PyTorch Memory Supersave

Contents

Introduction


```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-23-8b886acaef8a> in <module>
     10         optimizer.zero_grad()
     11 
---> 12         outputs = net(inputs)
     13         loss = criterion(outputs, labels)
     14         losses.append(loss.item())

                         8 frames

/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py in relu(input, inplace)
   1455         result = torch.relu_(input)
   1456     else:
-> 1457         result = torch.relu(input)
   1458     return result
   1459 

RuntimeError: CUDA out of memory. Tried to allocate 20.00 MiB (GPU 0; 14.76 GiB
total capacity; 13.34 GiB already allocated; 3.75 MiB free; 13.62 GiB reserved in
total by PyTorch) If reserved memory is >> allocated memory try setting
max_split_size_mb to avoid fragmentation.  See documentation for Memory Management
and PYTORCH_CUDA_ALLOC_CONF
```

Mo Parameters Mo Problem

Pytorch Memory Comprehension


```
def test_memory(in_size=100, out_size=10, hidden_size=100, optimizer_type=torch.optim.Adam, batch_size=1, device=0):
    sample_input = torch.randn(batch_size, in_size)
    model = nn.Sequential(nn.Linear(in_size, hidden_size),
                          *[nn.Linear(hidden_size, hidden_size) for _ in range(200)],
                          nn.Linear(hidden_size, out_size))
    optimizer = optimizer_type(model.parameters(), lr=.001)
    print("Beginning mem:", torch.cuda.memory_allocated(device))
    model.to(device)
    print("1 - After model to device:", torch.cuda.memory_allocated(device))
    for i in range(3):
        print("Iteration", i)
        a = torch.cuda.memory_allocated(device)
        out = model(sample_input.to(device)).sum() # Taking the sum here just to get a scalar output
        b = torch.cuda.memory_allocated(device)
        print("2 - After forward pass", torch.cuda.memory_allocated(device))
        print("2 - Memory consumed by forward pass", b - a)
        out.backward()
        print("3 - After backward pass", torch.cuda.memory_allocated(device))
        optimizer.step()
        print("4 - After optimizer step", torch.cuda.memory_allocated(device))
```

https://medium.com/deep-learning-for-protein-design/a-comprehensive-guide-to-memory-usage-in-pytorch-b9b7c78031d3

Pytorch Memory Comprehension

```
Maximum Memory Estimate 38.122 MB
Beginning mem: 8.238 MB (PyTorch caching)
After model to device: 8.238MB
Iteration 0
1 - After forward pass 13.409MB
2 - Memory consumed by forward pass 5.172MB
3 - After backward pass 16.476MB
4 - After optimizer step 32.951MB
Iteration 1
1 - After forward pass 38.122MB
2 - Memory consumed by forward pass 5.171MB
3 - After backward pass 32.951MB
4 - After optimizer step 32.951MB
```

Model Load	Model
Forward	Model + Activations
Backward	Model + Gradients
Optimizer	Above + Grad Moment
Next (Max)	Above + Activations

https://medium.com/deep-learning-for-protein-design/a-comprehensive-guide-to-memory-usage-in-pytorch-b9b7c78031d3
https://stackoverflow.com/questions/69125887/understanding-why-memory-allocation-occurs-during-inference-backpropagation-an

Pytorch Memory Comprehension


```
Beginning mem: 0.000 MB (PyTorch caching)
After model to device: 646.615MB
Iteration 0
1 – After forward pass 4891.966MB
2 – Memory consumed by forward pass 4239.056MB
3 – After backward pass 1303.353MB
4 – After optimizer step 1961.115MB
Iteration 1
1 – After forward pass 6202.006MB
2 – Memory consumed by forward pass 4234.595MB
3 – After backward pass 1961.115MB
4 – After optimizer step 1961.115MB
Iteration 2
1 – After forward pass 6200.957MB
2 – Memory consumed by forward pass 4233.547MB
3 – After backward pass 1961.115MB
4 – After optimizer step 1961.115MB
Took 7.198S
Training Done
```

Crop 32x32: 6202(MB)

```
Beginning mem: 0.000 MB (PyTorch caching)
After model to device: 646.615MB
Iteration 0
1 – After forward pass 9744.776MB
2 – Memory consumed by forward pass 9088.327MB
3 – After backward pass 1365.088MB
4 – After optimizer step 2014.199MB
Iteration 1
1 – After forward pass 11062.418MB
2 – Memory consumed by forward pass 9038.384MB
3 – After backward pass 2014.199MB
4 – After optimizer step 2014.199MB
Iteration 2
1 – After forward pass 11058.355MB
2 – Memory consumed by forward pass 9034.321MB
3 – After backward pass 2014.199MB
4 – After optimizer step 2014.199MB
Took 11.386S
Training Done
```

Crop 40x40: 9038(MB)

Effects of Image Size : ResNet-428 with Batch Size 512

ResNet Introduction

https://arxiv.org › cs

[1512.03385] Deep Residual Learning for Image Recognition

K He 저술 · 2015 · 132395회 인용 — Abstract: Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that ...

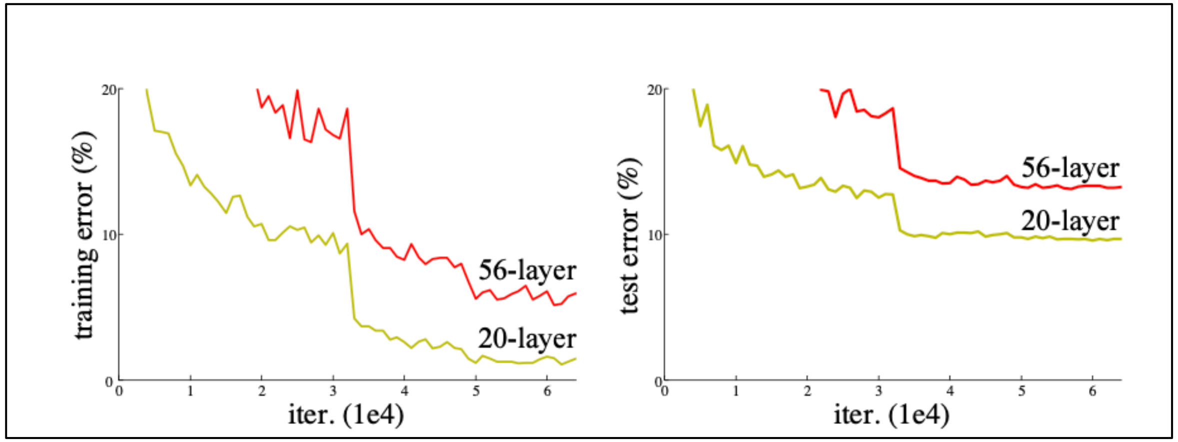
Cite as: arXiv:1512.03385

Most-cited papers [edit]

The most-cited paper in history is a paper by Oliver Lowry describing an assay to measure the concentration of proteins.[11] By 2014 it had accumulated more than 305,000 citations. The 10 most cited papers all had more than 40,000 citations.[12] To reach the top-100 papers required 12,119 citations by 2014.[12] Of Thomson Reuter's Web of Science database with more than 58 million items only 14,499 papers (~0.026%) had more than 1,000 citations in 2014.[12]

https://en.wikipedia.org/wiki/Citation_impact

ResNet Introduction



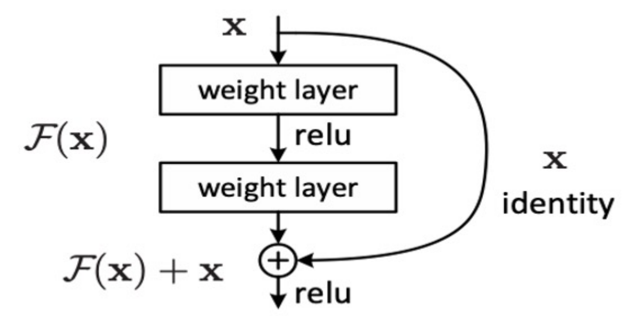
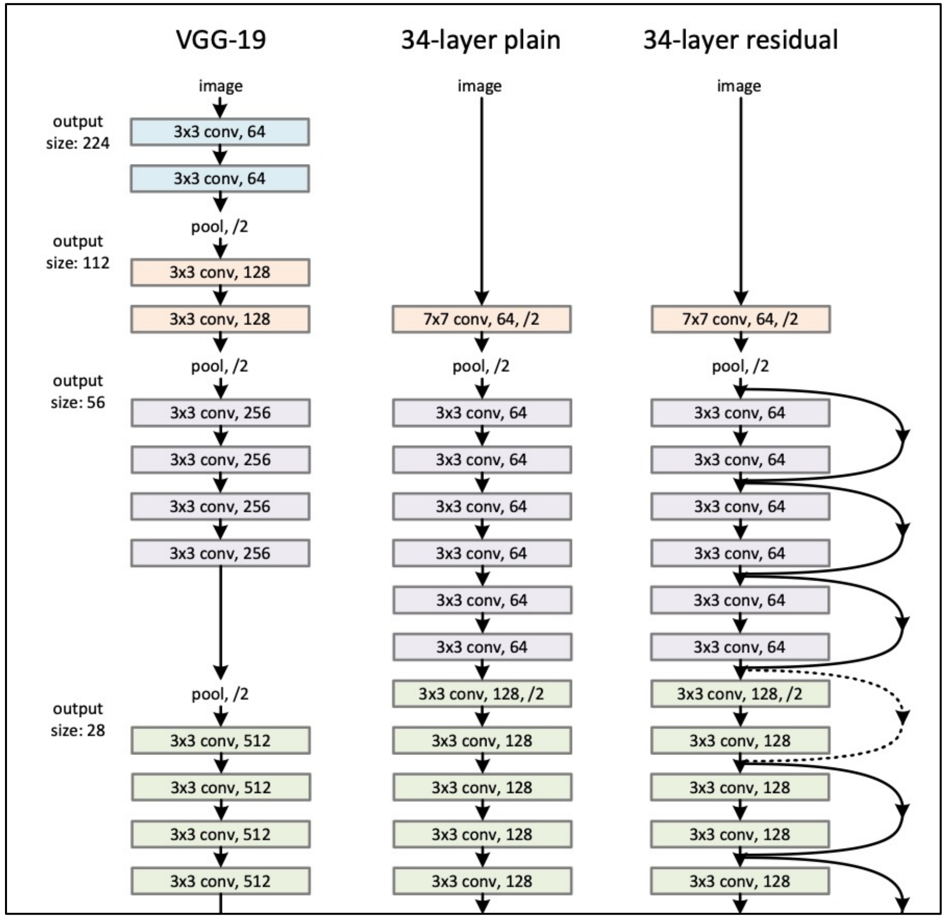


Figure 2. Residual learning: a building block.



He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

ResNet Introduction


```
class Bottleneck(nn.Module):
    expansion = 4
    def __init__(self, in_channels, out_channels, i_downsample=None, stride=1):
        super(Bottleneck, self).__init__()

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, padding=0)
        self.batch_norm1 = nn.BatchNorm2d(out_channels)

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=stride, padding=1)
        self.batch_norm2 = nn.BatchNorm2d(out_channels)

        self.conv3 = nn.Conv2d(out_channels, out_channels*self.expansion, kernel_size=1, stride=1, padding=0)
        self.batch_norm3 = nn.BatchNorm2d(out_channels*self.expansion)

        self.i_downsample = i_downsample
        self.stride = stride
        self.relu = nn.ReLU()

    def forward(self, x):
        identity = x.clone()
        x = self.relu(self.batch_norm1(self.conv1(x)))
        x = self.relu(self.batch_norm2(self.conv2(x)))
        x = self.conv3(x)
        x = self.batch_norm3(x)

        if self.i_downsample is not None:
            identity = self.i_downsample(identity)

        x+=identity
        x=self.relu(x)

        return x
```

```
class ResNet(nn.Module):
    def __init__(self, ResBlock, layer_list, num_classes, num_channels=3):
        super(ResNet, self).__init__()
        self.in_channels = 64

        self.conv1 = nn.Conv2d(num_channels, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.batch_norm1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.max_pool = nn.MaxPool2d(kernel_size = 3, stride=2, padding=1)

        self.layer1 = self._make_layer(ResBlock, layer_list[0], planes=64)
        self.layer2 = self._make_layer(ResBlock, layer_list[1], planes=128, stride=2)
        self.layer3 = self._make_layer(ResBlock, layer_list[2], planes=256, stride=2)
        self.layer4 = self._make_layer(ResBlock, layer_list[3], planes=512, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1,1))
        self.fc = nn.Linear(512*ResBlock.expansion, num_classes)

    def forward(self, x):
        x = self.relu(self.batch_norm1(self.conv1(x)))
        x = self.max_pool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = x.reshape(x.shape[0], -1)
        x = self.fc(x)

        return x
```

https://github.com/JayPatwardhan/ResNet-PyTorch/blob/master/ResNet/ResNet.py

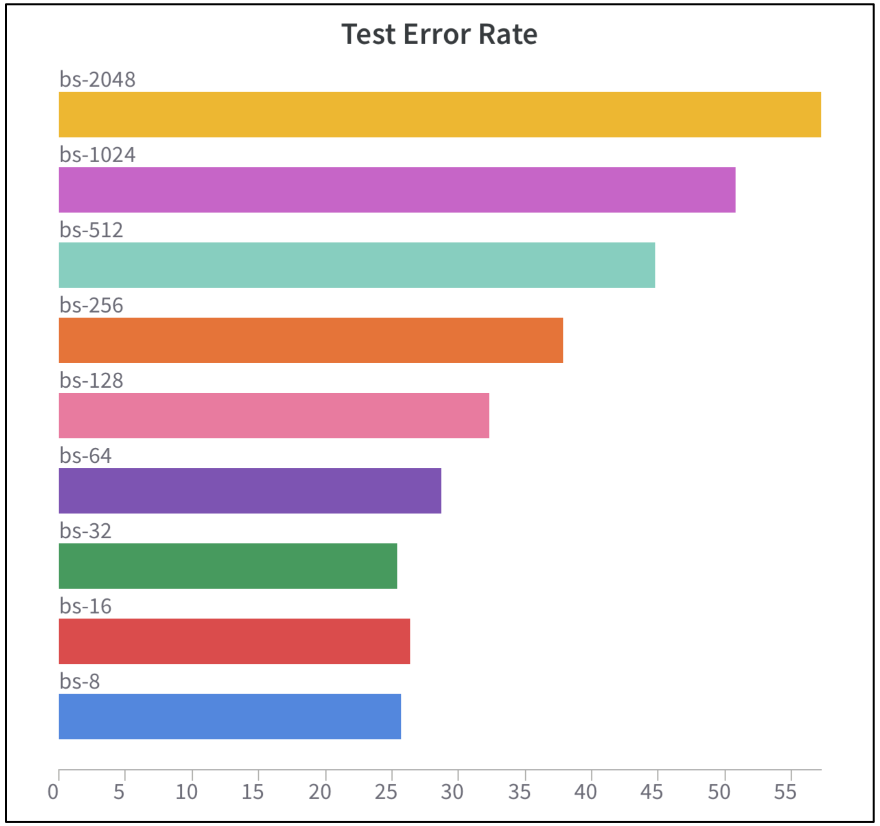
Parameter Freezing

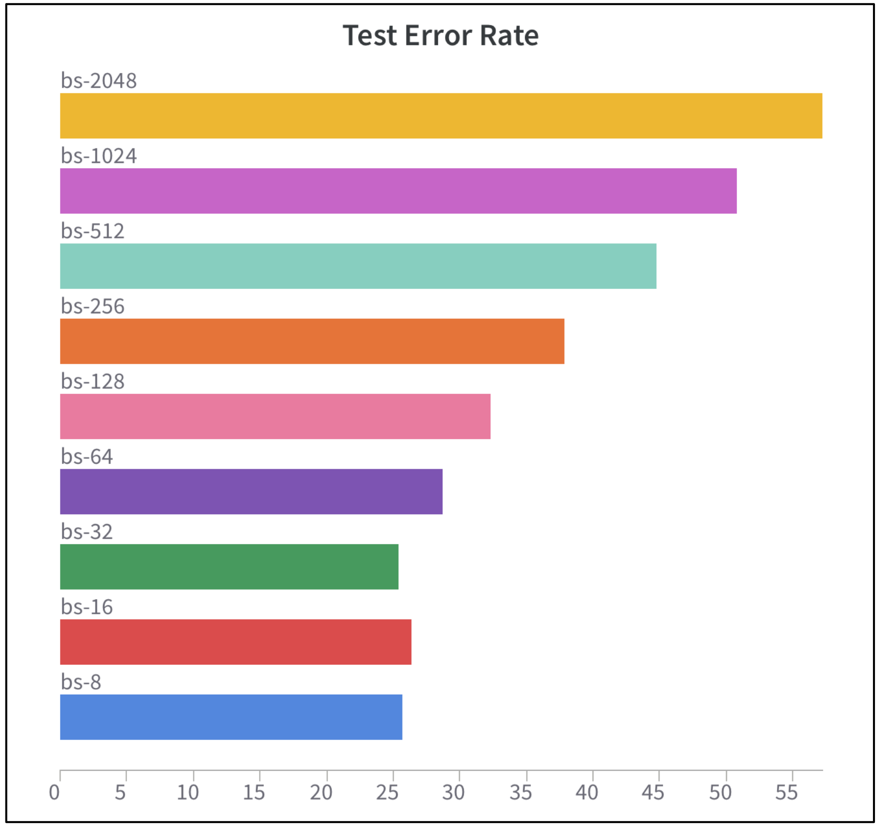

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opposed to before.
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
```

Fine Tuning | Feature Extractor if Transfer Learning

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Batch Size





What Size for Batch!?

https://wandb.ai/ayush-thakur/dl-question-bank/reports/What-s-the-Optimal-Batch-Size-to-Train-a-Neural-Network---VmlldzoyMDkyNDU

Batch Size

Batch Size	Memory (MB)	Time per Epoch (s)
256	7395	236
128	5541	310
64	4618	447
32	4166	837

ResNet-1580 with Third DownSampled Expanded on T4

Optimizer

Optimizer	Maximum Memory Usage (MB)
SGD	6177
RMS-prop	7399
ADAM	8620

ResNet with Various Optimizer on T4

Optimizer

SGD

```
Very First mem: 0.000 MB Beginning
After model to device: 1221.169MB
Maximum Memory Estimate 6177.533 MB
Beginning mem: 1221.169 MB (PyTorch caching)
Iteration 0
1 - After forward pass 4957.413MB
2 - Memory consumed by forward pass 3736.244MB
3 - After backward pass 2439.395MB
4 - After optimizer step 2439.395MB
Iteration 1
1 - After forward pass 6175.639MB
2 - Memory consumed by forward pass 3736.243MB
3 - After backward pass 2439.395MB
4 - After optimizer step 2439.395MB
Iteration 2
1 - After forward pass 6175.639MB
2 - Memory consumed by forward pass 3736.243MB
3 - After backward pass 2439.395MB
4 - After optimizer step 2439.395MB
```

Max 6177(MB)

RMSprop

```
Very First mem: 0.000 MB Beginning
After model to device: 1221.169MB
Maximum Memory Estimate 7399.751 MB
Beginning mem: 1221.169 MB (PyTorch caching)
Iteration 0
1 - After forward pass 4957.413MB
2 - Memory consumed by forward pass 3736.244MB
3 - After backward pass 2439.395MB
4 - After optimizer step 3658.407MB
Iteration 1
1 - After forward pass 7395.699MB
2 - Memory consumed by forward pass 3737.292MB
3 - After backward pass 3658.407MB
4 - After optimizer step 3658.407MB
Iteration 2
1 - After forward pass 7395.699MB
2 - Memory consumed by forward pass 3737.292MB
3 - After backward pass 3658.407MB
4 - After optimizer step 3658.407MB
```

Max 7399(MB)

ADAM

```
Very First mem: 0.000 MB Beginning
After model to device: 1221.169MB
Maximum Memory Estimate 8620.920 MB
Beginning mem: 1221.169 MB (PyTorch caching)
Iteration 0
1 - After forward pass 4957.413MB
2 - Memory consumed by forward pass 3736.244MB
3 - After backward pass 2439.395MB
4 - After optimizer step 4879.779MB
Iteration 1
1 - After forward pass 8617.071MB
2 - Memory consumed by forward pass 3737.292MB
3 - After backward pass 4879.779MB
4 - After optimizer step 4879.779MB
Iteration 2
1 - After forward pass 8617.071MB
2 - Memory consumed by forward pass 3737.292MB
3 - After backward pass 4879.779MB
4 - After optimizer step 4879.779MB
```

Max 8620(MB)

https://colab.research.google.com/drive/1bzCH3Yaq8gK0ZByxlcaRaj3pOc2u6zht

Half Accuracy Learning

Neural networks are commonly trained with 32-bit floating point (FP32) precision. That is, all numbers are stored in FP32 format and both inputs and outputs of arithmetic operations are FP32 numbers as well. New hardware, however, may have enhanced arithmetic logic unit for lower precision data types. For example, the previously mentioned **Nvidia V100 offers 14 TFLOPS in FP32 but over 100 TFLOPS in FP16.** As in Table 3, the **overall training speed is accelerated by 2 to 3 times after switching from FP32 to FP16 on V100.**

Using 16-bit Floats

The new RTX and Volta cards by nVidia support both 16-bit training and inference.

```
model = model.half()     # convert a model to 16-bit
input = input.half()     # convert a model to 16-bit
```

He, Tong, et al. "Bag of tricks for image classification with convolutional neural networks." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.
https://blog.paperspace.com/pytorch-memory-multi-gpu-debugging/
https://discuss.pytorch.org/t/training-with-half-precision/11815/3

Half Accuracy Learning


```
net = ResNet(Bottleneck, [3,8,512,3], num_classes=10).half().to('cuda')
print("After model to device:", f"{torch.cuda.memory_allocated('cuda') * 1e-6:.3f}MB")

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=0.0001)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor = 0.1, patience=5)

for epoch in range(1):
    losses = []
    running_loss = 0
    start = time.perf_counter()
    for i, inp in enumerate(trainloader):

        inputs, labels = inp
        inputs, labels = inputs.half().to('cuda'), labels.to('cuda')
```

Super Easy !

Half Accuracy Learning

Original	Half

Original

```
Beginning mem: 0.000 MB (PyTorch caching)
After model to device: 2370.277MB
1 - After forward pass 5856.441MB
2 - Memory consumed by forward pass 3484.066MB
3 - After backward pass 4760.031MB
4 - After optimizer step 7124.350MB
1 - After forward pass 10607.106MB
2 - Memory consumed by forward pass 3481.181MB
3 - After backward pass 7123.826MB
4 - After optimizer step 7123.826MB
1 - After forward pass 10607.106MB
2 - Memory consumed by forward pass 3481.705MB
3 - After backward pass 7123.826MB
4 - After optimizer step 7123.826MB
Took 7.550S
Training Done
```

Max 10607(MB)

Half

```
Beginning mem: 0.000 MB (PyTorch caching)
After model to device: 1213.885MB
1 - After forward pass 2960.459MB
2 - Memory consumed by forward pass 1745.786MB
3 - After backward pass 2426.082MB
4 - After optimizer step 3638.668MB
1 - After forward pass 5384.978MB
2 - Memory consumed by forward pass 1745.523MB
3 - After backward pass 3638.668MB
4 - After optimizer step 3638.668MB
1 - After forward pass 5384.978MB
2 - Memory consumed by forward pass 1745.523MB
3 - After backward pass 3638.668MB
4 - After optimizer step 3638.668MB
Took 7.259S
Training Done
```

Max 5384(MB)

ResNet-1580 with Batch Size 128

He, Tong, et al. "Bag of tricks for image classification with convolutional neural networks." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.

Gradient Checkpointing

Gradient checkpointing

The idea behind gradient checkpointing is pretty simple:
If I need some data that I have computed once, I don't need to store it: I can compute it again

So basically instead of storing all the layers' inputs, I will store a few **checkpoints** along the way during the forward pass, and when I need some input that I haven't stored I'll just recompute it from the last checkpoint.

Plus it's really easy to implement in Pytorch, especially if you have a nn.Sequential module. To apply it , I changed the line 9 of the log function as below:

Chen, Tianqi, et al. "Training deep nets with sublinear memory cost." *arXiv preprint arXiv:1604.06174* (2016).
https://www.sicara.fr/blog-technique/2019-28-10-deep-learning-memory-usage-and-pytorch-optimization-tricks
https://qywu.github.io/2019/05/22/explore-gradient-checkpointing.html
https://blog.paperspace.com/pytorch-memory-multi-gpu-debugging/

Gradient Checkpointing

Original	Gradient Checkpointing

```python
def forward(self, x):
    x = self.relu(self.batch_norm1(self.conv1(x)))
    x = self.max_pool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)
    x = x.reshape(x.shape[0], -1)
    x = self.fc(x)

    return x
```

```python
def forward(self, x):
    x = self.relu(self.batch_norm1(self.conv1(x)))
    x = self.max_pool(x)

    x = self.layer1(x)
    x = checkpoint(self.layer2, x)
    x = checkpoint(self.layer3, x)
    x = checkpoint(self.layer4, x)

    x = self.avgpool(x)
    x = x.reshape(x.shape[0], -1)
    x = self.fc(x)

    return x
```

from torch.utils.checkpoint import checkpoint

Gradient Checkpointing

Original

```
Beginning mem: 0.000 MB (PyTorch caching)
After model to device: 2370.277MB
Iteration 0
1 - After forward pass 5856.441MB
2 - Memory consumed by forward pass 3484.066MB
3 - After backward pass 4760.031MB
4 - After optimizer step 7124.350MB
Iteration 1
1 - After forward pass 10607.106MB
2 - Memory consumed by forward pass 3481.181MB
3 - After backward pass 7123.826MB
4 - After optimizer step 7123.826MB
Iteration 2
1 - After forward pass 10607.106MB
2 - Memory consumed by forward pass 3481.705MB
3 - After backward pass 7123.826MB
4 - After optimizer step 7123.826MB
Took 7.787S
Training Done
```

Max 10607(MB)

Gradient Checkpoint

```
Beginning mem: 0.000 MB (PyTorch caching)
After model to device: 2370.277MB
Iteration 0
1 - After forward pass 2488.788MB
2 - Memory consumed by forward pass 116.412MB
3 - After backward pass 4754.002MB
4 - After optimizer step 7120.156MB
Iteration 1
1 - After forward pass 7237.093MB
2 - Memory consumed by forward pass 115.363MB
3 - After backward pass 7119.632MB
4 - After optimizer step 7119.632MB
Iteration 2
1 - After forward pass 7237.093MB
2 - Memory consumed by forward pass 117.461MB
3 - After backward pass 7119.632MB
4 - After optimizer step 7119.632MB
Took 9.232S
Done
```

Max 7237(MB)

Half + GC

```
Beginning mem: 0.000 MB (PyTorch caching)
After model to device: 1213.885MB
Iteration 0
1 - After forward pass 1274.983MB
2 - Memory consumed by forward pass 60.311MB
3 - After backward pass 2424.116MB
4 - After optimizer step 3635.784MB
Iteration 1
1 - After forward pass 3696.095MB
2 - Memory consumed by forward pass 60.311MB
3 - After backward pass 3635.784MB
4 - After optimizer step 3635.784MB
Iteration 2
1 - After forward pass 3696.095MB
2 - Memory consumed by forward pass 60.311MB
3 - After backward pass 3635.784MB
4 - After optimizer step 3635.784MB
Took 8.609S
Done
```

Max 3696(MB)

ResNet-1580 with Batch Size 128

Reference

1. [ResNet Code] https://github.com/JayPatwardhan/ResNet-PyTorch/blob/master/ResNet/ResNet.py
2. [Torch Memory Comprehension] https://medium.com/deep-learning-for-protein-design/a-comprehensive-guide-to-memory-usage-in-pytorch-b9b7c78031d3
3. [Citation Impact] https://en.wikipedia.org/wiki/Citation_impact
4. [ResNet] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
5. [ResNet Code] https://github.com/JayPatwardhan/ResNet-PyTorch/blob/master/ResNet/ResNet.py
6. [Transfer Learn] https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
7. [Batch Size] https://wandb.ai/ayush-thakur/dl-question-bank/reports/What-s-the-Optimal-Batch-Size-to-Train-a-Neural-Network---VmlldzoyMDkyNDU
8. [Memory Tracker Code] https://colab.research.google.com/drive/1bzCH3Yaq8gK0ZByxlcaRaj3pOc2u6zht
9. [Half Accuracy Learning] He, Tong, et al. "Bag of tricks for image classification with convolutional neural networks." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.
10. [Various Ways of Saving Memories] https://blog.paperspace.com/pytorch-memory-multi-gpu-debugging/
11. [Half Accuracy Learning] https://discuss.pytorch.org/t/training-with-half-precision/11815/3
12. [Gradient Checkpointing Original Paper] Chen, Tianqi, et al. "Training deep nets with sublinear memory cost." *arXiv preprint arXiv:1604.06174* (2016).
13. [Gradient Checkpointing] https://www.sicara.fr/blog-technique/2019-28-10-deep-learning-memory-usage-and-pytorch-optimization-tricks
14. [Gradient Checkpointing] https://qywu.github.io/2019/05/22/explore-gradient-checkpointing.html

Thank You