在 GPU 上使用 PADDLE QUANTUM

1. 简介

> 注意，本篇教程具有时效性。同时不同电脑也会有个体差异性，本篇教程不保证所有电脑可以安装成功。

在深度学习中，大家通常会使用 GPU 来进行神经网络模型的训练，因为与 CPU 相比，GPU在浮点数运算方面有着显著的优势。因此，使用 GPU 来训练神经网络模型逐渐成为共同的选择。在 Paddle Quantum 中，我们的量子态和量子门也采用基于浮点数的复数表示，因此我们的模型如果能部署到 GPU 上进行训练，也会显著提升训练速度。

2. GPU 选择

在这里，我们选择 Nvidia 的硬件设备，其 CUDA(Compute Unified Device Architecture) 对深度学习的框架支持比较好。我们的 PaddlePaddle 也可以比较方便地安装在 CUDA 上。

3. 配置 CUDA 环境

3.1 安装 CUDA

这里，我们介绍如何在 x64 平台上的 Windows10 系统中配置 CUDA 环境。首先，在CUDA GPUs | NVIDIA Developer上查看你的GPU是否可以安装CUDA环境。然后，在NVIDIA 驱动程序下载下载你的显卡的最新版驱动，并安装到电脑上。

在飞桨的安装步骤中，我们发现，**Paddle Paddle 在 Windows 下仅支持 CUDA 9.0/10.0 的单卡模式；不支持 CUDA 9.1/9.2/10.1**，所以，我们需要安装 CUDA10.0（CUDA9.0在理论上也可以）。在CUDA Toolkit Archive | NVIDIA Developer找到 CUDA 10.0 的下载地址：CUDA Toolkit 10.0 Archive | NVIDIA Developer，下载CUDA后，运行安装。

在安装过程中，选择**自定义安装**，在 CUDA 选项中，勾选除 Visual Studio Intergration 外的其他内容（除非你理解 Visual Studio Intergration 的作用），然后除 CUDA 之外，其他选项均不勾选。然后安装位置选择默认位置（请留意你的 CUDA 的安装位置，后面需要设置环境变量），等待安装完成。

安装完成之后，打开 Windows 命令行，输入 `nvcc -V`，如果看到版本信息，则说明 CUDA 安装成功。

3.2 安装 cuDNN

在NVIDIA cuDNN | NVIDIA Developer下载 cuDNN，根据飞桨的安装步骤中的要求，我们**需要使用 cuDNN 7.6+**，因此我们下载支持 CUDA 10.0 的最新版 cuDNN 即可。下载完成 cuDNN 后进行解压缩。然后，假设我们的 CUDA 的安装路径为 `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0`，我们将 cuDNN 解压缩后里面的 `bin`、`include` 和 `lib` 中的文件都替换 CUDA 的安装路径下的对应文件（如果文件已存在则进行替换，如果未存在则直接粘贴到对应目录中）。到这里，cuDNN 也就安装完成了。

3.3 配置环境变量

接下来还需要配置环境变量。右键电脑桌面上的“此电脑”（或“文件资源管理器”左栏的“此电脑”），选择“属性”，然后选择左侧的“高级系统设置”，在“高级”这一栏下选择“环境变量”。

现在就进入到了环境变量的设置页面，在系统变量中选择 `Path`，点击“编辑”。在出现的页面中，查看是否有 `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin` 和 `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\libnvvp` 这两个地址（其前缀 `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0` 应该为你的 CUDA 的安装位置），如果没有，请手动添加。

3.4 验证是否安装成功

打开命令行，输入 `cd C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\extras\demo_suite` 进入到 CUDA 安装路径（这里也应该为你的 CUDA 的安装位置）。然后分别执行 `.\bandwidthTest.exe` 和 `.\deviceQuery.exe`，如果都出现 `Result = PASS`，则说明安装成功。

4. 在 CUDA 环境上安装 PADDLEPADDLE

根据飞桨的安装步骤中的说明，我们首先需要确定自己的python环境，用 `python --version` 来查看 python 版本，保证**python版本是3.5.1+/3.6+/3.7+**，并且用 `python -m ensurepip` 和 `python -m pip --version` 来查看 pip 版本，**确认是 9.0.1+**。然后，使用 `python -m pip install paddlepaddle-gpu==1.8.4.post107 -i https://pypi.tuna.tsinghua.edu.cn/simple` 来安装 GPU 版本的 PaddlePaddle。

5. 安装 PADDLE QUANTUM

下载 Paddle Quantum 的安装包，修改 `setup.py`，将其中的 `paddlepaddle` 改为 `paddlepaddle-gpu`，然后按照 Paddle Quantum 的安装要求，执行 `pip install -e .` 即可。

> 如果你是在一个新的 python 环境中安装了 paddlepaddle-gpu 和 paddle_quantum，请在新 python 环境中安装 jupyter，并在新的 jupyter 下重新打开本教程并运行。

6. 检测是否安装成功

打开我们 GPU 版本的 PaddlePaddle 环境，执行下面的命令，若输出为 `True` 则表示当前 PaddlePaddle 框架可以在GPU上运行。

```
import paddle
from paddle import fluid
print(fluid.is_compiled_with_cuda())
```

7. 使用教程和示例

在 Paddle Quantum 中，我们使用动态图机制来定义和训练我们的参数化量子线路。在这里，我们依然使用动态图机制，只需要定义动态图机制的运行设备即可。方式如下：

```
# 0 表示使用编号为0的GPU
place = fluid.CUDAPlace(0)
with fluid.dygraph.guard(place):
    # build and train your quantum circuit model
```

当我们想在 CPU 上运行时，也采用类似的方式，定义运行设备为CPU：

```
place = fluid.CPUPlace()
with fluid.dygraph.guard(place):
    # build and train your quantum circuit model
```

我们可以在命令行中输入 `nvidia-smi` 来查看 GPU 的使用情况，包括有哪些程序在哪些 GPU 上运行，以及其显存占用情况。

这里，我们以 VQE 为例来说明我们该如何使用 GPU。首先，导入相关的包并定义相关的变量和函数。

```python
import os
from numpy import concatenate
from numpy import pi as PI
from numpy import savez, zeros
from paddle import fluid
from paddle.complex import matmul, transpose
from paddle_quantum.circuit import UAnsatz

import matplotlib.pyplot as plt
import numpy
from paddle_quantum.VQE.chemistrysub import H2_generator
from time import time

Hamiltonian, N = H2_generator()


def U_theta(theta, Hamiltonian, N, D):
    """
    Quantum Neural Network
    """

    # 按照量子比特数量/网络宽度初始化量子神经网络
    cir = UAnsatz(N)

    # 内置的 {R_y + CNOT} 电路模板
    cir.real_entangled_layer(theta[:D], D)

    # 铺上最后一列 R_y 旋转门
    for i in range(N):
        cir.ry(theta=theta[D][i][0], which_qubit=i)

    # 量子神经网络作用在默认的初始态 |0000>上
    cir.run_state_vector()

    # 计算给定哈密顿量的期望值
    expectation_val = cir.expecval(Hamiltonian)

    return expectation_val


class StateNet(fluid.dygraph.Layer):
    """
    Construct the model net
    """

    def __init__(self, shape, param_attr=fluid.initializer.Uniform(
        low=0.0, high=2 * PI), dtype="float64"):
        super(StateNet, self).__init__()

        # 初始化 theta 参数列表，并用 [0, 2*pi] 的均匀分布来填充初始值
        self.theta = self.create_parameter(
          shape=shape, attr=param_attr, dtype=dtype, is_bias=False)
```

```
53
54      # 定义损失函数和前向传播机制
55      def forward(self, Hamiltonian, N, D):
56          # 计算损失函数/期望值
57          loss = U_theta(self.theta, Hamiltonian, N, D)
58
59          return loss
60
61  ITR = 80 # 设置训练的总迭代次数
62  LR = 0.2 # 设置学习速率
63  D = 2    # 设置量子神经网络中重复计算模块的深度 Depth
```

如果要使用GPU训练，则运行下面的程序：

```
1   # 0 表示使用编号为0的GPU
2   place_gpu = fluid.CUDAPlace(0)
3   with fluid.dygraph.guard(palce_gpu):
4
5           # 确定网络的参数维度
6           net = StateNet(shape=[D + 1, N, 1])
7
8           # 一般来说，我们利用Adam优化器来获得相对好的收敛
9           # 当然你可以改成SGD或者是RMS prop.
10          opt = fluid.optimizer.AdamOptimizer(
11            learning_rate=LR, parameter_list=net.parameters())
12
13          # 记录优化结果
14          summary_iter, summary_loss = [], []
15
16          # 优化循环
17          for itr in range(1, ITR + 1):
18
19              # 前向传播计算损失函数
20              loss = net(Hamiltonian, N, D)
21
22              # 在动态图机制下，反向传播极小化损失函数
23              loss.backward()
24              opt.minimize(loss)
25              net.clear_gradients()
26
27              # 更新优化结果
28              summary_loss.append(loss.numpy())
29              summary_iter.append(itr)
30
31              # 打印结果
32              if itr % 20 == 0:
33                  print("iter:", itr, "loss:", "%.4f" % loss.numpy())
34                  print("iter:", itr, "Ground state energy:",
35                        "%.4f Ha" % loss.numpy())
36
37
```

如果要使用CPU训练，则运行下面的程序：

```
# 表示使用 CPU
place_cpu = fluid.CPUPlace()
with fluid.dygraph.guard(place_cpu):

    # 确定网络的参数维度
    net = StateNet(shape=[D + 1, N, 1])

    # 一般来说，我们利用Adam优化器来获得相对好的收敛
    # 当然你可以改成SGD或者是RMS prop.
    opt = fluid.optimizer.AdamOptimizer(
      learning_rate=LR, parameter_list=net.parameters())

    # 记录优化结果
    summary_iter, summary_loss = [], []

    # 优化循环
    for itr in range(1, ITR + 1):

        # 前向传播计算损失函数
        loss = net(Hamiltonian, N, D)

        # 在动态图机制下，反向传播极小化损失函数
        loss.backward()
        opt.minimize(loss)
        net.clear_gradients()

        # 更新优化结果
        summary_loss.append(loss.numpy())
        summary_iter.append(itr)

        # 打印结果
        if itr % 20 == 0:
            print("iter:", itr, "loss:", "%.4f" % loss.numpy())
            print("iter:", itr, "Ground state energy:",
                  "%.4f Ha" % loss.numpy())
```

```
iter: 20 loss: -1.0669
iter: 20 Ground state energy: -1.0669 Ha
iter: 40 loss: -1.1129
iter: 40 Ground state energy: -1.1129 Ha
iter: 60 loss: -1.1163
iter: 60 Ground state energy: -1.1163 Ha
iter: 80 loss: -1.1172
iter: 80 Ground state energy: -1.1172 Ha
```

8. 总结

按照我们的测试，现在版本的 paddle_quantum 可以在 GPU 下运行，但是需要比较好的 GPU 资源才能体现出足够的加速效果。在未来的版本中，我们也会不断优化 paddle_quantum 在 GPU 下的性能表现，敬请期待。

参考资料

(1) Installation Guide Windows :: CUDA Toolkit Documentation

(2) Installation Guide :: NVIDIA Deep Learning cuDNN Documentation

(3) 开始使用_飞桨-源于产业实践的开源深度学习平台