In [1]:

```
from qiskit import *
from qiskit.quantum_info import random_statevector
from scipy.optimize import minimize
from scipy.optimize import Bounds
import numpy as np
import matplotlib.pyplot as plt
from qiskit.visualization import plot_histogram
import time
pi=np.pi
%matplotlib inline
backend = Aer.get_backend('statevector_simulator')
```

In [2]:

```
def norm(vector):
    return sum(abs(vector)**2)
```

In [3]:

```
def make_circuit(thetas,qr_length):
    qr=QuantumRegister(qr_length,'q')
    U_total=QuantumCircuit(qr)
    L=len(thetas)/(2*qr_length)
    for cL in range(int(L)):
        theta_cL = thetas[int(cL*2*qr_length):int((cL+1)*2*qr_length)]
        theta_odd_cL  = theta_cL[0:int(qr_length)]
        theta_even_cL = theta_cL[int(qr_length):int(2*qr_length)]

        U_odd_cL=QuantumCircuit(qr)
        for jo in range(qr_length):
            U_odd_cL.rx(theta_odd_cL[jo],jo)

        U_even_cL=QuantumCircuit(qr)
        for je in range(qr_length):
            U_even_cL.rz(theta_even_cL[je],je)
        for je in range(qr_length-1):
            U_even_cL.cz(je,range(je+1,qr_length))

        U_block_cL=U_odd_cL+U_even_cL
        U_total=U_total+U_block_cL
        U_total.barrier()

    return U_total
```

In [4]:

```
def _cost(thetas,qr_length,target_state):

    U_total=make_circuit(thetas,qr_length)

    job=execute(U_total,backend)
    result = job.result()
    final_state=result.get_statevector()

    cost=norm(final_state-target_state.data)
    return cost
```

In [5]:

```
# Definitions

qr_length=4
dim=2**qr_length
target_state=random_statevector(dim)
```

In [6]:

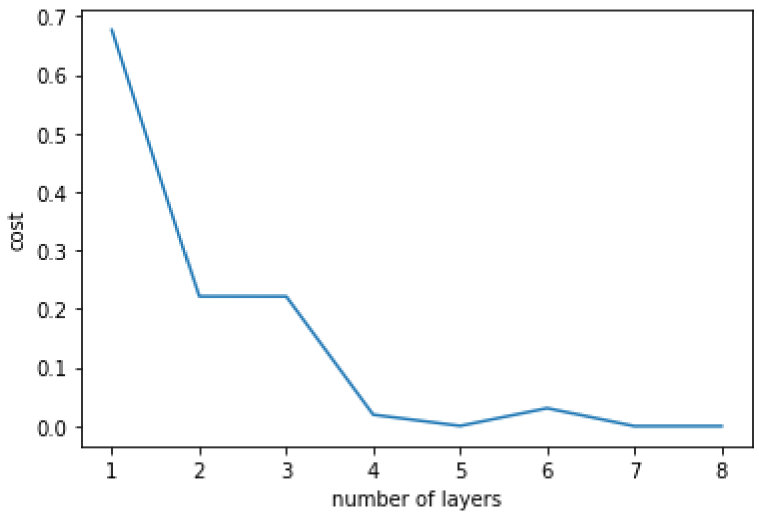
```
# Main Code

maxL=10
theta_set = np.ndarray(0)
cost=[]
for L in range(maxL):
    start=time.time()
    new_set    = np.random.rand(2*qr_length)*2*pi
    theta_set = np.append(theta_set,new_set)
    res=minimize(_cost,theta_set,args=(qr_length,target_state),bounds=Bounds(np.zero
s(len(theta_set)),np.ones(len(theta_set))*2*pi),method='Powell')
    print(res.success)
    theta_set=res.x
    cost.append(res.fun)
    print('cost vector is \n',cost)
    end=time.time()
    print('time elapsed for L=',L+1,'is',end-start)
    if (L>3 and cost[L-1]<1e-6 and cost[L]<1e-6):
        maxL=L+1
        print('The required number of layers to converge is L = ',maxL)
        break
```

```
True
cost vector is
 [0.6766155625997988]
time elapsed for L= 1 is 12.135694026947021
True
cost vector is
 [0.6766155625997988, 0.2217818210404969]
time elapsed for L= 2 is 37.6261522769928
True
cost vector is
 [0.6766155625997988, 0.2217818210404969, 0.22163578802499445]
time elapsed for L= 3 is 171.57781386375427
True
cost vector is
 [0.6766155625997988, 0.2217818210404969, 0.22163578802499445, 0.0195884
59741763875]
time elapsed for L= 4 is 510.01117062568665
True
cost vector is
 [0.6766155625997988, 0.2217818210404969, 0.22163578802499445, 0.0195884
59741763875, 0.0004730910048419851]
time elapsed for L= 5 is 1321.5265872478485
True
cost vector is
 [0.6766155625997988, 0.2217818210404969, 0.22163578802499445, 0.0195884
59741763875, 0.0004730910048419851, 0.030690545933776688]
time elapsed for L= 6 is 299.5301322937012
True
cost vector is
 [0.6766155625997988, 0.2217818210404969, 0.22163578802499445, 0.0195884
59741763875, 0.0004730910048419851, 0.030690545933776688, 7.043940849622
995e-10]
time elapsed for L= 7 is 1071.7042980194092
True
cost vector is
 [0.6766155625997988, 0.2217818210404969, 0.22163578802499445, 0.0195884
59741763875, 0.0004730910048419851, 0.030690545933776688, 7.043940849622
995e-10, 7.435303676339498e-10]
time elapsed for L= 8 is 1630.5092597007751
The required number of layers to converge is L =  8
```

In [7]:

```
plt.plot(range(1,maxL+1),cost)
plt.xlabel('number of layers')
plt.ylabel('cost')
plt.show()
```



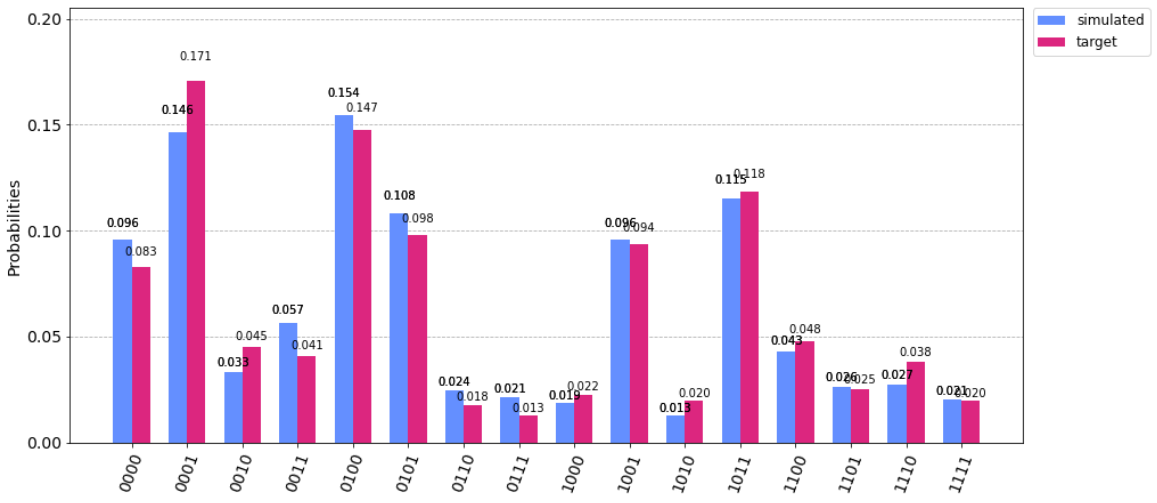
In [8]:

```
meas = QuantumCircuit(QuantumRegister(qr_length,'q'),ClassicalRegister(qr_length,'c
0'))
meas.measure(range(qr_length), range(qr_length))
final_circ = make_circuit(theta_set,qr_length)+meas
# final_circ.draw('mpl')
```

In [9]:

```
backend_sim = Aer.get_backend('qasm_simulator')
job_sim = execute(final_circ, backend_sim, shots=1024)
result_sim = job_sim.result()
counts = result_sim.get_counts()
plot_histogram([counts,target_state.sample_counts(1024)],figsize=(15,7),legend=['sim
ulated','target'])
```

Out[9]:



In [ ]:

In [ ]: