An Overview of Variational Quantum Algorithms

제 3회 양자정보과학 겨울학교

한국과학기술정보연구원(KISTI), 이동근

Variational Method

Approximation method to find the exact wave functions and energies.

A Trial Wave function: $|\Phi\rangle = \sum_i c_i |\phi_i\rangle$ ← Basis

$$\langle\Phi|\hat{H}|\Phi\rangle = \sum_{ij} c_i^* c_j \underbrace{\langle\phi_i|\hat{H}|\phi_j\rangle}_{:= H_{ij}} \qquad \langle\Phi|\Phi\rangle = \sum_{ij} c_i^* c_j \underbrace{\langle\phi_i|\phi_j\rangle}_{:= S_{ij}}$$

$$E = \frac{\langle\Phi|\hat{H}|\Phi\rangle}{\langle\Phi|\Phi\rangle}$$

Minimize E $\quad \frac{\delta E}{\delta c_i^*} = 0 \Rightarrow \frac{\sum_j c_j H_{ij} \langle\Phi|\Phi\rangle - c_j S_{ij} \langle\Phi|\hat{H}|\Phi\rangle}{\langle\Phi|\Phi\rangle^2} = 0$

varitional parameter

$$\Rightarrow \sum_j c_j H_{ij} - c_j S_{ij} E = 0 \Leftrightarrow \hat{H}|\Phi\rangle = E S |\Phi\rangle$$

$$\det(H - ES) = 0$$

Eigenvalue Prob.

Variational Method

Approximation method to find the exact wave functions and energies.

A Trial Wave function: $|\Phi\rangle = \sum_i c_i |\phi_i\rangle$

$$\langle\Phi|\hat{H}|\Phi\rangle = \sum_{ij} c_i^* c_j \langle\phi_i|\hat{H}|\phi_j\rangle \qquad \langle\Phi|\Phi\rangle = \sum_{ij} c_i^* c_j \langle\phi_i|\phi_j\rangle$$

$$E = \frac{\langle\Phi|\hat{H}|\Phi\rangle}{\langle\Phi|\Phi\rangle}$$

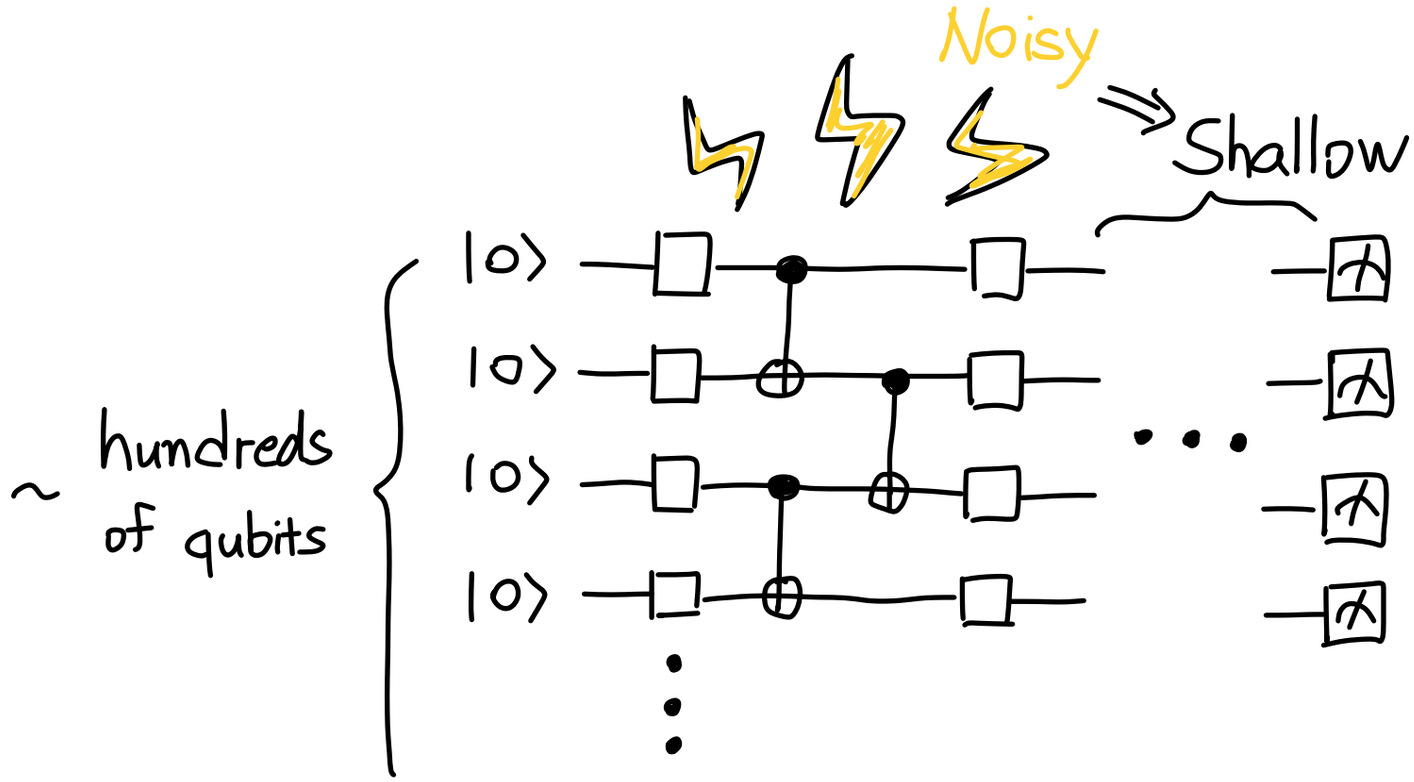
Assuming that states are normalized,

$$\hat{H}|\psi_i\rangle = E_i|\psi_i\rangle$$

$$E_0 \le E_1 \le E_2 \le \cdots$$

$$\begin{aligned}\langle\Phi|\hat{H}|\Phi\rangle &= \sum_{i,j}\langle\Phi|\psi_i\rangle\langle\psi_i|\hat{H}|\psi_j\rangle\langle\psi_j|\Phi\rangle \\ &= \sum_i E_i|\langle\psi_i|\Phi\rangle|^2 \ge \sum_i E_0|\langle\psi_i|\Phi\rangle|^2 = E_0\end{aligned}$$

Noise Intermediate-Scale Quantum (NISQ) computer?



IDEA: classical computing + quantum computing?

$$\min_{\mathbf{c}^*}\langle\Phi|\hat{H}|\Phi\rangle \Longrightarrow \min_{\boldsymbol{\theta}}\langle\Phi(\boldsymbol{\theta})|\hat{H}|\Phi(\boldsymbol{\theta})\rangle$$

Quantum part

$$C(\theta) = \langle\Phi(\theta)|\hat{H}|\Phi(\theta)\rangle$$

Classical part

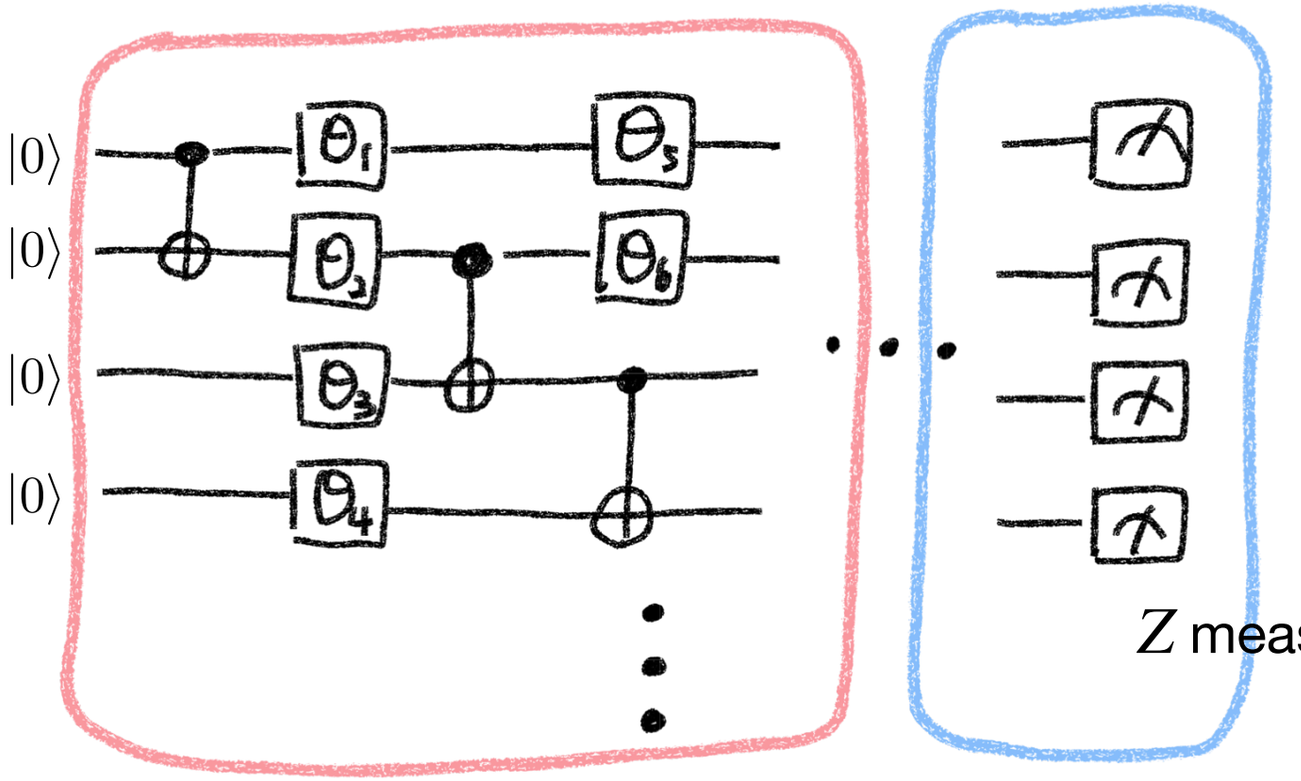
$$\theta_{new} \leftarrow \theta - \alpha\nabla C$$

Variational Quantum Eigensolver

Ansatz : $|\Phi(\theta)\rangle = U(\theta)|0\rangle$

Observable : $\hat{H} = \sum_i c_i \hat{P}_i$

$\hat{P}_i \in \{I, X, Y, Z\}^{\otimes n}$



Outcomes z, $p(z)$

$$C(\boldsymbol{\theta}) = \langle \Phi(\boldsymbol{\theta}) | \hat{H} | \Phi(\boldsymbol{\theta}) \rangle$$

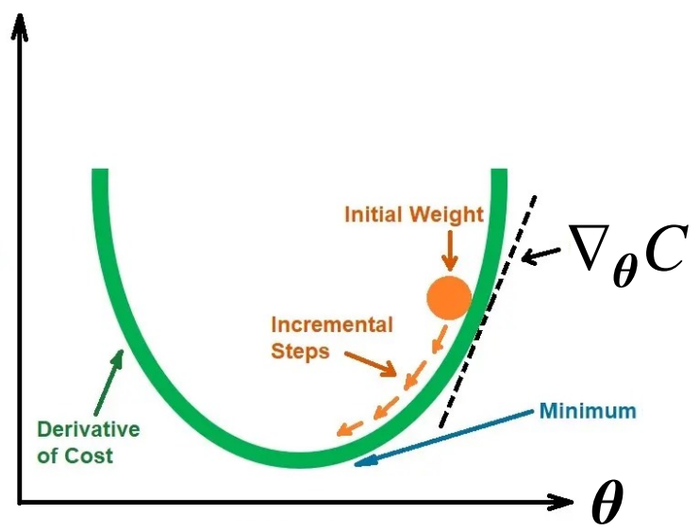
Z measurements

Gradient Descent Method

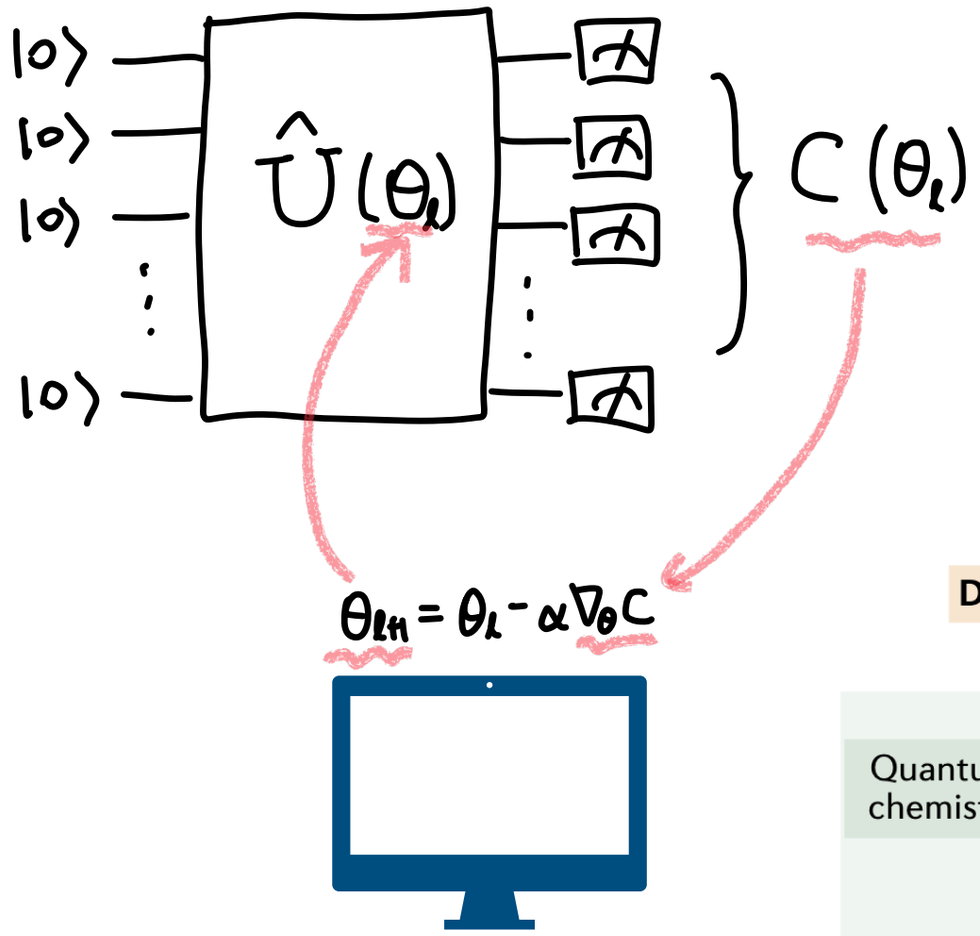
$$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} C$$

Parameter-Shift Rule

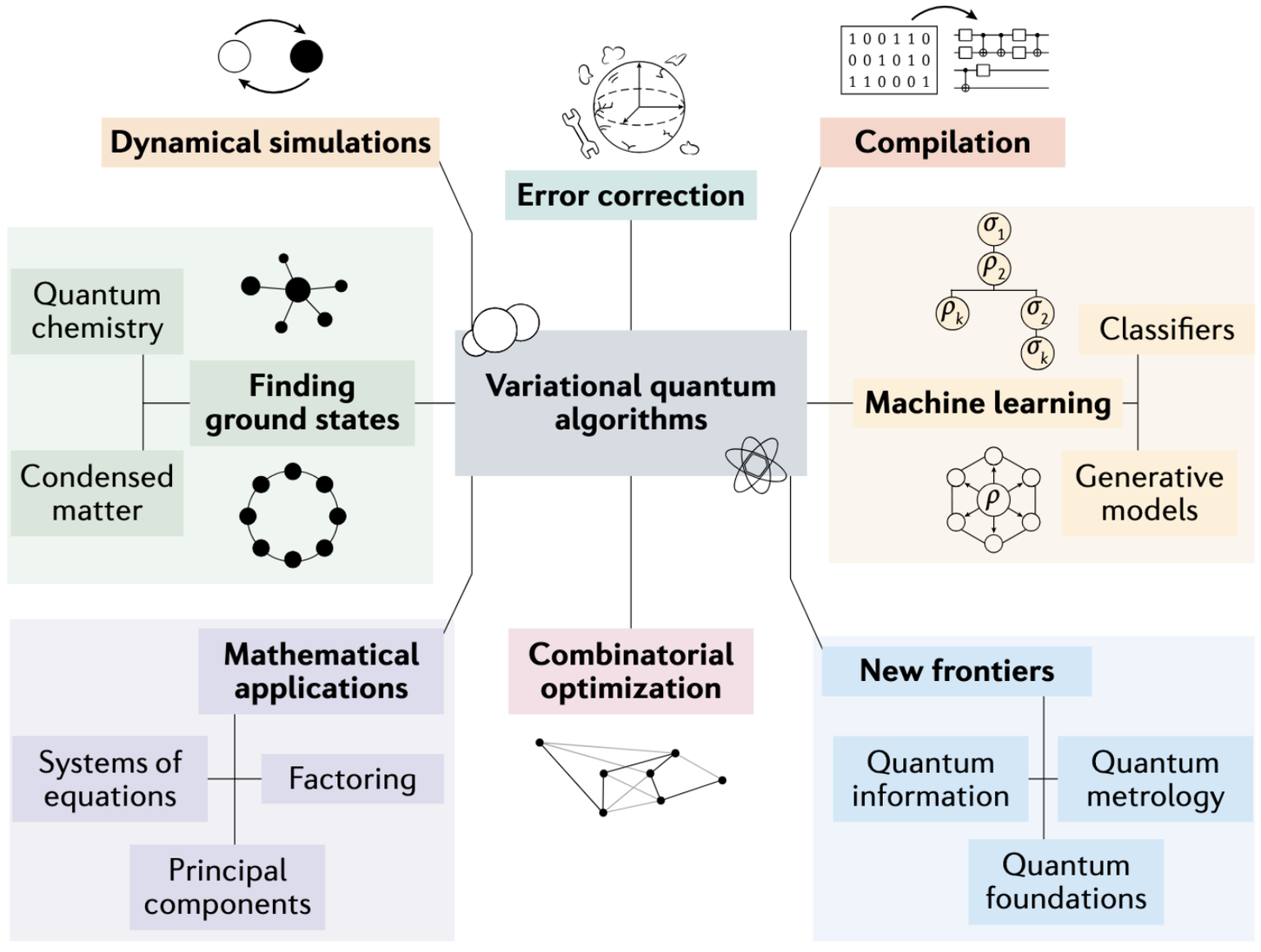
$$\nabla_{\theta_i} C(\theta_i) = c\,[C(\theta_i + s) - C(\theta_i - s)]$$



Variational Quantum Algorithms (VQA)



Applications of VQAs

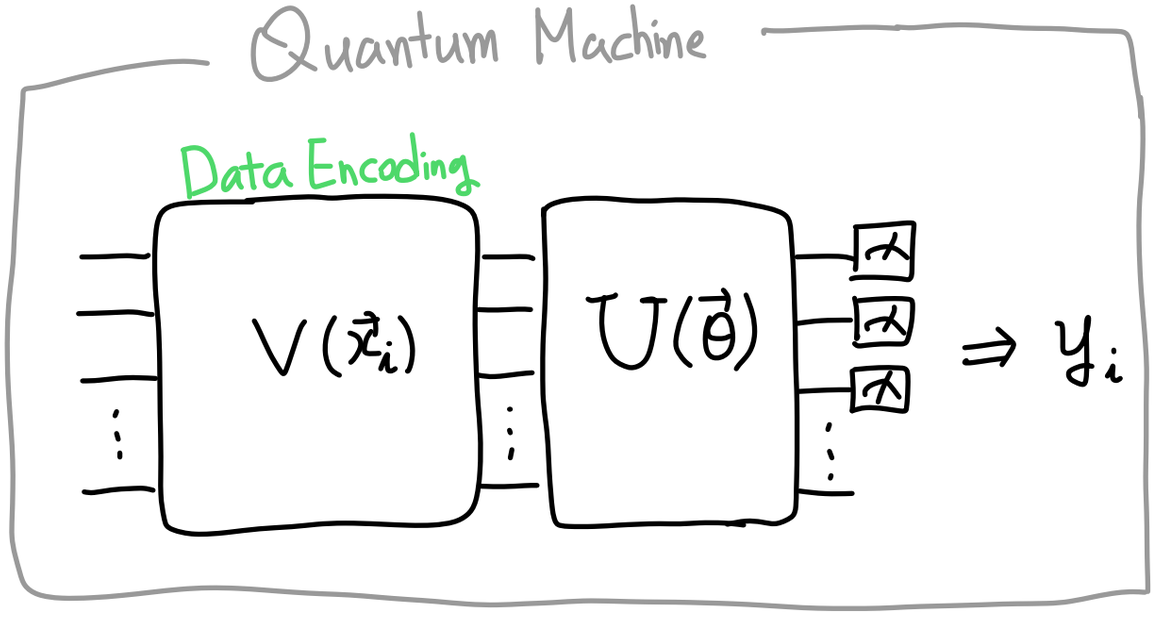


Cerezo, M. *et al*. Variational quantum algorithms. *Nat Rev Phys* **3**, 625–644 (2021).

Example: Variational Classifier

2. Iris classification

https://pennylane.ai/qml/demos/tutorial_variational_classifier/



Learning Training Dataset $D = \{\vec{x}_i, \overset{\{0,1\}}{\Downarrow} y_i\}_{1 \le i \in N}$

Aim : Test $D' = \{\vec{x}'_i\}$ & predict y'_i

Aim: Amplitude Encoding

Normalized 4-dim $\vec{x}$

↓

$x_0|00\rangle + x_1|01\rangle + x_2|10\rangle + x_3|11\rangle$

Real & Non negative

e.g.,

```
x = np.array([0.53896774, 0.79503606, 0.27826503, 0.0], requires_grad=False)
ang = get_angles(x)

dev = qml.device("default.qubit", wires=2)
@qml.qnode(dev)
def test(angles):

    statepreparation(angles)

    return qml.state()

state = test(ang)

print("x               : ", x)
print("angles          : ", ang)
print("amplitude vector: ", np.real(state))
```

Out:

```
x               :  [0.53896774 0.79503606 0.27826503 0.        ]
angles          :  [ 0.56397465 -0.          0.         -0.97504604  0.97504604]
amplitude vector:  [ 5.38967743e-01  7.95036065e-01  2.78265032e-01 -2.77555756e-17]
```

```
def get_angles(x):

    beta0 = 2 * np.arcsin(np.sqrt(x[1] ** 2) / np.sqrt(x[0] ** 2 + x[1] ** 2 + 1e-12))
    beta1 = 2 * np.arcsin(np.sqrt(x[3] ** 2) / np.sqrt(x[2] ** 2 + x[3] ** 2 + 1e-12))
    beta2 = 2 * np.arcsin(
        np.sqrt(x[2] ** 2 + x[3] ** 2)
        / np.sqrt(x[0] ** 2 + x[1] ** 2 + x[2] ** 2 + x[3] ** 2)
    )

    return np.array([beta2, -beta1 / 2, beta1 / 2, -beta0 / 2, beta0 / 2])
```

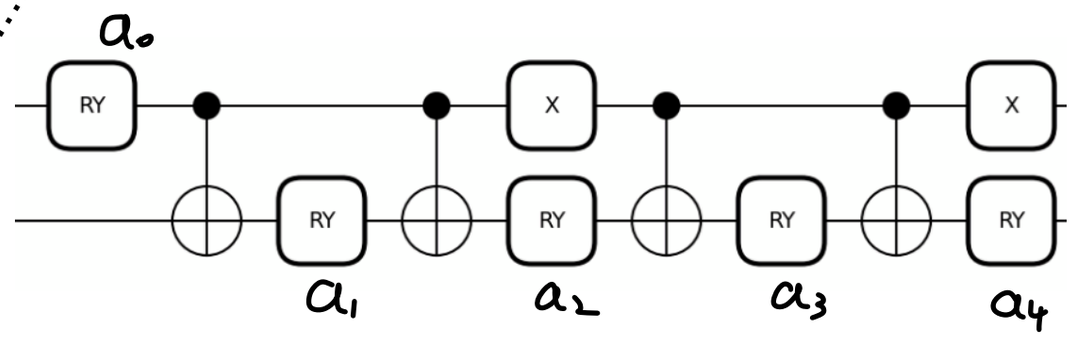
$\vec{x} = (x_0, x_1, x_2, x_3)$

$\beta_0 = 2\sin^{-1}\left(\frac{x_1^2}{\sqrt{x_0^2 + x_1^2}}\right), \quad \beta_1 = 2\sin^{-1}\left(\frac{x_3^2}{\sqrt{x_2^2 + x_3^2}}\right),$

$\beta_2 = 2\sin^{-1}\left(\frac{\sqrt{x_2^2 + x_3^2}}{|\vec{x}|}\right)$

$\vec{a} = (\beta_2, -\beta_1/2, \beta_1/2, -\beta_0/2, \beta_0/2)$

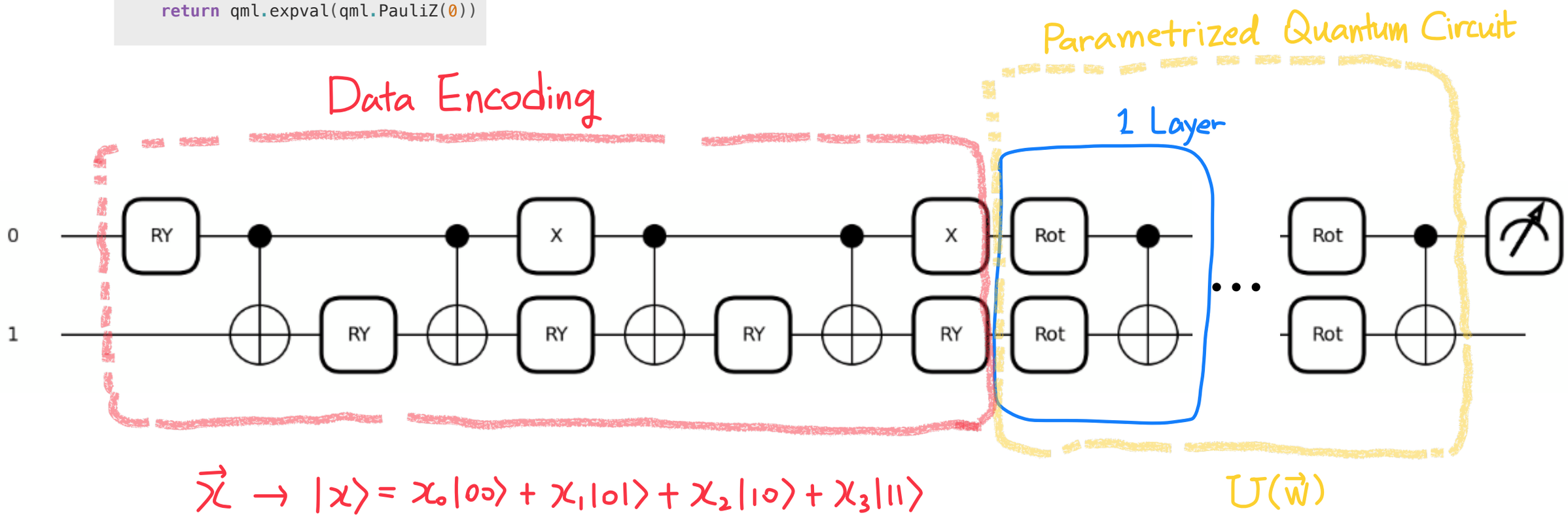
```
def statepreparation(a):
    qml.RY(a[0], wires=0)

    qml.CNOT(wires=[0, 1])
    qml.RY(a[1], wires=1)
    qml.CNOT(wires=[0, 1])
    qml.RY(a[2], wires=1)

    qml.PauliX(wires=0)
    qml.CNOT(wires=[0, 1])
    qml.RY(a[3], wires=1)
    qml.CNOT(wires=[0, 1])
    qml.RY(a[4], wires=1)
    qml.PauliX(wires=0)
```



$$R_y(\phi) = e^{-i\phi\sigma_y/2} = \begin{bmatrix} \cos(\phi/2) & -\sin(\phi/2) \\ \sin(\phi/2) & \cos(\phi/2) \end{bmatrix}.$$

```python
@qml.qnode(dev)
def circuit(weights, angles):
    statepreparation(angles)

    for W in weights:
        layer(W)

    return qml.expval(qml.PauliZ(0))
```

```python
def layer(W):
    qml.Rot(W[0, 0], W[0, 1], W[0, 2], wires=0)
    qml.Rot(W[1, 0], W[1, 1], W[1, 2], wires=1)
    qml.CNOT(wires=[0, 1])
```



After Z measurement on the first qubit,

Outcome: $z \in \{-1, 1\}$

Probability: $p(z)$

$$\sum_{z \in \{-1,1\}} z\, p(z) = \langle \hat{Z}_0 \rangle = \langle x | \hat{U}^\dagger(\mathbf{w}) \hat{Z}_0 \hat{U}(\mathbf{w}) | x \rangle$$

```python
def variational_classifier(weights, bias, angles):
    return circuit(weights, angles) + bias
```

$$\langle x | \hat{U}^\dagger(\mathbf{w}) \hat{Z}_0 \hat{U}(\mathbf{w}) | x \rangle + b$$

```python
def cost(weights, bias, features, labels):
    predictions = [variational_classifier(weights, bias, f) for f in features]
    return square_loss(labels, predictions)
```

$$C(\mathbf{w}, b) = \sum_{i=1}^{N} \left(\langle x_i | \hat{U}^\dagger(\mathbf{w}) \hat{Z}_0 \hat{U}(\mathbf{w}) | x_i \rangle + b - y_i \right)^2 / N$$

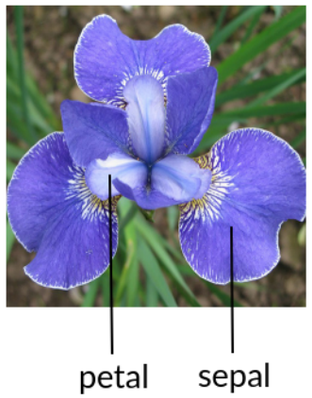
Iris Dataset

Four Attributes:

sepal length, width, petal length, width

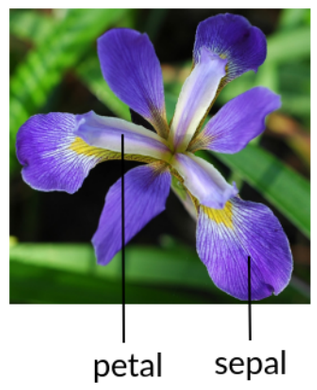
$$\mathbf{x}_n = (x_{n0}, x_{n1}, x_{n2}, x_{n3})$$

$(y_n = 0)$
iris setosa



$(y_n = 1)$
iris versicolor

$(y_n = 2)$
iris virginica



Three Classes:

$$y_n \in \{0, 1, 2\}$$

Original Data

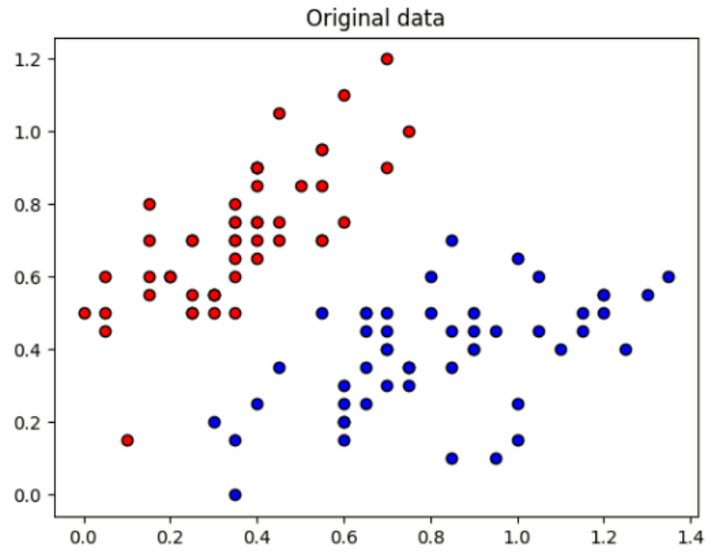
	sepal length	sepal width	petal length	petal width
$\mathbf{x}_1 \rightarrow$	5.1	3.5	1.4	0.2
	4.9	3.0	1.4	0.2
	4.7	3.2	1.3	0.2
	4.6	3.1	1.5	0.2
	5.0	3.6	1.4	0.2
	...	...	...	...
	6.7	3.0	5.2	2.3
	6.3	2.5	5.0	1.9
	6.5	3.0	5.2	2.0
		3.4	5.4	2.3
$\mathbf{x}_{150} \rightarrow$	5.9	3.0	5.1	1.8

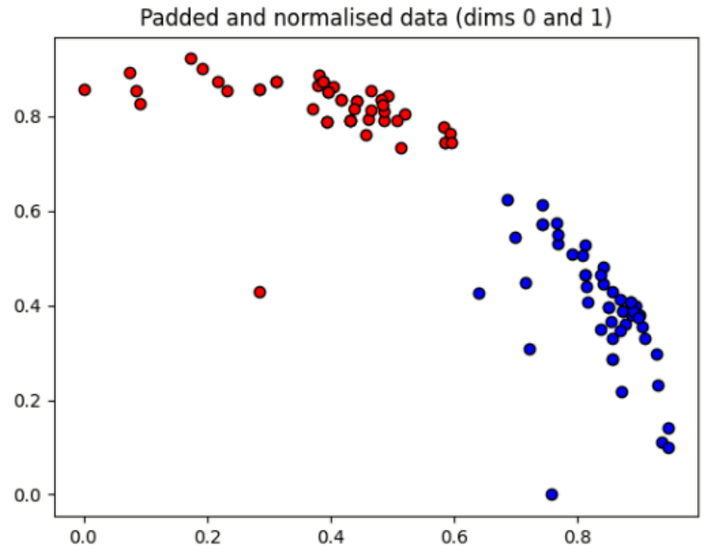
Preprocessed Data

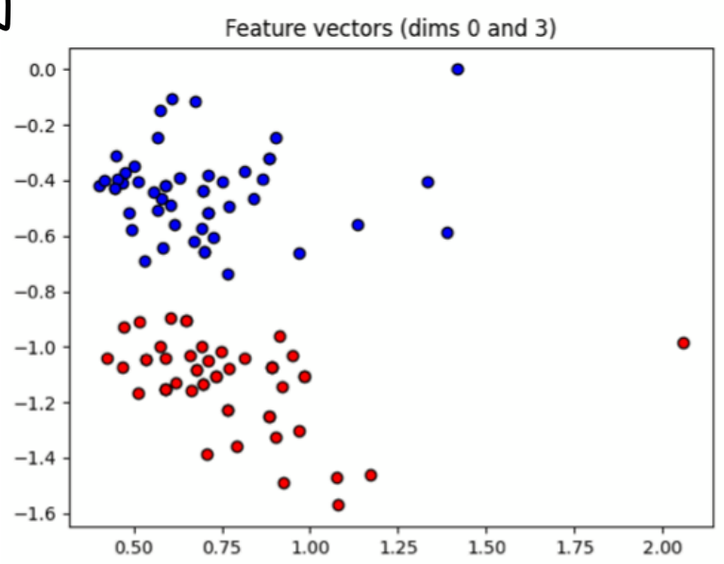
```
3.9999999999999112e-01 7.5000000000000000e-01 1.9999999999999556e-01 5.0000000000000278e-02 -1.0000000000000000e+00
3.0000000000002665e-01 5.0000000000000000e-01 1.9999999999999556e-01 5.0000000000000278e-02 -1.0000000000000000e+00
2.0000000000001776e-01 6.0000000000000888e-01 1.5000000000000222e-01 5.0000000000000278e-02 -1.0000000000000000e+00
1.4999999999999112e-01 5.5000000000000444e-01 2.5000000000000000e-01 5.0000000000000278e-02 -1.0000000000000000e+00
3.5000000000000888e-01 8.0000000000000444e-01 1.9999999999999556e-01 5.0000000000000278e-02 -1.0000000000000000e+00
5.5000000000002665e-01 9.4999999999999556e-01 3.4999999999999778e-01 1.5000000000000222e-01 -1.0000000000000000e+00
1.4999999999999112e-01 6.9999999999999556e-01 1.9999999999999556e-01 9.9999999999999167e-02 -1.0000000000000000e+00
3.5000000000000888e-01 6.9999999999999556e-01 2.5000000000000000e-01 5.0000000000000278e-02 -1.0000000000000000e+00
5.0000000000026645e-02 4.4999999999999556e-01 1.9999999999999556e-01 5.0000000000000278e-02 -1.0000000000000000e+00
3.0000000000002665e-01 5.5000000000000444e-01 2.5000000000000000e-01 0.0000000000000000e+00 -1.0000000000000000e+00
5.5000000000002665e-01 8.5000000000000888e-01 2.5000000000000000e-01 5.0000000000000278e-02 -1.0000000000000000e+00
2.5000000000000000e-01 6.9999999999999556e-01 3.0000000000000444e-01 5.0000000000000278e-02 -1.0000000000000000e+00
2.5000000000000000e-01 5.0000000000000000e-01 1.9999999999999556e-01 0.0000000000000000e+00 -1.0000000000000000e+00
0.0000000000000000e+00 5.0000000000000000e-01 5.0000000000004441e-02 0.0000000000000000e+00 -1.0000000000000000e+00
7.5000000000000000e-01 1.0000000000000000e+00 9.9999999999997780e-02 5.0000000000000278e-02 -1.0000000000000000e+00
7.0000000000001776e-01 1.2000000000000178e+00 2.5000000000000000e-01 1.5000000000000222e-01 -1.0000000000000000e+00
5.5000000000002665e-01 9.4999999999999556e-01 1.5000000000000222e-01 1.5000000000000222e-01 -1.0000000000000000e+00
3.9999999999999112e-01 7.5000000000000000e-01 1.9999999999999556e-01 9.9999999999999167e-02 -1.0000000000000000e+00
7.0000000000001776e-01 8.9999999999999112e-01 3.4999999999999778e-01 9.9999999999999167e-02 -1.0000000000000000e+00
3.9999999999999112e-01 8.9999999999999112e-01 2.5000000000000000e-01 9.9999999999999167e-02 -1.0000000000000000e+00
5.5000000000002665e-01 6.9999999999999556e-01 3.4999999999999778e-01 5.0000000000000278e-02 -1.0000000000000000e+00
3.9999999999999112e-01 8.5000000000000888e-01 2.5000000000000000e-01 1.5000000000000222e-01 -1.0000000000000000e+00
1.4999999999999112e-01 8.0000000000000444e-01 0.0000000000000000e+00 5.0000000000000278e-02 -1.0000000000000000e+00
3.9999999999999112e-01 6.4999999999999112e-01 3.4999999999999778e-01 2.0000000000000111e-01 -1.0000000000000000e+00
2.5000000000000000e-01 6.9999999999999556e-01 4.4999999999999556e-01 5.0000000000000278e-02 -1.0000000000000000e+00
3.5000000000000888e-01 5.0000000000000000e-01 3.0000000000000444e-01 5.0000000000000278e-02 -1.0000000000000000e+00
3.5000000000000888e-01 6.9999999999999556e-01 3.0000000000000444e-01 1.5000000000000222e-01 -1.0000000000000000e+00
4.5000000000001776e-01 7.5000000000000000e-01 2.5000000000000000e-01 5.0000000000000278e-02 -1.0000000000000000e+00
4.5000000000001776e-01 6.9999999999999556e-01 1.9999999999999556e-01 5.0000000000000278e-02 -1.0000000000000000e+00
2.0000000000001776e-01 6.0000000000000888e-01 3.0000000000000444e-01 5.0000000000000278e-02 -1.0000000000000000e+00
2.5000000000000000e-01 5.5000000000000444e-01 3.0000000000000444e-01 5.0000000000000278e-02 -1.0000000000000000e+00
5.5000000000002665e-01 6.9999999999999556e-01 2.5000000000000000e-01 1.5000000000000222e-01 -1.0000000000000000e+00
```

Visualization of Iris Dataset

```python
data = np.loadtxt("variational_classifier/data/iris_classes1and2_scaled.txt")
X = data[:, 0:2]

# pad the vectors to size 2^2 with constant values
padding = 0.3 * np.ones((len(X), 1))
X_pad = np.c_[np.c_[X, padding], np.zeros((len(X), 1))]

# normalize each input
normalization = np.sqrt(np.sum(X_pad ** 2, -1))
X_norm = (X_pad.T / normalization).T

# angles for state preparation are new features
features = np.array([get_angles(x) for x in X_norm], requires_grad=False)

# label
Y = data[:, -1]

print("First X sample (original)  :", X[0])
print("First X sample (padded)    :", X_pad[0])
print("First X sample (normalized):", X_norm[0])
print("First features sample      :", features[0])
```







Output:

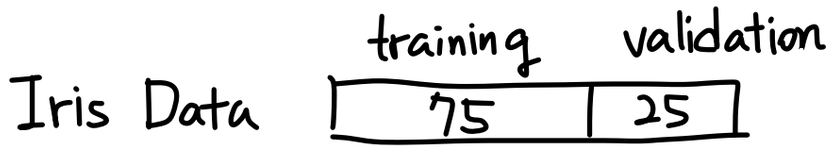
```
First X sample (original)  : [0.4  0.75]
First X sample (padded)    : [0.4  0.75 0.3  0.  ]
First X sample (normalized): [0.44376016 0.83205029 0.33282012 0.        ]
First features sample      : [ 0.67858523 -0.          0.         -1.080839    1.080839  ]
```

Implementation

```
np.random.seed(0)
num_data = len(Y)
num_train = int(0.75 * num_data)
index = np.random.permutation(range(num_data))
feats_train = features[index[:num_train]]
Y_train = Y[index[:num_train]]
feats_val = features[index[num_train:]]
Y_val = Y[index[num_train:]]

# We need these later for plotting
X_train = X[index[:num_train]]
X_val = X[index[num_train:]]
```

100

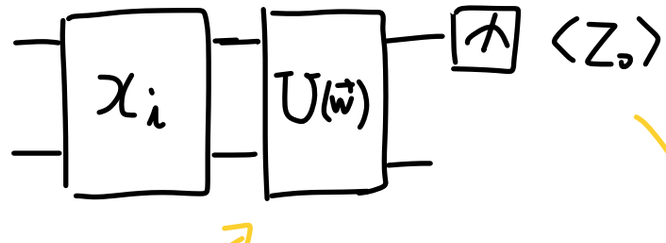
Iris Data




```
num_qubits = 2
num_layers = 6

weights_init = 0.01 * np.random.randn(num_layers, num_qubits, 3, requires_grad=True)
bias_init = np.array(0.0, requires_grad=True)
```

```
opt = NesterovMomentumOptimizer(0.01)
batch_size = 5

# train the variational classifier
weights = weights_init
bias = bias_init
for it in range(60):

    # Update the weights by one optimizer step
    batch_index = np.random.randint(0, num_train, (batch_size,))
    feats_train_batch = feats_train[batch_index]
    Y_train_batch = Y_train[batch_index]
    weights, bias, _, _ = opt.step(cost, weights, bias, feats_train_batch, Y_train_batch)

    # Compute predictions on train and validation set
    predictions_train = [np.sign(variational_classifier(weights, bias, f)) for f in feats_train]
    predictions_val = [np.sign(variational_classifier(weights, bias, f)) for f in feats_val]

    # Compute accuracy on train and validation set
    acc_train = accuracy(Y_train, predictions_train)
    acc_val = accuracy(Y_val, predictions_val)

    print(
        "Iter: {:5d} | Cost: {:0.7f} | Acc train: {:0.7f} | Acc validation: {:0.7f} "
        "".format(it + 1, cost(weights, bias, features, Y), acc_train, acc_val)
    )
```

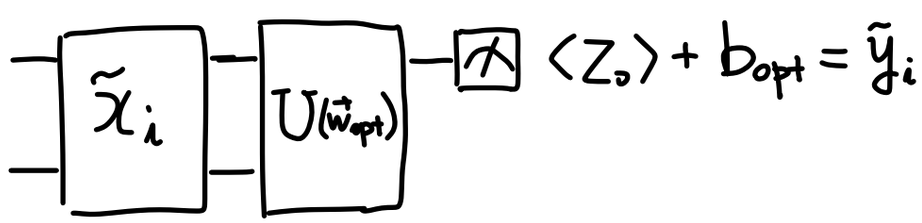
Supervised Learning $\min_{\vec{w},b} C(\vec{w},b)$

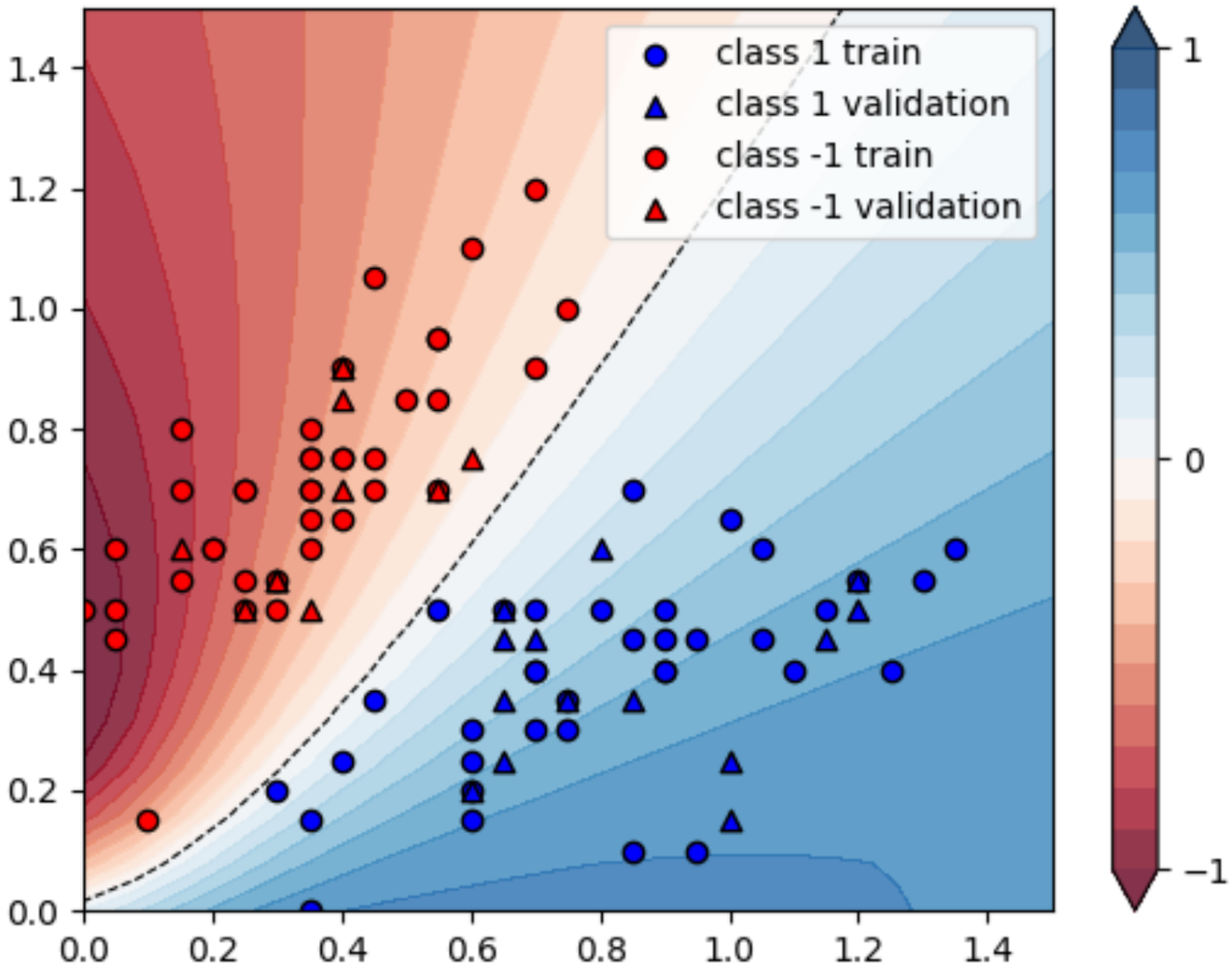


$$C(\vec{w},b) = \sum_i^N (y_i - \langle Z_0 \rangle - b)^2 / N$$

$$\vec{w}_{new} = \vec{w} - \alpha \nabla_{\vec{w}} C$$

$$b_{new} = b - \alpha \nabla_b C$$



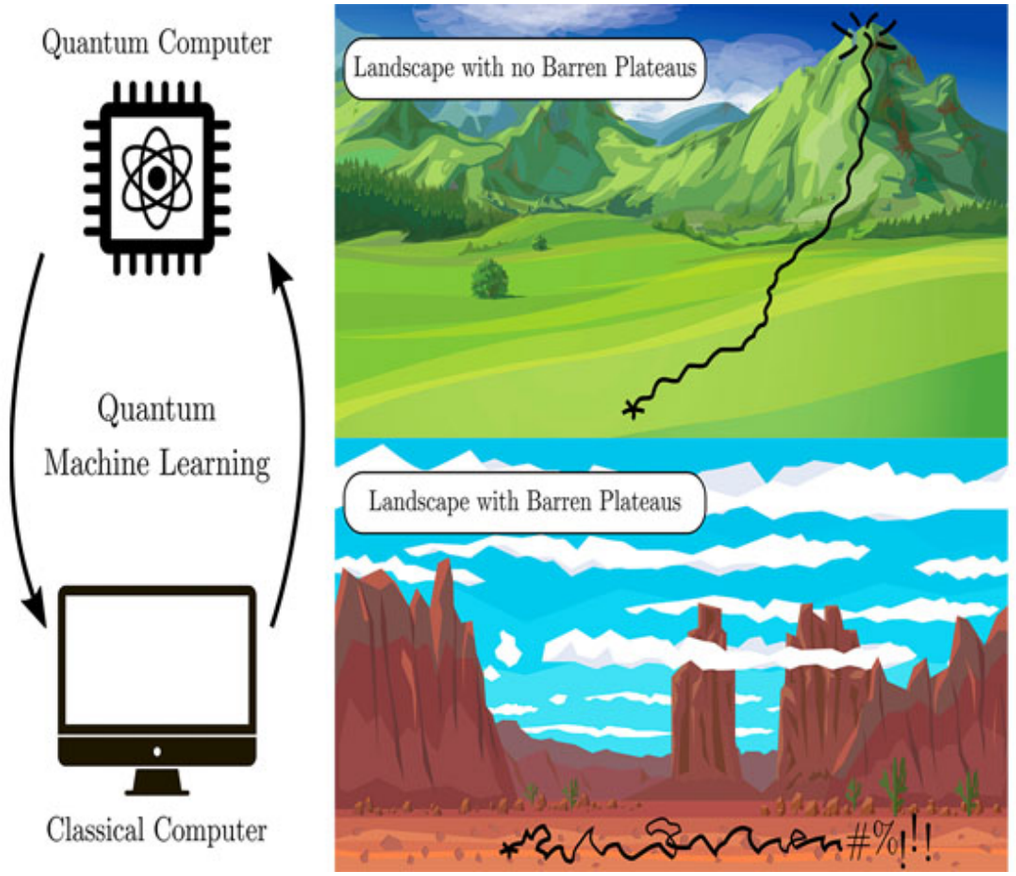

class 1 train
class 1 validation
class -1 train
class -1 validation
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1
0
−1

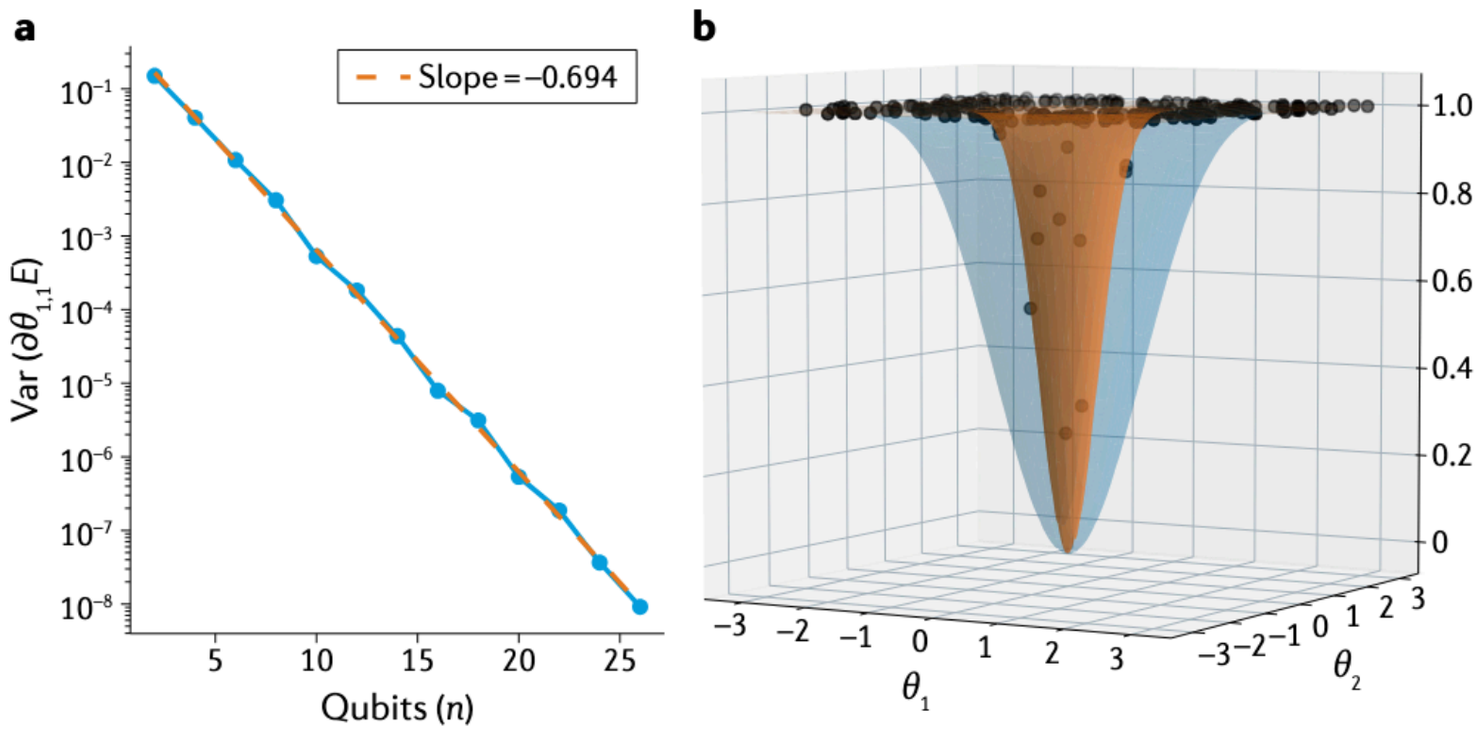
Challenges

1. **Trainability**
   Barren Plateaus, Ansatz and Initialization

2. **Efficiency of estimating the expectation value**
   Commuting sets of operators, Optimized Sampling

3. **Noise** from NISQ Devices





References

- Cerezo, M. et al. Variational quantum algorithms. Nat. Rev. Phys. 3, 625–644 (2021).

- Bharti, K. et al. Noisy intermediate-scale quantum algorithms. Rev. Mod. Phys. 94, 015004 (2022).

- PennyLane by Xanadu, https://pennylane.ai/

-