Profiling and Tuning PyTorch Models

@shagunsodhani

FACEBOOK AI

Who am I

1. Research Engineer at Facebook AI Research

2. Focusing on lifelong learning and reinforcement learning

3. Using and talking about PyTorch for about 3 years now

What is PyTorch

PyTorch is a Python package that provides two high-level features:

1. Tensor computation (like NumPy) with strong GPU acceleration
2. Deep neural networks built on a tape-based autograd system

Note: This slide is based on PyTorch 1.9.1

PyTorch Profiling 101

Setup PyTorch Profiler

1. No new library needed as profiler is built in PyTorch

2. `from torch import profiler`

3. Further Reading

Setup PyTorch Profiler

```
model = torchvision.models.resnet18()
inputs = torch.randn(5, 3, 224, 224)

with profiler.profile(activities=[profiler.ProfilerActivity.CPU], record_shapes=True) as prof:
    with profiler.record_function("model_inference"):
        model(inputs)
```

Measure

```
print(prof.key_averages().table(sort_by="cpu_time_total", row_limit=10))
```

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
model_inference	3.60%	26.986ms	99.83%	749.335ms	749.335ms	1
aten::conv2d	0.04%	265.000us	71.15%	534.099ms	26.705ms	20
aten::convolution	0.03%	258.000us	71.12%	533.834ms	26.692ms	20
aten::_convolution	0.07%	503.000us	71.08%	533.576ms	26.679ms	20
aten::mkldnn_convolution	70.90%	532.177ms	71.02%	533.073ms	26.654ms	20
aten::batch_norm	0.49%	3.642ms	13.25%	99.429ms	4.971ms	20
aten::_batch_norm_impl_index	0.06%	467.000us	12.76%	95.787ms	4.789ms	20
aten::native_batch_norm	7.48%	56.169ms	12.69%	95.247ms	4.762ms	20
aten::max_pool2d	0.01%	40.000us	7.33%	54.991ms	54.991ms	1
aten::max_pool2d_with_indices	7.32%	54.951ms	7.32%	54.951ms	54.951ms	1

Self CPU time total: 750.624ms

Measure

```
print(prof.key_averages().table(sort_by="cpu_time_total", row_limit=10))
```

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
model_inference	3.60%	26.986ms	99.83%	749.335ms	749.335ms	1
aten::conv2d	0.04%	265.000us	71.15%	534.099ms	26.705ms	20
aten::convolution	0.03%	258.000us	71.12%	533.834ms	26.692ms	20
aten::_convolution	0.07%	503.000us	71.08%	533.576ms	26.679ms	20
aten::mkldnn_convolution	70.90%	532.177ms	71.02%	533.073ms	26.654ms	20
aten::batch_norm	0.49%	3.642ms	13.25%	99.429ms	4.971ms	20
aten::_batch_norm_impl_index	0.06%	467.000us	12.76%	95.787ms	4.789ms	20
aten::native_batch_norm	7.48%	56.169ms	12.69%	95.247ms	4.762ms	20
aten::max_pool2d	0.01%	40.000us	7.33%	54.991ms	54.991ms	1
aten::max_pool2d_with_indices	7.32%	54.951ms	7.32%	54.951ms	54.951ms	1

Self CPU time total: 750.624ms

Optimize the setup

```
model = torchvision.models.resnet18()
inputs = torch.randn(5, 3, 224, 224)

with profiler.profile(activities=[profiler.ProfilerActivity.CPU], record_shapes=True) as prof:
    with profiler.record_function("model_inference"):
        with torch.inference_mode():
            model(inputs)
```

Optimize the setup

```
model = torchvision.models.resnet18()
inputs = torch.randn(5, 3, 224, 224)

with profiler.profile(activities=[profiler.ProfilerActivity.CPU], record_shapes=True) as prof:
    with profiler.record_function("model_inference"):
        with torch.inference_mode():
            model(inputs)
```

Measure

```
print(prof.key_averages().table(sort_by="cpu_time_total", row_limit=10))
```

```
---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                             Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                  model_inference         1.53%       7.094ms        99.99%     465.110ms     465.110ms             1
                     aten::conv2d         0.05%     216.000us        72.52%     337.325ms      16.866ms            20
                aten::convolution         0.05%     234.000us        72.47%     337.109ms      16.855ms            20
               aten::_convolution         0.10%     445.000us        72.42%     336.875ms      16.844ms            20
        aten::mkldnn_convolution        72.23%     335.997ms        72.33%     336.430ms      16.822ms            20
              aten::batch_norm          0.05%     233.000us        15.56%      72.384ms       3.619ms            20
    aten::_batch_norm_impl_index         0.07%     304.000us        15.51%      72.151ms       3.608ms            20
         aten::native_batch_norm        10.92%      50.772ms        15.43%      71.789ms       3.589ms            20
              aten::max_pool2d          0.02%     107.000us         8.81%      40.962ms      40.962ms             1
    aten::max_pool2d_with_indices        8.78%      40.855ms         8.78%      40.855ms      40.855ms             1
---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 465.157ms
```

Compare

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
model_inference	1.53%	7.094ms	99.99%	465.110ms	465.110ms	1
aten::conv2d	0.05%	216.000us	72.52%	337.325ms	16.866ms	20
aten::convolution	0.05%	234.000us	72.47%	337.109ms	16.855ms	20
aten::_convolution	0.10%	445.000us	72.42%	336.875ms	16.844ms	20
aten::mkldnn_convolution	72.23%	335.997ms	72.33%	336.430ms	16.822ms	20
aten::batch_norm	0.05%	233.000us	15.56%	72.384ms	3.619ms	20
aten::_batch_norm_impl_index	0.07%	304.000us	15.51%	72.151ms	3.608ms	20
aten::native_batch_norm	10.92%	50.772ms	15.43%	71.789ms	3.589ms	20
aten::max_pool2d	0.02%	107.000us	8.81%	40.962ms	40.962ms	1
aten::max_pool2d_with_indices	8.78%	40.855ms	8.78%	40.855ms	40.855ms	1

Self CPU time total: 465.157ms

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
model_inference	3.60%	26.986ms	99.83%	749.335ms	749.335ms	1
aten::conv2d	0.04%	265.000us	71.15%	534.099ms	26.705ms	20
aten::convolution	0.03%	258.000us	71.12%	533.834ms	26.692ms	20
aten::_convolution	0.07%	503.000us	71.08%	533.576ms	26.679ms	20
aten::mkldnn_convolution	70.90%	532.177ms	71.02%	533.073ms	26.654ms	20
aten::batch_norm	0.49%	3.642ms	13.25%	99.429ms	4.971ms	20
aten::_batch_norm_impl_index	0.06%	467.000us	12.76%	95.787ms	4.789ms	20
aten::native_batch_norm	7.48%	56.169ms	12.69%	95.247ms	4.762ms	20
aten::max_pool2d	0.01%	40.000us	7.33%	54.991ms	54.991ms	1
aten::max_pool2d_with_indices	7.32%	54.951ms	7.32%	54.951ms	54.951ms	1

Self CPU time total: 750.624ms

Setup PyTorch Profiler with GPU

```
model = models.resnet18().cuda()
inputs = torch.randn(5, 3, 224, 224).cuda()

with profiler.profile(activities=[
        profiler.ProfilerActivity.CPU, profiler.ProfilerActivity.CUDA], record_shapes=True) as prof:
    with profiler.record_function("model_inference"):
        model(inputs)

print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))
```

Measure

```
print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))
```

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg
model_inference	10.41%	3.664ms	62.54%	22.011ms	22.011ms	0.000us	0.00%	94.171ms	94.171ms
aten::conv2d	0.38%	135.000us	35.02%	12.327ms	616.350us	0.000us	0.00%	90.517ms	4.526ms
aten::convolution	0.33%	116.000us	34.64%	12.192ms	609.600us	0.000us	0.00%	90.517ms	4.526ms
aten::_convolution	0.74%	262.000us	34.31%	12.076ms	603.800us	0.000us	0.00%	90.517ms	4.526ms
aten::cudnn_convolution	12.34%	4.342ms	33.57%	11.814ms	590.700us	90.517ms	96.12%	90.517ms	4.526ms
sgemm_sm35_ldg_nn_64x16x64x16x16	0.00%	0.000us	0.00%	0.000us	0.000us	61.349ms	65.15%	61.349ms	189.349us
void implicit_convolve_sgemm<float, float, 1024, 6, ...	0.00%	0.000us	0.00%	0.000us	0.000us	14.192ms	15.07%	14.192ms	746.947us
void precomputed_convolve_sgemm<float, 128, 5, 5, 3,...	0.00%	0.000us	0.00%	0.000us	0.000us	7.936ms	8.43%	7.936ms	1.134ms
aten::batch_norm	0.53%	188.000us	9.29%	3.269ms	163.450us	0.000us	0.00%	2.009ms	100.450us
aten::_batch_norm_impl_index	0.58%	203.000us	8.75%	3.081ms	154.050us	0.000us	0.00%	2.009ms	100.450us

```
Self CPU time total: 35.197ms
Self CUDA time total: 94.171ms
```

Measure

Name	Self CUDA	Self CUDA %	CUDA total
model_inference	0.000us	0.00%	94.171ms
aten::conv2d	0.000us	0.00%	90.517ms
aten::convolution	0.000us	0.00%	90.517ms
aten::_convolution	0.000us	0.00%	90.517ms
aten::cudnn_convolution	90.517ms	96.12%	90.517ms
sgemm_sm35_ldg_nn_64x16x64x16x16	61.349ms	65.15%	61.349ms
void implicit_convolve_sgemm<float, float, 1024, 6, ...	14.192ms	15.07%	14.192ms
void precomputed_convolve_sgemm<float, 128, 5, 5, 3,...	7.936ms	8.43%	7.936ms
aten::batch_norm	0.000us	0.00%	2.009ms
aten::_batch_norm_impl_index	0.000us	0.00%	2.009ms

Self CPU time total: 35.197ms
Self CUDA time total: 94.171ms

Measure

Name	Self CUDA	Self CUDA %	CUDA total
model_inference	0.000us	0.00%	94.171ms
aten::conv2d	0.000us	0.00%	90.517ms
aten::convolution	0.000us	0.00%	90.517ms
aten::_convolution	0.000us	0.00%	90.517ms
aten::cudnn_convolution	90.517ms	96.12%	90.517ms
sgemm_sm35_ldg_nn_64x16x64x16x16	61.349ms	65.15%	61.349ms
void implicit_convolve_sgemm<float, float, 1024, 6, ...	14.192ms	15.07%	14.192ms
void precomputed_convolve_sgemm<float, 128, 5, 5, 3,...	7.936ms	8.43%	7.936ms
aten::batch_norm	0.000us	0.00%	2.009ms
aten::_batch_norm_impl_index	0.000us	0.00%	2.009ms

Self CPU time total: 35.197ms
Self CUDA time total: 94.171ms

Setup PyTorch Profiler with GPU

```
model = models.resnet18().cuda()
inputs = torch.randn(5, 3, 224, 224).cuda()

with profiler.profile(activities=[
        profiler.ProfilerActivity.CPU, profiler.ProfilerActivity.CUDA],
        record_shapes=True,
        profile_memory=True) as prof:
    with profiler.record_function("model_inference"):
        model(inputs)
```

Measure

Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg	CPU Mem	Self CPU Mem	CUDA Mem	Self CUDA Mem	# of Calls
967.000us	2.75%	967.000us	7.862us	0.000us	0.00%	0.000us	0.000us	20 b	20 b	366.50 Mb	366.50 Mb	123
99.000us	26.16%	9.204ms	460.200us	0.000us	0.00%	88.609ms	4.430ms	0 b	0 b	49.67 Mb	0 b	20
102.000us	25.87%	9.105ms	455.250us	0.000us	0.00%	88.609ms	4.430ms	0 b	0 b	49.67 Mb	0 b	20
185.000us	25.58%	9.003ms	450.150us	0.000us	0.00%	88.609ms	4.430ms	0 b	0 b	49.67 Mb	0 b	20
3.285ms	25.06%	8.818ms	440.900us	88.609ms	96.04%	88.609ms	4.430ms	0 b	0 b	49.67 Mb	-267.96 Mb	20
114.000us	7.34%	2.583ms	129.150us	0.000us	0.00%	2.015ms	100.750us	0 b	0 b	48.86 Mb	0 b	20
189.000us	7.02%	2.469ms	123.450us	0.000us	0.00%	2.015ms	100.750us	0 b	0 b	48.86 Mb	0 b	20
961.000us	6.48%	2.280ms	114.000us	2.015ms	2.18%	2.015ms	100.750us	0 b	0 b	48.86 Mb	0 b	20
107.000us	1.18%	415.000us	20.750us	0.000us	0.00%	0.000us	0.000us	0 b	0 b	48.82 Mb	0 b	20
8.000us	0.28%	98.000us	98.000us	0.000us	0.00%	407.000us	407.000us	0 b	0 b	11.54 Mb	0 b	1

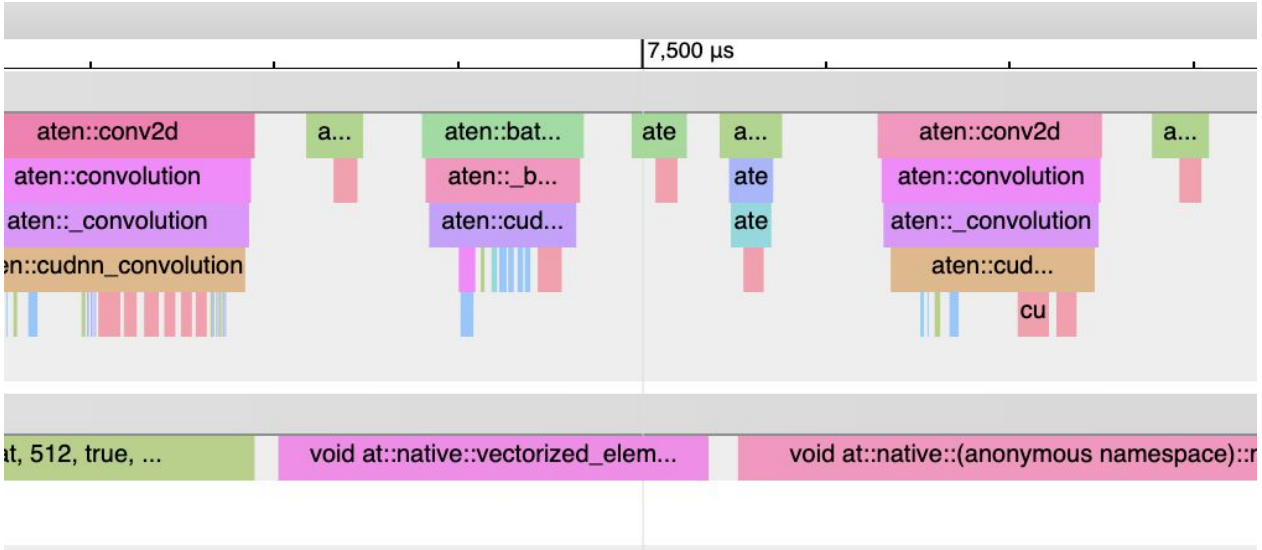
Setup Profile Trace

```
model = models.resnet18().cuda()
inputs = torch.randn(5, 3, 224, 224).cuda()

with profiler.profile(activities=[profiler.ProfilerActivity.CPU,
                                  profiler.ProfilerActivity.CUDA]) as prof:
    model(inputs)

prof.export_chrome_trace("trace.json")
```

Visualize

1. Go to chrome://tracing

2. Upload the trace



Some other useful things

1. Visualizing the data as flamegraphs

2. Using profiler to analyze long-running jobs

3. Using PyTorch profiler with Tensorboard

4. Using PyTorch profiler with VSCode

Tuning PyTorch Models

Multi-process Data Loading

1. A dataloader uses single-process data loading by default.

2. Therefore, data loading may block computing.

3. Enable multi-process data loading by passing `num_workers=<some positive number>` when creating the dataloader.

4. Further Reading

Memory Pinning

1. Host to GPU copies are much faster when they originate from pinned (page-locked) memory.

2. For data loading, passing `pin_memory=True` to dataloader constructor will put the fetched data Tensors in pinned memory.

3. This will enable faster data transfer to CUDA-enabled GPUs.

4. Further Reading

Inference Mode

1. Code run under this mode gets better performance by disabling view tracking and version counter bumps.

2. Use it only when you are certain your operations will have no interactions with autograd

3. Further Reading

```
>>> import torch
>>> x = torch.ones(1, 2, 3, requires_grad=True)
>>> with torch.inference_mode():
...   y = x * x
>>> y.requires_grad
False
```

Set grad to None

1. Can pass an additional argument called `set_to_none` when calling optimizer.zero_grad()

2. This sets the grad to None (and not 0)

3. Leads to lower memory footprint and modestly faster performance.

4. Caveats apply!

5. Further Reading

Enable cuDNN auto-tuner

1. Set `torch.backends.cudnn.benchmark=True`

2. Causes cuDNN to benchmark multiple convolution algorithms and select the fastest.

3. Can make the results non-deterministic.

4. Further Reading

Use DistributedDataParallel instead of DataParallel

1. DistributedDataParallel uses multiprocessing where a process is created for each GPU while DataParallel uses multithreading.

2. By using multiprocessing, each GPU has its dedicated process.

3. This avoids the performance overhead caused by GIL of Python interpreter

4. Further Reading

Fuse point-wise operators

1. Use torch.jit to fuse point-wise operators into a single operator (kernel call)

2. Pointwise operations are memory-bound, for each operation PyTorch launches a separate kernel.

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

4. Further Reading

```
def gelu(x):
  return x * 0.5 * (1.0 + torch.erf(x / 1.4))

@torch.jit.script
def fused_gelu(x):
  return gelu(x)
```

Use Fused Optimizers

1. Similar benefits as fusing PyTorch layers/operations.

2. Supported optimizers: FusedAdam, FusedLAMB, FusedNovoGrad, FusedSGD

3. Further Reading

Checkpoint intermediate buffers

1. For the backward pass, store the inputs of a few layers and recomput others during the backward pass.

2. This reduces the memory requirements, and enables increasing the batch size.

3. Further Reading

Avoid unnecessary CPU-GPU synchronization

1. Avoid operations that requires synchronization.

2. This includes:

A. print(cuda_tensor)
B. cuda_tensor.item()
C. cuda_tensor.cpu()
D. python control flow which depends on results of operations performed on cuda tensors e.g. if (cuda_tensor != 0).all()

Use mixed precision and AMP

1. Mixed precision: some operations use the torch.float32 (float) datatype and other operations use torch.float16 (half).

2. Some ops, like linear layers and convolutions, are much faster in float16.

3. Other ops, like reductions, often require the dynamic range of float32.

2. Further Reading

Thank you @shagunsodhani

FACEBOOK AI

References

1. https://nvlabs.github.io/eccv2020-mixed-precision-tutorial/
2. PyTorch Performance Tuning Guide
3. https://pytorch.org/tutorials/recipes/recipes/tuning_guide.html
4. https://pytorch.org/blog/introducing-pytorch-profiler-the-new-and-improved-performance-tool/