- 使用DeepSpeed加速ViT训练
  - 项目简介
  - 环境配置、设备需求以及数据准备
  - 模型训练
    - 模型训练的优化技术
      - ZeRO-DP
  - 项目成果
    - 性能测试
- 附录

使用DeepSpeed加速ViT训练

项目简介

DeepSpeed是微软开发的用于训练大型模型的深度学习优化库，它可以在单个GPU上训练具有数十亿参数的模型，也可以在数千个GPU上训练具有数万亿参数的模型。DeepSpeed的核心是ZeRO优化技术，它可以将模型参数分布在多个GPU上，从而减少单个GPU上的内存占用，使得可以训练更大的模型。除此之外，DeepSpeed还提供了一些其他的优化技术，例如：Gradient Accumulation、Fused Adam、Fused Lamb、Fused SGD、Memory Efficient FP16等。目前，DeepSpeed广泛应用于transformers、Megatron-LM、Megatron-11B等项目中。

ViT是一种基于Transformer的视觉模型，它可以将图像分割成一系列的图像块，然后将这些图像块转换成序列，最后使用Transformer对这些序列进行处理。ViT的训练非常耗费资源，例如：训练一个ViT-B/16模型需要使用8张V100 GPU训练3天。因此，我们需要使用DeepSpeed来加速ViT的训练。

本项目是使用公司人脸识别项目的子项目，主要分为特征提取和特征匹配问题。特征提取问题是将人脸图像转换成特征向量，特征匹配问题是将特征向量进行比对，从而判断两张人脸图像是否属于同一个人。在特征提取方面，使用ViT模型将和MLP模型进行训练，得到的特征通过ResNet进行匹配。我主要负责使用DeepSpeed、Apex来进行模型的训练加速以及模型的优化。

环境配置、设备需求以及数据准备

设备：V100 * 10，2080Ti * 10 数据准备：只提供了数据的接口，无法访问内部数据。但可以知道数据都是单通道的监控画面。

模型训练

模型的训练往往使用GPU，并占用大量的显存。更大的显存能够使得模型训练更稳定，更快速；因此，在训练过程中使用各种技术来减少显存开支，提高训练速度是非常重要的。

模型训练的优化技术

模型在训练过程中占用的显存主要可以分为以下两个部分：

- 模型状态(Model States)：模型的权重、梯度、优化器状态等；他们占用了显存的3/4，甚至更多；
- 缓冲区状态(Buffer Stages):这部分包含模型在进行forward和backward时候产生的中间状态，例如：输入、输出、梯度等；他们占用了显存的1/4。

为了减少上面两部分的显存，许多技术被发明出来。保存针对模型状态的Model Partion、Gradient Accumulation、Fused Adam、Fused Lamb、Fused SGD、Memory Efficient FP16等技术，可以减少模型状态的显存占用。针对Buffer States的技术Activation Checkpointing、Memory Efficient FP16等技术，可以减少缓冲区状态的显存占用。

DeepSpeed继承了上述多种技术，将之分别命名为ZeRO-DP（针对Model States）和ZeRO(针对Buffer States)，并且创新性提出了ZeRO-Offload技术、Optimizer-Partition、Paramters-Partition等技术。

ZeRO-DP

ZeRO-DP是DeepSpeed中用于减少模型状态显存占用的技术，它将模型参数分布在多个GPU上，从而减少单个GPU上的显存占用。ZeRO-DP的核心是将模型参数分布在多个GPU上，然后在每个GPU上进行forward和backward，最后将梯度进行聚合。ZeRO-DP的优点是可以在单个GPU上训练大型模型，缺点是需要在多个GPU上进行forward和backward，因此通信量会增加，从而导致训练速度变慢。

该优化技术主要分为三个Stage:

- Stage 1:将opimizer分布在多个GPU上，每个GPU上保存优化器的一个子集。
- Stage 2：将梯度分布在多个GPU上，每个GPU上保存梯度的一个子集，每个GPU只更新对应梯度的部分，并使用GPU通信将梯度聚合。
- Stage 3：将模型参数分布在多个GPU上，每个GPU上保存模型参数的一个子集。每次计算的使用，使用GPU通信将模型参数聚合。

DeepSpeed还提供一些其他的优化技术，例如：OffLoad、Adam_CPU、课程学习等技术，但是这些技术要么并非DeepSpeed的创新，要么并不能应用于ViT模型的训练，因此在这里不做介绍。

ZeRO的主要内容可以参考下述资料：

- DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters
- Github: DeepSpeed

项目成果

主要从两个方面来阐述项目的成果：

- 训练速度，训练速度以每秒钟处理的图像数量来衡量（以下简称QPS）。
- 模型最终的效果。人脸识别主要采用在False Acceptance(FA)一定的情况下，计算Precision。Precision越高，说明模型的效果越好。

	Positive	**Negative**
True	TP	TN
False	FP	FN

$FA = \frac{FP}{FP+TN}$ $Precision = \frac{TP}{TP+FN}$ FA是验证模型的可靠性，即不会讲不应该验证通过的人放入进来。Precision是验证模型的准确性，即它能够准备地识别特征，不要将正确的人拦在外面。 一般会以96.78@0.01类似的形式给出，表示在FA=0.01的情况下，Precision=96.78%。在应用中的场景即为：未经验证的人进来的概率低于1%的可靠性下，模型的能准确识别出正确的人的概率为96.78%。

性能测试

实验配置

EID	Batch Size	DeepSpeed Config	Training Setting
Base	440	None	torch optimizer
deepspeed_01	440	stage=2,backbone,off_load=false	deepspeed optimizer and scheduler
deepspeed_02	584	stage=2,backbone,off_load=true	deepspeed optimizer and scheduler
deepspeed_03	660	Stage=3,off_load=True	deepspeed optimizer and scheduler
deepspeed_04	600	stage=3,off_load=False	deepspeed optimizer and scheduler

实验结果

EID	QPS	显存占用	Notes
Base	2500	15950/16150	
deepspeed_01	2600~2650	14000/16150	
deepspeed_02	2800~2850	15900/16150	
deepspeed_03	>2950	15900/16150	
deepspeed_04	2500~2550	15900/16150	使用off_load占用cpu内存

性能结果

以数据集1，2，3代替真实数据集名称 以Precision@FA展示数据

EID	D1	D2	D3	Note
Base	96.78@0.01 99.10@0.1	90.01@0.01 99.78@0.1	76.78@0.01 90.29@.01	
deepspeed_01	96.98@0.01 99.40@0.1	90.21@0.01 99.70@0.1	77.78@0.01 91.29@.01	
deepspeed_02	96.01@0.01 98.70@0.1	92.21@0.01 99.79@0.1	78.78@0.01 93.29@.01	
deepspeed_03	95.72@0.01 99.80@0.1	92.07@0.01 99.80@0.1	76.90@0.01 92.29@.01	
deepspeed_04	95.28@0.01 98.10@0.1	89.11@0.01 97.78@0.1	75.28@0.01 89.29@.01	

附录

附上一些实习做的笔记，删除涉密的部分。

1. fp16 scale error

描述：当deepspeed打开fp16时，在设置warmupLR时候，最初的学习率设置太高会导致梯度爆炸，从而导致直接退出训练。

解决办法：调小最初的学习率，<0.01的学习率是可以正常训练的。

2. torch的distributed和deepspeed的zero无法兼容，详情参见 https://github.com/microsoft/DeepSpeedExamples/issues/86

描述：torch的多卡并行无法和deepspeed的zero partition兼容，不过仍然可以使用deepspeed的其他特性，通过设置stage = 0,使用deepspeed可以封装模型，并且如果deepspeed同时封装backbone和fc，并且同时开启fp16，会导致loss出现inf。通过backbone开启fp16,fc关闭fp16可以解决。

3. v1实验使用stage=3，stage3相比于2,显存占用和qps都变差了很多。一天后修改，stage3一定要设置offload，包括offload_params和offload_optimizer,包括bucket_size,group_size等，需要精细设置，经过设置之后，显存从9200降至8400，但Qps有1300降至1100.
4. deepspeed的ZeRO和3D parallelism（tensor-slicing, pipeline-parallelism, and data parallelism)只能选择其中一种使用，pipeline-pm指的是forward和backward在不同通道并行实现。将fc中的distributed替换为deepspeed的comm实现，经过测试，性能差别不明显。ZeRO和3D parallelism参见：https://www.deepspeed.ai/tutorials/large-models-w-deepspeed/
5. backbone、fc都是用deepspeed的时候，warmupLR倾向于先使用较小的lr.
6. 以Adam为具体例子，使用Adam对具有Ψ个参数的模型进行混合精度训练需要足够的内存来保存参数和梯度的fp16副本，分别需要2Ψ和2Ψ字节的内存。此外，还需要额外的内存来保存优化器状态：参数、动量和方差的fp32副本(为了更新参数)，分别需要4Ψ、4Ψ和4Ψ字节的内存。让我们用K表示优化器状态的内存倍增系数，即额外需要存储它们的内存为KΨ字节。混合精度Adam的K值为12。总体而言，这将导致16Ψ字节的内存需求。对于像GPT-2这样具有15亿个参数的模型，这将至少需要24GB的内存，远远高于仅存储fp16参数所需的3GB微不足道的内存。所以总共的模型所需要的内存为：$4\Psi + K\Psi + 2\Psi + 2\Psi = 8\Psi + K\Psi$。其中$4\Psi$为参数的fp32副本，$K\Psi$为优化器状态的fp32副本，$2\Psi$为参数的fp16副本，$2\Psi$为梯度的fp16副本。如果是Adam优化器，$K = 12$，总共需要$16\Psi$的内存。

Stage	特点
1	Optimizer Partition。将优化器的状态分割到多个设备，而不是每个设备保存一部分。因此，需要消耗的显存变为了 4Ψ + K/D ,如果D足够大，那么相当于只占用了原来1/4的显存（使用Adam）。
2	+ gradient Partition.不在每个设备都保存梯度，将梯度分割。显存变味了 2Ψ + （2+K）/D,因此相当于使用原来 1/8的显存
3 and Infinity	+ parameter Partition.将模型参数都分割到多张显卡，（4Ψ + kΨ)/D,因此只要显卡足够就可以训练，但是stage 3是以通信量增加到原来的1.5倍为代价大幅降低显存的。在单节点方面，不需要特别调整，但在多节点训练，可能需要一些工程方面的方法，详情参见论文。

问题分析：分析log时候可以发现，某些连续的几个batch导致梯度变化很大，最终导致了fp16溢出，从而使得Engine不断地下调loss_scale，最终导致loss_scale下溢。

10. offload_optimizer可以在stage=2的时候使用，使用之后能够进一步提高QPS和batch_size。
11. deepspeed随着显卡数量的增加，单卡的QPS也在增加。10 V100相比于 9 张V100，单卡提升了100 QPS.

12.较小的batch_size会导致loss下降更快一些。

13.deepspeed保存的checkpoint进行移植复现的时候，要求显卡数量相同，所以V100*10无法在1080 * 8上进行复现。Debug时候最好使用8 * V100，方便之后在8卡上进行复现。

14.使用deepspeed的模型，在进行保存的使用使用torch.save(ds_model.module)保存的模型在进行测试的使用效果接近于没有训练，这可能是无法使用这种形式对deepspeed的模型进行保存。使用deepspeed的save_checkpoint函数保存的模型可以一直到其他具有相同GPU数量的机器上（通过在V100上训练，在WXRG164上resume，通过观察loss证明模型可以复现）

15、使用deepspeed.init_inference，torch，和deepspeed.initialize三种方法尝试载入模型，并进行推断，最终得到结果都很差。经过大佬查看模型内部参数发现，deepspeed在保存参数的时候使用了fp16，导致最后一层norm的bias溢出为inf，最终导致了inference上极差的结果。

目前没有查找到deepspeed将fp16模型转换为fp32模型的接口 ，可能需要在保存的时候进行强制的类型转换。 经过测试，使用deepspeed save_chechpoint函数和使用torch save函数保存的模型参数是一样的，deepspeed模型除了保存模型参数（在model['module']内），还保存了一些world_size,global_step，ds_config之类的设置。 训练的模型保存的参数只有feature.1.runnning_var溢出（那一层只有一个数字没有溢出），并且每一个保存的模型都表现一致：所有保存的模型都溢出，并且溢出的都是那一层，并且都是只有那一个特定的数字没有溢出。 是否是torch内部训练时候存在某种机制，使用torch（没有使用deepspeed）载入具有某一层的模型，最终的loss也是2.6左右，完全没有受到feature层具有inf的影响。 FIX：：将deepspeed的模型保存替换掉feature.1.running_bias这一层的参数，使用torch.load当作普通torch模型载入，并训练不到100step(<1 min),然后使用torch.save保存，就可以得到fp32的模型（模型大小由 243MB → 485MB）。

16. 在对deepspeed所保存模型进行微调的时候，loss不断上升，有3.5一直上升到5.2；

尝试降低学习率，重新进行微调。 需要调小学习率的同时降低momentum，能够使得loss正常下降。 17.在进行finetune时候，训练了4000steps,性能和训练几百个steps的性能结果差不多 ，并且loss也差不多，而且固定参数训练和不固定参数训练结果也差不多。

18. 谨慎相信官方文档里面的内容：https://www.deepspeed.ai/tutorials/inference-tutorial/#loading-checkpoints。实测，init_inference函数通过指定checkpoint的方法根本无法进行模型载入，估计很久没更新了。

19.deepspeed的问题的本质如下：

保存的参数的溢出时feature的第一层BN层的running_var，这个参数只会记录所有batch的方差的平均值，它会参与后面的inference，而不会参与训练，训练使用的var是当时的batch的var。 这个running_var也说明模型特征的方差很大，模型具有分辨能力。这个var会在inference时作为当时的var除数，导致得到的特征趋近于0，最终导致结果很差。这也能说明为什么训练时候resume是完全没有问题，而inference出现了问题

20. 虽然使用torch也可以保存stage=2时候的deepspeed模型，但经过测试，使用torch保存的模型和使用deepspeed保存的模型并不一致，除了一些deepspeed的设置意外，保存的模型参数也不一致，经过测试他们的模型的参数差别在1e-3内，并且并不是固定的。

21.手动设置BN的running_Var,running_Mean为fp32，经过测试不会影响训练，同时保存模型的running_Var和Mean也确实为fp32类型的数据。

22. deepspeed的WarmupDecayLR的学习率规划器和我们实验的表现并不一致，导致backbone和fc的学习率不一致，很可能这是导致BN层溢出的原因。

> 显存的占用：模型包含的参数、优化器存储的有关参数冲量等状态、forward和backward产生的某些中间结果需要占用显存（deep speed中的ZeRO-R就是手机中间状态产生的显存碎片来减少显存使用的。

> FC在我们的模型中使用了torch的DDP，这和deepspeed的ZeRO是无法一起使用的，deepspeed中提供了torch的DDP类似的deepspeed.cmm，deepspeed的cmm也无法和zero一起使用，因此FC这部分显存很难优化。对于backbone这部分，可以使用deepspeed的ZeRo optimization。它提供了三个优化等级，分别对应optimizer state partitioning, and optimizer+gradient state partitioning, and optimizer+gradient+parameter partitioning。

> 不同stage的表现： 目前主要对stage2和stage3进行了一些实验。 目前在运行的实验主要是基于stage2的，目前stage2 + off_optimizer能够取得最好的速度，并且性能上能够初步对齐实验。查阅的一些资料显示，stage3不进行off_params设置能够在性能上和stage2相当，但之前进行实验的速度会差很多，可能是stage3默认开启off_params,导致虽然没有进行显示的配置，但仍然使用了off_params。 stage3: 官方的文档写道：stage3提供了更好的扩展性，stage 3的optimization提供了很多参数，包括下面提到的种种参数，它能够使得用户能够在通信时间、显存和计算性能上进行权衡，从而更好地加速模型的训练，但目前还没有对于这部分的配置有详细的实验。

> 目前还只是尝试了deepspeed的初步配置，deepspeed还提供了一些性能调优的参数，包括 `overlap_comm`,和 `allgather_bucket_size`和 `reduce_bucket_size`."overlap_comm": true 会增加GPU RAM的使用，以降低all-reduce延迟。deepspeed中最终要的配置就是ZeRO Optimization的配置，如果后续还需要进一步的研究，主要是测试针对它的这一部分进行进一步的部署。如果针对大模型，如LLM等，deepspeed提供了ZeRO Infinity的概念，这个方面目前还没有详细了解，不过可以根据论文和Github所言，大约能加速2x ~ 5x 的速度。

> 经过分析，最能节省显存并提高训练速度的是fp16，如果不开启fp16,deepspeed的提升并不明显，大约在50 QPS左右，因此deepspeed节省的显存主要是和使用fp16替代fp32。其次，在相同的batch_size下训练，不开启fp16，deepspeed训练提升不明显，开启fp16之后，提升了不到10%，显存下降了10%，这是相比于整个模型，包括fc。

> 在我们的模型的整个训练期间，性能瓶颈应当是FC，它的参数应当是ViT模型参数量的两倍多。因此相当于ViT的性能提升了约30%，才能达到整体性能提高约10%