

显存优化策略篇

来自：AiGC面试宝典

宁静致远

2024年01月27日 20:47



扫码
查看更

- 显存优化策略篇
 - 一、介绍一下 gradient accumulation 显存优化方式？
 - 二、介绍一下 gradient checkpointing 显存优化方式？
 - 致谢

一、介绍一下 gradient accumulation 显存优化方式？

正常情况下是一个batch之后统一计算梯度大小，gradient accumulation可以再指定个batch之后一起更新梯度，这种情况下，可以再batch_size很小的时候，提升真正的batch_size，是一种显存占用的优化算法。随着模型和数据规模越来越大，显存紧张的时候，需要把batch_size设置的很小，使用gradient accumulation的技术可以在实际上提高真正的batch_size【如果batch_size很小的话，会导致训练不稳定，收敛更慢】

梯度累积（Gradient Accumulation）是深度学习训练中的一种技术，用于在一次反向传播（backpropagation）中累积多个小批量数据的梯度，然后一次性更新模型参数。这个技术的主要目的是在内存有限的情况下，能够有效地使用大批量数据进行训练，从而提高模型性能。以下是梯度累积的详细解释：

- 背景：在深度学习中，通常会使用小批量随机梯度下降（Mini-batch Stochastic Gradient Descent，简称SGD）来训练模型。每个小批量数据都会计算一次梯度，并用这个梯度来更新模型参数。然而，在某些情况下，由于显存（GPU内存）的限制，无法一次性处理大批量数据。这可能会限制了模型的批量大小，从而影响了训练效率和性能。
- 梯度累积的原理：梯度累积的基本思想是，将多个小批量数据的梯度累积起来，然后一次性更新模型参数。具体操作是，对于每个小批量数据，计算其梯度，并将这些梯度累积在一起。当累积的梯度达到一定数量时（通常称为累积步数），才执行一次参数更新操作。
- 作用：梯度累积的主要作用有以下几点：
 - 内存效率：梯度累积允许在内存有限的情况下，使用更大的批量数据进行训练。虽然每个小批量数据的梯度会被累积，但累积的过程不会占用额外的内存空间，因此可以充分利用计算资源，提高训练效率。
 - 稳定性：大批量数据可能包含更全面和丰富的信息，可以减少梯度的方差，从而在训练过程中提供更稳定的梯度信号，有助于更快地收敛到较好的模型状态。
 - 参数更新频率控制：通过设置累积步数，可以控制参数更新的频率。这可以在训练过程中进行灵活的调整，以适应不同的硬件限制和训练需求。

尽管梯度累积可以提供上述优势，但也需要注意一些问题。较大的累积步数可能导致更新频率过低，从而降低训练速度。此外，累积梯度可能会导致一些优化算法的性能下降，因为一次性更新参数可能会影响动量和学习率等参数的计算。总之，梯度累积是一种有效的技术，可以在内存有限的情况下，充分利用大批量数据进行深度学习模型的训练，从而提高性能和效率。在使用梯度累积时，需要根据具体情况进行参数的设置和调整。

传统的梯度更新方式，对于每一个batch都进行损失计算和梯度更新：

```
for (inputs, labels) in data_loader:

    # extract inputs and labels
    inputs = inputs.to(device)
    labels = labels.to(device)
```

```

# passes and weights update
with torch.set_grad_enabled(True):

    # forward pass
    preds = model(inputs)
    loss = criterion(preds, labels)

    # backward pass
    loss.backward()

    # weights update
    optimizer.step()
    optimizer.zero_grad()

```

优化的梯度更新方式，计算的事loss的平均值，gradient_accumulation_steps 达到次数之后在进行参数更新：

```

gradient_accumulation_steps = 4

# loop through batches
for batch_idx, (inputs, labels) in enumerate(data_loader):

    # extract inputs and labels
    inputs = inputs.to(device)
    labels = labels.to(device)

    # passes and weights update
    with torch.set_grad_enabled(True):

        # forward pass
        preds = model(inputs)
        loss = criterion(preds, labels)

        loss /= gradient_accumulation_steps

        # backward pass => compute gradient
        loss.backward()

        if ((batch_idx + 1) % gradient_accumulation_steps == 0) or ((batch_idx+1)
== len(data_loader)):
            # weights update
            optimizer.step()
            optimizer.zero_grad()

```

二、介绍一下 gradient checkpointing 显存优化方式？

梯度检查点（Gradient Checkpointing）是一种优化深度学习模型训练中内存使用的技术。它通过在模型的计算图中插入检查点，将一部分计算推迟到后续步骤进行，从而减少内存占用。这有助于训练更大、更深的模型，以

及使用更大批量的数据，而无需占用大量的内存。

以下是梯度检查点的详细解释：

1. 背景：神经网络通常包含许多层和参数，模型的训练需要计算前向传播（forward pass）来获得预测结果，然后通过反向传播（backward pass）来计算梯度以进行参数更新。在大规模模型和数据集上训练时，这些计算可能会占用大量内存，尤其是在反向传播阶段需要存储大量的中间梯度。
2. 梯度检查点的原理：梯度检查点通过将计算图分段，在一部分计算完成后将部分中间结果存储起来，以便稍后在反向传播时使用。这些存储的中间结果称为“检查点”。在反向传播过程中，只需计算从检查点到当前节点的梯度，而不需要一次性计算从头开始的完整梯度流。这样，梯度检查点技术大大减少了内存需求，允许在有限内存下训练更大规模的模型。
3. 作用：梯度检查点技术有以下几个作用：
 - a. 减少内存压力：通过推迟部分计算和存储中间结果，梯度检查点技术降低了在反向传播过程中所需的内存量。这使得可以在有限的硬件资源下训练更大的模型。
 - b. 支持大批量训练：使用大批量数据进行训练可以加快收敛速度和稳定性，但可能会导致内存问题。梯度检查点允许在大批量训练中有效地使用内存。
 - c. 控制内存开销：梯度检查点技术允许研究人员和工程师在内存和计算资源有限的情况下仍然能够进行实验和研究，而无需投入昂贵的硬件。

需要注意的是，梯度检查点技术虽然在内存管理方面提供了优势，但同时也会引入一些计算开销，因为需要在前向传播过程中存储和管理检查点。因此，在实际使用时，需要综合考虑梯度检查点的优缺点。总之，梯度检查点是一种用于优化深度学习模型训练中内存使用的技术，通过延迟部分计算和存储中间结果，减少了反向传播过程中的内存需求，从而支持更大规模的模型和数据。