Liger Kernel

Triton In Real Life

Outline

1. LLM training bottleneck
2. Why use Triton?
3. Example 1: RMS Norm
4. Example 2: Fused Linear Cross Entropy
5. Tip1: Convergence Test
6. Tip2: Contiguity
7. Tip3: Address Range
8. Liger Kernel
9. Acknowledgement

LLM training bottleneck

1. Constant and unexpected OOM
2. My GPU util is 100% but it is still slow
    - Understanding NVIDIA GPU Performance: Utilization vs. Saturation (2023)
1. Profiler is the only key to understanding performance
    - Lecture 1 How to profile CUDA kernels in PyTorch
    - Lecture 16: On Hands Profiling

Let’s do a bit of profiling (live demo)

Time

- Many slow elementwise ops
- Many cuda launch overhead

Memory

- Cross entropy memory spike

Why Triton?

- Easier programming: Finish developing a kernel much faster than CUDA.
- Think in Numpy: Think in vectors (blocks) instead of elements (threads).
- Easier collaboration with AI researchers: AI researchers can easily pick up and extend.
- Python native: No need for five different file types just for one kernel.
- Clean dependency: Triton just works in most cases.

RMSNorm


Yahh writing triton is so easy!!
Trying to derive backprop
imgflip.com

Backprop 101

- Think element by element: scalar calculus is easier to derive than vector calculus
- Brush up on your calculus 101
- Matrix-Matrix product formula

In summary, we have derived the backpropagation expressions for the matrix-matrix product $Y = XW$:

$$\boxed{\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} W^T} \qquad \boxed{\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial Y}} \tag{26}$$

RMSNorm - backprop

$$y_i = \frac{x_i}{\sqrt{\frac{1}{n}\sum_k x_k^2}} w_i$$

$$dx_i = \frac{\partial o}{\partial x_i} = \sum_k \frac{\partial o}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_i}$$

Chain Rule: Because xi contributes to all yi

RMSNorm - backprop

Separate k = i and k != i

$$\text{if } k = i, \quad \frac{\partial y_i}{\partial x_i} = \left(\frac{\left(\frac{1}{n}\Sigma_k x_k^2\right)^{\frac{1}{2}} - \frac{1}{2}\left(\frac{1}{n}\Sigma_k x_k^2\right)^{\frac{-1}{2}}\left(\frac{2}{n}x_i\right)x_i}{\frac{1}{n}\Sigma_k x_k^2} \right) w_i$$

$$\text{let } RMS = \sqrt{\frac{1}{n}\Sigma_k x_k^2},$$

$$\frac{\partial y_i}{\partial x_i} = \left(\frac{RMS - \frac{1}{RMS} \cdot \frac{1}{n} \cdot x_i^2}{RMS^2} \right) w_i = \frac{w_i - \frac{1}{RMS^2} \cdot \frac{1}{n} \cdot x_i^2 \cdot w_i}{RMS}$$

RMSNorm - backprop

$$\text{if } k \neq i, \quad \frac{\partial y_j}{\partial x_i} = \left(\frac{\frac{-1}{2}(\frac{1}{n}\Sigma_k x_k^2)^{\frac{-1}{2}}(\frac{2}{n}x_i)x_j}{\frac{1}{n}\Sigma_k x_k^2} \right) w_j$$

$$= \frac{-\frac{1}{RMS} \cdot \frac{1}{n} \cdot x_i \cdot x_j \cdot w_j}{RMS^2} = \frac{-\frac{1}{RMS^2} \cdot \frac{1}{n} \cdot x_i \cdot x_j \cdot w_j}{RMS}$$

RMSNorm - backprop

$$\partial x_i = \frac{\partial o}{\partial x_i} = \frac{\partial o}{\partial y_i}\frac{\partial y_i}{\partial x_i} + \sum_{j \neq i} \frac{\partial o}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

$$= \frac{1}{RMS}\left[w_i \cdot \partial y_i - \sum_j \frac{1}{RMS^2} \cdot \frac{1}{n} \cdot x_i \cdot x_j \cdot w_j \cdot \partial y_j\right]$$

vector notation:

$$\partial \vec{x} = \frac{1}{RMS}\left[\partial \vec{y} \cdot \vec{w} - \frac{1}{n} \cdot \frac{1}{RMS^2} \cdot ((\partial \vec{y} \cdot \vec{w}) \odot \vec{x}) \cdot \vec{x}\right]$$

Surprisingly Clean!

RMSNorm - tricks

1. In-place tensor reuse: reuse dY to store the value of dX to save memory
2. Cache rms: save flops by caching relatively small tensor

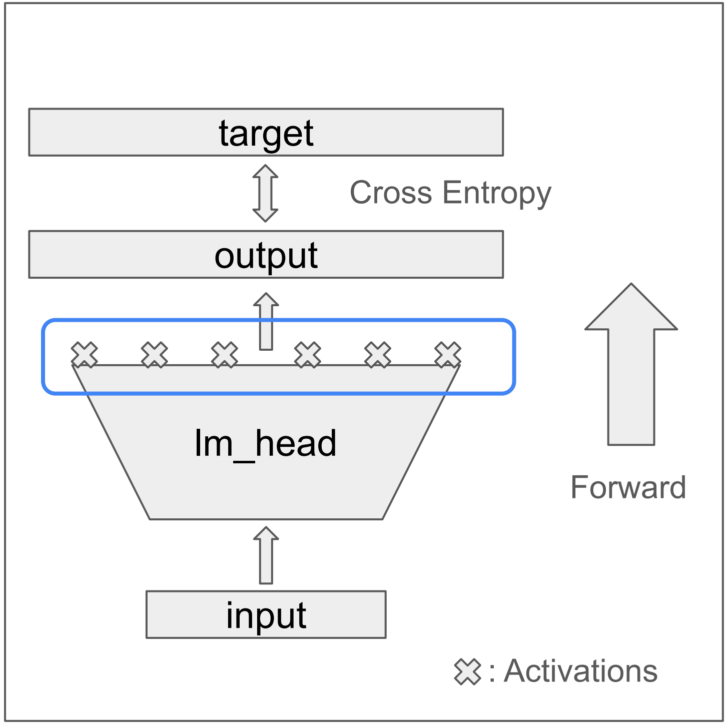
https://github.com/linkedin/Liger-Kernel/blob/main/src/liger_kernel/ops/rms_norm.py

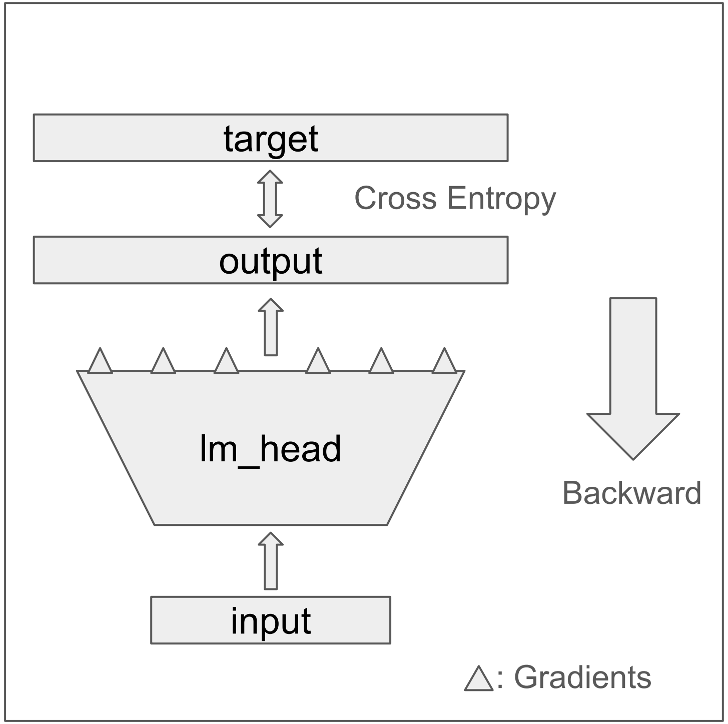
RMSNorm - live coding!

1. Correctness test: Ensure that the kernel is as precise as the original implementation. Any deviation could impact model convergence or even cause critical errors.
2. Performance test: Confirm that the kernel is more time- and memory-efficient than the original version. Without these improvements, re-implementing in Triton would not be justified.

RMSNorm: Verifying Correctness and Performance (Google Collab)

Fused Linear Cross Entropy





Problem: Large Vocab Size!!

Linear layer gradient computation

$$\vec{y} = W\vec{x}$$

by following a similar setup like RMS norm, we get

$$\frac{\partial o}{\partial \vec{x}} = W^T \partial \vec{y}$$

Cross entropy gradient computation

$$l = -\sum_{j}^{C} y_j \log \frac{\exp(x_j)}{\sum_{i=1}^{C} \exp(x_i)}$$

$$\frac{\partial l}{\partial x_k} = \frac{\partial}{\partial x_k}\left(-y_k \log \frac{\exp(x_k)}{\sum_{i=1}^{C} \exp(x_i)}\right) + \frac{\partial}{\partial x_k}\left(-\sum_{j \neq k}^{C} y_j \log \frac{\exp(x_j)}{\sum_{i=1}^{C} \exp(x_i)}\right)$$

First term,

$$
\begin{aligned}
&\frac{\partial}{\partial x_k}\left(-y_k \log \frac{\exp(x_k)}{\sum_{i=1}^{C}\exp(x_i)}\right) \\
&= -y_k \frac{\sum_{i=1}^{C}\exp(x_i)}{\exp(x_k)} \cdot \left(\frac{\left(\sum_{i=1}^{C}\exp(x_i)\exp(x_k) - (\exp(x_k))^2\right.}{\left(\sum_{i=1}^{C}\exp(x_i)\right)^2}\right) \\
&= \frac{-y_k\left(\sum_{i=1}^{C}\exp(x_i) - \exp(x_k)\right)}{\sum_{i=1}^{C}\exp(x_i)}
\end{aligned}
$$

Second term,

$$\frac{\partial}{\partial x_k}\left(-\sum_{j\neq k}^{C} y_j \log \frac{\exp(x_j)}{\sum_{i=1}^{C}\exp(x_i)}\right)$$

$$= -\sum_{j\neq k}^{C} y_j \frac{\sum_{i=1}^{C}\exp(x_i)}{\exp(x_j)}\left(\frac{-\exp(x_j)\exp(x_k)}{\left(\sum_{i=1}^{C}\exp(x_i)\right)^2}\right)$$

$$= \sum_{j\neq k}^{C} y_j \frac{\exp(x_k)}{\sum_{i=1}^{C}\exp(x_i)}$$

Finally,

$$\frac{\partial l}{\partial x_k} = -y_k + \sum_{j}^{C} y_j \frac{\exp(x_k)}{\sum_{i=1}^{C} \exp(x_i)} = -y_k + \mathrm{softmax}(x_k)$$

$$\text{when } y_k = 1, \quad \frac{\partial l}{\partial x_k} = -1 + \mathrm{softmax}(x_k)$$

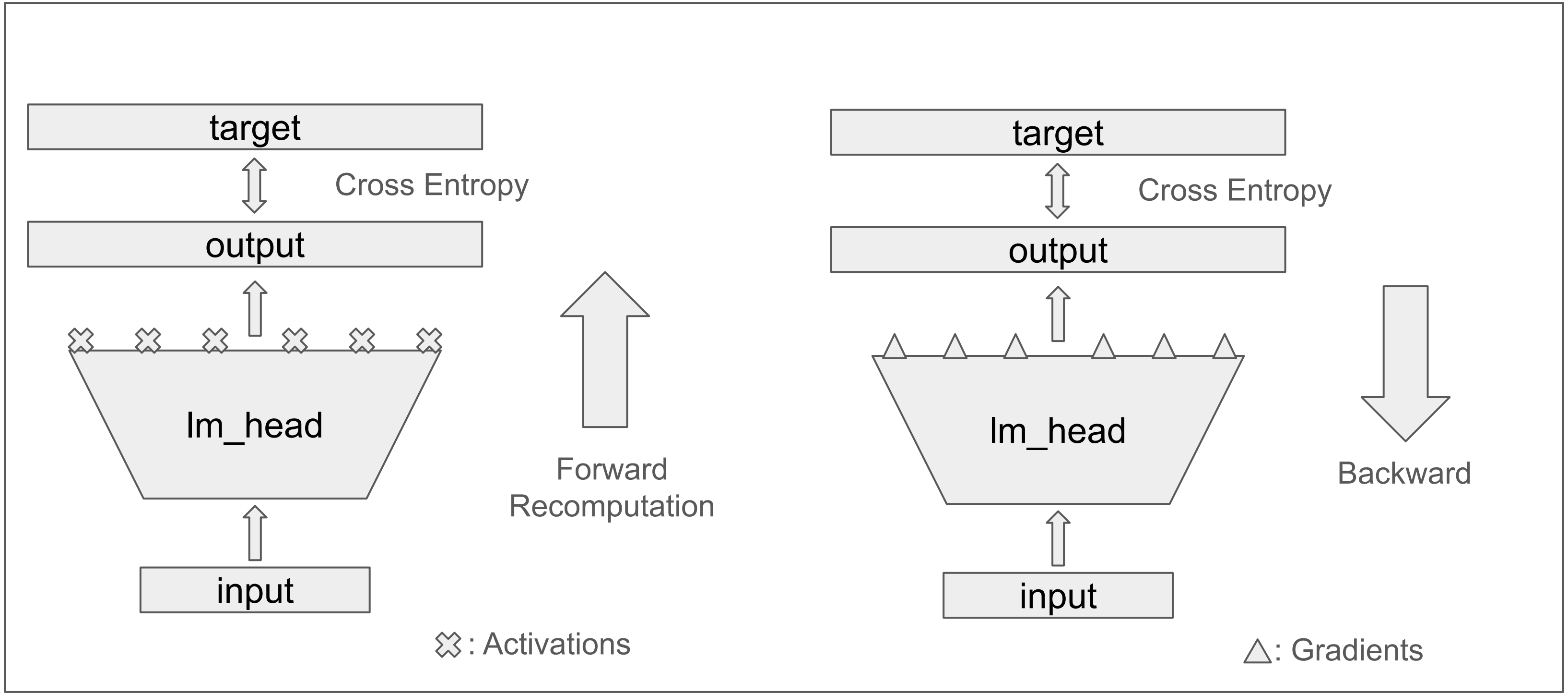
$$\text{when } y_k = 0, \quad \frac{\partial l}{\partial x_k} = \mathrm{softmax}(x_k)$$

Fused Linear Cross Entropy

Three levels of optimization

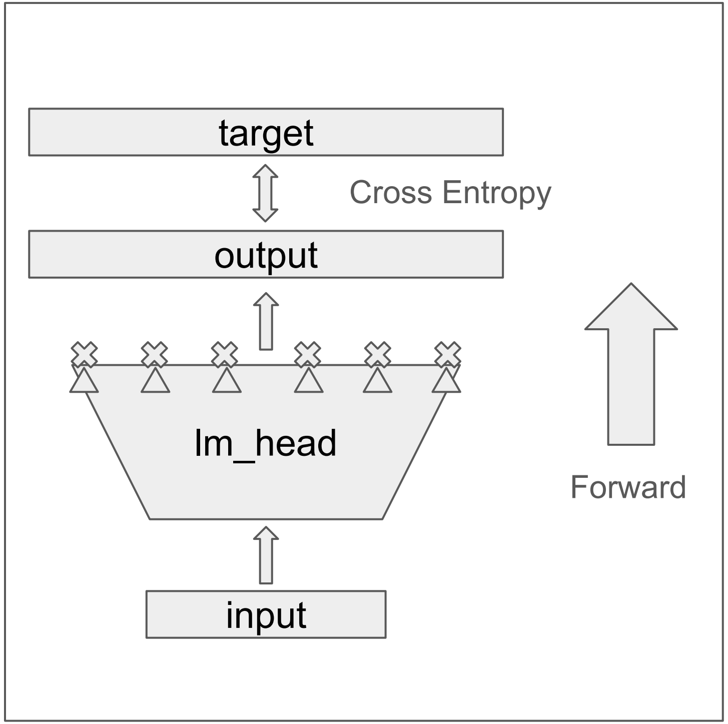
1. Gradient checkpointing
2. Gradient-in-forward
3. Chunking

Fused Linear Cross Entropy - Gradient Checkpointing



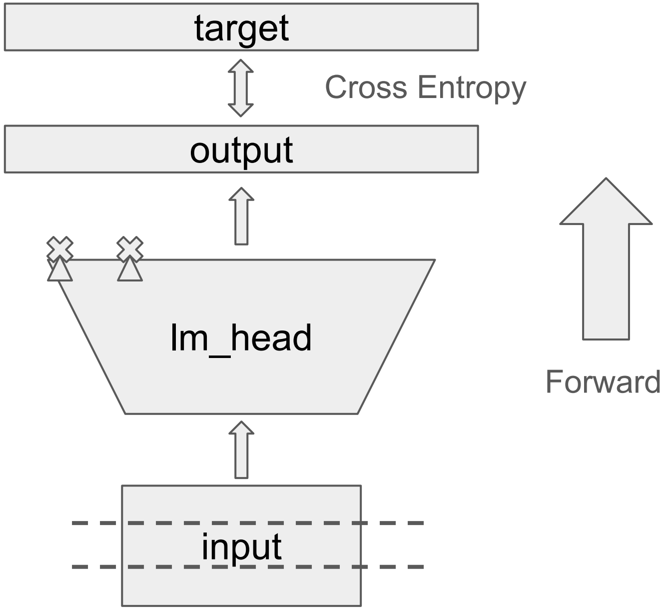
Recompute forward again in backward, so we don't have to persist activation

Fused Linear Cross Entropy - Gradient-in-forward



We can get rid of forward recomputation by computing gradient at forward pass

Fused Linear Cross Entropy - Chunking



Ingest the input chunk-by-chunk, so we only materialize a chunk of activations/gradients at a moment

Fused Linear Cross Entropy - Live Coding

[FusedLinearCrossEntropy: Verifying Memory Reduction (Google Collab)]

Convergence Test: compare layer-by-layer

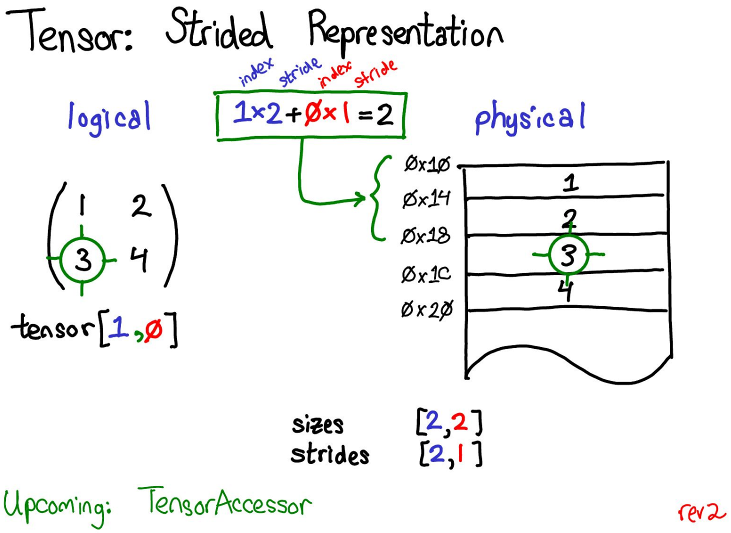
- Unit correctness and performance test is not sufficient for production use
- The contiguity, the tensor shape, and the dtype might be different in production
- Mimic the real production training to validate the logits, weights, and loss

Convergence Comparison: Triton Kernel Patched vs. Original Model Layer-by-Layer (Google Collab)

Contiguity is the hidden killer

- Contiguity can cause silent bug and takes you hours to debug
- Understand logical v.s. physical view

[Contiguity is the hidden killer (Google Collab)]



[PyTorch internals : ezyang's blog]

Illegal memory access due to overflow

- Program id is stored in int32
- If program_id x stride >= max(int32), you need to use int64

[Address Overflow (Google Collab)]

Liger Kernel: Efficient Triton Kernels for LLM Training

https://github.com/linkedin/Liger-Kernel

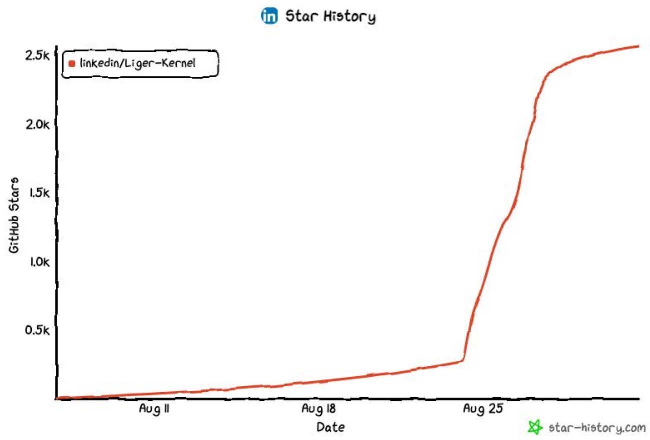
Stable		Nightly		Discord
downloads 9k	pypi v0.2.1	downloads 23k	pypi v0.2.1.dev20240831160751	CUDA MODE 8633 members

LIGER KERNEL

Installation | Getting Started | Examples | APIs | Structure | Contributing

▶ Latest News 🔥

Liger (Linkedin GPU Efficient Runtime) Kernel is a collection of Triton kernels designed specifically for LLM training. It can effectively increase multi-GPU **training throughput by 20%** and reduces **memory usage by 60%**. We have implemented **Hugging Face Compatible** `RMSNorm`, `RoPE`, `SwiGLU`, `CrossEntropy`, `FusedLinearCrossEntropy`, and more to come. The kernel works out of the box with Flash Attention, PyTorch FSDP, and Microsoft DeepSpeed. We welcome contributions from the community to gather the best kernels for LLM training.



OSS Reception

Andrej Karpathy @karpathy

looks very nice! makes me want to write a 100% triton nanoGPT :)

kalomaze @kalomaze

from @intervitens
> previously I was maxing out vram with cpu param offload, and now even without offloading, I only get 75% vram usage

It actually... just works™
(4b FFT on 4x3090s, bs1 @ 8192 context)

Mitko Vasilev • 1st
CTO
2h •

Me: Linkedin I want a better feed algorithm
Linkedin: The best I can give you is Liger (LinkedIn GPU Efficient Runtime)

LinkedIn open-sourceing Triton kernels for LLM fine-tuning was definitely not on my 2024 GenAI bingo card. The LinkedIn ML Infrastructure team, inspired by llm.c, Unsloth, and FlashAttention, etc cool open-source projects have developed experimental Triton kernels that reduce memory usage by half during LLM training. I like small teams in big corp coding cool stuff and then going through all the bureaucracy to open-source it under a BSD license. Very cool LinkedIn ML infra team!

Yam Peleg @Yampeleg

What is the most random event you can make up right now?

Exactly!

LinkedIN just dropped the highest performance gpu kernels (triton) for training LLMs.

(WTF?? 😅)

Throughout up by up to 20%
Mem reduced by up to 60%

Out of the box support HF models.

github.com/linkedin/Liger...

Acknowledgement

- @claire_yishan for the LOGO design
- Unsloth and flash-attn for inspiration in Triton kernels for training
- tiny shakespeare dataset by Andrej Karpathy for convergence testing
- Efficient Cross Entropy for lm_head + cross entropy inspiration
- The AWESOME CUDA (Triton? lol) MODE community <3
- We would like to thank Animesh Singh Haowen Ning Yanning Chen for the leadership support, Shao Tang Qingquan Song Yun Dai Vignesh Kothapalli Jason Zhu Steven Shimizu Shivam Sahni zmerchant@linkedin.com for the technical contribution.