Fused Cross Entropy Loss

In this document I describe a fused gemm softmax cross entropy loss kernel. Here I will give some background on the cross entropy loss operation and some of its bottlenecks.

In machine learning softmax cross entropy loss is a standard loss function for any classification task. The loss function is typically defined as follows:

$$CE(s, y) = -\log\left(\frac{\exp s^y}{\sum_i \exp s^i}\right) \tag{1}$$

$$= -\log\left(\text{softmax}(s)\right)_y \tag{2}$$

$$= -\log(s^y) + \text{logsumexp}_i(s^i) \tag{3}$$

Where $s \in \mathbb{R}^d$ is some vector of scores produced by a model, y is the index of the correct category, and d is the number of categories to be predicted over. A common use case for this loss function is next token prediction for large language model pretraining. Suppose some large language model has a vocabulary size v and an embedding size e. If a sequence of size l is fed to an llm, the model will produce an output matrix of token embeddings size $X \in \mathbb{R}^{l \times e}$. The loss for that sequence is then computed as

$$L(X, y) = \frac{1}{l}\sum_i CE(S_i, y_i) \tag{4}$$

$$S = XV^T \tag{5}$$

Where $y \in \mathbb{R}^l$ is now a vector of indices representing the next tokens in the sequence. $V \in \mathbb{R}^{v \times e}$ in equation (4) is the llm's vocabulary matrix.

This operation is typically computed following algorithm 1:

Algorithm 1 naive GEMM softmax cross entropy loss

$S \leftarrow \text{matmul}(X, V^T)$
$m \leftarrow \max_v(S)$
$S \leftarrow S - m$
$\text{gold} \leftarrow \text{take}_v(S, y)$
$\text{loss} \leftarrow -\log(\text{gold}) + \text{logsumexp}_v(S)$

The subtraction of the max in algorithm 1 is done for numerical stability. Typically this algorithm is implemented using an ML framework (i.e. pytorch, tensorflow, jax) and for the most part results in separate kernel launches for each operation.

One major downside of implementing this operation with separate kernel launches comes from line 1 of algorithm 1. Specifically, storing S in global memory can be an issue because of its large dimensions. For small language model training on a single gpu, typical values for v, l, and e are $v = 50257, l = 20480$, and $e = 768$. With these values, S has size 50257×20480. When using gradient checkpointing, the size of S is almost always a memory bottleneck for the training process.

The approach I took for this project is to fuse the GEMM operation with softmax cross entropy loss function so that the matrix S never has to be saved in global memory. The approach I took is show in algorithm 2

In algorithm 2 we fuse the normal tiled GEMM operation with softmax cross entropy loss. This is achieved by computing an online softmax on the GEMM tiles. This can be achieved by keeping running statistics for the max and normalizer values (a similar trick is performed in the flash attention paper).

Algorithm 2 fused GEMM softmax cross entropy loss

Divide X into N tiles sized (BL,E)
Divide V into M tiles sized (BV,E)
Create normalizer_arr sized (N,BL)
Create max_arr sized (N,BV)
for $i \leftarrow 1 \dots N$ **do**
    load X_i into smem
    Initialize running_max $\leftarrow (-\infty)_{BL} \in \mathbb{R}^{BL}$
    Initialize running_normalizer $\leftarrow (0)_{BL} \in \mathbb{R}^{BL}$
    for $j \leftarrow 1 \dots M$ **do**
        load V_j into smem
        $C \leftarrow \text{GEMM}(X_i, V_j)$
        local_max $\leftarrow \max_v(C)$
        $C \leftarrow C -$local_max
        local_normalizer $\leftarrow \sum_v \exp(C_v)$
        new_running_max $\leftarrow$ max(running_max, local_max)
        $\alpha_l \leftarrow \exp($new_running_max $-$ local_max$)$
        $\alpha_r \leftarrow \exp($new_running_max $-$ running_max$)$
        running_normalizer $\leftarrow \alpha_l$local_normalizer $+ \alpha_r$running_normalizer
        running_max $\leftarrow$ new_running_max
    end for
    normalizer_arr$[i] \leftarrow$ running_normalizer
    max_arr$[i] \leftarrow$ running_max
end for
normalizer_arr$\leftarrow$ flatten(normalizer_arr)
max_arr$\leftarrow$ flatten(max_arr)
gold $\leftarrow \text{dot}(\text{take}_v(V, y), X) -$ max_arr
loss $\leftarrow -\log($gold$) + \log($normalizer_arr$)$