QUANTUM MACHINE LEARNING

[BASICS]

QUANTUM COMPUTING >> COMPUTING INSPIRED FROM QUANTUM PHENOMENON

PRE REQUISTIES : DIRAC NOTATION [KET,BRA], QUANTUM ALGEBRA AND GATES

- **Flipping a classical coin :**

2 discrete states => 1 (H) or 0 (T) [bits]

- **Flipping a quantum coin : [imaginary thing for explaining]**

1 quantum state => $|\psi> = \alpha|0> + \beta|1>$ [a qubit]

We always use orthogonal basis vectors |0> and |1> to describe and measure quantum states.

Superposition of both states with basis vectors (1 0) and (0 1) with $\alpha^2 + \beta^2 = 1$

After observing a quantum state it, collapses into a definite state.

[Inspiring Experiment – YDSE]

QUBIT INTUITION?

- Suppose Alice flip a classical coin and Bob flips a quantum coin both at t=0s and both take measurement at t=10s.

- **Alice - Classical Coin: Let a**t t=0.5s, the coin would fall to ground and take a definite state, either H or T. therefore, state is fixed from t=0.5s onwards although measurement not made.
- Result : Probabilistic state 50% from t=0 to t=0.5 and fixed state thereafter.

QUBIT INTUITION?

- **Bob - Quantum Coin:** It would be in a probabilistic state $|\psi>$ till we make a measurement and then it collapses at t=10s. It would also be possible to set to 70% head chances in this magical coin while tossing !! $|\psi> = \alpha|0> + \beta|1>$ **[by controlling alpha and beta values]**
- Result : Any probabilistic state from t=0 to t=10s and fixed state thereafter.
- **This classical coin represents a bit in classical computer and this quantum coin represents a qubit in quantum computer.**

WHY QUANTUM ML ?

- Ans : Because of the exponential SpeedUp it provides than classical ML. It can reduce an O(N) complexity to $O(\log_2 N)$.

HOW EXACTLY SPEED UP HAPPENS?

The extended Question :
Suppose we are applying ML model on an NxM dataset consisting of M feature vectors with each being [x1 x2 x3 … xN], So, for each feature we require N times no. of bits in xi to store and so space complexity being O(N). How can we even talk of storing in something less than O(N) physical with making all sense ? What's the intuition ?

HOW EXACTLY SPEED UP HAPPENS ?

- Lets try to store [x1 x2 … xN] with a superposition of basis states :

$$x_1\begin{pmatrix}1\\0\\\dots\\0\end{pmatrix}+ x_2\begin{pmatrix}0\\1\\\dots\\0\end{pmatrix} + \dots+ x_{\mathrm{N}}\begin{pmatrix}0\\0\\\dots\\1\end{pmatrix}$$

Now, A single qubit can be $|\psi> = \alpha|0> + \beta|1>$ with 2 basis states

2 qubits has basis states |00>, |01>, |10>, |11> count as 4

Where |00> is tensor product of |0> and |0> $=\begin{pmatrix}1\\0\end{pmatrix}*\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}$

$$|\psi\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix}\otimes\begin{pmatrix}\gamma\\\delta\end{pmatrix} = \begin{pmatrix}\alpha\begin{pmatrix}\gamma\\\delta\end{pmatrix}\\\beta\begin{pmatrix}\gamma\\\delta\end{pmatrix}\end{pmatrix} = \begin{pmatrix}\alpha\gamma\\\alpha\delta\\\beta\gamma\\\beta\delta\end{pmatrix}$$

HOW EXACTLY SPEED UP HAPPENS ?

- So, when we take N = 4, the classical vector [x1 x2 x3 x4] can be encoded into $|\psi>$:

$$|\psi\rangle = \frac{x_1}{\sqrt{\sum_{i=1}^{4}|x_i|^2}}|00\rangle + \frac{x_2}{\sqrt{\sum_{i=1}^{4}|x_i|^2}}|01\rangle + \frac{x_3}{\sqrt{\sum_{i=1}^{4}|x_i|^2}}|10\rangle + \frac{x_4}{\sqrt{\sum_{i=1}^{4}|x_i|^2}}|11\rangle$$

Where |00> is $\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$, |01> is $\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$, |10> is $\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$, |11> is $\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$, 4 basis vectors stored in 2 qubits. Hence the speedup occurs from O(N) to O($\log_2$N)

Notably $|\psi>$ contains all the information as in classical vector but uses only $\log_2$N qubits

Fact: with just 275 qubits we can represent more basis states than number of atoms in the observable universe!

INTUITION OF $|\psi>$?

Classical Database

Quantum Database $|\psi>$ [distribution cloud containing full info about classical database]

Aij and nearby cells value can be seen

TAKE MEASUREMENT for Aij

The entire $|\psi>$ collapses to Aij value. Multiple measurement require to see other values

INTUITION OF $|\psi>$?

- INTUITION of this Quantum Database $|\psi>$:

➢ When we look into a classical database lets say on the ith row, jth column, to read its value, our eyes can also simultaneously look into other entries like i+1th row and jth column.

➢ But in a Quantum database $|\psi>$ [like a probability cloud] contains all the information preserved as that of a classical database and we can **take measurement** of a particular entry only by collapse the wavefunction ψ. So when we are reading one entry, we cannot simultaneously look into other entries in this special database.

- So, something similar storage can happen in a quantum computer's RAM called QRAM !

CODE IT OUT ?

Use IBM's qiskit library:

```
import numpy as np
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
from qiskit.quantum_info import Statevector
x1 = 2
x2 = 3
x3 = 4
x4 = 6
# Define the classical vector
x = np.array([x1, x2, x3, x4])
# Normalize the classical vector
norm = np.linalg.norm(x)
x_normalized = x / norm
```

Colab Notebook link:
BasicQML.ipynb - Colab (google.com)

CODE IT OUT ?

```
# Create the quantum state vector based on the normalized classical vector
quantum_state = np.zeros(4, dtype=complex)
for i in range(4):
  quantum_state[i] = x_normalized[i]

# Create a quantum circuit with two qubits
qc = QuantumCircuit(2)

# Initialize the quantum circuit with the quantum state vector
qc.initialize(quantum_state, [0, 1])

# Simulate the circuit and get the statevector
backend = Aer.get_backend('statevector_simulator')
qc_compiled = transpile(qc, backend)
result = backend.run(qc_compiled).result()
statevector = result.get_statevector()
statevector
```

```
Statevector([0.24806947+0.j, 0.3721042 +0.j, 0.49613894+0.j,
             0.74420841+0.j],
            dims=(2, 2))
```

NOW HOW YOU APPLY ML MODEL TO THIS $|\psi>$?

Let's take a classification task for example :

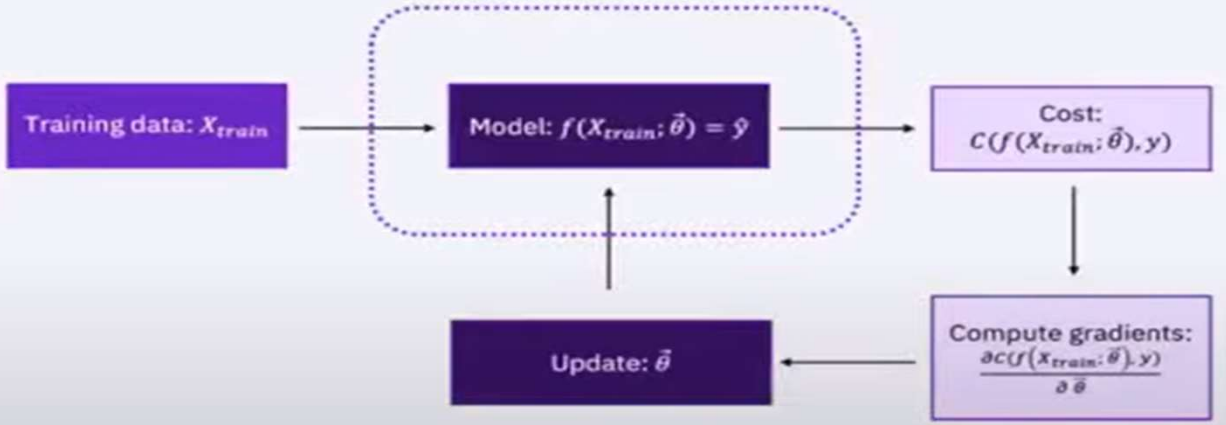
Task: Train a quantum circuit on labelled samples in order to predict labels for new data

Step 1: Encode the classical data into a quantum state

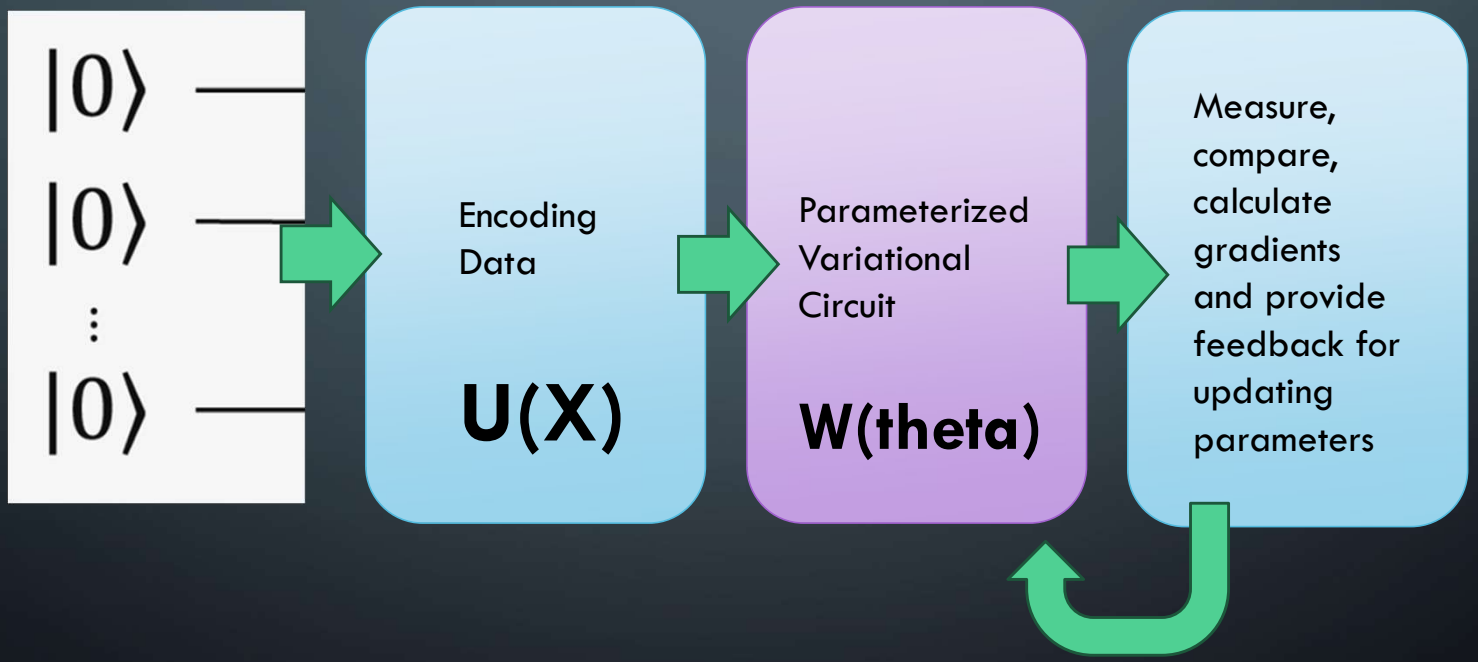
Step 2: Apply a parameterized model

Step 3: Measure the circuit to extract labels

Step 4: Use optimization techniques (like gradient descent) to update model parameters



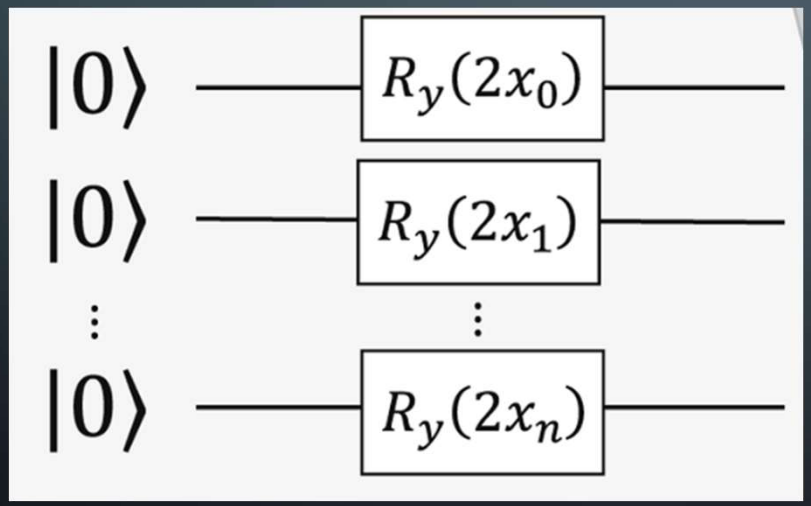
OVERVIEW OF TRAINING:



DATA ENCODING

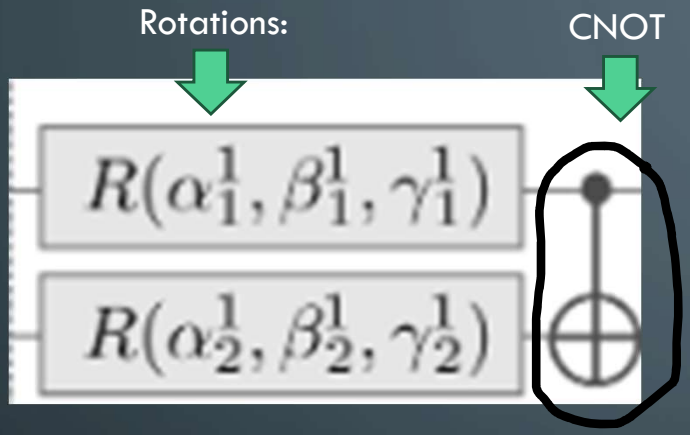
- We already did data amplitude encoding.
- Another way is by Angle encoding:

Suppose X = [x1 x2 … xN]



Where Ry denotes rotation of the qubits in Bloch sphere about Y-axis and the amount of angle it rotates, depending on the values in the X.

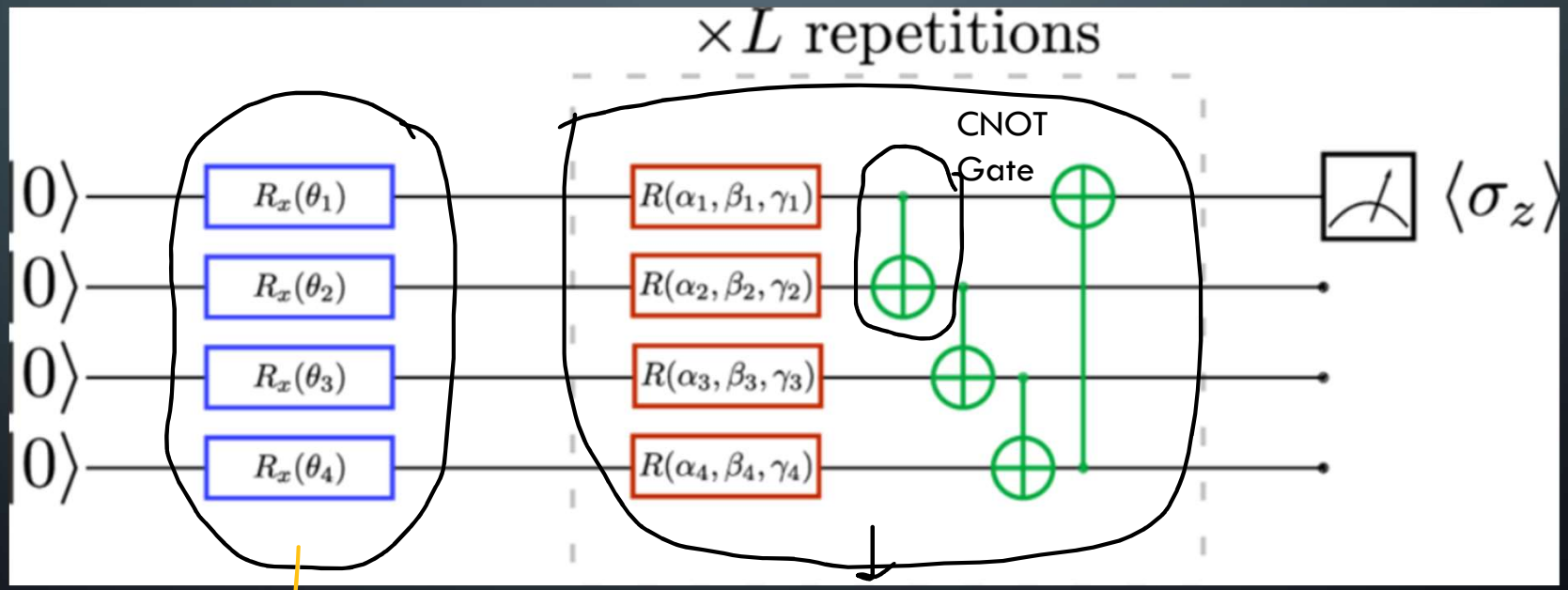
PARAMETERIZED VARIATION CIRCUIT



- The variational circuit consists of layers of parameterized gates (like $Ry(\theta)$Ry(θ), $Rz(\phi)$Rz(ϕ)) and entangling gates (like CNOT).
- Parameter Optimization: The parameters of these gates are **initially set randomly and then optimized using a classical optimizer** based on the performance of the model.

NOTE : Entangling gates (like CNOT) are crucial as they allow the model to create correlations between different qubits. **This helps the model learn complex patterns that depend on the relationships between input features.**

SO WE NOW HAVE THIS:



Encoding step

Entanglement and parameterized rotations

NOW WE GET SOME OUTPUT

- Suppose, we have 2 qubits and perform measurements over and over again to get a probability distribution over all possible states:
- |00> ➔0.7
- |01> ➔0.2
- |10> ➔0
- |11> ➔0.1

PARITY POST PROCESSING

- This is a way of assigning labels,

even parity of qubits => class +1

odd parity of qubits => class -1

$P(\hat{y} = 1) = 0.8\ P(\hat{y} = -1) = 0.2$

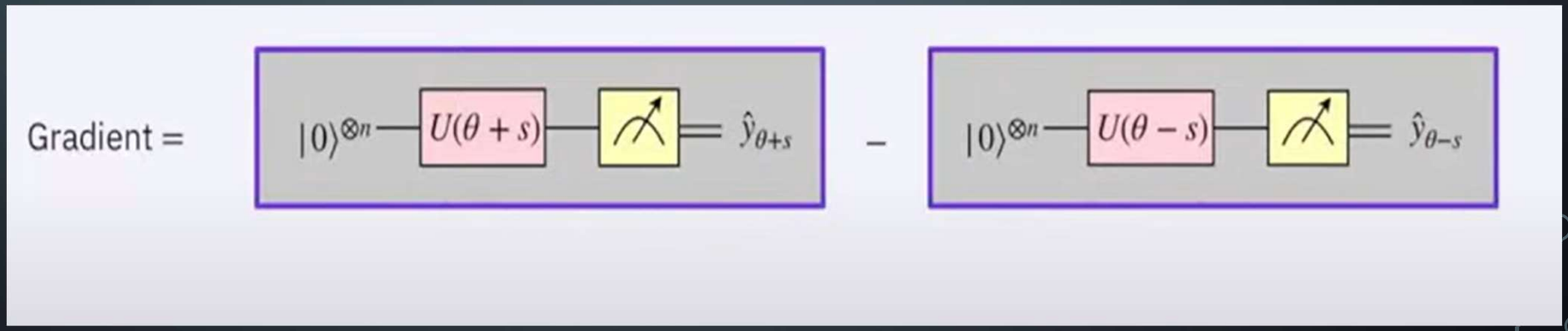
|00> ➔0.7
|01> ➔0.2
|10> ➔0
|11> ➔0.1

Another method : [first qubit method]

Some other ways of assigning labels say that taking only first measurement of the qubit is sufficient by encoding the lost data before taking the measurement in the variational circuit itself.

IS CALCULATION OF GRADIENT OF THESE CIRCUITS POSSIBLE ?

- Yes, it is. And then we can go ahead with applying gradient descent.
- Gradient is given by :



- There are python packages to calculate this gradient

CODE ?

Select a database:

Full code in colab notebook:
Quantum Classifer.ipynb - Colab (google.com)

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Normalize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Encode labels to binary
encoder = LabelEncoder()
y = encoder.fit_transform((y == 0).astype(int))  # Only classify setosa vs non-setosa
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

```
Training set shape: (120, 4)
Test set shape: (30, 4)
```

ENCODE ALL THE TRAINING EXAMPLES

```python
from qiskit import QuantumCircuit
def create_quantum_circuit(data):
    """Creates a quantum circuit to encode classical data."""
    num_features = len(data)
    qc = QuantumCircuit(num_features)

    for i in range(num_features):
        qc.ry(data[i], i)  # Encode data using Ry rotation gates

    return qc
# Example encoding for the first training sample
qc = create_quantum_circuit(X_train[0])
print(qc.draw())
```

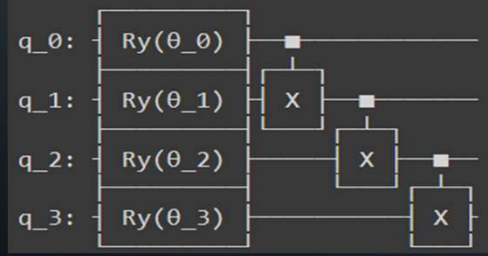
```
q_0: ┤ Ry(-1.5065) ├
     ├ Ry(1.2492) ┤
q_1: ┤ Ry(-1.5676) ├
     ├ Ry(-1.3154) ┤
q_2: ┤             ├
q_3: ┤             ├
```

CREATE PARAMETERIZED VARIATIONAL QUANTUM CIRCUITS

```python
from qiskit.circuit import Parameter

def create_variational_circuit(num_qubits):
    """Creates a parameterized variational quantum circuit."""
    qc = QuantumCircuit(num_qubits)
    # Define parameters
    theta = [Parameter(f'θ_{i}') for i in range(num_qubits)]
    for i in range(num_qubits): #applying rotations
        qc.ry(theta[i], i)
    for i in range(num_qubits - 1): #applying CNOT entanglements
        qc.cx(i, i + 1)
    return qc, theta

# Example variational circuit for 4 features
var_qc, params = create_variational_circuit(4)
print(var_qc.draw())
```



COMPOSE THE ENCODING STEP AND THE VARIATIONAL STEP

```
from qiskit import transpile
from qiskit_aer import Aer
from scipy.optimize import minimize
#from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

simulator = Aer.get_backend('qasm_simulator') #quantum simulator

# Define the full quantum circuit for one data point
def create_full_circuit(data, params_values):
    qc = create_quantum_circuit(data)
    var_qc, params = create_variational_circuit(data.shape[0])
    bound_var_qc = var_qc.assign_parameters({params[i]: params_values[i] for i in range(len(params))})
    qc = qc.compose(bound_var_qc)
    #var_qc.draw()
    #qc.draw()
    qc.measure_all()
    return qc
```

DESIGN AN OBJECTIVE COST FUNCTION TO MINIMIZE [MSE]

```python
# Cost function to minimize
def cost_function(params_values):
    total_cost = 0
    for i, data in enumerate(X_train):
        qc = create_full_circuit(data, params_values)
        transpiled_qc = transpile(qc, simulator)
        result = simulator.run(transpiled_qc, shots=1024).result()
        counts = result.get_counts()
        # Convert measurement results to binary outcomes
        predictions = (counts.get('0000', 0) + counts.get('0001', 0) >= counts.get('1110', 0) + counts.get('1111', 0))
        total_cost += (predictions - y_train[i]) ** 2
    return total_cost / len(X_train)
```

OPTIMIZE THE PARAMTERS

```
# Optimize the parameters
initial_params = np.random.rand(4) * np.pi
result = minimize(cost_function, initial_params, method='COBYLA')

# Optimized parameters
optimal_params = result.x
print("Optimal parameters:", optimal_params)
```

```
Optimal parameters: [2.80677181 0.63010604 4.72033923 4.44707588]
```

COBYLA (Constrained Optimization BY Linear Approximations) is a numerical optimization method designed for solving non-linear programming problems with inequality constraints.

EVALUATE ON TEST SET

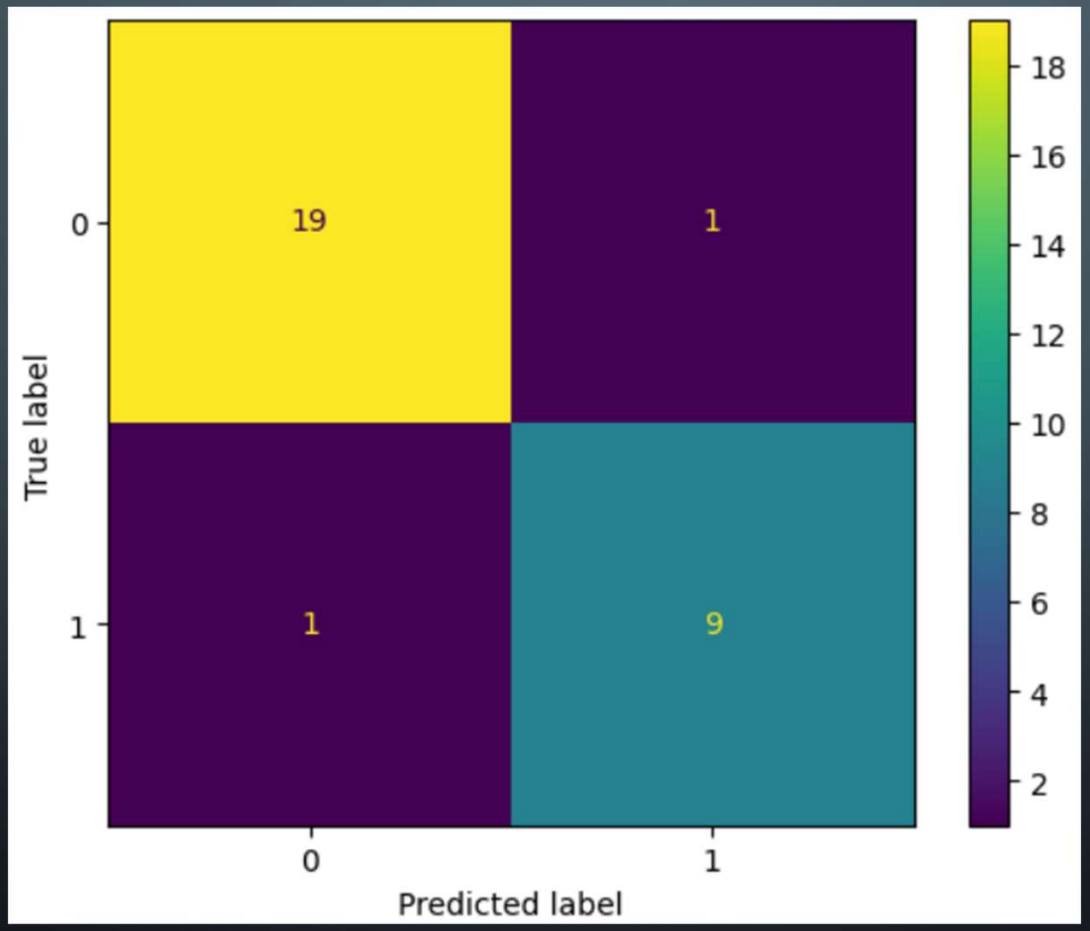
```python
# Evaluate on the test set
def predict(data, params_values):
    qc = create_full_circuit(data, params_values)
    transpiled_qc = transpile(qc, simulator)
    result = simulator.run(transpiled_qc, shots=1024).result()
    counts = result.get_counts()
    prediction = (counts.get('0000', 0) + counts.get('0001', 0) >= counts.get('1110', 0) + counts.get('1111', 0))
    return prediction

# Calculate accuracy
correct_predictions = 0
for i, data in enumerate(X_test):
    prediction = predict(data, optimal_params)
    if prediction == y_test[i]:
        correct_predictions += 1

accuracy = correct_predictions / len(X_test)
print("Test set accuracy:", accuracy)
```

```
Test set accuracy: 0.9666666666666667
```

PLOT THE CONFUSION MATRIX



We can see most of the concentration is along the diagonal

QUANTUM ADVERSARIAL MACHINE LEARNING

- Introduction :

➢ Quantum computing harnesses quantum mechanical phenomena, such as superposition and entanglement, to perform computations that would be infeasible for classical computers.

➢ Adversarial machine learning involves crafting inputs to deceive machine learning models.

➢ QAML aims to investigate the intersection of these fields, exploring both offensive and defensive applications.

GENERATORS AND DISCRIMINATORS

- Generators: The generator's role is to create data that is as similar as possible to real data. It learns to map random noise vectors (latent space) to the data distribution of the training set.

- Discriminator: The discriminator's role is to distinguish between real data (from the training set) and fake data (produced by the generator).

THE OBJECTIVE FUNCTION

- The objective function for GANs combines the objectives of both the generator and the discriminator into a single value function $V(D,G)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

MAXIMIZING THE DISCRIMINATOR'S OBJECTIVE:

- The discriminator aims to correctly classify real and fake data.
- The term $Ex{\sim}p$data$(x)[\log D(x)]$ represents the expected value of the logarithm of the discriminator's output for real data samples x. This term is maximized when $D(x)$ is close to 1 (indicating real data).
- The term $Ez{\sim}pz(z)[\log(1-D(G(z)))]$ represents the expected value of the logarithm of $1-D(G(z))$ for fake data samples generated by $G(z)$. This term is maximized when $D(G(z))$ is close to 0 (indicating less fake data).

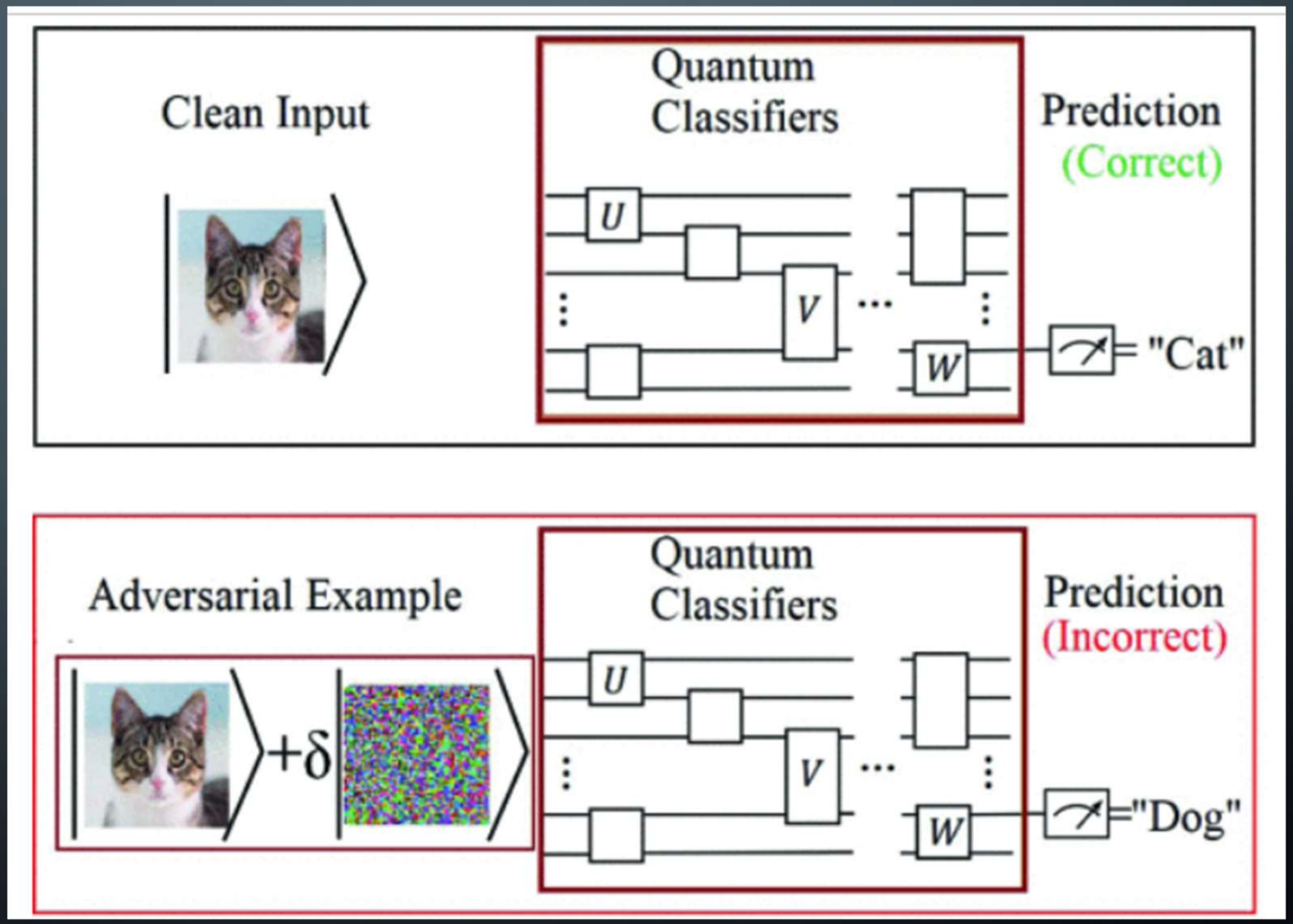
$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

QUANTUM ADVERSARIAL ATTACKS

- Arunachalam and de Wolf (2017) demonstrated that quantum computers could theoretically generate adversarial examples faster than classical counterparts due to their ability to perform certain linear algebra operations more efficiently.
- Objective: Generate adversarial examples x' from input x such that the perturbation $\delta = x' - x$ fools the model.

ADVERSARIAL EXAMPLE

the effectiveness of an attack on a dataset is measured by its misclassification rate: per centage of correctly predicted samples which are misclassified after the attack.



NOW HOW TO CORRECT THE MODEL AFTER AN ADVERSARIAL ATTACK ?

- First, let's do what's most intuitive. Simply include our adversarial examples in the training and retrain the model.
- **Adversarial training** enhances model robustness by exposing the model to adversarial examples during training. This strategy helps the model to generalize better and defend against adversarial attacks in real-world scenarios.

QUANTUM ROBUSTNESS CERTIFICATION

- Objective: Certify that a model f is robust within a certain perturbation bound ϵ.
- Mathematical Approach: Use quantum algorithms to solve optimization problems that verify robustness:

$$\text{certify}(f, x, \epsilon) = \min_{\|\delta\| \leq \epsilon} \mathcal{L}(f(x), f(x + \delta))$$

where L is a loss function measuring the discrepancy between outputs.

SHOW IN CODE HOW ACCURACY ACTUALLY INCREASES AFTER ADVERSARIAL TRAINING ?

Train on MNIST DATASET normally, validation accuracy starts from 95.7% around

```python
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical

# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)

# Define a simple neural network model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
1875/1875 [==============================] - 14s 7ms/step - loss: 0.2534 - accuracy: 0.9278 - val_loss: 0.1432 - val_accuracy: 0.9571
```

CREATE A QUANTUM CIRCUIT AND EXECUTE IT TO GET PERTURBATION

```
from qiskit_aer import Aer
from qiskit.circuit import QuantumCircuit, Parameter
import numpy as np
def generate_adversarial_example(x, epsilon=0.01):
    qc = QuantumCircuit(1, 1)
    param = Parameter('θ')
    qc.rx(x.flatten()[0] * np.pi, 0)
    qc.ry(x.flatten()[1] * np.pi, 0)
    qc.rz(param, 0)
    qc.measure(0, 0)

    backend = Aer.get_backend('qasm_simulator')
    param_value = np.random.uniform(0, np.pi)
    job = backend.run(qc.assign_parameters({param: param_value}),shots=1024)
    result = job.result()
    counts = result.get_counts()

    perturbation = epsilon * (counts.get('0', 0) - counts.get('1', 0)) / 1024.0
    adv_x = x + perturbation
    return np.clip(adv_x, 0, 1)  # Ensure pixel values are in [0, 1]
# Generate adversarial examples for a batch of test images
x_test_adv = np.array([generate_adversarial_example(x) for x in x_test])
```

RETRAIN BY COMBINING ADVERSARIAL EXAMPLES

```
# Combine original and adversarial examples
x_train_combined = np.concatenate((x_train, x_train))
y_train_combined = np.concatenate((y_train, y_train))

# Generate adversarial examples for training
x_train_adv = np.array([generate_adversarial_example(x) for x in x_train])
x_train_combined = np.concatenate((x_train, x_train_adv))

# Re-train the model with adversarial examples
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train_combined, y_train_combined, epochs=5, validation_data=(x_test, y_test))
```

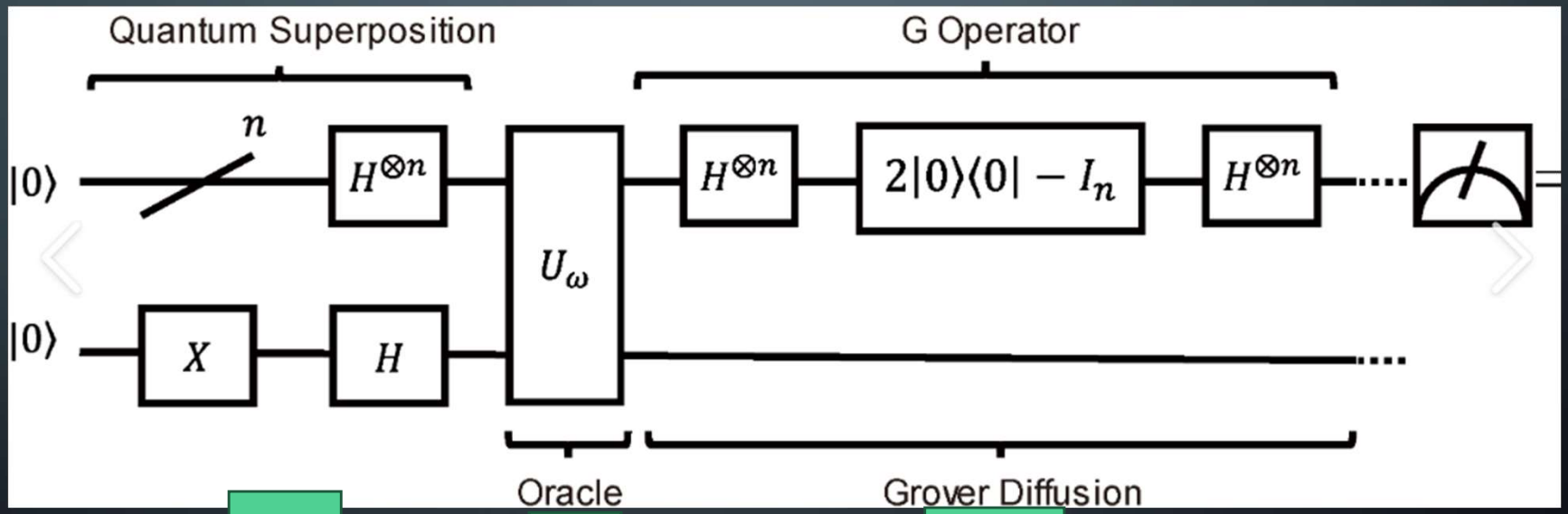
```
Epoch 1/5
3750/3750 [==============================] - 16s 4ms/step - loss: 0.0331 - accuracy: 0.9896 - val_loss: 0.0741 - val_accuracy: 0.9786
Epoch 2/5
3750/3750 [==============================] - 15s 4ms/step - loss: 0.0214 - accuracy: 0.9932 - val_loss: 0.0822 - val_accuracy: 0.9765
Epoch 3/5
3750/3750 [==============================] - 15s 4ms/step - loss: 0.0157 - accuracy: 0.9951 - val_loss: 0.0835 - val_accuracy: 0.9778
Epoch 4/5
3750/3750 [==============================] - 15s 4ms/step - loss: 0.0114 - accuracy: 0.9964 - val_loss: 0.0945 - val_accuracy: 0.9772
Epoch 5/5
3750/3750 [==============================] - 16s 4ms/step - loss: 0.0084 - accuracy: 0.9974 - val_loss: 0.0963 - val_accuracy: 0.9791
<keras.src.callbacks.History at 0x7f8f233460b0>
```

We can see that the validation accuracy in adversarial training is somewhat greater

FAST ADVERSARIAL EXAMPLE GENERATION WITH GROVER'S ALGORITHM

- Quantum Speedup: Quantum algorithms can solve optimization problems underlying adversarial attacks more efficiently. For example, the Grover-based search can find adversarial examples quadratically faster.
- Grover's Algorithm makes "searching an unsorted database to find a particular element" in O($\sqrt{n}$)time.

GROVER'S ALGORITHM



Generates uniform superposition

Phase inversion

Diffuser, inversion about mean

GROVER'S ALGORITHM

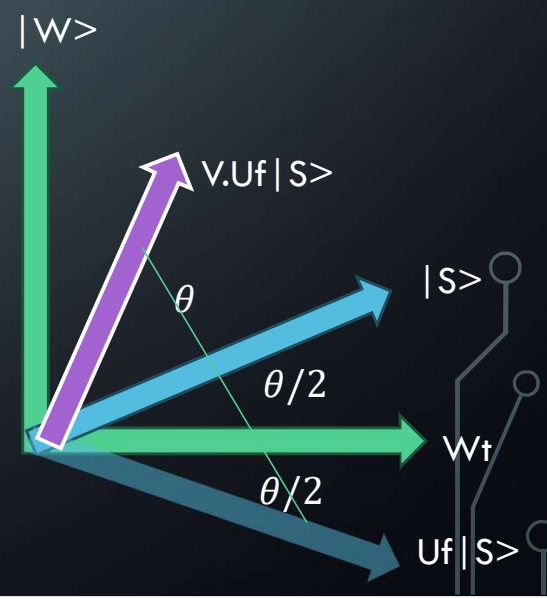
On applying Hadamard Gate, on every qubit, we obtain a uniform superposition |S>

$$H^{\theta n}|0>=\frac{1}{\sqrt{2^n}}\sum_{k\in\{0,1\}^n}^{1}|x>=|s>$$

Then we define an oracle black box which would answer us whenever we need to stop the iterations. Let f(x) = 1 ,when x = w, 0 otherwise
So, Uf: |w> ➔ -|w>, |x> ➔ |x>

Uf|S> = (II – 2 |wXw|)|S>

Lastly, inversion about mean: V = 2|SXS| - II
(V.Uf)^r |S>
Using, r$\theta+\theta/2$ = pi/2, and also sin$\theta/2 = 1/\sqrt{2^n}$
We can proof r to be O($\sqrt{n}$)



GROVER'S ALGO - CODE

```
# Define the oracle for Grover's algorithm
def oracle(qc, qubits, ancilla):
    qc.cz(qubits[0], ancilla)  # Simplified example, real implementation depends on the problem

# Define the Grover diffusion operator
def diffuser(qc, qubits):
    n = len(qubits)
    for q in qubits:
        qc.h(q)
    for q in qubits:
        qc.x(q)
    qc.h(qubits[-1])

    # Add the multi-controlled Toffoli (MCX) gate
    mcx = MCXGate(n-1)
    qc.append(mcx, qubits)

    qc.h(qubits[-1])
    for q in qubits:
        qc.x(q)
    for q in qubits:
        qc.h(q)
```

Flips the target qubit iff all control qubits are in state 1

ORACLE : The oracle applies a controlled-Z gate (phase flip). In simple words, ask the oracle which perturbation is causing a misclassification ?

Diffusion operator inverts the amplitudes of all states about the average amplitude, effectively amplifying the amplitudes of the marked states (potential solutions).

GROVER'S ALGO - CODE

```
# Create a quantum circuit
n_qubits = int(np.ceil(np.log2(X_train.shape[1])))
qc = QuantumCircuit(n_qubits + 1)  # +1 for the ancilla qubit

# Initialize to superposition
for q in range(n_qubits):
    qc.h(q)

# Apply Grover's algorithm
iterations = int(np.pi / 4 * np.sqrt(2 ** n_qubits))

for _ in range(iterations):
    oracle(qc, list(range(n_qubits)), n_qubits)
    diffuser(qc, list(range(n_qubits)))

# Measure the qubits
qc.measure_all()

# Execute the quantum circuit
backend = Aer.get_backend('qasm_simulator')
result = backend.run(qc, shots=1024).result()
counts = result.get_counts()
```

After running the quantum circuit and measuring the qubits, the resulting counts reveal which perturbations have the highest probability amplitude. The bitstring with the highest count is most likely the perturbation that the oracle (misclassification check) marked as causing a misclassification.

```
# Analyze the results to find the adversarial example
most_common_bitstring = max(counts, key=counts.get)
adversarial_sample = original_sample.copy()
```

WHITE BOX ATTACK

- White-box attacks are adversarial attacks where the attacker has complete knowledge of the model architecture, parameters, and training data. This knowledge allows the attacker to craft precise adversarial examples that can fool the model into making incorrect predictions.
- Fast Gradient Sign Method (FGSM): [paper by Ian Goodfellow]

FGSM generates adversarial examples by using the gradient of the loss with respect to the input data. It aims to find the **direction in which the input can be perturbed to maximize the loss,** thereby causing the model to misclassify the input.

FAST GRADIENT SIGN METHOD (FGSM):

- x : The original input
- y : The true label of x
- θ : The model parameters
- $J(\theta,x,y)$: The cost function used to train the model
- The adversarial example $x$adv is generated as follows:
- $x\text{adv}=x+\eta\cdot\text{sign}(\nabla xJ(\theta,x,y))$
- η : The perturbation magnitude (a small scalar value controlling the strength of the attack)
- sign(·) : The sign function that takes the sign of the gradient

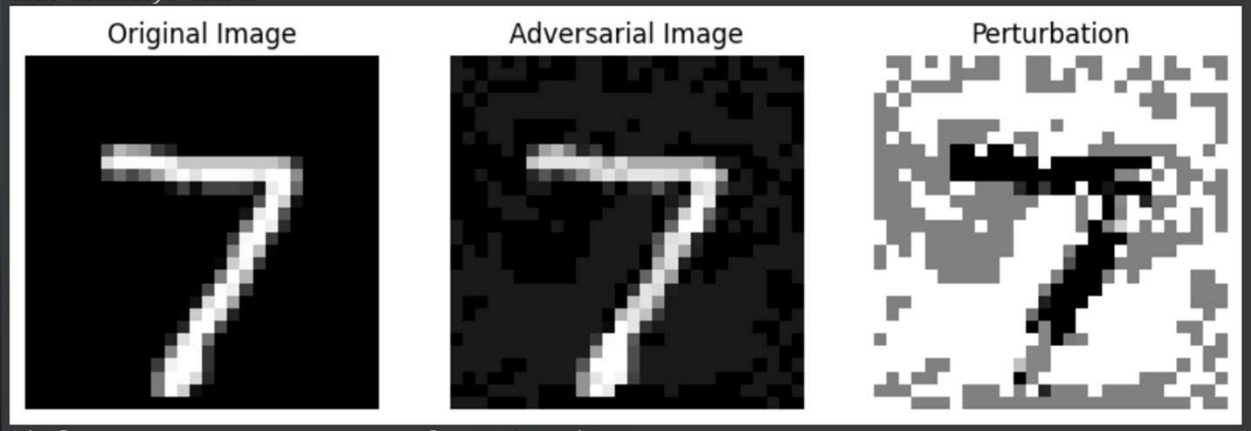
FGSM CODE WITH DEMO

```python
# FGSM attack function
def fgsm_attack(model, x, y, epsilon):
    x_adv = tf.convert_to_tensor(x)
    with tf.GradientTape() as tape:
        tape.watch(x_adv)
        prediction = model(x_adv)
        loss = tf.keras.losses.categorical_crossentropy(y, prediction)
    gradient = tape.gradient(loss, x_adv)
    signed_grad = tf.sign(gradient)
    x_adv = x_adv + epsilon * signed_grad
    x_adv = tf.clip_by_value(x_adv, 0, 1)  # Ensure values remain in [0, 1]
    return x_adv
```

Select the adversarial example with maximum losses towards the training gradient

RESULTS

```
Epoch 4/5
422/422 [==============================] - 7s 15ms/step - loss: 0.1018 - accuracy: 0.9710 - val_loss: 0.0948 - val_accuracy: 0.9745
Epoch 5/5
422/422 [==============================] - 4s 9ms/step - loss: 0.0819 - accuracy: 0.9760 - val_loss: 0.0942 - val_accuracy: 0.9728
313/313 [==============================] - 1s 2ms/step - loss: 0.0965 - accuracy: 0.9703
Test accuracy: 0.9703
```




```
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 22ms/step
Original Prediction: 7, Adversarial Prediction: 3
```

OTHER ATTACK METHODS

- Additive Universal Adversarial Perturbations: [in paper]

Given the complete knowledge of a dataset, the strength of perturbation which can make the quantum model to misclassify can be mathematically calculated:

Theorem 1 *For an additive universal adversarial perturbation p applied on inputs of classifier $\mathcal{Q}$, a strength of perturbation $\|p\| \in \mathbb{R}$ will cause $\mathcal{Q}$ to classify all inputs as c (class to which $p/\|p\|$ belongs) if:*

$$\|p\| \geq \frac{2}{\epsilon_c \sqrt{4 - \epsilon_c^2}}$$

where ϵ_c is given by:

$$\epsilon_c = \sqrt{1 + \frac{1}{2d} \cdot \left(\hat{p}^T M^c \hat{p} - \hat{p}^T M^{c'} \hat{p}\right)} - 1$$

where $\hat{p} = p/\|p\|$ and c' is the class with highest output probability for $\hat{p}$ after c.

This is a white box attack method since we need to have information about the dataset, the class with the highest and second highest probability

ANY BOUNDS FOR PERTURBED INPUT ?

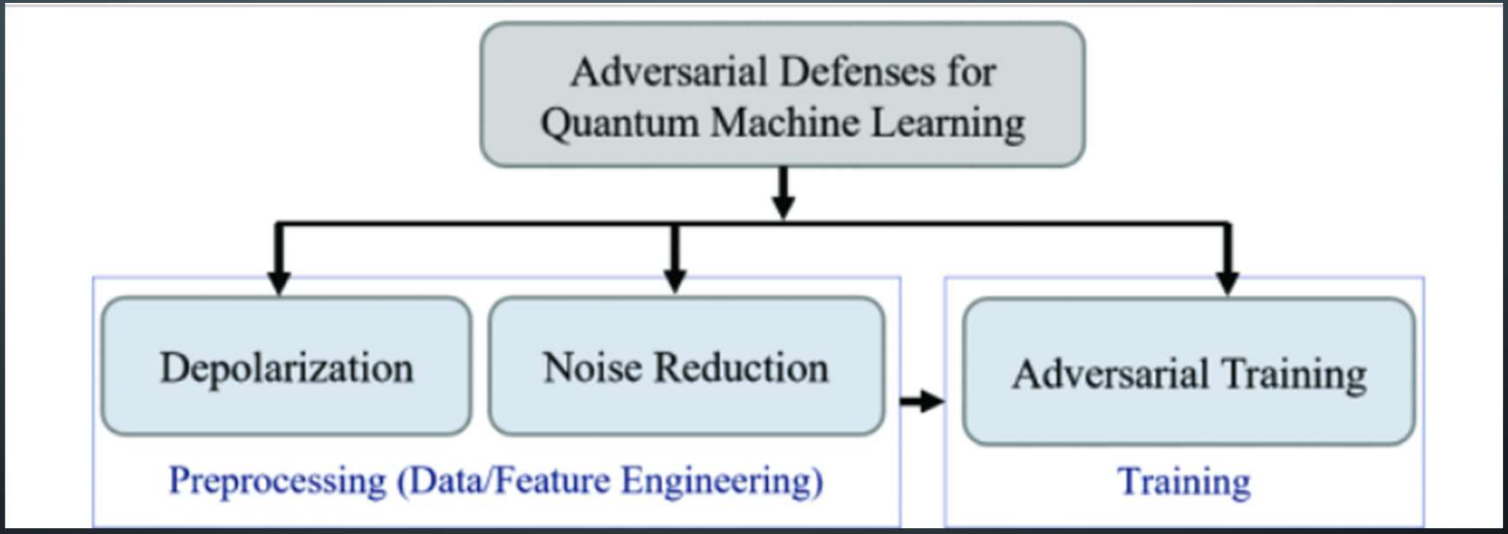
- To make sure our perturbed input doesn't greatly vary from actual quantum dataset, we can simply include it in the loss function as done in paper:
- $L_U = L_{fool} + \alpha L_{fid}$, quite obviously, higher α would mean more more strictness on perturbed state and less misclassification allowed

where L_{fool} is the fooling loss and L_{fid} is a fidelity-based loss of the form:

- $L_{fid} = (1 - F(|\psi\rangle, |\phi\rangle))^2$,

$F(|\psi\rangle, |\phi\rangle) = |\langle\psi|\phi\rangle|^2$ is the fidelity between the input state and the perturbed state. α is a hyper-parameter controlling the trade-off between misclassification and fidelity.

DEFENSE MECHANISMS ?



DEFENSE MECHANISM - DEPOLARIZATION

- Depolarization is a technique used to mitigate the effects of noise in quantum systems. In the context of quantum adversarial machine learning, depolarization can be used to reduce the impact of adversarial perturbations on quantum data.
- Depolarizing Channel: This is a common quantum noise model where a quantum state $\rho\rho$ is transformed into a mixture of itself and the maximally mixed state I/d (where I is the identity matrix and d is the dimension of the Hilbert space). The depolarizing channel with depolarization parameter p can be represented as:

$$E(\rho)=(1-p)\rho+pId$$

Here, p represents the probability of the quantum state being depolarized.

- Defense Mechanism: By applying a depolarizing channel during preprocessing, we can reduce the influence of adversarial perturbations by making the quantum states more resilient to small changes

DEFENSE MECHANISM – NOISE REDUCTION

- Noise reduction techniques aim to filter out unwanted disturbances from the quantum data before it is used for training or inference.
- **Quantum Error Correction (QEC):** This involves encoding the quantum information in a way that allows the detection and correction of certain types of errors without directly measuring the quantum state. QEC can help protect quantum states from both natural noise and adversarial perturbations.
- **Noise Filtering:** This technique involves the use of quantum algorithms designed to suppress specific types of noise. For example, the use of spectral methods to filter out high-frequency noise components from the quantum data.