Deep Learning (IST, 2023-24)

Practical 09: Attention Mechanisms

André Martins, Ben Peters, Chryssa Zerva, Duarte Alves

Pen-and-Paper Exercises

The following questions should be solved by hand. You can use, of course, tools for auxiliary numerical computations.

Question 1

Let

$$\boldsymbol{x}^{(1)} = [-2.0, 1.0, 0.5]^\top, \quad \boldsymbol{x}^{(2)} = [1.0, 1.5, -0.5]^\top, \quad \boldsymbol{x}^{(3)} = [-1.5, 1.0, -0.5]^\top, \quad \boldsymbol{x}^{(4)} = [-2.0, -2.5, 1.5]^\top$$

be an sequence of length 4, where each element is a vector in $\mathbb{R}^3$. We let $\boldsymbol{X} = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}]^\top \in \mathbb{R}^{4\times 3}$ be the resulting input matrix.

1. Let $\boldsymbol{q} = [-2.0, 1.0, -1.0]^\top$ be a query vector. Compute the attention probabilities and resulting output vector induced by this query on $\boldsymbol{X}$ using scaled dot product attention.

2. Let us suppose now that we want to compute (single-head) self-attention on this input. Assuming that the projection matrices for queries, keys, and values are respectively

$$\boldsymbol{W}_Q = \begin{bmatrix} 1 & -1.5 \\ 0 & 2 \\ -0.5 & -1 \end{bmatrix}, \quad \boldsymbol{W}_K = \begin{bmatrix} -1.5 & -1 \\ 2.5 & 0 \\ 0.5 & -1 \end{bmatrix}, \quad \boldsymbol{W}_V = \begin{bmatrix} 1 & 2.5 \\ -0.5 & -2 \\ 0 & -1 \end{bmatrix},$$

   compute the query, key, and value matrices ($\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V}$), and the resulting output vector $\boldsymbol{Z}$. Plot the attention map.

3. Let us now assume that we have a second attention head whose parameters $\boldsymbol{W}_Q^{(2)}$, $\boldsymbol{W}_K^{(2)}$, $\boldsymbol{W}_V^{(2)}$ are matrices with all-ones (keeping the first attention head). Using

$$\boldsymbol{W}_O = \begin{bmatrix} -1 & 1.5 & 2 \\ 0 & -1 & -2 \\ 1 & -1.5 & 0 \\ 2 & 0 & 1 \end{bmatrix},$$

   compute the resulting output vector $\boldsymbol{Z}$.

Question 2

In this exercise, you will rewrite the attention mechanism **without** using matrix-matrix multiplications.

1. First, write the output of the dot product attention mechanism $\boldsymbol{z}_i$ as a function of the query vector $\boldsymbol{q}_i \in \mathbb{R}^{d_Q}$, the keys $\boldsymbol{k}_j \in \mathbb{R}^{d_K}$ (for $j \in \{1, \ldots, L\}$) and the values $\boldsymbol{v}_k \in \mathbb{R}^{d_V}$ (for $k \in \{1, \ldots, L\}$). In this exercise, consider only a single head and disregard the projection matrices.

2. Now, write the full multi-head attention mechanism output $\boldsymbol{z}_i$ as a function of the input vectors $\boldsymbol{x}_i \in \mathbb{R}^D$ and $\boldsymbol{x}_j \in \mathbb{R}^D$ (for $i, j \in \{1, \ldots, L\}$). Consider H heads and the projection matrices $\boldsymbol{W}_Q^{(h)} \in \mathbb{R}^{D \times d_Q}$, $\boldsymbol{W}_K^{(h)} \in \mathbb{R}^{D \times d_K}$, $\boldsymbol{W}_V^{(h)} \in \mathbb{R}^{D \times d_V}$ and $\boldsymbol{W}_O^{(h)} \in \mathbb{R}^{H d_V \times d_O}$.

Programming Exercises

The following exercises should be solved using Python, you can use the corresponding practical's notebook for guidance.

Question 3

In this exercise, you will implement an attention mechanism for the simple sequence-to-sequence task of string reversal. Given a source string over some alphabet (in our case, just the first four letters of the Roman alphabet), the model's task is to return the string in reversed order. For this task, we will use randomly generated training and validation data. A simple LSTM-based sequence-to-sequence model has already been implemented for you in the attached notebook `attention.ipynb`.

1. Train a unidirectional sequence-to-sequence model on the string reversal task for 30 epochs. Reasonable hyperparameters are already included in the notebook. Observe the results.

2. In this exercise, you will implement a simple but effective style of attention mechanism called *dot-product attention* (the same as in Question 1 but without the scale factor $\sqrt{d}$). Dot-product attention works as an extra layer inside an RNN decoder that allows it to make more focused use of the hidden states computed by the encoder. The mechanism receives two inputs at time step t: the *query* $\boldsymbol{s}_t$ is the output of the decoder RNN; the *context* $\boldsymbol{F} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_S]$ is the sequence of all the hidden states computed by the encoder RNN. The attention mechanism first computes an unnormalized attention score between target position t and source position s with a simple dot product, and then normalizes with softmax:

$$z_{ts} = \boldsymbol{s}_t^\top \boldsymbol{h}_s$$
$$\boldsymbol{a}_t = \mathrm{softmax}(\boldsymbol{z}_t).$$

   Then, $\boldsymbol{a}_t$ is used to compute a weighted sum $\boldsymbol{c}_t = \boldsymbol{F}\boldsymbol{a}_t$ over the source hidden states.

   Finally, $\boldsymbol{c}_t$ is concatenated to $\boldsymbol{s}_t$ and they are passed through a feed-forward layer, returning the *attentional hidden state* $\tilde{\boldsymbol{s}}_t = \tanh(W_c[\boldsymbol{c}_t; \boldsymbol{s}_t])$.

   Your goal is to implement dot-product attention in the `DotProdAttention` pytorch module in `attention.ipynb`. After you implement it, train the model with the same hyperparameters as in the previous exercise. How does performance compare?

3. Visualize the attention distributions returned by your model. Do they look the way you expected they would?