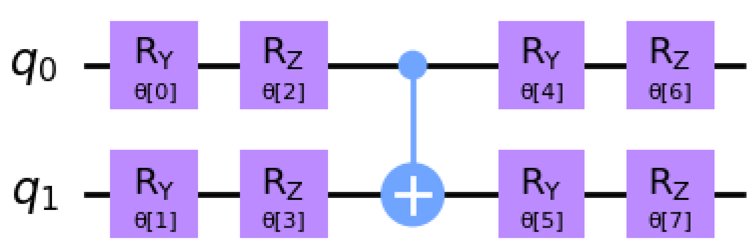
In [1]:
```python
import json
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import clear_output
from qiskit import QuantumCircuit
from qiskit.circuit import ParameterVector
from qiskit.circuit.library import ZFeatureMap, ZZFeatureMap
from qiskit.quantum_info import SparsePauliOp
from qiskit_algorithms.optimizers import COBYLA
from qiskit_algorithms.utils import algorithm_globals
from qiskit_machine_learning.algorithms.classifiers import NeuralNetworkClas
from qiskit_machine_learning.neural_networks import EstimatorQNN
from sklearn.model_selection import train_test_split

algorithm_globals.random_seed = 12345
```

In [2]:
```python
# We now define a two qubit unitary as defined in [3]
def conv_circuit(params):
    target = QuantumCircuit(2)
    target.ry(params[0], 0)
    target.ry(params[1], 1)
    target.rz(params[2], 0)
    target.rz(params[3], 1)
    target.cx(0, 1)
    target.ry(params[4], 0)
    target.ry(params[5], 1)
    target.rz(params[6], 0)
    target.rz(params[7], 1)
    return target


# Let's draw this circuit and see what it looks like
params = ParameterVector("θ", length=8)
circuit = conv_circuit(params)
circuit.draw("mpl", style="clifford")
```

Out[2]:



In [3]:
```python
def conv_layer(num_qubits, param_prefix):
    qc = QuantumCircuit(num_qubits, name="Convolutional Layer")
    qubits = list(range(num_qubits))
    param_index = 0
    params = ParameterVector(param_prefix, length=num_qubits * 8)
    for q1, q2 in zip(qubits[0::2], qubits[1::2]):
        qc = qc.compose(conv_circuit(params[param_index : (param_index + 8)]
```

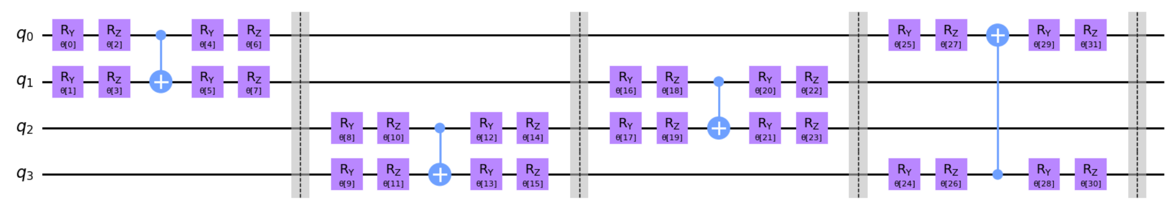
```
        qc.barrier()
        param_index += 8
    for q1, q2 in zip(qubits[1::2], qubits[2::2] + [0]):
        qc = qc.compose(conv_circuit(params[param_index : (param_index + 8)]
        qc.barrier()
        param_index += 8

    qc_inst = qc.to_instruction()

    qc = QuantumCircuit(num_qubits)
    qc.append(qc_inst, qubits)
    return qc


circuit = conv_layer(4, "θ")
circuit.decompose().draw("mpl", style="clifford")
```

Out[3]:

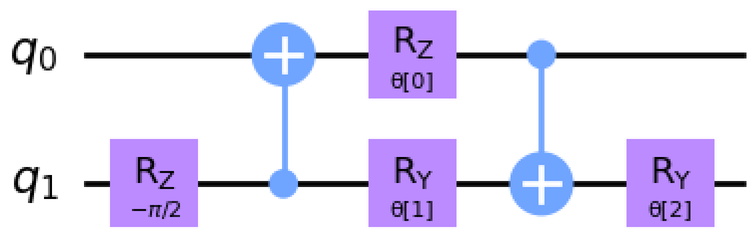


In [4]:
```
def pool_circuit(params):
    target = QuantumCircuit(2)
    target.rz(-np.pi / 2, 1)
    target.cx(1, 0)
    target.rz(params[0], 0)
    target.ry(params[1], 1)
    target.cx(0, 1)
    target.ry(params[2], 1)

    return target


params = ParameterVector("θ", length=3)
circuit = pool_circuit(params)
circuit.draw("mpl", style="clifford")
```

Out[4]:



In [5]:
```
def pool_layer(sources, sinks, param_prefix):
    num_qubits = len(sources) + len(sinks)
    qc = QuantumCircuit(num_qubits, name="Pooling Layer")
    param_index = 0
    params = ParameterVector(param_prefix, length=num_qubits // 2 * 3)
```

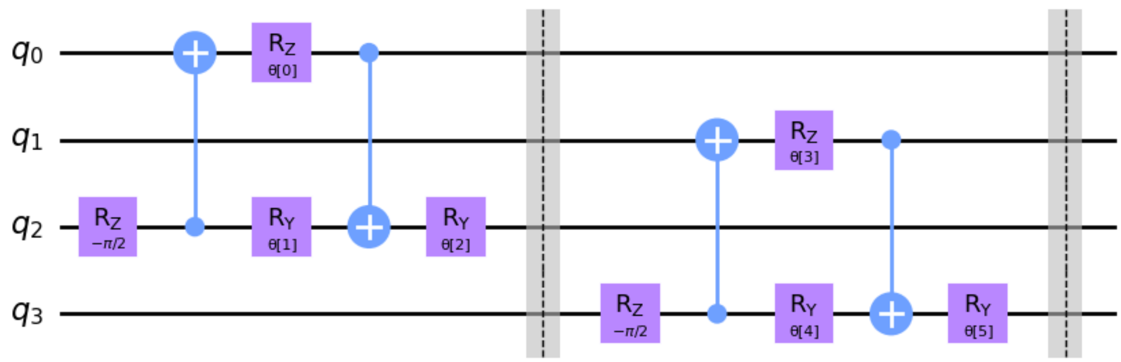
```
    for source, sink in zip(sources, sinks):
        qc = qc.compose(pool_circuit(params[param_index : (param_index + 3)]
        qc.barrier()
        param_index += 3

    qc_inst = qc.to_instruction()

    qc = QuantumCircuit(num_qubits)
    qc.append(qc_inst, range(num_qubits))
    return qc


sources = [0, 1]
sinks = [2, 3]
circuit = pool_layer(sources, sinks, "θ")
circuit.decompose().draw("mpl", style="clifford")
```

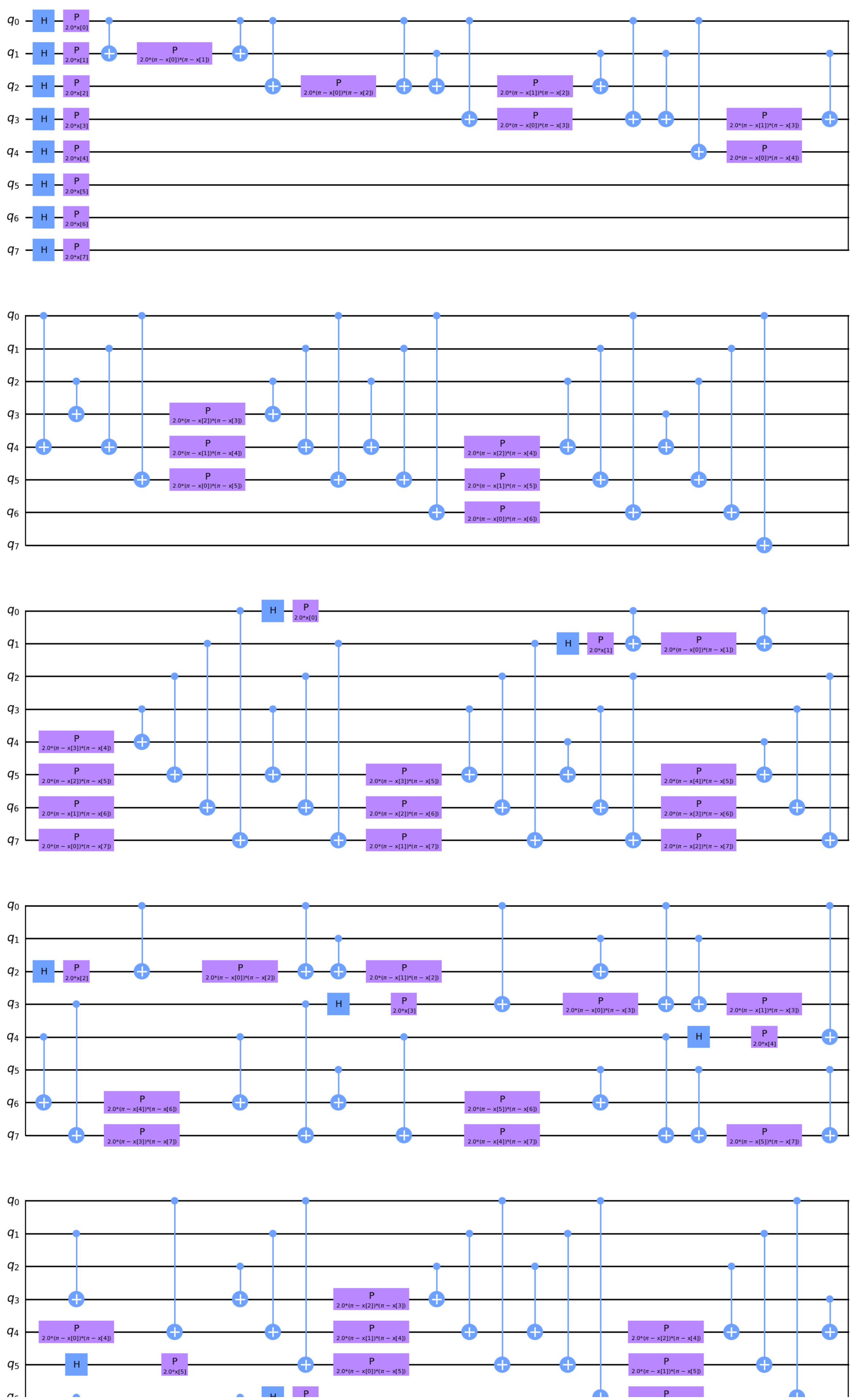
Out[5]:



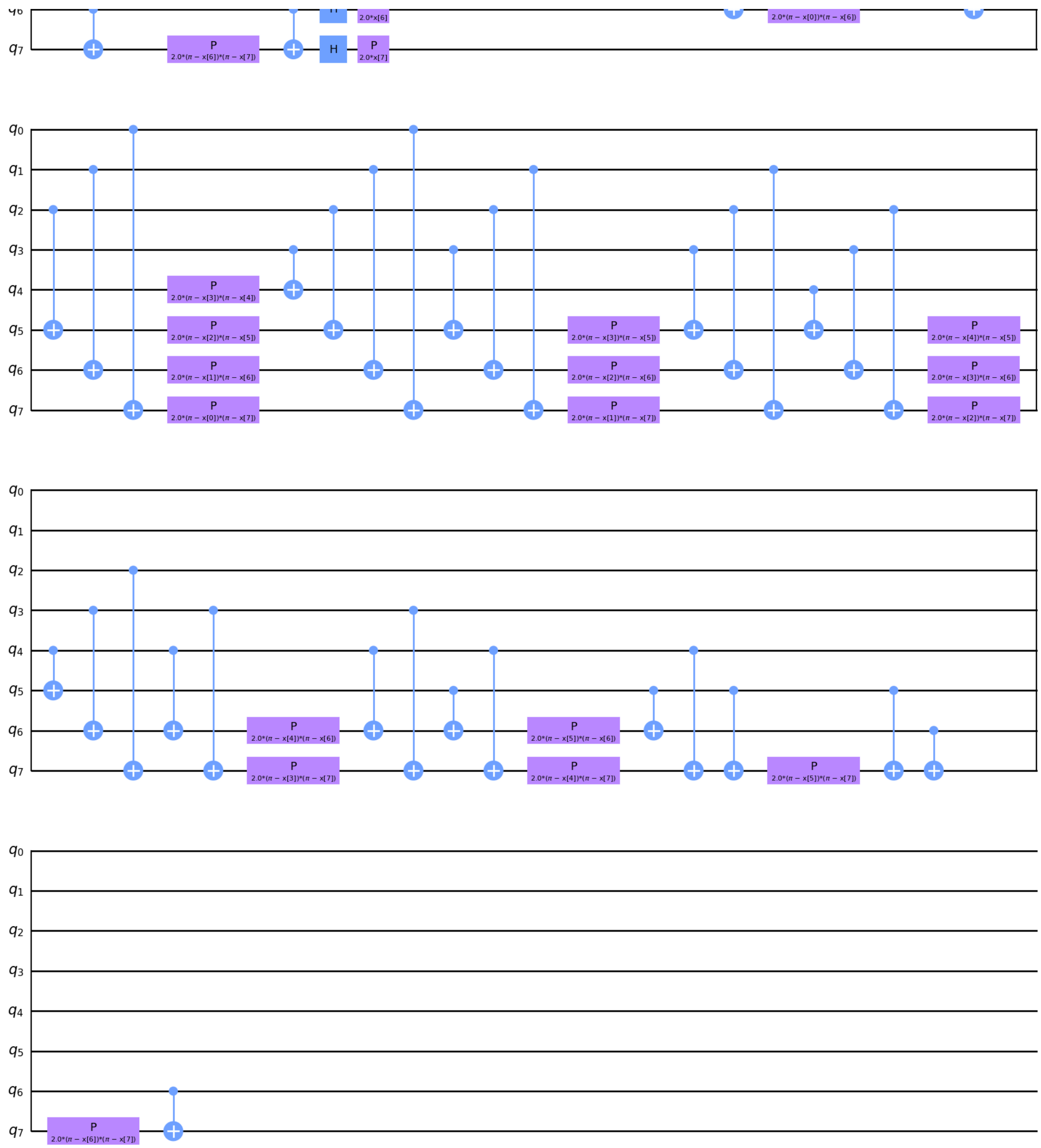
In [6]:

```
feature_map = ZZFeatureMap(8)
feature_map.decompose().draw("mpl", style="clifford")
```

Out[6]:



q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
H
P
2.0*x[0]
2.0*x[1]
2.0*x[2]
2.0*x[3]
2.0*x[4]
2.0*x[5]
2.0*x[6]
2.0*x[7]
2.0*(π − x[0])*(π − x[1])



In [7]:

```
feature_map = ZZFeatureMap(8)

ansatz = QuantumCircuit(8, name="Ansatz")

# First Convolutional Layer
ansatz.compose(conv_layer(8, "c1"), list(range(8)), inplace=True)

# First Pooling Layer
ansatz.compose(pool_layer([0, 1, 2, 3], [4, 5, 6, 7], "p1"), list(range(8)),

# Second Convolutional Layer
ansatz.compose(conv_layer(4, "c2"), list(range(4, 8)), inplace=True)

# Second Pooling Layer
ansatz.compose(pool_layer([0, 1], [2, 3], "p2"), list(range(4, 8)), inplace=

# Third Convolutional Layer
```

```
ansatz.compose(conv_layer(2, "c3"), list(range(6, 8)), inplace=True)

# Third Pooling Layer
ansatz.compose(pool_layer([0], [1], "p3"), list(range(6, 8)), inplace=True)

# Combining the feature map and ansatz
circuit = QuantumCircuit(8)
circuit.compose(feature_map, range(8), inplace=True)
circuit.compose(ansatz, range(8), inplace=True)

observable = SparsePauliOp.from_list([("Z" + "I" * 7, 1)])
```

In [8]:
```
from qiskit_aer import AerSimulator, Aer
from qiskit_aer import AerError

try:
    simulator_gpu = Aer.get_backend('aer_simulator')
    simulator_gpu.set_options(device='GPU')
except AerError as e:
    print(e)

from qiskit.primitives import Sampler, BackendSampler, BackendEstimator

sampler = BackendSampler(simulator_gpu)

estimator = BackendEstimator(backend=simulator_gpu)
```

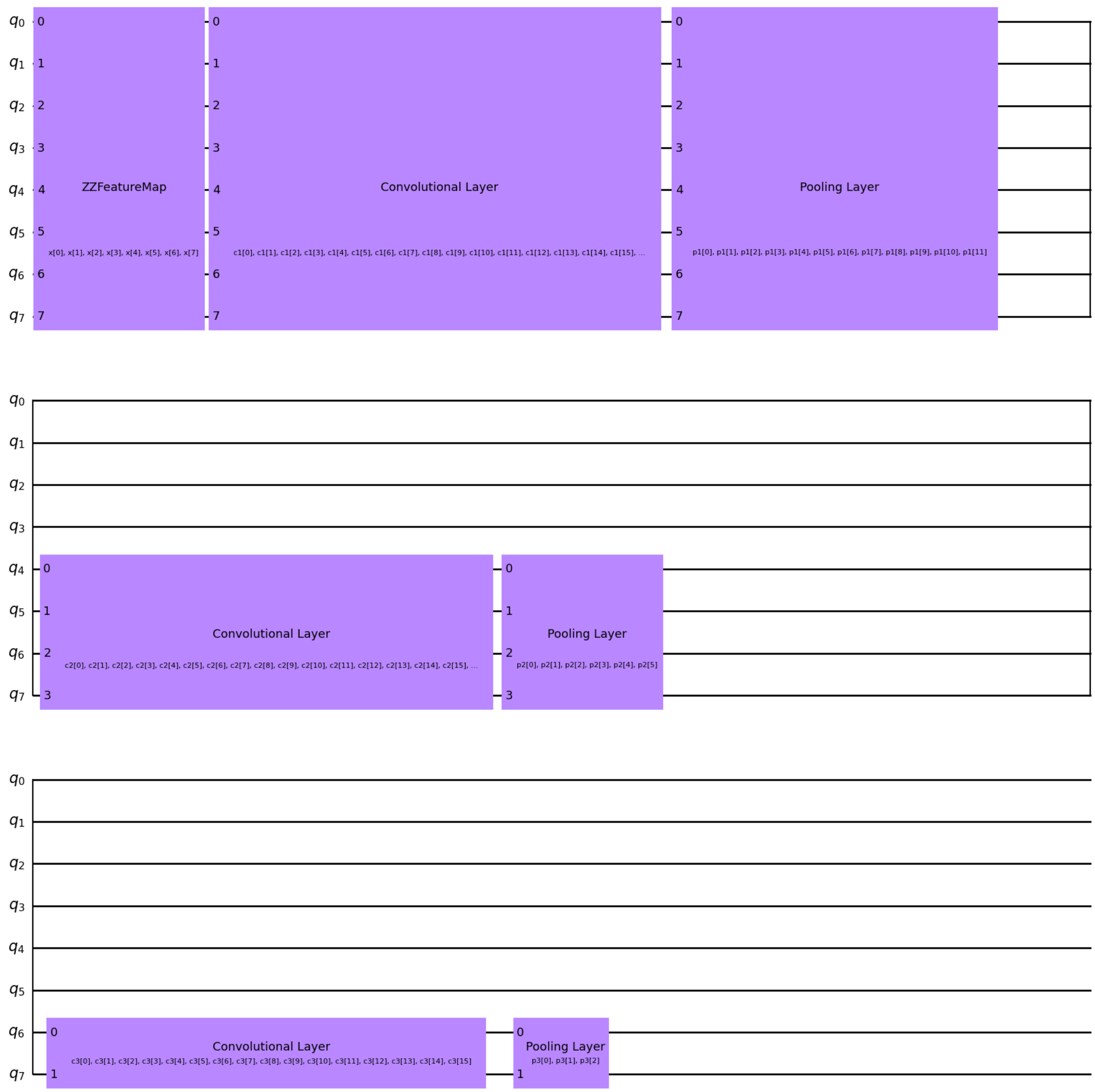
In [9]:
```
Aer.backends()
```

Out[9]:
```
[AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator'),
 QasmSimulator('qasm_simulator'),
 StatevectorSimulator('statevector_simulator'),
 UnitarySimulator('unitary_simulator')]
```

In [10]:
```
# we decompose the circuit for the QNN to avoid additional data copying
qnn = EstimatorQNN(
    circuit=circuit.decompose(),
    observables=observable,
    input_params=feature_map.parameters,
    weight_params=ansatz.parameters,
)
```

In [11]:
```
circuit.draw("mpl", style="clifford")
```

Out[11]:



In [12]:

```
def callback_graph(weights, obj_func_eval):
    #clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    #plt.title("Objective function value against iteration")
    #plt.xlabel("Iteration")
    #plt.ylabel("Objective function value")
    #plt.plot(range(len(objective_func_vals)), objective_func_vals)
    #plt.show()
```

In [13]:

```
classifier = NeuralNetworkClassifier(
    qnn,
    optimizer=COBYLA(maxiter=100),  # Set max iterations here
    callback=callback_graph,
    loss='cross_entropy'
)
```

In [14]:

```
import pandas as pd

df1 = pd.read_csv("fault.csv")
```

```
df2 = pd.read_csv("faultlabel.csv")
X_data=np.array(df1.iloc[:,0:10])
Y_data=np.array(df2.iloc[:,0:2])
```

In [15]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncode

scaler_mm = MinMaxScaler()
X_data_temp = scaler_mm.fit_transform(X_data)
X_data_temp = PCA(n_components=8).fit_transform(X_data_temp)
```

In [16]:
```
ohe_transformer = OneHotEncoder(sparse_output = False)
Y_data_temp = Y_data
```

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X_data_temp, Y_data_temp
```

In [18]:
```
import time

start = time.time()
plt.rcParams['figure.figsize'] = [12, 6]

objective_func_vals = []

y_train_fit = y_train.T[0]

classifier.fit(X_train, y_train_fit)

elapsed = time.time() - start

print("Time elapsed: ", elapsed)
```

```
Time elapsed:  7126.5441699028015
```

In [19]:
```
# score classifier
print(f"Accuracy from the train data : {np.round(100 * classifier.score(X_tr
```

```
Accuracy from the train data : 40.37%
```

In [20]:
```
y_test_score = np.array(y_test).T[0]
print(f"Accuracy from the train data : {np.round(100 * classifier.score(X_te
```

```
Accuracy from the train data : 40.84%
```

In [21]:
```
objective_func_vals
```

```
Out[21]: [9.566345273167073,
          9.950782032134098,
          8.516491917154546,
          8.993729287736937,
          9.227311047643655,
          8.687664179467115,
          9.039582094779588,
          8.93935450027027,
          8.041252806814287,
          8.7605392560615,
          7.826583177314451,
          7.393701361160823,
          7.849176364344413,
          7.672035322967629,
          8.563007433087812,
          7.846665722783309,
          7.631319058646819,
          8.509498217916663,
          7.424551617770169,
          7.197843602495047,
          8.369283613971593,
          8.059092322734243,
          7.77207177083269,
          7.407514808071961,
          7.018988734723191,
          7.418509044234091,
          6.87918957870523,
          6.6180612558540215,
          7.337227492187725,
          6.314320850212639,
          7.0406571852122966,
          6.74859263333921,
          7.0955124708584,
          6.581050323859263,
          6.5430530563265785,
          7.273129978587532,
          6.450285252971487,
          7.267252640295513,
          6.727595022033137,
          7.282956221088007,
          6.692424386666182,
          7.3379601729331405,
          8.058700057970329,
          7.157330006875227,
          6.396665451962954,
          6.941466683704933,
          6.703817658207377,
          6.980422263180457,
          6.581280308081779,
          6.888690592997655,
          7.129896023476378,
          6.6530234185221975,
          7.271737125776483,
          6.189447693475198,
          6.861702531595399,
          5.876940503335002,
```

```
6.259000898767815,
6.724316430119913,
6.199738372038809,
6.872318174498332,
5.939883256627687,
6.143922548766498,
6.730168294133137,
6.895721744908391,
6.277349028623487,
7.301823513333052,
6.4834050346921,
6.8208038661218,
6.972331622450143,
6.488620224073222,
6.618114469595834,
8.4027807025667,
6.622758947127651,
6.883419429876299,
6.226465570233262,
6.589025948747023,
5.964423802283557,
7.22561486922916,
5.96595620332056,
6.424055894581743,
5.4187872510981405,
6.331189736071469,
7.29091695173248,
6.90797732361555,
6.159393109585649,
7.745186441669924,
4.896304382731509,
6.294985980903155,
6.459153302942613,
5.203530887131014,
5.263963814236685,
5.8035562173044575,
7.4747977569057,
5.970421920487186,
4.9497970517707675,
5.847200756373094,
5.9961584912024986,
6.8153718618567645,
6.3327005537937175,
5.309126335885622]
```

In [ ]: