PyTorch | 优化神经网络训练的17种方法

CV开发者都爱看的 极市平台 2023-05-22 22:00:10 发表于广东 手机阅读

作者 | LORENZ KUHN

来源 | 人工智能前沿讲习

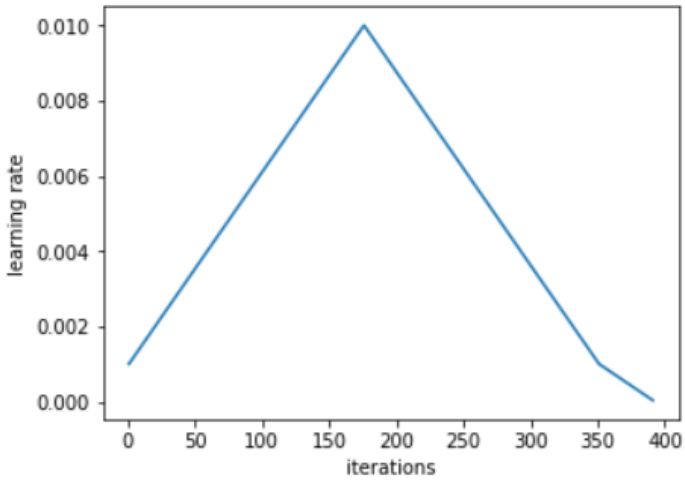
编辑 | 极市平台

极市导读

本文介绍在使用 PyTorch 训练深度模型时最省力、最有效的 17 种方法。该文所提方法，都是假设你在 GPU 环境下训练模型。具体内容如下。>>加入极市CV技术交流群，走在计算机视觉的最前沿

01 考虑换一种学习率 schedule

学习率 schedule 的选择对模型的收敛速度和泛化能力有很大的影响。Leslie N. Smith 等人在论文《Cyclical Learning Rates for Training Neural Networks》、《Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates 》中提出了周期性（Cyclical）学习率以及 1Cycle 学习率 schedule。之后，fast.ai 的 Jeremy Howard 和 Sylvain Gugger 对其进行了推广。下图是 1Cycle 学习率 schedule 的图示：



Sylvain 写到：1Cycle 包括两个等长的步幅，一个步幅是从较低的学习率到较高的学习率，另一个是回到最低水平。最大值来自学习率查找器选取的值，较小的值可以低十倍。然后，这个周期的长度应该略小于总的 epochs 数，并且，在训练的最后阶段，我们应该允许学习率比最小值小几个数量级。与传统的学习率 schedule 相比，在最好的情况下，该 schedule 实现了巨大的加速（Smith 称之为超级收敛）。例如，使用 1Cycle 策略在 ImageNet 数据集上训练 ResNet-56，训练迭代次数减少为原来的 1/10，但模型性能仍能比肩原论文中的水平。在常见的体系架构和优化器中，这种 schedule 似乎表现得很好。

Pytorch 已经实现了这两种方法：「torch.optim.lr_scheduler.CyclicLR」和「torch.optim.lr_scheduler.OneCycleLR」。

参考文档：https://pytorch.org/docs/stable/optim.html

02 在 DataLoader 中使用多个 worker 和页锁定内存

当使用 torch.utils.data.DataLoader 时，设置 num_workers > 0，而不是默认值 0，同时设置 pin_memory=True，而不是默认值 False。

参考文档：https://pytorch.org/docs/stable/data.html

来自 NVIDIA 的高级 CUDA 深度学习算法软件工程师 Szymon Micacz 就曾使用四个 worker 和页锁定内存（pinned memory）在单个 epoch 中实现了 2 倍的加速。人们选择 worker 数量的经验法则是将其设置为可用 GPU 数量的四倍，大于或小于这个数都会降低训练速度。请注意，增加 num_workers 将增加 CPU 内存消耗。

03 把 batch 调到最大

把 batch 调到最大是一个颇有争议的观点。一般来说，如果在 GPU 内存允许的范围内将 batch 调到最大，你的训练速度会更快。但是，你也必须调整其他超参数，比如学习率。一个比较好用的经验是，batch 大小加倍时，学习率也要加倍。

OpenAI 的论文《An Empirical Model of Large-Batch Training》很好地论证了不同的 batch 大小需要多少步才能收敛。在《How to get 4x speedup and better generalization using the right batch size》一文中，作者 Daniel Huynh 使用不同的 batch 大小进行了一些实验（也使用上面讨论的 1Cycle 策略）。

最终，他将 batch 大小由 64 增加到 512，实现了 4 倍的加速。然而，使用大 batch 的不足是，这可能导致解决方案的泛化能力比使用小 batch 的差。

04 使用自动混合精度（AMP）

PyTorch 1.6 版本包括对 PyTorch 的自动混合精度训练的本地实现。这里想说的是，与单精度 (FP32) 相比，某些运算在半精度 (FP16) 下运行更快，而不会损失准确率。AMP 会自动决定应该以哪种精度执行哪种运算。这样既可以加快训练速度，又可以减少内存占用。

在最好的情况下，AMP 的使用情况如下：

```
import torch
# Creates once at the beginning of training
scaler = torch.cuda.amp.GradScaler()


for data, label in data_iter:
   optimizer.zero_grad()
   # Casts operations to mixed precision
   with torch.cuda.amp.autocast():
      loss = model(data)

   # Scales the loss, and calls backward()
   # to create scaled gradients
   scaler.scale(loss).backward()

   # Unscales gradients and calls
   # or skips optimizer.step()
   scaler.step(optimizer)

   # Updates the scale for next iteration
   scaler.update()
```

05 考虑使用另一种优化器

AdamW 是由 fast.ai 推广的一种具有权重衰减（而不是 L2 正则化）的 Adam，在 PyTorch 中以 torch.optim.AdamW 实现。AdamW

似乎在误差和训练时间上都一直优于 Adam。Adam 和 AdamW 都能与上面提到的 1Cycle 策略很好地搭配。

目前，还有一些非本地优化器也引起了很大的关注，最突出的是 LARS 和 LAMB。NVIDA 的 APEX 实现了一些常见优化器的融合版本，比如 Adam。与 PyTorch 中的 Adam 实现相比，这种实现避免了与 GPU 内存之间的多次传递，速度提高了 5%。

06 cudNN 基准

如果你的模型架构保持不变、输入大小保持不变，设置 torch.backends.cudnn.benchmark = True。

07 小心 CPU 和 GPU 之间频繁的数据传输

当频繁地使用 tensor.cpu() 将张量从 GPU 转到 CPU（或使用 tensor.cuda() 将张量从 CPU 转到 GPU）时，代价是非常昂贵的。item() 和 .numpy() 也是一样可以使用. detach() 代替。

如果你创建了一个新的张量，可以使用关键字参数 device=torch.device(cuda:0) 将其分配给 GPU。

如果你需要传输数据，可以使用. to(non_blocking=True)，只要在传输之后没有同步点。

08 使用梯度 / 激活 checkpointing

Checkpointing 的工作原理是用计算换内存，并不存储整个计算图的所有中间激活用于 backward pass，而是重新计算这些激活。我们可以将其应用于模型的任何部分。

具体来说，在 forward pass 中，function 会以 torch.no_grad() 方式运行，不存储中间激活。相反的是， forward pass 中会保存输入元组以及 function 参数。在 backward pass 中，输入和 function 会被检索，并再次在 function 上计算 forward pass。然后跟踪中间激活，使用这些激活值计算梯度。

因此，虽然这可能会略微增加给定 batch 大小的运行时间，但会显著减少内存占用。这反过来又将允许进一步增加所使用的 batch 大小，从而提高 GPU 的利用率。

尽管 checkpointing 以 torch.utils.checkpoint 方式实现，但仍需要一些思考和努力来正确地实现。Priya Goyal 写了一个很好的教程来介绍 checkpointing 关键方面。

Priya Goyal 教程地址：https://github.com/prigoyal/pytorch_memonger/blob/master/tutorial/Checkpointing_for_PyTorch_models.ipynb

09 使用梯度积累

增加 batch 大小的另一种方法是在调用 optimizer.step() 之前在多个. backward() 传递中累积梯度。

Hugging Face 的 Thomas Wolf 的文章《Training Neural Nets on Larger Batches: Practical Tips for 1-GPU, Multi-GPU & Distributed setups》介绍了如何使用梯度累积。梯度累积可以通过如下方式实现：

```
model.zero_grad()                                   # Reset gradients tensors
for i, (inputs, labels) in enumerate(training_set):
    predictions = model(inputs)                     # Forward pass
    loss = loss_function(predictions, labels)       # Compute loss function
    loss = loss / accumulation_steps                # Normalize our loss (if averaged)
    loss.backward()                                 # Backward pass
    if (i+1) % accumulation_steps == 0:             # Wait for several backward steps
        optimizer.step()                            # Now we can do an optimizer step
        model.zero_grad()                           # Reset gradients tensors
        if (i+1) % evaluation_steps == 0:           # Evaluate the model when we...
            evaluate_model()                        # ...have no gradients accumulate
```

这个方法主要是为了规避 GPU 内存的限制而开发的。

10 使用分布式数据并行进行多 GPU 训练

加速分布式训练可能有很多方法，但是简单的方法是使用 torch.nn.DistributedDataParallel 而不是 torch.nn.DataParallel。这样一来，每个 GPU 将由一个专用的 CPU 核心驱动，避免了 DataParallel 的 GIL 问题。

分布式训练文档地址：https://pytorch.org/tutorials/beginner/dist_overview.html

11 设置梯度为 None 而不是 0

梯度设置为. zero_grad(set_to_none=True) 而不是 .zero_grad()。这样做可以让内存分配器处理梯度，而不是将它们设置为 0。正如文档中所说，将梯度设置为 None 会产生适度的加速，但不要期待奇迹出现。注意，这样做也有缺点，详细信息请查看文档。

文档地址：https://pytorch.org/docs/stable/optim.html

12 使用. as_tensor() 而不是. tensor()

torch.tensor() 总是会复制数据。如果你要转换一个 numpy 数组，使用 torch.as_tensor() 或 torch.from_numpy() 来避免复制数据。

13 必要时打开调试工具

PyTorch 提供了很多调试工具，例如 autograd.profiler、autograd.grad_check、autograd.anomaly_detection。请确保当你需要调试时再打开调试器，不需要时要及时关掉，因为调试器会降低你的训练速度。

14 使用梯度裁剪

关于避免 RNN 中的梯度爆炸的问题，已经有一些实验和理论证实，梯度裁剪（gradient = min(gradient, threshold)）可以加速收敛。HuggingFace 的 Transformer 实现就是一个非常清晰的例子，说明了如何使用梯度裁剪。本文中提到的其他一些方法，如 AMP 也可以用。

在 PyTorch 中可以使用 torch.nn.utils.clip_grad_norm_来实现。

15 在 BatchNorm 之前关闭 bias

在开始 BatchNormalization 层之前关闭 bias 层。对于一个 2-D 卷积层，可以将 bias 关键字设置为 False：torch.nn.Conv2d(..., bias=False, ...)。

16 在验证期间关闭梯度计算

在验证期间关闭梯度计算，设置：torch.no_grad() 。

17 使用输入和 batch 归一化

要再三检查一下输入是否归一化？是否使用了 batch 归一化？

原文链接：https://efficientdl.com/faster-deep-learning-in-pytorch-a-guide/

公众号后台回复“对比学习综述”获取最新对比学习PDF资源

极市干货

极视角动态：极视角亮相BEYOND Expo，澳门特别行政区经济财政司司长李伟农一行莅临交流｜极视角助力构建城市大脑中枢，芜湖市湾沚区智慧城市运行管理中心上线！

数据集： 60+开源数据集资源大合集（医学图像、卫星图像、语义分割、自动驾驶、图像分类等）

多模态学习： CLIP：大规模语言-图像对比预训练实现不俗 Zero-Shot 性能｜ALBEF：图文对齐后再融合，借助动量蒸馏高效学习多模态表征

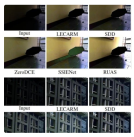
喜欢此内容的人还喜欢

YOLOv5帮助母猪产仔？南京农业大学研发母猪产仔检测模型并部署到Jetson Nano开发板

极市平台

ICCV23 | 将隐式神经表征用于低光增强，北大张健团队提出NeRCo

极市平台



实践教程 | 使用 OpenCV 进行特征提取（颜色、形状和纹理）

极市平台