Due date: November 19, 2024
Please submit a pdf with answers (add figures if appropriate)
as well as the notebook used to generate them.

In this exercise, we revisit the classification problem from Exercise 4, i.e., we want to train a model to predict the number between 0 and 9 shown on an image. We will explore the *attention* mechanism, another important method in machine learning.

1 Theoretical background (3 bonus pt)

The defining equation of the attention mechanism is given by

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QW^Q(KW^K)^T}{\sqrt{d_{\text{model}}}}\right)VW^V. \tag{1}$$

For simplicity, assume that K, Q, V all have the same dimensionality $K, Q, V \in \mathbb{R}^{n_{\text{batch}} \times d_{\text{seq}} \times d_{\text{model}}}$, where n_{batch} is the batch size, d_{seq} the sequence length and d_{model} the feature dimension. Further, assume that $W^K, W^Q, W^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$. Briefly answer the following questions:

1. What is the meaning of all appearing quantities and how are they used in the attention mechanism?
2. What is the attention map (also known as attention weights)? What is its dimension? How can we interpret it?
3. What is self attention? How does it relate to Eq. (1)?
4. The attention update in Eq. (1) is sometimes referred to as a fully-connected graph neural network. Why?

2 Implementation (7 pt)

2.1 Scaled dot product attention

Create a class `ScaledDotProductAttention` whose forward pass implements Eq. (1). The class needs three linear layers corresponding to the three matrices W^Q, W^K and W^V. The softmax function should be applied along the last axis. You can create a random array `torch.rand.rand(batch_size, d_seq, d_model)` to test that your implementation correctly processes the data.

2.2 Attention block

Create a class `SelfAttentionBlock` whose forward pass implements the following *residual* update:

$$x \leftarrow x + \text{Attention}(x, x, x) \tag{2}$$
$$x \leftarrow x + \text{MLP}(x) \tag{3}$$

with $x \in \mathbb{R}^{n_{\text{batch}} \times d_{\text{seq}} \times d_{\text{model}}}$. Accordingly, besides one attention layer this class needs one (small) multilayer perceptron (MLP). Both transformations should not change the dimensionality of the data. Create a random array `torch.rand.rand(batch_size, d_seq, d_model)` to check that the data is processed in the correct way. Why do we use a residual update?

2.3 Vision transformer

Finally, we bring it all together. Create a class `VisionTransformer` consisting of a linear embedding layer, n_{layers} self-attention blocks (you can simply combine multiple `SelfAttentionBlock` using `nn.Sequential`), a class token $c \in \mathbb{R}^{1 \times 1 \times d_{\text{model}}}$ and a linear output layer.

Starting with a batch of data $x \in \mathbb{R}^{n_{\text{batch}} \times d_{\text{seq}} \times d_x}$, the forward pass should implement the following steps (the dimensionality refers to the array after the step has been applied):

1. Embed the data into a higher-dimensional feature space: $x \in \mathbb{R}^{n_{\text{batch}} \times d_{\text{seq}} \times d_{\text{model}}}$
2. Concatenate the class token: $x \in \mathbb{R}^{n_{\text{batch}} \times 1 + d_{\text{seq}} \times d_{\text{model}}}$ (see notebook for tips)
3. Applying multiple self-attention blocks
4. Using the linear output layer, map the features associated with the class token to a lower dimensional array of shape $x \in \mathbb{R}^{n_{\text{batch}} \times n_{\text{classes}}}$.

$n_{\text{classes}}, d_{\text{model}}, n_{\text{layers}}$, and d_x should be given as an initialization input to the model.

3 Training (3 pt)

3.1 Patching

Attention computes the "interaction" between all elements of a sequence. However, for images consisting of many pixels it becomes expensive to compute the pairwise interaction between

all pixels. This problem can be circumvented by dividing the image into multiple patches and computing the interaction between the patches rather than between all pixels. We therefore require a preprocessing step which divides a batch of images $x \in \mathbb{R}^{n_{\text{batch}} \times n_{\text{pixels}} \times n_{\text{pixels}}}$ into a batch of patches of shape $x \in \mathbb{R}^{n_{\text{batch}} \times n_{\text{patches}} \times n_{\text{p}}^2}$, where n_{p}^2 is the number of pixels in the patch (this is our d_x) and n_{patches} is the number of patches (this is the sequence length d_{seq}). Have a look at the notebook.

3.2 Fitting

Fit a Vision Transformer for the classification of MNIST. Use 5 layers and a model dimension of 64. How long does the model need for convergence and how good is the classification accuracy?

3.3 Attention map

Finally, we want to visualize the attention map of the first `SelfAttentionBlock`. Pick a random example from the test set, compute the attention map using the weights of the attention block, and visualize it together with the test image.