深度之眼
deepshare.net

GPU分布式训练

导师：GAUSS

深度之眼
deepshare.net

目录

深度之眼
deepshare.net

分布式训练理论简介

分布式训练简介

总的来说，分布式训练分为这几类：

按照并行方式来分：模型并行 vs 数据并行

按照更新方式来分：同步更新 vs 异步更新

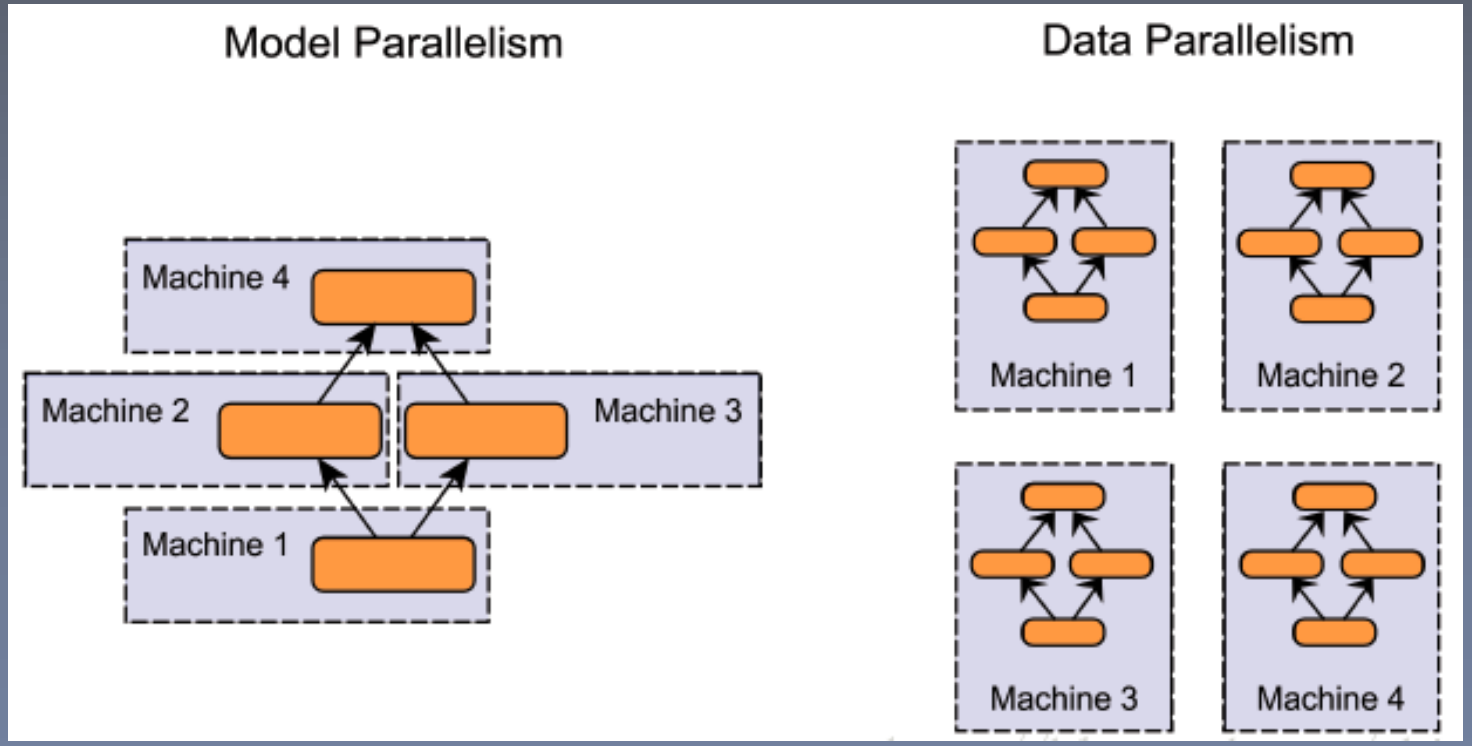
按照算法来分：Parameter Server算法 vs AllReduce算法

模型并行 vs 数据并行

假设我们有n张GPU：

模型并行：不同的GPU输入相同的数据，运行模型的不同部分，比如多层网络的不同层；

数据并行：不同的GPU输入不同的数据，运行相同的完整的模型。



模型并行 vs 数据并行

当模型非常大，一张GPU已经存不下的时候，可以使用模型并行，把模型的不同部分交给不同的机器负责，但是这样会带来很大的通信开销，而且模型并行各个部分存在一定的依赖，规模伸缩性差。因此，通常一张可以放下一个模型的时候，会采用数据并行的方式，各部分独立，伸缩性好。

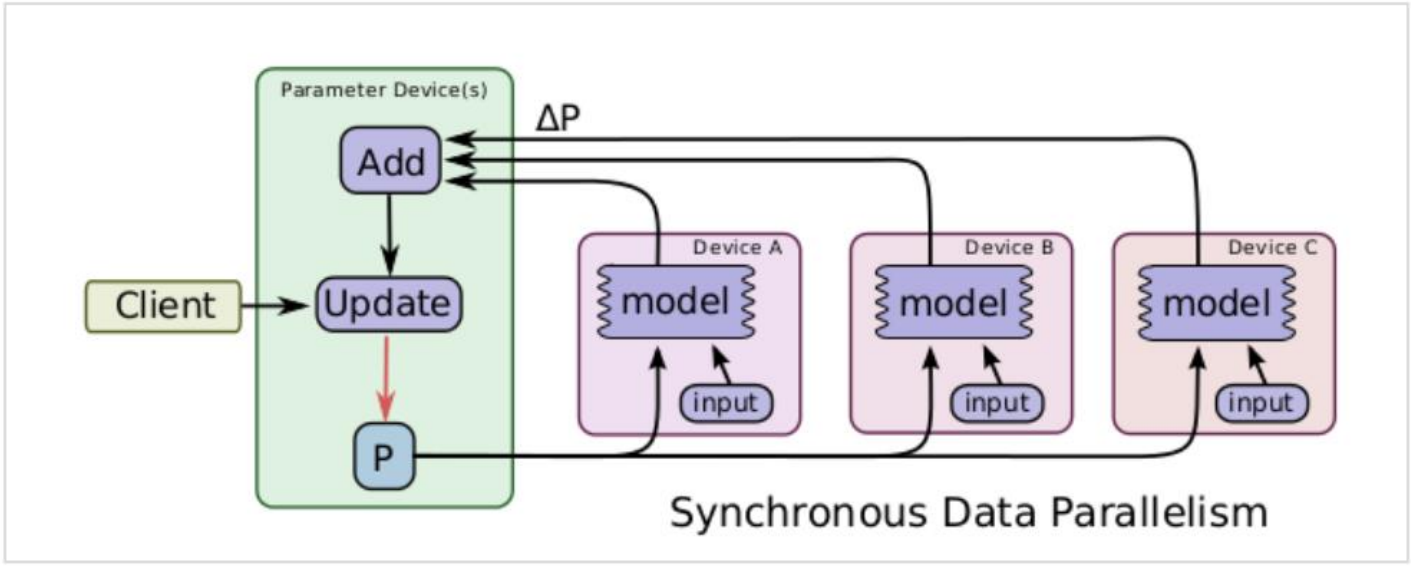
同步更新 vs 异步更新

对于数据并行来说，由于每个GPU负责一部分数据，那就涉及到如果更新参数的问题，分为同步更新和异步更新两种方式。

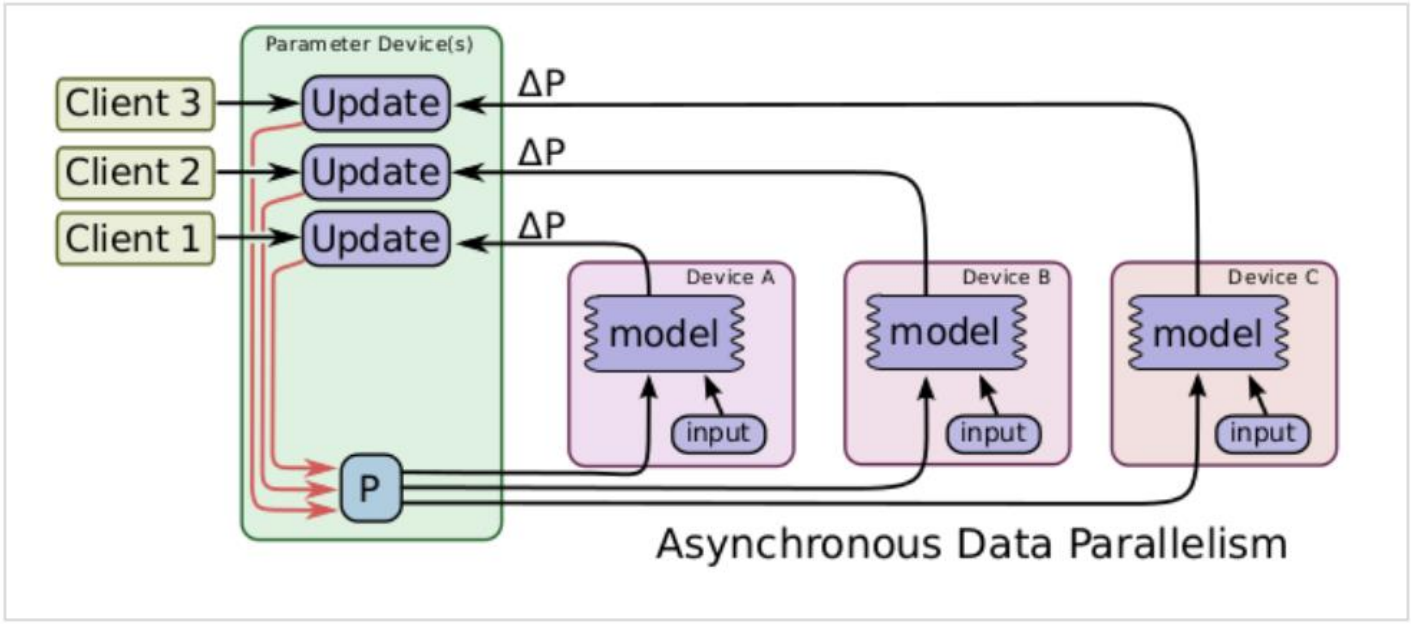
- 同步更新：每个batch所有GPU计算完成后，再统一计算新权值，然后所有GPU同步新值后，再进行下一轮计算。
- 异步更新：每个GPU计算完梯度后，无需等待其他更新，立即更新整体权值并同步。

同步更新 vs 异步更

同步更新有等待，速度取决于最慢的那个GPU；异步更新没有等待，但是涉及到更复杂的梯度过时，loss下降抖动大的问题。所以实践中，一般使用同步更新的方式。



同步更新



异步更新

Parameter Server算法 vs Ring AllReduce算法

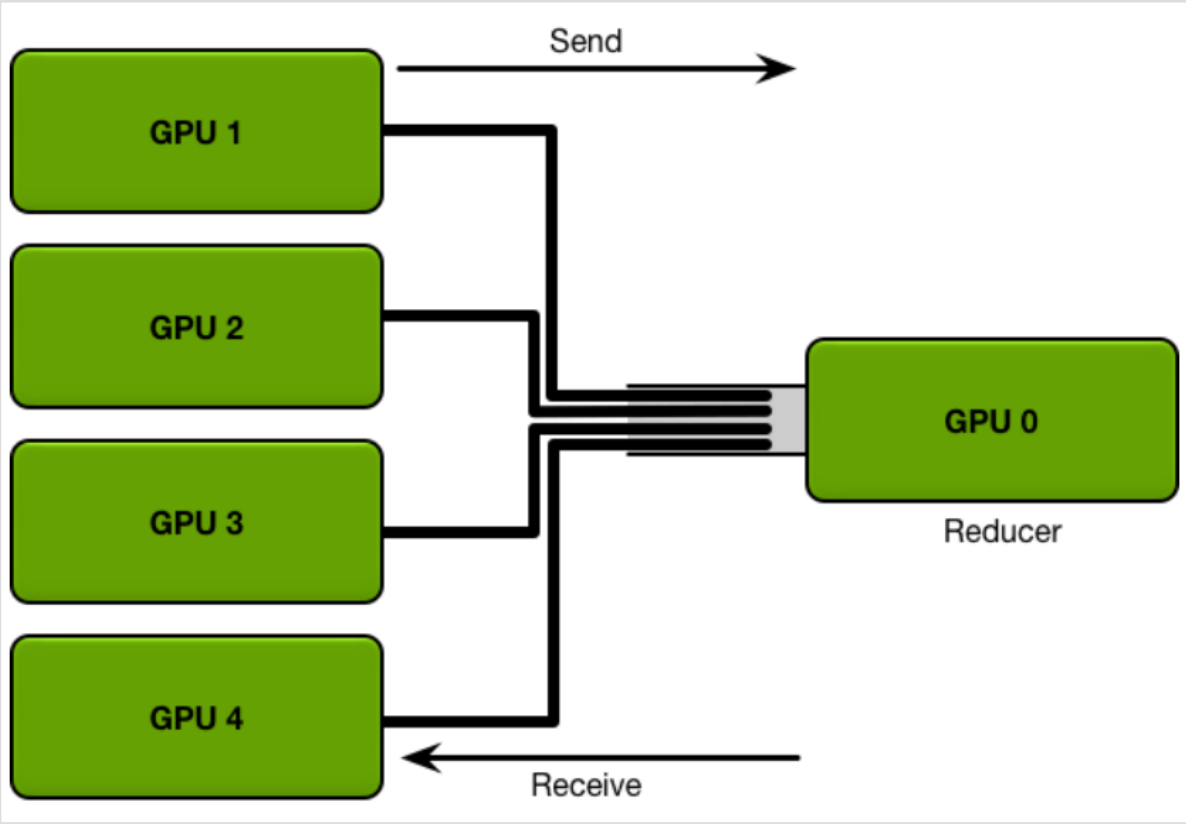
这里讲一下常用的两种参数同步的算法：PS 和 Ring AllReduce。

假设有5张GPU：

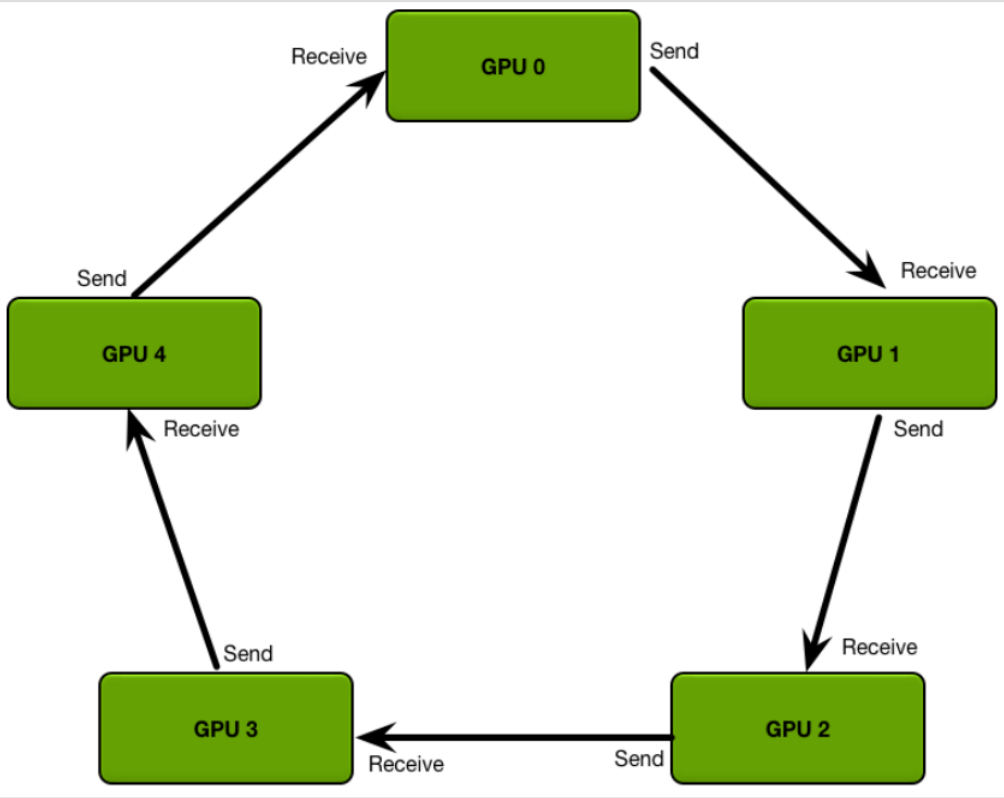
Parameter Server：GPU 0将数据分成五份分到各个卡上，每张卡负责自己的那一份mini-batch的训练，得到grad后，返回给GPU 0上做累积，得到更新的权重参数后，再分发给各个卡。

Ring AllReduce：5张以环形相连，每张卡都有左手卡和右手卡，一个负责接收，一个负责发送，循环4次完成梯度累积，再循环4次做参数同步。分为Scatter Reduce和All Gather两个环节。

Parameter Server算法 vs Ring AllReduce算法



Parameter server算法架构



Ring AllReduce算法架构

深度之眼
deepshare.net

分布式训练API接口

tf.distribute.Strategy简介

tf.distribute.Strategy是TensorFlow API，用于在多个GPU，多台机器或TPU之间分布式训练。使用此API，你可以在更改最少的代码情况下分发现有模型和训练代码。

tf.distribute.Strategy 设计时考虑了以下主要目标：

- 易于使用并支持多个用户细分，包括研究人员，ML工程师等。
- 开箱即用地提供良好的性能。
- 轻松切换策略。

tf.distribute.Strategy可以与Keras之类的高级API 一起使用，也可以用于分发自定义训练模型（以及通常使用TensorFlow进行的任何计算）。

tf.distribute.Strategy简介

tf.distribute.Strategy打算涵盖不同方面的许多用例。目前支持其中一些组合，将来会添加其他组合。其中一些轴是：

同步训练与异步训练：这是通过数据并行性分布训练的两种常用方法。在同步培训中，所有工作人员都同步地对输入数据的不同片段进行培训，并在每个步骤中汇总梯度。在异步培训中，所有工作人员都在独立训练输入数据并异步更新变量。通常情况下，同步训练通过全约简和参数服务器架构的异步支持。

硬件平台：您可能希望将培训扩展到一台计算机上的多个GPU或网络中的多台计算机（每个具有0个或多个GPU）或Cloud TPU上。

tf.distribute.Strategy简介

为了支持这些用例，提供了六种策略。当前在TF 2.2中的哪些方案中支持其中的哪些：

Training API	MirroredStrategy	TPUStrategy	MultiWorkerMirroredStrategy	CentralStorageStrategy	ParameterServerStrategy
Keras API	Supported	Supported	Experimental support	Experimental support	Supported planned post 2.3
Custom training loop	Supported	Supported	Experimental support	Experimental support	Supported planned post 2.3
Estimator API	Limited Support	Not supported	Limited Support	Limited Support	Limited Support

MirroredStrategy

tf.distribute.MirroredStrategy支持在一台机器上在多个GPU上进行同步分布式训练。每个GPU设备创建一个副本。模型中的每个变量都将在所有副本之间进行保存。这些变量共同构成一个称为的概念变量MirroredVariable。通过应用相同的更新，这些变量彼此保持同步。

高效的all-reduce算法用于在设备之间传递变量更新。 all-reduce通过将张量相加来汇总所有设备上的张量，并使它们在每个设备上可用。这是一种非常有效的融合算法，可以大大减少同步的开销。根据设备之间可用的通信类型，有许多all-reduce算法和实现可用。默认情况下，它使用NVIDIA NCCL作为全缩减实现。您可以从我们提供的其他选项中进行选择，也可以自己编写。

MirroredStrategy

这是最简单的创建方法MirroredStrategy：

```
mirrored_strategy = tf.distribute.MirroredStrategy()
```

这将创建一个MirroredStrategy实例，该实例将使用TensorFlow可见的所有GPU，并将NCCL用作跨设备通信。

如果您只想使用计算机上的某些GPU，可以这样做：

```
mirrored_strategy = tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])
```

TPUStrategy

tf.distribute.experimental.TPUStrategy让你在Tensor处理单元（TPU）上运行TensorFlow训练。TPU是Google的专用ASIC，旨在极大地加快机器学习的工作量。可在Google Colab，TensorFlow Research Cloud和Cloud TPU上使用它们。

就分布式训练架构而言，TPUStrategy是相同的MirroredStrategy-它实现了同步分布式训练。TPU提供了自己的跨多个TPU核心的高效all-reduce和其他集合操作的实现，这些核心在TPUStrategy使用。

TPUStrategy

这是实例化的方式TPUStrategy：

```
cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu=tpu_address)
tf.config.experimental_connect_to_cluster(cluster_resolver)
tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
tpu_strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
```

该TPUClusterResolver实例有助于定位TPU。在Colab中，你无需为其指定任何参数。

如果要将其用于Cloud TPU：

- 你必须在TPU参数中指定TPU资源的名称。
- 你必须在程序开始时显式初始化TPU系统。在将TPU用于计算之前，这是必需的。初始化TPU系统还会擦除TPU内存，因此，请务必首先完成此步骤，以免丢失状态。

MultiWorkerMirroredStrategy

tf.distribute.experimental.MultiWorkerMirroredStrategy与MirroredStrategy非常相似。它实现了跨多个工作机器的同步分布式训练，每个工作机器可能具有多个GPU。与MirroredStrategy相似，它会在所有工作机器的每台设备上的模型中创建所有变量的副本。

它使用CollectiveOps作为多机器all-reduce通信方法，用于使变量保持同步。集合运算是TensorFlow图中的单个运算，它可以根据硬件，网络拓扑和张量大小在TensorFlow运行时中自动选择all-reduce算法。

MultiWorkerMirroredStrategy

这是最简单的创建方法MultiWorkerMirroredStrategy：

```
multiworker_strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy()
```

MultiWorkerMirroredStrategy现在还可以在集体操作的两种不同实现之间进行选择。CollectiveCommunication.RING基于环的集合使用gRPC作为通信层实现。CollectiveCommunication.NCCL使用Nvidia的NCCL实施集体。CollectiveCommunication.AUTO将选择推迟到运行时。集体实施的最佳选择取决于GPU的数量和种类以及集群中的网络互连。您可以通过以下方式指定它们：

```
multiworker_strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy(
    tf.distribute.experimental.CollectiveCommunication.NCCL)
```

CentralStorageStrategy

tf.distribute.experimental.CentralStorageStrategy也进行同步训练。变量不会被镜像，而是被放置在CPU上，并且操作会在所有本地GPU之间复制。如果只有一个GPU，则所有变量和操作都将放置在该GPU上。

创建CentralStorageStrategyby 的实例：

```
central_storage_strategy = tf.distribute.experimental.CentralStorageStrategy()
```

这将创建一个CentralStorageStrategy实例，该实例将使用所有可见的GPU和CPU。在副本上对变量的更新将在应用于变量之前进行汇总。

ParameterServerStrategy

tf.distribute.experimental.ParameterServerStrategy支持多台机器上的参数服务器训练。在此设置中，某些机器被指定为工作器，而另一些被指定为参数服务器。模型的每个变量都放在一个参数服务器上。计算在所有工作程序的所有GPU之间复制。

在代码方面，它看起来与其他策略类似：

```
ps_strategy = tf.distribute.experimental.ParameterServerStrategy()
```

深度之眼
deepshare.net

Keras方式分布式训练

Keras方式分布式训练

我们已经将TensorFlow的Keras API规范实现集成tf.distribute.Strategy到tf.keras其中 。 是用于构建和训练模型的高级API。通过集成到后端，我们使您可以无缝地分发在Keras训练框架中编写的训练。

您需要在代码中进行以下更改：

- 创建一个适当的实例tf.distribute.Strategy。
- 将Keras模型，优化器和指标的创建移到内部strategy.scope。

Keras方式分布式训练

我们支持所有类型的Keras模型-顺序，功能和子类化。

这是一段代码的片段，用于一个非常简单的Keras模型（具有一个密集层）：

```
mirrored_strategy = tf.distribute.MirroredStrategy()
with mirrored_strategy.scope():
        model = tf.keras.Sequential([tf.keras.layers.Dense(1, input_shape=(1,))])
        model.compile(loss='mse', optimizer='sgd')
dataset = tf.data.Dataset.from_tensors(([1.], [1.])).repeat(100).batch(10)
model.fit(dataset, epochs=2)
model.evaluate(dataset)
```

Keras方式分布式训练

接下来，我们创建输入数据集并调用tf.distribute.Strategy.experimental_distribute_dataset以根据策略分配数据集。

```
dataset = tf.data.Dataset.from_tensors(([1.], [1.])).repeat(100).batch(global_batch_size)
dist_dataset = mirrored_strategy.experimental_distribute_dataset(dataset)
```

深度之眼
deepshare.net

自定义方式分布式训练

自定义方式分布式训练

我们将用于tf.GradientTape计算梯度，优化器将应用这些梯度来更新模型变量。为了分发此训练步骤，我们放入一个函数train_step并将其tf.distrbute.Strategy.run与我们dist_dataset之前创建的数据集输入一起传递给：

```
·loss_object = tf.keras.losses.BinaryCrossentropy(from_logits=True, reduction=tf.keras.losses.Reduction.NONE)

def compute_loss(labels, predictions):
  per_example_loss = loss_object(labels, predictions)
  return tf.nn.compute_average_loss(per_example_loss, global_batch_size=global_batch_size)

def train_step(inputs):
  features, labels = inputs

  with tf.GradientTape() as tape:
    predictions = model(features, training=True)
    loss = compute_loss(labels, predictions)

  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))
  return loss

@tf.function
def distributed_train_step(dist_inputs):
  per_replica_losses = mirrored_strategy.run(train_step, args=(dist_inputs,))
  return mirrored_strategy.reduce(tf.distribute.ReduceOp.SUM, per_replica_losses, axis=None)
```

自定义方式分布式训练

上面的代码中还有一些其他注意事项：

我们使用**tf.nn.compute_average_loss**计算损失。tf.nn.compute_average_loss对每个示例求和，然后将总和除以**global_batch_size**。这很重要，因为稍后在每个副本上计算出梯度后，它们会通过对它们求和而在副本之间进行汇总。

我们使用tf.distribute.Strategy.reduce API来汇总所返回的结果tf.distribute.Strategy.run。tf.distribute.Strategy.run从策略中的每个本地副本返回结果，并且有多种方法可以使用此结果。您可以reduce让他们获得合计值。您还tf.distribute.Strategy.experimental_local_results可以获取结果中包含的值列表，每个本地副本一个。

在apply_gradients分配策略范围内被调用时，其行为将被修改。具体而言，在同步训练期间将梯度应用于每个并行实例之前，它会执行所有梯度的总和。

限制GPU内存增长

默认情况下，TensorFlow会映射CUDA_VISIBLE_DEVICES该进程可见的所有GPU中的几乎所有GPU内存（视情况而定 ）。这样做是为了通过减少内存碎片来更有效地使用设备上相对宝贵的GPU内存资源。为了将TensorFlow限制为一组特定的GPU，我们使用该tf.config.experimental.set_visible_devices方法。

限制GPU内存增长

第一种选择是通过调用来打开内存增长tf.config.experimental.set_memory_growth，该尝试尝试仅分配运行时分配所需的GPU内存：开始分配的内存很少，随着程序的运行和需要更多的GPU内存，我们扩展分配给TensorFlow进程的GPU内存区域。请注意，我们不会释放内存，因为它可能导致内存碎片。要打开特定GPU的内存增长，请在分配任何张量或执行任何操作之前使用以下代码。

限制GPU内存增长


```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
  # Restrict TensorFlow to only use the first GPU
  try:
    tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
    logical_gpus = tf.config.experimental.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPU")
  except RuntimeError as e:
    # Visible devices must be set before GPUs have been initialized
    print(e)
```

限制GPU内存增长

第二种方法是使用来配置虚拟GPU设备，tf.config.experimental.set_virtual_device_configuration并对要在GPU上分配的总内存设置硬限制。

限制GPU内存增长


```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
  # Restrict TensorFlow to only allocate 1GB of memory on the first GPU
  try:
    tf.config.experimental.set_virtual_device_configuration(
        gpus[0],
        [tf.config.experimental.VirtualDeviceConfiguration(memory_limit=1024)])
    logical_gpus = tf.config.experimental.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
  except RuntimeError as e:
    # Virtual devices must be set before GPUs have been initialized
    print(e)
```

虚拟多个GPU

针对多个GPU的开发将使模型可以使用其他资源进行扩展。如果在具有单个GPU的系统上进行开发，我们可以使用虚拟设备模拟多个GPU。这样可以轻松测试多GPU设置，而无需其他资源。

```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
  # Create 2 virtual GPUs with 1GB memory each
  try:
    tf.config.experimental.set_virtual_device_configuration(
        gpus[0],
        [tf.config.experimental.VirtualDeviceConfiguration(memory_limit=1024),
         tf.config.experimental.VirtualDeviceConfiguration(memory_limit=1024)])
    logical_gpus = tf.config.experimental.list_logical_devices('GPU')
    print(len(gpus), "Physical GPU,", len(logical_gpus), "Logical GPUs")
  except RuntimeError as e:
    # Virtual devices must be set before GPUs have been initialized
    print(e)
```

深度之眼
deepshare.net

实战六：Google涂鸦识别挑战赛GPU分布式训练

实战六：Google涂鸦识别挑战赛GPU分布式训练

背景数据相关介绍：

参考week5

- Keras方式分布式训练模型
- 自定义方式分布式训练模型

深度之眼
deepshare.net

总结

本节小结

Summary

GPU分布式训练	分布式训练理论简介	
	分布式训练API接口	
	Keras方式分布式训练	
	自定义方式分布式训练	
	实战六：Google涂鸦识别挑战赛GPU分布式训练	

结语

我 说

看过千万代码，不如实践一把！

深度之眼
deepshare.net

联系我们：

电话：18001992849

邮箱：service@deepshare.net

Q Q： 2677693114

公众号

客服微信