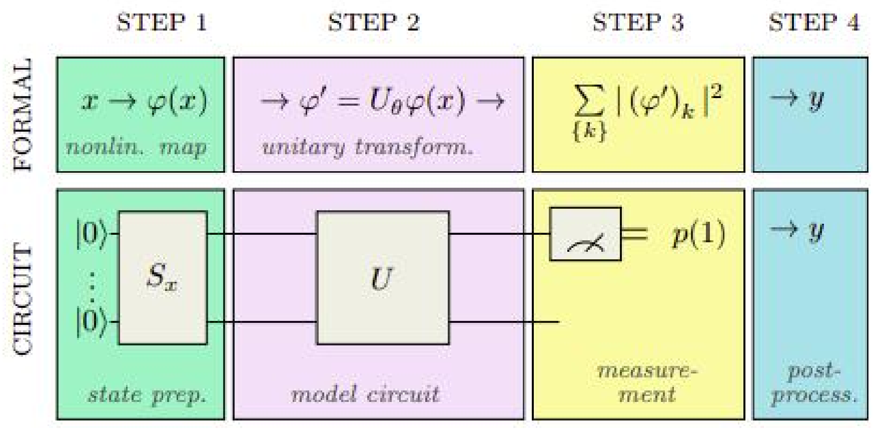
Quantum Machine Learning (QML) is an interdisciplinary field that combines quantum computing with machine learning. One popular approach in QML is using **variational methods with quantum circuits**, such as the **Variational Quantum Classifier (VQC)**. **Pennylane** is a quantum machine learning library that integrates with various quantum computing frameworks and provides tools for building and training quantum circuits.

Concepts Behind Variational Quantum Classifier (VQC)

- **Quantum Circuits**: A quantum circuit is a sequence of quantum gates applied to qubits. The gates manipulate the qubits' states to perform quantum computations.
- **Variational Quantum Algorithms**: These algorithms use quantum circuits with parameters that are adjusted to minimize a cost function. The goal is to optimize these parameters to achieve the best performance on a given task.
- **Variational Quantum Classifier (VQC)**: In the context of classification, VQC uses a quantum circuit to prepare a quantum state that represents data. The circuit has trainable parameters. The output of the circuit is used to make classification predictions.
- **Cost Function**: The cost function measures how well the quantum circuit's output matches the expected classification results. It is minimized using classical optimization techniques.
- **Pennylane**: Pennylane is a library for quantum machine learning that provides tools for building and training quantum circuits. It integrates with various quantum computing frameworks like Qiskit, Cirq, and others.

The Variational Quantum Classifier (VQC) is consists of three parts:

1. Encoding or Embedding;
2. Parametrized Quantum Circuit (Ansatz);
3. Loss Function.



NB: Image from Schuld et al.

Implementation Steps Using Pennylane

Summary : Implementing a Variational Quantum Classifier (VQC) using Pennylane.

· **Setup**: A quantum device and a quantum circuit are defined using Pennylane.

· **Circuit Definition**: A quantum circuit (ansatz) with parameterized gates is created.

- **Cost Function**: A cost function is defined to evaluate the performance of the classifier.
- **Data Preparation**: Synthetic data is generated for training.
- **Training**: The quantum circuit's parameters are optimized using the Adam optimizer.
- **Prediction**: The trained circuit is used to make predictions on new data.
- **Visualization**: Plot the results to get sense if things :

1. **Import libraries**

```
import pennylane as qml
import numpy as np
from pennylane import numpy as pnp
from pennylane.optimize import AdamOptimizer
```

2. **Define the Quantum Device**

```
dev = qml.device('default.qubit', wires=2)
```

3. **Define the Quantum Circuit (Ansatz)**

- AngleEmbedding encodes the classical data x into the quantum state.
- BasicEntanglerLayers applies a series of parameterized gates.
- qml.expval(qml.PauliZ(0)) measures the expectation value of the Pauli-Z operator on the first qubit.

```
@qml.qnode(dev)
def circuit(params, x):
    qml.templates.AngleEmbedding(x, wires=[0, 1])
    qml.templates.BasicEntanglerLayers(params, wires=[0, 1])
        return qml.expval(qml.PauliZ(0))
```

4. **Define the Cost Function**

The cost function measures the **classification error**

```
def cost(params, x, y):
    predictions = np.array([circuit(params, xi) for xi in x])
    predictions = (predictions > 0).astype(int)
    return np.mean(predictions != y)
```

5. **Generate Dummy Data**

Create some **synthetic data for training**

```
np.random.seed(60)
X = np.random.rand(10, 2)  # 10 samples with 2 features each
Y = (np.sum(X, axis=1) > 1).astype(int)  # Labels based on sum of features
```

6. Optimize the Circuit

Set up the **optimizer and train the circuit**

```
params = pnp.random.uniform(-np.pi, np.pi, (3, 2))  # 3 layers, 2 qubits
optimizer = AdamOptimizer(0.1)
for epoch in range(100):
    params, cost_val = optimizer.step_and_cost(lambda p: cost(p, X, Y), params)
    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Cost: {cost_val}")
```

7. Make Predictions

Use the **trained parameters to make predictions**

```
def predict(x, params):
    pred_probs = np.array([circuit(params, xi) for xi in x])
    return (pred_probs > 0).astype(int)

predictions = predict(X, params)
print("Predictions:", predictions)
print("True Labels:", Y)
```

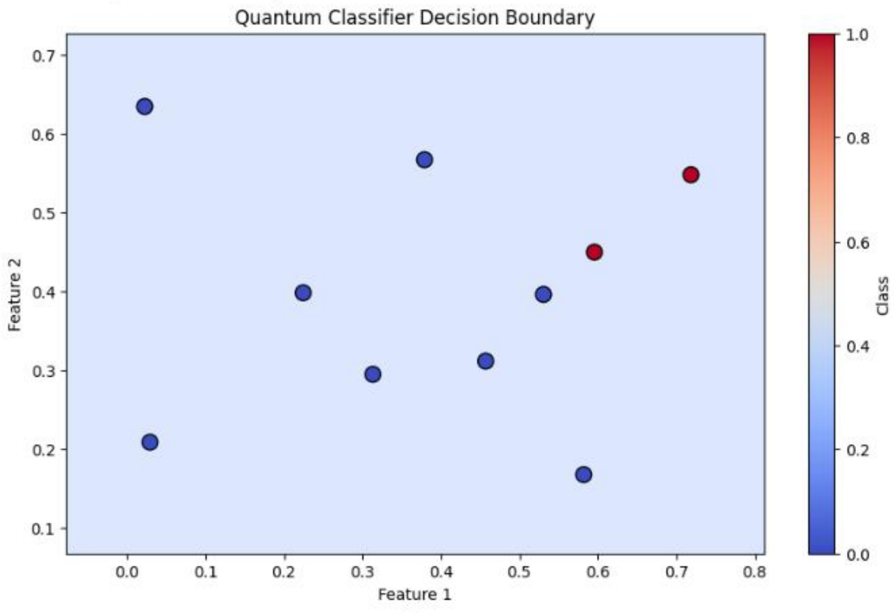
8. Visualization

```
# Plot the decision boundary
def plot_decision_boundary(pred_func, X, Y, title="Decision Boundary"):
    # Create a grid of points
    x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
    y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Predict on the grid
    Z = pred_func(np.c_[xx.ravel(), yy.ravel()], params)
    Z = Z.reshape(xx.shape)

    # Plot the decision boundary
    plt.figure(figsize=(10, 6))
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
    plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolor='k', cmap='coolwarm', marker='o', s=100)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(title)
    plt.colorbar(label='Class')
    plt.show()
# Plot the decision boundary
plot_decision_boundary(predict, X, Y, title="Quantum Classifier Decision Boundary")
# Draw the quantum circuit
import matplotlib.pyplot as plt
from pennylane import draw
# Drawing the circuit
def draw_circuit(params):
    return qml.draw(circuit)(params, X[0])  # Drawing for the first sample in X
# Plot the circuit
circuit_diagram = draw_circuit(params)
print(circuit_diagram)
```

```
Epoch 0, Cost: 0.2
Epoch 10, Cost: 0.2
Epoch 20, Cost: 0.2
Epoch 30, Cost: 0.2
Epoch 40, Cost: 0.2
Epoch 50, Cost: 0.2
Epoch 60, Cost: 0.2
Epoch 70, Cost: 0.2
Epoch 80, Cost: 0.2
Epoch 90, Cost: 0.2
Epoch 100, Cost: 0.2
Epoch 110, Cost: 0.2
Predictions: [0 0 0 0 0 0 0 0 0 0]
True Labels: [0 1 0 0 0 1 0 0 0 0]
```



```
0: ─╭AngleEmbedding(M0)─╭BasicEntanglerLayers(M1)─┤  <Z>
1: ─╰AngleEmbedding(M0)─╰BasicEntanglerLayers(M1)─┤

M0 =
[0.37909853 0.56709817]
M1 =
[[ 0.75084566 -3.12934988]
 [-2.91747163 -0.77415407]
 [-1.78163217 -0.59787346]]
```