AUTOMATIC MIXED PRECISION TRAINING

Michael Carilli

Using mixed precision and Volta/Turing your networks can be:

- 2 – 4X faster
- more memory-efficient
- just as accurate

with no architecture or hyperparameter changes.

MAXIMIZING MODEL PERFORMANCE

FP16 input enables Volta/Turing Tensor Cores for Matrix Multiplies and Convolutions

$$\underset{\text{FP16 or FP32}}{D} = \underset{\text{FP16}}{\left(\begin{matrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{matrix}\right)} \underset{\text{FP16}}{\left(\begin{matrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{matrix}\right)} + \underset{\text{FP16 or FP32}}{\left(\begin{matrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{matrix}\right)}$$

125 TFlops Throughput: **8X** more than FP32 on Volta V100

MAXIMIZING MODEL PERFORMANCE

Assign each operation its optimal precision

FP16 with Tensor Cores

8X compute throughput

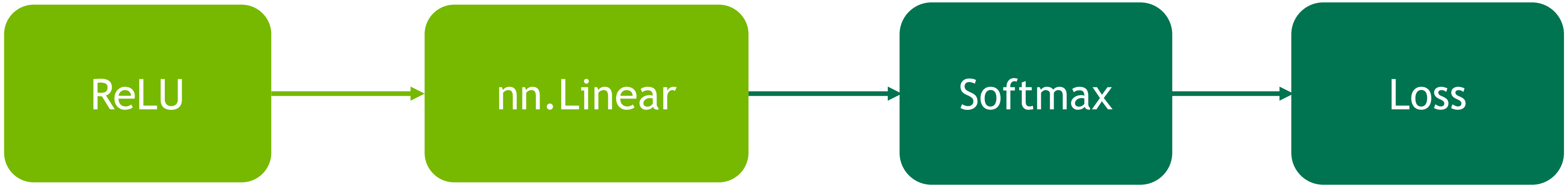
2X memory throughput

1/2X memory storage

FP32

Wider dynamic range

Increased precision captures small accumulations



MIXED PRECISION IN PRACTICE: SPEED

Mixed Precision vs. FP32

BERT	3.25X - 4.25X speedup*
Jasper	2.2X - 3X speedup**
Mask-RCNN	1.2X - 1.5X speedup†
FAIRseq	4X speedup
GNMT	2X speedup

* https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/LanguageModeling/BERT
** https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/SpeechRecognition/Jasper
† https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Segmentation/MaskRCNN

MIXED PRECISION IN PRACTICE: ACCURACY

Same accuracy as FP32, with no hyperparameter changes

Model	FP32	Mixed Precision*
AlexNet**	56.77%	56.93%
VGG-D	65.40%	65.43%
GoogLeNet (Inception v1)	68.33%	68.43%
Inception v2	70.03%	70.02%
Inception v3	73.85%	74.13%
Resnet50	75.92%	76.04%
BERT Fine-Tuning†	91.18%	91.24%

* Same hyperparameters and learning rate schedule as FP32.
** Sharan Narang, Paulius Micikevicius *et al.*, "Mixed Precision Training“, ICLR 2018
† https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/LanguageModeling/BERT

AUTOMATIC MIXED PRECISION (AMP)

Existing FP32 (default) script

->

Change 3 lines of Python

->

Accelerate your training with mixed precision

EXAMPLE

```
N, D_in, D_out = 64, 1024, 512
x = torch.randn(N, D_in, device="cuda")
y = torch.randn(N, D_out, device="cuda")

model = torch.nn.Linear(D_in, D_out).cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")

for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    optimizer.zero_grad()
    with amp.scale_loss(loss, optimizer) as scaled_loss:
        scaled_loss.backward()
    optimizer.step()
```

TRY AMP TODAY

Available through NVIDIA Apex utilities:
https://github.com/nvidia/apex

Full API documentation:
https://nvidia.github.io/apex/

Landing pages contain a link to this talk.

AMP Examples:
https://github.com/NVIDIA/DeepLearningExamples

AMP IN PYTORCH CORE

COMING SOON (TARGET Q4 2019)

- Gradient scaling and autocasting as modular components
- Intended support for networks that use:
  - JIT
  - multiple models/optimizers/losses
  - gradient accumulation
  - gradient checkpointing
  - gradient penalty (double-backward)
  - custom optimizers
- Let us know what you need!
  - API discussion: https://github.com/pytorch/pytorch/issues/25081
  - Gradient scaling PR: https://github.com/pytorch/pytorch/pull/26512

NVIDIA®