Prof. Dr. Andreas Hotho, Albin Zehe, Jonas Kaiser
Data Science Chair

03./04.07.2024

9. Assignment in "Machine Learning for Natural Language Processing"

Summer Term 2024

1 General Questions

1. A convolution transforms the input by weighting the input values with a learned filter. A self-attention layer also combines the values that are computed from the input using weights. What are the differences between the two?

> First, a convolution only spans a certain part of the input, based on the filter size. A transformer usually takes the whole input into account. Second, the convolution learns weights for the input combination. These weights are learned and fixed. The self-attention uses "dynamically computed weights" based on the input, since it computes keys, queries, and values from the input and then weights the values based on the compatibility function (e.g. the dot product) of queries and keys.

2. Explain why we need positional encodings when training a transformer but not for simple multi layer perceptrons or recurrent neural networks!

> Transformers require positional encodings because they lack built-in sequential processing mechanisms and rely heavily on self-attention, which is permutation invariant and re-uses the same transformation weights across all positions in the sequence when creating the Key, Query and Value representation for each position. Changing the order of the inputs would not alter the outcome of the attention layers.
>
> RNNs don't need positional encodings due to their inherent recurrence, which captures sequential dependencies.
>
> MLPs don't need positional encodings either, as they learn a weight for each combination of nodes between one layer and the next. This means that the order of the input is highly relevant to how the model processes it since there

are independent weights for each possible position. This also makes MLPs inefficient for working with sequential data since the model cannot share parameters and needs to learn how to handle every possible input at all positions independently.

3. The basic transformer architecture uses attention in two different ways per layer. Explain how they differ and what function each one serves intuitively!

   - Self-Attention: In self-attention, the model computes attention weights between different positions within the same sequence. This allows the model to capture relationships between tokens in the input sequence. Intuitive function: "How important is each token in relation to others in the same sequence?"
   - Cross-Attention/Encoder-Decoder Attention In encoder-decoder attention, the model computes attention weights between the output of the encoder and the decoder's input at each position. This allows the model to focus on specific parts of the input sequence when generating the output. Intuitive function: "What part of the input sequence is most relevant for generating this particular token in the output?"

2 Byte-Pair Encoding

Given a (very small) dataset consisting of the words “hello”, “hell”, “love”, and “lovely”.

Perform byte pair encoding using these words for 3 steps to create a new vocabulary and rules to encode new words. If there are multiple subtoken pairs with the same count, use the alphabetically first pair.

We start with a vocabulary consisting of all characters (subtokens) in our words plus the end-of-word token “</w>”. Then, we iteratively (3 times) combine the most common pairs of subtokens to a new subtoken and save this rule for later.

Initial vocabulary: h e l o v y </w>

The words can be written as: h e l l o </w>, h e l l </w>, l o v e </w>, l o v e l y </w>

Counts per subtoken pair: he: 2, el: 3, ll: 2, lo: 3, ov: 2, ve: 2, ly: 1, o</w>: 1, l</w>: 1, e</w>: 1, y</w>: 1

Both el and lo have three occurrences, thus we use el and combine them to a new subtoken.

Vocabulary after first iteration: h e l o v y </w> el

The words can be written as: h el l o </w>, h el l </w>, l o v e </w>, l o v el y </w>

Counts per subtoken pair: hel: 2, ell: 2, lo: 3, ov: 2, ve: 1, vel: 1, ely: 1, o</w>: 1, l</w>: 1, e</w>: 1, y</w>: 1

lo is the most common pair, thus we combine it to a new subtoken.

Vocabulary after second iteration: h e l o v y </w> el lo

The words can be written as: h el lo </w>, h el l </w>, lo v e </w>, lo v el y </w>

Counts per subtoken pair: hel: 2, ello: 1, lo</w>: 1, ell: 1, l</w>: 1, lov: 2, ve: 1, e</w>: 1, vel: 1, ely: 1, y</w>: 1

Both hel and lov have two occurrences, thus we use hel and combine them to a new subtoken.

Vocabulary after third/final iteration: h e l o v y </w> el lo hel

The words can be written as: hel lo </w>, hel l </w>, lo v e </w>, lo v el y </w>