Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Alan Akbik

Berlin, 07.06.2024
Lehrstuhl Maschinelles Lernen

Introduction to Natural Langauge Processing

Exercise 7

Submission: Hand your homework until 17.06.2024 (Monday), 23:59 via Github Classroom. Each exercise is worth a total of **10 points** and you need **80%** (of total points over all exercises) to be admitted to the final exam. Sometimes, there are optional tasks that will give you extra points. Exercises in Github Classroom should be completed and submitted individually. However, we encourage you to work on the problems in teams. Please note the information available in Moodle: `https://hu.berlin/nlp24-moodle`.

Github Classroom: `https://classroom.github.com/a/7c6VLTIh`

Task 1 (Dot-product Attention) $3 + 4 + 3 = 10$ points

In this task, you will be implementing dot-product attention for a sequence-to-sequence model, and train one model with attention and one without. As you recall from the last exercise, this type of models are used for a task like e.g. machine translation. LSTMs can handle this task pretty well, however they may start to struggle whenever they have to encode multiple potentially rare tokens. In such cases the attention mechanism may be very useful.
Attention can be generally defined as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(QK^T\right)V$$

In folder `task_1` you are provided with a file `models.py`, in which the model class `Seq2Seq()` is implemented. You will need to:

(a) Complete the dot-product attention calculation in the `forward()` pass of the `Attention()` class. In our case, the keys and values are the same and correspond to the encoder LSTM outputs, while the query is a hidden state produced by the decoder at each decoding step.

The attention calculation consists of 3 steps:

1. Compute the attention scores QK^T (multiply the current decoder hidden states and encoder outputs)
2. Normalize the attention scores by passing them through softmax to obtain the attention weights: $\text{softmax}\left(QK^T\right)$
3. Multiply the weights with values V (encoder outputs)

The output can be seen as a weighted sum of the encoder hidden states that captures the relevant information from the input sequence, focusing on parts that are most relevant to the current output being generated.

Since we are training on mini-batches, use the `torch.matmul()` operation to multiply the queries, keys and values hidden with each other. In order to do it correctly, pay attention to the dimensions of the input matrices and transpose them accordingly. E.g. the output after step 1 should be a 3D tensor of shape $(mini\_batch\_size \times 1 \times source\_seq\_len)$. The target shape of the attention output is listed in the code.

(b) Implement the `decode_attention()` method of the Seq2Seq model in `models.py`. This method allows for training a sequence-to-sequence model with dot-product attention. The method should:

- Embed sentences in target language (pass the one-hot encoded target language tokens through the decoder embedding layer).
- For each target sequence token, pass it through the decoder LSTM together with the encoder LSTM outputs, calculate the attention, concatenate it with the decoder output and pass through the prediction layer.
- Collect the predictions for all target sequence tokens and calculate the log probabilities for all tokens.

(c) Train one model without the attention mechanism, and one model with the attention mechanism on top using the following commands:

```
python run_task_1.py no_attention
```

```
python run_task_1.py attention
```

The training script is ready to be run as is, and only requires the correct implementation of the attention mechanism and the decode method with attention in order to train the model with attention. The results will be stored in the `results.txt` file. If everything is implemented correctly, the model with attention should outperform the model without attention.

Afterwards, visualize the attention scores of the trained model on a three sentence pairs. You can come up with your own examples for that. Use the `plot_util.py` script for that.

Task 2 (Self-Attention: Bonus Task) **(3 + 2 = 5) points**

In this task, you will be implementing a self-attention mechanism that will be used on top of an LSTM for the textual entailment task. You will also train one model with and one model without the attention mechanism.
Compared to the dot-product attention from Task 1, in self-attention, the queries, keys, and values all come from the same input sequence. Each position in the sequence is transformed into a query, key, and value vector.
In folder `task_2` you are provided with a file `models.py`, in which the model class `LSTMSelfAttentionModel()` is implemented. You will need to:

(a) Implement the self-attention mechanism inside the `SelfAttention()` class. You will need to instantiate the query, key and value matrices as `torch.nn.linear()` instances, and complete the forward pass implementation.

Similarly to the first task, you need to:

- Pass the LSTM outputs through the query and key and value matrices. Multiply the resulting queries and keys with each other to get the attention scores. Again, pay attention to the matrix shapes in order to get correct results.
- Pass them through the softmax layer to normalize them.
- Multiply the normalized scores with the values to get the final output.

(b) Train one model without the attention mechanism, and one model with the attention mechanism on top using the following commands:

```
python run_task_2.py no_attention
python run_task_2.py attention
```

Again, the training script is ready to be run as is, and only requires the correct implementation of the attention mechanism in order to train the model with self-attention.

The results will be stored in the `results.txt` file as well. If everything is implemented correctly, the model with attention should outperform the model without attention.