


Multi-head Attention

Implementation in MagmaDNN

LI Xiaoyang [†] LAU Wing-Lim [‡]

[†]Department of Computer Science, City University of Hong Kong
[‡]Department of Mathematics, The Chinese University of Hong Kong

July 26, 2023

Supervisor: Dr. Kwai-Lam Wong

Overview

1. Introduction
2. Related works
3. Methodology
4. Experiment
5. Conclusion
6. Limitations
7. Future Directions

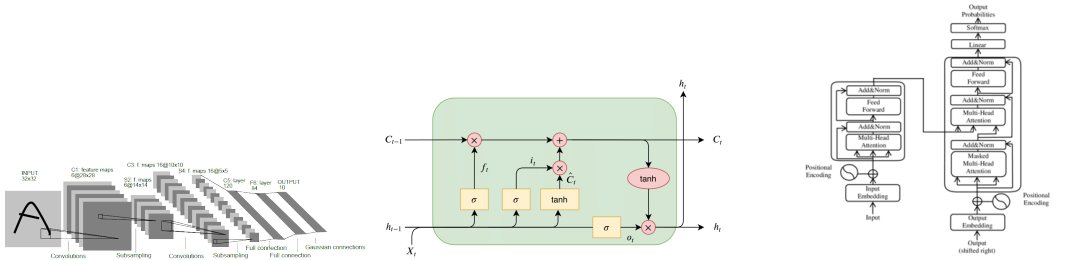
MagmaDNN

MagmaDNN [1] is:

- a MAGMA-driven deep learning library
- a simple, modularized framework for deep learning
- accelerated for heterogeneous computing systems

MAGMA
MATRIX ALGEBRA ON GPU AND MULTICORE ARCHITECTURES

MagmaDNN



(a) CNN (b) RNN (c) Transformer

Figure: Deep learning models

Currently, MagmaDNN supports several deep learning models:

- CNN variants (e.g. UNet [2])
- RNN variants (e.g. LSTM [3])

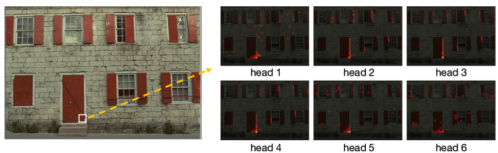
However, it has not yet supported attention, which is key to any transformer model.

Multi-head Attention

Multi-head attention (MHA) is a variant of attention for both vision and NLP tasks (e.g. medical segmentation [4, 5], image recognition [6], semantic segmentation [7], machine translation [8]).

- input transformed into multiple representations
- each processed by a separate attention mechanism in parallel
- outputs then concatenated and passed to a linear layer

Why Attention?



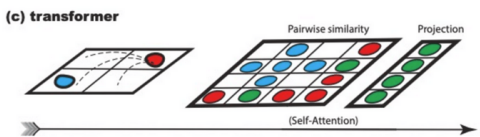


Figure: Long-range dependencies

The idea of MHA is to

- enable the model to attend to different parts of input simultaneously
- capture complex relationships between the input tokens
- allow each head to attend to a different aspect, e.g. positional information, syntax, semantic content
- create a stack of **N** self attention layer to capture both local dependencies and global dependencies between distant tokens

Formulation - Classic Attention

The attention mechanism can be seen as matching a query against a set of keys with associated values.

- Q, K, V: query, key, value matrices (each row is a single query, key, or value)
- The attention coefficients are computed as [9]:

Attention

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

- (i, j)-entry of $QK^\top$ is the dot product of q_i and k_j, which measures the similarity between query i and key j
- Row i of the attention coefficient matrix is a probabilistic linear combination of all the values (weighted over the similarities of query i with each key)

Formulation - Multi-head Attention

For MHA, we do linear projections before and after the attention:

Multi-head Attention

$$\text{MHA}(Q, K, V) = [h_1, \ldots, h_n] W^O$$
$$h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Each h_i is called a head, which resembles attending to an aspect of the input.

Related Works

Sequence transduction

Let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_m)$, where $x_i, y_i \in \mathbb{R}^d$.

We map the sequence X of symbol representations to another sequence Y.

Method	Complexity	Sequential Operation	Maximum Path Length
Self-Attention	$O(n^2 d)$	$O(1)$	$O(1)$
Convolutional	$O(knd^2)$	$O(1)$	$O(\log_k(n))$
Recurrent	$O(nd^2)$	$O(n)$	$O(n)$

Table: Comparison among related methods [9]

Implementation - Initialization

During initialization,

- options, configurations, memory space, tensor descriptors, inputs, weights, outputs are allocated and initialized
- weights are initialized via Xavier initialization [10] so that the variance of weights are stable across every layer, which prevents gradient explosion or vanishing

Xavier initialization: Sampling from a uniform distribution $U[-\beta, \beta]$, with

$$\beta = \sqrt{\frac{6}{N_{\text{prev}} + N_{\text{curr}}}},$$

where N_{prev} and N_{curr} are the respective sizes of the previous and the current layers.

Implementation - Forward Pass

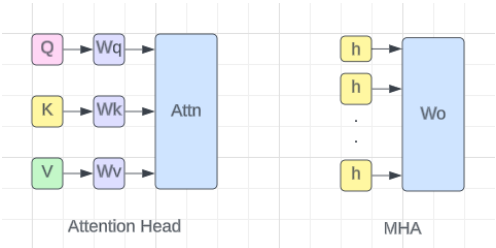


Figure: Multi-head attention, where h denotes Attention Head module on the left

During the forward pass,

- input is divided into 3 components: Q, K, V
- Q, K, V, weights and descriptors are passed to the cuDNN API `cudnnMultiHeadAttnForward()` to compute the attention coefficient matrix

Implementation - Backpropagation

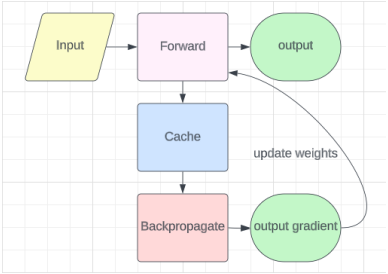


Figure: Trainable parameters: Projection weights W^Q, W^K, W^V and W^O

During backpropagation,

- inputs, weights and descriptors are passed to `cudnnMultiHeadAttnBackwardData()` and `cudnnMultiHeadAttnBackwardWeights()` to compute the derivatives of the output with respect to data and weights

Experiment - Training Setup

The purpose of the experiments is to compare the training speed and prediction loss of different implementations for predicting all-zero in various settings.

- on one single GPU NVIDIA GeForce GTX 1650.
- inputs of sizes $(= [3 \times 4 \times 4], [3 \times 8 \times 8], [3 \times 16 \times 16], [3 \times 32 \times 32])$
- learning rates $(= 10^{-3}, 10^{-4}, 10^{-5})$
- batch size $(= 1, 4, 8)$.
- identical inputs sampled from a uniform distribution $(X \sim U[-1.0, 1.0])$ are used as training data whereas all-zero outputs are used as ground truth.
- use mean square error as the evaluation metric, which can be formulated as:

$$\text{MSE}(P, G) = \frac{1}{N} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} \sum_{k=0}^{l-1} \|P_{i,j,k} - G_{i,j,k}\|_2,$$

where P denotes the prediction mask, G denotes the ground truth mask and n, m, l denotes the batch size, projection dimension, sequence length, respectively.

Experiment - Results

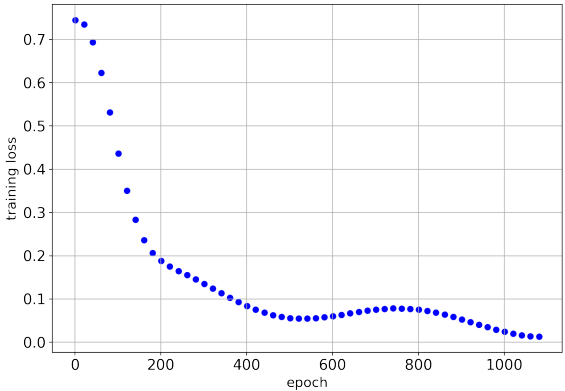


Figure: Training loss visualization (learning rate = 10^{-5})

Experiment - Results

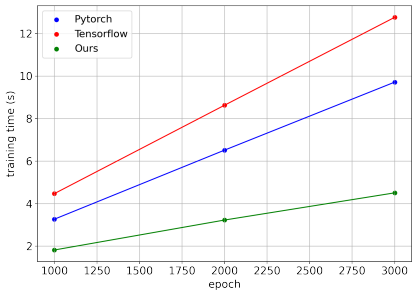


Figure: input size $= [3 \times 4 \times 4]$

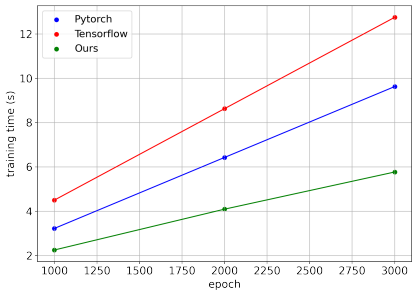


Figure: input size $= [3 \times 8 \times 8]$

Experiment - Results

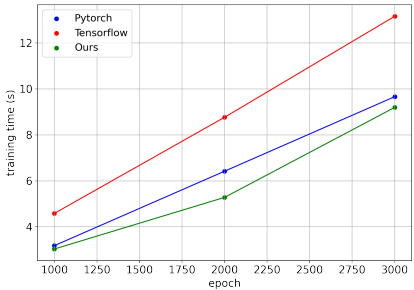


Figure: input size $= [3 \times 16 \times 16]$

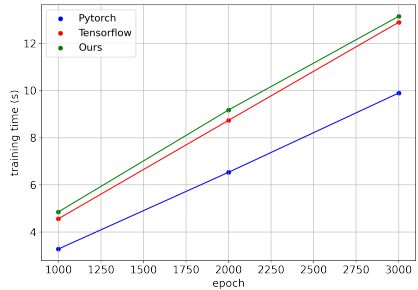


Figure: input size $= [3 \times 32 \times 32]$

Experiment - Results

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: Quantitative comparison on prediction loss, lower loss is better ($\downarrow$)

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: Training time for 1000 epochs (#batch $= 100$, batch size $= 8$)

Conclusion

We conclude our contributions in two aspects:

- We present an implementation of multi-head attention layer in MagmaDNN framework, making the development of transformer architecture possible for MagmaDNN library
- We compare the performance among our multi-head layer with PyTorch and TensorFlow implementations. Compared with competitors, our layer outperforms them by a clear margin in terms of the best-epoch prediction loss, despite reasonable extra training time for large-scale data

Limitations

Notably, our implementation is not flawless. There are two main obstacles for our method to further improve:

- inputs of large size
- gradient explosion

Future Directions - Linearized attention

We briefly summarize the idea of linearized attention [11, 12, 13] to benefit the future development of self-attention mechanism and variants in MagmaDNN framework:

- propose new kernel functions.
- linearize softmax function and approximate the attention matrix

Expand the softmax function as:

$$y^m = \sum_{n=1}^{m} \frac{v^n \phi(k^n, q^m)}{\sum_{l=1}^{m} \phi(k^l, q^m)}$$

Future Directions - Linearized attention

For some kernel function ϕ, which, in the classic setting of softmax function, is defined as: $\phi(k, q) = \exp(k^\top q)$. The general idea of linearized function is to replace the kernel function by some other candidates. For example, in [11], the kernel function is chosen as $\phi(k, q) = \widetilde{\phi}(k)^\top \widetilde{\phi}(q)$ for some function $\widetilde{\phi}$, so that

$$y^m = \sum_{n=1}^{m} \frac{(v^n \widetilde{\phi}(k^n)^\top)\widetilde{\phi}(q^m)}{(\sum_{l=1}^{m} \widetilde{\phi}(k^l)^\top)\widetilde{\phi}(q^m)}$$

and therefore

$$\sum_{n=1}^{m} (v^n \widetilde{\phi}(k^n)^\top)\widetilde{\phi}(q^m) = (\sum_{n=1}^{m} v^n \otimes \widetilde{\phi}(k^n)^\top)\widetilde{\phi}(q^m).$$

It has been shown that switching the order of matrix multiplication improves the time efficiency of self-attention for long sequences [14].

Future Directions - Graph Attention Networks

Core component: graph attentional layer

- Input: node features $\mathbf{h} = [h_1, \ldots, h_N]$, $h_i \in \mathbb{R}^F$
- Output: new node features $\mathbf{h}' = [h'_1, \ldots, h'_N]$, $h'_i \in \mathbb{R}^{F'}$

Formulation:

$$h'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} W h_j\right)$$

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

$$e_{ij} = a(Wh_i, Wh_j)$$

σ : nonlinearity (softmax/sigmoid)

W : $F' \times F'$ learnable weight matrix

$\mathcal{N}_i$: neighbors of node i

a : attention mechanism, $\mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$

Future Directions - Graph Attention Networks

In the original paper [15], a is chosen as

$$a(x, y) = \text{LeakyReLU}(c^\top [x \parallel y])$$

where $c \in \mathbb{R}^{2F'}$ is a learnable weight vector and $\parallel$ denotes concatenation.

One direction is to replace the attention mechanism a by our own MHA layer. We still need more testing to see if this works well.

Acknowledgement

We express our greatest gratitude towards:

- **National Science Foundation** for funding this project
- **The Chinese University of Hong Kong** for funding this project
- **City University of Hong Kong** for funding this project
- **The University of Tennessee, Knoxville** for their support during this project
- **Dr. Kwai-Lam Wong** for his guidance during this project

Q & A

References

[1] D. Nichols, N.-S. Tomov, F. Betancourt, S. Tomov, K. Wong, and J. Dongarra, "MagmaDNN: Towards high-performance data analytics and machine learning for data-driven scientific computing," in *ISC High Performance*, (Frankfurt, Germany), Springer International Publishing, Springer International Publishing, 2019-06 2019.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III* (N. Navab, J. Hornegger, W. M. W. III, and A. F. Frangi, eds.), vol. 9351 of *Lecture Notes in Computer Science*, pp. 234–241, Springer, 2015.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] Y. Gao, M. Zhou, and D. N. Metaxas, "UTNet: A hybrid transformer architecture for medical image segmentation," in *Medical Image Computing and Computer Assisted Intervention - MICCAI 2021 - 24th International Conference, Strasbourg, France, September 27 - October 1, 2021, Proceedings, Part III* (M. de Bruijne, P. C. Cattin, S. Cotin, N. Padoy, S. Speidel, Y. Zheng, and C. Essert, eds.), vol. 12903 of *Lecture Notes in Computer Science*, pp. 61–71, Springer, 2021.

[5] Y. Gao, M. Zhou, D. Liu, and D. N. Metaxas, "A data-scalable transformer for medical image segmentation: Architectures, model efficiency, and benchmarks," *CoRR*, vol. abs/2203.00131, 2023.

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

[7] 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.

[8] 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.

References

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 5998–6008, 2017.

[10] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010* (Y. W. Teh and D. M. Titterington, eds.), vol. 9 of *JMLR Proceedings*, pp. 249–256, JMLR.org, 2010.

[11] Z. Qin, W. Sun, H. Deng, D. Li, Y. Wei, B. Lv, J. Yan, L. Kong, and Y. Zhong, "cosFormer: Rethinking softmax in attention," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022.

[12] I. Schlag, K. Irie, and J. Schmidhuber, "Linear transformers are secretly fast weight programmers," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366, PMLR, 2021.

[13] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlós, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller, "Rethinking attention with performers," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021.

[14] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are RNNs: Fast autoregressive transformers with linear attention," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, vol. 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165, PMLR, 2020.

[15] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.