# Optimizing Your PyTorch Code: Unlocking Performance and Efficiency

A Guide to Boosting Speed and Resource Usage

Hovhannes Tamoyan



### DataLoader

Multi-process Data Loading

- DataLoader uses a single-process by default.
- blocking computation code with data loading.

More: <u>https://pytorch.org/docs/stable/data.html</u>

• The GIL prevents true fully parallelizing Python code across threads ->

• To perform multi-process data loading set the argument num\_workers= $R_+$ .

## DataLoader

Multi-process Data Loading

- from the worker processes.
- memory usage is number of workers \* size of parent process).
- such as Pandas, Numpy or PyArrow objects.

More: <a href="https://github.com/pytorch/pytorch/issues/13246">https://github.com/pytorch/pytorch/issues/13246</a>

• After several iterations, the loader worker processes will consume the same amount of CPU memory as the parent process for all Python objects in the parent process which are accessed

• This can be problematic if the Dataset contains a lot of data (e.g., you are loading a very large list of filenames at Dataset construction time) and/or you are using a lot of workers (overall

• The simplest workaround is to replace Python objects with non-refcounted representations



### DataLoader

Memory Pinning

- To speed up the host the Dataset transfer from CPU to GPU enable pin\_memory.
- This lets the DataLoader allocate the samples in page-locked/ pinned memory, which speeds-up the transfer to GPU.
- To put the fetched data tensors in pinned memory set pinn\_memory=True

More: <u>https://pytorch.org/docs/stable/data.html</u>



## **DistributedDataParallel**

DistributedDataParallel

- DistributedDataParallel uses multiprocessing where a process is created for each GPU
- DataParallel uses multithreading.
- During multiprocessing, each GPU has its dedicated process, this avoids the performance overhead caused by GIL.
- Recommended to use DistributedDataParallel, instead of DataParallel to do multi-GPU training, even if there is only a single node.
- Use torch.distributed.launch utility to launch your program utilizing DistributedDataParallel.

More: <u>https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html</u>

More: <u>https://pytorch.org/docs/stable/distributed.html</u>

### Gradients

Inference Mode

- Inference code run under this mode gets better performance by disabling view tracking and version counter bumps
- Make sure your operations will have no interactions with autograd
- Note that unlike some other mechanisms that locally enable or disable grad, entering inference\_mode also disables forward-mode AD.

More: <u>https://pytorch.org/docs/stable/generated/torch.inference\_mode.html</u>



### Gradients

No grad Mode

with torch.no\_grad():

x = torch.randn(1)

y = x + 1

y.requires\_grad = True

z = y + 1

print(z.grad\_fn)

> <AddBackward0 object at 0x7fe9c6eafdf0>

Inference Mode with torch.inference\_mode(): x = torch.randn(1)y = x + 1

### y.requires\_grad = True

> RuntimeError: Setting requires\_grad=True on inference tensor outside InferenceMode is not allowed.



## Gradients

Set grad to None

- not 0.
- Leads to a lower memory footprint and modestly faster performance.
- Caveats apply:
  - will behave differently.
  - receive a gradient.
  - a gradient of 0 and in the other it skips the step altogether).

More: <a href="https://pytorch.org/docs/stable/generated/torch.optim.Optimizer.zero\_grad.html">https://pytorch.org/docs/stable/generated/torch.optim.Optimizer.zero\_grad.html</a>

• Pass an additional argument set\_to\_none=True when calling optimizer.zero\_grad() to set the grade to None and

• When the user tries to access a gradient and perform manual ops on it, a None attribute or a Tensor full of Os

• The operation followed by a backward pass, guarantees the .grads to be None for params that did not

o torch.optim optimizers have a different behavior if the gradient is 0 or None (in one case it does the step with

## **Operator Fusing**

- Pointwise operations
- launches a separate kernel.
- call).

More: <u>https://pytorch.org/docs/stable/jit.html#</u>

Pointwise operations are memory-bound, for each operation PyTorch

• Use the torch.jit to fuse pointwise operators into a single operator (kernel

• Fused operator launches only one kernel for multiple fused pointwise ops.

## **Operator Fusing**

### Optimizers

- fusing vertically on top of that.
- fused > foreach > for-loop.
- supported optimizers: FuseAdam, FuseLAMBD, FusedNovoGrad, FusedSGD.

More: <u>https://pytorch.org/docs/stable/optim.html</u>

• PyTorch has 3 major categories of oprimizers: for-loop, foreach (multi-tensor), and fused. • Think of foreach implementations as fusing horizontally and fused implementations as

## Checkpoint intermediate buffers

Intermediate layer storing

- For the backward pass, store the inputs of a few layers and recompute others during the backward pass.
- This reduces the memory requirements, and enables increasing the batch size

More: <u>https://pytorch.org/docs/stable/checkpoint.html</u>

## **Avoid CPU-GPU Synchronizations**

Avoid operations that requires synchronization such as:

- print(cuda\_tensor)
- cuda\_tensor.item()
- cuda\_tensor.cpu()
- python control flow which depends on tl tensors e.g. if (cuda\_tensor ≠ 0).all()

More: <u>https://pytorch.org/tutorials/recipes/recipes/tuning\_guide.html#avoid-unnecessary-</u> <u>cpu-gpu-synchronization</u>

• python control flow which depends on the results of operations performed on cuda

## **Use Mixed Precision and AMP**

Mixed Precision

- Some operations use the torch.float32 data type and other operations use torch.float16.
- Some operations, such as linear layers and convolutions are much faster in float16.
- Other operations like reductions often require the dynamic range of float32.
- AMP tries to match each op to its appropriate data type.

More: <u>https://pytorch.org/docs/stable/amp.html</u>

## bfloat16 Data Type

### bfloat16

- Neural networks are more sensitive to the size of the exponent than the size of the mantissa.
- Provides identical behavior for underflows, overflows, and NaNs.
- bfloat16 is a drop-in replacement for float32 when training and running deep neural networks.

### More: <u>https://cloud.google.com/tpu/docs/</u> <u>bfloat16</u>

### **Floating Point Formats**

### bfloat16: Brain Floating Point Format

Exponent: 8 bits							Mantissa (Significand): 7 bits							
Е	Е	Е	Е	Е	Е	E	м	м	М	м	м	м	м	

### fp32: Single-precision IEEE Floating Point Format



### fp16: Half-precision IEEE Floating Point Format

 Exponent: 5 bits
 Mantissa (Significand): 10 bits

 S
 E
 E
 E
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M
 M<

Range: ~5.96e<sup>-8</sup> to 65504

Range: ~1e<sup>-38</sup> to ~3e<sup>38</sup>

Range: ~1e<sup>-38</sup> to ~3e<sup>38</sup>



# Profilers

Hovhannes Tamoyan

A Guide to Boosting Speed and Resource Usage

## **PyTorch Profiler**

torch.profiler

- during training and inference.

More: <u>https://pytorch.org/docs/stable/profiler.html</u>

• PyTorch Profiler is a tool that allows the collection of performance metrics Profiler's context manager API can be used to better understand what model operators are the most expensive, examine their input shapes and stack traces, study device kernel activity and visualize the execution trace.

### **Trace Viewers**

### Tools

- chrome://tracing
- https://www.speedscope.app/
- <u>https://www.tensorflow.org/</u> <u>tensorboard/</u> <u>tensorboard\_profiling\_keras</u>

	🙆 Time Order 💽 Left	Heavy 婱 Sandwich	python3 (pid 5815), thread 5815 (python3) (tid 5815) (2/3)
	🗢 Total	🔷 Self	Symbol Name
	17.90ms (31%)	17.90ms (31%)	cudaDeviceSynchronize {"External id":8132,"cbid":165,"correlation":8132}
	39.36ms (69%)	12.33ms (22%)	<pre>model_inference {"External id":2049,"Ev Idx":0}</pre>
	1.31ms (2.3%)	1.31ms (2.3%)	aten::resize_ {"External id":2099,"Ev Idx":50,"Input Dims":[[0],[],"Input type":["float","",""]}
	1.12ms (2.0%)	1.10ms (1.9%)	aten::copy_ {"External id":2410,"Ev Idx":361,"Input Dims":[[16,12,64,40],[16,12,64,40],[]],"Input type":["float","float","Scalar"]}
	1.02ms (1.8%)	1.02ms (1.8%)	aten::empty {"External id":2051,"Ev Idx":2,"Input Dims":[[],[],[],[],[],[],"Input type":["","Scalar","","","Scalar",""]}
	238.00µs (0.42%)	169.00µs (0.30%)	aten::addmm {"External id":2115,"Ev Idx":66,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd thre
	173.00µs (0.30%)	128.00µs (0.22%)	aten::addmm {"External id":3322,"Ev Idx":1273,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd th
	134.00µs (0.23%)	99.00µs (0.17%)	aten::addmm {"External id":3197,"Ev Idx":1148,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd th
	98.00µs (0.17%)	98.00µs (0.17%)	cudaLaunchKernel {"External id":4629,"cbid":211,"correlation":4629}
	130.00µs (0.23%)	97.00μs (0.17%)	aten::addmm {"External id":2214,"Ev Idx":165,"Input Dims":[[3072],[640,768],[768,3072],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd t
	113.00µs (0.20%)	96.00µs (0.17%)	aten::add {"External id":2977,"Ev Idx":928,"Input Dims":[[16,40,768],[16,40,768],[]],"Input type":["float","float","Scalar"],"Fwd thread id":0,"Sequence numb
	156.00µs (0.27%)	93.00µs (0.16%)	aten::addmm {"External id":3618,"Ev Idx":1569,"Input Dims":[[768],[16,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd thr
	128.00µs (0.22%)	92.00µs (0.16%)	aten::addmm {"External id":2490,"Ev Idx":441,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd thr
	120.00µs (0.21%)	88.00µs (0.15%)	aten::addmm {"External id":3072,"Ev Idx":1023,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd th
	121.00µs (0.21%)	86.00µs (0.15%)	aten::addmm {"External id":3134,"Ev Idx":1085,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd th
	132.00µs (0.23%)	86.00µs (0.15%)	aten::addmm {"External id":2447,"Ev Idx":398,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd thr
	130.00µs (0.23%)	85.00μs (0.15%)	aten::addmm {"External id":2589,"Ev Idx":540,"Input Dims":[[3072],[640,768],[768,3072],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd t
	118.00µs (0.21%)	83.00µs (0.14%)	aten::addmm {"External id":2464,"Ev Idx":415,"Input Dims":[[3072],[640,768],[768,3072],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd t
	111.00µs (0.19%)	83.00µs (0.14%)	aten::addmm {"External id":2322,"Ev Idx":273,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd thr
	120.00µs (0.21%)	80.00µs (0.14%)	aten::addmm {"External id":2248,"Ev Idx":199,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd thr
	109.00µs (0.19%)	79.00µs (0.14%)	aten::addmm {"External id":3498,"Ev Idx":1449,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd th
	115.00µs (0.20%)	79.00µs (0.14%)	aten::addmm {"External id":2259,"Ev Idx":210,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd thr
	110.00µs (0.19%)	78.00µs (0.14%)	aten::addmm {"External id":3339,"Ev Idx":1290,"Input Dims":[[3072],[640,768],[768,3072],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd
	110.00µs (0.19%)	78.00µs (0.14%)	aten::addmm {"External id":2884,"Ev Idx":835,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd thr
	112.00µs (0.20%)	78.00µs (0.14%)	aten::addmm {"External id":2723,"Ev Idx":674,"Input Dims":[[768],[640,3072],[3072,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd t
	134.00µs (0.23%)	78.00µs (0.14%)	aten::addmm {"External id":2240,"Ev Idx":191,"Input Dims":[[768],[640,768],[768,768],[],[]],"Input type":["float","float","float","Scalar","Scalar"],"Fwd thr
_	107.00us (0.19%)	77.00us (0.13%)	aten::addmm {"External id":2839."Ev Idx":790."Input Dims": [[3072].[640.768].[768.3072].[].[]]."Input type": ["float"."float"."float"."Scalar"."Scalar"]."Fwd t

TensorBoard	PYTORCH_PROFILER										D G	-
	<	Operator Viev	V									
NORMAL	DIFF	All operators ( Top operators to show 10										
Runs		Host Self Time (us) ⑦ Host Total Time (us) ⑦										
bert-base-cased	•				<ul> <li>aten::addmi</li> <li>aten::empty</li> </ul>	m ,				aten::linear	Im	
Views					<ul> <li>aten::add</li> <li>aten::copy_</li> <li>aten::native</li> </ul>	_layer_norm		5%	32.9%	<ul> <li>aten::matm</li> <li>aten::empty</li> <li>aten::resha</li> </ul>	ul / upe	
Operator		5.6%		53.4%	<ul> <li>aten::bmm</li> <li>aten::view</li> <li>aten::linear</li> </ul>		5. 5.	3% 9%		<ul> <li>aten::clone</li> <li>aten::layer_</li> <li>aten::native</li> </ul>	_norm e_layer_norm	n
Workers		6.1%	11.9%		● aten::t ● aten::gelu			9.2%		<ul><li>aten::ones</li><li>aten::add</li></ul>		
cad58c8e3c23_459	•							27.4	%			
				Group By Operator -				Search b	y Name			
		Name		Device Self Duration (us)	Device Total Duration (us)	Host Self Duration (us)	Host Total Duration (us)	Tensor Cores Eligible	Tensor Cores Self(%)	Tensor Cores Total(%)		
		aten::empty	201	0	0	1966	1966	No	0	0	View Ca	allStack
		aten::fill_	1	0	0	155	155	No	0	0	View Ca	allStack
		aten::ones	1	0	0	47	1040	No	0	0	View Ca	allStack

id":0,"Sequence number": id":0,"Sequence number". d":0,"Sequence number":94... :0,"Sequence number":74... id":0,"Sequence number":1... id":0,"Sequence number". j":0,"Sequence number":11. id":0,"Sequence number":... id":0,"Sequence number":1... id":0,"Sequence number":... j":0,"Sequence number":97... i":0,"Sequence number":93... i":0,"Sequence number":91... 1 id":0,"Sequence number"... id":0,"Sequence number":1... 

hread id":0,"Sequence number": er":1098} ead id":0,"Sequence number":13... ead id":0,"Sequence number":87... read id":0,"Sequence number":1... read id":0,"Sequence number":1.. ad id":0,"Sequence number":85.. read id":0,"Sequence number": read id":0,"Sequence number":. ead id":0,"Sequence number":80. ead id":0,"Sequence number":76... read id":0,"Sequence number":1... ead id":0,"Sequence number":77. thread id":0,"Sequence number". ad id":0,"Sequence number":10. read id":0,"Sequence number": ead id":0,"Sequence number":76...

ead id":0,"Sequence number":703} nread id":0,"Sequence number":1… nread id":0,"Sequence number":1…

ort 👒 System 🕴 Help

Hovhannes Tamoyan

# Extras

A Guide to Boosting Speed and Resource Usage

## Debugging Parallel Ops

CUDA\_LAUNCH\_BLOCKING=1.



### You can force synchronous computation by setting environment variable

## Annotate Tensor Shapes

class NN:

embedding: "(V, E)" = torch.zeros(V, E)

assert(str(tuple(var.shape)) == NN.\_\_\_annotations\_\_\_["embedding"])

More: <u>https://github.com/ofnote/tsalib</u>

## cuDNN Benchmark

A bool that, if True, causes cuDNN to enable the inbuilt auto-tuner to find the best algorithm to use for your hardware: torch.backends.cudnn.benchmark

# Thank You

Hovhannes Tamoyan