Neural Networks with qiskit

Quantum Computing refers to the use of quantum mechanical phenomena such as *superposition* and *entanglement* to perform computation. To understand quantum phenomena, it's important to understand qubit, the unit of quantum information, and the concepts of superposition and entanglement.

Qubit or Quantum Bit is the unit of quantum information, analogous to the 'bit' in classical computing. In order to differentiate between a classical bit and a qubit, Dirac notation (kit notation) is used. So, the qubits are represented as |0〉 and |1〉 and are often read as 'zero state' and 'one state'.

The 'NeuralNetwork' represents the interface for all neural networks available in Qiskit Machine Learning. It exposes a forward and a backward pass taking the data samples and trainable weights as input. A 'NeuralNetwork' does not contain any training capabilities, these are pushed to the actual algorithms / applications. Thus, a 'NeuralNetwork' also does not store the values for trainable weights. In the following, different implementations of this interfaces are introduced.

Suppose a 'NeuralNetwork' called nn. Then, the nn.forward(input, weights) pass takes either flat inputs for the data and weights of size nn.num_inputs and nn.num_weights, respectively. NeuralNetwork supports batching of inputs and returns batches of output of the corresponding shape.

Neurons and Weights A neural network is ultimately just an elaborate function that is built by composing smaller building blocks called neurons. A neuron is typically a simple, easy-to-compute, and nonlinear function that maps one or more inputs to a single real number. The single output of a neuron is typically copied and fed as input into other neurons. Graphically, we represent neurons as nodes in a graph and we draw directed edges between nodes to indicate how the output of one neuron will be used as input to other neurons. It's also important to note that each edge in our graph is often associated with a scalar-value called a weight. The idea here is that each of the inputs to a neuron will be multiplied by a different scalar before being collected and processed into a single value. The objective when training a neural network consists primarily of choosing our weights such that the network behaves in a particular way.

Feed Forward Neural Networks It is also worth noting that the particular type of neural network we will concern ourselves with is called a feed-forward neural network (FFNN). This means that as data flows through our neural network, it will never return to a neuron it has already visited. Equivalently, you could say that the graph which describes our neural network is a directed acyclic graph (DAG). Furthermore, we will stipulate that neurons within the same layer of our neural network will not have edges between them.

IO Structure of Layers The input to a neural network is a classical (real-valued) vector. Each component of the input vector is multiplied by a different weight and fed into a layer of neurons according to the graph structure of the network. After each neuron in the layer has been evaluated, the results are collected into a new vector where the i'th component records the output of the i'th neuron. This new vector can then be treated as an input for a new layer, and so on. We will use the standard term hidden layer to describe all but the first and last layers of our network.

**So How Does Quantum Enter the Picture? * To create a quantum-classical neural network, one can implement a hidden layer for our neural network using a parameterized quantum circuit. By "parameterized quantum circuit", we mean a quantum circuit where the rotation angles for each gate are specified by the components of a classical input vector. The outputs from our neural network's previous layer will be collected and used as the inputs for our parameterized circuit. The measurement statistics of our quantum circuit can then be collected and used as inputs for the following layer.

```
!pip install qiskit
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.7/dist-packages (0.25.0)
Requirement already satisfied: qiskit-ibmq-provider==0.12.2 in /usr/local/lib/python3.7/dist-packages (from qiskit) (0.12.2)
Requirement already satisfied: qiskit-aer==0.8.0 in /usr/local/lib/python3.7/dist-packages (from qiskit) (0.8.0)
Requirement already satisfied: qiskit-terra==0.17.0 in /usr/local/lib/python3.7/dist-packages (from qiskit) (0.17.0)
Requirement already satisfied: qiskit-ignis==0.6.0 in /usr/local/lib/python3.7/dist-packages (from qiskit) (0.6.0)
Requirement already satisfied: qiskit-aqua==0.9.0 in /usr/local/lib/python3.7/dist-packages (from qiskit) (0.9.0)
Requirement already satisfied: requests>=2.19 in /usr/local/lib/python3.7/dist-packages (from qiskit-ibmq-provider==0.12.2->qiskit) (2.23.0
Requirement already satisfied: requests-ntlm>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-ibmq-provider==0.12.2->qiskit) (
Requirement already satisfied: websockets>=8 in /usr/local/lib/python3.7/dist-packages (from qiskit-ibmq-provider==0.12.2->qiskit) (8.1)
Requirement already satisfied: numpy>=1.13 in /usr/local/lib/python3.7/dist-packages (from qiskit-ibmq-provider==0.12.2->qiskit) (1.19.5)
Requirement already satisfied: urllib3>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from qiskit-ibmq-provider==0.12.2->qiskit) (1.24.
Requirement already satisfied: nest-asyncio!=1.1.0,>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-ibmq-provider==0.12.2->qi
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-ibmq-provider==0.12.2->qiskit)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-aer==0.8.0->qiskit) (1.4.1)
Requirement already satisfied: pybind11>=2.6 in /usr/local/lib/python3.7/dist-packages (from qiskit-aer==0.8.0->qiskit) (2.6.2)
Requirement already satisfied: fastjsonschema>=2.10 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra==0.17.0->qiskit) (2.15.0)
Requirement already satisfied: retworkx>=0.8.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra==0.17.0->qiskit) (0.8.0)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra==0.17.0->qiskit) (5.4.8)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra==0.17.0->qiskit) (2.6.0)
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra==0.17.0->qiskit) (1.7.1)
Requirement already satisfied: python-constraint>=1.4 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra==0.17.0->qiskit) (1.4.0)
Requirement already satisfied: ply>=3.10 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra==0.17.0->qiskit) (3.11)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra==0.17.0->qiskit) (0.3.3)
Requirement already satisfied: setuptools>=40.1.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-ignis==0.6.0->qiskit) (54.2.0)
Requirement already satisfied: yfinance<=0.1.55 in /usr/local/lib/python3.7/dist-packages (from qiskit-aqua==0.9.0->qiskit) (0.1.55)
Requirement already satisfied: h5py<=3.1.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-aqua==0.9.0->qiskit) (2.10.0)
Requirement already satisfied: fastdtw<=0.3.4 in /usr/local/lib/python3.7/dist-packages (from qiskit-aqua==0.9.0->qiskit) (0.3.4)
Requirement already satisfied: scikit-learn<=0.24.1,>=0.20.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-aqua==0.9.0->qiskit) (0
Requirement already satisfied: pandas<=1.2.3 in /usr/local/lib/python3.7/dist-packages (from qiskit-aqua==0.9.0->qiskit) (1.1.5)
Requirement already satisfied: docplex<=2.20.204; sys_platform != "darwin" in /usr/local/lib/python3.7/dist-packages (from qiskit-aqua==0.9
Requirement already satisfied: dlx<=1.0.4 in /usr/local/lib/python3.7/dist-packages (from qiskit-aqua==0.9.0->qiskit) (1.0.4)
Requirement already satisfied: quandl<=3.6.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-aqua==0.9.0->qiskit) (3.6.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.19->qiskit-ibmq-provider==0.12
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.19->qiskit-ibmq-provider==0.12.2->q
```

```
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.19->qiskit-ibmq-provider==0.1
Requirement already satisfied: cryptography>=1.3 in /usr/local/lib/python3.7/dist-packages (from requests-ntlm>=1.1.0->qiskit-ibmq-provider
Requirement already satisfied: ntlm-auth>=1.0.2 in /usr/local/lib/python3.7/dist-packages (from requests-ntlm>=1.1.0->qiskit-ibmq-provider=
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.8.0->qiskit-ibmq-provider==0.12.
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.7/dist-packages (from sympy>=1.3->qiskit-terra==0.17.0->qiskit) (1.2.
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-packages (from yfinance<=0.1.55->qiskit-aqua==0.9.0->qi
Requirement already satisfied: lxml>=4.5.1 in /usr/local/lib/python3.7/dist-packages (from yfinance<=0.1.55->qiskit-aqua==0.9.0->qiskit) (4
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn<=0.24.1,>=0.20.0->qiskit-aqua==0.9
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas<=1.2.3->qiskit-aqua==0.9.0->qiskit) (201
Requirement already satisfied: more-itertools in /usr/local/lib/python3.7/dist-packages (from quandl<=3.6.0->qiskit-aqua==0.9.0->qiskit) (8
Requirement already satisfied: inflection>=0.3.1 in /usr/local/lib/python3.7/dist-packages (from quandl<=3.6.0->qiskit-aqua==0.9.0->qiskit)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.7/dist-packages (from cryptography>=1.3->requests-ntlm>=1.1.0->qiskit-i
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from cffi>=1.12->cryptography>=1.3->requests-ntlm>=1.1.
```

```
import numpy as np

from qiskit import Aer, QuantumCircuit
from qiskit.circuit import Parameter
from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap
from qiskit.opflow import StateFn, PauliSumOp, AerPauliExpectation, ListOp, Gradient
from qiskit.utils import QuantumInstance
```

```
# set method to calculcate expected values
expval = AerPauliExpectation()

# define gradient method
gradient = Gradient()

# define quantum instances (statevector and sample based)
qi_sv = QuantumInstance(Aer.get_backend('statevector_simulator'))

# we set shots to 10 as this will determine the number of samples later on.
qi_qasm = QuantumInstance(Aer.get_backend('qasm_simulator'), shots=10)
```

```
!pip install qiskit_machine_learning
from qiskit_machine_learning.neural_networks import OpflowQNN
```

```
Requirement already satisfied: qiskit_machine_learning in /usr/local/lib/python3.7/dist-packages (0.1.0)
Requirement already satisfied: fastdtw in /usr/local/lib/python3.7/dist-packages (from qiskit_machine_learning) (0.3.4)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.7/dist-packages (from qiskit_machine_learning) (5.4.8)
Requirement already satisfied: sparse in /usr/local/lib/python3.7/dist-packages (from qiskit_machine_learning) (0.12.0)
Requirement already satisfied: setuptools>=40.1.0 in /usr/local/lib/python3.7/dist-packages (from qiskit_machine_learning) (54.2.0)
Requirement already satisfied: qiskit-terra>=0.17.0 in /usr/local/lib/python3.7/dist-packages (from qiskit_machine_learning) (0.17.0)
Requirement already satisfied: scipy>=1.4 in /usr/local/lib/python3.7/dist-packages (from qiskit_machine_learning) (1.4.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from qiskit_machine_learning) (1.19.5)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-packages (from qiskit_machine_learning) (0.22.2.post1)
Requirement already satisfied: numba>=0.49 in /usr/local/lib/python3.7/dist-packages (from sparse->qiskit_machine_learning) (0.51.2)
Requirement already satisfied: ply>=3.10 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra>=0.17.0->qiskit_machine_learning) (3.
Requirement already satisfied: retworkx>=0.8.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra>=0.17.0->qiskit_machine_learnin
Requirement already satisfied: python-constraint>=1.4 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra>=0.17.0->qiskit_machine_
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra>=0.17.0->qiskit_machine_learning) (0.
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra>=0.17.0->qiskit_machine_
Requirement already satisfied: fastjsonschema>=2.10 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra>=0.17.0->qiskit_machine_le
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra>=0.17.0->qiskit_machine_learnin
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.7/dist-packages (from qiskit-terra>=0.17.0->qiskit_machine_learning) (1
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->qiskit_machine_learning)
Requirement already satisfied: llvmlite<0.35,>=0.34.0.dev0 in /usr/local/lib/python3.7/dist-packages (from numba>=0.49->sparse->qiskit_mach
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.8.0->qiskit-terra>=0.17.0->qiski
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.7/dist-packages (from sympy>=1.3->qiskit-terra>=0.17.0->qiskit_machin
```

```
# construct parametrized circuit
params1 = [Parameter('input1'), Parameter('weight1')]
qc1 = QuantumCircuit(1)
qc1.h(0)
qc1.ry(params1[0], 0)
qc1.rx(params1[1], 0)
qc_sfn1 = StateFn(qc1)

# construct cost operator
H1 = StateFn(PauliSumOp.from_list([('Z', 1.0), ('X', 1.0)]))

# combine operator and circuit to objective function
op1 = ~H1 @ qc_sfn1
print(op1)
```

```
ComposedOp([
  OperatorMeasurement(1.0 * Z
  + 1.0 * X),
  CircuitStateFn(
       ┌───┐┌─────────────┐┌──────────────┐
  q_0: ┤ H ├┤ RY(input1) ├┤ RX(weight1) ├
       └───┘└─────────────┘└──────────────┘
  )
])
```

```
# construct OpflowQNN with the operator, the input parameters, the weight parameters,
# the expected value, gradient, and quantum instance.
```

```
qnn1 = OpflowQNN(op1, [params1[0]], [params1[1]], expval, gradient, qi_sv)
```

```
# define (random) input and weights
input1 = np.random.rand(qnn1.num_inputs)
weights1 = np.random.rand(qnn1.num_weights)
```

```
# QNN forward pass
qnn1.forward(input1, weights1)
```

```
array([[0.25014613]])
```

```
# QNN batched forward pass
qnn1.forward([input1, input1], weights1)
```

```
array([[0.25014613],
       [0.25014613]])
```

```
# QNN backward pass
qnn1.backward(input1, weights1)
```

```
(array([[[-1.40430685]]]), array([[[0.10742726]]]))
```

```
# QNN batched backward pass
qnn1.backward([input1, input1], weights1)
```

```
(array([[[-1.40430685]],

        [[-1.40430685]]]), array([[[0.10742726]],

        [[0.10742726]]]))
```

```
op2 = ListOp([op1, op1])
qnn2 = OpflowQNN(op2, [params1[0]], [params1[1]], expval, gradient, qi_sv)
```

```
# QNN forward pass
qnn2.forward(input1, weights1)
```

```
array([[0.05629417, 0.05629417]])
```

```
# QNN backward pass
qnn2.backward(input1, weights1)
```

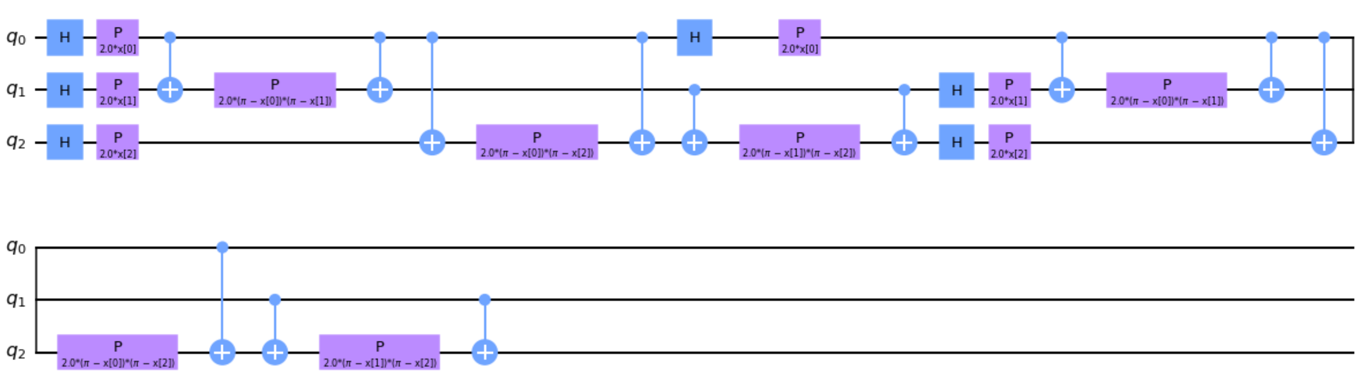
```
(array([[[-1.40430685],
         [-1.40430685]]]), array([[[0.10742726],
         [0.10742726]]]))
```

```
from qiskit_machine_learning.neural_networks import TwoLayerQNN
```

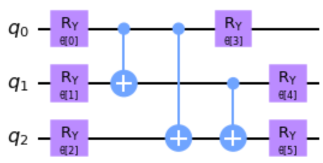
```

# specify the number of qubits
num_qubits = 3
```

```
!pip install pylatexenc
# specify the feature map
fm = ZZFeatureMap(num_qubits, reps=2)
fm.draw(output='mpl')
```

```
Requirement already satisfied: pylatexenc in /usr/local/lib/python3.7/dist-packages (2.10)
```




```
# specify the ansatz
ansatz = RealAmplitudes(num_qubits, reps=1)
ansatz.draw(output='mpl')
```




```
# specify the observable
observable = PauliSumOp.from_list([('Z'*num_qubits, 1)])
print(observable)
```

```
1.0 * ZZZ
```

```
# define two layer QNN
qnn3 = TwoLayerQNN(num_qubits,
                   feature_map=fm,
                   ansatz=ansatz,
                   observable=observable, quantum_instance=qi_sv)
```

```
# define (random) input and weights
input3 = np.random.rand(qnn3.num_inputs)
weights3 = np.random.rand(qnn3.num_weights)
```

```
# QNN forward pass
qnn3.forward(input3, weights3)
```

```
array([[-0.2791278]])
```

```
# QNN backward pass
qnn3.backward(input3, weights3)
```

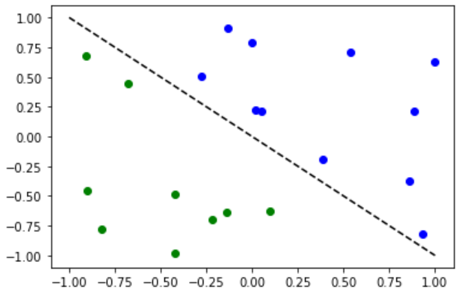
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-d6547a5f1220> in <module>()
      1 # QNN backward pass
----> 2 qnn3.backward(input3, weights3)

NameError: name 'qnn3' is not defined
```

SEARCH STACK OVERFLOW

Classification with qiskit

```
import numpy as np
import matplotlib.pyplot as plt

from torch import Tensor
from torch.nn import Linear, CrossEntropyLoss, MSELoss
from torch.optim import LBFGS

from qiskit  import Aer, QuantumCircuit
from qiskit.utils import QuantumInstance
from qiskit.opflow import AerPauliExpectation
from qiskit.circuit import Parameter
from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap
from qiskit_machine_learning.neural_networks import CircuitQNN, TwoLayerQNN
from qiskit_machine_learning.connectors import TorchConnector

qi = QuantumInstance(Aer.get_backend('statevector_simulator'))
```

```
num_inputs = 2
num_samples = 20
X = 2*np.random.rand(num_samples, num_inputs) - 1
y01 = 1*(np.sum(X, axis=1) >= 0)  # in { 0,  1}
y = 2*y01-1                       # in {-1, +1}

X_ = Tensor(X)
y01_ = Tensor(y01).reshape(len(y)).long()
y_ = Tensor(y).reshape(len(y), 1)

for x, y_target in zip(X, y):
    if y_target == 1:
        plt.plot(x[0], x[1], 'bo')
    else:
        plt.plot(x[0], x[1], 'go')
plt.plot([-1, 1], [1, -1], '--', color='black')
plt.show()
```




```
# set up QNN
qnn1 = TwoLayerQNN(num_qubits=num_inputs, quantum_instance=qi)

# set up PyTorch module
initial_weights = 0.1*(2*np.random.rand(qnn1.num_weights) - 1)
model1 = TorchConnector(qnn1, initial_weights=initial_weights)
```

```
# test with a single input
model1(X_[0, :])
```

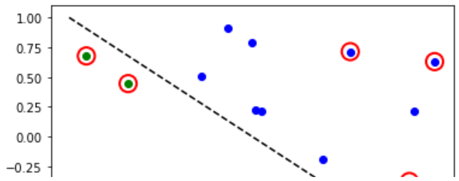
```
tensor([0.0052], grad_fn=<_TorchNNFunctionBackward>)
```

```
# define optimizer and loss
optimizer = LBFGS(model1.parameters())
f_loss = MSELoss(reduction='sum')

# start training
model1.train()   # set model to training mode

# define objective function
def closure():
    optimizer.zero_grad()          # initialize gradient
    loss = f_loss(model1(X_), y_)  # evaluate loss function
    loss.backward()                # backward pass
    print(loss.item())             # print loss
    return loss

# run optimizer
optimizer.step(closure)
```

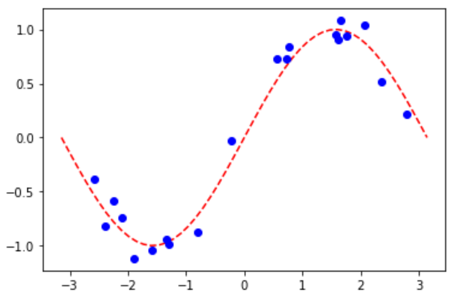
```
18.377609252929688
17.415512084960938
16.71610450744629
16.455629348754883
16.44353485107422
16.303260803222656
16.29787254333496
16.297473907470703
16.297313690185547
16.29725456237793
16.29725456237793
tensor(18.3776, grad_fn=<MseLossBackward>)
```

```
# evaluate model and compute accuracy
# The red circles indicate wrongly classified data points.
y_predict = []
for x, y_target in zip(X, y):
    output = model1(Tensor(x))
    y_predict += [np.sign(output.detach().numpy())[0]]

print('Accuracy:', sum(y_predict == y)/len(y))

# plot results
# red == wrongly classified
for x, y_target, y_p in zip(X, y, y_predict):
    if y_target == 1:
        plt.plot(x[0], x[1], 'bo')
    else:
        plt.plot(x[0], x[1], 'go')
    if y_target != y_p:
        plt.scatter(x[0], x[1], s=200, facecolors='none', edgecolors='r', linewidths=2)
plt.plot([-1, 1], [1, -1], '--', color='black')
plt.show()
```

```
Accuracy: 0.6
```




```
num_samples = 20
eps = 0.2
lb, ub = -np.pi, np.pi
f = lambda x: np.sin(x)

X = (ub - lb)*np.random.rand(num_samples, 1) + lb
y = f(X) + eps*(2*np.random.rand(num_samples, 1)-1)
plt.plot(np.linspace(lb, ub), f(np.linspace(lb, ub)), 'r--')
plt.plot(X, y, 'bo')
plt.show()
```




```
# construct simple feature map
param_x = Parameter('x')
feature_map = QuantumCircuit(1, name='fm')
feature_map.ry(param_x, 0)

# construct simple feature map
param_y = Parameter('y')
ansatz = QuantumCircuit(1, name='vf')
ansatz.ry(param_y, 0)

# construct QNN
qnn3 = TwoLayerQNN(1, feature_map, ansatz, quantum_instance=qi)
print(qnn3.operator)

# set up PyTorch module
initial_weights = 0.1*(2*np.random.rand(qnn3.num_weights) - 1)
model3 = TorchConnector(qnn3, initial_weights)
```

```
ComposedOp([
  OperatorMeasurement(1.0 * Z),
  CircuitStateFn(
       ┌───────┐┌───────┐
  q_0: ┤ fm(x) ├┤ vf(y) ├
       └───────┘└───────┘
  )
])
```

```
# define optimizer and loss function
optimizer = LBFGS(model3.parameters())
f_loss = MSELoss(reduction='sum')

# start training
model3.train()   # set model to training mode

# define objective function
def closure():
    optimizer.zero_grad(set_to_none=True)        # initialize gradient
    loss = f_loss(model3(Tensor(X)), Tensor(y))  # compute batch loss
    loss.backward()                              # backward pass
    print(loss.item())                           # print loss
    return loss

# run optimizer
optimizer.step(closure)
```

```
19.104398727416992
2.496434211730957
0.37157225608825684
0.31468555331230164
0.31463682651519775
```

```
0.31463685631752014
tensor(19.1044, grad_fn=<MseLossBackward>)
```

```python
# plot target function
plt.plot(np.linspace(lb, ub), f(np.linspace(lb, ub)), 'r--')

# plot data
plt.plot(X, y, 'bo')

# plot fitted line
y_ = []
for x in np.linspace(lb, ub):
    output = model3(Tensor([x]))
    y_ += [output.detach().numpy()[0]]
plt.plot(np.linspace(lb, ub), y_, 'g-')
plt.show()
```