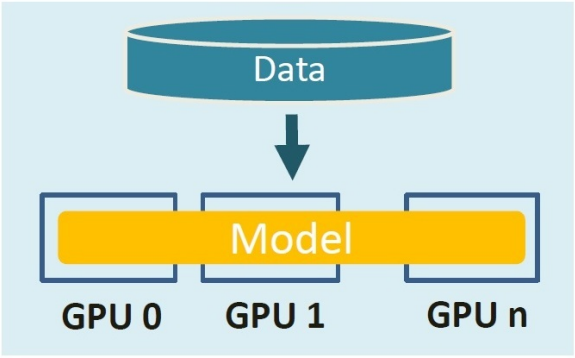
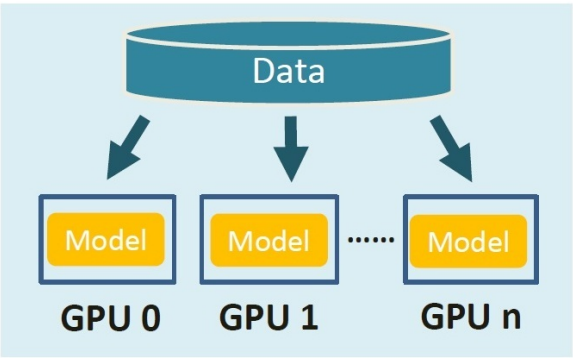
深度学习分布式训练相关介绍 - Part 1 多GPU训练

zhuanlan.zhihu.com/p/70312627

Zhang Bin 心有猛虎 细嗅蔷薇





本篇文章主要是对深度学习中运用多GPU进行训练的一些基本的知识点进行的一个梳理

文章中的内容都是经过认真地分析，并且尽量做到有所考证

抛砖引玉，希望可以给大家有更多的启发，并能有所收获

介绍

大多数时候，梯度下降算法的训练需要较大的Batch Size才能获得良好性能。而当我们选择比较大型的网络时候，由于GPU资源有限，我们往往要减少样本数据的Batch Size。

> 当GPU无法存储足够的训练样本时，我们该如何在更大的batch size上进行训练？

面对这个问题，事实上我们有几种工具、技巧可以选择，它们也是下文中将介绍的内容。

在这篇文章中，我们将探讨：

- 多GPU训练和单GPU训练有什么区别
- 如何最充分地使用多GPU机器
- 如何进行多机多卡训练？

更多关于多机多卡的分布式训练的详细架构理解和实践请参考我的下一篇文章：

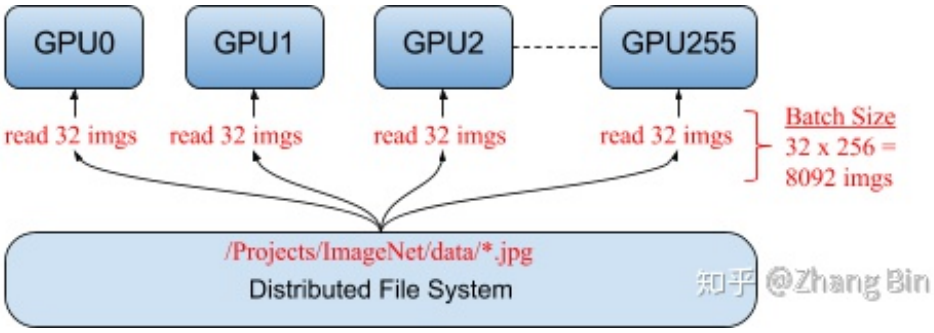
Zhang Bin：深度学习分布式训练相关介绍 - Part 2 详解分布式训练架构PS-Worker与Horovod
zhuanlan.zhihu.com
本文章介绍的内容在框架间是通用的，代码示例为：在不借助外部框架的情况下，将单GPU训练TensorFlow代码改为支持多GPU的训练代码

单GPU训练 vs 多GPU训练

单GPU训练 一般代码比较简单，并且能满足我们的基本需求，通常做法是设定变量CUDA_VISIBLE_DEVICES的值为某一块GPU来Mask我们机器上的GPU设备，虽然有时当我们忘了设定该变量时程序会自动占用所有的GPU资源，但如果没有相应的代码去分配掌控GPU资源的使用的话，程序还是只会利用到第一张卡的计算资源，其他的资源则仅是占用浪费状态。

多GPU训练 则可以从两个方面提升我们模型训练的上限：1. 超过单卡显存上限的模型大小， 2. 更大的Batch Size和更快训练速度。相应的，目前各大主流框架的多GPU训练一般存在两种模式：

- **模型并行** ：分布式系统中的不同GPU负责网络模型的不同部分，进而可以 **构建超过单卡显存容量大小的模型** 。比如，可以将神经网络的不同层分配到不同的GPU设备，或者将不同的参数变量分配到不同的GPU设备。
- **数据并行** ：不同的 GPU设备有同一模型的多个副本，将数据分片并分配到每个GPU上，然后将所有GPU的计算结果按照某种方式合并，进而可以**增加训练数据的Batch Size**。



此外，从主机的数量的角度来讲还存在 **单机多卡**和**多机多卡（分布式）** 的区别：

- **单机多卡**：只需运行一份代码，由该代码分配该台机器上GPU资源的使用
- **多机多卡**：每台机器上都需要运行一份代码，机器之间需要互相通信传递梯度，并且模型参数的更新也存在**同步**训练模式和**异步**训练模式的区别

多GPU机器的充分利用

这一节，将详细探讨一下多GPU的机器该如何利用。

对于多GPU训练，一般我们需要用**数据并行**的模式比较多，通过增大 Batch Size 并辅以较高的 Learning Rate 可以加快模型的收敛速度。

由于我们模型是通过若干步梯度反向传播来迭代收敛模型的参数，而每一步的梯度由一个Batch内样本数据的损失情况（Loss）得到，因此当 Batch Size 比较大时，可以减少 Batch样本的分布和整体样本的分布偏差太大的风险，进而使得每一步的梯度更新方向更加准确。

比如，单卡训练 InceptionResNet 网络最大Batch Size为100， 学习率为0.001。采用4张卡去训练时，可以设置Batch Size为400， 学习率为 0.002。在代码中对每一个Batch 400 切分成 4*100，然后给到不同GPU卡上的模型上去训练。

下面主要介绍一下单机多卡训练的细节及部分Tensorflow代码：

- 数据切片
- 模型构建
- 梯度反传

1. 数据分片

对于多GPU训练，我们首先要做的就是对训练数据进行分片，对于单机多卡模型，其数据的分片可以在代码的内部进行， 而对于多机多卡模型，多机之间没有必要进行数据切分方面的通信，建议的做法是先在本地做好数据的分配，然后再由不同的机器读取不同的数据。

Tensorflow 单机多卡的数据切片代码如下，其中 `training_iterator` 为采用 `tf.data.Dataset` 构建的训练数据pipeline， `num_tower` 为当前机器上可见的（CUDA_VISIBLE_DEVICES）GPU数量。

`tower` 指模型的副本。

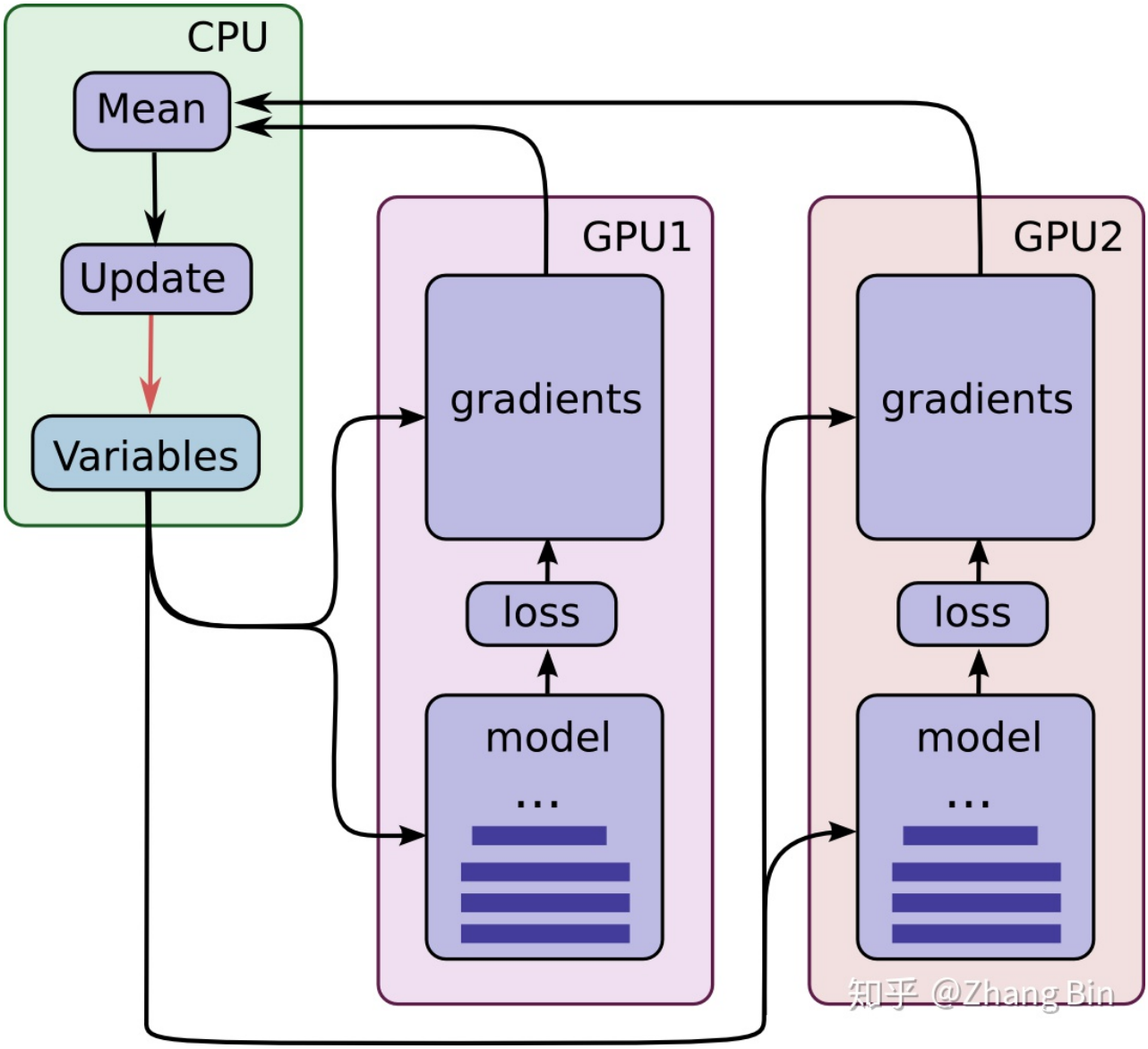
```
mnist_input, mnist_label = training_iterator.get_next()
tower_inputs = tf.split(mnist_input, num_towers)
tower_labels = tf.split(mnist_label, num_towers)
```

通过 `tf.split` 函数，将输入的数据按照GPU的数量平分。

2. 构建模型

有了分好的数据，下一步骤则是需要将模型放置在我们所有的GPU上，并将切片好的数据传入。

Tensorflow可以指定将不同的任务分配到不同的设备上， GPU支持大量并行计算，因此可以将模型的运算、梯度的计算分配到GPU上来，而变量的存储、梯度的更新则可以由CPU来执行，如下图所示



下面我们来看一下TensorFlow的相关代码

最外层的 `for i in range(num_towers):` 使我们需要构建模型 `num_towers` 次。

下一层的 `with tf.device(device_string % i):` 代表着每次我们构建模型时，模型放置的设备名称， 如果为GPU机器 `device_string = '/gpu:%d'` ，如果为CPU机器 `device_string = '/cpu:%d'`

再下一层 `with (tf.variable_scope(("tower"), reuse=True if i > 0 else None)):` 通过 scope 的reuse 使得我们再后续放置模型的时候，不同GPU之间的模型参数共享。

最后一层 `with (slim.arg_scope([slim.model_variable, slim.variable], device="/cpu:0" if num_gpus != 1 else "/gpu:0")):` 使得我们将构建模型中的变量放到CPU上来， 而运算则仍保留在该GPU上。

```
tower_gradients = []
tower_predictions = []
tower_label_losses = []
optimizer = tf.train.AdamOptimizer(learning_rate=1E-3)

for i in range(num_towers):
    with tf.device(device_string % i):
        with (tf.variable_scope(("tower"), reuse=True if i > 0 else None)):
            with (slim.arg_scope([slim.model_variable, slim.variable],
                                 device="/cpu:0" if num_gpus != 1 else "/gpu:0")):

                x = tf.placeholder_with_default(tower_inputs[i], [None, 224,224,3], name="input")
                y_ = tf.placeholder_with_default(tower_labels[i], [None, 1001], name="label")
                # logits = tf.layers.dense(x, 10)
                logits = build_model(x)
                predictions = tf.nn.softmax(logits, name="predictions")
                loss = tf.losses.softmax_cross_entropy(onehot_labels=y_, logits=logits)
                grads = optimizer.compute_gradients(loss)

                tower_gradients.append(grads)
                tower_predictions.append(predictions)
                tower_label_losses.append(loss)
```

3. 梯度反传

上一步已经得到了各个tower上的梯度，下面则需要将这些梯度结合起来并进行反向传播以更新模型的参数。

梯度结合的代码可以参考下面的函数

```
def combine_gradients(tower_grads):
    """Calculate the combined gradient for each shared variable across all towers.

    Note that this function provides a synchronization point across all towers.

    Args:
    tower_grads: List of lists of (gradient, variable) tuples. The outer list
      is over individual gradients. The inner list is over the gradient
      calculation for each tower.
    Returns:
     List of pairs of (gradient, variable) where the gradient has been summed
     across all towers.
    """
    filtered_grads = [[x for x in grad_list if x[0] is not None] for grad_list in tower_grads]
    final_grads = []
    for i in range(len(filtered_grads[0])):
        grads = [filtered_grads[t][i] for t in range(len(filtered_grads))]
        grad = tf.stack([x[0] for x in grads], 0)
        grad = tf.reduce_sum(grad, 0)
        final_grads.append((grad, filtered_grads[0][i][1],))

    return final_grads
```

通过 `combine_gradients` **来计算合并梯度，再通过** `optimizer.apply_gradients` **对得到的梯度进行反向传播**

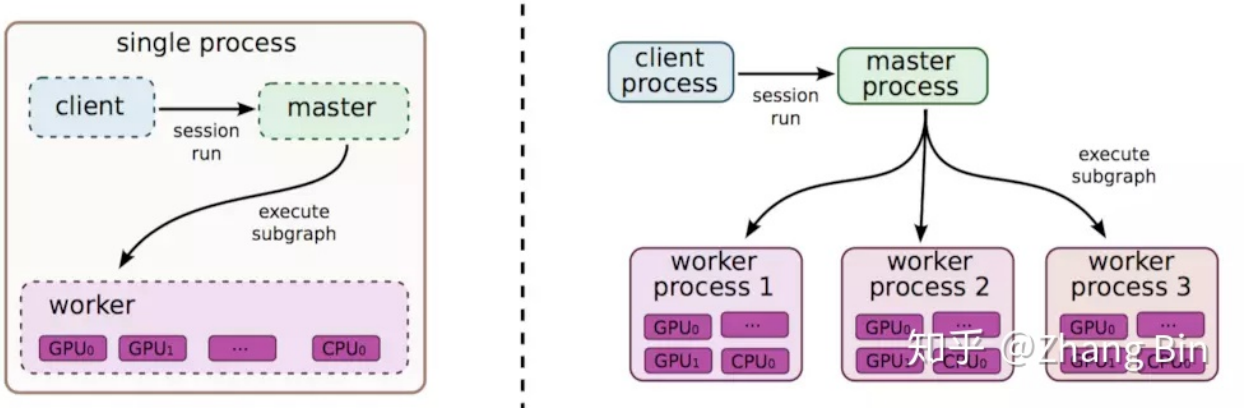
```
merged_gradients = combine_gradients(tower_gradients)
train_op = optimizer.apply_gradients(merged_gradients, global_step=global_step)
```

最后再通过 `sess.run(train_op)` **对执行。**

多机多卡 分布式训练

多机多卡相比较于单机多卡，其使得模型训练的上限进一步突破。一般我们一台服务器只支持8张GPU卡，而采用分布式的多机多卡训练方式，可以将几十甚至几百台服务器调度起来一起训练一个模型。

但相比于单机多卡，多机多卡分布式训练方式的配置更复杂一些，不仅要保证多台机器之间是可以互相通信的，还需要配置不同机器之间的角色以及不同机器之间梯度传递。



- TensorFlow 原生 PS架构
  在Parameter server架构（PS架构）中，集群中的节点被分为两类：parameter server和worker。其中parameter server存放模型的参数，而worker负责计算参数的梯度。在每个迭代过程，worker从parameter sever中获得参数，然后将计算的梯度返回给parameter server，parameter server聚合从worker传回的梯度，然后更新参数，并将新的参数广播给worker。
- Uber 开发的Horovod架构 - 支持 TensorFlow, Keras, PyTorch, and Apache MXNet
  Horovod 是一套面向 TensorFlow 的分布式训练框架，由 Uber 构建并开源，目前已经运行于 Uber 的 Michelangelo 机器学习即服务平台上。Horovod 能够简化并加速分布式深度学习项目的启动与运行。Horovod架构采用全新的梯度同步和权值同步算法，叫做 ring-allreduce。此种算法各个节点之间只与相邻的两个节点通信，并不需要参数服务器。因此，所有节点都参与计算也参与存储。

关于分布式训练的更多介绍，请参考我的下一篇文章

Zhang Bin：深度学习分布式训练相关介绍 - Part 2 详解分布式训练架构PS-Worker与Horovod
zhuanlan.zhihu.com

相关参考链接

TensorFlow - Multi GPU Computation

分布式tensorflow（一）

TensorFlow分布式全套（原理，部署，实例）

distributeTensorflowExample