变分量子态对角化算法（VQSD）

概览

- 在本案例中，我们将通过 Paddle Quantum 训练量子神经网络来完成量子态的对角化
- 首先，让我们通过下面几行代码引入必要的library和package。

```
import numpy
from numpy import diag
import scipy
from paddle import fluid
from paddle_quantum.circuit import UAnsatz
from paddle_quantum.utils import dagger
from paddle.complex import matmul, trace, transpose
```

背景

量子态对角化算法（VQSD，Variational Quantum State Diagonalization）[1-3] 目标是输出一个量子态的特征谱，即其所有特征值。求解量子态的特征值在量子计算中有着诸多应用，比如可以用于计算保真度和冯诺依曼熵，也可以用于主成分分析。

- 量子态通常是一个混合态，表示如下 $\rho_{\text{mixed}} = \sum_i P_i |\psi_i\rangle\langle\psi_i|$
- 作为一个简单的例子，我们考虑一个2量子位的量子态，它的特征谱为 $(0.5, 0.3, 0.1, 0.1)$，我们先通过随机作用一个酉矩阵来生成具有这样特征谱的随机量子态。

```
scipy.random.seed(13)                       # 固定随机种子，方便复现结果
V = scipy.stats.unitary_group.rvs(4)        # 随机生成一个酉矩阵
D = diag([0.5, 0.3, 0.1, 0.1])              # 输入目标态 rho 的谱
V_H = V.conj().T
rho = V @ D @ V_H                           # 通过逆向的谱分解生成 rho
print(numpy.around(rho, 4))                 # 打印量子态 rho
```

```
[[ 0.2569+0.j     -0.012 +0.0435j -0.0492-0.0055j -0.0548+0.0682j]
 [-0.012 -0.0435j  0.2959-0.j      0.1061-0.0713j -0.0392-0.0971j]
 [-0.0492+0.0055j  0.1061+0.0713j  0.2145-0.j      0.0294-0.1132j]
 [-0.0548-0.0682j -0.0392+0.0971j  0.0294+0.1132j  0.2327+0.j    ]]
```

搭建量子神经网络

- 在这个案例中，我们将通过训练量子神经网络QNN（也可以理解为参数化量子电路）来学习量子态的特征谱。这里，我们提供一个预设的2量子位量子电路。
- 我们预设一些该参数化电路的参数，比如宽度为2量子位。
- 初始化其中的变量参数，θ代表我们量子神经网络中的参数组成的向量。

```
N = 2              # 量子神经网络的宽度
SEED = 14          # 固定随机种子
THETA_SIZE = 15    # 量子神经网络中参数的数量

def U_theta(theta, N):
    """
    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)

    # 调用内置的量子神经网络模板
    cir.universal_2_qubit_gate(theta)

    # 返回量子神经网络所模拟的酉矩阵 U
    return cir.U
```

配置训练模型 - 损失函数

- 现在我们已经有了数据和量子神经网络的架构，我们将进一步定义训练参数、模型和损失函数。
- 通过作用量子神经网络$U(\theta)$在量子态ρ后得到的量子态记为$\tilde{\rho}$，我们设定损失函数为$\tilde{\rho}$与用来标记的量子态$\sigma = 0.1|00\rangle\langle 00| + 0.2|01\rangle\langle 01| + 0.3|10\rangle\langle 10| + 0.4|11\rangle\langle 11|$的内积。
- 具体的，设定损失函数为 $\mathcal{L}(\boldsymbol{\theta}) = \mathrm{Tr}(\tilde{\rho}\sigma)$.

```
# 输入用来标记的量子态sigma
sigma = diag([0.1, 0.2, 0.3, 0.4]).astype('complex128')

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

    def __init__(self, shape, rho, sigma, param_attr =
        fluid.initializer.Uniform(
        low=0.0, high=2 * numpy.pi, seed=SEED), dtype='float64'):
        super(Net, self).__init__()


```

```
15
16          # 将 Numpy array 转换成 Paddle 动态图模式中支持的 variable
17          self.rho = fluid.dygraph.to_variable(rho)
18          self.sigma = fluid.dygraph.to_variable(sigma)
19
20          # 初始化 theta 参数列表，并用 [0, 2*pi] 的均匀分布来填充初始值
21          self.theta = self.create_parameter(
22          shape=shape, attr=param_attr, dtype=dtype, is_bias=False)
23
24      # 定义损失函数和前向传播机制
25      def forward(self, N):
26
27          # 施加量子神经网络
28          U = U_theta(self.theta, N)
29
30          # rho_tilde 是将 U 作用在 rho 后得到的量子态 U*rho*U^dagger
31          rho_tilde = matmul(matmul(U, self.rho), dagger(U))
32
33          # 计算损失函数
34          loss = trace(matmul(self.sigma, rho_tilde))
35
36          return loss.real, rho_tilde
```

配置训练模型 - 模型参数

在进行量子神经网络的训练之前，我们还需要进行一些训练的超参数设置，主要是学习速率 (LR, learning rate)和迭代次数(ITR, iteration)。这里我们设定学习速率为0.1, 迭代次数为50次。读者不妨自行调整来直观感受下超参数调整对训练效果的影响。

```
1  ITR = 50  # 设置训练的总的迭代次数
2  LR = 0.1  # 设置学习速率
```

进行训练

- 当训练模型的各项参数都设置完成后，我们将数据转化为Paddle动态图中的变量，进而进行量子神经网络的训练。
- 过程中我们用的是Adam Optimizer，也可以调用Paddle中提供的其他优化器。
- 我们将训练过程中的结果依次输出。

```
1  # 初始化paddle动态图机制
2  with fluid.dygraph.guard():
3
4      # 确定网络的参数维度
5      net = Net(shape=[THETA_SIZE], rho=rho, sigma=sigma)
6
```

```
    # 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.AdagradOptimizer(
          learning_rate=LR, parameter_list=net.parameters())

    # 优化循环
    for itr in range(ITR):

        # 前向传播计算损失函数并返回估计的能谱
        loss, rho_tilde = net(N)
        rho_tilde_np = rho_tilde.numpy()

        # 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()

        # 打印训练结果
        if itr % 10 == 0:
            print('iter:', itr, 'loss:', '%.4f' % loss.numpy()[0])

```

```
iter: 0 loss: 0.2354
iter: 10 loss: 0.1912
iter: 20 loss: 0.1844
iter: 30 loss: 0.1823
iter: 40 loss: 0.1813
```

总结

根据上面训练得到的结果，通过大概50次迭代，我们就比较好的完成了对角化。我们可以通过打印 $\tilde{\rho} = U(\boldsymbol{\theta})\rho U^{\dagger}(\boldsymbol{\theta})$ 的来验证谱分解的效果。特别的，我们可以验证它的对角线与目标谱非常接近。

```
print("The estimated spectrum is:",
numpy.real(numpy.diag(rho_tilde_np)))
print("The target spectrum is:", numpy.diag(D))
```

```
The estimated spectrum is:
[0.49401064 0.30357179 0.10224927 0.10016829]
The target spectrum is:
[0.5 0.3 0.1 0.1]
```

参考文献

(1) Larose, R., Tikku, A., Neel-judy, É. O., Cincio, L. & Coles, P. J. Variational quantum state diagonalization. npj Quantum Inf. (2019) doi:10.1038/s41534-019-0167-6.

(2) Nakanishi, K. M., Mitarai, K. & Fujii, K. Subspace-search variational quantum eigensolver for excited states. Phys. Rev. Res. 1, 033062 (2019).

(3) Cerezo, M., Sharma, K., Arrasmith, A. & Coles, P. J. Variational Quantum State Eigensolver. arXiv:2004.01372 (2020).