Quantum Neural Network 회로 구성

https://youtu.be/CVY30tzUm40

QNN과 Quantum gate

- QNN을 구성하는 Parameterized Quantum Circuit에서는 다음과 같은 게이트들이 주로 사용
- H : 중첩 상태로 만들기 위함
- Ry : y축 기준 큐비트 회전 연산 (큐비트의 상태를 바꿈)
  - z축에 대한 값이 변화 → 0 또는 1이 될 결과 확률 (확률 진폭의 제곱)이 변함
  - Rx도 동일한 확률이 나오지만, Ry는 실수 확률 진폭을 갖고, Rx는 복소수 확률 진폭을 가짐 ; Ry가 더 주로 사용되는 듯
- X : NOT 연산 → 0 또는 1이 될 확률을 바꾸는 의미
- Ry, Rx, X 등의 게이트들은 Control qubit을 통해 얽힘 상태로 구성됨 → 큐비트 상태를 제어하므로 매우 중요..
  → 이를 통해 회로를 조건부 확률처럼 제어할 수도 있음
  → CX, CCX, CRy, CCRy 등

큐비트 측정

- **측정**
  - 큐비트를 얽힘 상태로 두고 마지막 큐비트만을 측정하여 해당 값을 실제 Label과 비교하는 경우
  - 큐비트를 얽힘 상태로 두고 모든 큐비트를 측정하는 경우
  - **적어도 하나는 측정해야 함**
  - 큐비트에 입력데이터를 인코딩한 후, 양자 게이트를 적용하고 측정

    → 즉, **입력 데이터가 회로의 게이트를 거친 후 측정까지 하고, 해당 값이 정답 label이 나오도록**
  - **Z-measure를 하는 이유?**

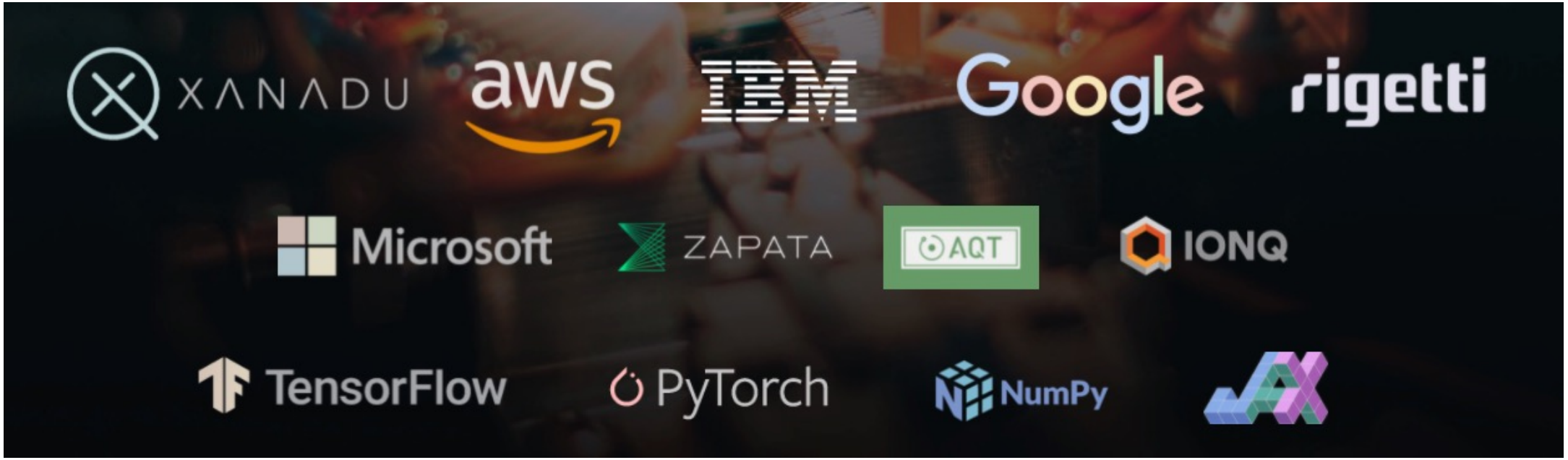
    Z축을 기저로 하여 측정할 경우, 0 또는 1로 결정

회로 구성 방법

- 회로를 구성하는 게이트나 데이터 인코딩 방식은 논문마다 다름
  - **실험을 통해 게이트들을 직접 구성**
  - pennylane과 같은 프레임워크를 활용하여 **랜덤 회로 또는 주어진 회로 사용**

PennyLane

- 양자 및 하이브리드 양자 고전 신경망의 최적화를 위한 Python 3 소프트웨어 프레임워크
- 다양한 플러그인 제공
- Tensorflow, Pytorch, Cirq, Qiskit, IBM Q, Amazon braket 등과 호환 가능
- TF, Pytorch의 함수 사용 가능, Keras layer로 작성 가능
- 회로를 분할하여 병렬처리 가능, 여러 회로 통합 가능, 다른 디바이스에서 동시 수행 가능 등 다양한 기능



Pennylane - 간단한 회로 구성

```python
import pennylane as qml
from pennylane import expval
from pennylane.optimize import GradientDescentOptimizer

dev = qml.device('default.qubit', wires=1)

@qml.qnode(dev)
def circuit1(var):
    qml.RX(var[0], wires=0)
    qml.RY(var[1], wires=0)
    return expval(qml.PauliZ(0))

opt = GradientDescentOptimizer(0.25)

var = [0.1, 0.2]
for it in range(30):
    var = opt.step(circuit1, var)
    print("Step {}: cost: {}".format(it, circuit1(var)))
```

Pennylane - 필요한 기능

```
def loss(labels, predictions):
    # Compute loss
    ...

def regularizer(var):
    # Compute regularization penalty
    ...

def statepreparation(x):
    # Encode x into the quantum state
    ...
```

```
def layer(W):
    # Layer of the model
    ...

def circuit3(x, weights):
    # Encode input x into quantum state
    statepreparation(x)
    # Execute layers
    for W in weights:
        layer(W)
    return ... # Return expectation(s)

def model(x, var):
    weights = var[0]
    bias = var[1]
    return circuit3(x, weights) + bias

def cost(var, X, Y):
    # Compute prediction for each input
    preds = [model(x, var) for x in X]
    # Compute the cost
    loss = loss(Y, preds)
    regul = regularizer(var)
    return loss + 0.01 * regul
```

Pennylane - 코드

- Qiskit + pennylane

Pennylane – keras와의 호환

```
0: ──╭AngleEmbedding(M0)──╭BasicEntanglerLayers(M1)──┤ ⟨Z⟩
1: ──╰AngleEmbedding(M0)──╰BasicEntanglerLayers(M1)──┤ ⟨Z⟩
```

```python
import pennylane as qml

n_qubits = 2
dev = qml.device("default.qubit", wires=n_qubits)

@qml.qnode(dev)
def qnode(inputs, weights):
    qml.AngleEmbedding(inputs, wires=range(n_qubits))
    qml.BasicEntanglerLayers(weights, wires=range(n_qubits))
    #print(weights)
    #drawer = qml.draw(qnode)
    #print(drawer(inputs, weights))
    return [qml.expval(qml.PauliZ(wires=i)) for i in range(n_qubits)]
```

데이터 인코딩

레이어

```
0: ──╭AngleEmbedding(M0)──╭RandomLayers(M1)──┤ ⟨Z⟩
1: ──╰AngleEmbedding(M0)──╰RandomLayers(M1)──┤ ⟨Z⟩
```

```python
import pennylane as qml

n_qubits = 2
dev = qml.device("default.qubit", wires=n_qubits)

@qml.qnode(dev)
def qnode(inputs, weights):
    qml.AngleEmbedding(inputs, wires=range(n_qubits))
    qml.RandomLayers(weights, wires=range(n_qubits))
    #print(weights)
    #drawer = qml.draw(qnode)
    #print(drawer(inputs, weights))
    return [qml.expval(qml.PauliZ(wires=i)) for i in range(n_qubits)]
```

Pennylane – keras와의 호환

```
n_layers = 6
weight_shapes = {"weights": (n_layers, n_qubits)}


qlayer = qml.qnn.KerasLayer(qnode, weight_shapes, output_dim=n_qubits)
```

```
clayer_1 = tf.keras.layers.Dense(2)
clayer_2 = tf.keras.layers.Dense(2, activation="softmax")
model = tf.keras.models.Sequential([clayer_1, qlayer, clayer_2])
```

```
clayer_1 = tf.keras.layers.Dense(2)
clayer_2 = tf.keras.layers.Dense(2, activation="softmax")
model = tf.keras.models.Sequential([qlayer])
```

```
opt = tf.keras.optimizers.SGD(learning_rate=0.2)
model.compile(opt, loss="mae", metrics=["accuracy"])
```

Pennylane – keras와의 호환

- 양자 회로만 사용할 수도 있고, hybrid 형태로 사용 가능
- 주어진 간단한 데이터(parity bit)로 실험한 결과 (basic entangler layers 사용),
  고전 레이어를 출력층으로 사용한 경우가 양자 회로만 사용한 것보다 낮은 loss 및 높은 정확도 달성

With classical layer

```
Epoch 1/6
30/30 - 9s - loss: 0.4367 - accuracy: 0.6733 - val_loss: 0.3004 - val_accuracy: 0.8000 - 9s/epoch - 314ms/step
Epoch 2/6
30/30 - 9s - loss: 0.2712 - accuracy: 0.7867 - val_loss: 0.2226 - val_accuracy: 0.8400 - 9s/epoch - 312ms/step
Epoch 3/6
30/30 - 10s - loss: 0.2066 - accuracy: 0.8267 - val_loss: 0.1947 - val_accuracy: 0.8000 - 10s/epoch - 321ms/step
Epoch 4/6
30/30 - 9s - loss: 0.1830 - accuracy: 0.8533 - val_loss: 0.1794 - val_accuracy: 0.8400 - 9s/epoch - 312ms/step
Epoch 5/6
30/30 - 9s - loss: 0.1693 - accuracy: 0.8600 - val_loss: 0.1730 - val_accuracy: 0.8600 - 9s/epoch - 311ms/step
Epoch 6/6
30/30 - 9s - loss: 0.1529 - accuracy: 0.8667 - val_loss: 0.1935 - val_accuracy: 0.8400 - 9s/epoch - 315ms/step
```

Only Quantum layer

```
Epoch 1/6
30/30 - 9s - loss: 0.5467 - accuracy: 0.5333 - val_loss: 0.3565 - val_accuracy: 0.7400 - 9s/epoch - 300ms/step
Epoch 2/6
30/30 - 9s - loss: 0.3474 - accuracy: 0.7467 - val_loss: 0.3397 - val_accuracy: 0.8000 - 9s/epoch - 295ms/step
Epoch 3/6
30/30 - 9s - loss: 0.3448 - accuracy: 0.7667 - val_loss: 0.3559 - val_accuracy: 0.7000 - 9s/epoch - 297ms/step
Epoch 4/6
30/30 - 9s - loss: 0.3525 - accuracy: 0.7600 - val_loss: 0.3750 - val_accuracy: 0.7800 - 9s/epoch - 302ms/step
Epoch 5/6
30/30 - 9s - loss: 0.3515 - accuracy: 0.7200 - val_loss: 0.3534 - val_accuracy: 0.7200 - 9s/epoch - 300ms/step
Epoch 6/6
30/30 - 9s - loss: 0.3503 - accuracy: 0.7800 - val_loss: 0.3612 - val_accuracy: 0.7400 - 9s/epoch - 302ms/step
```

입력 데이터 인코딩 회로

- Angle Embedding

  입력 데이터를 Rotation 게이트의 회전 각으로 사용 (Rx, Ry, Rz, default = Rx)

```
qml.AngleEmbedding(inputs, wires=range(n_qubits))
```

- Amplitude Embedding

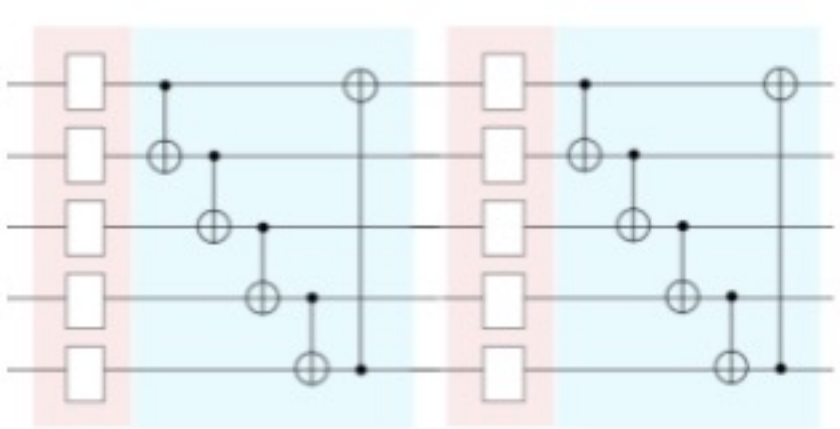
  2^n개의 입력 데이터를 n개의 큐비트에 임베딩

  벡터 표현을 위해 입력 feature는 자동 패딩 및 정규화 됨

  → 큐비트가 2개, 입력 데이터가 2개면 입력데이터를 4차원으로 zero-padding

Layers

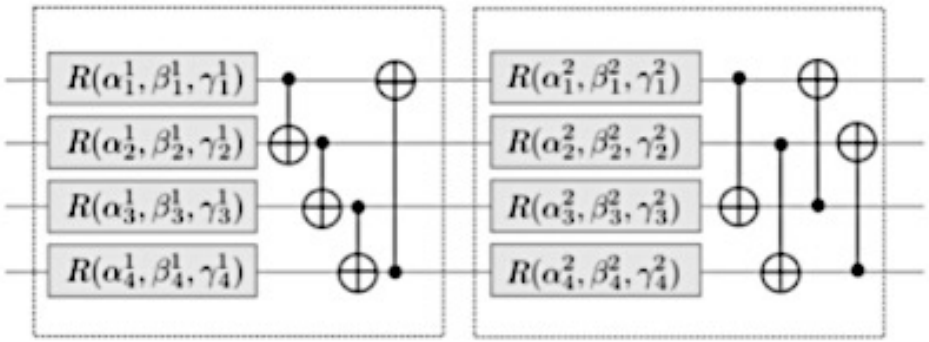
- Basic Entangler Layers



```
qml.BasicEntanglerLayers(weights, wires=range(n_qubits))
```

- 매개변수
  가중치(회전 각)와 큐비트(wire), 회전게이트 (default = Rx)
- 이웃 큐비트들끼리 연결하고, 마지막 큐비트는 첫번째 큐비트와 얽힘

- Strongly Entangler Layers *



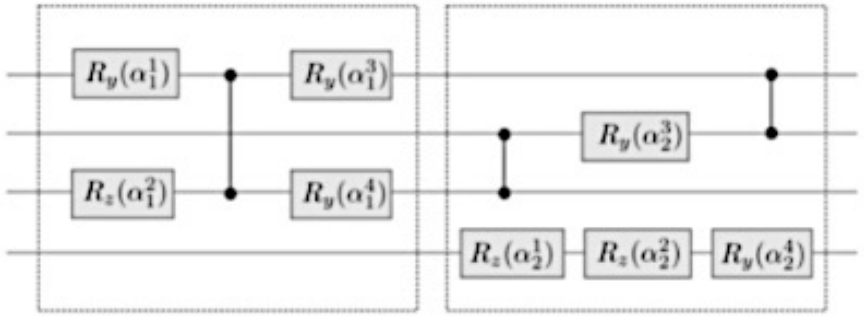
r=1 r=2

- 매개변수
  가중치(회전 각)와 큐비트(wire), r(레이어 반복 수),
  얽힘 게이트 (default = CNOT),
- r값만큼
  - 레이어 반복
  - r번째 레이어에서의 얽힘
  → i번째 큐비트와 (i+r mod M(큐비트 수)) 번째 큐비트

*Maria Schuld, Alex Bocharov, Krysta M. Svore, and Nathan Wiebe Phys. Rev. A 101, 032308 – Published 6 March 2020. 기반

Layers

- Random Layers



- 매개변수

  가중치(회전 각)와 큐비트(wire), 얽힘 게이트 비율
- 게이트 및 큐비트 무작위 구성

  → 단일 큐비트 회전 및 두 개의 큐비트에 대한 얽힘 회전

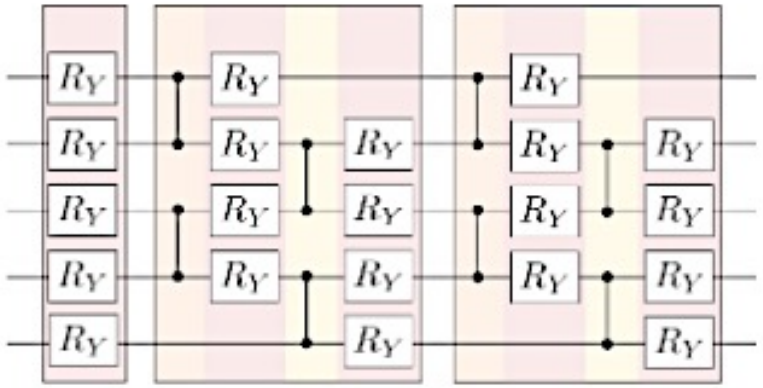
  → 얽힘 게이트 비율이 0.3이면,

  전체 게이트가 30개일 때, 10개는 얽힘 회전 게이트로 설정

```
qml.RandomLayers(weights, wires=range(n_qubits))
```

Simplified Two Design

- Simplified Two Design



- 매개변수

  초기 가중치, 가중치(회전 각)와 큐비트(wire)
- 초기 레이어 :Ry
- 이후 레이어

  - Ry, Controlled Z로만 구성
  - 각 큐비트 당 Ry 게이트 쌍 하나씩 (게이트 쌍에 대해 동일한 가중치)
  - 즉, M-1개의 가중치 쌍 가짐

    → 가중치는 (레이어 개수, M-1, 2)의 형태

*Cerezo, M., Sone, A., Volkoff, T. et al. Cost function dependent barren plateaus in shallow parametrized quantum circuits. Nat Commun 12, 1791 (2021). 에서 제안

관련 논문 간단한 소개

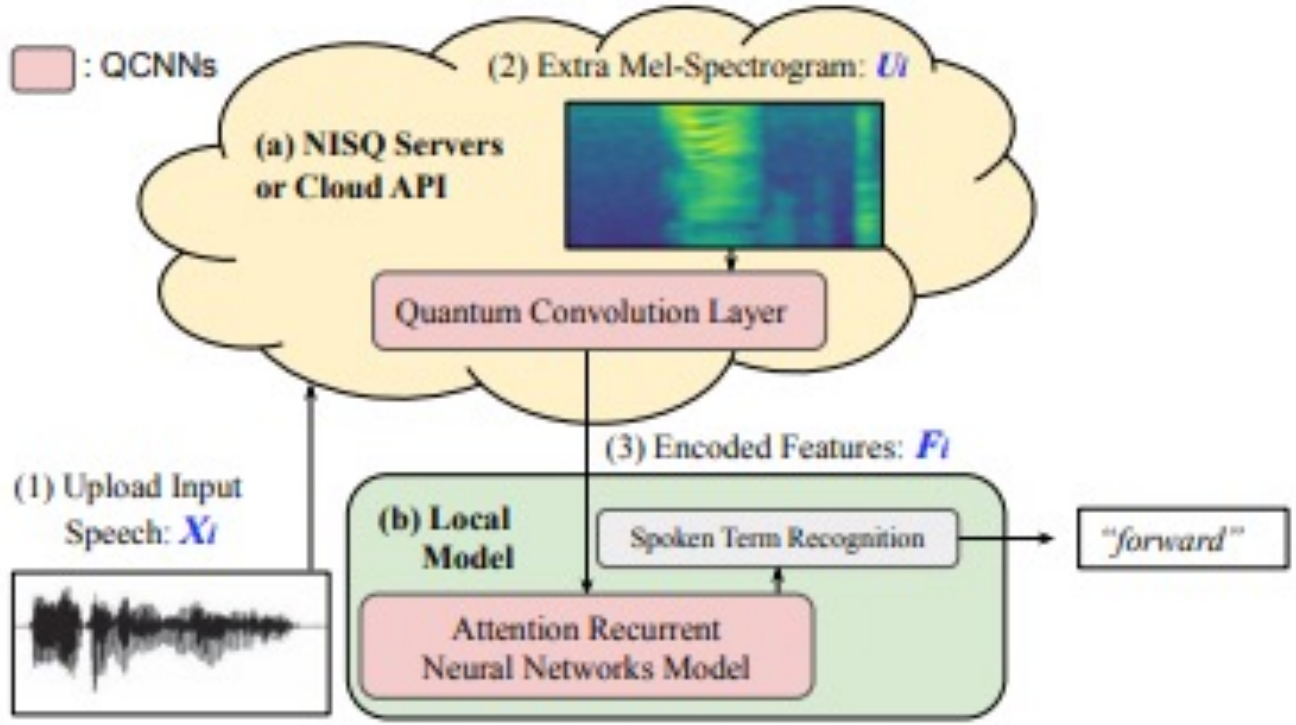
DECENTRALIZING FEATURE EXTRACTION WITH QUANTUM CONVOLUTIONAL NEURAL NETWORK FOR AUTOMATIC SPEECH RECOGNITION

Chao-Han Huck Yang[1] *Jun Qi*[1] *Samuel Yen-Chi Chen*[2] *Pin-Yu Chen*[3]
Sabato Marco Siniscalchi[1,4,5] *Xiaoli Ma*[1] *Chin-Hui Lee*[1]

[1]School of Electrical and Computer Engineering, Georgia Institute of Technology, USA
[2]Brookhaven National Laboratory, NY, USA and [3]IBM Research, Yorktown Heights, NY, USA
[4]Faculty of Computer and Telecommunication Engineering, University of Enna, Italy
[5]Department of Electronic Systems, NTNU, Trondheim, Norway

1. 연합 학습 (데이터 프라이버시 보장 위해 cloud와 local로 나누어 학습)
2. 양자 컨볼루션 신경망 사용 (cloud), 시계열 신경망 사용 (local)
3. 음성 인식 (10개의 label)
4. QCNN을 통한 음성 데이터 특징 추출 / RNN을 통한 분류 작업

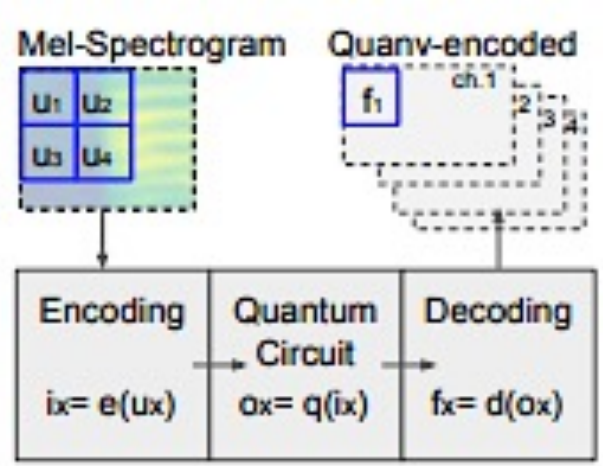
시스템 구성도


: QCNNs
(2) Extra Mel-Spectrogram: Ui
(a) NISQ Servers or Cloud API
Quantum Convolution Layer
(3) Encoded Features: Fi
(1) Upload Input Speech: Xi
(b) Local Model
Spoken Term Recognition
"forward"
Attention Recurrent Neural Networks Model

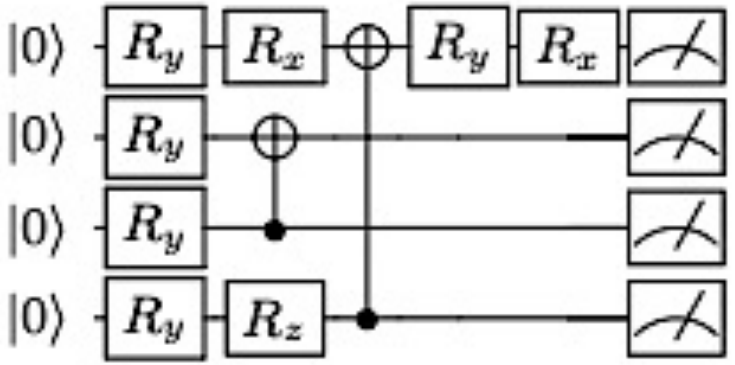
회로 구성

양자 회로를 구성하여 양자 커널(고전 CNN에서의 filter)로 사용

→ 입력 데이터의 특징을 추출



(a) QCNN Computing Process.



(b) Deployed Quantum Circuit.

$$f_i = \mathbf{Q}(\mathbf{u}_i, \mathbf{e}, \mathbf{q}, \mathbf{d}), \quad \text{where} \quad \mathbf{u}_i = \text{Mel-Spectrogram}(\mathbf{x}_i).$$

fx : 인코딩 된 feature
q : 양자 회로 파라미터
x : 입력 데이터
e : 인코딩
d : 디코딩 (measurement)

Phase 1: $\Phi_1 = R_y|0\rangle R_y|0\rangle R_y|0\rangle R_y|0\rangle.$
Phase 2: $\Phi_2 = (R_x R_y|0\rangle)\text{CNOT}(R_y|0\rangle)R_y|0\rangle R_z R_y|0\rangle.$
Phase 3: $\Phi_3 = \text{CNOT}((R_x R_y|0\rangle))\text{CNOT}(R_y|0\rangle)R_y|0\rangle R_z R_y|0\rangle.$
Phase 4: $\Phi_4 = R_x R_y \Phi_3$

이 때 사용한 회로는 Pennylane에서 배포한 랜덤 회로를 사용

실험 결과

- 실험 결과 분석은 음성 신호 관련이라 생략
- 고전 신경망만 사용한 것보다 QCNN을 사용하여 특징을 추출하였을 때 더 높은 정확도를 얻음
- 큐비트 관련 분석

  3x3 커널(9개의 큐비트)를 사용할 때가 2x2 보다 성능이 안 좋음

  → 오류 발생으로 인한 결과로 생각됨
- 큐비트에는 오류가 발생하게 됨

  그러나 현재 NISQ에서는 오류 정정까지는 어려운 상황이므로 커널이 크다고 좋은 성능을 얻지는 않음

향후 계획

- 양자 게이트나 얽힘, 측정 관련한 개념에 대해 좀 더 공부
- 데이터 인코딩 부분 세부사항 공부
- 간단한 고전 암호 데이터부터 원하는 형태로 회로 구성해볼 계획

  → Amplitude Embedding 적용해볼 생각… (2^n개의 데이터를 n개의 큐비트로 표현하는 방식)

감사합니다.