Acceleration of the Attention Computation for Transformer

Senyu Tong
Carnegie Mellon University
Pittsburgh, PA 15213
senyut@andrew.cmu.edu

Yile Liu
Carnegie Mellon University
Pittsburgh, PA 15213
yilel@andrew.cmu.edu

1 Summary

We are going to implement the multi-head self-attention mechanism that is applied in Transformer, an architecture that aims to solve sequence-to-sequence tasks while handling long-range dependencies with ease, using C++ and CUDA.

2 Background

The Transformer, Vaswani et al. [2017], is an encoder/decoder network that performs sequence-to-sequence modeling, and achieves state-of-the-art performance for translation tasks. Unlike classical Neural Machine Translation models, it is based mainly on attention mechanism, which is further applied by BERT (Bidirectional Encoder Representations from Transformers), Devlin et al. [2018], GAT (graph attention network), Veličković et al. [2017], and many other fields in deep learning.

We present the model architecture in Figure 1. The input and output layers are words embedding from a sentence that is fed into the model; the positional encoding layer adds temporal information into the sequences; The Encoder/Decoder layers perform sequence-to-sequence modeling with multi-head Scaled Dot-Product Attention Mechanism; finally the output probability layer is a lookup table from continuous word representations to discrete tokens.

In this project, we focus on the computation of multi-head attention, the fundamentally important feature of the model. As illustrated in Figure 2, for each input matrix X, where the i^{th} row is the vector embedding for i^{th} word in the sentence, we first extract Query, Key, and Value matrices Q, K, V by multiply X with learnable weights W^Q, W^K, W^V. Then we get $Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$, where d_k is the dimension of key vectors (each row of K). We do the same calculation with different weights for h times to get h different such matrices, and then we concatenate them to form $MultiHead(Q,K,V) = Concat(head_1, ..., head_h)W^O$ where W^O is also an learnable parameter matrix.

To train such a model is always extremely time-consuming, and sometimes requires over 100 million parameters which will often break the memory limits. In this project, we aim to explore possible parallel strategy with respect to matrices dimensions and also the number of heads to accelerate the computation for multi-head attention.

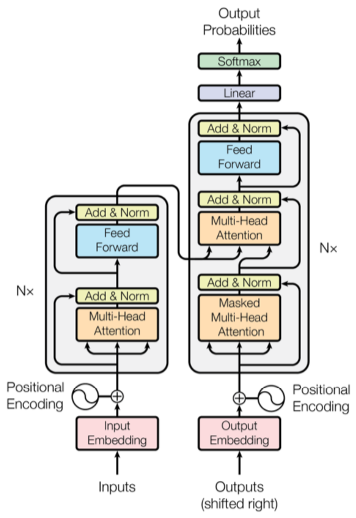

Figure 1: The Transformer - model architecture

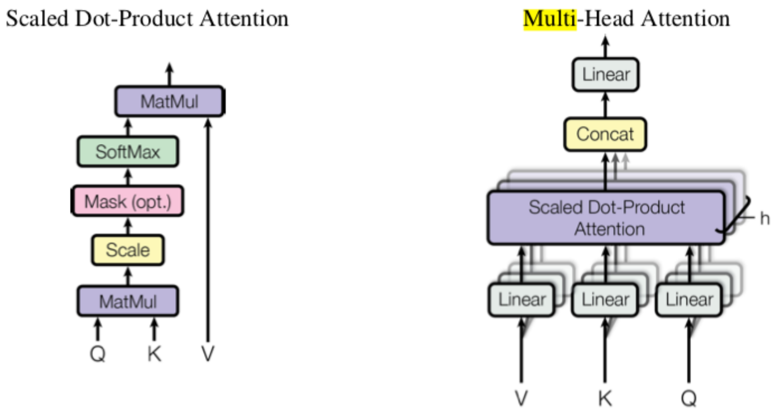

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

3 Challenges

This project presents several challenges.

- The source code of the implementation of Transformers are implemented in Python, we need to find an efficient way to implement in C++ in order to utilize CUDA.
- The architecture of this model involves many matrix operations of large size, so we need to find an efficient way to parallelize matrix multiplication.
- Transformer utilized several attention layers which are inherently parallel work. We want to find an efficient way to parallelize on different layers and combine with our CUDA implementation.
- We may need to implement the backpropagation for this model explicitly, or we may write wrappers to extend pyTorch on our implementation to utilize the graph-based calculation and conduct experiments. The error detection and error-handling can become tricky when we do these.

4 Resources

We need computers with GPUs to run the experiment, so we will use GHC Machines with NVIDIA GeForce GTX 1080 GPUs, the same hardware as in Assignment 2. We will refer to the implementation of Transformer in harvardnlp Klein et al. [2017].

5 Goals and Deliverables

Our plan is to first implement the attention calculation correctly using C++. Then we will optimize the performance by carrying out the computation on GPU. The possible parallelization will occur at multiple heads and matrix multiplications.

Comparing to the most basic sequential implementation, the optimization should be able to achieve 10x speedup. However, since PyTorch is already utilizing some mature parallelization in the computation, our goal is to achieve the same performance as PyTorch. Since there are still many uncertainties during the implementation, we will adjust our goals accordingly.

6 Platform Choice

Our implementation will be based on C++ and CUDA.

7 Schedule

Table 1: Project Schedule

Week	Tasks
Apr 13-17	1. Study PyTorch source code, 2. Implement CPU version of attention calculation
Apr 20-24	Implement GPU version of attention calculation
Apr 27-30	Optimize Efficiency
May 1-4	Wrap up, clean code and work on final report

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.