Accelerating Transformers with Better Transformer’s “Fastpath”

Traditional PyTorch Transformer implementation based on executing sequence of PyTorch operations. This implementation misses several optimization opportunities.

Current Transformer and MultiHeadAttention modules offer great flexibility, but this flexibility comes at the price of performance penalties. For example, the ability for the user to specify an arbitrary Python activation function prevents fusing the common ReLu and Gelu activation functions.

The Better Transformer “fastpath” for common execution scenarios implements
• operator fusion,
• optimized kernels targeting particular hardware targets, and
• skipping of pad tokens in NLP inputs during processing where batch elements have variable sequence length.

Better Transformer
Fused operations and improved kernels deliver up to 2x performance advantage with more efficient fused kernels and reduced dispatch overhead.

Transformers for NLP workloads: Handling variable Sequence Length

Many Transformer applications target Natural Language Processing where inputs may have widely varying lengths. Prior to Better Transformer, this was handled with padding tokens. Make all inputs the length of the longest input. Wasteful processing of padding tokens have no effect on final result.

While padding overhead can be reduced for model training by combining inputs of matching length into tokens, inference must process inputs as they are received. As a result, much as 75% of processed inputs tokens may correspond to padding symbols.

NestedTensors

NestedTensors are a new torch.Tensor subclass that allow to express non-rectangular torch.Tensors, i.e., tensors with variable row length. This enables PyTorch to represent data with variable row-length efficiently as tensors, e.g., sentences of different lengths for NLP processing, or access pattern histories for recommendation engines. Sequence-length information enables kernels to suppress computing on pad tokens.

Better Transformer 2.0: Custom kernels

Recent advances in Transformer implementation have included scaled dot product attention (SDPA) kernels optimized for reference locality and memory efficiency. These kernels have achieved top performance in the latest MLPerf benchmark results:
• Flash attention to optimize for I/O access patterns (Stanford U)
• xFormers memory-efficient SDPA kernels (FAIR)

BT2.0 picks from multiple custom kernels based on target architecture and input characteristics.

Library Integration

Better Transformer is part of the standard PyTorch Transformer API and transparently accelerates applications without change to application logic.

In addition, we have enabled application libraries such as torchtext to use the PT Transformer APIs and benefit from these BT speedups transparently. Enabling for additional libraries is in progress.

References


Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, Christopher Ré (2022). “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness”, arxiv 2205.14135