INTRODUCTION TO CLASSICAL MACHINE LEARNING AND NEURAL NETWORKS

Marco Maronese - PhD student @ University of Bologna
• Machine Learning Introduction
• Classical Support Vector Machine (SVM)
• Neural Networks
Machine Learning Introduction
MACHINE LEARNING

Concept

- Manual
- Machine learning
- Rule-based

Complexity of rules: simple to complex
Problem scale: small to large
Branches of machine learning

**Supervised**
- Labeled data
- Rule
  - Data
  - Answers

**Unsupervised**
- Correlations
- Rule
  - Data

**Reinforced**
- Strategy
- Environment
  - Actions
  - Answers
  - Rule
MACHINE LEARNING

Branches of machine learning

Supervised
- Classification
- Regression

Unsupervised
- Clustering
- Data Generation

Reinforced
- Game theory
- Robotics
SUPERVISED LEARNING

APPLICATIONS

• Identification of faces in images
• Identification of pedestrian
• Classification of texts
• Bioinformatics research
• Classification of remotely sensed images
SUPERVISED LEARNING

Classification Problem

Classification Rule

\[ y = \begin{cases} 
1, & \text{if } f(x, \theta) > \text{threshold} \\
0, & \text{if } f(x, \theta) \leq \text{threshold} 
\end{cases} \]
SUPERVISED LEARNING

Classification Problem

Classification Rule

\[ y = \begin{cases} 
1, & \text{if } f(x, \Theta) > \text{threshold} \\
0, & \text{if } f(x, \theta) \leq \text{threshold}
\end{cases} \]
SUPERVISED LEARNING

Classification Problem

Classification Rule

\[ y = \begin{cases} 
1, & \text{if } f(x, \theta) > \text{threshold} \\
0, & \text{if } f(x, \theta) \leq \text{threshold} 
\end{cases} \]

Support Vector Machine Rule

\[ y = \text{sign}(\varphi(\tilde{x}) \cdot \tilde{w} + b) \]
SUPERVISED LEARNING

Classification Problem

Classification Rule

\[
y = \begin{cases} 
1, & \text{if } f(x, \theta) > \text{threshold} \\
0, & \text{if } f(x, \theta) \leq \text{threshold}
\end{cases}
\]

Support Vector Machine Rule

\[
y = \text{sign}\left(\tilde{\varphi}(\tilde{x}) \cdot \tilde{w} + b\right)
\]

Non linear feature map
Classical Support Vector Machine (SVM)
Support Vector Machine

Overview

Support vector

\[ x_i \text{ such that } y_i (x_i \cdot w - b) = 1 \]

Margin width

\[ \frac{b + 1}{\|w\|} - \frac{b - 1}{\|w\|} = \frac{2}{\|w\|} \]
SUPPORT VECTOR MACHINE

Overview

Objective:

\[ \text{minimize} \quad \| w \| \]

Constraints

\[ x_i \cdot w - b \leq -1 \quad \text{for} \quad y_i = -1 \]
\[ x_i \cdot w - b \geq 1 \quad \text{for} \quad y_i = 1 \]
SUPPORT VECTOR MACHINE

Overview

Objective:
minimize $\|w\|$

Constraints
$y_i (x_i \cdot w - b) - 1 \geq 0 \quad \forall i$
SUPPORT VECTOR MACHINE

Optimization

\[ L_P \equiv \frac{1}{2} \|w\|^2 - \sum_{i=1}^{l} \alpha_i \left[ y_i (x_i \cdot w + b) - 1 \right] \]
SUPPORT VECTOR MACHINE

Optimization

\[ L_P \equiv \frac{1}{2} \| w \|^2 - \sum_{i=1}^{l} \alpha_i [y_i (x_i \cdot w + b) - 1] \]
SUPPORT VECTOR MACHINE

Optimization

\[ L_P \equiv \frac{1}{2} \|w\|^2 - \sum_{i=1}^{l} \alpha_i \left[ y_i (x_i \cdot w + b) - 1 \right] \]

\[ \frac{\partial L_P}{\partial w} = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{l} \alpha_i y_i x_i \]

\[ \frac{\partial L_P}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{l} \alpha_i y_i = 0 \]

\[ L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \]
Non-linear case

Input space

\[ x_i = (x_{0,i}, x_{1,i}, \ldots) \]

Feature space

\[ \varphi(x_i) = (x_{0,i}, x_{1,i}, \ldots, \phi(x_{0,i}, x_{1,i}, \ldots)) \]
SUPPORT VECTOR MACHINE

Non-linear case

Input space

\[ x_i = (x_{0,i}, x_{1,i}, \ldots) \]

Feature space

\[ \varphi(x_i) = (x_{0,i}, x_{1,i}, \ldots, \phi(x_{0,i}, x_{1,i}, \ldots)) \]

Feature map
SUPPORT VECTOR MACHINE

Non-linear case

\[ L_P = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{l} \alpha_i \left[ y_i \left( \phi(x_i) \cdot w + b \right) - 1 \right] \]

\[ \frac{\partial L_P}{\partial w} = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i) \]

\[ \frac{\partial L_P}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{l} \alpha_i y_i = 0 \]

\[ L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i) \cdot \phi(x_j) \]
Non-linear case

\[ L_P = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{l} \alpha_i \left[ y_i \left( \varphi(x_i) \cdot w + b \right) - 1 \right] \]

\[ \frac{\partial L_P}{\partial w} = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{l} \alpha_i y_i \varphi(x_i) \]

\[ \frac{\partial L_P}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{l} \alpha_i y_i = 0 \]

\[ L_D = \sum_{i} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \varphi(x_i) \cdot \varphi(x_j) \]

Kernel matrix \( K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j) \)
Neural Networks
$F(x) = \varphi \left( \sum_{i=0}^{m} x_i w_i + b \right) = \begin{cases} 1, & \text{if } \sum_{i=0}^{m} x_i w_i + b > 0 \\ 0, & \text{otherwise} \end{cases}$
Hornik theorem

\[ F(x) = \sum_{i=1}^{N} w_i' \varphi (w_i \cdot x + b_i) \]

- 1 output
- 1 hidden layer
- N hidden neurons
Deep supervised learning

Before Training: Random parameters
Before Training: Random parameters
Deep supervised learning

Before Training: Random parameters

**LOSS function**

The distance between the neural network predictions and the labels of the training set

- **MSE**
  \[ \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \]

- **Cross-Entropy**
  \[ \mathcal{L}(\theta) = - \sum_{i=0}^{N} \hat{y}_i \cdot \log(y_i) \]
Neural Networks

Deep supervised learning

Before Training: Random parameters

 LOSS function

The distance between the neural network predictions and the labels of the training set

- MSE

\[
\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2
\]

- Cross-Entropy

\[
\mathcal{L}(\theta) = - \sum_{i=0}^{N} \hat{y}_i \cdot \log(y_i)
\]
NEURAL NETWORKS

Deep supervised learning

Before Training: Random parameters

Loss function optimization

Gradient descent

$$\theta \leftarrow \theta - \alpha \nabla L(\theta)$$
Deep supervised learning

Before Training: Random parameters

Loss function optimization

Gradient descent:
\[ \theta \leftarrow \theta - \alpha \nabla L(\theta) \]

Backpropagation:
Compute the gradient by chain rule
Deep supervised learning

After Training: Optimal parameters

Loss function optimization

Gradient descent

\[ \theta \leftarrow \theta - \alpha \nabla L(\theta) \]

Backpropagation: Compute the gradient by chain rule

https://playground.tensorflow.org/
QUANTUM MACHINE LEARNING ON NISQ DEVICES

Marco Maronese – PhD student @ University of Bologna
INDEX

- Quantum Machine Learning on NISQ devices
- Quantum Support Vector Machine (QSVM)
- Quantum Neural Networks (QNN)
Quantum Machine Learning on NISQ devices
APPLICATIONS

- Drug discovery
- Finance
- Space
- Cybersecurity
QUANTUM MACHINE LEARNING

Possible advantage

Linear algebra

Operations on gate-model quantum computers follow the linear algebra rules

Quantum mechanics is intrinsically probabilistic

Explore more paths with quantum phenomena
QUANTUM MACHINE LEARNING

Overview

• First generation of QML
  o Accelerated linear algebra on quantum computers
  o Only applies to fault-tolerant quantum computers
QUANTUM MACHINE LEARNING

Overview

• First generation of QML
  o Accelerated linear algebra on quantum computers
  o Only applies to fault-tolerant quantum computers

• Second generation of QML
  o Low-depth quantum circuit learning (QNN)
  o Applicable to noisy-intermediate quantum (NISQ) devices
QUANTUM MACHINE LEARNING

Overview

• First generation of QML
  o Accelerated linear algebra on quantum computers
  o Only applies to fault-tolerant quantum computers

• Second generation of QML
  o Low-depth quantum circuit learning (QNN)
  o Applicable to noisy-intermediate quantum (NISQ) devices
QUANTUM MACHINE LEARNING

Frameworks

QISKIT

Most used quantum SDK with access of quantum devices
Machine learning modules

PENNYLANE

For hybrid quantum-classical computation
Integration of pytorch and tensorflow with different quantum SDK

TF QUANTUM

For hybrid quantum-classical computation
Equivalence between quantum and tensor operations
QUANTUM MACHINE LEARNING

NISQ application

Supervised learning:

Quantum Support Vector Machine (QSVM) for regression and classification

Unsupervised learning:

Quantum Generative Adversarial Network (QGAN) hybrid quantum-classical generative model

Reinforcement learning:

Reinforcement learning with QVC hybrid reinforcement learning algorithm
QUANTUM MACHINE LEARNING

Ideal blocks

- **Feature map**: Store the inputs in a quantum state
- **Variational circuit**: Learnable parameter circuit
- **Expectation value**: Measurements introducing non-linearity
QUANTUM MACHINE LEARNING

Encoding problems

- No QRAM to store data
- Encoding data generally hard
- Better if are intrinsically quantum such as quantum states of a system
Quantum Support Vector Machine (QSVM)
Use case

Breast cancer Wisconsin

https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/version/2
Use case

Breast cancer Wisconsin

Dataset with 2 feature

2 qubits to performe QSVM

https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/version/2
QUANTUM SUPPORT VECTOR MACHINE

Quantum advantage:

More complex feature map at low computational cost
QUANTUM SUPPORT VECTOR MACHINE

Intuition

Non-linear feature map

\[ \tilde{x} \mapsto |\Phi(\tilde{x})\rangle = U_{\Phi(\tilde{x})}|0\rangle^\otimes n \]

\[ U_{\Phi(\tilde{x})} = \exp\left( i \sum_{s \subseteq [n]} \phi_s(\tilde{x}) \prod_{j \in s} Z_j \right) \]
QUANTUM SUPPORT VECTOR MACHINE

Feature map

\[ U_{\Phi(\tilde{x})} = \exp \left( i \sum_{S \subseteq [n]} \phi_S(\tilde{x}) \prod_{j \in S} Z_j \right) \]

- First order expansion (1 or more qubits, no entanglement)

\[ S \in \{0, 1, \ldots, n - 1\} \quad \phi_i(x) = x_i \]

- Second order expansion (2 or more qubits, entanglement)

\[ S \in \{0, 1, \ldots, n - 1, (0, 1), (1, 2), \ldots, (n - 2, n - 1)\} \quad \phi_{(i,j)}(x) = (\pi - x_i)(\pi - x_j) \]
QUANTUM SUPPORT VECTOR MACHINE

- Variational quantum circuit

Using variational quantum eigensolver (VQE)

- Kernel matrix estimation

Using quantum support vector machine (QSV)
QUANTUM SUPPORT VECTOR MACHINE

Kernel estimation

\[ K(\vec{x}_i, \vec{x}_j) = \left| \langle \Phi(\vec{x}_i) | \Phi(\vec{x}_j) \rangle \right|^2 = \left| \langle 0 | U_{\Phi(\vec{x}_j)}^\dagger U_{\Phi(\vec{x}_i)} | 0 \rangle \otimes n \right|^2 \]

\[ p_{0...0} \sim K(\vec{x}_i, \vec{x}_j) \]
Quantum variational circuit

\[ p_y(x) = \langle \Phi(x) | W^\dagger(\theta) M \cdot W(\theta) | \Phi(x) \rangle \]

If \[ \hat{p}_y(x) > \hat{p}_{-y}(x) - yb \]

\[ \hat{m}(x) = y \]

Cost function

\[ \text{Pr}(\hat{m}(x) \neq m(x)) \]

Learnable parameters

\( (\theta, b) \)
QUANTUM SUPPORT VECTOR MACHINE

Varitional circuit ansatz

\[ W(\theta) \]

\[ \begin{align*}
  \text{qubit 1} & \langle 0 \rangle & U^{1,0}(\theta) & U^{1,1}(\theta) & U^{1,D}(\theta) \\
  \text{qubit 2} & \langle 0 \rangle & U^{2,0}(\theta) & U^{2,1}(\theta) & U^{2,D}(\theta) \\
  \text{qubit N} & \langle 0 \rangle & U^{N,0}(\theta) & U^{N,1}(\theta) & U^{N,D}(\theta)
\end{align*} \]

\[ U^0(\theta) \quad U^1(\theta) \quad U^D(\theta) \]

\[ D \text{- times} \]

\[ \langle \Psi(\theta) \rangle \]
QUANTUM SUPPORT VECTOR MACHINE

Quantum variational circuit

- Expectation value of an operator $f$

$$\langle \Phi(x) | W^\dagger(\theta) f W(\theta) | \Phi(x) \rangle = \frac{1}{2^n} \sum_{\alpha} w_{\alpha}(\theta) \Phi_{\alpha}(x)$$

- Classification rule

$$\tilde{m}(x) = \text{sign} \left( \frac{1}{2^n} \sum_{\alpha} w_{\alpha}(\theta) \Phi_{\alpha}(x) + b \right)$$
Quantum Neural Networks (QNN)
VARIATIONAL QUANTUM ALGORITHM

General idea

STEP 1: $x \rightarrow \varphi(x)$
- nonlin. map

STEP 2: $\varphi' = U_\theta \varphi(x) \rightarrow$
- unitary transform.

STEP 3: $\sum_{\{k\}} |(\varphi')_k|^2$

STEP 4: $\rightarrow y$
- post-process.

CIRCUIT

$S_x$
- state prep.

$U$
- model circuit

Measurement

$\rho = p(1)$

$\rightarrow y$
QUANTUM NEURAL NETWORK

**Definition**

- Sequence of continuously-parametrized rotations executed on QPU
- Measure observable expectation value (non-linearity)
- CPU optimization suggests new parameters to minimize the expectation value

[Diagram of quantum neural network]

[Definition link]: https://qml.entropicalabs.io/
QUANTUM NEURAL NETWORK

Problems

- Data must be prepared on the fly
- QPU needs full quantum program for each run (microseconds for each run)
- Relative high latency CPU-QPU (ms)
Hybrid approach
Classical optimization is used
Hybrid approach

Classical optimization is used

But the gradient is computed by QPU

Parameter shift

$$\partial_{\theta} f(\theta) = c[f(\theta + s) - f(\theta - s)]$$
QUANTUM NEURAL NETWORK

Hybrid approach
Classical optimization is used

But the gradient is computed by QPU

Parameter shift
\[
\partial_\theta f(\theta) = c[f(\theta + s) - f(\theta - s)]
\]

Exact
QUANTUM NEURAL NETWORK

Parameter shift

\[ x \xrightarrow{\text{quantum node}} \nabla f(x, \theta) \]

\[
\begin{array}{c}
|0\rangle -|0\rangle - U(x, \theta + s) - U(x, \theta - s)
\end{array}
\]

\[
|0\rangle -|0\rangle - U_N(\theta_N) \cdots U_{i+1}(\theta_{i+1}) \hat{B} U_{i+1}(\theta_{i+1}) \cdots U(\theta) \]

State

\[ |\psi_{i-1}\rangle = U_{i-1}(\theta_{i-1}) \cdots U_1(\theta_1) U_0(x) |0\rangle \]

Operator (Observable)

\[ \hat{B}_{i+1} = U_N^{\dagger}(\theta_N) \cdots U_{i+1}^{\dagger}(\theta_{i+1}) \hat{B} U_{i+1}(\theta_{i+1}) \cdots U(\theta) \]

Expectation value

\[ f(x; \theta) = \langle \psi_{i-1} | U_i(\theta_i) \hat{B}_{i+1} U_i(\theta_i) | \psi_{i-1} \rangle = \langle \psi_{i-1} | M_{\theta_i}(\hat{B}_{i+1}) | \psi_{i-1} \rangle \]
Parameter shift

\[ |\psi_{i-1}\rangle = U_{i-1}(\theta_{i-1}) \cdots U_1(\theta_1)U_0(x)|0\rangle \]

Operator (Observable)

\[ \hat{B}_{i+1} = U_N^\dagger(\theta_N) \cdots U_{i+1}^\dagger(\theta_{i+1}) \hat{B} U_{i+1}(\theta_{i+1}) \cdots U_N(\theta_N) \]

Expectation value

\[ f(x; \theta) = \langle \psi_{i-1}|U_i^\dagger(\theta_i)\hat{B}_{i+1}U_i(\theta_i)|\psi_{i-1}\rangle = \langle \psi_{i-1}|\mathcal{M}_\theta(\hat{B}_{i+1})|\psi_{i-1}\rangle. \]

\[ \nabla_{\theta_i} f(x; \theta) = \langle \psi_{i-1}|\nabla_{\theta_i} \mathcal{M}_\theta(\hat{B}_{i+1})|\psi_{i-1}\rangle \]
Quantum Neural Network

**Parameter shift**

\[ x \rightarrow \nabla f(x, \theta) \]

**State**

\[ |\psi_{i-1}\rangle = U_{i-1}(\theta_{i-1}) \cdots U_1(\theta_1)U_0(x)|0\rangle \]

**Operator (Observable)**

\[ \hat{B}_{i+1} = U_N^{\dagger}(\theta_N) \cdots U_{i+1}^{\dagger}(\theta_{i+1})\hat{B}U_{i+1}(\theta_{i+1}) \cdots U_N(\theta_N) \]

**Expectation value**

\[ f(x; \theta) = \langle \psi_{i-1}|U_i^{\dagger}(\theta_i)\hat{B}_{i+1}U_i(\theta_i)|\psi_{i-1}\rangle = \langle \psi_{i-1}|\mathcal{M}_{\theta_i}(\hat{B}_{i+1})|\psi_{i-1}\rangle \]

**Pauli rotation**

\[ U_i(\theta_i) = \exp\left(-i\frac{\theta_i}{2}\hat{P}_i\right) \]

\[ \nabla_{\theta_i} U_i(\theta_i) = -i\frac{1}{2}\hat{P}_i U_i(\theta_i) = -i\frac{1}{2} U_i(\theta_i)\hat{P}_i \]
State
\[ |\psi_{i-1}\rangle = U_{i-1}(\theta_{i-1}) \cdots U_1(\theta_1)U_0(x)|0\rangle \]

Operator (Observable)
\[ \hat{B}_{i+1} = U_N^\dagger(\theta_N) \cdots U_{i+1}^\dagger(\theta_{i+1}) \hat{B} U_{i+1}(\theta_{i+1}) \cdots U_N(\theta_N) \]

Expectation value
\[ f(x; \theta) = \langle \psi_{i-1} | U_i^\dagger(\theta_i) \hat{B}_i U_i(\theta_i) | \psi_{i-1} \rangle = \langle \psi_{i-1} | \mathcal{M}_{\theta_i}(\hat{B}_i) | \psi_{i-1} \rangle. \]

Pauli rotation
\[ U_i(\theta_i) = \exp\left(-i\frac{\theta_i}{2} \hat{P}_i\right) \]

\[ \nabla_{\theta_i} U_i(\theta_i) = -i \frac{\theta_i}{2} \hat{P}_i U_i(\theta_i) = -i \frac{\theta_i}{2} U_i(\theta_i) \hat{P}_i \]
Parameter shift

\[ \nabla f(x, \theta) = \nabla_x f(x, \theta) = \frac{\partial}{\partial x} f(x, \theta) \]

Commutator

\[ [X, Y] = XY - YX \]

\[ \nabla \theta_i f(x; \theta) = \frac{i}{2} \langle \psi_{i-1} | U_i^\dagger(\theta_i) \left[ P_i \hat{B}_{i+1} - \hat{B}_{i+1} P_i \right] U_i(\theta_i) | \psi_{i-1} \rangle \]

\[ = \frac{i}{2} \langle \psi_{i-1} | U_i^\dagger(\theta_i) \left[ P_i \hat{B}_{i+1} \right] U_i(\theta_i) | \psi_{i-1} \rangle, \]

\[ \nabla \theta_i U_i(\theta_i) = \exp \left( -i \frac{\theta_i}{2} \hat{P}_i \right) \]

State

\[ |\psi_{i-1} \rangle = U_{i-1}(\theta_{i-1}) \cdots U_1(\theta_1) U_0(x) |0 \rangle \]

Operator (Observable)

\[ \hat{B}_{i+1} = U_N^\dagger(\theta_N) \cdots U_{i+1}^\dagger(\theta_{i+1}) \hat{B} U_{i+1}(\theta_{i+1}) \cdots U_N(\theta_N) \]

Expectation value

\[ f(x; \theta) = \langle \psi_{i-1} | U_i^\dagger(\theta_i) \hat{B}_{i+1} U_i(\theta_i) | \psi_{i-1} \rangle = \langle \psi_{i-1} | M_{\theta_i}(\hat{B}_{i+1}) | \psi_{i-1} \rangle. \]

Pauli rotation

\[ U_i(\theta_i) = \exp \left( -i \frac{\theta_i}{2} \hat{P}_i \right) \]

\[ \nabla \theta_i U_i(\theta_i) = -i \hat{P}_i U_i(\theta_i) = -i \frac{1}{2} U_i(\theta_i) \hat{P}_i \]
QUANTUM NEURAL NETWORK

Parameter shift

\[ x \rightarrow \text{quantum node} \rightarrow \nabla f(x, \theta) \]

State

\[ |\psi_{i-1}\rangle = U_{i-1}(\theta_{i-1}) \cdots U_1(\theta_1)U_0(x)|0\rangle \]

Operator (Observable)

\[ \hat{B}_{i+1} = U_N(\theta_N) \cdots U_{i+1}^\dagger(\theta_{i+1}) \hat{B} U_{i+1}(\theta_{i+1}) \cdots U_N(\theta_N) \]

Expectation value

\[ f(x; \theta) = \langle \psi_{i-1} | U_i^\dagger(\theta_i) \hat{B}_i U_i(\theta_i) |\psi_{i-1}\rangle = \langle \psi_{i-1} | M_{\theta_i}(\hat{B}_{i+1}) |\psi_{i-1}\rangle \]

\[ \nabla_\theta f(x; \theta) = \frac{1}{2} \left[ f(x; \theta + \frac{\pi}{2}) - f(x; \theta - \frac{\pi}{2}) \right] \]
Variational circuit ansatz
Choose a compromise between
• Computational cost (depth)
• Expressiveness of the unitary operator
Variational circuit ansatz
Choose a compromise between
• Computational cost (depth)
• Expressiveness of the unitary operator

Idle circuit | Circuit A | Circuit B | Arbitrary unitary
---|---|---|---
$|0\rangle \rightarrow I$ | $|0\rangle \rightarrow H \rightarrow R_Z$ | $|0\rangle \rightarrow H \rightarrow R_Z \rightarrow R_X$ | $|0\rangle \rightarrow U$
QUANTUM NEURAL NETWORK
QUANTUM NEURAL NETWORK

• Circuits written with Cirq and converted into tensors
• Expectation value of an operator (OPs)
• Backpropagation for parameter optimization
TensorFlow Quantum

Classical Data: integers/floats/strings

TF Keras Models

TF Layers

TF Ops

TF Execution Engine

Quantum Data: Circuits/Operators

TFQ Layers

TFQ Differentiators

TFQ Ops

TFQ qsim

Cirq

Cirq (Programming framework)

Quantum Engine (Cloud service)

Quantum Hardware

Simulators

Open source

Proprietary

TPU

GPU

CPU

QPU

TensorFlow Quantum

OpenFermion

ReCirq

Qsim Simulator

Certifiable Random Number Generation

CINECA

QUANTUM COMPUTING LAB
EXAMPLES

QNN example 1

Quantum data input  Quantum model  Expectation

Classifier

P₁  P₂
EXAMPLES

QNN example 1

Quantum data input → Ry(α) → $\langle Z \rangle$ → Classifier

P₁, P₂

QNN example 2: mnist classification

Quantum data input → Quantum model → $\langle Z \rangle$
**BARREN PLATEAUS**

Pauli Rotation

\[
\exp\left\{\frac{iP\theta}{2}\right\} = I \cos \frac{\theta}{2} + iP \sin \frac{\theta}{2}
\]

A random variational circuit with gates chosen from the set \{RX, RY, RZ\}

Expectation value of a Hermitian observable along a slice in the parameter space
BARREN PLATEAUS

Pauli Rotation

\[ \exp\left(\frac{iP\theta}{2}\right) = I \cos \frac{\theta}{2} + iP \sin \frac{\theta}{2} \]

Periodic function → Saddle points

A random variational circuit with gates chosen from the set \{RX, RY, RZ\}

Expectation value of a Hermitian observable along a slice in the parameter space
In classical optimization, it is suggested that saddle points, not local minima, provide a fundamental impediment to rapid high-dimensional non-convex optimization. (Dauphin et al., 2014)

For a wide class of reasonable parameterized quantum circuits, the probability that the gradient along any reasonable direction is non-zero to some fixed precision is exponentially small as a function of the number of qubits. (McClean et al., 2018)
FUTURE PERSPECTIVES

- QRAM for data encoding
- Fault tolerant QC for better variational circuit ansatz
- More qubits

Third generation of QML
- No barren plateaus
- No heuristic model