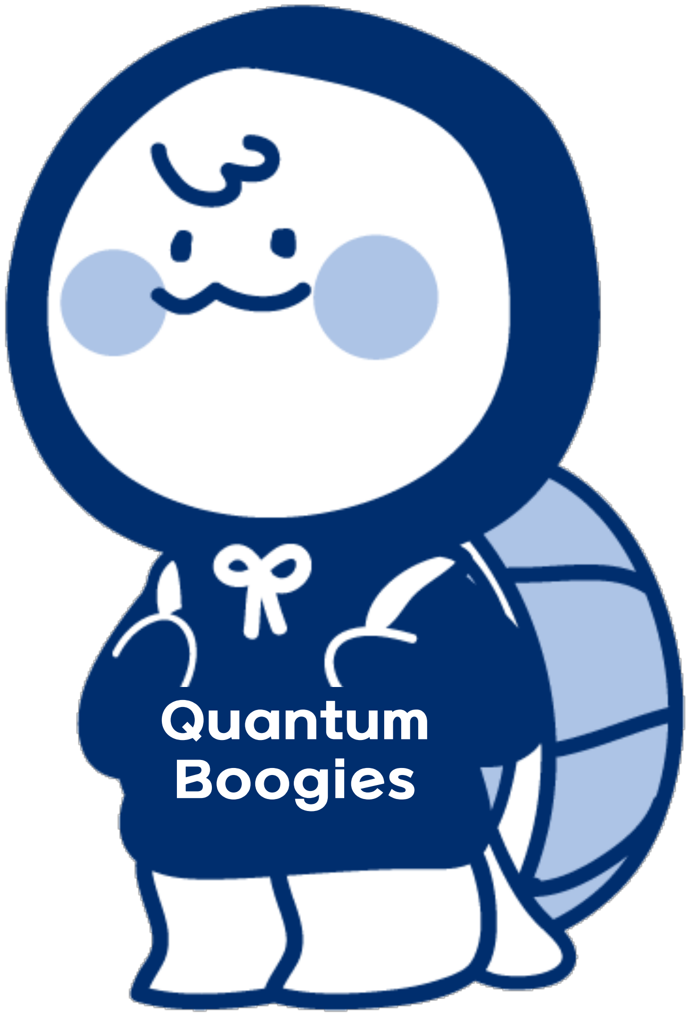
Quantum Neural Distinguisher for Differential Cryptanalysis

For Simplified DES

https://youtu.be/1MeE5NJU-Us



Quantum Boogies : 김현지, 임세진, 강예준, 김원웅

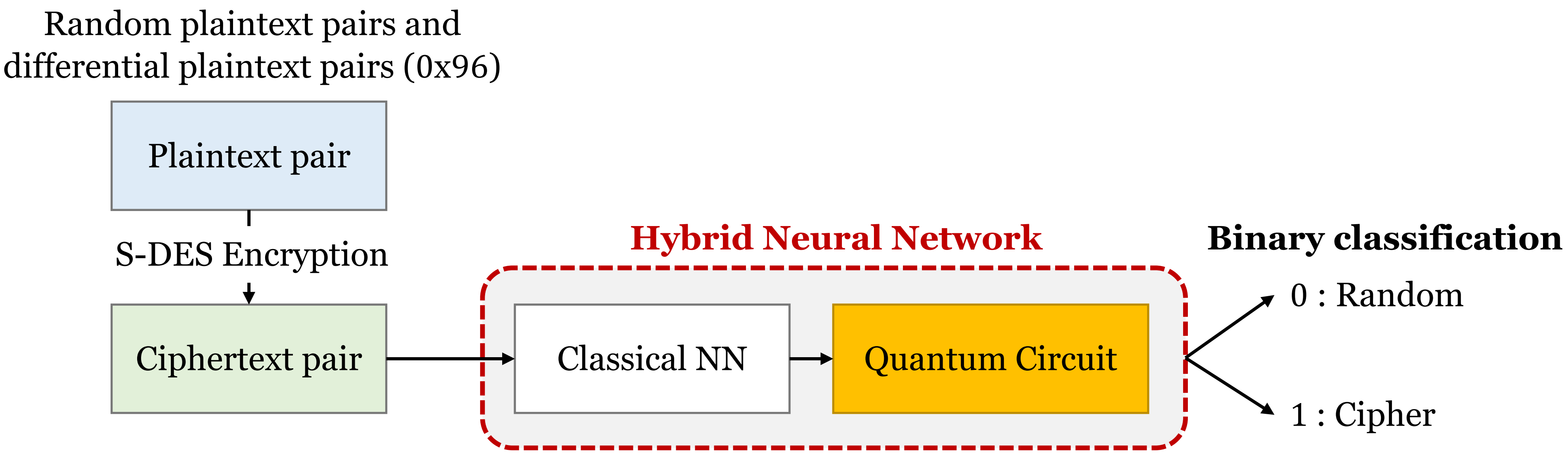
Contents

- **Differential Cryptanalysis**
  - Cryptanalysis techniques for block ciphers
  - Probabilistically predict the change of the output according to the change of the input
  - Process
    - Step 1: Find the difference characteristics
    - **Step 2:** Find the plaintext pair (P,P'(= P⊕input difference)) that satisfies the input difference
    - Step 3: Brute force all round keys
- **Neural Distinguisher**
  - For differential cryptanalysis, it is necessary to find a pair of plaintext that satisfies the difference (Step2)

    → Using neural distinguisher for step 2
  - Distinguish between random and ciphertext pairs
  - When classified as random data, it is not used for differential cryptanalysis (Abort)

- **Quantum Neural Distinguisher (Our work)**
  - We implemented the neural distinguisher using quantum-classical hybrid neural network.
  - 1-qubit hybrid neural network
  - Higher accuracy, fewer epoch and fewer training data than classical neural network

- **Design of quantum neural distinguisher**
  - Encryption the plaintext pairs (random and difference)
  - Ciphertext pairs are input to quantum-classical hybrid neural network
  - Classical NN → Quantum circuit (quantum layer) → Classification (Random or Cipher)



Quantum neural distinguisher

- **Data (S-DES)**
  - 10-bit key, 8-bit plaintext/ciphertext, 2-round
  - Input difference : 0x96 (10010110)
  - Details
    - Random plaintext pair $(P0, P1)$
    - Difference plaintext pair $(P0, P0 \oplus 0x96)$
    - Encrypt plaintext pairs

      → Random ciphertext pairs (label 0) and difference ciphertext pairs (label 1)
    - All ciphertext pairs are expressed as bits to form a data set

Data		Label
Random Pair 0	Random Pair 1	0
⋮	⋮	⋮
Difference Pair 0	Difference Pair 1	1

Random/Difference Pair 0 (Bit)			Random/Difference Pair 1 (Bit)		Label
1	0	⋯	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮
1	1	⋯	0	1	1

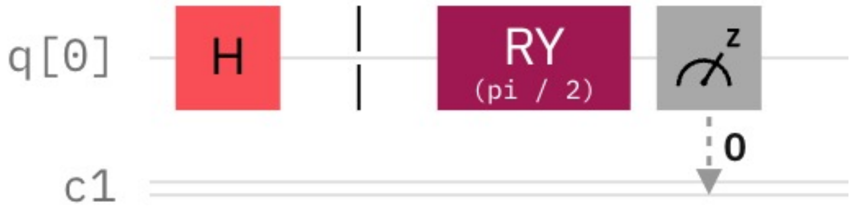
02 Quantum neural distinguisher

- **Quantum-classical hybrid neural network**
  - Architecture of hybrid neural network

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.input = nn.Linear(16, 32)
        self.dropout = nn.Dropout2d()
        self.fc0 = nn.Linear(32, 16)
        self.fcout = nn.Linear(16, 1)

        self.hybrid = Hybrid(qiskit.Aer.get_backend('aer_simulator'), 100, np.pi / 2)

    def forward(self, x):
        x = F.relu(self.input(x))
        x = self.dropout(x)
        x = F.relu(self.fc0(x))
        x = self.dropout(x)
        x = self.fcout(x)
        x = self.hybrid(x)
        return torch.cat((x, 1 - x), -1)
```

  - Architecture of quantum circuit

```python
self._circuit.h(all_qubits)
self._circuit.barrier()
self._circuit.ry(self.theta, all_qubits)

self._circuit.measure_all()
```

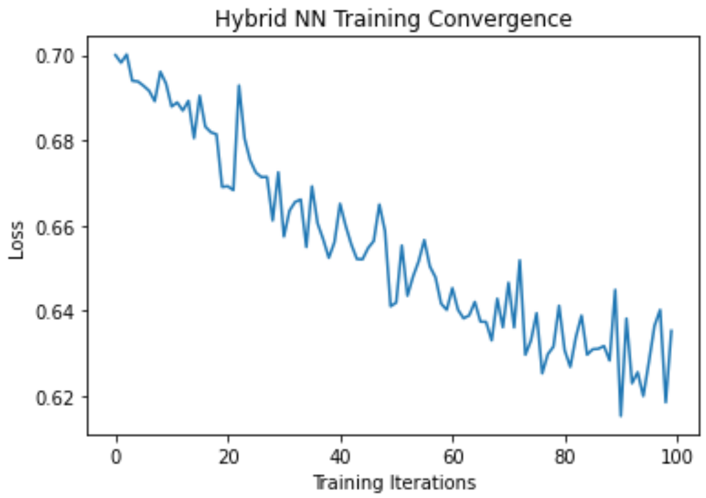


03 Classical Neural Distinguisher vs. Quantum Neural Distinguisher for S-DES

Architecture	Accuracy			Epoch	Shots	The number of data
	Training	Validation	Test			
Classical	63.0%	48.5%	50.0%	50	–	500
Classical	72.3%	55.0%	53.5%	50	–	1000
Quantum-classical hybrid	79.6%	56.2%	52.0%	30	100	500
Quantum-classical hybrid	78.4%	57.2%	55.8%	50	100	500
Quantum-classical hybrid	73.2%	56.7%	56.9%	30	100	1000
Quantum-classical hybrid	75.6%	57.9%	57.3%	50	100	1000
Quantum-classical hybrid	80.6%	58.4%	57.5%	100	100	1000
Quantum-classical hybrid	82.9%	55.8%	58.8%	100	1024	1000

03 Discussion

- **The number of data : 500**
  - **Classical** : Failed (Accuracy 50%)
  - **Quantum hybrid** : Succeeded (Accuracy exceeding 50%)
    - Accuracy 52.0% (shots=100, epoch=30)
    - Accuracy 55.8% (shots=100, epoch=50)
- **The number of data : 1000**
  - **Classical** : Succeeded (Accuracy 53.5% (epoch=50))
  - **Quantum hybrid** : Succeeded (Accuracy exceeding 50%)
    - Accuracy 56.9% (shots=100, epoch=30)
    - Accuracy 57.3% (shots=100, epoch=50)
    - Accuracy 57.5% (shots=100, epoch=100)
    - Accuracy 58.8% (shots=1024, epoch=100) : the highest accuracy
- Higher loss than classical networks, but higher accuracy.



03 Discussion

- **Quantum advantage**
  - In all cases, the hybrid neural network achieved higher accuracy than the classical neural network.
  - Better performance can be achieved with fewer epochs and fewer training data.
  - In one case, the classical neural network failed, but quantum hybrid succeeded.
  - Successfully used in step 2 of the differential cryptanalysis.

04 Conclusion and future work

- **Conclusion**
  - We implemented a quantum neural distinguisher using a hybrid neural network.
  - Comparison with a classical neural network.
  - Hybrid neural network has quantum advantages
    - Higher accuracy, fewer training data and fewer epochs
  - Quantum neural distinguisher can be successfully used for differential cryptanalysis for S-DES.
- **Future work**
  - Try to perform cryptanalysis for other cipher (e.g. S-AES, DES, Speck).
  - Improving our accuracy
  - Apply QSVM (only quantum circuits) to a quantum distinguisher.

해커톤에서 알게 된 것

- **Fake device**
  - 100 큐빗 디바이스를 사용할 수는 있으나 매우 느리고 노이즈가 심함

    → 원래 0.7에서 시작해서 0.6정도까지 loss가 감소

    그러나 동일한 작업 시에도 0.8에서 감소하지 않다가 10%정도 학습했을 때부터 loss 증가

```python
from qiskit.test.mock import FakeWashington
from qiskit.providers.aer import AerSimulator

device_backend = FakeWashington()
device_backend

sim_washington = AerSimulator.from_backend(device_backend)
```

```
Training [2%]   Loss: 0.8153
Training [4%]   Loss: 0.7953
Training [6%]   Loss: 0.7713
Training [8%]   Loss: 0.7993
Training [10%]  Loss: 0.7893
Training [12%]  Loss: 0.8253
Training [14%]  Loss: 0.8553
Training [16%]  Loss: 0.8473
Training [18%]  Loss: 0.8013
Training [20%]  Loss: 0.8433
```

- **Qiskit runtime**
  - 더 빠르다고 했는데 동일한 회로로 실험했을 때 더 느림..

```python
backend = BasicAer.get_backend('qasm_simulator') # the

start = time.time()

result = execute(bell, backend, shots=2000).result()
counts  = result.get_counts(bell)

print("time : ", time.time()-start)

print("counts:", counts)
```

```
time :  0.015025138854980469
counts: {'00': 1002, '11': 998}
```

```python
with Sampler(circuits=bell, service=service, options={ "backend": "ibmq_qasm_simulator" }) as sampler:

    start = time.time()

    result1 = sampler(circuit_indices=[0], shots=2000)
    print("time : ", time.time()-start)

    print(result1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: __call__ keyword argument
  """
time :  6.578749418258667
SamplerResult(quasi_dists=[{'11': 0.4855, '00': 0.5145}], metadata=[{'header_metadata': {}, 'shots': 2000}])
```

- **Qiskit 메모리는 8GB이상 쓸 수 없어서 로컬 사용 권장**
  - Colab에 Qiskit 설치 후 실행했더니 더 빠름
  - 원래 1시간 이상 걸리던 작업이 20분 정도면 가능

Thank you for your attention.