Proposal for QML

The language of Quantum Computing is Linear Algebra. Since it involves linear transformations by unitary Matrices on the state vector, it is intuitive to think that these UNitary Transformations are differentiable. And since they are differentiable, it is natural that they are learnable.

Given parameters theta that defines a Unitary Transformation matrix U(theta), we can learn the theta. The UNitary Transformation U(theta) produces desired expectation values given an input state vector.

Example:

Let's look at an example from penny lane: [basic qml tutorial]

We will try and do the same task of learning params[0] and params[1] for Rx and Ry gate, respectively, to flip the given state |0> to state | one>

```
import MyQuantumSimulator
import torch
```

Lets define the circuit.

```
def circuit(params):
   qc = MyQuantumSimulator.Circuit(1)
   qc.Rx(params[0],nth_qubit=0)
   qc.Ry(params[1],nth_qubit=0)
   return qc.expected_value_Z()
```

And the cost

```
def cost(params):
   return circuit(params)
```

Initializing the parameters

```
params = torch.tensor([[0.011], [0.012]],
requires_grad=True, device=device,
dtype=torch.cfloat)
print("cost before training",cost(params))
print("###############################")
```

We use an Adam optimizer with learning rate 0.2 and update the params 100 times using the calculated gradients

```
optimizer = torch.optim.Adagrad([params], lr=0.2)
steps = 100
for i in range(steps):
   # update the circuit parameters
   loss = cost(params)
   loss.backward()
   optimizer.step()
   optimizer.zero_grad()
   if (i + 1) % 5 == 0:
       print("Cost after step {:5d}: {:.7f}".format(i + 1, cost(params)[0][0]))
```

Finally we print the rotation parameters learnt to transform |0> to |1> state using Rx and Ry gate.

```
print("################################################")
print("Optimized rotation angles: {}".format(params))
```

Output:

```
cost before training tensor([[1.0000+0.j]], device='cuda:0', grad_fn=<MmBackward0>)
################################################
Cost after step     5: -0.1311301+0.0000000j
Cost after step    10: -0.7951872+0.0000000j
Cost after step    15: -0.9560710+0.0000000j
Cost after step    20: -0.9907501+0.0000000j
Cost after step    25: -0.9980597+0.0000000j
Cost after step    30: -0.9995933+0.0000000j
Cost after step    35: -0.9999148+0.0000000j
Cost after step    40: -0.9999821+0.0000000j
Cost after step    45: -0.9999965+0.0000000j
Cost after step    50: -0.9999993+0.0000000j
Cost after step    55: -0.9999999+0.0000000j
Cost after step    60: -1.0000000+0.0000000j
Cost after step    65: -1.0000000+0.0000000j
Cost after step    70: -1.0000000+0.0000000j
Cost after step    75: -1.0000000+0.0000000j
Cost after step    80: -1.0000000+0.0000000j
Cost after step    85: -1.0000000+0.0000000j
Cost after step    90: -1.0000000+0.0000000j
Cost after step    95: -1.0000000+0.0000000j
```

```
Cost after step   100: -1.0000000+0.0000000j
################################################
Optimized rotation angles: tensor([[-1.5593+0.j],
        [ 1.5823+0.j]], device='cuda:0', requires_grad=True)
```

We can see that the cost before learning was 1(which is the Z expectation value of |0>). Our learning algorithm minimizes this expectation value from 1 to -1. (Note that -1 is the expectation value of |1> state)

Therefore we demonstrate that this Quantum Simulator has differentiable and Learnable Transformations