PaddlePaddle 基本使用概念

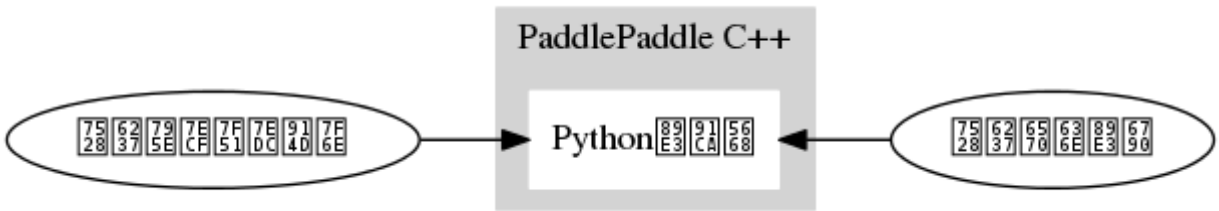
PaddlePaddle是一个神经网络学习框架。其单机进程为 `paddle train`。 单机的所有设备使用，均在单机进程内调度完成。 而多机辅助进程 `paddle pserver` 负责联合多个单机进程进行通信，进而充分利用集群的计算资源。 PaddlePaddle同时以 `swig api` 的形式，提供训练结果模型预测的方法和自定义训练流程。

下面我们会分别介绍主要进程 `paddle train` 中的一些概念。这些概念会对如何使用PaddlePaddle有一定的帮助。 了解这些概念的前提是，读者已经了解 基本的神经网络/机器学习原理和概念 。同时，如果想要了解PaddlePaddle实现中的一些概念，请参考 PaddlePaddle编程中的基本概念 。

Contents

PaddlePaddle 的进程模型

PaddlePaddle进程内嵌了一个 `python` 解释器。 这个 `python` 解释器负责解析用户定义的神经网络配置，和解析用户数据，并将用户数据传入给 PaddlePaddle。



所以，PaddlePaddle单机训练进程，`paddle train` ，对于用户的主要接口语言为 python。 主要需要用户配置的两个文件为 `DataProvider` 和训练文件 `TrainerConfig` 。

DataProvider

DataProvider是 `paddle train` 的数据提供器。 它负责将用户的原始数据转换成 PaddlePaddle可以识别的数据类型。每当 PaddlePaddle 需要新的数据训练时，都会调用 DataProvider 返回数据。 当所有数据读取完一轮后， DataProvider 便返回空数据通知 PaddlePaddle。 PaddlePaddle负责在下一轮训练开始前，将DataProvider重置。

需要注意的是，DataProvider在PaddlePaddle中是被训练逻辑调用的关系， 而不是新的数据驱动训练。并且所有的 `shuffle` ，和一些随机化的噪声添加，都应该在 DataProvider 阶段完成。

为了方便用户使用自己的数据格式， PaddlePaddle 提供了 PyDataProvider 来处理数据。 并且在这个Provider中，PaddlePaddle的 C++ 部分接管了如何shuffle，处理 batch，GPU/CPU通信，双缓冲，异步读取等问题。 用户可以参考 PyDataProvider 的相关文档，继续深入了解 DataProvider 的使用。

训练文件

训练文件是PaddlePaddle中配置神经网络结构、学习优化算法、数据传入方式的地方。 训练文件是一个python文件，使用命令行参数 `--config` 传给 paddle 的主程序。 例如:

```
paddle train --config=trainer_config.py
```

一个典型简单的训练文件可能为

```
from paddle.trainer_config_helpers import *

define_py_data_sources2(
    train_list='train.list',
    test_list='test.list',
    module='provider',
    obj='process')
settings(
    batch_size=128,
    learning_rate=1e-3,
    learning_method=AdamOptimizer(),
    regularization=L2Regularization(0.5))

img = data_layer(name='pixel', size=28 * 28)

hidden1 = simple_img_conv_pool(
    input=img, filter_size=3, num_filters=32, pool_size=3, num_channel=1)

hidden2 = fc_layer(
    input=hidden1,
    size=200,
    act=TanhActivation(),
    layer_attr=ExtraAttr(drop_rate=0.5))
predict = fc_layer(input=hidden2, size=10, act=SoftmaxActivation())

outputs(
    classification_cost(
        input=predict, label=data_layer(
            name='label', size=10)))
```

下面我们详细的介绍一下训练文件中各个模块的概念。

trainer_config_helpers

PaddlePaddle的配置文件与PaddlePaddle C++端通信的最基础协议是 `protobuf` 。而为了避免用户直接写比较难写的 protobuf string，我们书写了一个helpers来生成这个protobuf包。所以在文件的开始，import这些helpers函数。

需要注意的是，这个 `paddle.trainer_config_helpers` 包是标准的python包，这意味着用户可以选择自己喜欢的 `ide` 或者编辑器来编写Paddle的配置文件，这个python包注释文档比较完善，并且考虑了IDE的代码提示与类型注释。

data_sources

data_sources是配置神经网络的数据源。这里使用的函数是 `define_py_data_sources2` ，这个函数是定义了使用 PyDataProvider 作为数据源。 而后缀 `2` 是Paddle历史遗留问题，因为Paddle之前使用的 PyDataProvider 性能较差，所以完全重构了一个新的 PyDataProvider 。

data_sources里面的 train_list 和 test_list 指定的是训练文件列表和测试文件列表。 如果传入一个字符串的话，是指一个训练列表文件。这个训练列表文件中包含的是每一个训练或者测试文件的路径。如果传入一个list的话，则会默认生成一个 list 文件，再传入给 train.list 或者 test.list 。

而 `module` 和 `obj` 指定了 DataProvider 的模块名和函数名。

更具体的使用，请参考 PyDataProvider 。

settings

settings 是神经网络训练算法相关的设置项。包括学习率，batch_size，优化算法，正则方法等等。具体的使用方法请参考 settings 文档。

网络配置

上述网络配置中余下的部分均是神经网络配置。第一行是定义一个名字叫“pixel”的 `data_layer` 。每一个layer返回的都是一个 `LayerOutput` 对象。 这里第一层的输出对象是 `img` 。然后这个对象传输给了另一个 layer 函数， `simple_img_conv_pool` 。`simple_img_conv_pool` 是一个组合层，包括了图像的卷积 (convolution) 和池化(pooling)， 并继续接了一个全连接层(`fc_layer`)，然后再接了一个Softmax的全连接层。

最终，网络配置输出了 `classification_cost` 。标记网络输出的函数为 `outputs` 。网络的输出是神经网络的优化目标，神经网络训练的时候，实际上就是 要最小化这个输出。

在神经网络进行预测的时候，实际上网络的输出也是通过 `outputs` 标记。

Layer、Projection、Operator

PaddlePaddle的网络基本上是基于Layer来配置的。所谓的Layer即是神经网络的某一层， 而神经网络的某一层，一般是封装了许多复杂操作的操作集合。比如最简单的 `fc_layer` ，也包括矩阵乘法，多输入的求和，和activation。

```
data = data_layer(name='data', size=200)
out = fc_layer(input=data, size=200, act=TanhActivation())
```

而对于更灵活配置需求，可能这样基于Layer的配置是不灵活的。于是 PaddlePaddle 提供 了基于 Projection 或者 Operator 的配置。使用Projection和Operator需要与 `mixed_layer` 配合使用。 `mixed_layer` 是将layer中的元素累加求和， 并且做一个 `activation` ， 而这个layer具体如何

计算，是交由内部的Projection 和 Operator 定义。Projection是指含有可学习参数的操作，而Operator不含有可学习的 参数，输入全是其他Layer的输出。

例如，和 `fc_layer` 同样功能的 `mixed_layer` 。

```
data = data_layer(name='data', size=200)
with mixed_layer(size=200) as out:
        out += full_matrix_projection(input=data)
```

PaddlePaddle可以使用的mixed layer 配置出非常复杂的网络，甚至可以直接配置一个完整的LSTM。 用户可以参考 mixed_layer 的相关文档进行配置。

如何利用单机的所有GPU或所有CPU核心

PaddlePaddle的单机进程 `paddle train` 可以充分利用一台计算机上所有的GPU资 源或者CPU。

如果要使用机器上多块GPU，使用如下命令即可:

```
paddle train --use_gpu=true --trainer_count=4  # use 4 gpu card, 0, 1, 2, 3
```

如果要使用机器上多块CPU, 使用如下命令即可:

```
paddle train --trainer_config=4  # use 4 cpu cores.
```

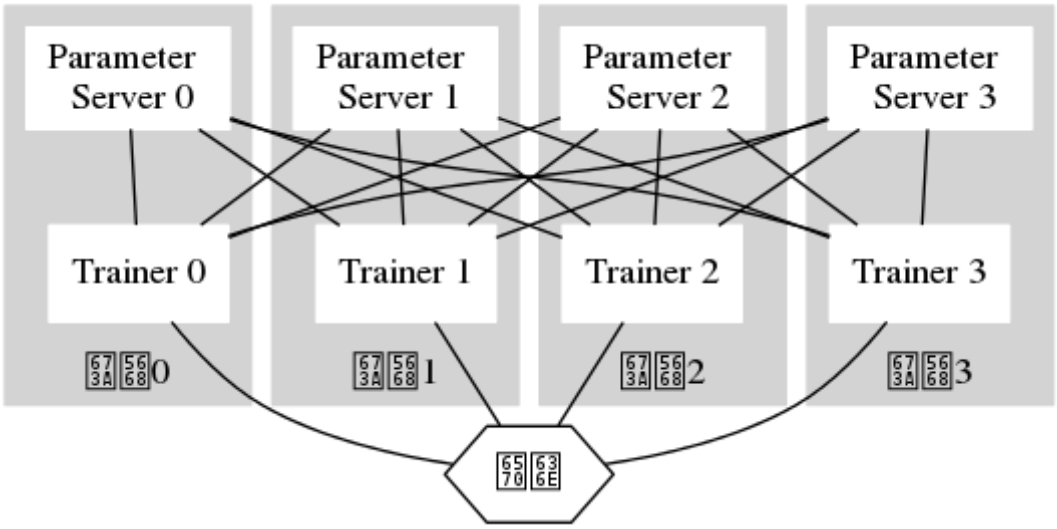
对于其他设置GPU的选择情况，例如选择第0、2号GPU显卡，则可以使用 `CUDA_VISIBLE_DEVICES` 环境变量来选择部分的显卡。 具体可以参考连接`masking-gpus`_ 。 可以使用的命令为

```
env CUDA_VISIBLE_DEVICES=0,2 paddle train --use_gpu=true --trainer_config=2
```

如何利用多台机器的计算资源训练神经网络

PaddlePaddle多机使用的经典方法是通过 `Parameter Server` 来对多机的 `paddle train` 进行同步。 而多机训练神经网络，首先要讲数据切分到不同的机器上。 切分数据文件的方式在PaddlePaddle的开源实现中并没有提供工具包。 但是切分数据并不是一件非常复杂的事情，也不是神经网络实现的重点。

多机训练过程中，经典的拓扑结构如下:



图中每个灰色方块是一台机器，在每个机器中，先去启动一个 `paddle pserver` 进程，并确定整体的端口号。可能的参数是:

```
paddle pserver --port=5000 --num_gradient_servers=4 --nics='eth0'
```

这里说明系统的 `paddle pserver` 的起始端口是 `5000` ，并且有四个训练进程(`gradient_servers`，Paddle同时将 `paddle train` 进程称作 `GradientServer` 。因为其为负责提供Gradient的进程)。 而对于训练进程的话，则需要在 `paddle pserver` 启动之后，再在各个节点上运行如下命令:

```
paddle train --port=5000 --pservers=192.168.100.101,192.168.100.102,192.168.100.103,192
```

对于简单的多机协同使用上述方式即可。同时，pserver/train 通常在高级情况下，还有两个参数需要设置，他们是

- —ports_num: 一个 pserver进程共绑定多少个端口用来做稠密更新。默认是1
- —ports_num_for_sparse: 一个pserver进程共绑定多少端口用来做稀疏更新，默认是0

使用手工指定端口数量，是因为Paddle的网络通信中，使用了 `int32` 作为消息长度，比较容易在大模型下溢出。所以，在 `paddle pserver` 进程中可以启动多个子线程去接受 trainer 的数据，这样单个子线程的长度就不会溢出了。但是这个值不可以调的过大，因为增加这个值，还是对性能，尤其是内存占用有一定的开销的，另外稀疏更新的端口如果太大的话，很容易某一个参数服务器没有分配到任何参数。

详细的说明可以参考，使用 集群训练Paddle 。