Microsoft

人工智能系统 System for AI

深度学习中的分布式训练——系统

Distributed training systems

课程概要

分布式训练系统简介

主流深度学习框架中的分布式训练

- TensorFlow
- PyTorch

并行训练库

- Horovod

分布式训练系统

分布式训练系统

定义：能够分布式地执行深度学习的训练的系统

通常分为以下三个组成部分

- 分布式用户接口
  - 用户通过接口，实现模型的分布化
- 执行单节点训练
  - 产生本地执行的逻辑
- 通信协调
  - 实现多节点之间的通信协调

意义：提供易于使用，高效率的分布式训练

分布式训练系统分类对比

按照实现方式的不同，分为：

- **框架内嵌**分布式训练系统
- **跨框架通用**分布式训练系统

define comm. implicitly

define comm. explicitly

focus on data-parallelism

	TensorFlow
分布式用户接口	strategy list: PS, AllReduce
执行单节点训练	TensorFlow
通信协调	gRPC, libRDMA, NCCL

只支持单框架

分布式训练系统之

TensorFlow

TensorFlow

TensorFlow通过不同的API支持多种分布式策略(distributed strategies)

Training API	MirroredStrategy	TPUStrategy	MultiWorker-MirroredStrategy	CentralStorage-Strategy	ParameterServer-Strategy	OneDeviceStrategy
Keras API	Supported	Experimental support	Experimental support	Experimental support	Supported planned post 2.0	Supported
Custom training loop	Experimental support	Experimental support	Support planned post 2.0	Support planned post 2.0	No support yet	Supported
Estimator API	Limited Support	Not supported	Limited Support	Limited Support	Limited Support	Limited Support

https://www.tensorflow.org/guide/distributed_training

TensorFlow

TensorFlow早在版本(v0.8)中就加入了基于**参数服务器**“Parameter Server”的分布式训练

思路：多worker独立进行本地计算，分布式共享参数

https://www.tensorflow.org/guide/distributed_training

TensorFlow

TensorFlow参数服务器用户接口

- 定义模型
  - 指定节点信息（PS,worker）
  - Worker 包含“原模型”逻辑
- 执行模型
  - 指定角色 job_name： ps/worker
  - 指定index: 自己是第几个ps/worker

```
tf.train.ClusterSpec({
    "worker": [
        "worker0.example.com:2222",
        "worker1.example.com:2222",
        "worker2.example.com:2222",
    ],
    "ps": [
        "ps0.example.com:2222",
        "ps1.example.com:2222",
    ]})

if job_name == "ps":
   server.join()
elif job_name == "worker":
   ...
```

https://www.tensorflow.org/guide/distributed_training

TensorFlow

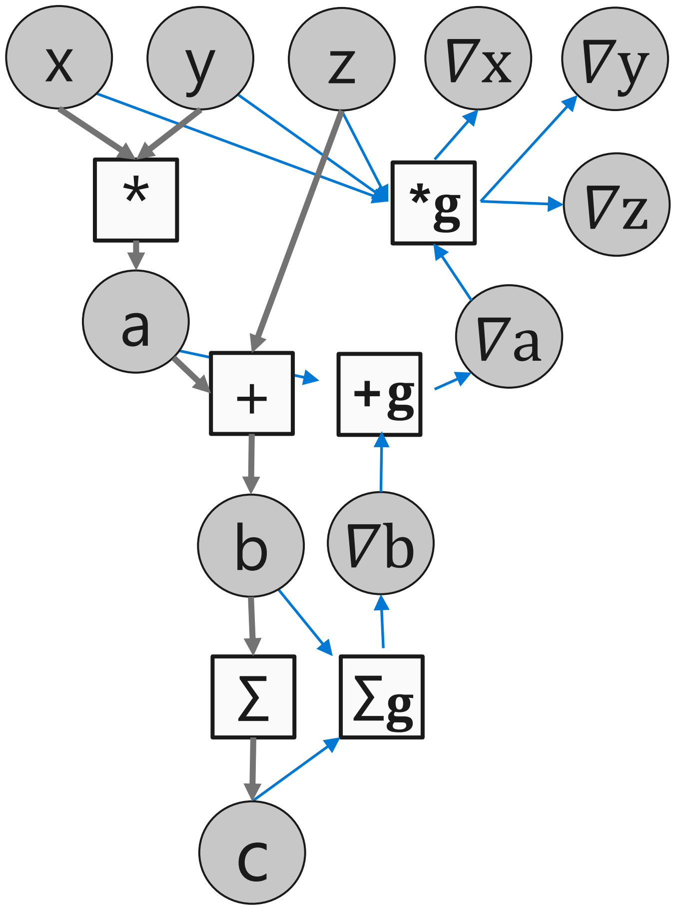
底层实现：数据流图的分布式切分

https://www.tensorflow.org/guide/distributed_training

TensorFlow

回顾：数据流图

数据流图作为中间表示

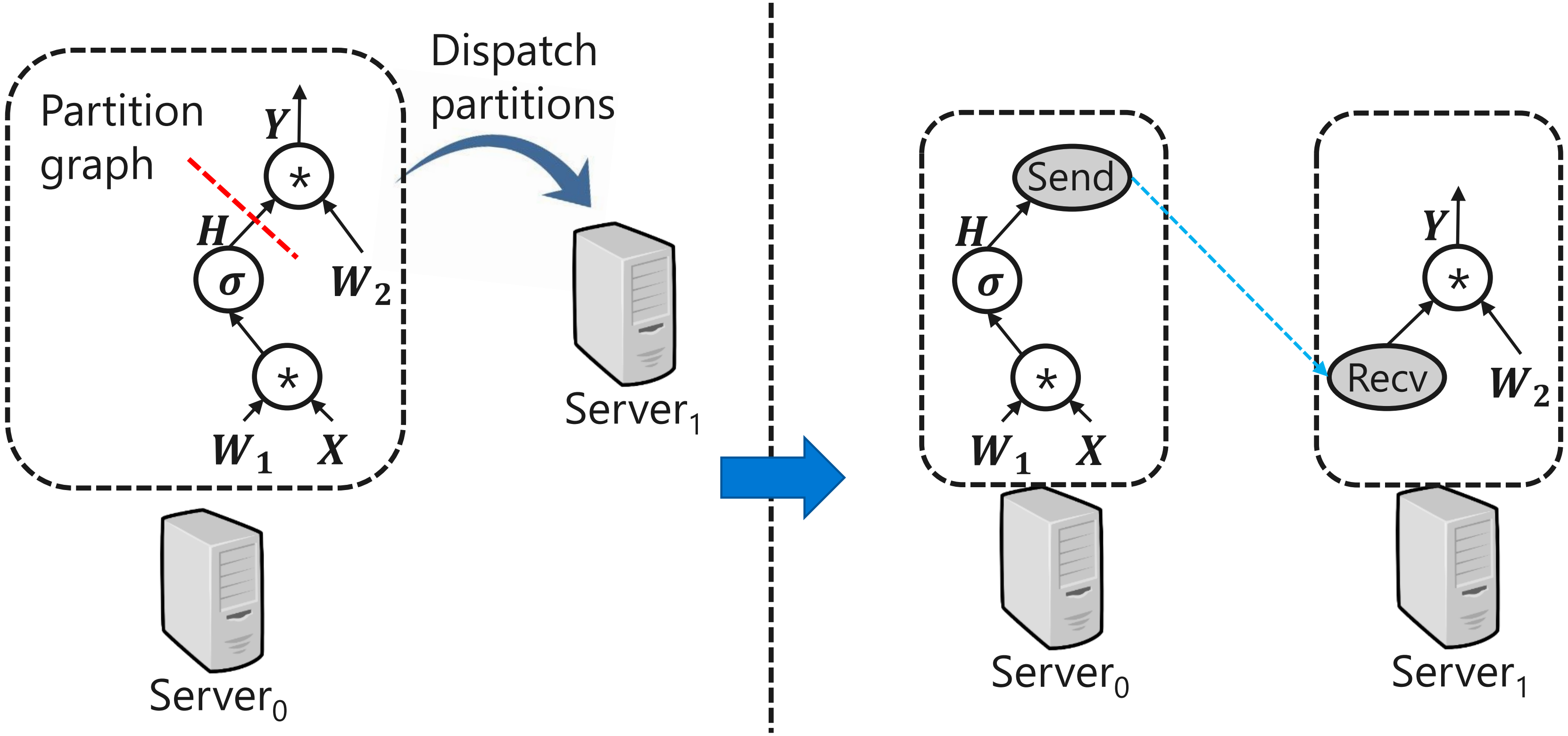


TensorFlow

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x,grad_y,grad_z = tf.gradients(c, [x,y,z])

with tf.Session() as sess:
  sess.run([grad_z], feed_dict=values)
```

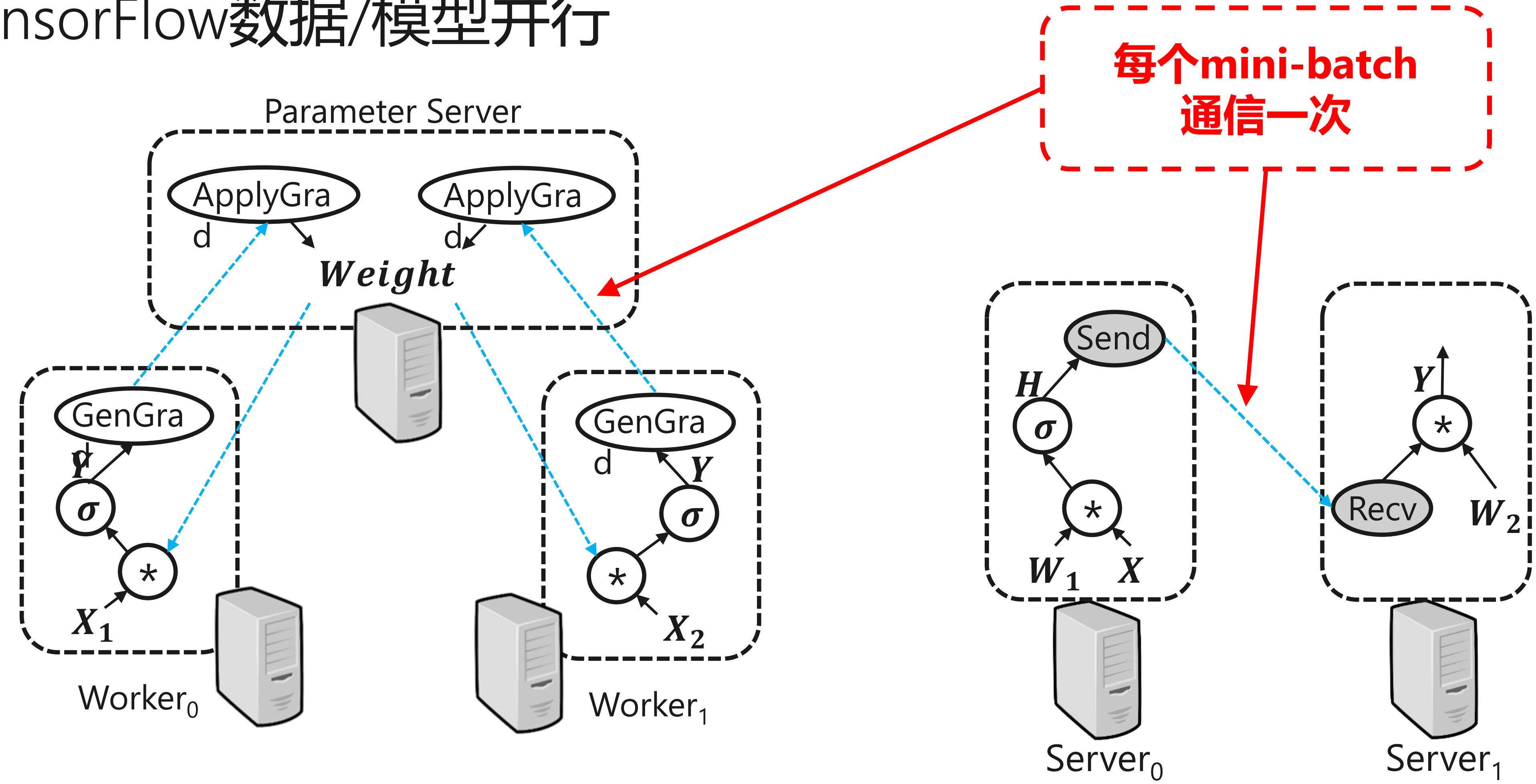
TensorFlow

图的跨节点切分



TensorFlow

TensorFlow数据/模型并行



数据并行

模型并行

TensorFlow

通信实现

- 点对点通信 Send/Recv
  - gRPC (TCP/IP)
  - gRPC (TCP for tensor metadata, RDMA for payload)
- 集中式通信 All-Reduce
  - gRPC (TCP/IP) //CollectiveCommunication.RING
  - nvidia NCCL (GPUDirect RDMA)

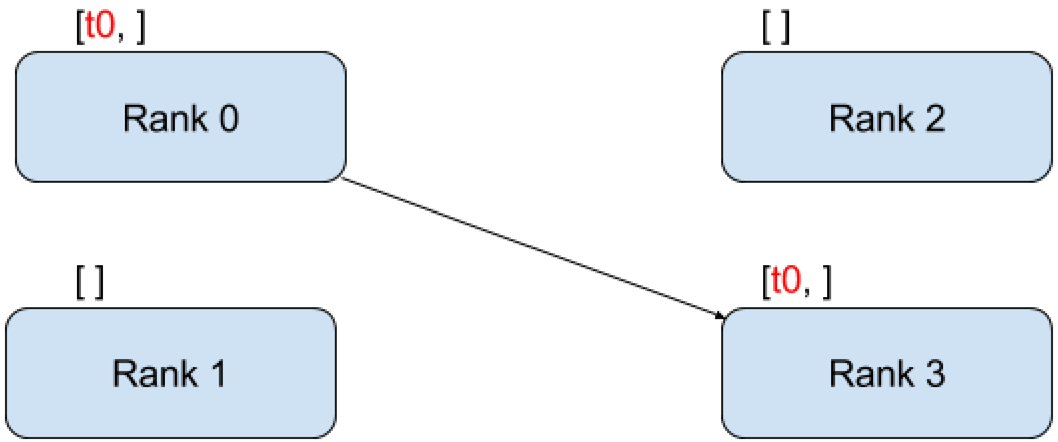
分布式训练系统之

PyTorch

PyTorch 同样包含 点对点通信 和 集中式通信

PyTorch

PyTorch 同样包含 点对点通信 和 集中式通信



PyTorch

PyTorch 点对点通信

可以实现用户指定的同步send/recv
例如：

rank 0 *send* -> rank 1 *recv*

```python
"""Blocking point-to-point communication."""

def run(rank, size):
    tensor = torch.zeros(1)
    if rank == 0:
        tensor += 1
        # Send the tensor to process 1
        dist.send(tensor=tensor, dst=1)
    else:
        # Receive tensor from process 0
        dist.recv(tensor=tensor, src=0)
    print('Rank ', rank, ' has data ', tensor[0])
```

https://pytorch.org/tutorials/intermediate/dist_tuto.html

PyTorch

PyTorch 点对点通信 （异步）

可以实现用户指定的异步send/recv

例如：

rank 0 *send* -> rank 1 *recv*

```python
"""Non-blocking point-to-point communication."""

def run(rank, size):
    tensor = torch.zeros(1)
    req = None
    if rank == 0:
        tensor += 1
        # Send the tensor to process 1
        req = dist.isend(tensor=tensor, dst=1)
        print('Rank 0 started sending')
    else:
        # Receive tensor from process 0
        req = dist.irecv(tensor=tensor, src=0)
        print('Rank 1 started receiving')
    req.wait()
    print('Rank ', rank, ' has data ', tensor[0])
```

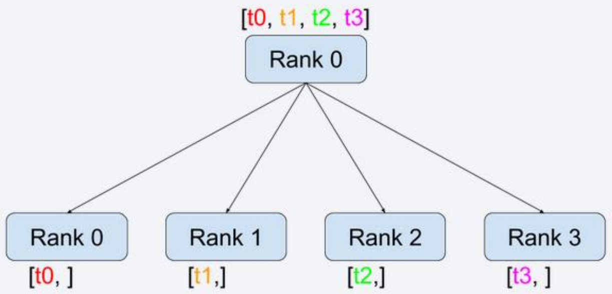
https://pytorch.org/tutorials/intermediate/dist_tuto.html

PyTorch

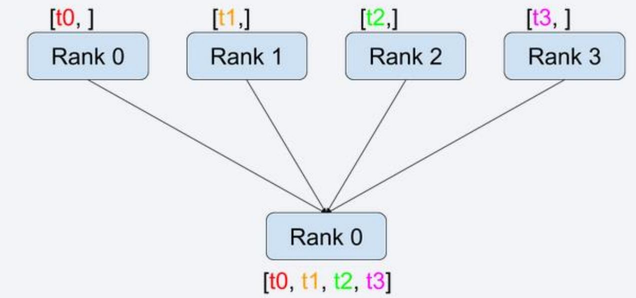
PyTorch 集中式通信

一对多: Scatter / Broadcast
多对一: Gather / Reduce
多对多: All-Reduce / AllGather

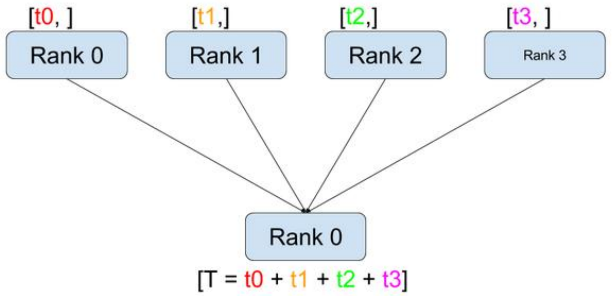
https://pytorch.org/tutorials/intermediate/dist_tuto.html



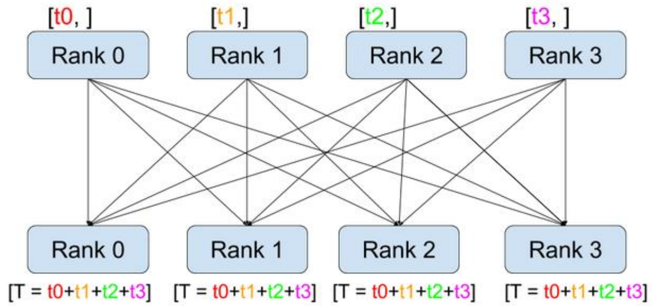
Scatter



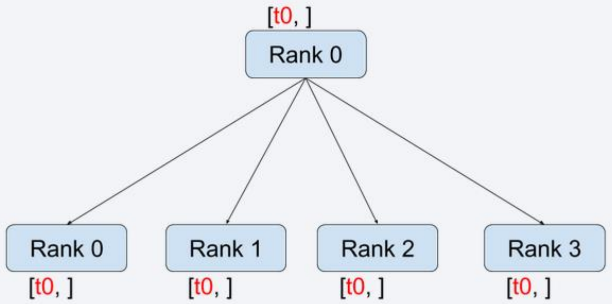
Gather



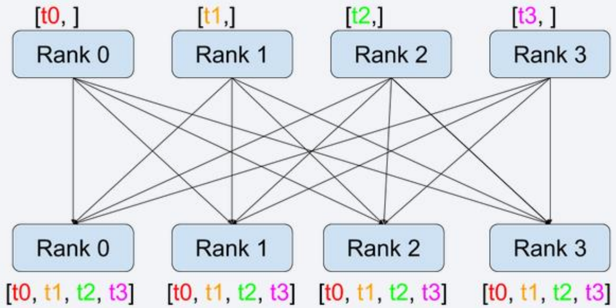
Reduce



All-Reduce



Broadcast



All-Gather

PyTorch

PyTorch 集中式通信

例： All-Reduce

```
""" All-Reduce example."""
def run(rank, size):
    """ Simple point-to-point communication. """
    group = dist.new_group([0, 1])
    tensor = torch.ones(1)
    dist.all_reduce(tensor, op=dist.reduce_op.SUM, group=group)
    print('Rank ', rank, ' has data ', tensor[0])
```

一对多: Scatter / Broadcast
多对一: Gather / Reduce
多对多: All-Reduce / AllGather

https://pytorch.org/tutorials/intermediate/dist_tuto.html

PyTorch

示例：分布式MNIST

```python
""" Distributed Synchronous SGD Example """
def run(rank, size):
    torch.manual_seed(1234)
    train_set, bsz = partition_dataset()
    model = Net()
    optimizer = optim.SGD(model.parameters(),
                          lr=0.01, momentum=0.5)

    num_batches = ceil(len(train_set.dataset) / float(bsz))
    for epoch in range(10):
        epoch_loss = 0.0
        for data, target in train_set:
            optimizer.zero_grad()
            output = model(data)
            loss = F.nll_loss(output, target)
            epoch_loss += loss.item()
            loss.backward()
            average_gradients(model)
            optimizer.step()
        print('Rank ', dist.get_rank(), ', epoch ',
              epoch, ': ', epoch_loss / num_batches)
```

```python
""" Gradient averaging. """
def average_gradients(model):
    size = float(dist.get_world_size())
    for param in model.parameters():
        dist.all_reduce(param.grad.data, op=dist.reduce_op.SUM)
        param.grad.data /= size
```

PyTorch

通信实现

- MPI
  - 通用接口，可调用 Open-MPI, MVAPICH2, Intel MPI, etc.
- NCCL
  - GPU通信优化，**仅支持集中式通信**
- Gloo
  - by Facebook

分布式训练系统
之

HOROVOD

Horovod

"Horovod is a distributed deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet. The goal of Horovod is to make distributed deep learning fast and easy to use."

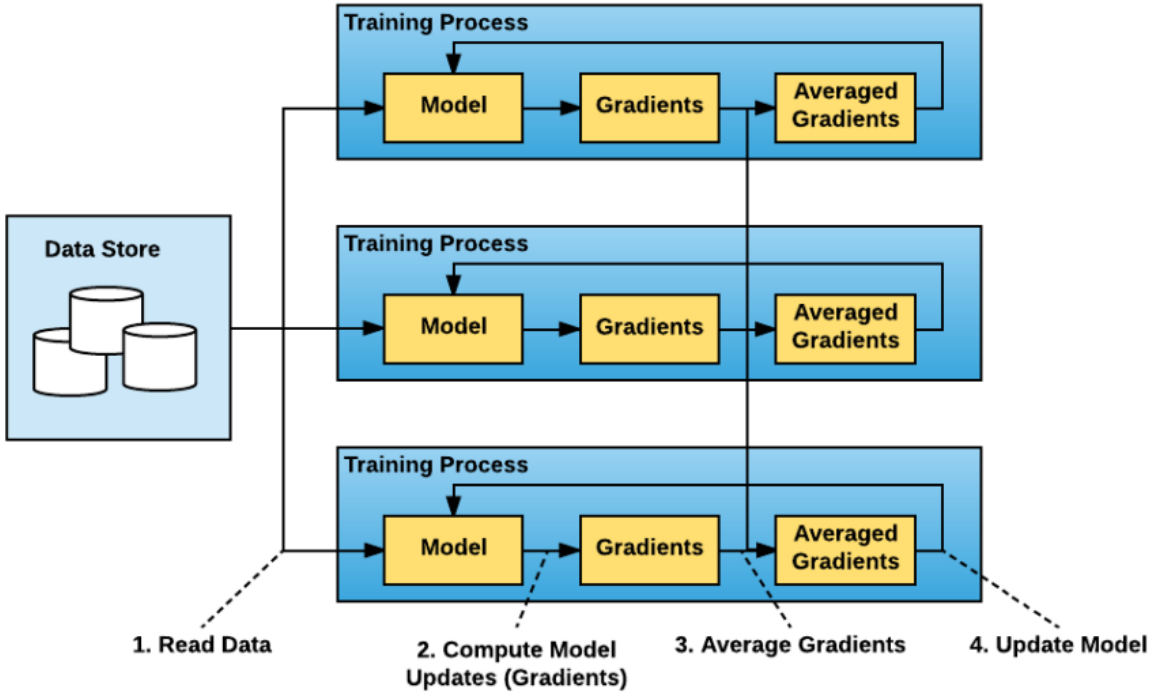
特点：

- 针对数据并行
- 广泛支持多训练平台
- 强调易用性

Horovod

~~“博尔不纯，杂而不精”~~

专注同步数据并行



Horovod

用户接口

- 模型代码修改

  opt = DistributedOptimizer(opt)

- 模型执行

  mpirun –n <worker number> train.py

```
# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01 * hvd.size())

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes during
# initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# Save checkpoints only on worker 0 to prevent other workers from corrupting them.
checkpoint_dir = '/tmp/train_logs' if hvd.rank() == 0 else None

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when done
# or an error occurs.
with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                       config=config,
                                       hooks=hooks) as mon_sess:
  while not mon_sess.should_stop():
    # Perform synchronous training.
    mon_sess.run(train_op)
```

Horovod 实现

通过DistributedOptimizer 插入gradient allreduce逻辑

TensorFlow：

插入通信 allreduce operator

PyTorch：

插入通信 allreduce 函数

底层调用统一的allreduce功能模块

Horovod 实现

协调机制算法

目标：确保allreduce的执行全局统一进行

- 每个worker拥有allreduce 执行队列，初始为空
- 全局master，维护每个worker各个gradients 状态

执行：

- worker_i产生梯度g_j后会调用allreduce(g_j)，通知master “g_j[i] ready”
- 当master收集到所有”g_j[*] ready”, 通知所有worker将g_i加入allreduce执行队列
- worker背景线程不断pop allreduce队列并执行

思考题

问：为什么需要确保allreduce全局统一执行?

- 确保相同执行顺序，保证针对同一个梯度进行操作
- allreduce通常是同步调用，提前执行的成员会空等，导致资源浪费

Horovod

通信实现

- MPI
  - 通用接口，可调用 Open-MPI, MVAPICH2, Intel MPI, etc.
- NCCL
  - GPU通信优化，**仅支持集中式通信**
- Gloo
  - by Facebook

其它

- 分布式训练下的数据读取
  - 本地读取：预先推送数据片到worker
  - 分布式文件系统：框架内嵌并行读取功能，多worker分片读取自己的部分

- 容错处理和故障恢复
  - 机器学习的内禀特性：训练数据丢弃不敏感（部分worker更新）
  - 模型checkpoint：save/load

思考题

问：为什么模型训练通常需要分布式进行，而分布式模型预测并不常见?

计算模式不同：

训练需要各个worker保持通信，从而协调统一地**更新**模型参数；

预测中的模型参数是**固定**的，各个worker分别使用只读副本，无需相互通信协调

本章小结

基于内嵌分布式策略的训练系统：针对TensorFlow为代表的基于数据流图的深度学习框架，根据内置规则，自动完成图切分和通信

基于提供通信原语分布式训练系统：针对解释执行深度学习框架（PyTorch），能支持更为灵活的分布式策略

支持多框架下数据并行训练的分布式系统：Horovod专注于简便易用的数据并行

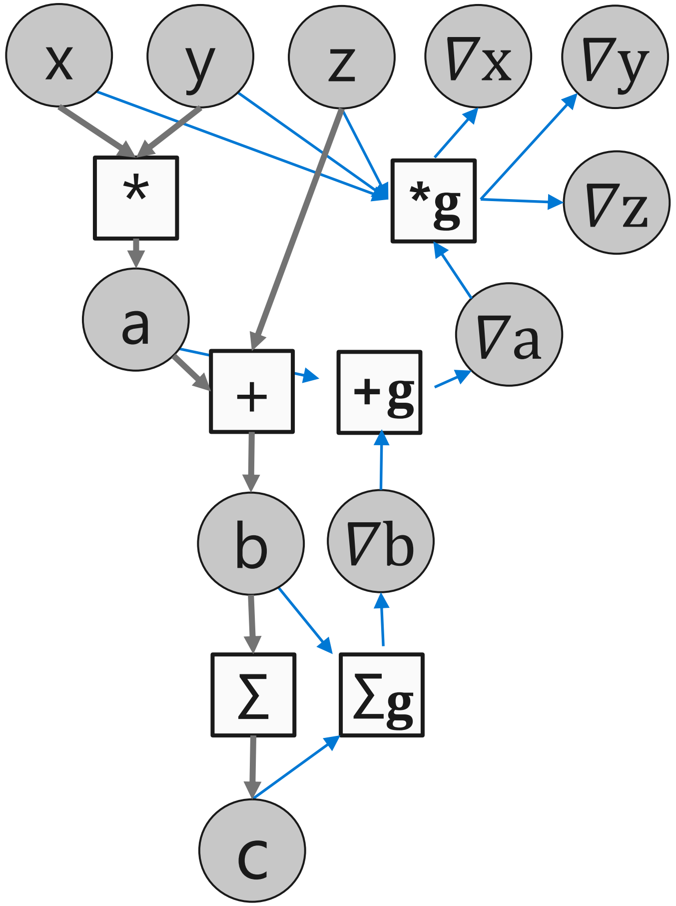
参考文献

- https://www.tensorflow.org/guide/distributed_training
- https://pytorch.org/tutorials/intermediate/dist_tuto.html
- Sergeev et.al., Horovod: fast and easy distributed deep learning in TensorFlow
- https://eng.uber.com/horovod/
- https://www.slideshare.net/AlexanderSergeev4/horovod-distributed-tensorflow-made-easy

TensorFlow

回顾：数据流图

数据流图作为中间表示



TensorFlow

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x,grad_y,grad_z = tf.gradients(c, [x,y,z])

with tf.Session() as sess:
  sess.run([grad_z], feed_dict=values)
```

本章小结

对于数据流图TensorFlow代表了一类基于内嵌分布式策略的系统

PyTorch提供了更为丰富的通信原语，能支持更为灵活的分布式策略

Horovod专注于简便易用的数据并行