MagmaDNN is a neural network library in C++ aiming at optimizing towards heterogeneous architectures, i.e., multi-core CPUs and GPUs. Currently, no implementation of the multi-head attention layer, which is a core component of transformer models, is provided by MagmaDNN library, despite the popularity and significance of transformer models in various tasks including vision tasks such as medical segmentation [1, 2], image recognition [3], semantic segmentation [4], and natural language processing tasks such as machine translation [5].

To bridge the gap, we present an implementation of the multi-head attention layer in MagmaDNN framework. Our implementation improves the prediction loss by 20.41% compared with TensorFlow implementation, despite consuming extra training time (epoch = 1000, learning rate = 10^-3, batch size = 8, input size = [3 x 8 x 8]). Compared with PyTorch implementation, our method also outperforms it by a clear margin in terms of prediction loss.

The multi-head attention can be formulated as follows:

\[
MHA(Q, K, V) = [h_1, \ldots, h_n]W^O
\]

where \( Q, K \) and \( V \) are the query, key and value matrices, \( \alpha \) is a scaling parameter, and all the \( W \)'s are learnable weights.

We conduct pseudo training experiments to compare the average training speed of different implementations for one single batch input of size \([3 \times 4 \times 4], [3 \times 8 \times 8], [3 \times 16 \times 16], [3 \times 32 \times 32]\) (epoch = 3000).

As shown in the figures 3, 4, 5 and 6, our multi-head attention layer has a faster training speed when the input size is \([3 \times 4 \times 4], [3 \times 8 \times 8] \) or \([3 \times 16 \times 16] \), but has a slower training speed when the input size is \([3 \times 32 \times 32] \).

Our contributions can be concluded in two aspects:

(1) We present an implementation of the multi-head attention layer in MagmaDNN framework, making the development of transformer architecture possible for MagmaDNN library.

(2) We compare the performance of our multi-head layer with PyTorch's and TensorFlow's implementations. Compared with them, our layer outperforms them by a clear margin in the best-epoch prediction loss, despite reasonable extra training time for large-scale data.

**References**


