School of Informatics

NLU
CW2

s2172485 s2153223
April 2022

Date: Friday 1st April, 2022

1 Question 1

Criteria	value
BLEU score	10.91
validation-set perplexity	27
final training loss	2.142

Table 1: Report for Baseline NMT model

1.1 Comment A

1. # final_hidden_states.size = [num_layers, batch_size, 2*hidden_size]
   # final_cell_states.size = [num_layers, batch_size, 2*hidden_size]

2. When self.bidirectional is set to True, the sequence is fed to the lstm encoder in forward(left to right) and backward(right to left) order. Then, the forward hiden states and the backward hidden states are concatenated to be the final hidden states:

   final_hidden_state = [hidden_states(forward); hidden_states(backward)].

   The original final hidden states of shape (2*num_layers, batch_size, hidden_size) are converted into the shape of (num_layers, batch_size, 2*hidden_size) after going through the bidirectional lstm. Similarly, the final cell states are also spliced by forward cell states and backward cell states, and finally converted to the shape of (num_layers, batch_size, 2*hidden_size).

3. The differences between final_hidden_states and final_cell_states:

   The hidden states of each time step store the memory of the current time step, and each hidden state will be updated as the time step changes, that is, $h^{(t)}$ and $h^{(t-1)}$ may be very different. However, the cell states of each time step store the memory of all past time steps: $c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \hat{c}^{(t)}$, which means that $c^{(t)}$ is the $c^{(t-1)}$ passed from the previous state and add some value. Therefore, $c^{(t)}$ and $c^{(t-1)}$ do not change much, which provides long-term memory in LSTM.

1.2 Comment B

1. # scr_mask.size = [batch_size, 1, src_time_steps]
   # attn_weights.size = [batch_size, 1, src_time_steps]
   # attn_context.size = [batch_size, output_dims]
   # context_plus_hidden.size = [batch_size, 2*output_dims]
   # attn_out.size = [batch_size, output_dims]

2. After going through the self.score function, the attention scores are obtained. When the source mask exists, we can apply the mask to attention scores, so that the padding parts are assigned as float('-inf'). After that, the attention scores go through a softmax function, and the scores are distributed between 0 and 1 as attention weights. Finally, we perform batch matrix-matrix product on the attention weights and encoder output and obtain the attention context. The dimension of size 1 is removed after the product, resulting in size of [batch_size, output_dims].

3. The purpose of applying a mask is to force the attention weights of the padding part to become 0. Since the padding parts have nothing to do with the meaning of the sequences, we need to keep their attention weights to be 0. Therefore, before going through softmax function, the attention scores of the padding part should be set to float('-inf'). Mask makes padding not affect the meaning of the sequences, and makes the model perform efficient operation in batches after padding.

1.3 Comment C

1. # projected_encoder_out.size = [batch_size, output_dims, src_time_steps]
   # attn_scores.size = [batch_size, 1, src_time_steps]

2. Attention scores are obtained by performing batch matrix-matrix product from the transformed target input matrix and the transformed encoder output matrix.
   The scoring function is described in Luong *et al.* [1]: $\text{score}(h_t, \bar{h}_s) = h_t^\top W_a \bar{h}_s$.
   First, we apply a linear transformation in self.src_projection function to encoder_out, converting its dimension from input_dims to output_dims (input_dims is equal to output_dims in this coursework). The dimensions dim1 and dim2 are swapped after the linear transformation, and we get projected_encoder_out of size [batch_size, output_dims, src_time_steps]. Then, we insert a tensor of size one into dim1 of tgt_input. The size of new tgt_input matrix is [batch_size, 1, input_dims]. Finally, we performs a batch matrix-matrix product of the new tgt_input matrix and projected_encoder_out matrix. The size of the score tensor is [batch_size, 1, src_time_steps].

3. For each batch, the dot product can be performed on two-dimensional unsqueezed tgt_input matrix(size=[1, input_dims]) and two-dimensional projected_encoder_out matrix (size= [output_dims, src_time_steps]) to obtain the attention score matrix (size=[1, src_time_steps]). In this process, encoder and decoder representations are aligned.
   Batch matrix multiplication elevates the dot product to a three-dimensional operation, implementing the matrix dot product for all batches at one time by inserting batch-sized tensors into dim0 of all two-dimensional tensors. Therefore, even if the tensor is three-dimensional, the representations of the encoder and decoder are still aligned.

1.4 Comment D

1. # tgt_inputs.size = [batch_size, tgt_time_steps]
   # tgt_hidden_states[0].size = [batch_size, hidden_size]
   # tgt_hidden_states is a list of size num_layers. Each tensor in it has a size [batch_size, hidden_size]
   # tgt_cell_states[0].size = [batch_size, hidden_size]
   # tgt_cell_states is a list of size num_layers. Each tensor in it has a size [batch_size, hidden_size]
   # input_feed.size = [batch_size, hidden_size]

2. The variables in the decoder are tgt_hidden_states, tgt_cell_states and input_feed. When cached_state does not exist, the variables are initialized as zero tensors of size of [batch_size, hidden_size].

3. cached_state == None indicates incremental_state is None which means the decoder has no cached state and is in training state. During the training procedure, the model could

obtain the whole sentence. However, when translating sentences, the model may obtain the source word one by one, which is a incremental mode. At the same time of receiving input words, the model needs to output the translation, so the previous state needs to be cached in order to provide information for later words. Hence, when the model could obtain the whole sentence for one time, incremental_state is None and thus cached_state==None.

4. When the model includes Attention layer, the input_feed is the output of the Attention layer. Otherwise, input_feed is the hidden state of the previous time step of the decoder. After that, input_feed will provide information to the decoder at the current time step to participate in the input.

1.5 Comment E

1. # input_feed.size = [batch_size, hidden_size]
   # step_attn_weights.size = [batch_size, src_time_steps]
   # attn_weights[:, j, :].size = [batch_size, src_time_steps]
   # input_feed.size = [batch_size, hidden_size]

2. In the Attention layer, we use the encoder outputs and target hidden state to calculate the attention scores, attention weights and attention context in turn, and finally get the attention output. The attention output is the input_feed of the decoder, which can be used as the input of the decoder at the next time step.

3. The attention function is given the last target state as one of its inputs to provide context information. It is used to calculate the attention scores together with encoder outputs in all time steps.

4. The purpose of the dropout layer is to prevent the model from overfitting and provide more robustness to the input_feed. input_feed will randomly drop some of the units after the dropout operation, and preventing overfitting.

1.6 Comment F

1. # output.size = [batch_size, tgt_time_step, target_dictionary_length]
   # _.size = [batch_size, tgt_time_step, src_time_step]

2. Description:
   line1: Input the tokens, sequence lengths of the source language and the inputs of the target language into the model, and obtain the output of the model.
   line2: Use the cross-entropy function as the loss function and calculate the loss.
   line3: Calculate the gradient of the loss by backpropagation.
   line4: Gradient clipping is performed to prevent exploding gradients.
   line5: Update the parameters of the model.
   line6: Reset the gradients for the next batch.

2 Question 2

1. Suppose the end of sentence is not considered.

   Total count of words in the English training set: 124031

Total count of words in the German training set: 112572

The number of word types in the English training set: 8326

The number of word types in the German training set: 12504

2. 3909 word tokens will be replaced by <UNK>in English. 7460 word tokens will be replaced by <UNK>in German. The total vocabulary size in English will be 4418 (8326-3909+1). The total vocabulary size in German will be 5045 (12504-7460+1).

3. By looking at the sorted replaced words, we have the following discoveries.

   - A proportion of the words are rare, such as *rossa*, *kuhne*, *federalism* and so on.
   - Some words are inflectional morphology of a same word. For example, *identify* and *identifying*, *employer* and *employers*, etc. They are not really the rare words.
   - Some words are other words attached with punctuation, such as *everything'*, where the word *everything* itself is a common word and has appeared more than twenty times in the training data.

   Suggestions to improve the tokenization:

   - Consider tokens as subword units [2] so that (a) the suffix such as *ing*, *ed*, *s* will not cause a common word being classified as a rare word by the model, and (b) the tokens are handled in a more general way and more information could be captured. For instance, the model will be able to extract the information that words *apple* and *apples* indicate the same item with different quantity.
   - Split punctuation marks that at the edge of the words out so that the words *everything'* could be considered as the word *everything* and the punctuation mark '.

4. 478 tokens are shared by English and German. Due to this similarity, some specific types of rare words could be translated directly.

   - There is a proportion of rare words that have appeared only once are digits, and the meaning of digits are universal across the world.
   - Punctuation marks like '(', ')', '...' can have the similar meaning in most languages. However, some punctuation marks may be part of the language, such as ' and -.
   - Specific names of entities could be copied directly provided the source language and the target language share an alphabet [2], such as people's names, places, etc. However, more processes are required to recognize the names.

5. **Sentence length.** By looking at the numbers above, the total count of English words are greater than the total count of German words. It could be inferred that English sentences are generally longer than German sentences. However, the number of word types in English training set is smaller than the number of word types in German training set, and the total vocabulary size in English is smaller than that in German as well. Hence, it could be inferred that German is more concise but more complex in the vocabulary level, and one German word may correspond to a combination of several English words. Thus, word-to-word translation strategy is infeasible.

   Tokenization process. Tokenization process determines the input and output of the model. When tokenization is performed in more detail, the input sentence will become longer and the dependency between tokens becomes longer as well, and vice versa. In

addition, tokenization greatly influences the quality of machine translation [3]. When words are split into subwords and punctuation is separated, the vocabulary is reduced and the number of examples of each word increases [3]. Some authors regard combinations of some words as a single token, so that the phrase *a furniture shop* will be recognized as a shop rather than a furniture [4]. It changes what the model would learn.

Unknown words. As the unknown words are not recorded as tokens, they will not be part of the input of the model. The model will not learn the unknown words. When the model encounters the unknown words in future translation tasks, it cannot produce the corresponding translation of the unknown words. A common idea to handle the unknown words is to copy the unknown words directly to the translation output [2] [5]. When the source sentence contains a large number of rare words, the translation is likely to fail [6].

The following code is used to figure out the sub-question 1, 2, and 3 of Question 2.

```
def q2_123(lang):

    wordlist = []
    wordcount = {}
    wordtypes = set()

    filename = 'europarl_raw/train.' + lang

    with open(filename, 'r') as inf:
        for line in inf:
            for word in line.strip('\n').split():
                wordtypes.add(word)
                wordlist.append(word)
                if word in wordcount:
                    wordcount[word] += 1
                else:
                    wordcount[word] = 1
    inf.close()

    print('words of', lang, 'training set:', len(wordlist))
    print('word types of', lang, 'training set:', len(wordtypes))

    word_unk = []
    for word in wordcount:
        if wordcount[word] == 1:
            word_unk.append(word)

    print('words appear once:', len(word_unk))

    word_unk.sort()

    with open('q2.txt', 'w') as q2_file:
        for word in word_unk:
            q2_file.write(word + '\n')
```

```
    q2_file.close()

lang = 'en'
q2_123(lang)

lang = 'de'
q2_123(lang)
```

The following code is used to answer sub-question 4 of Question 2.

```
def q2_4(fn1, fn2):

    with open(fn1, 'r') as f1:
        tokens1 = [line.split()[0] for line in f1]
    f1.close()

    with open(fn2, 'r') as f2:
        tokens2 = [line.split()[0] for line in f2]
    f2.close()

    tokens1.sort()
    tokens2.sort()

    intersect = set(tokens1).intersection(set(tokens2))

    print('intersection:', intersect)
    print('size of the intersection set:', len(intersect))

fn1 = 'europarl_prepared/dict.en'
fn2 = 'europarl_prepared/dict.de'
q2_4(fn1, fn2)
```

3 Question 3

1. By always choosing the token with the maximum probability, the model can ignore global information [7]. For example, in the sentence *'eating toys is dangerous'*, the probability that *toys* follows *eating* could be very little since the phrase *eating toys* rarely appears in daily life. In machine translation tasks, given the source sentence *'I will miss my plane'*, as the model only focuses on the probability of the third word and the previous word, supposing the former translation is more common, it may translate the word *miss* to a word with maps to the meaning *'fell or suffer from the lack of'* instead of *'a failure to hit'*. In such scenario, the maximum probability may lead to an incorrect answer.

2. The pseudocode of the LSTM with beam search and greedy decoder is shown in Algorithm 1.

3. One strategy is to apply length normalization [8]. Originally, the sentence with the highest probability is chosen in beam search. The idea of length normalization is to divide the score of the sentence by its length. Another approach is to add an adjustable reward to

Algorithm 1 LSTM with Beam Search and Greedy Decoder

Require: N: beam width; *source sentence*: the sentence which is going to be translated by the model

Ensure: $final\_sequence$: the final translated sentence

function LSTM_BeamSearch(N, *source sentence*)
    $previous\_sequence \leftarrow < START >$ # the first word
    $previous\_state \leftarrow$ InitialState # initialize the state
    while *source sentence* has the next word **do**
        for $i \leftarrow 0$ *to* N **do** # traverse the beam width
            # previous_sequence[i] contains the i-th best sequence so far, and current_prob[i]
            # contains the probability that each word attached to the previous_sentence[i].
            $current\_prob[i], current\_state[i] =$ Decoder($previous\_sequence[i], previous\_state[i]$)
            # Suppose previous_sequence[i] = AB, then current_prob[i] contains the probabilities
            # of ABA, ABB, ABC, ...
        end for
        for $i \leftarrow 0$ *to* N **do** # traverse the beam width
            # find the sequences correspond to the highest N probabilities in current_prob
            $current\_sequence[i] =$ argmax($current\_prob$)
            # prepare for the next step decoding
            $previous\_sequence[i] = current\_sequence[i]$
            $previous\_state[i] = current\_state[i]$
        end for
    end while
    # find the sequence that corresponds to the highest probability in current_prob
    $final\_sequence =$ argmax($current\_prob$)
    return $final\_sequence$
end function

each output word so that long sentences are applied with higher reward in accumulation [9].

4 Question 4

1. To train the 2encoder-3decoder model, we use the following command:

```
python train.py --save-dir question4 --log-file question4/log.out \\
--data europarl_prepared --encoder-num-layers 2 --decoder-num-layers 3
```

2. After changing the number of layers in the encoder = 2, decoder = 3, the performance of the model gets worse. Compared with the baseline model, after adding layers, both the training loss and the validation-set perplexity increase, and the BLEU score also decreases. Table 2 shows the performance of Baseline model and 2encoder-3decoder model on training, validation and test sets.

NMT Model	BLEU score	validation-set perplexity	final training loss
Baseline	10.91	27	2.142
Model(2encoder-3decoder)	9.97	29	2.326

Table 2: Report for the Baseline model and the 2encoder-3decoder model

Figure 1 shows the loss curves of baseline model and 2encoder-3decoder model on the training set and validation set. Both models perform well on the training set, but worse on the validation set, with slow loss reduction after 40 epochs. Therefore, there is a difference in performance on the training set and on the validation set which illustrates the over-fitting problem in both models. However, we can see from the figure that the training loss and validation loss of the 2encoder-3decoder model are higher than those of the baseline model. This shows that adding the layers of the model will exacerbate the problem of over-fitting, so the performances of the 2encoder-3decoder model on the training, validation and test sets are worse.

5 Question 5

In this section, we implemented the Lexical model according to Section4 of Nguyen and Chiang (2017) [10]. Table 3 shows the performance of the Baseline model and the Lexical model on the traininig, validation and test sets. Compared to the Baseline model, the final traininig loss and validation-set perplexity of the lexical model are lower, and the BLEU score is higher, which means that the Lexical model has better performance at all stages.

NMT Model	BLEU score	validation-set perplexity	final training loss
Baseline	10.91	27	2.142
Lexical Model	12.68	24.1	1.836

Table 3: Report for the Baseline model and the Lexical model

The performance of the Lexical model is consistent with Nguyen and Chiang's [10] conclusion that adding lexical translation will improve the performance of the model and enhance the

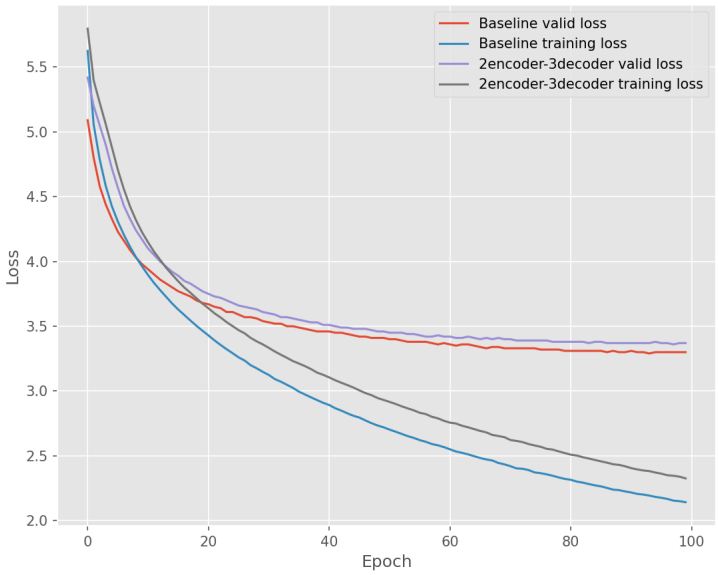

Figure 1: Loss of the Baseline model and the 2encoder-3decoder model on training and validation set

translation ability of rare words. Since the initial network pays more attention to the context information, the model prefers to generate sequences which fit the context, i.e. fluent and common. As the model has little knowledge about the context of rare words, it 'ignores' the rare words when trying to generate a fluent sequence. The lexical layers solve the problem by directly performing on source embeddings, and thus the connection between source words and target words is established. It brings more information of source words, being incorporated to the rest of the model to reduce the influence of the context.

Table 4 shows some example translations of the baseline model and the lexical model in line 20, line 204 and line 432. It can be found that when encountering an rare words in source sentence, the baseline tends to generate sequence formed by fluent and common phrases and nearly 'ignores' the rare words, while the lexical model copies the rare words to the output sequence. It fits the idea that when the NMT model finds an unknown word, it is a common option to directly copy it to the translation output [2] [5]. For rare words such as *randzio*, *2008* and *watson*, the lexical model can translate them correctly, while the baseline model cannot. This also illustrates that the lexical model significantly improves the translation quality of low-resource language pairs.

Code implementation:

```
# __QUESTION-5: Add parts of decoder architecture corresponding to the LEXICAL
# MODEL here
# TODO: ---------------------------------------------------------------------- CUT
self.lexical_hidden = nn.Linear(embed_dim, embed_dim, bias=False)
self.lexical_out = nn.Linear(embed_dim, len(dictionary))
# TODO: ---------------------------------------------------------------------- /CUT

# __QUESTION-5: Compute and collect LEXICAL MODEL context vectors here
# TODO: ---------------------------------------------------------------------- CUT
f_t = torch.tanh(torch.bmm(step_attn_weights.unsqueeze(dim=1),
```

line20	Source	herr präsident , mir geht es um den wortlaut bei einem bericht von frau **randzio**-plath .
	Reference	mr president , this is the text relating to the **randzio** - plath report .
	Baseline	mr president , i am going to make a report on the report of mrs energy report.
	Lexical	mr president , i am talking about the report of mrs **randzio** -de-de-de-de).
line204	Source	für **2008** wird eine steigerung auf 7 596 us-dollar vorausgesagt .
	Reference	it is forecast to increase to usd 7 596 in **2008** .
	Baseline	for the end , it will be a drop to the criminal proceedings .
	Lexical	for **2008** , the budget is spent to 7 7 to 7 .
line432	Source	ich verstehe daher nicht , was herr **watson** , der vorsitzende der alde-fraktion eben gesagt hat .
	Reference	thus , i am at a loss to understand what mr **watson** , the chairman of the group of the alliance of liberals and democrats for europe , has just said .
	Baseline	i therefore do not say what mr poettering has said by the rapporteur .
	Lexical	i am not therefore , mr **watson** , to be seen by mr watson , the eu has said .

Table 4: Example translations of the Baseline model and the Lexical model. **bold** indicate the rare word.

```
src_embeddings.transpose(0, 1)))
h_t = torch.tanh(self.lexical_hidden(f_t)) + f_t # ht = tanh(Wft) + ft
lexical_contexts.append(h_t)
# TODO: ---------------------------------------------------------------------- /CUT

# __QUESTION-5: Incorporate the LEXICAL MODEL into the prediction of target
# tokens here
# TODO: ---------------------------------------------------------------------- CUT
lex_out = torch.cat(lexical_contexts, dim=1)
decoder_output += self.lexical_out(lex_out)
# TODO: ---------------------------------------------------------------------- /CUT
```

6 Question 6

6.1 Comment A

1. # embeddings.size = [batch_size, src_time_steps, num_features]
2. Positional embeddings aims to add positional information to the encoder and decoder, such as the position that the token is in the sequence, or the order of tokens.
3. The embeddings in the transformer model requires position embeddings because the transformer needs the information of the sequence order. [11]. The information of sequence order influences the meaning of a sentence, e.g. the sentences *Lucy loves the cat* and *The cat loves Lucy* have exactly different meanings. As there is no recurrence or convolution

layer in the model, no information of sequence order is recorded, and thus the position embeddings are added. LSTM is based on recurrent neural network, where the order information is recorded in each time step, so that position embeddings are not needed.

6.2 Comment B

1. # self_attn_mask.size = [tgt_time_steps, tgt_time_steps]
2. self_attn_mask is a square matrix where the upper triangular part of the matrix is -inf while the other part (i.e. the bottom triangular part) is 0. The mask will be combined with the output embeddings so that for each position i, the output information after i is masked. It ensures that the model at each position i could only use the information of the outputs that are in former of the position i, and thus the prediction of the word at position i only depends on the prediction of words before it [11].
3. Masking happens in the decoder rather than the encoder because we only need to mask queries, and the unmasked output of the encoder will also be used as keys and values to calculate multi-head attention with queries in the decoder. Therefore, we receive the unmasked output of the encoder to calculate keys and values, and mask the output of the encoder to calculate queries.
4. In incremental decoding, each time the decoder receives the output embeddings of a new word as input. The model only has one query, and the keys and values all come from the prevoius hidden states. Thus, without masking, the decoder still never uses the outputs of the later words.

6.3 Comment C

1. # forward_state.size = [batch_size, tgt_time_steps, len(dictionary)]
2. The linear projection after the decoder layers performs a classification task, where the number of classes is the size of the vocabulary. It classifies each embedding to the most possible word.
3. At the end of the encoder we do not need to turn embeddings to words since the decoder still needs to use the embeddings as keys and values. Hence no transformation between embeddings and words at the end of the encoder or the beginning of the decoder. In the beginning of the encoder, we transform words to embeddings through an embedding layer by calling nn.Embeddings. The embedding layer also performs a kind of linear projection which turns the vectors of vocabulary size to vectors of hidden size.
4. By setting feature_out=True, the words are still represented by embeddings which contains features of each word. The output represents the features of the sentence.

6.4 Comment D

1. # query.size = [src_time_steps, batch_size, num_features]

   # key.size = [src_time_steps, batch_size, num_features]

   # value.size = [src_time_steps, batch_size, num_features]

key_padding_mask.size = [batch_size, src_time_steps]

(output) state.size = [tgt_time_steps, batch_size, num_features]

_.size = [num_heads, batch_size, tgt_time_steps, src_time_steps]

2. For training and calculation, sentences in a batch needs to have the same length, and thus some paddings are injected when the input sentence is transformed to embeddings. We do not want to train the model to handle the paddings so that the paddings are masked out.

3. The output shape of 'state' Tensor will transform from [src_time_steps, batch_size, num_features] to [tgt_time_steps, batch_size, num_features] after multi-head attention.

6.5 Commend E

1. # query.size = [tgt_time_steps, batch_size, num_features]

   # key.size = [src_time_steps, batch_size, num_features]

   # value.size = [src_time_steps, batch_size, num_features]

   # key_padding_mask = None

   # (output) state.size = [tgt_time_steps, batch_size, embed_dim]

   # (output) attn.size = [num_heads, batch_size, tgt_time_steps, src_time_steps]

2. Self attention performs on queries. After the self attention of queries is calculated, the encoder attention layer receives it as input queries, receives the encoder output to calculate keys and values, and uses all of them to calculate the multi-head attention.

3. key_padding_mask appears in the encoder of the transformer. The purpose of using padding mask is to let the model ignore the padding tokens. attn_mask appears in the decoder of the transformer. When processing token at position i, it adds masks to the tokens after position i to make sure that the decoder only uses the outputs of the words in former of the current one.

4. The calculation of encoder attention does not require attn_mask because (1) queries have already been masked in the self-attention layer (i.e. self.self_attn), and (2) the layer receives the output of the encoder to calculate keys and values, where keys and values will not be masked. They provide the global information (information of all tokens) to calculate multi-head attention with the masked queries.

7 Question 7

NMT Model	BLEU score	validation-set perplexity	final training loss
Baseline	10.91	27	2.142
Lexical Model	12.68	24.1	1.836
Transformer	11.03	43.7	1.364

Table 5: Report for the LSTM-based model and the Transformer model

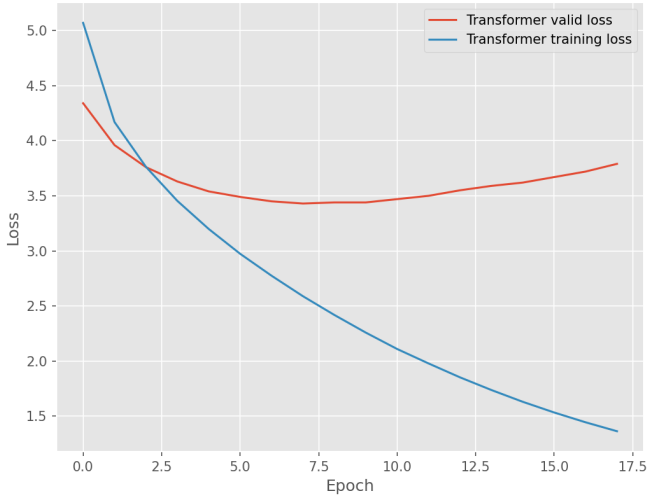

Figure 2: Loss of the Transformer model on training and validation set

According to Table 5, Transformer shows higher validation perplexity and lower training loss than LSTM-based models. The BLEU score of the Transformer model is higher than the LSTM baseline model but lower than the LSTM lexical model. Hence, the transformer has better training performance and worse validation performance than LSTM-based models, and the test performance of the transformer is better than the LSTM baseline model but worse than the LSTM lexical model. According to the BLEU scores, it could be concluded that the transformer model is generally better than LSTM baseline model but worse than the LSTM lexical model. By checking the translation results, it can be found that there are many cases the transformer model fails to translate some words when the LSTM lexical model succeeds (Table 6).

As it has a low training loss besides the high validation perplexity, it is very likely that overfitting occurs during the training process of the transformer model. To confirm it, curves of training loss and validation loss across each epoch are draw in Figure 2. According to Figure 2, the loss of the training set keeps decreasing, while the loss of the validation set first decreases and then increases significantly, which confirms the over-fitting problem in the Transformer model. In addition, we found that the over-fitting problem is exacerbated when using the same training data compared to the LSTM-based model. We explore the dataset and the model structure, concluding two reasons that cause overfitting happening.

Insufficient data. After exploring the dataset, we found that there are only 10,000 sentences contained in the training set, with 12,504 and 8,326 word types in German and English respectively. And the average sentence lengths for German and English in the training set are 11.3

and 12.4 words. The data is not enough for the model to be well trained, thus makes all models more or less over-fit [12].

Complex structure. Transformer is more complex, and complex models are more prone to get overfitted. Multi-head attention leads to more connections between layers in the transformer model than other two models, which means more decision boundaries will be learned by the model. For each token, the LSTM model learns limited number of cells, whereas the transformer learns a number of attentions especially when the sentence is long. It makes the transformer model more complex and brings it more powerful learning ability, while making it easier to overfit as well [13].

Following ways may mitigate the overfitting problem and improve the performance of the model.

1. Train on a large dataset so that the model could generalize better [12].
2. Data augmentation is a choice when the available data is limited. For instance, [14] proposed a data augmentation approach, which randomly masks some input data. Some researchers try the data augmentation method in [14] on their transformers and achieve good results [15].
3. Apply with L2 Regularization, which is to add a term to the cost function and restrict the weights from being too large so that the model will not become too complex [12].
4. Save the best model during training. It means saving the model with the lowest validation perplexity. It is noticed that the lowest validation perplexity of the transformer model among all epochs is 30.9, much less than the final validation perplexity 43.7. After this epoch, the validation perplexity starts to grow and overfitting happens. Hence, the model at the epoch when validation perplexity does not show an increase is expected to have better performance.

Code Implementation:

```
# TODO: REPLACE THESE LINES WITH YOUR IMPLEMENTATION ----------------------- CUT

b, t, h, k = batch_size, tgt_time_steps, self.num_heads, self.head_embed_size

# [t/s, b, embed_dim] where s = key.size()[0]
query_i = self.q_proj(query)
key_i = self.k_proj(key)
value_i = self.v_proj(value)

# [t/s, b, h, k]
query_i = query_i.contiguous().view(-1, b, h, k)
key_i = key_i.contiguous().view(-1, b, h, k)
value_i = value_i.contiguous().view(-1, b, h, k)

# [h, b, t/s, k]
query_i = query_i.transpose(0, 2)
key_i = key_i.transpose(0, 2)
value_i = value_i.transpose(0, 2)

# [h*b, t/s, k]
```

line44	Source	so etwas kann mitunter erfolgreich sein , aber auch manchmal fehlschlagen .
	Reference	such things can sometimes be **successful** or **sometimes** fail .
	Baseline	this can be a hope , but also , but also , but it is enough .
	Lexical	so can be only need to be **successful** , but **sometimes** sometimes sometimes sometimes sometimes sometimes .
	Transformer	there can be no confusion but be seen as a long time .
line13	Source	sie ist äußerst besorgniserregend .
	Reference	it is most **worrying** .
	Baseline	it is very important .
	Lexical	you are extremely extremely **worrying** .
	Transformer	it is very important .
line242	Source	europa 2020 (eingereichte entschließungsanträge) : siehe protokoll
	Reference	europe **2020** (**motions** for resolutions tabled) : see minutes
	Baseline	european union (rule 142) : see minutes
	Lexical	europe **2020** (**motions**) : see minutes
	Transformer	europe (pl) : see minutes

Table 6: Example translations of the Baseline model, the Lexical model, and the Transformer model. **bold** indicate the word correctly translated by the LSTM Lexical model but failed to be translated by others.

```
query_i = query_i.contiguous().view(h * b, -1, k)
key_i = key_i.contiguous().view(h * b, -1, k)
value_i = value_i.contiguous().view(h * b, -1, k)

# [b*h, t, s]
qk = torch.bmm(query_i, key_i.transpose(1, 2))
qk /= self.head_scaling

if key_padding_mask is not None: # [b, seq_len]
    key_padding_mask = key_padding_mask.unsqueeze(1) # [b, 1, seq_len]
    key_padding_mask = key_padding_mask.repeat(h, 1, 1)
    qk.masked_fill(key_padding_mask, float('-inf'))

if attn_mask is not None:  # [seq_len, seq_len]
    attn_mask = attn_mask.unsqueeze(dim=0)
    qk += attn_mask

qk = torch.softmax(qk, dim=-1)  # qk:[b*h, t, s]

attn = torch.bmm(qk, value_i)   # [b*h, t, k]
attn = attn.contiguous().view(h, b, -1, k).transpose(0, 2)
attn = attn.contiguous().view(-1, b, h*k)
attn = self.out_proj(attn)

attn_weights = qk.contiguous().view(h, b, t, -1) if need_weights else None
```

```
# TODO: ------------------------------------------------------------------- CUT
```

References

[1] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[2] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[3] Miguel Domingo, Mercedes Garcıa-Martınez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. How much does tokenization affect neural machine translation? *arXiv preprint arXiv:1812.08621*, 2018.

[4] Richard A Hudson. *Word grammar*. Blackwell Oxford, 1984.

[5] Rebecca Knowles and Philipp Koehn. Context and copying in neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3034–3041, 2018.

[6] Junjie Hu, Mengzhou Xia, Graham Neubig, and Jaime Carbonell. Domain adaptation of neural machine translation by lexicon induction. *arXiv preprint arXiv:1906.00376*, 2019.

[7] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*, 2016.

[8] Sébastien Jean, Orhan Firat, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. Montreal neural machine translation systems for wmt'15. In *Proceedings of the tenth workshop on statistical machine translation*, pages 134–140, 2015.

[9] Wei He, Zhongjun He, Hua Wu, and Haifeng Wang. Improved neural machine translation with smt features. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[10] Toan Nguyen and David Chiang. Improving lexical choice in neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 334–343, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[12] Michael A Nielsen. *Neural networks and deep learning*, volume 25.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[14] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.

[15] Albert Zeyer, Parnia Bahar, Kazuki Irie, Ralf Schlüter, and Hermann Ney. A comparison of transformer and lstm encoder decoder models for asr. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 8–15. IEEE, 2019.