Qiskit Document Tutorials 勉強会
Machine Learning

Daiki Murata

本日はこちらの２トピックを取り上げます。

1. Quantum Neural Network

2. Neural Network Classifier & Regressor

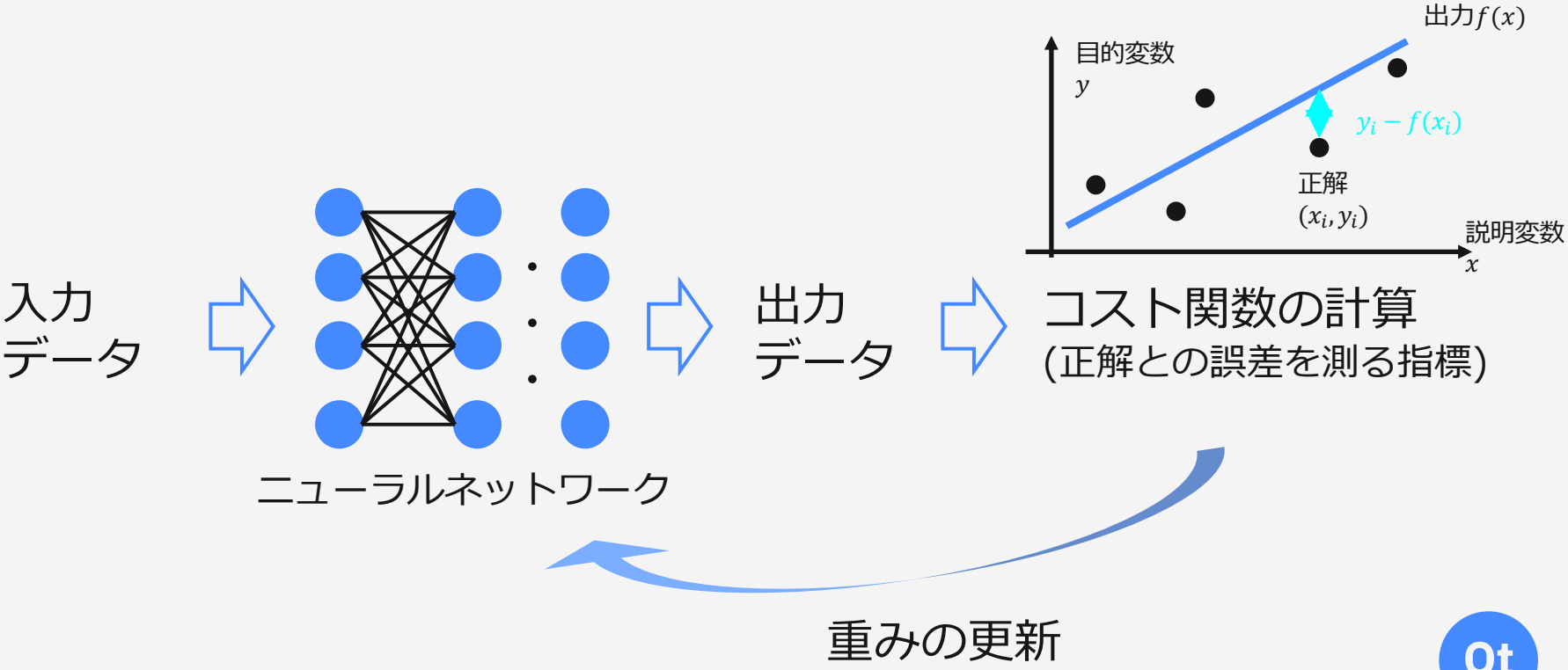
量子機械学習とは
量子計算を機械学習タスクに適用して計算を加速させようという試みです。

量子ニューラルネットワーク(QNN)は
量子回路を学習モデルに見立てて機械学習タスクを行います。

重ね合わせや量子もつれを活用して古典コンピューターで表現しにくい非線形モデルを効率的に構築することが期待されます。

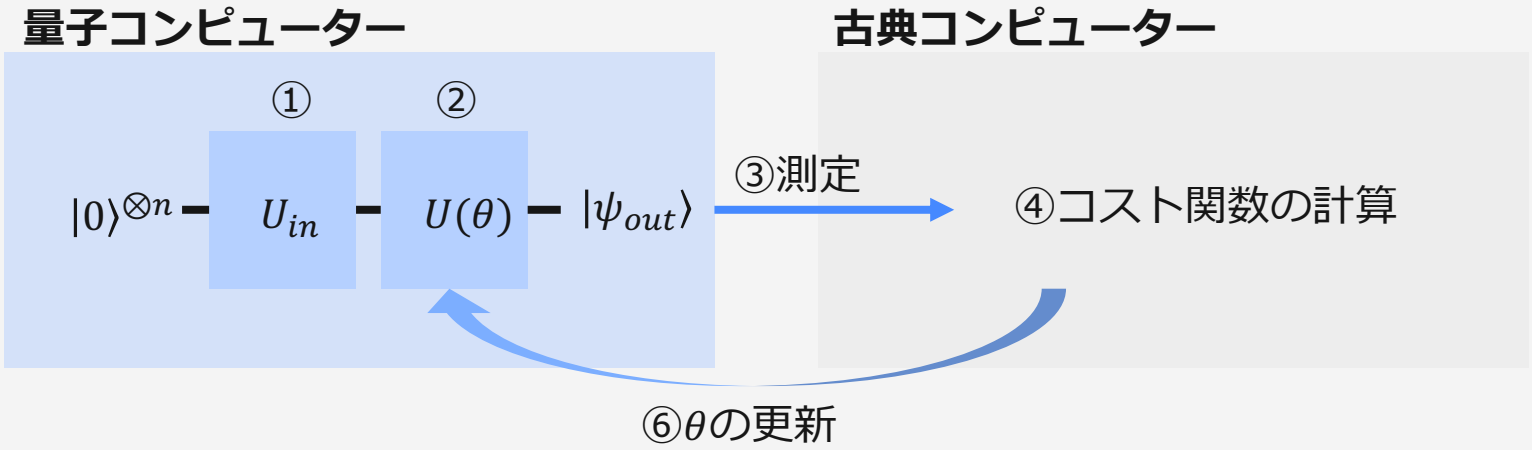
ニューラルネットワークの学習

一般的にパラメータ付き線形変換と活性化関数(非線形) を使って
パラメータ最適化によりモデルを構築します。



量子ニューラルネットワーク(QNN)の学習

QNNではパラメータ付き量子回路を用いて学習させます。

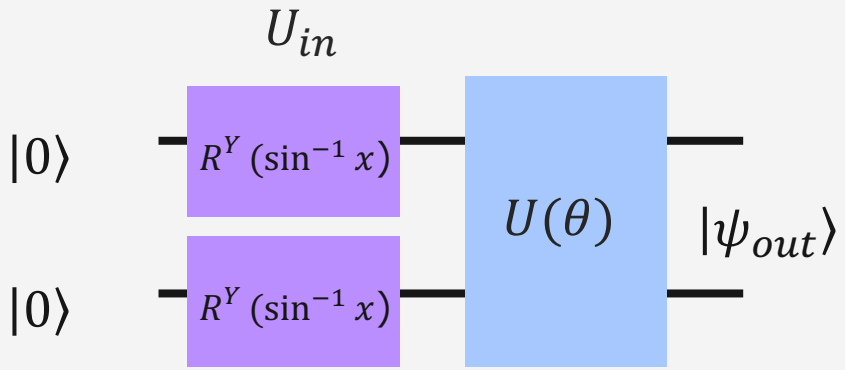


①入力値$\boldsymbol{x}$をパラメータに持つ量子回路U_{in}を通して量子状態にエンコード
②パラメータθに持つ変分回路を通して出力状態を生成
③量子ビットを測定して出力を求める
④コスト関数を計算する
⑤コスト関数が小さくなるようにθを更新して最適化θ^*を求める

QNNのいいところ

量子回路で表現する利点の一つは、複数量子ビットのテンソル積構造により指数関数的に多くの基底関数を利用できる点です。

シンプルな例を考えます



$$R^Y(\sin^{-1}x) = \sqrt{1-x^2}I - ixY$$

入力状態の密度演算子

$$\rho_{in}(x) = |\psi_{in}(x)\rangle\langle\psi_{in}(x)|$$

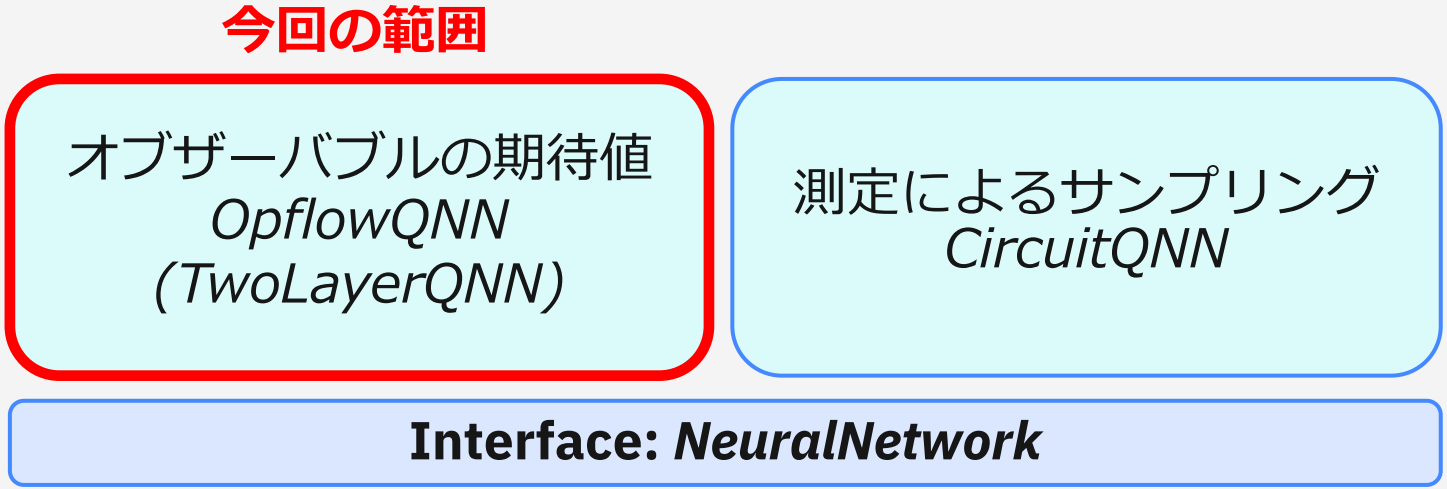
$$= \left[\frac{I + xX + \sqrt{1-x^2}Z}{2}\right]^{\otimes 2}$$

X_1X_2の期待値を計算するとx^2

入力データを非線形関数で変換してエンコーディングが可能

QiskitでQNNを実装する

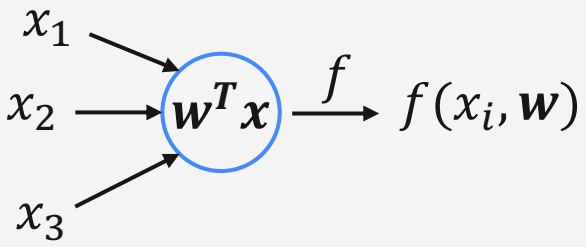
QNNの構築の方法は大きく分けると２種類に分けることが出来ます。



前提知識：順伝播と逆伝播計算

古典ニューラルネットワークでは学習にあたって順伝播計算と
誤差逆伝播計算を行う必要があります。

- 順伝播計算
  入力から出力を求める



- 逆伝播計算
  重みパラメータ更新のために必要なコスト関数の勾配を求める

$$L(\boldsymbol{w}) = \frac{1}{2}\sum \left(y_i - f(x_i, \boldsymbol{w})\right)^2$$

コスト関数の計算
(例：二乗誤差)

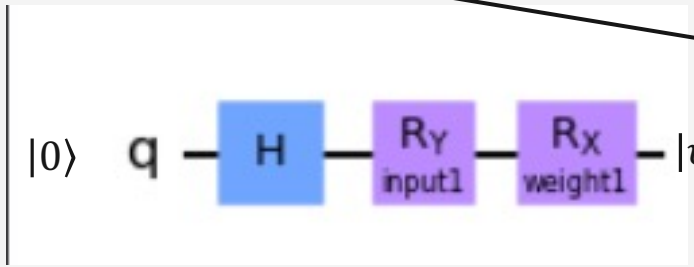
$$w' = w - \epsilon \nabla_w E(\boldsymbol{w})$$

重みの更新

QNNの順伝播計算

入力から出力(指定した演算子の期待値)を求める

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



$\text{op1}=\langle\psi_{out}|Z+X|\psi_{out}\rangle$

QNNの順伝播計算method

```
qnn1.forward(input1, weight1)
```

✓ 0.1s

```
array([[0.74378101]])
```

期待値$\langle\psi_{out}|O|\psi_{out}\rangle$を計算して検証

```
from qiskit.opflow.converters import CircuitSampler
param_dict = dict(zip(params1, [input1[0], weight1[0]]))
op1_binded = op1.assign_parameters(param_dict)
expectation = expval.convert(op1_binded)
sampler = CircuitSampler(qi_qasm).convert(expectation)
sampler.eval().real
```

✓ 0.1s

```
0.7437810137213214
```


Qt

QNNの逆伝播計算

重みパラメータ更新のために必要なコスト関数の勾配を求める

$$x' = x - \epsilon \nabla_x E(\boldsymbol{w})$$

QNNの逆伝播計算method

```
qnn1.backward(input1, weight1)
✓ 0.1s
(array([[[-0.90556062]]]), array([[[0.26008885]]]))
```

$$\frac{\partial \langle \psi_{out}|O|\psi_{out}\rangle}{\partial x_{input}}$$

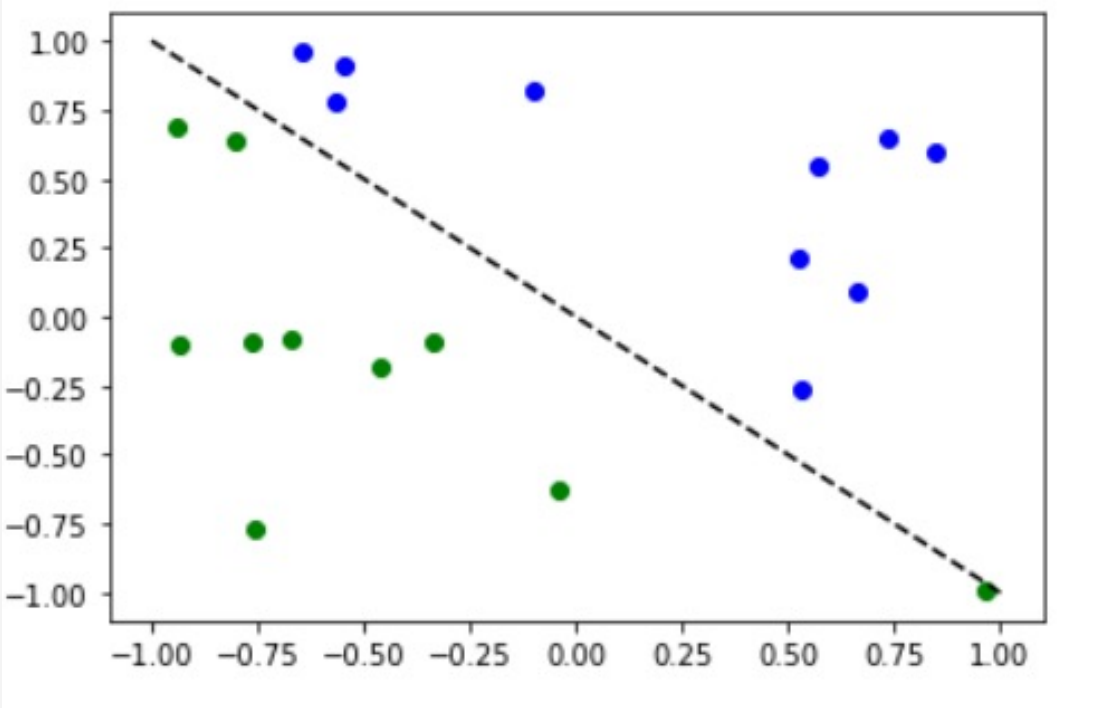
$$\frac{\partial \langle \psi_{out}|O|\psi_{out}\rangle}{\partial x_{weight}}$$

期待値$\langle \psi_{out}|O|\psi_{out}\rangle$のパラメータ微分をパラメータシフト法で計算して検証

```
state_grad = Gradient(grad_method="param_shift").convert(operator=op1, params=params1)
result = state_grad.assign_parameters(param_dict).eval()
np.array(result).real
✓ 0.1s
array([-0.90556062,  0.26008885])
```

QNNを使った分類タスク

ランダムに生成した20サンプルを$y = -x$を境界に二値分類してみます



Qt

QNNを使った分類タスク

例として*TwoLayerQNN*を使って分類タスクを解いていきます。

```
opflow_qnn = TwoLayerQNN(num_inputs, quantum_instance=qi_qasm)
opflow_classifier = NeuralNetworkClassifier(opflow_qnn, optimizer=COBYLA())
```

特徴量マッピング
(古典データ→量子データ)

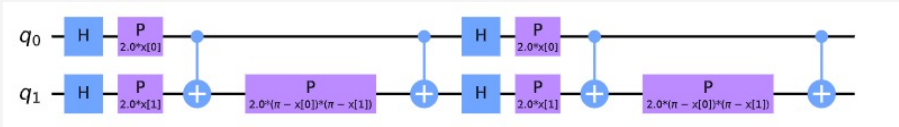
$|0\rangle^{\otimes 2}$ — U_{in} — $U(\boldsymbol{w})$ — $|\psi_{out}\rangle$

変分回路

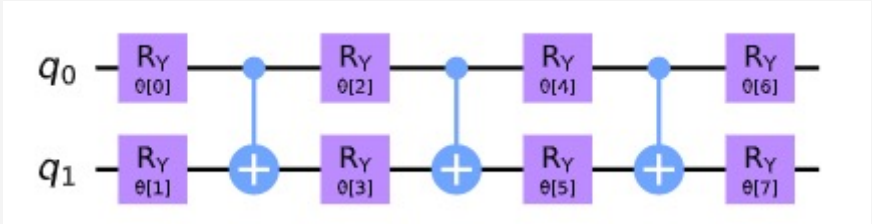
ZZ Feature Map ・・・ $U_{\Phi(\boldsymbol{x})}\,H^{\otimes 2}U_{\Phi(\boldsymbol{x})}\,H^{\otimes 2}$

$$\Phi(\boldsymbol{x}) = \exp\left[i\left(\phi_1(\boldsymbol{x})ZI + \phi_2(\boldsymbol{x})ZI + \phi_{1,2}(\boldsymbol{x})ZZ\right)\right]$$

$$\phi_S: x \to \begin{cases} x_i & (S=\{i\}) \\ (\pi - x_i)(\pi - x_j) & (S=\{i,j\}) \end{cases}$$



Real Amplitude

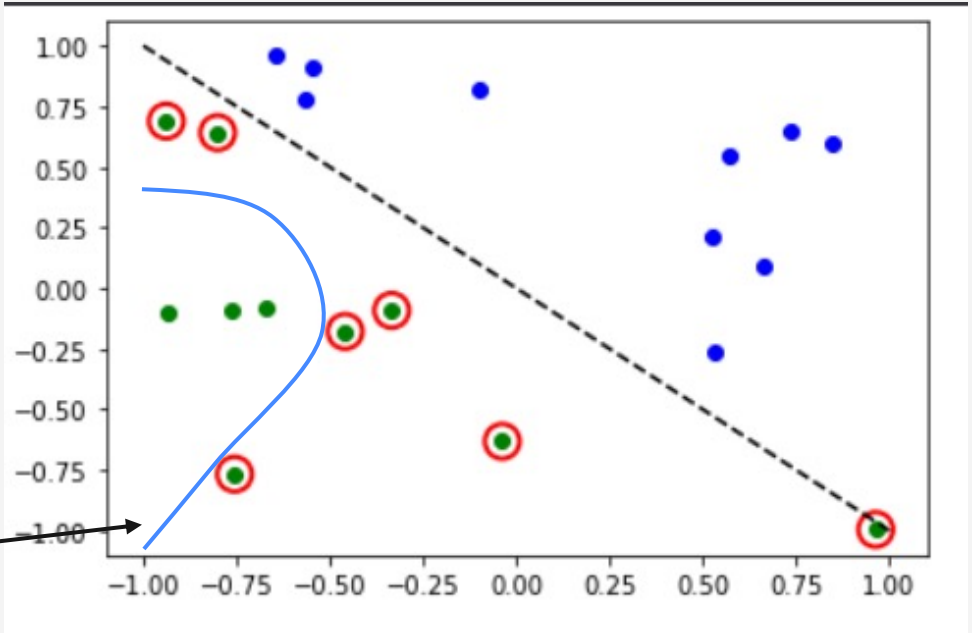


学習結果

〇は学習した結果誤分類してしまったサンプルです。ひどい結果、、

```
opflow_classifier.fit(X, y)
opflow_classifier.score(X, y)
✓ 9.6s
0.65
```

学習結果の境界イメージ



試しにFeature Mapを変えてみる

$R_y(x)$のみの非常にシンプルなFeature Mapを適用すると正解率95%になりました。