This text is intended to be a short description of our Python script for the Perceiver model’s architecture, which was created based on the details given in the Perceiver: General Perception with Iterative Attention paper[[1]](#footnote-1) and the Dive into Deep Learning book[[2]](#footnote-2).

Let’s start with the MultiHeadAttention class, which is utilized to perform self-attention of the inputs. Firstly, the inputs (queries, keys and values) are passed through a layer normalization layer (see page 17/43 of the above-mentioned paper). Next, as per the instructions from the Dive into Deep Learning book, the shape of the inputs (queries, keys and values) is transformed from (*batch size, number of QKVs, dimension size*) to *(batch size x number of heads, number of QKVs, dimension size / number of heads)* in order to perform multi-head self-attention (where the number of head is a hyper-parameter). It then proceeds to perform the attention scoringmethod (which is pre-defined as the scaled dot production via another Python class) for the inputs with attached learnable parameters. The output from this operation is re-transformed back into the original input’s shape, then again having learnable parameters attached to it. Afterwards, it is passed through a residual connection and layer normalization layer, as shown in the book. Finally, the output will be passed through a dense block (as described in the paper), the details of which shall be presented later.

For the CrossAttention class, based on page 16/43 – chapter C: Architectural details of the Perceiver paper, firstly the latent array will be subjected to layer normalization, then passed through a linear layer so that its final dimension (or number of channels) is equal to that of keys and values. Similar to multi-head attention module, the inputs of keys and values are layer-normalized and have learnable parameters attached to them. Afterwards, the attention scoring method is performed on the inputs – consisting of processed latent array, keys and values, and its output will be passed through a linear layer so its final dimension is again the original dimension of the latent array. This is done so that we can then calculate the residual connection of the output with the original latent array, which is finally passed through the dense block to return the final output of this module.

So far we have mentioned the dense block twice, which acts as the last processing layer of both Attention Modules. The dense block comprises a layer normalization, a linear layer which is followed by a GELU activation function, and finally another linear layer.

The Cross Attention and Multi-Head Attention (for Self-Attention) modules comprise the core components of one Perceiver Block. Specifically, inside a Perceiver Block, the latent array and key-value inputs will firstly be passed through a number of consecutive cross-attention modules, afterwards their final outputs will be fed to a number of self-attention modules. The number of cross-attention and self-attention modules in one Perceiver Block are hyper-parameters that we can choose.

The constructed Perceiver Blocks will form our Perceiver Module. Before going into details about the Perceiver, we need to discuss how the latent array will be initialized. The latent array is defined using the LatentArray class, which takes its inputs for the size and dimension of the latent array, then utilizes the nn.Parameter method to create an initial tensor object with the pre-defined size and dimension. The values of this tensor are then generated using a truncated normal distribution with mean 0, standard deviation 0.02 and truncation bounds [-2, 2] as stated in page 17/43 of the paper. It will be repeated a number of times that are equal to the batch size for the first dimension (so the attention scoring method involving the keys and values can be calculated). After the latent array is initialized, it will be fed along side the keys and values to a number of Perceiver Blocks (the exact number is defined as a hyper-parameter). Next, as per the instruction from the paper, the outputs are averaged over the index dimension or size, then fed to a fully connected neural network with the final layer being of the same size with the number of target labels. It is finally passed through a Softmax function to calculate the probability of each label, which we can use to obtain the predictions of the Perceiver model.

A diagram that displays our Perceiver model’s architecture can be seen below:



1. https://arxiv.org/abs/2103.03206 [↑](#footnote-ref-1)
2. https://d2l.ai/index.html [↑](#footnote-ref-2)