量子计算
—编程篇

Quantum Computer

网址：www.qubits.top

作者：Calvin Tang

邮箱：179209347@qq.com

介绍

本开发教程基于本源量子的Qpanda框架的python版 – **PyQPanda** 编写。

- 一种功能齐全，运行高效的量子软件开发工具包
- QPanda 2是由本源量子开发的开源量子计算框架，它可以用于构建、运行和优化量子算法。
- QPanda 2作为本源量子计算系列软件的基础库，为OriginIR、Qurator、量子计算服务提供核心部件。

本教程包含：
1.变量
2.运算符
3.可变量子逻辑门
4.优化算法（梯度下降法）

QPanda使用文档：
https://pyqpanda-toturial.readthedocs.io/zh/latest/index.html

Github & Gitee 代码地址：
https://github.com/mymagicpower/quantum/tree/main/quantum_qpanda/vqc
https://gitee.com/mymagicpower/quantum/tree/main/quantum_qpanda/vqc

变量

变量类是实现符号计算的用户类，用于存储特定混合量子经典网络的变量。 通常任务是优化变量以最小化成本函数。 变量可以是标量，矢量或矩阵。

变量具有树形结构，它可以包含子节点或父节点。如果变量没有子节点，那么我们称之为叶子节点。我们可以将变量设置为特定值，另一方面，如果变量的所有叶子节点都已设置了值，我们也可以获得该变量的值。

我们可以通过传入一个浮点型的数据来构造一个标量变量，也可以通过传入numpy库生成的多维数组来构造一个矢量或矩阵变量。

```
v1 = var(1)
a = np.array([[1.],[2.],[3.],[4.]])
v2 = var(a) b = np.array([[1.,2.],[3.,4.]])
v3 = var(b)
```

0_Var.py

```
from pyqpanda import *
import numpy as np
if __name__=="__main__":
    m1 = np.array([[1., 2.],[3., 4.]])
    v1 = var(m1)
    m2 = np.array([[5., 6.],[7., 8.]])
    v2 = var(m2)
    sum = v1 + v2
    minus = v1 - v2
    multiply = v1 * v2
    print("v1: ", v1.get_value())
    print("v2: ", v2.get_value())
    print("sum: " , eval(sum))
    print("minus: " , eval(minus))
    print("multiply: " , eval(multiply))
    m3 = np.array([[4., 3.],[2., 1.]])
    v1.set_value(m3)

    print("sum: " , eval(sum))
    print("minus: " , eval(minus))
    print("multiply: " , eval(multiply))
```

运算符

VQNet包含许多类型的运算符， 这些运算符可以操作变量或占位符。 其中，一些运算符是经典运算符，例如加，减，乘，除，指数，对数和点乘等。 VQNet还包含来自经典机器学习中所有常见操作符。
VQNet还包含有量子运算符，它们是qop和qop pmeasure。 另外，qop和qop pmeasure与量子计算机芯片有关，使用它们需要量子环境。 为此，我们需要在使用这两个运算符时需要添加额外的参数。 一般来说，我们需要添加量子机器的引用和申请的量子比特来提供量子环境。
VQNet定义了如下表所示运算符，所有运算符返回的都是类型为 var 的变量。

运算符	描述
`plus`	加。示例：a + b，其中a和b都是类型为 `var` 的变量。
`minus`	减。示例：a - b。
`multiply`	乘。示例：a * b。
`divide`	除。示例：a / b。
`exponent`	指数。示例：exp(a)。
`log`	对数。示例：log(a)。
`polynomial`	幂。示例：poly(a, 2)。
`dot`	矩阵点乘。示例：dot(a, b)。
`inverse`	求矩阵逆。示例：inverse(a)。
`transpose`	矩阵转置。示例：transpose(a)。
`sum`	矩阵求和。示例：sum(a)。
`stack`	矩阵按列(axis = 0)或行(axis = 1)方向进行拼接。示例：stack(axis, a, b, c)。
`subscript`	下标操作。示例：a[0]。
`qop`	量子操作，它将 `VQC` 和几个哈密顿量作为输入，并输出输入哈密顿量的期望。示例：如果w表示 `VQC` 的变量，qop(VQC(w)，Hamiltonians，[量子环境])。
`qop_pmeasure`	量子操作，它类似于qop。 qop_pmeasure的输入包括VQC，待测量的量子位和分量。它可以计算由待测量的量子比特构成的子空间中的所有投影状态的概率，并返回其中的一些。分量存储目标投影状态的标签。 很明显，任何投射状态的概率都可以看作哈密顿量的期望，所以qop_pmeasure是qop的一个特例。示例：qop_pmeasure(VQC(w), components, [量子环境])。
`sigmoid`	激活函数。示例：sigmoid(a)。
`softmax`	激活函数。示例：softmax(a)。
`cross_entropy`	交叉熵。示例：crossEntropy(a, b)。
`dropout`	dropout函数。示例：dropout(a, b)。

可变量子逻辑门

要在VQNet中使用量子操作 qop 或 qop_pmeasure ，就必须要包含可变量子线路(VariationalQuantumCircuit，别名 VQC)，而可变量子逻辑门则是构成 VQC的基本单位。

可变量子逻辑门(VariationalQuantumGate，别名 VQG)，内部维护着一组变量参数以及一组常量参数。 在构造 VQG 的时候只能对其中一组参数进行赋值。若含有一组常量参数，则可以通过 VQG 生成含确定参数的普通量子逻辑门, 若含有变量参数，则可以动态修改参数值，并生成对应的参数的普通量子逻辑门。

目前在 pyQPanda 中定义了如下可变量子逻辑门，它们都继承自 VariationalQuantumGate 。

VQG	别名
VariationalQuantumGate_H	VQG_H
VariationalQuantumGate_RX	VQG_RX
VariationalQuantumGate_RY	VQG_RY
VariationalQuantumGate_RZ	VQG_RZ
VariationalQuantumGate_CZ	VQG_CZ
VariationalQuantumGate_CNOT	VQG_CNOT

可变量子逻辑门

我们可以通过向可变量子线路中插入可变量子逻辑门，来使用可变量子逻辑门。我们可以向需要传入参数的可变量子逻辑门中传入变量参数，例如我们对可变量子逻辑门RX和RY传入变量参数x和y。也可以对可变量子逻辑门传入常量参数，例如RZ我们传入了一个常量参数0.12。 我们可以通过修改变量的参数，从而来改变可变量子逻辑门中的参数。

```
x = var(1) y = var(2) vqc = VariationalQuantumCircuit()
vqc.insert(VariationalQuantumGate_H(q[0]))
vqc.insert(VariationalQuantumGate_RX(q[0], x))
vqc.insert(VariationalQuantumGate_RY(q[1], y))
vqc.insert(VariationalQuantumGate_RZ(q[0], 0.12))
vqc.insert(VariationalQuantumGate_CZ(q[0], q[1]))
vqc.insert(VariationalQuantumGate_CNOT(q[0], q[1]))
circuit1 = vqc.feed()
x.set_value(3)
y.set_value(4)
circuit2 = vqc.feed()
```

1_VQC_VGQ.py

```
from pyqpanda import *
if __name__=="__main__":
    machine = init_quantum_machine(QMachineType.CPU)
    q = machine.qAlloc_many(2)
    x = var(1)
    y = var(2)
    vqc = VariationalQuantumCircuit()
    vqc.insert(VariationalQuantumGate_H(q[0]))
    vqc.insert(VariationalQuantumGate_RX(q[0], x))
    vqc.insert(VariationalQuantumGate_RY(q[1], y))
    vqc.insert(VariationalQuantumGate_RZ(q[0], 0.12))
    vqc.insert(VariationalQuantumGate_CZ(q[0], q[1]))
    vqc.insert(VariationalQuantumGate_CNOT(q[0], q[1]))
    circuit1 = vqc.feed()
    prog = QProg()
    prog.insert(circuit1)
    print(convert_qprog_to_originir(prog, machine))
    x.set_value([[3.]])
    y.set_value([[4.]])
    circuit2 = vqc.feed()
    prog2 = QProg()
    prog2.insert(circuit2)
    print(convert_qprog_to_originir(prog2, machine))
```

优化算法（梯度下降法）

VQNet中优化算法的使用，包括经典梯度下降算法和改进后的梯度下降算法，它们都是在求解机器学习算法的模型参数，即无约束优化问题时，最常采用的方法之一。
pyQPanda 中实现的算法：
VanillaGradientDescentOptimizer 、 MomentumOptimizer 、 AdaGradOptimizer 、 RMSPropOptimizer 和 AdamOptimizer，它们都继承自 Optimizer 。

```
VanillaGradientDescentOptimizer.minimize(
loss, # 损失函数
0.01, # 学习率
1.e-6) # 结束条件
MomentumOptimizer.minimize(
loss, # 损失函数
0.01, # 学习率
 0.9) # 动量系数
AdaGradOptimizer.minimize(
loss, # 损失函数
0.01, # 学习率
0.0, # 累加量起始值
1.e-10)# 很小的数值以避免零分母
```

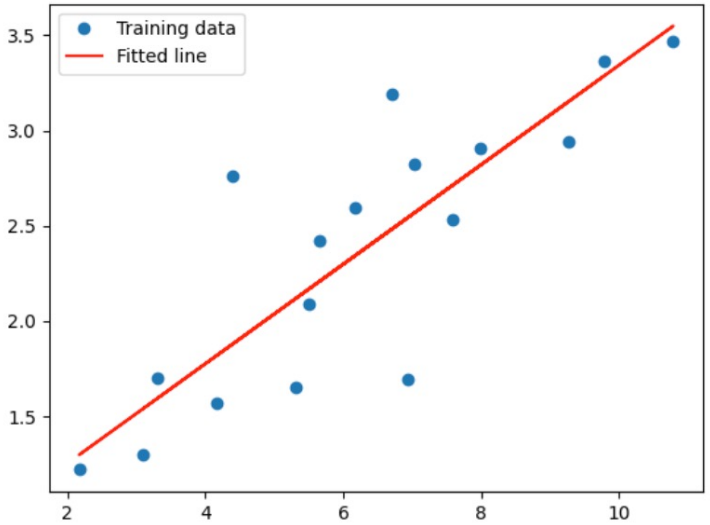
```
RMSOptimizer.minimize(
loss, # 损失函数
0.01, # 学习率
0.9, # 历史或即将到来的梯度的贴现因子
1.e-10)# 很小的数值以避免零分母

AdamOptimizer.minimize(
loss, # 损失函数
0.01, # 学习率
0.9, # 一阶动量衰减系数
0.999, # 二阶动量衰减系数
1.e-10)# 很小的数值以避免零分母
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/GradientOptimizer.html

优化算法（梯度下降法）

本章节将讲解VQNet中优化算法的使用，包括经典梯度下降算法和改进后的梯度下降算法，它们都是在求解机器学习算法的模型参数，即无约束优化问题时，最常采用的方法之一。 我们在 pyQPanda 中实现了这些算法：
VanillaGradientDescentOptimizer 、 MomentumOptimizer 、 AdaGradOptimizer 、 RMSPropOptimizer 和 AdamOptimizer ，
它们都继承自 Optimizer 。



2_Optimizer.py

```
...
x = np.array([3.3, 4.4, 5.5, 6.71, 6.93, 4.168, 9.779, 6.182, 7.59,
              2.167, 7.042, 10.791, 5.313, 7.997, 5.654, 9.27,3.1])
y = np.array([1.7, 2.76, 2.09, 3.19, 1.694, 1.573, 3.366, 2.596, 2.53,
              1.221, 2.827, 3.465, 1.65, 2.904, 2.42, 2.94,1.3])
X = var(x.reshape(len(x), 1))
Y = var(y.reshape(len(y), 1))
W = var(0, True)
B = var(0, True)
Y_ = W*X + B
loss = sum(poly(Y_ - Y, var(2))/len(x))
optimizer = VanillaGradientDescentOptimizer.minimize(loss, 0.01, 1.e-6)
leaves = optimizer.get_variables()
for i in range(1000):
    optimizer.run(leaves, 0)
    loss_value = optimizer.get_loss()
    print("i: ", i, " loss: ", loss_value, " W: ", eval(W,True), " b: ", eval(B,
True))
w2 = W.get_value()[0, 0]
b2 = B.get_value()[0, 0]
...
```

https://pyqpanda-toturial.readthedocs.io/zh/latest/GradientOptimizer.html

Thank You