Attention Algorithm: Implementation in MagmaDNN

Wing-Lim Lau [1] Xiao-Yang Li [2]

[1]The Chinese University of Hong Kong [2]City University of Hong Kong

Abstract

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 $= [3 \times 8 \times 8]$). Compared with PyTorch implementation, our method also outperforms it by a clear margin in terms of prediction loss.

Formulation

The multi-head attention can be formulated as follows:

$$\text{MHA}(Q, K, V) = [h_1, \dots, h_n]W^O \quad (1)$$

$$h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (2)$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\alpha}\right)V \quad (3)$$

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

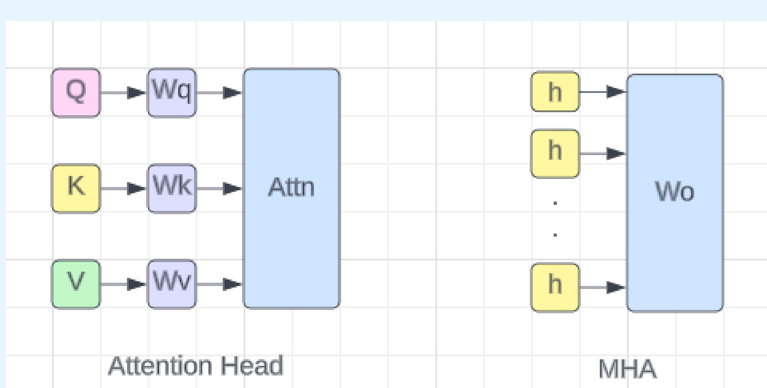

Figure 1. Structure of multi-head attention

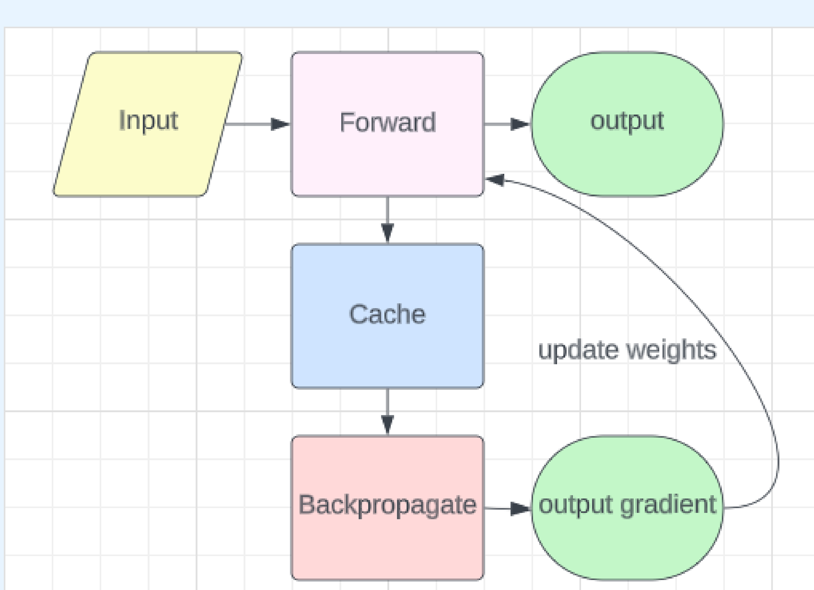

Figure 2. Multi-head attention flowchart

Results

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 $[3 \times 32 \times 32]$.

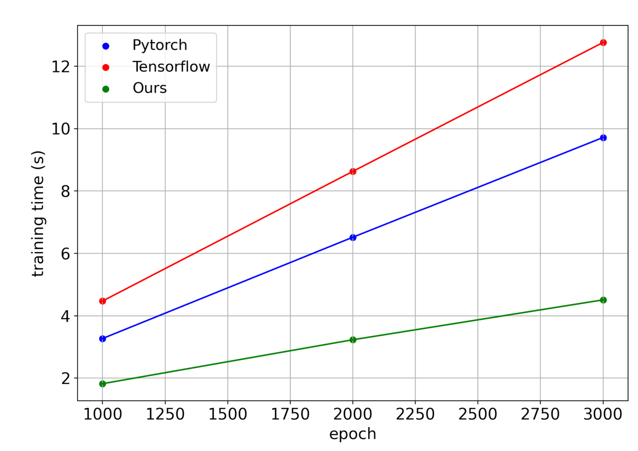

Figure 3. input size = $[3 \times 4 \times 4]$

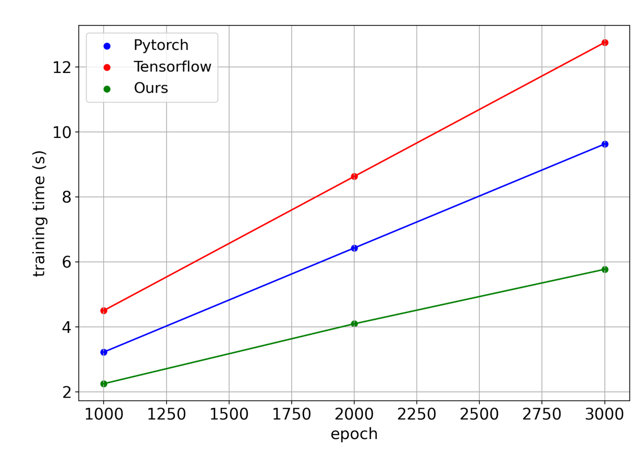

Figure 4. input size = $[3 \times 8 \times 8]$

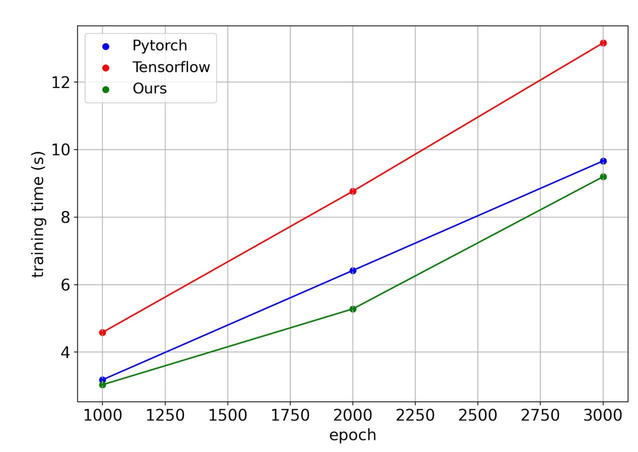

Figure 5. input size = $[3 \times 16 \times 16]$

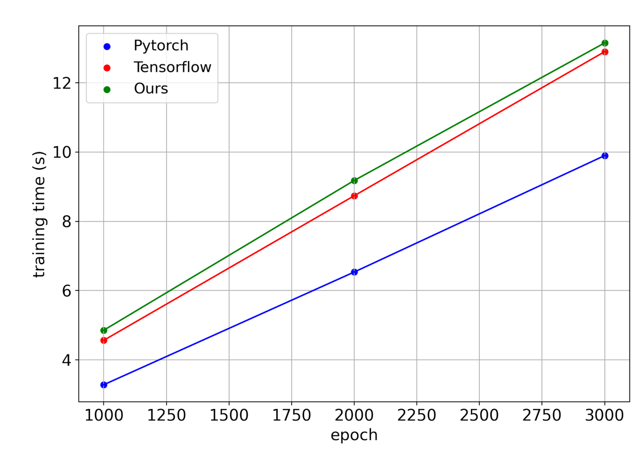

Figure 6. input size = $[3 \times 16 \times 16]$

To further compare the performance, we sampled 800 data (batch size = 8) from a uniform distribution $X \sim U[-1, 1]$ and trained all the models to predict all-zero masks for 1000 epochs. We demonstrate the best-epoch prediction losses of the three in Table 2.

Input Size	Ours (s)	Pytorch (s)	Tensorflow (s)
$3 \times 4 \times 4$	**448.6**	854.4	845.4
$3 \times 8 \times 8$	**583.0**	854.5	841.6
$3 \times 16 \times 16$	937.5	858.2	**850.9**
$3 \times 32 \times 32$	1550.5	865.5	**862.6**

Table 1. Training time for 1000 epochs (#batch $= 100$, batch size $= 8$)

Input Size	Ours (10^{-4})	Pytorch (10^{-4})	Tensorflow (10^{-4})
$3 \times 4 \times 4$	2.634	0.467	**0.341**
$3 \times 8 \times 8$	**0.554**	1.956	0.697
$3 \times 16 \times 16$	**0.0565**	5.523	3.638
$3 \times 32 \times 32$	**0.0555**	11.03	3.595

Table 2. Quantitative comparison on prediction loss, lower loss being better (↓)

Implementation

Initialization All options and configurations are initialized. The memory space required is allocated and initialized via tensor descriptors.

Forward pass Refer to section Formulation.

Backpropagation The trainable parameters of multi-head attention layer are the projection weights W_q, W_k, W_v and W_o. The gradient of attention output w.r.t. projection weights is given by $\frac{\partial out}{\partial W} = \frac{\partial out}{\partial[\hat{Q},\hat{K},\hat{V}]} \times \frac{\partial[\hat{Q},\hat{K},\hat{V}]}{\partial[W_q,W_k,W_v]} = \frac{\partial out}{\partial[\hat{Q},\hat{K},\hat{V}]} \times \frac{\partial[\hat{Q},\hat{K},\hat{V}]}{\partial W}$. We introduce two separate functions, mha_grad_data_device and mha_grad_data_device_weights, to calculate the two terms simultaneously.

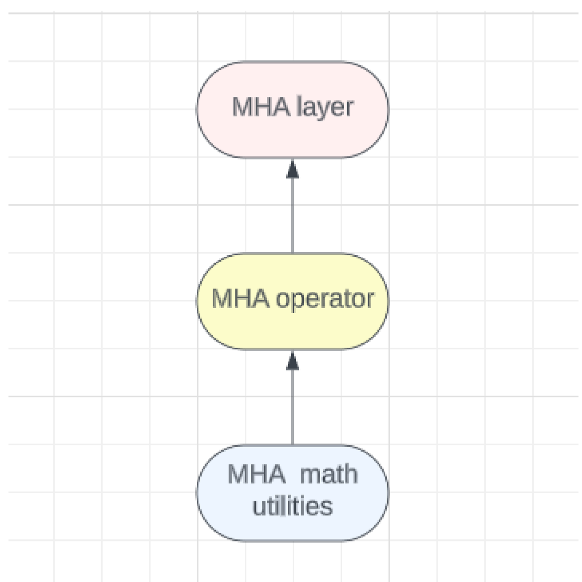

Figure 7. Overview of multi-head attention implementation

Conclusion

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

[1] Y. Gao, M. Zhou, and D. N. Metaxas, "UTNet: A hybrid transformer architecture for medical image segmentation," *CoRR*, vol. abs/2107.00781, 2021.

[2] Y. Gao, M. Zhou, D. Liu, Z. Yan, S. Zhang, and D. N. Metaxas, "A data-scalable transformer for medical image segmentation: Architecture, model efficiency, and benchmark," 2023.

[3] Z. Shen, I. Bello, R. Vemulapalli, X. Jia, and C. Chen, "Global self-attention networks for image recognition," *CoRR*, vol. abs/2010.03019, 2020.

[4] Z. Huang, X. Wang, Y. Wei, L. Huang, H. Shi, W. Liu, and T. S. Huang, "CCNet: Criss-cross attention for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 6, pp. 6896–6908, 2023.

[5] J. Song, S. Kim, and S. Yoon, "AligNART: Non-autoregressive neural machine translation by jointly learning to estimate alignment and translate," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021* (M. Moens, X. Huang, L. Specia, and S. W. Yih, eds.), pp. 1–14, Association for Computational Linguistics, 2021.