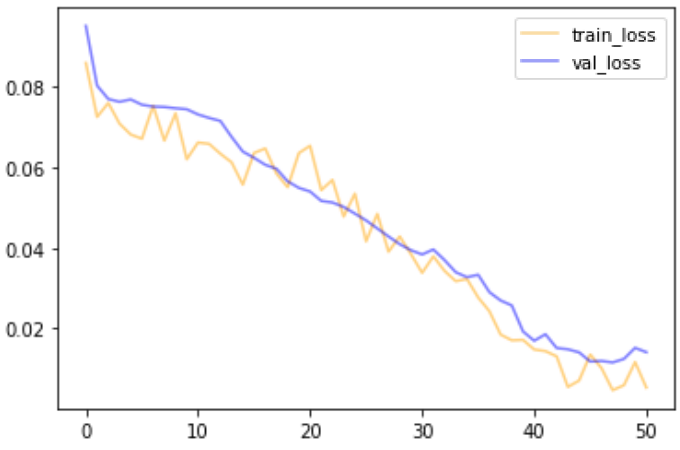
7 Experiments

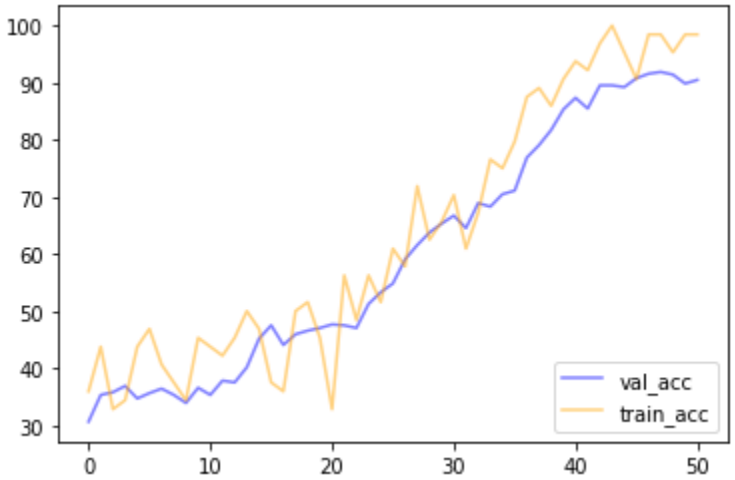
7.1

Trained the specified model with the given hyper-parameters. The model stops training once it reaches over 90% validation accuracy.
(I also used 0.2 dropout rate in the nn.Transformer because the model seemed to overfit the training set very quickly).

7.2

Loss and accuracy curves:





Example output from the validation set:

```
Q:  What is the hundred thousands digit of 22225209?
A:  <sos>2<eos>
P:  <sos>2<eos>

Q:  What is the ten millions digit of 82446846?<pad><pad><pad><pad><pad>
A:  <sos>8<eos>
P:  <sos>8<eos>

Q:  What is the millions digit of
24377448?<pad><pad><pad><pad><pad><pad><pad><pad><pad>
A:  <sos>4<eos>
P:  <sos>4<eos>
```

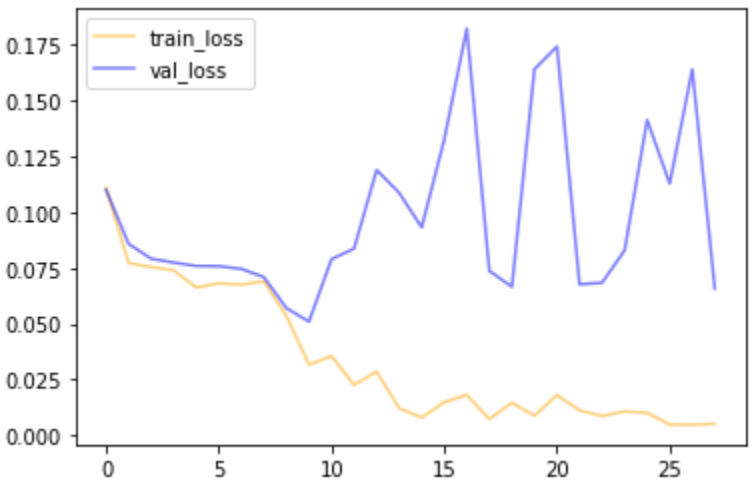
7.2

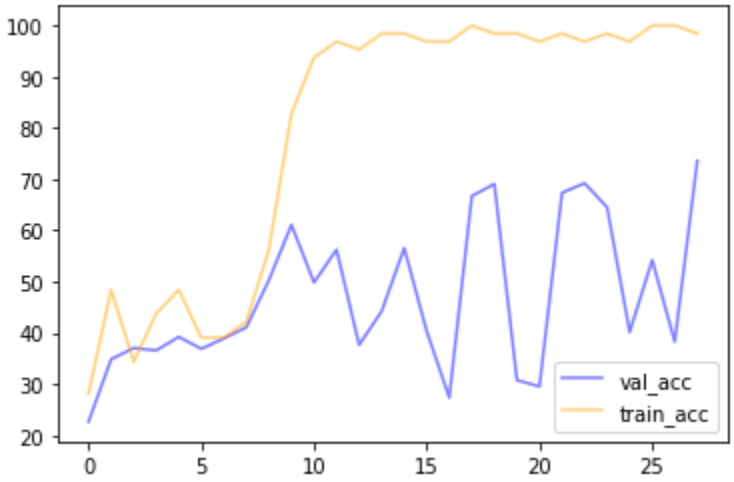
Test #1:

Reduced all the hyperparameters by half, reduced number of encoder and decoder layers by 1:

```
d_model=128
nhead=4
num_encoder_layers=2
num_decoder_layers=1
dim_feedforward=512
dropout=0.1
```

The model overfits the training set and is not able to generalize, as shown by its performance on the validation set that is very unstable:



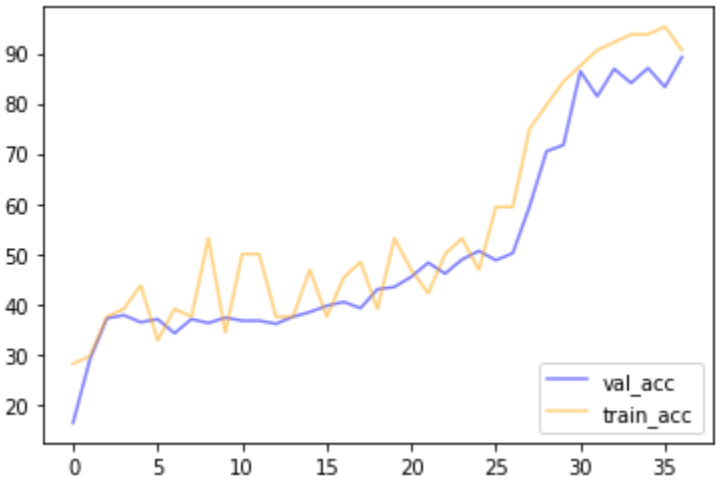


Test #2:

Just reducing d_model and dim_feedforward by half:

```
d_model=128
nhead=8
num_encoder_layers=3
num_decoder_layers=2
dim_feedforward=512
dropout=0.2
```

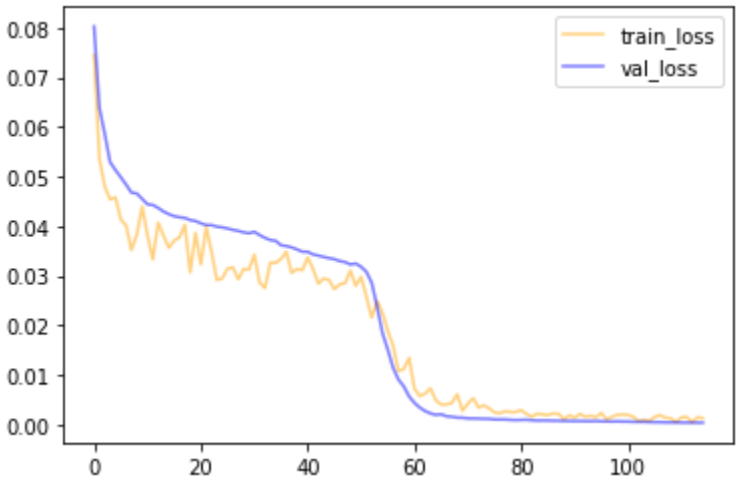
The model is able to reach 90% of accuracy and the training curves seem reasonable.

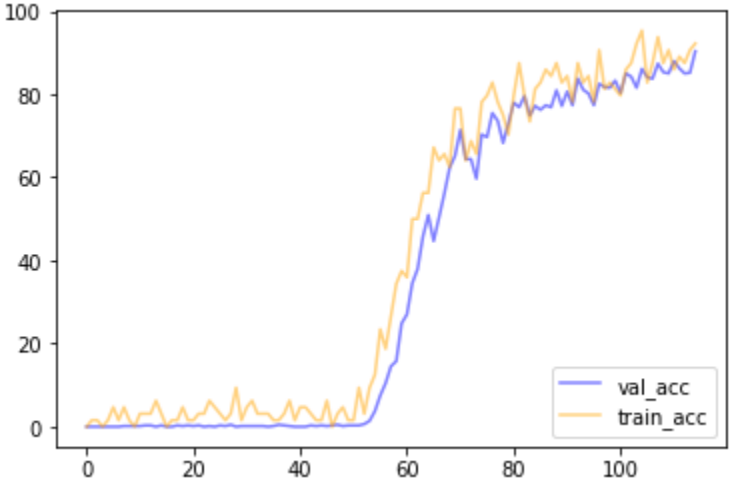


Yes, the dimensions of the model can be reduced obtaining similar performance.
(The model has 1,790,222 trainable parameters, the previously used model had 4,492,814 trainable parameters)

7.4 compare - sort

Trained the model with the compare-sort dataset until it achieved 90% validation accuracy:





Example output from the validation set:

Q: Sort 2, -3, -4, -5, -1339, 3, -140 in decreasing order.<pad>
A: <sos>3, 2, -3, -4, -5, -140, -1339<eos><pad>
P: <sos>3, 2, -3, -4, -5, -140, -1339<eos><pad>

Q: Sort -1, 161, -13, -5, -2, 4, 8 in ascending order.<pad><pad><pad><pad><pad>
A: <sos>-13, -5, -2, -1, 4, 8, 161<eos><pad><pad><pad><pad>
P: <sos>-13, -5, -2, -1, 4, 8, 161<eos><pad><pad><pad><pad>

Q: Sort -15, 1, -4/2343, -3.<pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>...
A: <sos>-15, -3, -4/2343, 1<eos><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>
P: <sos>-15, -3, -4/2343, 1<eos><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>

The task is much harder, it took more than 3 hours of training, the model needs to output sequences of different length, and arrange multiple tokens in the correct way, previously it just had to pick one correct token from the sequence.