Hybrid Quantum-Classical Neural Networks with PyTorch and Qiskit

Sunny Gao
高嘉陽

高嘉陽

1. 成大物理系畢業(專題教授:梁永成)
2. 台大應物所畢業(指導教授:管希聖)
3. 研究領域:量子計算、機器學習
4. 準備讀PhD，歡迎分享留學經驗

Outline

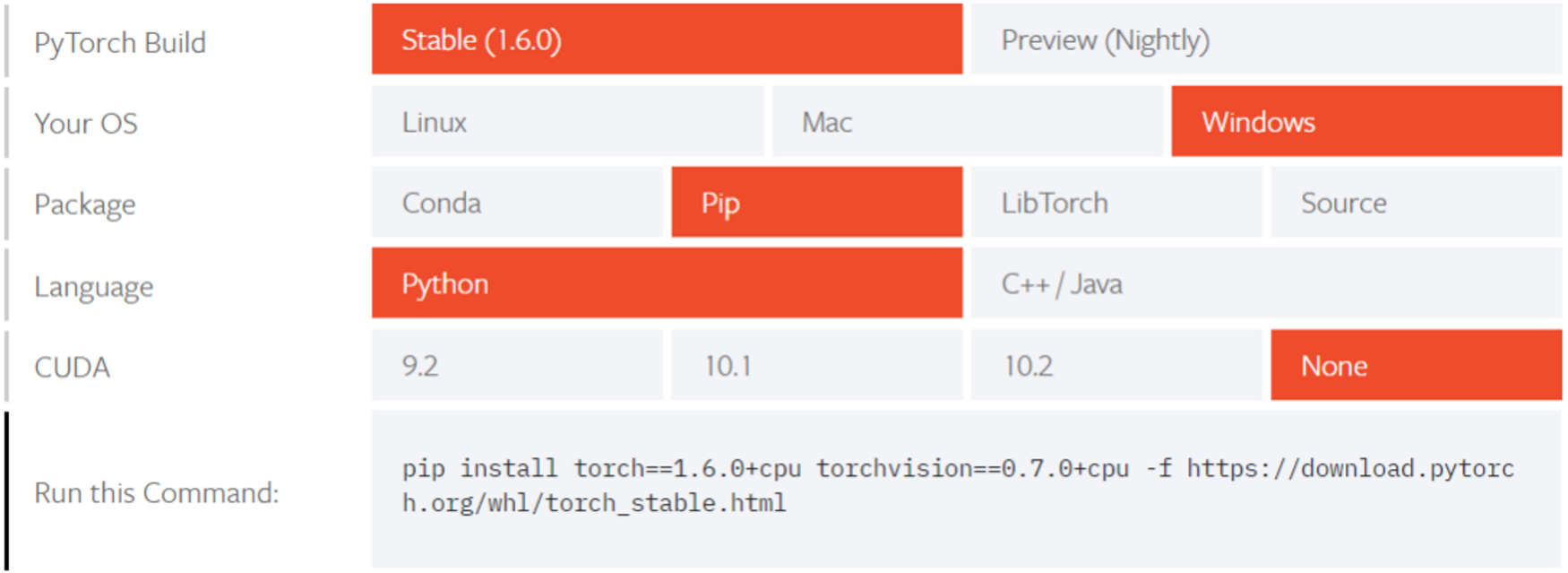
1. Install Pytorch
2. Basic concepts of Neural Network and Variational Quantum Circuit
3. Demo

Install Pytorch

1. https://pytorch.org/get-started/locally/

2. google colab (already installed)

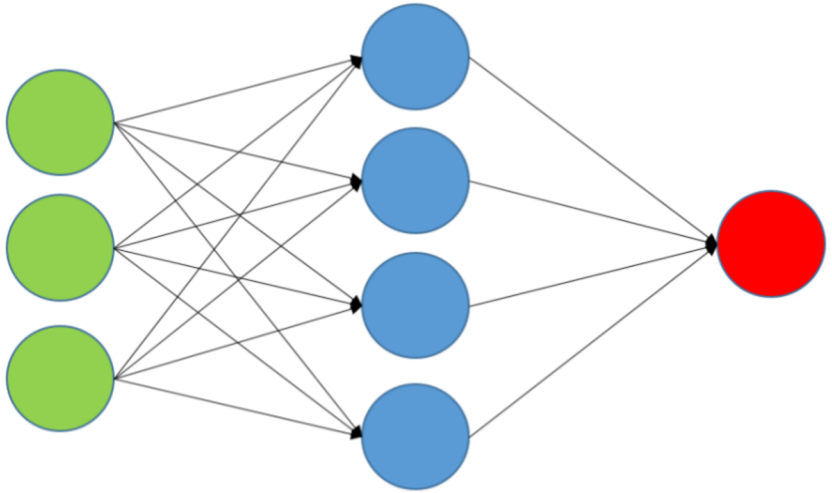
Install Pytorch



Feed Forward Neural Network

Neural network:

1. Linear term:wx+b



2. Nonlinear activation function: z=tanh(wx+b)

VQC as ML model

PHYSICAL REVIEW A **98**, 032309 (2018)

Quantum circuit learning

K. Mitarai,[1,*] M. Negoro,[1,2] M. Kitagawa,[1,3] and K. Fujii[2,4,†]

[1]*Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan*

[2]*JST, PRESTO, 4-1-8 Honcho, Kawaguchi, Saitama 332-0012, Japan*

[3]*Quantum Information and Quantum Biology Division, Institute for Open and Transdisciplinary Research Initiatives, Osaka University, Osaka 560-8531, Japan*

[4]*Graduate School of Science, Kyoto University, Yoshida-Ushinomiya-cho, Sakyo-ku, Kyoto 606-8302, Japan*

(Received 12 January 2018; published 10 September 2018)

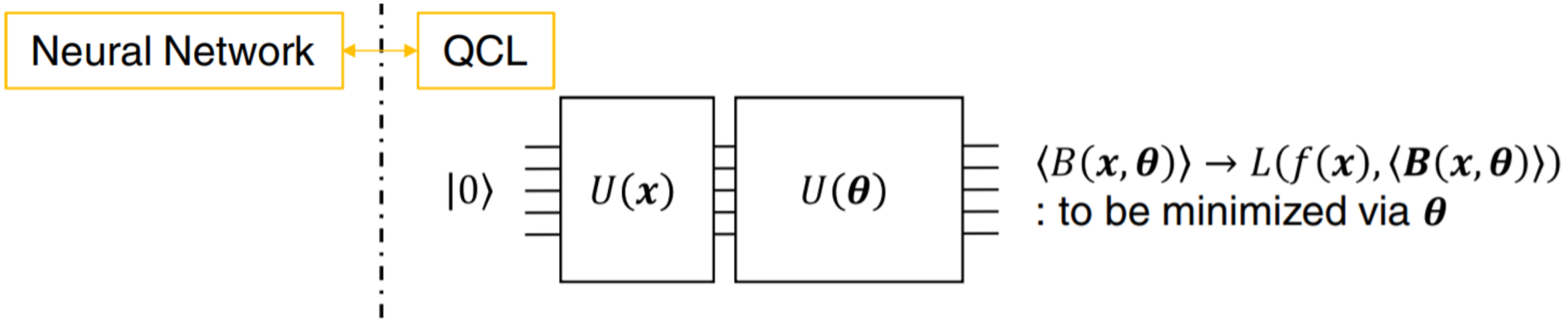
We propose a classical-quantum hybrid algorithm for machine learning on near-term quantum processors, which we call quantum circuit learning. A quantum circuit driven by our framework learns a given task by tuning parameters implemented on it. The iterative optimization of the parameters allows us to circumvent the high-depth circuit. Theoretical investigation shows that a quantum circuit can approximate nonlinear functions, which is further confirmed by numerical simulations. Hybridizing a low-depth quantum circuit and a classical computer for machine learning, the proposed framework paves the way toward applications of near-term quantum devices for quantum machine learning.

DOI: 10.1103/PhysRevA.98.032309

Variational Quantum Circuit

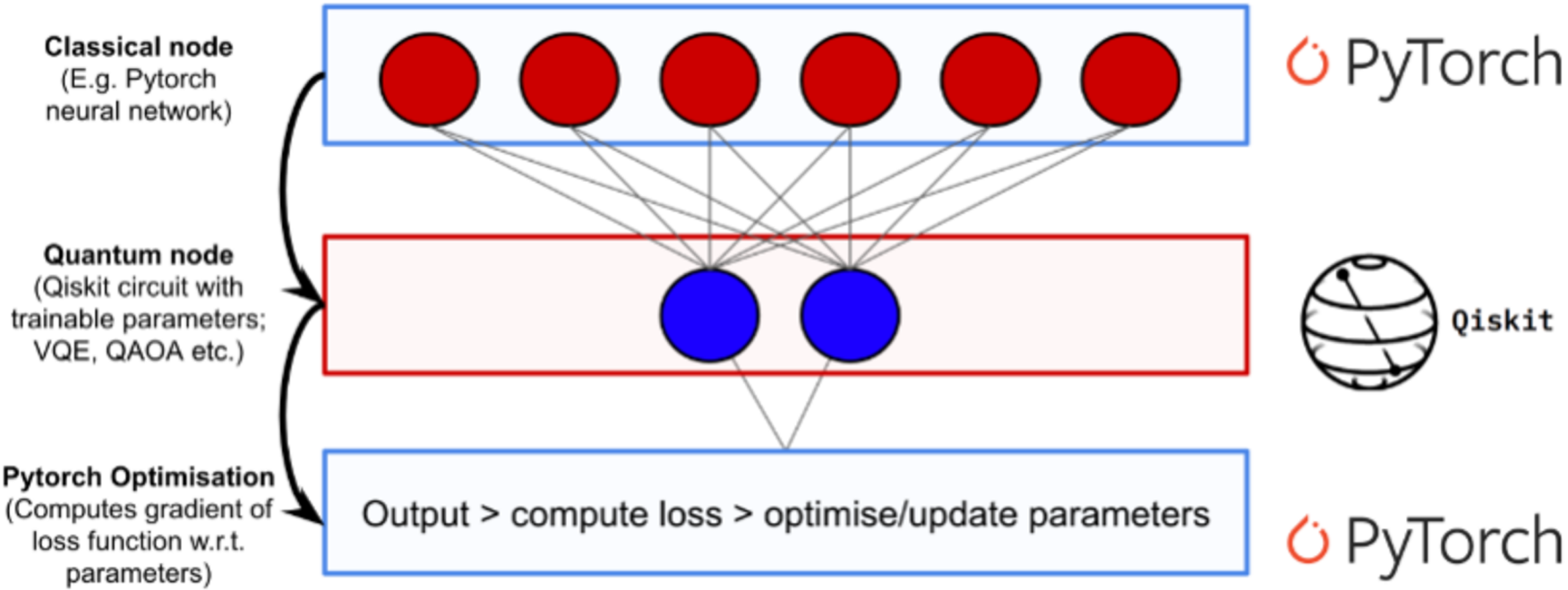
Input: x

Output: expectation value of chosen observable



https://journals.aps.org/pra/abstract/10.1103/PhysRevA.98.032309

Architecture today



https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

Let’s code

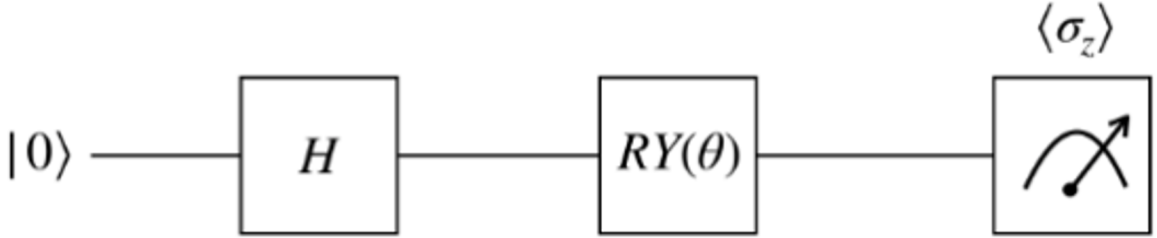
3.1 Imports

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

import torch
from torch.autograd import Function
from torchvision import datasets, transforms
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F

import qiskit
from qiskit.visualization import *
```

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

3.2 Create a "Quantum Class" with Qiskit

1-qubit circuit with one trainable quantum parameter



Quantum circuit

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

```
class QuantumCircuit:
    """
    This class provides a simple interface for interaction
    with the quantum circuit
    """

    def __init__(self, n_qubits, backend, shots):
        # --- Circuit definition ---
        self._circuit = qiskit.QuantumCircuit(n_qubits)

        all_qubits = [i for i in range(n_qubits)]
        self.theta = qiskit.circuit.Parameter('theta')

        self._circuit.h(all_qubits)
        self._circuit.barrier()
        self._circuit.ry(self.theta, all_qubits)

        self._circuit.measure_all()
        # ---------------------------

        self.backend = backend
        self.shots = shots

    def run(self, thetas):
        job = qiskit.execute(self._circuit,
                             self.backend,
                             shots = self.shots,
                             parameter_binds = [{self.theta: theta} for theta in thetas])
        result = job.result().get_counts(self._circuit)
```

Let's test the implementation

```
simulator = qiskit.Aer.get_backend('qasm_simulator')

circuit = QuantumCircuit(1, simulator, 100)
print('Expected value for rotation pi {}'.format(circuit.run([np.pi])[0]))
circuit._circuit.draw()
```

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

3.3 Create a "Quantum-Classical Class" with PyTorch

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

```python
class HybridFunction(Function):
    """ Hybrid quantum - classical function definition """

    @staticmethod
    def forward(ctx, input, quantum_circuit, shift):
        """ Forward pass computation """
        ctx.shift = shift
        ctx.quantum_circuit = quantum_circuit

        expectation_z = ctx.quantum_circuit.run(input[0].tolist())
        result = torch.tensor([expectation_z])
        ctx.save_for_backward(input, result)

        return result

    @staticmethod
    def backward(ctx, grad_output):
        """ Backward pass computation """
        input, expectation_z = ctx.saved_tensors
        input_list = np.array(input.tolist())

        shift_right = input_list + np.ones(input_list.shape) * ctx.shift
        shift_left = input_list - np.ones(input_list.shape) * ctx.shift

        gradients = []
        for i in range(len(input_list)):
            expectation_right = ctx.quantum_circuit.run(shift_right[i])
            expectation_left  = ctx.quantum_circuit.run(shift_left[i])

            gradient = torch.tensor([expectation_right]) - torch.tensor([expectation_left])
            gradients.append(gradient)
        gradients = np.array([gradients]).T
        return torch.tensor([gradients]).float() * grad_output.float(), None, None
```

Parameter shift rule

$$\frac{\partial \langle M \rangle}{\partial \theta_j} = \frac{\langle M \rangle_{\theta + \frac{\pi}{2} e_j} - \langle M \rangle_{\theta - \frac{\pi}{2} e_j}}{2}$$

3.4 Data Loading and Preprocessing

1. Hybrid neural network to classify 0 or 1 from the MNIST dataset.

1. load MNIST and filter for pictures containing 0's and 1's.

Download MNIST dataset

Training data

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

```
# Concentrating on the first 100 samples
n_samples = 100

X_train = datasets.MNIST(root='./data', train=True, download=True,
                         transform=transforms.Compose([transforms.ToTensor()]))

# Leaving only labels 0 and 1
idx = np.append(np.where(X_train.targets == 0)[0][:n_samples],
                np.where(X_train.targets == 1)[0][:n_samples])

X_train.data = X_train.data[idx]
X_train.targets = X_train.targets[idx]

train_loader = torch.utils.data.DataLoader(X_train, batch_size=1, shuffle=True)
```

try

Show the data

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

```
n_samples_show = 6

data_iter = iter(train_loader)
fig, axes = plt.subplots(nrows=1, ncols=n_samples_show, figsize=(10, 3))

while n_samples_show > 0:
    images, targets = data_iter.__next__()

    axes[n_samples_show - 1].imshow(images[0].numpy().squeeze(), cmap='gray')
    axes[n_samples_show - 1].set_xticks([])
    axes[n_samples_show - 1].set_yticks([])
    axes[n_samples_show - 1].set_title("Labeled: {}".format(targets.item()))

    n_samples_show -= 1
```

try

Pick 0 and 1

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

Testing data

```
n_samples = 50

X_test = datasets.MNIST(root='./data', train=False, download=True,
                        transform=transforms.Compose([transforms.ToTensor()]))

idx = np.append(np.where(X_test.targets == 0)[0][:n_samples],
                np.where(X_test.targets == 1)[0][:n_samples])

X_test.data = X_test.data[idx]
X_test.targets = X_test.targets[idx]

test_loader = torch.utils.data.DataLoader(X_test, batch_size=1, shuffle=True)
```

try

3.5 Creating the Hybrid Neural Network

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=5)
        self.dropout = nn.Dropout2d()
        self.fc1 = nn.Linear(256, 64)
        self.fc2 = nn.Linear(64, 1)
        self.hybrid = Hybrid(qiskit.Aer.get_backend('qasm_simulator'), 100, np.pi /

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = self.dropout(x)
        x = x.view(-1, 256)
```

3.6 Training the Network

```
model = Net()
optimizer = optim.Adam(model.parameters(), lr=0.001)
loss_func = nn.NLLLoss()

epochs = 20
loss_list = []

model.train()
for epoch in range(epochs):
    total_loss = []
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        # Forward pass
        output = model(data)
        # Calculating loss
        loss = loss_func(output, target)
        # Backward pass
        loss.backward()
        # Optimize the weights
        optimizer.step()

        total_loss.append(loss.item())
    loss_list.append(sum(total_loss)/len(total_loss))
    print('Training [{:.0f}%]\tLoss: {:.4f}'.format(
```

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

Plot the training graph

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

```
plt.plot(loss_list)
plt.title('Hybrid NN Training Convergence')
plt.xlabel('Training Iterations')
plt.ylabel('Neg Log Likelihood Loss')
```

3.7 Testing the Network

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

```
model.eval()
with torch.no_grad():

    correct = 0
    for batch_idx, (data, target) in enumerate(test_loader):
        output = model(data)

        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()

        loss = loss_func(output, target)
        total_loss.append(loss.item())

    print('Performance on test data:\n\tLoss: {:.4f}\n\tAccuracy: {:.1f}%'.format(
        sum(total_loss) / len(total_loss),
        correct / len(test_loader) * 100)
        )
```

Show the predictions

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

```
n_samples_show = 6
count = 0
fig, axes = plt.subplots(nrows=1, ncols=n_samples_show, figsize=(10, 3))

model.eval()
with torch.no_grad():
    for batch_idx, (data, target) in enumerate(test_loader):
        if count == n_samples_show:
            break
        output = model(data)

        pred = output.argmax(dim=1, keepdim=True)

        axes[count].imshow(data[0].numpy().squeeze(), cmap='gray')

        axes[count].set_xticks([])
        axes[count].set_yticks([])
        axes[count].set_title('Predicted {}'.format(pred.item()))

        count += 1
```

See your Qiskit

https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

```
import qiskit
qiskit.__qiskit_version__
```

try

Discussion

1. Any Quantum advantage?
2. My research topic
3. Need more sophisticated quantum circuit, for example, more entanglement.