Quantum Machine Learning

量子算法导论

Cirq

Cirq 是谷歌研究推出的开源量子计算软件库, 于2018年发布. 开发人员可以构建并运行包含所有相关一元、二元和三元门的量子算法. 目前, Cirq并不提供访问谷歌量子计算机的途径. 我们将使用Cirq内置的量子计算模拟器(称为Simulator)在本地执行量子算法.

首先通过一个简单的量子电路模拟来熟悉 Cirq . 在 Cirq 中, 量子比特通常使用 LineQubit 或GridQubit 选项来定义 . LineQubit 允许在一维格点上定义量子比特, 而 GridQubit 则允许在二维格点上定义量子比特 . 使用Cirq的GridQubit功能, 可以定义一个初始化为基态 $|\varphi\rangle$ 的量子比特, 并在其上应用 Hadamard 变换 $H=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$. 在 Cirq 中 , Hadamard 变换被定义为 H.

其实现过程如代码所示 , 使用 Cirq 中的测量功能在计算基础上测量新状态. 在许多其他量子计算软件包中, 测量通常需要显式定义用于存储测量结果的经典寄存器, 但在 Cirq 中并不需要.

在 Cirq 中 , 所有对量子比特的操作都以量子电路的形式定义. 一旦电路被定义 , 可以使用 Cirq模拟器运行 100 次相同电路的模拟并测量结果 . Cirq 具有统计功能 , 可获取每个测量结果的计数. 量子电路中任何状态的测量都可以与一个 key 关联 , 一旦模拟器运行完成 , 就可以通过 key 访问结果 .

```
# 导入cirq库
import cirq
# 定义一个量子比特
qubit = cirq. GridQubit(0,  0)
# 在Cirq中创建一个量子电路
circuit = cirq. Circuit([cirq. H(qubit),  cirq. measure(qubit,  key='m')])
print("电路如下")
print(circuit)
sim = cirq. Simulator()
output = sim. run(circuit,  repetitions=100)
print("测量输出:")
print(output)
print("统计如下")
print(output. histogram(key='m'))
```

输出结果为

```
电路如下
(0,  0): -H-M('m')-
测量输出:
m=11101011100000000100111000110100101100001001001101110010
10110011001011111101001101101110101010011101l0
统计如下
Counter({1: 52,  0: 48})
```

由代码输出可知, 在相同的量子比特副本上测量处于相等叠加态$1/\sqrt{2}(|0\rangle+|1\rangle)$的情况下, 得到48次状态为0的测量和52次状态为1的测量,

随着进行测量的量子状态数量的增加, 对于计算基态 $|0\rangle$ 和 $|1\rangle$ 的概率分布趋近于均匀分布:

$$\lim_{n\to\infty} P_n(0) = \lim_{n\to\infty} P_n(1) = \frac{1}{2}$$

其中 n 表示进行测量的量子状态数量, 而$P_n(0)$和$P_n(1)$分别表示从这 n 个测量中得到的基态 $|0\rangle$ 和 $|1\rangle$ 概率, 随着测量次数 n 趋近于无穷大, 观察到状态 $|0\rangle$ 的概率 $P_n(0)$ 和观察到状态 $|1\rangle$ 的概率 $P_n(1)$ 都趋向于 $\frac{1}{2}$, 表明在长期内测量到任一状态的可能性是相等的.

现在对不同的 n 值模拟H门变换, 并观察概率序列如何逐渐收敛到它们的期望值. 代码中创建一个名为 hadamard_state_measurement 的函数, 用于计算这些不同 n 值下的概率.

```python
# 导入 cirq 包
import cirq
import matplotlib. pyplot as plt
# 定义一个函数,  用于进行 Hadamard 状态的多次测量
def hadamard_state_measurement(copies):
    # 定义一个量子比特
    qubit = cirq. GridQubit(0,  0)
    # 在 cirq 中创建一个量子电路
    circuit = cirq. Circuit([
    cirq. H(qubit),   # 对量子比特应用 Hadamard 门
    cirq. measure(qubit,  key='m')  # 进行测量
    ])
    print("电路如下: ")
    print(circuit)
    sim = cirq. Simulator()
    output = sim. run(circuit,  repetitions=copies)  # 重复测量指定次数
    res = output. histogram(key='m')  # 获取测量结果的直方图
    prob_0 = dict(res)[0] / copies  # 计算测量结果为 0 的概率
    print(prob_0)
    return prob_0
```

```
# 主函数用于绘制概率收敛图
def main(copies_low=10,  copies_high=1000):

    probability_for_zero_state_trace = []  # 保存测量状态为 0 的概率
    copies_trace = []  # 保存测量次数
    for n in range(copies_low,  copies_high):
        copies_trace.append(n)
        prob_0 = hadamard_state_measurement(n)  # 获得测量状态为 0 的概率
        probability_for_zero_state_trace.append(prob_0)
    # 绘制概率收敛图
    plt. plot(copies_trace,  probability_for_zero_state_trace)
    plt. xlabel('测量次数')
    plt. ylabel("状态0的概率")
    plt. title("状态0概率的收敛序列")
    plt. show()
if __name__ == '__main__':
    main()
```

代码计算了相同量子态的不同测量中态 $|0\rangle$ 的概率, 这些量子处于态 $1/\sqrt{2}(|0\rangle+|1\rangle)$. 由图可得, 随着 n 的增加, 态 $|0\rangle$ 的概率逐渐向 $\frac{1}{2}$ 收敛, 并且震荡逐渐减小.

Qiskit

Qiskit是IBM推出的开源量子计算软件库, 于2017年发布.Qiskit 即 Quantum Information Science Kit , 其量子计算堆栈主要包括以下四个组件：

1. Qiskit Terra ： 提供构建量子电路的所有基本组件.
2. Qiskit Aer ： 您可以使用Aer工具开发噪声模型, 模拟在实际量子计算设备中可能发生的逼真噪声.Aer还提供了一个C++模拟器框架.
3. Qiskit Ignis ： 这是一个用于分析和最小化量子电路中噪声的框架.
4. Qiskit Agua ： 包含跨领域算法和在量子真实设备或模拟器上运行这些算法的逻辑.
   如下代码使用 Qiskit 实现与之前在Cirq中演示的相同程序：在Hadamard变换后测量一个量子比特.

```
#在Hadamard变换后测量一个量子比特
import numpy as np
from qiskit import QuantumCircuit,  execute,  Aer
from qiskit.visualization import plot_histogram
# 使用Aer的qasm_simulator
simulator = Aer.get_backend('qasm_simulator')
# 创建一个带有1个量子比特和1个经典比特的量子电路
circuit = QuantumCircuit(1,  1)
# 在量子比特0上添加H门（Hadamard变换）
circuit.h(0)
```

```
# 将量子测量映射到经典寄存器
circuit.measure([0],  [0])
# 在qasm模拟器上执行电路
job = execute(circuit,  simulator,  shots=100)
# 获取结果
result = job.result()
# 返回测量计数
counts = result.get_counts(circuit)
print("\n0和1的总计数为:",  counts)
# 绘制电路图
print(circuit.draw(output='text'))
```

输出结果如下

```
0和1的总计数为: {'0': 56,  '1': 44}
     ┌───┐ ┌─┐
  q: ┤ H ├─┤M├
     └───┘ └╥┘
c: 1/═══════╩═
            0
```

Qiskit 中使用 `QuantumCircuit` 定义量子电路.同时, 在定义电路时亦由 `QuantumCircuit` 定义所需的量子比特. `QuantumCircuit` 的其他输入是用于存储测量结果的经典比特.由于要测量叠加状态的量子比特的状态, 我们将需要一个用于测量的经典比特.与 Cirq 不同, Qiskit 中必须显式地定义用于存储测量结果的经典寄存器或比特.

在 Qiskit 中, Hadamard 变换 H 是通过使用 `QuantumCircuit` 创建的电路进行的.使用 `circuit.h(0)` 在唯一的量子比特上定义 Hadamard 变换.需要注意, 用于保存测量结果的经典比特在 Qiskit 中不会隐式地与量子比特关联, 我们需要在使用电路的 `measure` 功能进行测量时编写这种映射关系.

Qiskit导入的模拟器由 Aer 框架中引入的.与在 Cirq 中模拟量子电路的 `run` 命令类似, 在 Qiskit 中使用 `execute` 命令.

Bell 态的创建与测量

对于两个量子比特 A 和 B 的 Bell 态如下所示：

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

在下列代码中, 首先在状态 $|\psi_A\rangle = |0\rangle$ 的量子比特 A 上应用 Hadamard 变换 H 来创建贝尔态, 从而创建叠加态 $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.然后在状态 $|\psi_B\rangle = |0\rangle$ 的量子比特 B 上基于量子比特 A 作为控制位应用控制非门（CNOT门）, 通常称为CNOT门.

```
import cirq

# 使用 LineQubit 定义两个量子比特
q_register = [cirq.LineQubit(i) for i in range(2)]
# 在量子比特 0 上应用Hadamard门, 然后进行CNOT操作的电路定义
cirquit = cirq.Circuit([cirq.H(q_register[0]),  cirq.CNOT(q_register[0],
q_register[1])])

# 对两个量子比特进行测量
cirquit.append(cirq.measure(*q_register,  key='z'))

# 输出量子电路
print("电路")
print(cirquit)

# 定义量子模拟器
sim = cirq.Simulator()

# 对电路进行100次迭代模拟
output = sim.run(cirquit,  repetitions=100)

# 输出测量结果
print("测量输出")
print(output.histogram(key='z'))
```

结果如下

```
电路
0: ———H———@———M('z')———
          |   |
1: ———————X———M—————————
测量输出
Counter({3: 52,  0: 48})
```

上述代码使用了 Cirq 的 `LineQubit` 选项定义参与 Bell 态的两个量子比特.对 Bell 态进行测量时, 得到几乎相等比例的整数结果：0 和 3 .整数结果0表示状态 $|00\rangle$, 而结果3表示状态 $|11\rangle$.

现在按照下列代码中的示例, 在 Qiskit 中实现 Bell 态的创建和测量.

```
import numpy as np
from qiskit import QuantumCircuit,  execute,  Aer
from qiskit.visualization import plot_histogram
```

```
# 使用Aer的qasm_simulator
simulator = Aer.get_backend('qasm_simulator')
# 创建一个作用于q寄存器的量子电路
circuit = QuantumCircuit(2,  2)

# 在量子比特 0 上添加Hadamard门
circuit.h(0)
# 在控制量子比特 0 和目标量子比特 1 上添加CX (CNOT)门
circuit.cx(0,  1)

# 将量子测量映射到经典比特
circuit.measure([0,  1],  [0,  1])
# 在qasm模拟器上执行电路
job = execute(circuit,  simulator,  shots=100)

# 获取结果
result = job.result()
# 返回测量计数
counts = result.get_counts(circuit)
print("\n00和11的总计数为:",  counts)
# 绘制电路
print(circuit.draw(output='text'))
```

结果如下

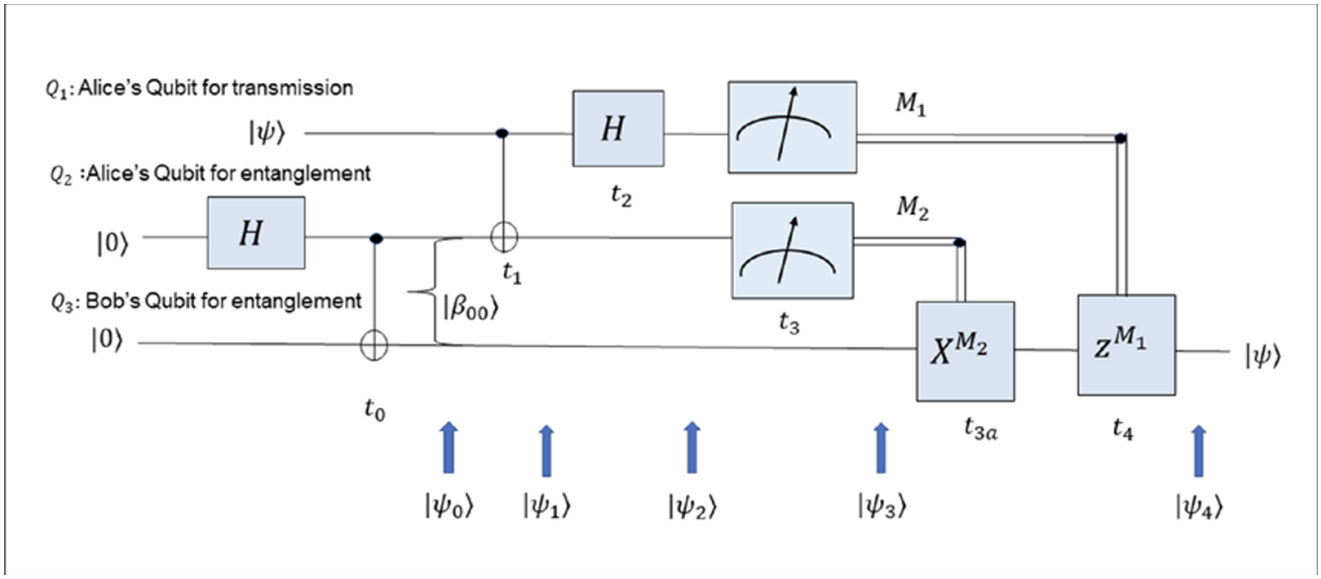
```
00和11的总计数为: {'00': 45,  '11': 55}
     ┌───┐     ┌┐
q_0: ┤ H ├──■──┤M├──
     └───┘┌─┴─┐└╥┘┌┐
q_1: ─────┤ X ├─╫─┤M├
          └───┘ ║ └╥┘
c: 2/═══════════╩══╩═
                0  1
```

输出中可以看出, 在对 Bell 态进行测量时, Qiskit 对态 $|00\rangle$ 和 $|11\rangle$ 进行了几乎均等的采样.

量子传送

量子传送是一种在发送方和接收方之间传输量子状态的方法, 无需使用任何通信通道.尊重传统 , 将量子态的发送方称为 Alice , 接收方称为 Bob , 以保持参考的一致性.图中显示

了量子传送电路的高级电路图.



前文中指出量子算法通过在量子比特之间创建有意义的关联来受益于量子纠缠, 这些关联的性质远比经典系统能够实现的要强大, 因为即使量子粒子分开很远, 它们也能表现出高度的相关性.

在量子传送中, Alice 和 Bob 通过量子纠缠使他们的控制量子比特共享一个贝尔态.Alice 想要发送给 Bob 一个量子比特状态$|\psi\rangle$.将要传输的这个量子比特称为 Q_1 , 而用于共享 Bell 态的控制量子比特 Alice 和 Bob 分别为 Q_2 和 Q_3.

以下是与量子传送算法相关的步骤：

1. 将控制量子比特 Q_2 和 Q3 初始化为状态 $|0\rangle$, 并将待传输的量子比特 Q_1 初始化为状态 $|\psi\rangle$.
2. 通过在 Q_2 上应用 Hadamard 变换 H 与在 Q_3 上应用 CNOT 操作, 在 Q_2 和 Q_3 之间创建贝尔态 $|00\rangle$ 和 $|11\rangle$.
3. 一旦 Alice 的和 Bob 的控制量子比特 Q_2 和 Q3 之间建立了 Bell 态, 就在 Alice 的两比特 Q_1 和 Q_2 上应用 CNOT 操作, 其中 Q_1 作为控制量子比特, Q_2 作为目标量子比特.接着, 在 Q_1 上应用Hadamard变换, 然后测量 Alice 的两比特 Q_1 和 Q_2 .用 M_1 和 M_2 表示 Q_1 和 Q_2 的测量状态.
4. 根据测量状态 M_2 , 以 M_1 作为控制量子比特, 在 Bob 的量子比特 Q_3 上应用 CNOT 操作.最后, 在 Bob 的量子比特 Q_3 上应用条件 Z 操作, 以 M_1 为测量状态.
5. 此时, Bob 的量子比特 Q_3 具有 Alice 传输的状态 $|\psi\rangle$.

下列代码使用 Cirq 中实现量子传送算法, 并通过传输等叠加态 $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ 进行说明.通常, 要传输的状态 $|\psi\rangle$ 可以通过将从状态 $|0\rangle$ 转换为所需状态 $|\psi\rangle$ 所需的电路来指定.在 `quantum_teleportation` 中使用 `qubit_to_send_op` 变量进行如上指定.例如, 要传输等叠加态 $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, 通过变量 `qubit_to_send_op` 将 `cirq.H` 操作器发送到 `quantum_teleportation` 中.Hadamard 操作符将初始化为状态 $|0\rangle$ 的量子比特 Q_1 转换为等叠加态.当然也可以使用 `qubit_to_send_op` 传输不同的状态.一旦量子比特状态已被传

输, 可以测量 Bob 的量子比特 Q_3, 以查看测量结果的分布是否与传输波形的概率分布相等.

```
import cirq

def quantum_teleportation(qubit_to_send_op='H',  num_copies=100):
    Q1,  Q2,  Q3 = [cirq.LineQubit(i) for i in range(3)]
    cirquit = cirq.Circuit()

    """
    Q1: Alice 要发送给 Bob 的量子比特状态
    Q2: Alice 的控制量子比特
    Q3: Bob 的控制量子比特

    根据 qubit_to_send_op 为 Q1 设置一个状态:
    实现的操作符有 H、X、Y、Z、I
    """
    if qubit_to_send_op == 'H':
        cirquit.append(cirq.H(Q1))
    elif qubit_to_send_op == 'X':
        cirquit.append(cirq.X(Q1))
    elif qubit_to_send_op == 'Y':
        cirquit.append(cirq.X(Q1))
    elif qubit_to_send_op == 'I':
        cirquit.append(cirq.I(Q1))
    else:
        raise NotImplementedError("尚未实现")

    # 使 Alice 和 Bob 的控制量子比特 Q2 和 Q3 产生纠缠
    cirquit.append(cirq.H(Q2))
    cirquit.append(cirq.CNOT(Q2,  Q3))

    # 使用 CNOT 操作将 Alice 的数据量子比特 Q1 与控制量子比特 Q2 相关联
    cirquit.append(cirq.CNOT(Q1,  Q2))

    # 使用 Hadamard 变换在 +/- 基础上转换 Alice 的数据量子比特 Q1
    cirquit.append(cirq.H(Q1))

    # 测量 Alice 的量子比特 Q1 和 Q2
    cirquit.append(cirq.measure(Q1,  Q2))

    # 在测量后, 使用 Alice 的控制量子比特 Q2 对 Bob 的量子比特 Q3 进行 CNOT
操作
    cirquit.append(cirq.CNOT(Q2,  Q3))

    # 在测量后, 使用 Alice 的控制量子比特 Q1 对 Bob 的量子比特 Q3 进行条件 Z
```

```
操作
    cirquit.append(cirq.CZ(Q1,  Q3))

    # 测量最终传输给 Bob 的状态在 Q3 中
    cirquit.append(cirq.measure(Q3,  key='Z'))

    print("电路")
    print(cirquit)

    sim = cirq.Simulator()
    output = sim.run(cirquit,  repetitions=num_copies)

    print("测量输出")
    print(output.histogram(key='Z'))

if __name__ == '__main__':
    quantum_teleportation(qubit_to_send_op='H')
```

结果如下

```
电路
0: ──H───────@──H──M───────@──────────────
             │      │       │
1: ──H──@────X──────M──@────┼──────────────
        │               │   │
2: ─────X───────────────X──@──M('Z')──────
测量输出
Counter({0: 53,  1: 47})
```

从测量结果中可以看出, Alice 已经成功将等叠加态传输给了 Bob.

量子随机数生成器

在经典计算机中, 大多数随机数生成器实际上并非真正随机, 因为它们是通过算法以确定性的方式生成的, 因此遵循可再现性的规范.确切地说, 经典随机数生成器从一个初始种子状态开始, 使用该种子状态生成的随机数序列总是相同的.因此, 我们可以看到这些随机数生成器生成的数字序列模拟了随机数序列的特性, 同时又是确定性的.这些确定性随机数生成器例程被称为伪随机生成器.伪随机数具有可重现性和速度的优势, 但不能安全地用于诸如使用随机密码键来安全传输数据的密码学应用.

伪随机生成器的相反是硬件随机数生成器, 它利用诸如量子过程、光电效应等物理过程生成随机数.由于这些物理过程是高度不可预测的, 它们可以作为真正随机数生成器的良好基础, 可用于安全应用, 如密码学.

现在将演示使用多个量子比特的随机整数生成器.在量子计算中这将变得非常简单, 如下所示：

1. 确定表示要抽样的整数值范围所需的量子比特数量.例如, 如果需要从0到7的八个整数中抽样, 我们将需要 $\log_2\ 8 = 3$ 个量子比特.
2. 通过在初始处于 $|0\rangle$ 状态的每个量子比特上应用 Hadamard 变换来创建等叠加态.等叠加态由以下表示

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes} = \frac{1}{2^{\frac{n}{2}}}\sum_{x=0}^{2^n-1}|x\rangle$$

其中 $|x\rangle$ 表示计算基态 $|\ x_0x_1\ldots x_{n-1}\rangle$ 的整数值且 $x_i \in \{0,1\}$.

3. 将计算基态映射到实际整数, 并将映射存储在一个名为 `s2n_map` 的字典中.如果要抽样的整数范围从零开始, 从计算基态到实际整数的字典可以是二进制到十进制转换, 如下所示：

$$x_0x_1\ldots x_{n-1} = \sum_{i=0}^{n-1} x_i 2^{n-1-i}$$

如果范围从偏移量 b 开始, 我们可以将从计算基态到要抽样的整数值的映射如下：

$$x_0x_1\ldots x_{n-1} \rightarrow \sum_{i=0}^{n-1} x_i 2^{n-1-i} + b = 0$$

4. 一旦定义了映射, 可以对等叠加态 $|\psi\rangle$ 进行测量, 并使用字典映射 `s2n_map` 将测量的计算基态映射到整数值.

下列代码使用 10 个量子比特从 0 到 2^{10} 生成随机数.由于从 0 开始抽样, 算法的偏移量 b 为0.

```
import cirq
import numpy as np

def random_number_generator(low=0,  high=2**10,  m=10):
    """
    :param low: 要生成的数字的下限
    :param high: 要生成的数字的上限
    :param m: 要输出的随机数的数量
    :return: 随机数的字符串
    """
    # 确定所需的量子比特数
    qubits_required = int(np.ceil(np.log2(high - low)))
    print(qubits_required)
```

```
    # 定义量子比特
    Q_reg = [cirq.LineQubit(c) for c in range(qubits_required)]

    # 定义电路
    circuit = cirq.Circuit()
    circuit.append(cirq.H(Q_reg[c]) for c in range(qubits_required))
    circuit.append(cirq.measure(*Q_reg,  key="z"))
    print(circuit)

    # 模拟电路
    sim = cirq.Simulator()
    num_gen = 0
    output = []

    while num_gen <= m:
        result = sim.run(circuit,  repetitions=1)
        rand_number = result.data.values()[0][0] + low
        if rand_number < high:
            output.append(rand_number)
            num_gen += 1

    return output

if __name__ == '__main__':
    output = random_number_generator()
    print(output)
```

结果如下

```
10
0: ──H──M('z')──
        │
1: ──H──M───────
        │
2: ──H──M───────
        │
3: ──H──M───────
        │
4: ──H──M───────
        │
5: ──H──M───────
        │
6: ──H──M───────
        │
7: ──H──M───────
```

```
          │
8: ──H──M─────────
          │
9: ──H──M─────────
[542,  210,  376,  534,  216,  363,  89,  933,  629,  611,  847]
```

从随机数生成器中, 可以看到它包括在每个量子比特上应用 Hadamard 算符后进行测量.从量子随机数生成器的输出中, 可以看到它生成的20个数字的样本均值接近于从中抽样的数字的均值, 即假设均匀分布的0到2^{10} .