NLU+ coursework 2

s2041285 s2099657

February 2021

1 Question 1

- A: If self.bidirectional is set to True, hidden states and cell states of the last time step in every layer are concatenated from original shape of [num_layers*2, batch_size, hidden_size] to [num_layers, batch_size, hidden_size*2]. 'final_cell_states' uses linear update of the last time step's cell state, thus it could carry long term information. 'final_hidden_states' uses non-linear update of the current time step's cell state. It more cares about short-term information.
- B: The attention context vector is a weighted average of all time steps' encoder output, the weight distribution depends on the similarity between the target output and each time step of the encoder output. For each batch, we have to pad shorter sentences with token '0' to make sure all the sentences have the same length. Therefore, '0' tokens should not be considered in attention calculation. We mask the attention score of token '0' as '-inf' to make their weights approach to 0.
- C: For each batch, the attention scores are the the dot products between the target vector and the linear projection of the encoder output. Matrix multiplication could calculate the dot product of the decoder representation and each encoder representation, and this is used to measure the similarity of the encoder and the decoder representation.
- D: The decoder state first is initialized as zero matrix. There are two cases when 'cache_state == None'. First, for 'incremental_state' in 'forward()' function of LSTMDecoder is set to None, 'cache_state' is always None. Another case is when 'incremental_state' is a dictionary and it does not have the key for the model instance. 'input_feed' is used for calculation of the input for the next LSTM unit.
- E: The attention combines the previous target hidden state and the output of the encoder as the new 'input_feed'. The attention combines the previous target state and the encoder outputs to see which encoder outputs could be more similar, which is more helpful for prediction. Using dropout could make the model have stronger generalization ability.
- F: The code is to pass a batch of training data into the model, calculate the training loss, and then update the gradients of the parameters. The gradients are clipped in case they are too big.

2 Question 2

- 1. In English data, there are 124111 word tokens and 8329 word types. In German data, there are 112621 word tokens and 12505 word types.
- 2. In English data, there are 3910 word tokens will be replaced by <UNK>, and the subsequent total vocabulary size is 4420. In German data, there are 7460 word tokens will be replaced by <UNK>, and the subsequent total vocabulary size is 5046.
- 3. For both English and German data, words containing digits would be commonly replaced. For example, word 'h-0254/03' appears only once in both English and German data. Replacing these rare words does not affect the test set since there is quite low probability for these rare words to appear in the test set. Even if the same token appears in the test set, the token is less likely to be translated correctly because the corresponding embedding cannot be learnt well.
- 4. There are 1461 unique vocabulary tokens that are the same between both languages. Our model will learn to focus more attention on the same token when generating the corresponding token, and this similarity could also provide alignment information.
- 5. Different sentence length cause little effect on NMT system since short sentences are padded. The token ratio (num unique tokens : num total tokens) is nearly 1 : 10, which may increase difficulty in training the word embeddings. Unknown words in the test sentences are replaced with <UNK>, which may cause some information loss and wrong translation, which may cause difficulties in understanding of translated sentences.

3 Question 3

- 1. Greedy decoding selects the token with maximum probability at each time step based on previous sequence, but it does not consider the probability of the whole sequence. What we really need is the sequence with the maximum probability. Using greedy decoding mostly may cause local optimum. Figure 1 shows an example of two kinds of English translation, meaning Bob will visit Japan in January. If 'Bob' and 'is' have been selected, then 'going' is more probable because the combination 'is going' is more common. Then using greedy decoding, 'Bob is going' will be selected. However, the first translation is more suitable than the second one.

(1) Bob is visiting Japan in January
(2) Bob is going to visit Japan in January

Figure 1: Example of two kinds of English translation

- 2. To implement beam search in the decoder, we do not choose the maximum probability token for each time step, instead, we choose several tokens with top K largest probabilities calculated from the softmax layer each time step to increase the size of search space. The detailed procedures are as follows:
  - Assume the beam width is K. For the first time step, choose the tokens with the top K largest probability scores based on the output of softmax.
  - Subsequently, when predicting the word for time step t > 1, we have K previously predicted word combinations. We iteratively feed each possible word combination $\{w_k^1, w_k^2, ..., w_k^{t-1}\}, k = 1, 2, ..., K$ to the decoder and choose the tokens with the top K largest probability scores for time step t, thus we have K^2 word combinations denoted as $\{w_k^1, ..., w_k^{t-1}, w_{k,m}^t\}, k = 1, 2, ..., K, m = 1, 2, ..., K$, with its probability denoted as $P(w_{k,m}^t|x, w_k^1, ...w_k^{t-1})P(w_k^1, ..., w_k^{t-1})$, then we choose K word combinations with largest probabilities.
  - When predicting for the last time step, for each previously predicted word combination, we choose the most probable token. Then we choose the word combination with the largest sentence probability from the K combinations as the final prediction.
- 3. Decoder aims to maximize the probability of the predicted sequence. The probability of the sequence is calculated using the product of all the words' probabilities in the sequence. Since word probability is less than 1, long sentences have smaller probabilities than shorter ones. Thus, the decoder favours short sentences. Using length normalization could introduce the problem of generating much longer sequences.

4 Question 4

- 1. Figure 2 shows the changes (in red) we made in example.sh. Specifically, we added two command lines: '–encoder-num-layers 2' and '–decoder-num-layers 3'.

```
### NAME YOUR EXPERIMENT HERE ##
EXP_NAME="encoder_2_decoder_3"
#################################

## Local variables for current experiment
EXP_ROOT="${RESULTS_ROOT}/${EXP_NAME}"
DATA_DIR="${ROOT}/europarl_prepared"
TEST_EN_GOLD="${ROOT}/europarl_raw/test.en"
TEST_EN_PRED="${EXP_ROOT}/model_translations.txt"
mkdir -p ${EXP_ROOT}

# Train model. Defaults are used for any argument not specified here. Use "\" to add
arguments over multiple lines.
python train.py --save-dir "${EXP_ROOT}" \
        --log-file "${EXP_ROOT}/log.out"  \
        --data "${DATA_DIR}" \
        --encoder-num-layers 2 \
        --decoder-num-layers 3 \
```

Figure 2: Command lines for encoder layers = 2 and decoder layers = 3

- 2. In baseline, the best valid perplexity is **27.2**, the final epoch training loss is **2.141**, the test BLEU is **11.11, 39.9/13.8/7.0/3.9**. In the deeper network with 2-layer encoder and 3-layer decoder, the best

valid perplexity is **29.8**, the final epoch training loss is **2.412**, the test BLEU is **9.37, 39.2/12.7/5.8/2.9**. The deeper model has worse performance than the baseline model on the training, dev, and test sets. The possible reason is that for deeper model, when calculating the gradients, the parameters of the downside layer have to be back-propagated from the upper layers, thus suffer from vanishing gradients.

5 Question 5

	Training Loss	val perplexity	BLEU
Baseline	2.141	27.2	11.11, 39.9/13.8/7.0/3.9
Lexical-LSTM	1.836	23.8	12.68, 44.4/17.0/8.2/4.2

Table 1: Training loss, val perplexity and BLEU score for baseline model and lexical LSTM.

Example 1	Reference	we need to take action in **iran** .
	Lexical	we must have **iran** in iran .
	Baseline	we must have a blind eye .
Example 2	Reference	there have been very serious violations of human rights in **china** and ...
	Lexical	it has been a very strong - in **china** , and that must not forget .
	Baseline	it has been a very difficult contribution for many people , which must not forget .
Example 3	Reference	i would add that the objective for **2020** is 80 grams per kilometre .
	Lexical	i would like to assume that the aim is the aim of the year **2020** .
	Baseline	i would like to ask that the same thing is the year of the year behind the year .
Example 4	Reference	it is forecast to increase to usd **7** 596 in **2008** .
	Lexical	for **2008** , the budget is spent to **7** 7 to 7 .
	Baseline	for the end , it will be a temporary for electoral campaign to be treated on tomorrow .
Example 5	Reference	the treaty now has the democratic endorsement of all **27** member states .
	Lexical	the treaty has now been improved by all **27** member states of all member states .
	Baseline	the treaty of the treaty has been given the member states to play a country .

Table 2: Five translation examples from reference, lexical model and baseline model.

As shown in table 1, lexical model has lower training loss, lower dev perplexity, and higher test BLEU score than baseline model, which means the addition lexical translation is beneficial to performance by all these three automatic metrics.

From our observation, by addition of lexical translation, more words shared by the two languages such as numbers and some country names could be translated correctly compared to the baseline model, as shown in table 2. In table 2, the words in bold in the five examples are those that can be translated correctly in lexical model but cannot by baseline model. Given the fact that the source token embeddings have to pass through a LSTM network and attention mechanism before being utilized by the decoder, the baseline model is prone to generate a token that fits the context rather than the corresponding source token.

By adding a simple neural network which processes the source tokens and feeds them to the decoder output, the decoder tends to generate a token based on both the context and the target source token. As a result, theoretically we could see more unigrams being translated correctly. However, due to the limitation of the baseline model and the added simple neural network, we can only see this obvious improvement in the shared words between these two languages, such as numbers and country names. That means the equivalent effectiveness of the added neural network is just a mapping function from source tokens to the corresponding output tokens.

To conclude, the lexical model could take both the context and the target token into consideration when generating the output, thus unigrams may be translated more accurately. For training loss, dev perplexity and test BLEU, they only consider whether the translated words are the same as the words in reference translation. In other words, the synonym translations are taken as incorrect, thus we could see an improvement using lexical model compared to baseline with these three metrics.

6 Question 6

- A. Positional embeddings are used to incorporate word order information to the input. Commonly a word plays different roles in different positions. Without positional embeddings, the attention for each word pair will be the same regardless of their positions in the input sequence. In LSTM, the hidden state is computed by the previous hidden state and the input token at each time step. LSTM learns sequential information of a sentence, which depends on word order.

- B. When predicting token for time step i, self_attn_mask is to mask the input after time step i. Without self_attn_mask, the whole input is fed into the decoder, then the model will learn to generate the output token directly from input since the answer is fed as part of the input. In the encoder, we need to attend to every other token when calculating the attention for a particular toke, thus we do not need mask in the encoder. Incremental decoding decoders tokens one by one. When predicting the token at the time step i, we only have the previous tokens, then there is no need to implement the mask.

- C. Linear projection is needed to project previous forward_state to a vector representing the probabilities of the dictionary words to be the next word. The dimensionality of forward_state after this line is [**batch_size, tgt_time_steps, dictionary_size**]. If 'features_only=True', the output represents the feature vector of the final transformer decoder layer.

- D. To keep sentence lengths in a batch are the same, we pad shorter sentences with token '0'. Therefore, when calculating attention, these token '0' should not be taken into calculation of attention weights and required to be masked. Output shape of **state** is [**tgt_time_steps, batch_size, embed_dim**].

- E. Self attention calculates the attention scores for the decoder input, while the encoder attention calculates the attention scores with encoder outputs as keys and values and outputs with the self attention as queries.

  The key_padding_mask is for masking the padding token '0' in the input, which could avoid attention weights to be allocated to the padding tokens. The attn_mask is used in case the model learns the answer directly from the input.

  The value used is the output from the encoder, and the whole encoder output fed into the decoder will not affect the prediction, If attn_mask is adopted here, the token prediction only depends on the previous source tokens, which does not make sense. Thus, we do not need attn_mask in Encoder Attention.

7 Question 7

The code for forward method in the **MultiHeadAttention** is as follows:

```
def forward(self, query, key, value, key_padding_mask=None, attn_mask=None, need_weights=True):
    # Get size features
    tgt_time_steps, batch_size, embed_dim = query.size()
    assert self.embed_dim == embed_dim
    '''
    ___QUESTION-7-MULTIHEAD-ATTENTION-START
    '''

    #linear projections of the input key, query, value
    k = self.k_proj(key.transpose(0,1))
    q = self.q_proj(query.transpose(0,1))
    v = self.v_proj(value.transpose(0,1))

    #reshape k, q , v into self.num_heads heads with size of self.head_embed_size
    k = k.reshape(batch_size, -1, self.num_heads, self.head_embed_size)
    q = q.reshape(batch_size, -1, self.num_heads, self.head_embed_size)
    v = v.reshape(batch_size, -1, self.num_heads, self.head_embed_size)

    q = q.transpose(1,2)
    k = k.transpose(1,2)
    v = v.transpose(1,2)

    #get attention weights through dot product of k and q,
    #and rescaling using self.head_scaling
    attn_weights = torch.matmul(q / self.head_scaling, k.transpose(2,3))

    # key_padding_mask is for masking padded tokens
    if key_padding_mask is not None:
        key_padding_mask = key_padding_mask.unsqueeze(dim=1).unsqueeze(dim=2)
        #shape of attn_weights: [batch, num_head, time_steps, time_steps]
        attn_weights.masked_fill_(key_padding_mask, float('-inf'))

    #attn_mask foces model looking forward
    if attn_mask is not None:
        #shape of attn_mask: [time_step, time_step]
        attn_mask = attn_mask.unsqueeze(dim=0).unsqueeze(dim=0)
```

```
    attn_weights += attn_mask

#normalize the attn_weights and get the attention by calculating weighted
#sum of the values using the normalized attn_weights
attn_weights = F.softmax(attn_weights,dim=-1)
attn = torch.matmul(attn_weights, v)

attn = attn.transpose(1,2)
attn = attn.reshape(batch_size,tgt_time_steps,embed_dim)

attn = self.out_proj(attn)
attn = attn.transpose(0,1)

attn_weights = attn_weights.reshape(self.num_heads, batch_size,
             tgt_time_steps, key.size(0)) if need_weights else None

'''
___QUESTION-7-MULTIHEAD-ATTENTION-END
'''

return attn, attn_weights
```

	Training Loss	val perplexity	BLEU
Baseline	2.141	27.2	11.11, 39.9/13.8/7.0/3.9
deeper-LSTM	2.412	29.8	9.37, 39.2/12.7/5.8/2.9
Lexical-LSTM	1.836	23.8	12.68, 44.4/17.0/8.2/4.2
transformer	1.364	31	10.79, 41.5/14.9/7.0/3.5

Table 3: Training loss, val perplexity and BLEU score for different models

As table 3 shows, transformer has lower training loss but higher validation perplexity than the LSTM-based models. For BLEU score, transformer performs better than the deeper-LSTM, but worse than the other two LSTM-based models.

Transformer has the lowest training loss but highest validation perplexity among these models, we can infer that Transformer overfits the training data. It is possible because the transformer is a little complex given the relatively small training corpus, we find that transformer has 2707,652 parameters in total, which is almost twice of that in LSTM baseline model (1456,644 parameters). Although the transformer achieves much worse validation performance compared to LSTM models, it could achieve comparable performance with Baseline model in test BLEU. One possible reason could be that there is no target translations (reference) to be fed into the model, which causes a gap in test and validation performance, and we can infer that this gap in LSTM models may be larger than that in transformer.

The poor quality and quick convergence of the model are evaluated in dataset and model size aspects. From dataset aspect, the size of training dataset is small, which has only about 100,00 sentences. Lack of training samples could make the model have poor generalization ability. From model size aspect, the number of parameters of transformer model is nearly two times larger than that of LSTM models. Larger model with more parameters could be more representative than simpler models, thus the transformer model fits the dataset and converges quickly.

To improve model performance, first we could enlarge the training dataset. By enlarging training dataset, we could enlarge the size of dictionary and have more training samples, which could make training more thoroughly and generalization better. Secondly, we could utilize the results from all layers in the encoder and decoder, in precise, we pass the result from every layer to the output of the following layers, like a DenseNet[1]. In this way, the information flow in the whole net will be enhanced.

8 Reference

[1] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Wein-berger. Densely connected convolutional networks. InProceedings of theIEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017