子空间搜索- 量子变分特征求解器

(SUBSPACE-SEARCH VQE)

概览

- 在本案例中，我们将展示如何通过Paddle Quantum训练量子神经网络来求解量子系统的整个能量谱。
- 首先，让我们通过下面几行代码引入必要的library和package。

```
import numpy
from paddle.complex import matmul, transpose
from paddle import fluid
from paddle_quantum.circuit import UAnsatz
from paddle_quantum.utils import random_pauli_str_generator,
pauli_str_to_matrix, dagger
```

背景

- 量子计算在近期内备受瞩目的一个应用就是变分量子特征求解器(VQE, variational quantum eigensolver (VQE)) [1-3].
- VQE是量子化学在近期有噪量子设备（NISQ device）上的核心应用之一，其中一个功能比较强大的版本是SSVQE [4]，其核心是去求解一个物理系统的哈密顿量的基态和**激发态**的性质。数学上，可以理解为求解一个厄米矩阵(Hermitian matrix)的特征值及其对应的特征向量。该哈密顿量的特征值组成的集合我们称其为能谱 (Energy spectrum)。
- 接下来我们将通过一个简单的例子学习如何通过训练量子神经网络解决这个问题，即求解出给定哈密顿量 H 的能谱。

SSVQE分析物理系统的基态和激发态的能量

- 对于具体需要分析的分子，我们需要其几何构型 (geometry)、电荷 (charge) 以及自旋多重度 (spin multiplicity) 等多项信息来建模获取描述系统的哈密顿量。具体的，通过我们内置的量子化学工具包可以利用 fermionic-to-qubit 映射的技术来输出目标分子的量子比特哈密顿量表示。
- 作为简单的入门案例，我们在这里提供一个简单的随机2量子比特哈密顿量作为例子。

```
N = 2          # 量子比特数/量子神经网络的宽度
SEED = 14      # 固定随机种子
```

```
# 生成用泡利字符串表示的随机哈密顿量
hamiltonian = random_pauli_str_generator(N, terms=10)
print("Random Hamiltonian in Pauli string format = \n", hamiltonian)

# 生成哈密顿量的矩阵信息
H = pauli_str_to_matrix(hamiltonian, N)
```

```
Random Hamiltonian in Pauli string format =
 [[0.2882902599422752, 'x0,x1'], [-0.37132004698018894, 'z0,z1'],
[0.3394848692992223, 'x1'], [0.5983056387866057, 'y1'],
[0.5931450194526844, 'y1,y0'], [-0.7752917446753211, 'y1'],
[0.5394363496035564, 'z0,y1'], [-0.1389123713189584, 'z1,y0'],
[0.23481997155046552, 'x1'], [-0.8059387465325567, 'z1,y0']]
```

搭建量子神经网络（QNN）

- 在实现SSVQE的过程中，我们首先需要设计量子神经网络QNN（也即参数化量子电路）。在本教程中，我们提供一个预设的适用于2量子比特的通用量子电路模板。理论上，该模板具有足够强大的表达能力可以表示任意的2-量子比特逻辑运算 (5)。具体的实现方式是需要 3个 $CNOT$ 门加上任意15个单比特旋转门 $\in \{R_y, R_z\}$。
- 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，一共有15个参数。

```
THETA_SIZE = 15  # 量子神经网络中参数的数量

def U_theta(theta, N):
    """
    U_theta
    """
    # 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，具体的理论可以参考 (4).
- 通过作用量子神经网络 $U(\theta)$ 在1组正交的初始态上 (方便起见，可以取计算基 $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$)，我们将得到输出态 $\{|\psi_1(\theta)\rangle, |\psi_2(\theta)\rangle, |\psi_3(\theta)\rangle, |\psi_4(\theta)\rangle\}$。
- 进一步，在SSVQE模型中的损失函数一般由每个输出量子态 $|\psi_k(\theta)\rangle$ 关于哈密顿量 H 的能量期望值 (expectation value) 的加权求和给出。这里我们默认权重向量 $\vec{w} = [4, 3, 2, 1]$。
- 具体的损失函数 (loss function) 定义为：

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{k=1}^{2^n} w_k * \langle\psi_k(\theta)| H |\psi_k(\theta)\rangle$$

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

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

    # 定义损失函数和前向传播机制
    def forward(self, H, N):

        # 施加量子神经网络
        U = U_theta(self.theta, N)

        # 计算损失函数
        loss_struct = matmul(matmul(dagger(U), H), U).real

        # 输入计算基去计算每个子期望值，相当于取 U^dagger*H*U 的对角元
        loss_components = [
            loss_struct[0][0],
            loss_struct[1][1],
            loss_struct[2][2],
            loss_struct[3][3]
        ]

        # 最终加权求和后的损失函数
        loss = 4 * loss_components[0] + 3 * loss_components[1]
             + 2 * loss_components[2] + 1 * loss_components[3]

        return loss, loss_components
```

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)、迭代次数(ITR, iteration)和量子神经网络计算模块的深度 (Depth, D)。这里我们设定学习速率为0.3, 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。

```
ITR = 50 # 设置训练的总迭代次数
LR = 0.3 # 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。
- 我们可以将训练过程中的每一轮loss打印出来。

```
# 初始化paddle动态图机制
with fluid.dygraph.guard():

    # 我们需要将 Numpy array 转换成 Paddle 动态图模式中支持的 variable
    hamiltonian = fluid.dygraph.to_variable(H)

    # 确定网络的参数维度
    net = Net(shape=[THETA_SIZE])

    # 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(1, ITR + 1):

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

        # 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])
```

测试效果

我们现在已经完成了量子神经网络的训练，我们将通过与理论值的对比来测试效果。

- 理论值由numpy中的工具来求解哈密顿量的各个特征值；
- 我们将训练QNN得到的各个能级的能量和理想情况下的理论值进行比对。
- 可以看到，SSVQE训练输出的值与理想值高度接近。

```
print('The estimated ground state energy is: ', loss_components[0].numpy())
print('The theoretical ground state energy: ', numpy.linalg.eigh(H)[0][0])

print('The estimated 1st excited state energy is: ', loss_components[1].numpy())
print('The theoretical 1st excited state energy: ', numpy.linalg.eigh(H)[0][1])

print('The estimated 2nd excited state energy is: ', loss_components[2].numpy())
print('The theoretical 2nd excited state energy: ', numpy.linalg.eigh(H)[0][2])

print('The estimated 3rd excited state energy is: ', loss_components[3].numpy())
print('The theoretical 3rd excited state energy: ', numpy.linalg.eigh(H)[0][3])
```

```
The estimated ground state energy is:  [-1.90448246]
The theoretical ground state energy:  -1.9113115824996965
The estimated 1st excited state energy is:  [-0.90911539]
The theoretical 1st excited state energy:  -0.9051492736630313
The estimated 2nd excited state energy is:  [1.30471078]
The theoretical 2nd excited state energy:  1.2814396957014094
The estimated 3rd excited state energy is:  [1.50888706]
The theoretical 3rd excited state energy:  1.5350211604613184
```

参考文献

(1) Peruzzo, A. et al. A variational eigenvalue solver on a photonic quantum processor. Nat. Commun. 5, 4213 (2014).

(2) McArdle, S., Endo, S., Aspuru-Guzik, A., Benjamin, S. C. & Yuan, X. Quantum computational chemistry. Rev. Mod. Phys. 92, 015003 (2020).

(3) Cao, Y. et al. Quantum chemistry in the age of quantum computing. Chem. Rev. 119, 10856–10915 (2019).

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

(5) Vatan, F. & Williams, C. Optimal quantum circuits for general two-qubit gates. Phys. Rev. A 69, 032315 (2004).