**Q1**

INFO: Epoch 099: loss 2.141 | lr 0.0003 | num\_tokens 13.4 | batch\_size 10 | grad\_norm 29.32 | clip 0.999

INFO: Epoch 099: valid\_loss 3.31 | num\_tokens 13.8 | batch\_size 500 | valid\_perplexity 27.3

BLEU = 11.11, 39.9/13.8/7.0/3.9 (BP=1.000, ratio=1.036, hyp\_len=6519, ref\_len=6295)

(A)

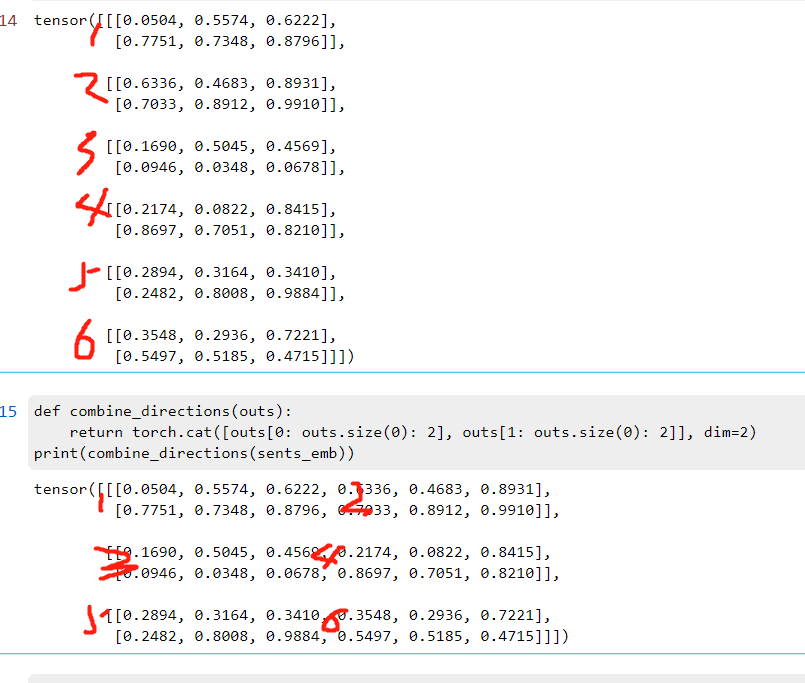
1.

final\_hidden\_states: torch.Size([1, 10, 128])

final\_cell\_states: torch.Size([1, 10, 128])

2.

We define a function *combine\_directions(outs)* that could split the tensor *outs* into two groups based on the parity of the index in tensor’s first dimension, concatenates the two groups of tensors in the 2nd dimension(dim=2). Then applying this function on tensor final\_hidden\_states and final\_cell\_states , transform their size from [2,10,64] into [1,10,128] separately.



3.

The final\_cell\_states is basically the global or aggregate memory of the LSTM network over all time-steps, which should contain information of all words right from the start of the last word (i.e. all the words in the sentence).

The final\_hidden\_states represents the characterization of the last time-step’s data that can illustrate how the specific hidden state is more concerned with the most recent time-step.

(B)

1.

attn\_weights: torch.Size([10, 1, 16])

attn\_context: torch.Size([10, 128])

context\_plus\_hidden: torch.Size([10, 256])

attn\_out: torch.Size([10, 128])

2.

Depending on requirement, we add mask into the attention score.

Apply softmax function to transform the attention score into weight (i.e.the probability that each category is taken based on the last dimension of tensor attn\_scores, i.e., src\_time\_steps). Then, batch matrix-matrix product function (torch.bmm) is applied to calculate the result of attn\_context from the production of previous encoder’s output and the weight maxtrix (attn\_weights)

3.

In case ‘cheating’ from the later input words and make sure that the embedding never rely on the later input words. We need to ensure that the transformer cannot look forward in the input when generating the output.

(C)

1.

projected\_encoder\_out: torch.Size([10, 128, 16])

attn\_scores: torch.Size([10, 1, 16])

2.

src\_projection project the encoder\_out into the specified output\_dim. Through the similarity product between tgt\_input.unsqueeze(dim=1), projected\_encoder\_out get the attention score, the more similar between two vector the higher score will get.

3.

Calculate the similarity between encoder and decoder representations.

(D)

1.

input\_feed: torch.Size([10, 128])

2.

There are two ways of initializing the decoder state. If there is cache of previous state, just load the tgt\_hidden\_states, tgt\_cell\_states, input\_feed from the cache. Otherwise, initializing 0 matrix based on the dimensional requirement.

3.

tgt\_inputs.size()[0], self.hidden\_size: 10, 128

4.

(E)

1.

input\_feed : torch.Size([10, 128])

step\_attn\_weights: torch.Size([10, 10])

attn\_weights: torch.Size([10, 9, 10])

2.

3.

4.

(F)

sample[**'src\_tokens'**], sample[**'src\_lengths'**], sample[**'tgt\_inputs'**] as the input to the seq2seq model and return the output;

using crossEntropyLoss to define the loss function and the input parameters are output with shape \_\_\_\_\_ and target tokens with shape \_\_\_\_\_

Using loss to backword updated the weights

The norm is computed over all gradients together with **clip threshold of gradients = 4**

Performs a single optimization step

Sets the gradients of all optimized :class:`torch.Tensor` s to zero.

**Q2**

1.

English: token count: 124111, word type: 8329

German: token count: 112621, word type: 12505

2.

English: unknown count: 3910, subsequent count: 4420

German: unknown count: 7460, subsequent count: 5046

3.

These tokens are Inflectional and Derivational.

No really. \_\_\_

Subword tokenization or apply transformation to get the root of the word

4.

754

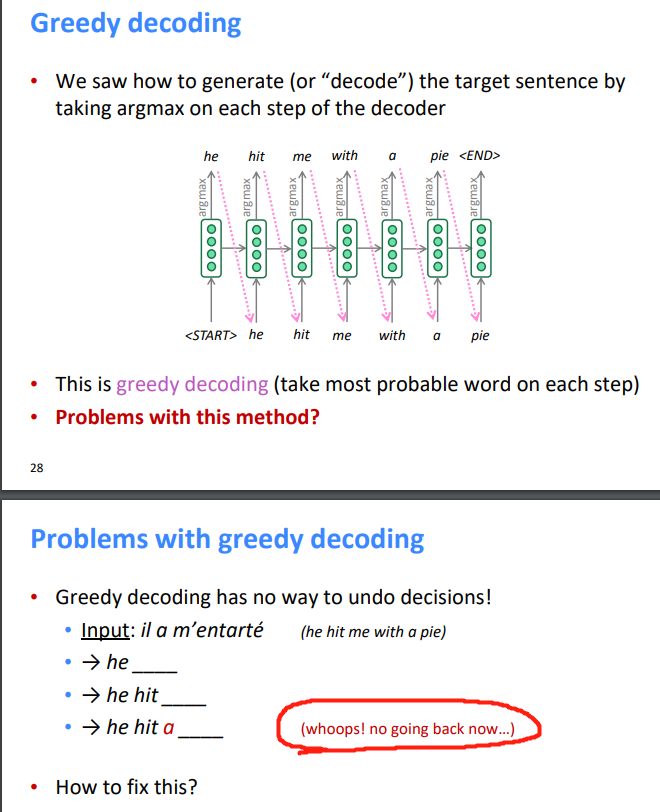
\_\_\_\_

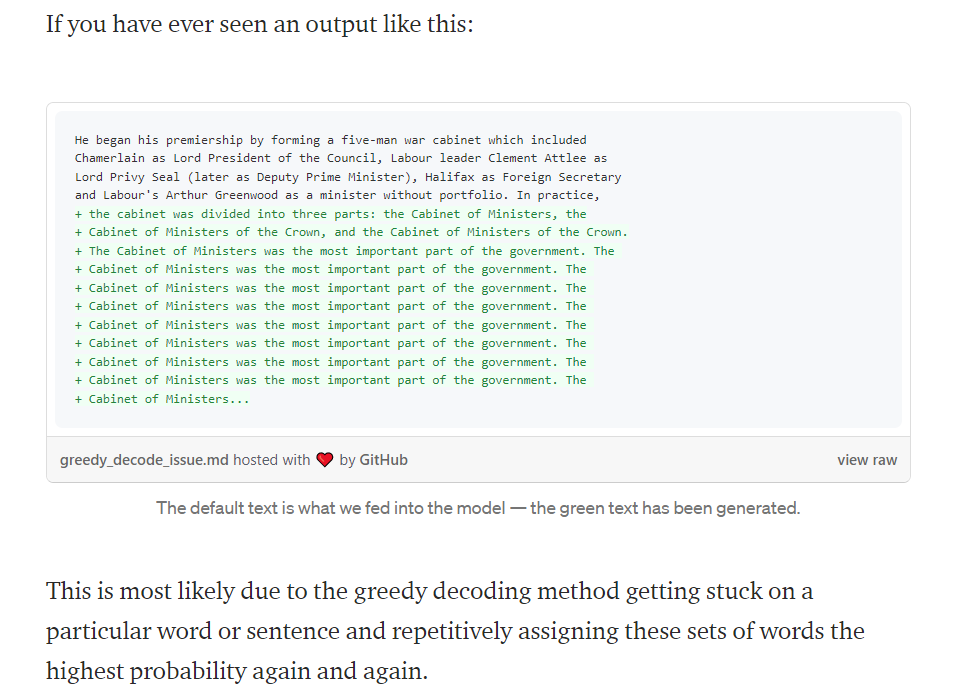
5.

\_\_\_

**Q3**

1.





(many sentences in the same beam may be very similar) and (ii) the decoding level (words are repeated during one iteration of decoding). In the next two sections we

2.

i. current prob, current state = decoder(previous word, previous state) # use the decode to generate the probability of all word at current step

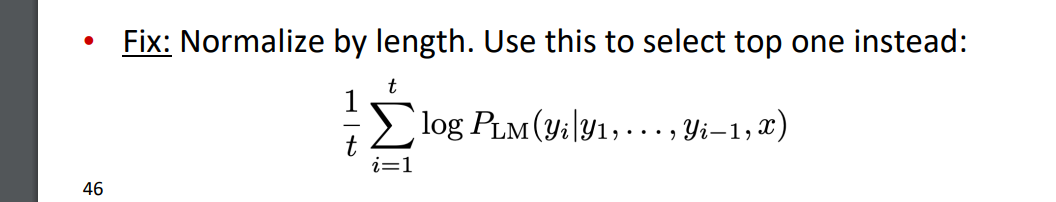
ii. current word list = argmax\_k(current prob) ;score = score(current word list)# find out the k words with largest local probability( Take top k words and compute scores)

iii. for previous word = current word[n]; previous state = current state[n] # prepare for the next step decoding for all k candicates (For each of the k hypotheses, find top k next words and calculate scores)

iv. Of these k^2 hypotheses, just keep k with highest scores

v. When a hypothesis produces <END>, that hypothesis is complete.

3.



**Q4**

1.

python train.py --encoder-num-layers 2 --decoder-num-layers 3

INFO: Epoch 089: loss 2.412 | lr 0.0003 | num\_tokens 13.4 | batch\_size 10 | grad\_norm 22.98 | clip 0.999

INFO: Epoch 089: valid\_loss 3.41 | num\_tokens 13.8 | batch\_size 500 | valid\_perplexity 30.2

2.

(1)

Training Loss&valid perplexity:

Old:

INFO: Epoch 099: loss 2.141 | lr 0.0003 | num\_tokens 13.4 | batch\_size 10 | grad\_norm 29.32 | clip 0.999

INFO: Epoch 099: valid\_loss 3.31 | num\_tokens 13.8 | batch\_size 500 | valid\_perplexity 27.3

New:

INFO: Epoch 089: loss 2.412 | lr 0.0003 | num\_tokens 13.4 | batch\_size 10 | grad\_norm 22.98 | clip 0.999

INFO: Epoch 089: valid\_loss 3.41 | num\_tokens 13.8 | batch\_size 500 | valid\_perplexity 30.2

BLEU

Old: BLEU = 11.11, 39.9/13.8/7.0/3.9 (BP=1.000, ratio=1.036, hyp\_len=6519, ref\_len=6295)

New: BLEU = 9.37, 39.2/12.7/5.8/2.9 (BP=0.975, ratio=0.976, hyp\_len=6142, ref\_len=6295)

(2)

(3)

(4)

**Q5**

INFO: Epoch 055: loss 1.836 | lr 0.0003 | num\_tokens 13.4 | batch\_size 10 | grad\_norm 29.83 | clip 1

INFO: Epoch 055: valid\_loss 3.18 | num\_tokens 13.8 | batch\_size 500 | valid\_perplexity 24.1

BLEU = 13.06, 45.5/17.6/8.5/4.5 (BP=0.986, ratio=0.986, hyp\_len=6209, ref\_len=6295)

Examples:

**Q6**

(A)

1.

Embeddings：torch.Size([10, 8, 128])

(B)

1.

Attn： torch.Size([11, 11])

(C)

1.

forward\_state： torch.Size([10, 15, 4420])

(D)

1.

State： torch.Size([8, 10, 128])

(E)

1.

State： torch.Size([11, 10, 128])

Attn： torch.Size([20, 11, 10])

**Q7**

INFO: Epoch 021: loss 1.356 | lr 0.0003 | num\_tokens 13.4 | batch\_size 10 | grad\_norm 51.49 | clip 0.999

INFO: Epoch 021: valid\_loss 3.74 | num\_tokens 13.8 | batch\_size 500 | valid\_perplexity 42.3

BLEU = 11.98, 42.9/16.3/8.0/4.1 (BP=0.972, ratio=0.972, hyp\_len=6120, ref\_len=6295)

**REFERENCE**

<https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2#:~:text=The%20cell%20state%20is%20meant,the%20previous%20time%2Dstep's%20data>.

