circuit_diagrams

February 10, 2020

1 Quantum Circuits

1.1 Dependencies and Imports

[1]:
```
# Read packages into Python library:
import pennylane as qml
from pennylane import numpy as np
```

1.2 Introduction

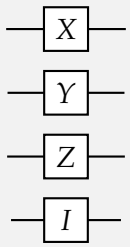
Now that we have all of the basics of linear algebra and tensor products out of the way, let's have a look at multi-qubit gates in more depth and in the context of quantum circuits. Quantum circuits are diagrams that show how quantum logic gates act on qubits. They are a special case of a more general structure called **tensor networks**. We will talk about tensor networks, quantum complexity via entanglement entropy, and some of the historical motivations from various areas of physics in the appendix. For now, let's focus on some of the basics and how to interpret quantum circuit diagrams.

1.3 Circuit Diagrams of Common Gates

In this section we will use the operators listed below in circuit diagrams in order to learn to convert quantum circuit diagrams into linear algebra operations.

1.3.1 Pauli Gates

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{1}$$

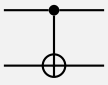


1.3.2 The Hadamard Gate

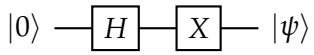
$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2}$$

1.3.3 The "CNOT" Gate

$$\mathbf{CX} = \begin{pmatrix} I & 0 \\ 0 & X \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3}$$



As an example, let's try converting the following circuit diagram into linear algebra:



$$\begin{aligned}
X \cdot H|0\rangle &= X \cdot H \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} && (4)\\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} && (5)\\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (6)\\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (7)\\
&= \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} && (8)\\
&= \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) && (9)\\
&= |r\rangle \quad \text{(spin-right in the X-basis)} && (10)\\
&= |\psi\rangle && (11)
\end{aligned}$$

In code, we can construct a PennyLane `qnode()` that performs these gate operations and then samples the circuit

[2]:
```
# Create a device to run the code
dev = qml.device("default.qubit", wires=1, shots=1)

#Create the qnode
@qml.qnode(dev)
def circuit():
    qml.Hadamard(wires=[0])
    qml.PauliX(wires=[0])
    return qml.sample(qml.PauliX(0))

print(circuit())
```

```
1
```

Notice, we compute the sample in the X-basis by having the `qnode()` return `qml.sample(qml.PauliX(0))`. In the previous chapter, we always sampled using the Z-basis by having the circuit return `qml.sample(qml.PauliZ(0))` We will discuss this in more depth when we talk about **measurements** and **expectation values**. Since the result of the circuit gave the state

$$|\psi\rangle = |r\rangle \tag{12}$$

$$= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{13}$$

the sampling returned a "$+1$" value. If we instead run the same curcuit, but prepare the intitial state to be $|1\rangle$ we will get a different result:

[3]:
```
#Define an array corresponding to the initial state |1>
u = np.array([1])

#Create the qnode
@qml.qnode(dev)
def circuit():
    qml.BasisState(u, wires=[0])
    qml.Hadamard(wires=[0])
    qml.PauliX(wires=[0])
    return qml.sample(qml.PauliX(0))

print(circuit())
```

```
-1
```

Working through the linear algebra, we get the following:

$$X \cdot H|1\rangle = X \cdot H \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{14}$$

$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{15}$$

$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{16}$$

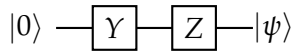
$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{17}$$

$$= \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \tag{18}$$

$$= |l\rangle \tag{19}$$

So, we should indeed expect a sample value of "-1" when measuring in the X-basis.

Let's work through another example. Let's take the following diagram and convert it to linear algebra:



$$Z \cdot Y|0\rangle = Z \cdot Y \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{20}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{21}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ i \end{pmatrix} \tag{22}$$

$$= \begin{pmatrix} 0 \\ -i \end{pmatrix} \tag{23}$$

$$= |\psi\rangle \tag{24}$$

Notice,

$$\begin{pmatrix} 0 \\ -i \end{pmatrix} = -i \begin{pmatrix} 0 \\ 1 \end{pmatrix} = -i|1\rangle \tag{25}$$

We can write a circuit that performs these operations, and then samples the circuit in the Z-basis:

[4]:
```
#Create the qnode
@qml.qnode(dev)
def circuit():
    qml.PauliY(wires=[0])
    qml.PauliZ(wires=[0])
    return qml.sample(qml.PauliZ(0))

print(circuit())
```

```
-1
```

Due to the way measurements behave, we are always going to get a value of "-1" when sampling the state $-i|1\rangle$. We'll explain why later on. We can write a circuit that performs these operations, and then samples the circuit in the X-basis:

[5]:
```
#Create the qnode
@qml.qnode(dev)
def circuit():
    qml.PauliY(wires=[0])
    qml.PauliZ(wires=[0])
    return qml.sample(qml.PauliX(0))

print(circuit())
```

```
-1
```

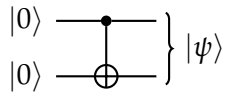
In this `qnode()`, the X-basis sample value is a quantum phenomenon not seen in classical phsyics. In fact if we run the circuit several times by defining a new "20-shot device" we will see that we get a mixture of "$+1$" and "-1" sample values:

[6]:
```
# Create a device to run the code
dev2 = qml.device("default.qubit", wires=1, shots=20)

#Create the qnode
@qml.qnode(dev2)
def circuit():
    qml.PauliY(wires=[0])
    qml.PauliZ(wires=[0])
    return qml.sample(qml.PauliX(0))

print(circuit())
```

```
[ 1 -1 -1  1  1  1 -1  1 -1  1  1  1 -1 -1  1 -1  1  1  1  1]
```

Explaining this strange behavior of measuring something different each time and not getting a single sample value for every measurement is something we will dig into when discussing measurements. Even though we can work out a unique pure state using linear algebra, measurements behave in a way me might not intially expect. In order to get the uniqe pure state out of the measurement, we have to perform a measurement in the basis defined by that state.

Let's have a look at a more complicated circuit involving two qubits now:



Let's work out the linear algebra:

$$\mathbf{CX}|00\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{26}$$

$$= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{27}$$

$$= |00\rangle \tag{28}$$

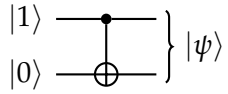
$$= |\psi\rangle \tag{29}$$

Now, let's write a circuit that performs these operations and samples the circuit in the Z-basis one time. We will need a device with two wires, and we have to tell the `qml.CNOT()` function which wires to operate on:

[7]:
```
# Create a device to run the code
dev3 = qml.device("default.qubit", wires=2, shots=1)

#Create the qnode
@qml.qnode(dev3)
def circuit():
    qml.CNOT(wires=[0,1])
    return qml.sample(qml.PauliZ(0))

print(circuit())
```

```
1
```

Let's do the almost same thing but lets make the following change to the circuit:



This is prepared in the $|10\rangle$ basis state, so we should expect a different outcome state. Let's have a look at the math:

$$\mathbf{CX}|10\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{30}$$

$$= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{31}$$

$$= |11\rangle \tag{32}$$

$$= |\psi\rangle \tag{33}$$

As we can see, the second qubit value is flipped now. Let's build a circuit that performs these operations an samples in the Z-basis on both qubits once:

[8]:
```
# Create a device to run the code
dev3 = qml.device("default.qubit", wires=2, shots=1)

# Define an array corresponding to the initial basis state:
ud = np.array([1,0])

#Create the qnode
@qml.qnode(dev3)
def circuit():
    qml.BasisState(ud, wires=[0,1])
    qml.CNOT(wires=[0,1])
    return qml.sample(qml.PauliZ(0)), qml.sample(qml.PauliZ(1))

print(circuit())
```

```
[[-1]
 [-1]]
```

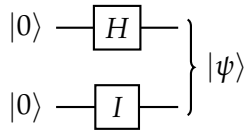
We can see, since the final state is $|\psi\rangle = |11\rangle$, we get "$-1$" sample values for the functions

```
qml.sample(qml.PauliZ(0))
qml.sample(qml.PauliZ(1))
```

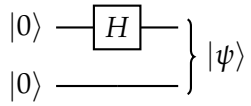
In general, if we perform the **CNOT** operation on two qubits in some initial basis state we get the following output states:

$$\mathbf{CX}|00\rangle = |00\rangle \tag{34}$$
$$\mathbf{CX}|01\rangle = |01\rangle \tag{35}$$
$$\mathbf{CX}|10\rangle = |11\rangle \tag{36}$$
$$\mathbf{CX}|11\rangle = |10\rangle \tag{37}$$

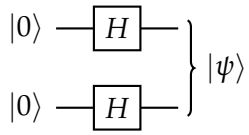
It is important to mention at this point that the following circuit diagram:



Is equivalent to this diagram:



Much more complicated examples can of course be created, but the general idea remains the same. Absence of a gate means the identity gate. We generally can stack gates to get their tensor product. For example the following circuit:



Can be interpreted in terms of linear algebra as follows:

$$(H \otimes H)|00\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\left(\begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix}\right) \tag{38}$$

$$= \frac{1}{2}\begin{pmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{pmatrix}\begin{pmatrix}1\\0\\0\\0\end{pmatrix} \tag{39}$$

$$= \begin{pmatrix}1/2\\1/2\\1/2\\1/2\end{pmatrix} \tag{40}$$

This can also be computer in a slightly different but completely equivalent way which may be easier for some people to parse:

$$(H \otimes H)|00\rangle = H|0\rangle \otimes H|0\rangle \tag{41}$$

$$= \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} \tag{42}$$

$$= \frac{1}{2}\left(\begin{pmatrix}1\\1\end{pmatrix} \otimes \begin{pmatrix}1\\1\end{pmatrix}\right) \tag{43}$$

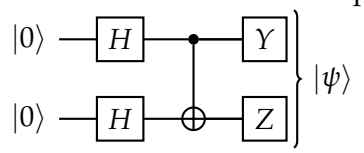
$$= \begin{pmatrix}1/2\\1/2\\1/2\\1/2\end{pmatrix} \tag{44}$$

Let's implement this in PennyLane but with the initial basis state $|10\rangle$:

[9]:
```
# Create a device to run the code
dev3 = qml.device("default.qubit", wires=2, shots=1)

# Define an array corresponding to the initial basis state:
ud = np.array([1,0])

#Create the qnode
@qml.qnode(dev3)
def circuit():
    qml.BasisState(ud, wires=[0,1])
    qml.Hadamard(wires=[0])
    qml.Hadamard(wires=[1])
    return qml.sample(qml.PauliZ(0)), qml.sample(qml.PauliZ(1))

print(circuit())
```

```
[[-1]
 [-1]]
```

Let's look at one final example that is more complicated:



This one is a little tricky if you don't know about tensor products so let's go through it very carefully. First, operate on each qubit with the Hadamard gate to turn the "ket-0" into a superposition:

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}. \tag{45}$$

Now, take the two qubits in superposition (two copies of the above computation), and take their tensor product:

$$H|0\rangle \otimes H|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \tag{46}$$

$$= \frac{1}{2}\begin{pmatrix}1\\1\\1\\1\end{pmatrix} \tag{47}$$

$$= \begin{pmatrix}1/2\\1/2\\1/2\\1/2\end{pmatrix} \tag{48}$$

Next, operate on this by the **CNOT**-gate

$$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}1/2\\1/2\\1/2\\1/2\end{pmatrix} = \begin{pmatrix}1/2\\1/2\\1/2\\1/2\end{pmatrix} \tag{49}$$

Next, we operate on the tensor product vector with the operator $Y \otimes Z$:

$$Y \otimes Z \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix} \tag{50}$$

$$= \begin{pmatrix} 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i \\ i & 0 & 0 & 0 \\ 0 & -i & 0 & 0 \end{pmatrix} \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix} \tag{51}$$

$$= \begin{pmatrix} -i/2 \\ i/2 \\ i/2 \\ -i/2 \end{pmatrix} \tag{52}$$

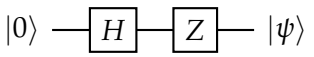
Now, let's implement some PennyLane code to do the same operations on two qubits:

[10]:
```
# Create a device to run the code
dev3 = qml.device("default.qubit", wires=2, shots=1)

#Create the qnode
@qml.qnode(dev3)
def circuit():
    qml.Hadamard(wires=[0])
    qml.Hadamard(wires=[1])
    qml.CNOT(wires=[0,1])
    qml.PauliY(wires=[0])
    qml.PauliZ(wires=[1])
    return qml.sample(qml.PauliZ(0)), qml.sample(qml.PauliZ(1))

print(circuit())
```

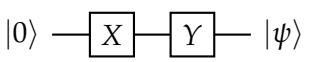
```
[[-1]
 [ 1]]
```

1.3.4 Exercises

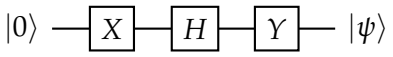
1. Write PennyLane code to construct and measure the following circuit in the Z-basis. Compute the linear algebra by hand to see the output states.



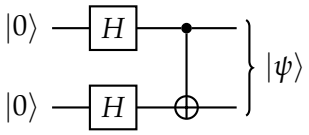
2. Write PennyLane code to construct and measure the following circuit in the Z-basis. Compute the linear algebra by hand to see the output states.



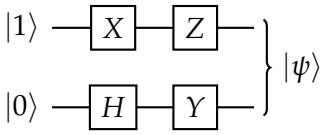
3. Write PennyLane code to construct and measure the following circuit in the Z-basis. Compute the linear algebra by hand to see the output states.



4. Write PennyLane code to construct and measure the following circuit in the Z-basis. Compute the linear algebra by hand to see the output states.



5. Write PennyLane code to construct and measure the following circuit in the Z-basis. Compute the linear algebra by hand to see the output states.



6. Write PennyLane code to construct and measure the following circuit in the Z-basis. Compute the linear algebra by hand to see the output states.