Quantum Machine Learning

Topics of 24-Nov-2025

Eric Michiels
Quantum Computational Scientist
Qiskit Advocate

Learning Goals

- Explain what QML is and **where Quantum Computing connects to classical Machine Learning.**
  - ✓Apply Quantum **Vocabulary** and key terms to Machine Learning.
  - ✓Identify **key components of a QML workflow**.
- Identify **different types of QML** and distinguish between them.
  - ✓Implement **Quantum Kernel** methods and **Variational Quantum Classifiers**.
- Identify where QML is **most promising** and where it is **not**.
  - ✓Make recommendations for **where QML might benefit an organization**
  - ✓**Adjust** an Example problem to own Datasets.
- Be aware of **issues in QML** like *training time, noise, and errors*.

Topics

1. Introduction
2. Recap of Machine Learning
3. Data Encoding
4. Quantum Kernel methods and Support Vector Machines
5. Variational Quantum Classifiers and Neural Networks

Introduction

End of Topic 1

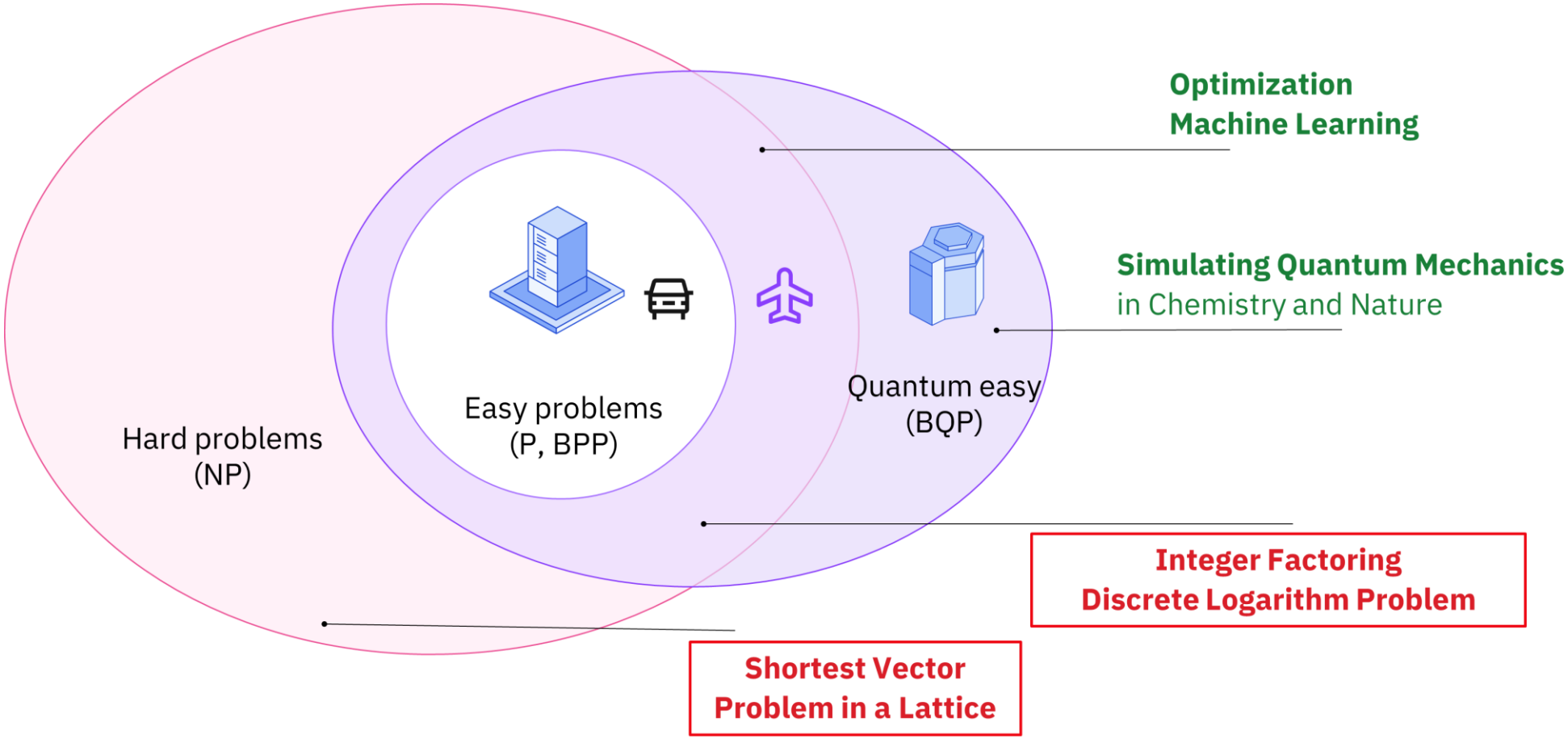
Motivation for Quantum Machine Learning

- Quantum Machine Learning or QML is one of many exciting areas by Quantum Computing that can **compliment existing Classical workflows**
- Our focus is on how Quantum Computing can **fit into existing Machine Learning workflows**
- We have seen a problem resolved for the first time on a Quantum Computer **and then** people find a way to do it on a **Classical** supercomputer!

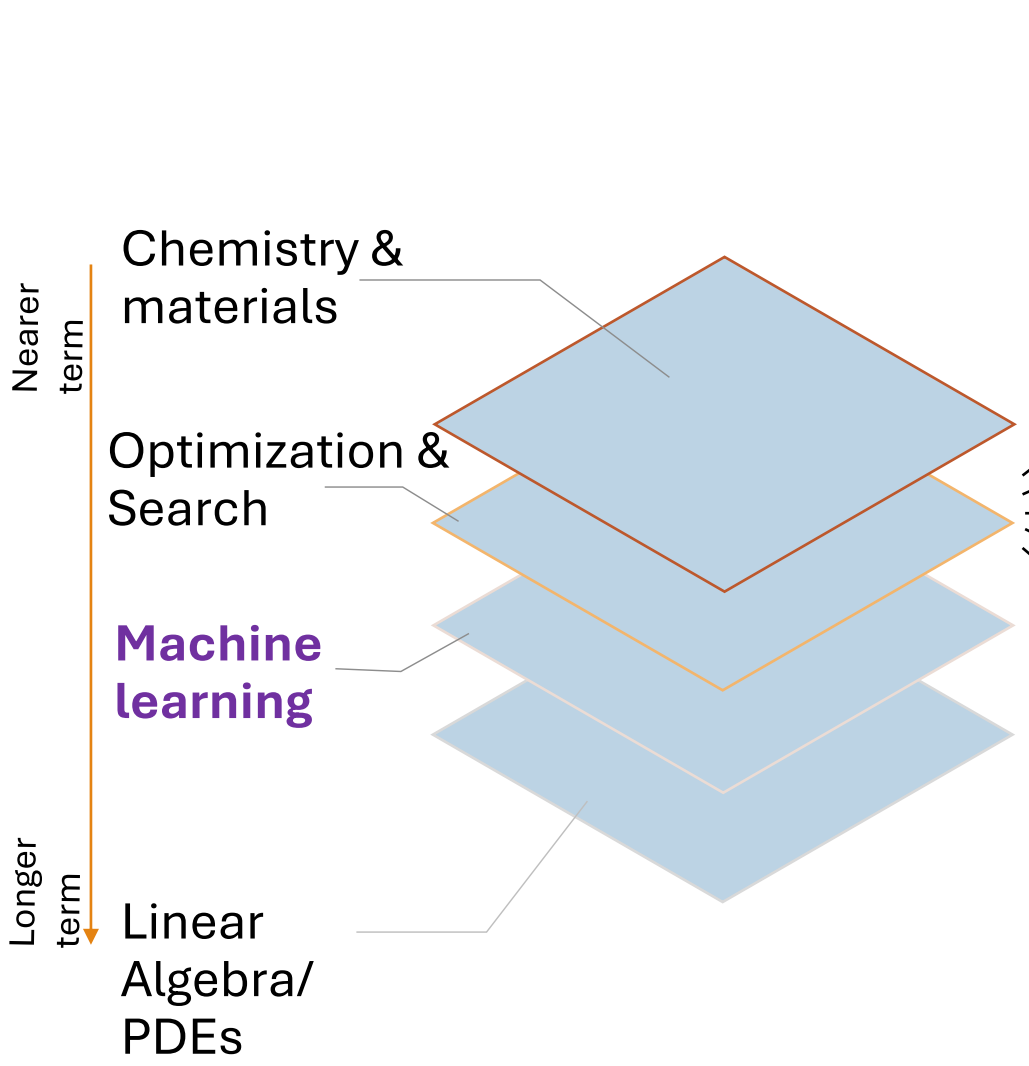
  ✓***In other words, Quantum Computing and Classical Computing are pushing each other to their limits.***

- **There are specific problems with Quantum Computing can have provable advantage over Classical Computing**
  - Thanks to progress in **Error Correction** and **number of available Qubits**.
- But this is **still a time of exploration**, searching for **Quantum Amenable Data Assets** and useful **Quantum Feature Maps**

Quantum and Classical Computing are Complementary



Quantum Computing Key Use Case Categories



Aerospace	High-performance materials Design optimization Aircraft design and simulation
Automotive	Advanced batteries Multi-objective vehicle routing optimization Auto design and simulation
Chemicals	Lower energy fertilizer manufacturing Efficient Catalysts Accelerated product development
Electronics & semiconductors	Product design Process development Finite element simulation
Energy	Risk of grid bottlenecks Grid optimization with EVs Carbon Sequestration
Financial Services	Portfolio optimization Derivatives/options pricing Anti-money laundering
Health Care & Life Sciences	Drug discovery and drug design optimization mRNA & protein structure predictions Disease target & biomarker discovery

(IBM) Quantum Development Roadmap

Scale + Speed + Quality + Stability

	2016–2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2033+
	Ran quantum circuits on IBM Quantum Platform	Released multi-dimensional roadmap publicly with initial focus on scaling	Enhanced quantum execution speed by 100x with Qiskit Runtime	Brought dynamic circuits to unlock more computations	Enhanced quantum execution speed by 5x with Quantum Serverless and execution modes	Demonstrated accurate execution of a quantum circuit at a scale beyond exact classical simulation (5K gates on 156 qubits)	Deliver quantum + HPC tools that will leverage Nighthawk, a new higher-connectivity quantum processor able to execute more complex circuits	Enable the first examples of quantum advantage using a quantum computer with HPC	Improve quantum circuit quality to allow 10K gates	Improve quantum circuit quality to allow 15K gates	Deliver a fault-tolerant quantum computer with the ability to run 100M gates on 200 logical qubits	Beyond 2033, quantum computers will run circuits comprising a billion gates on up to 2000 logical qubits, unlocking the full power of quantum computing
						Code assistant						
Applying algorithms to applications						Functions		Use case benchmarking toolkit	Computation libraries			
Discovering new algorithms for advantage						Advanced classical transpilation tools	Advanced classical mitigation tools	Utility mapping tools			Circuit libraries	
Orchestrating workloads for quantum + HPC					Resource Management							
					Qiskit Serverless	Plugins for HPC	C API	Profiling tools		Workflow accelerators		
					Execution modes							
Accurately and efficiently executing on quantum computers			Qiskit Runtime									
	IBM Quantum Experience		OpenQASM 3	Dynamic Circuits	Error mitigation	200K CLOPS	Utility-scale dynamic circuits				Fault-tolerant ISA	
Hardware →	Early	Falcon		Eagle		Heron (5K)	Nighthawk (5K)	Nighthawk (7.5K)	Nighthawk (10K)	Nighthawk (15K)	Starling (100M)	Blue Jay (1B)
	Canary 5 qubits Albatross 16 qubits Penguin 20 qubits Prototype 53 qubits	Benchmarking 27 qubits		Benchmarking 127 qubits		Error mitigation 5K gates 133 qubits	Error mitigation 5K gates 120 qubits	Error mitigation 7.5K gates 120 qubits Up to 120x3 = 360 qubits	Error mitigation 10K gates 120 qubits Up to 120x9 = 1080 qubits	Error mitigation 15K gates 120 qubits Up to 120x9 = 1080 qubits	Fault-tolerant 100M gates 200 logical qubits	Fault-tolerant 1B gates 2000 logical qubits

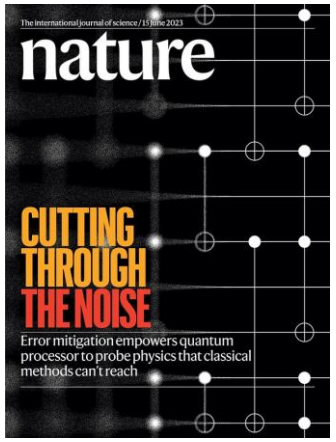
Completed
On target

Quantum Utility Computing

Quantum Advantage

Full Fault Tolerant Quantum Computers

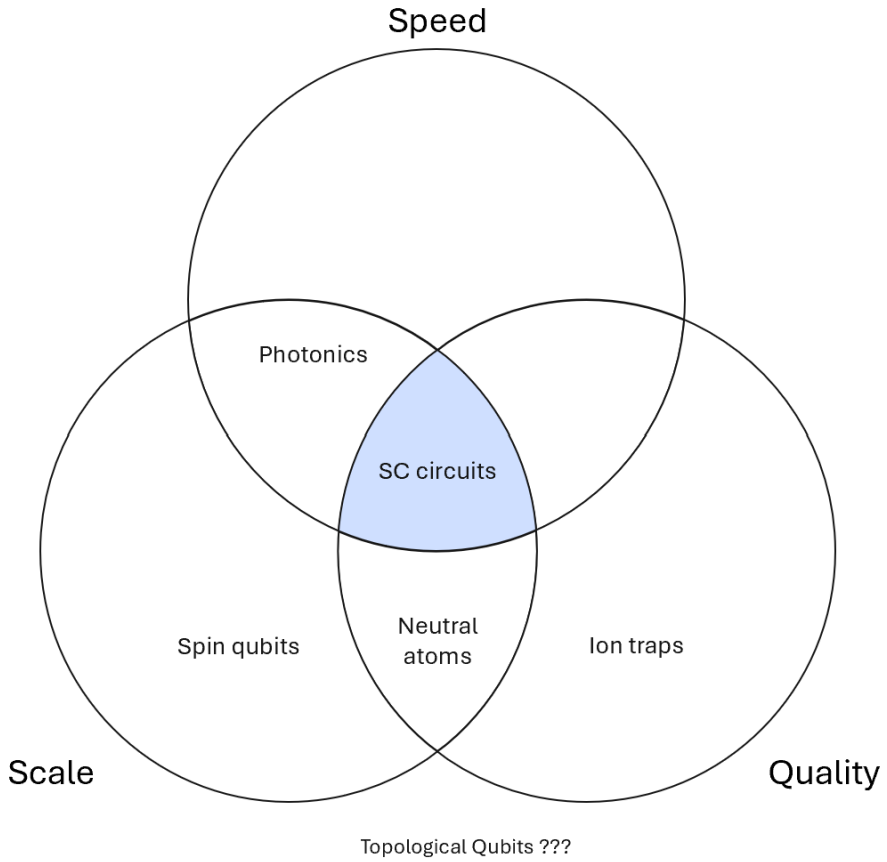
Quantum Error Mitigation, Error Correction

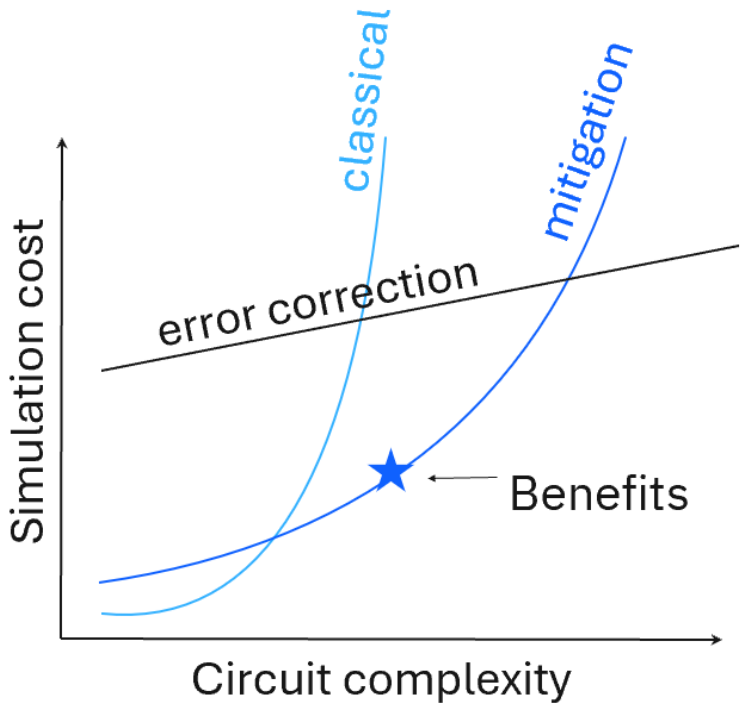


Dynamic Decoupling
Twirled Readout Extinction
Probabilistic Error Cancelation
Zero-Noise Extrapolation

Gross Code (qLDPC)

Quantum Computing Hardware Implementations





Focus of this Course Module

- We will restrict ourselves to applying **Quantum Algorithms to Classical Data = "CQ"**
- There are two reasons for this:
  1. Machine Learning problems with Classical Data are already **well studied**
  2. There is **no QRAM yet** - methods that begin with Quantum Data are still far from applicability to industry.
- Two types of Machine Learning of particular interest:
  - Supervised Learning
  - Unsupervised Learning

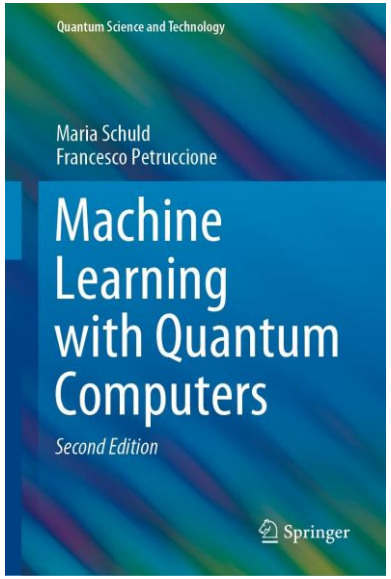
Positioning Quantum Machine Learning (QML)

Type of Data
Data Generating System

Type of Algorithm
Processing Device

Quantum \ Classical	Classical	Quantum
Classical	CC	**CQ**
Quantum	QC	QQ

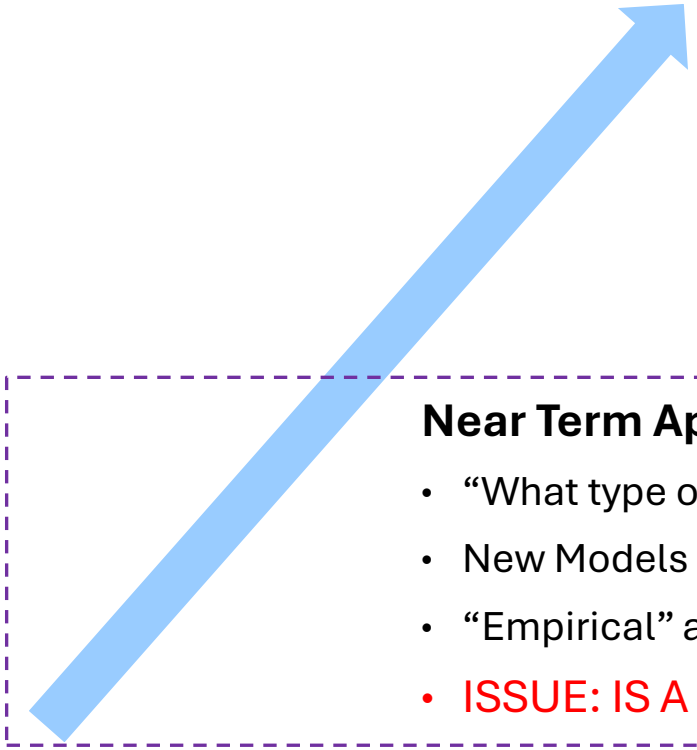
Maria Schuld, Francesco Petruccione
Machine Learning with Quantum Computers", Page 7



Caveats

- Any discussion of the usefulness of Quantum Computing in Machine Learning must be accompanied by the appropriate cautions:
  - The nuance in **Data Set Selection**
  - The **challenges** in reaching **Utility Scale**
  - **Shift away** from **trying to outperform Classical Machine Learning** algorithms on data that ***are already handled efficiently and well by Classical Algorithms***
  - Refocus to investigating **new types of Feature Maps** that can be useful

Near-Term and Fault-Tolerant Methods



Emerging Approach - Third Wave

- Based on deeper understanding of ML Potential of Quantum phenomena
- Combine the CML knowledge with Quantum Information Theory
- Train Quantum Models we cannot simulate any more …
- **QUANTUM INSPIRED METHODS**

> 2021

Near Term Approach - Second Wave

- "What type of ML Model fits the physical characteristics of a small-scale Quantum Device?"
- New Models and Algorithms derived
- "Empirical" and "Heuristic" mindset
- ISSUE: IS A CLASSICAL ML MINDSET SUFFICIENT… ?

> 2017

Long Term Benefit – First Wave

- Assumes Fault-Tolerant Quantum Computers
- Rather "Academic" and "Mathematical" mindset
- ISSUE: WE ONLY HAVE NISQ TODAY …

> 2013

Managing Expectations

- Many Data Sets used in QML applications are "Feature Engineered".

Is this cheating?

- No: **some Quantum Feature Maps behave differently from Classical Feature Maps.**

- The task at hand is to explore circuits in **the context of complex data structures**
- *Specific questions to be addressed*:
  - What **Quantum Circuits** are most likely to behave in **novel ways** compared to classical alternatives?
  - Are there real-world problems that **involve Data** with properties **best explored** using these novel **Quantum Circuits**?
  - Do these Quantum Circuits **Scale** on (near term) Quantum Computers?

Managing Expectations (Cont.)

- **High dimensionality** of entangled Quantum states is not exponential parallelism and is **not a sufficient condition** for increased power in Machine Learning
  - It is true that with 4 Qubits in superposition you get 16 states that exist simultaneously and that operations can be performed on this entire superposition
- **But this statement is inadequate:**
  - States in Quantum Computers are described by probability amplitude that can be complex numbers and not just real numbers.
  - Measurements eliminate the multiple states that can be taken at once
- **No Feature Map, no Algorithm** in this course is put forth as a *quick path* to better Machine Learning results for overall **general problems**
  - No such Feature Map, no such algorithm exists
- **We present Quantum Tools to be used in exploration of useful Quantum Computing**

Dequantization

- **Dequantization refers to the replacement of a given Quantum Algorithm with a classical algorithm that performs similarly for a given set of tasks.**
  - The Classical Algorithm performs only polynomial slower than the Quantum Algorithm
- *Several QML algorithms have been dequantized in recent years*.
  - This has led to important insights into the potential advantages and limitations of QML.
- ***Ewin Tang*** discovered a classical algorithms that could perform recommendation tasks at speeds previously thought to be achievable only by Quantum Computers
- Some dequantization cases have shown that **when efficient data encoding or data loading time is not included, the Quantum Algorithm no longer outperforms its classical counterpart**
- *And even if a Quantum Algorithm cannot be dequantized, that does not mean it is more efficient or scalable than all Classical Algorithms*

arXiv:1807.04271 (cs)

[Submitted on 10 Jul 2018 (v1), last revised 9 May 2019 (this version, v3)]

A quantum-inspired classical algorithm for recommendation systems

Ewin Tang

https://arxiv.org/abs/1807.04271

Dequantizing quantum machine learning models using tensor networks

Seongwook Shin, Yong Siah Teo,* and Hyunseok Jeong†

Department of Physics and Astronomy, Seoul National University, 08826 Seoul, South Korea

(Received 30 July 2023; accepted 18 April 2024; published 29 May 2024)

https://journals.aps.org/prresearch/pdf/10.1103/PhysRevResearch.6.023218

Proof of Existence

- Some scientists were able to show exponential **speed up using Quantum Kernel methods** trough judicious choice of the classification problem:
  - This classification problem was **carefully chosen.**
  - It is a classically **hard problem**.
  - The Quantum Algorithm shows a **speed up.**
  - The approach is end-to-end and it **includes data encoding time.**
  - The approach is **robust**: Data can be classified and separated by a **wide margin** using the Quantum Algorithm.

Article | Published: 12 July 2021

A rigorous and robust quantum speed-up in supervised machine learning

Yunchao Liu, Srinivasan Arunachalam & Kristan Temme

https://www.nature.com/articles/s41567-021-01287-z

Quantum Advantage Assessment

Proof of Existence (Cont.)

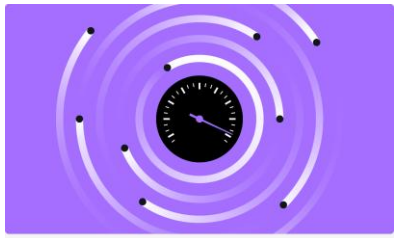
To state it in another way:

- Problems **do exist** in which Quantum Kernels can yield an exponential speed up
- It is **not** realistic to expect a Quantum speed up for Machine Learning tasks that classical computers **already** do quite well.
- **Identifying ideal cases for the exploration of Quantum utility** is an enormous responsibility that is for example taken care of in the IBM Quantum Network and in other Research Communities

Qiskit SDK

The **Lingua Franca** of Quantum Computing;
write once and execute quantum circuits on **10+** different hardware providers

- IBM Quantum
- AQT
- IQM
- Azure Quantum
- Alice & Bob
- IonQ
- Amazon Braket
- Superstaq
- Quantinuum
- Rigetti



Benchpress

Quantum SDK Preferred
(2024 Unitary Foundation Survey)

74%

Qiskit contributors that are external to IBM

74%

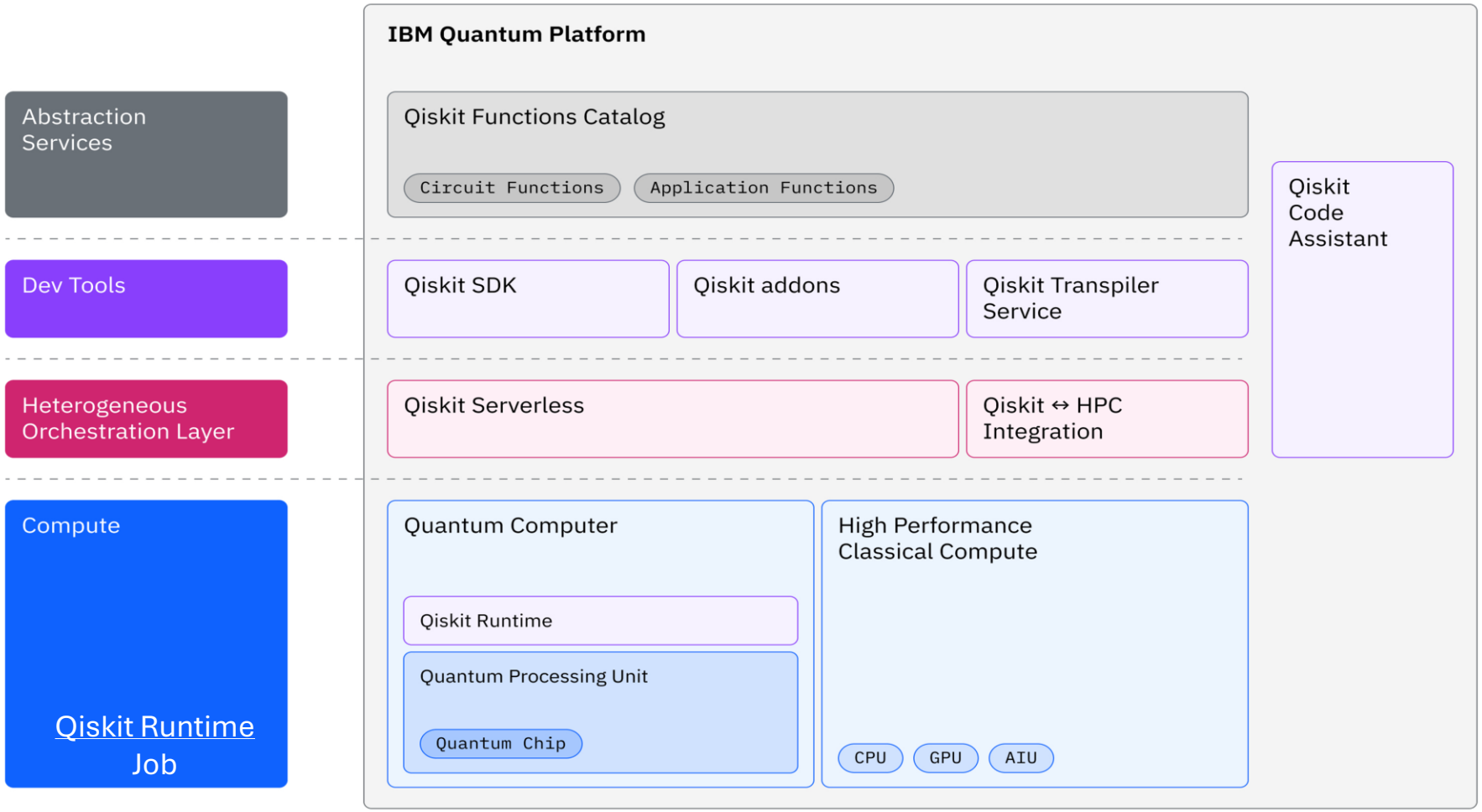
Dependent Qiskit projects

4000+

Accelerate Research with the Qiskit Software Stack

Qiskit 2.0

- Includes the foundation of a C API
- Can natively interface with almost every programming language (Fortran, C++, Rust, Python, Julia, and more)



Visual Studio code extension

In collaboration with the watsonx and Granite team

https://quantumalgorithmzoo.org/

By Stephen Jordan, with 435 contributions

Qiskit Patterns

- Provide a framework for approaching Quantum Computing at the utility scale.
- This framework consists of four steps that are very general and can be applied to most problems (though in some workstreams, certain steps may be iterated multiple times).

1. Step 1: Map classical inputs to a Quantum problem
2. Step 2: Optimize problem for Quantum execution
3. Step 3: Execute using Qiskit Runtime Primitives
4. Step 4: Analyzing and Post-Processing

Exercise Environment

[qBraid | The Quantum Computing Platform]

[Welcome To Colab - Colab]

[Qiskit]

[Quantum Platform]

[EACMichiels/UNILU]

https://github.com/EACMichiels/UNILU

QML Introduction - Simulator - Student.ipynb

dataset_graph7.csv

Exercise 1

- Calculate one element of the Kernel Matrix using a Quantum Feature Map
- Apply Qiskit Patterns
- Execute on a real Quantum Computer after Transpilation

- REMARK: several concepts will be explained later in-depth

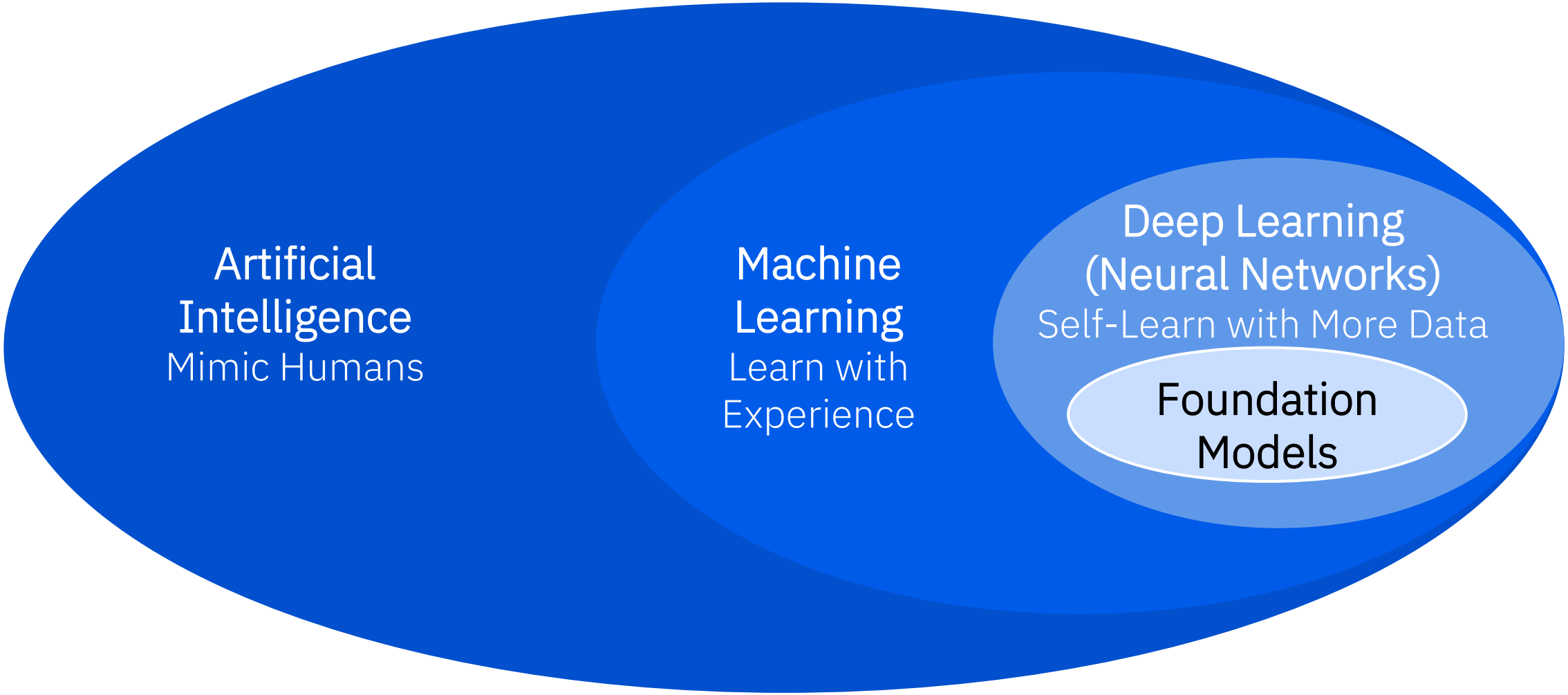
Introduction

End of Topic 1

Review of Relevant Machine Learning Models

Topic 2

Artificial Intelligence



Types of Machine Learning

Supervised learning:

- Training data is labelled.
- The algorithms learn the relationship between feature data and their corresponding labels.
- This will be generalized to unseen data.
- Common tasks are **Classification and Regression**.

Unsupervised learning:

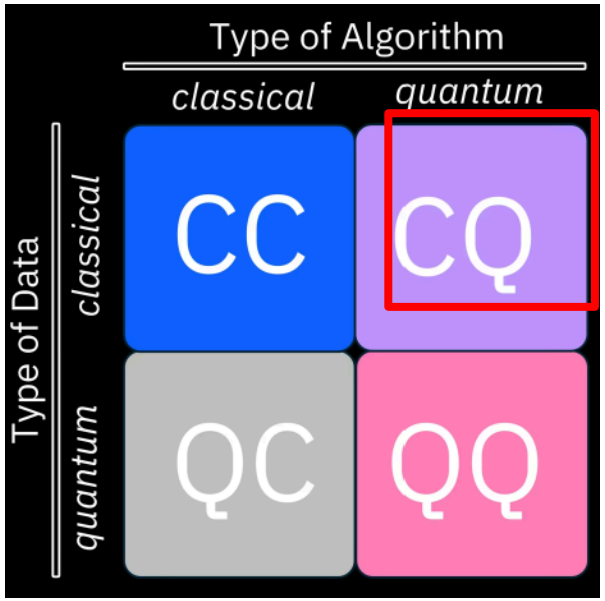
- Data are un-labelled.
- Algorithms discover patterns and structure in data.
- Common tasks are **Clustering**, Dimensionality Reduction, Generative Adversarial Networks and Variational Auto-Encoders

Reinforcement learning:

- An agent takes actions and receives feedback from its environment.
- This feedback can be **rewards and punishments**.
- The agent learns to take the correct set of actions to perform a specific task.

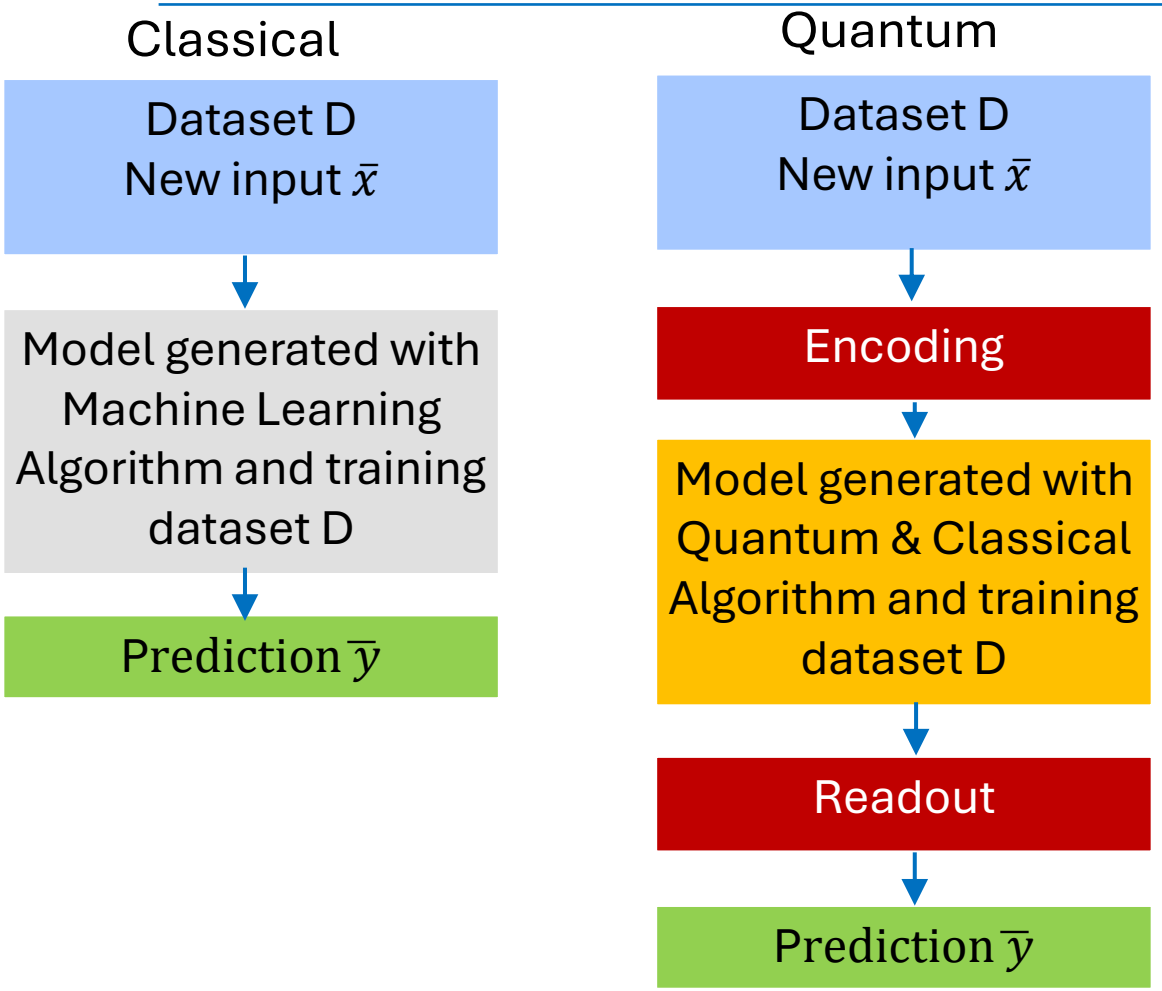
Introducing Quantum to Machine Learning

- We consider the **type of algorithm** as well as the **type of data**
- **Classical Data** are images, video, sound, text that we can store on a classical computer.
- **Quantum Data**
  - Can be taught as a set of *measurement outcomes* from a Quantum Device.
  - Can refer to *states prepared* on a Quantum Computer by another algorithm.
  - In the future it can refer to *date a stored in QRAM* or Quantum Random Access Memory
- **CC is precisely the *Classical Machine Learning* setting**
- QQ means Quantum Data and a Quantum Algorithm
- **CQ is what we usually refer to with *Quantum Machine Learning***
  - The Dataset is classical.
  - The processing device executing the machine learning algorithm is a Quantum Computer.



We will focus on CQ Algorithms

The Key Challenge in QML



Data Encoding

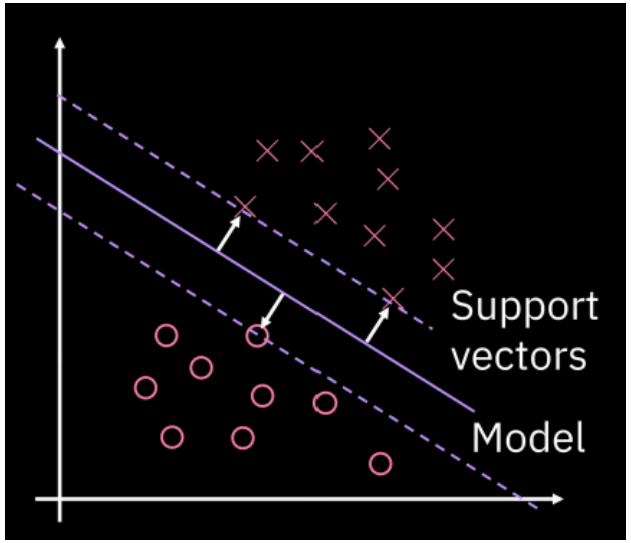
Sample encoding methods

- Basis encoding
- Amplitude encoding
- Angle encoding
- Phase encoding
- Dense Angle Encoding
- Data Encoding as a *Feature Map*
- … *(ongoing innovation)*

M. Schuld, F. Petruccione: Supervised Learning with Quantum Computers
M. Schuld, N. Killoran: arxiv 1803.07128

Support Vector Machines (SVM)

- Suppose a task of binary classification which a 2-dimensional feature space as in the picture
- To perform classification for this dataset we try to find a line or **hyperplane** that separates the two classes.
- There are infinitely many separating hyper planes.
- How do we define the optimal one new line?
- The idea is to **maximize the margin** which is the distance to the nearest points in each class
- Data points with the smallest distance to the decision boundary are called **support vectors**



Support Vector Machines (Cont.)

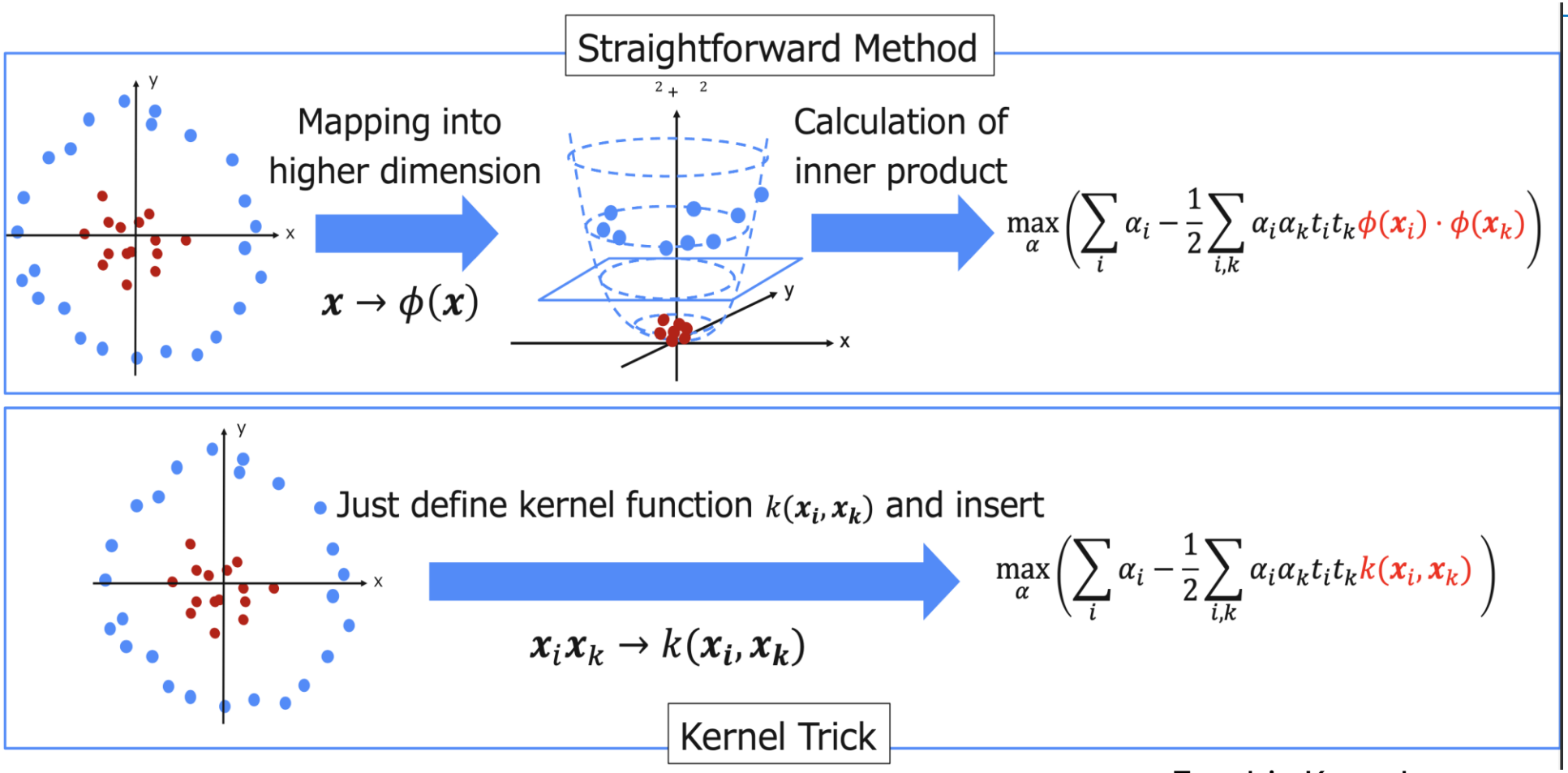
- A **linear decision boundary** can be described in several ways.
- The most straightforward is the **primal formulation f_1.**
  - θ is the set of parameters defining the hyperplane
  - It is the vector of tunable parameters that the model must learn.
  - $\vec{x}$ is the data asset.
  - b is a constant shift.
  - Φ is a mapping from the space of input data points to (an often (but not necessarily) higher dimensional space

$$f_1(\vec{x}) = \Theta^T \Phi(\vec{x}) + b$$

$$f_2(\vec{x}) = \sum_{i=1}^{n} \alpha_i y_i \Phi^T(\vec{x}_i)\Phi(\vec{x}) + b$$

- With some mathematical manipulation it is possible to create a **dual formulation f_2**
  - In this formulation the α_i parameters must be optimized.
- The difference is that we have here **an inner product between feature vectors** and **not** the feature vector and the learnable parameters.
- **The dual formulation is more useful than the primal formulation**.

The SVM Kernel Trick



For this Kernel,
Quantum Computing is applied

Mapping data into a higher dimensional space is called "Kernelling"

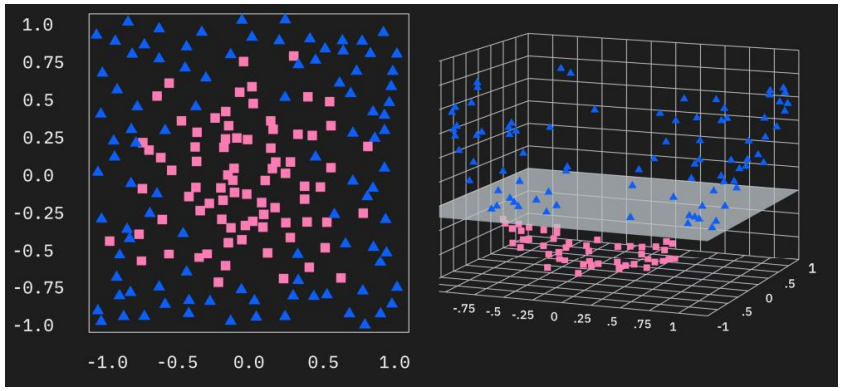
There are different mapping or transformation functions

Most of them are implemented in Data Science programming languages

Kernel Methods and how Quantum can play a role

Moving to higher dimensional spaces

- We will focus on mapping to higher dimensions and the "**Kernel Trick**"
- High dimensional Quantum States are not sufficient for improvement over classical methods
- At the *left* we see that there is **no linear decision boundary** that can separate the 2-dimensional data classes.
- When we add a third feature to the feature space the data **become linear separate**, as we see on *the right*
- In this example the new feature is a product of the previous two features x_1 and x_2
- We can run the SVM successfully on this higher dimensional feature space



$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\vec{\Phi}(\vec{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

- Φ maps, the input data space to a higher dimension.
- **There are also models that make use of mapping to lower dimensions.**
- The mapping to higher dimensions is easy to visualize and understand

Kernel Methods and how Quantum can play a role

Why is the Dual Form useful?

$$f_1(\vec{x}) = \Theta^T \Phi(\vec{x}) + b$$

$$f_2(\vec{x}) = \sum_{i=1}^{n} \alpha_i y_i \Phi^T(\vec{x}_i)\Phi(\vec{x}) + b$$

- **We can replace the original feature vector $\vec{x}$ with the feature mapped vectors**
- But if we do this in the Primal Form, we must compute the inner products between the parameters and a potentially very high dimensional feature map.
- **In the Dual Form we see that these inner products are replaced by inner products between feature mapped vectors of different inputs**
- And for some Feature Maps it is possible to write the inner product of feature mapped vectors as a simple function $k(\overrightarrow{x_i}, \overrightarrow{x_j})$ of the original lower dimensional variables $\overrightarrow{x_i}$ and $\overrightarrow{x_j}$
- For some choices of Φ we may even be able to write k as a simple function of the lower dimensional inner product $\overrightarrow{x_i}^T \overrightarrow{x_j}$
- ***This is computational is very beneficial because we can access the space in which data are linearly separable, without the cost of manipulation it in higher dimensions.***

Kernel Methods and how Quantum can play a role

Why is the Dual Form useful (Cont.)?

- It is possible to create a Kernel Matrix k
- Each pair of data vectors in the Dataset are used as **arguments of the function k**.
- This matrix is symmetric and positive semi-definite.
- Once we can compute the Kernel Matrix, we can find the optimal parameters α_i, with:
  - Quadratic programming
  - Sequential minimal optimization

$$k = \begin{pmatrix} k(\vec{x}_1, \vec{x}_1) & k(\vec{x}_1, \vec{x}_2) & \dots \\ k(\vec{x}_2, \vec{x}_1) & k(\vec{x}_2, \vec{x}_2) & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

Formally, a real symmetric matrix $A \in \mathbb{R}^{n \times n}$ is **positive semi-definite** if for **all** nonzero vectors $x \in \mathbb{R}^n$:

$$x^\top A x \geq 0$$

$$A^T = A$$

All Eigenvalues ≥ 0

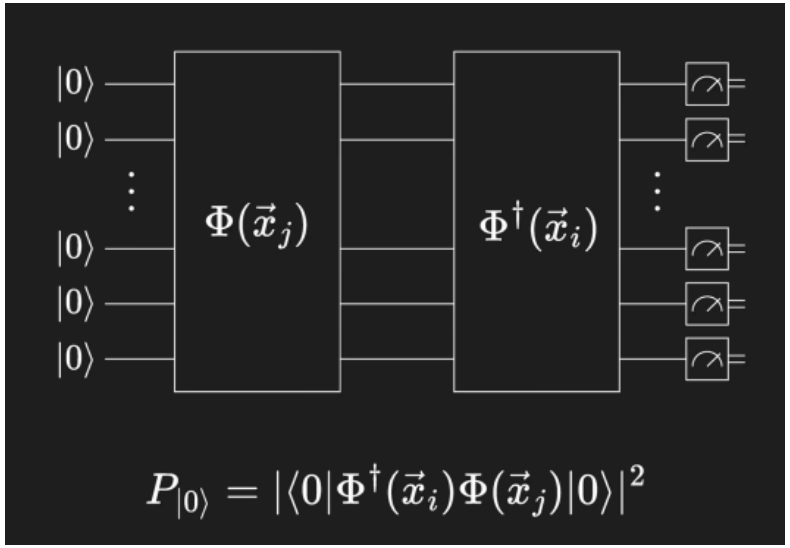
A can be written as $B^T B$

- *All this assumes that there exists an efficiently calculable kernel corresponding to a Feature Map that makes the data classes linear separable.*
- A novel approach is **Quantum Kernel Estimation**

Kernel Methods and how Quantum can play a role

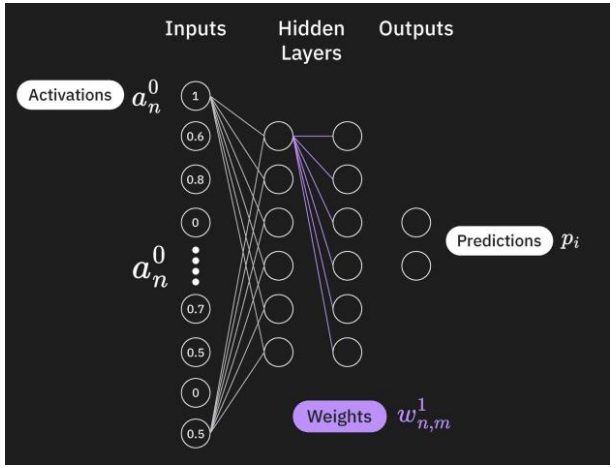
Quantum Kernels

- **We interpret the encoding of an input $\vec{x}$ into a Quantum State $|\Phi(\vec{x})\rangle$ as a Feature Map**
- The inner product of two Quantum States $\langle\psi|\phi\rangle$ is highly related to the *probability of measuring the state ϕ when the state is ψ*
- We can estimate the inner product of the two mapped data points $\vec{\Phi}(\vec{x}_i)$ and $\vec{\Phi}(\vec{x}_j)$ by *making sufficiently many measurements of the resulting circuit*
- We can run *SVM optimization classically on the Kernel Matrix to learn the tunable parameters*



Variational Quantum Classifiers and Neural Networks

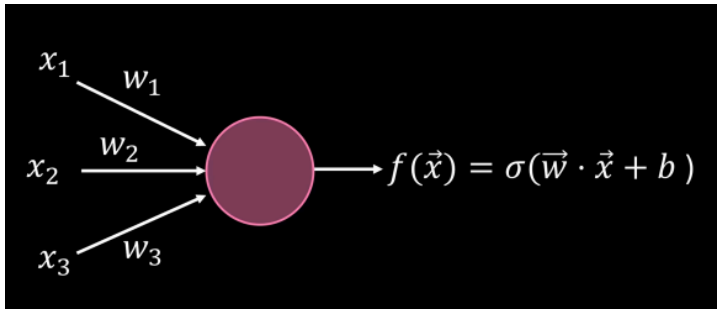
- Another near term QML algorithm is **Variational Quantum Circuit** or VQC (or PQC ☹)
- VQC also refers to **Variational Quantum Classifiers**.
- A VQC has a structure that is **similar to a classical Neural Network.**
- In those cases, a VQC is described as a **Quantum Neural Network** or **QNN**
- Understand that VQCs are **more general** and do not need to follow Neural Network structure
- *Neural Networks are computational models which are inspired by the structure and functions of neurons in the human brain*.
- These neurons are organized in layers.
- The neurons are connected through weights.
- The first layer is the input layer.
- The activations of neurons in this layer are fed directly from the data to be analyzed
- The final layer is an output layer that describes the categorization.



Example: classify an image as having 90% chance of being a dog and 10% chance of being a wolf

Variational Quantum Classifiers and Neural Networks
Perceptron

- The neurons in each layer process signals that they receive from a previous layer and transmit them to the next layer through weights.
- One neuron is a building block of a neural network and called a **Perceptron.**
- A **Perceptron** takes in an **input vector** $\vec{x}$ and computes its inner product with a trainable **weight vector** $\vec{w}$ plus some bias.
- The Perception applies **a non-linear activation function** σ on top of this computation
- Non-linear activation functions are fundamental in Neural Networks
- A function like f is calculated at every Neuron.



$$f(\vec{x}) = \sigma(\vec{w} \cdot \vec{x} + \vec{b})$$

Variational Quantum Classifiers and Neural Networks
Cost Function

- We start the **Neural Network** with a random set of weights and biases or with a reasonable starting configuration.
- Then the idea is to check how well the Neural Network classifies and then improve it.
- We use a **Cost Function** to describe how our new network deviates from the correct classification
- Cost Function C:

$$\frac{1}{N}\sum_{i=1}^{N_{\text{train}}}\sum_{j=1}^{N_{\text{outputs}}}(v_{i,j}-p_{i,j})^2$$

  - ✓ $v_{i,j}$ = actual value of an image i from input training data for output j
    - Example: for a "dog" neuron it is 1.0 and for all other neurons it is 0.0
  - ✓ $p_{i,j}$ = predicted value
- This formula not only captures whether the **right category was most activated**, but also if incorrect activations are reduced

Variational Quantum Classifiers and Neural Networks

Quantum Perceptron

- Quantum Circuits only implement **Unitary Operations** which are simply **Linear**
- We need to consider how to implement **Non-Linearity** with Quantum Circuits.
- There are different methods we can use to introduce Non-Linearity to Quantum Circuits.
  - Use **Measurements**
  - Include **Quantum Fourier Transform** based methods
  - **Dynamic Circuits** with mid-circuit measurements
  - **Tracing** Qubits **out** of the Circuit (after applying a CNOT for example)

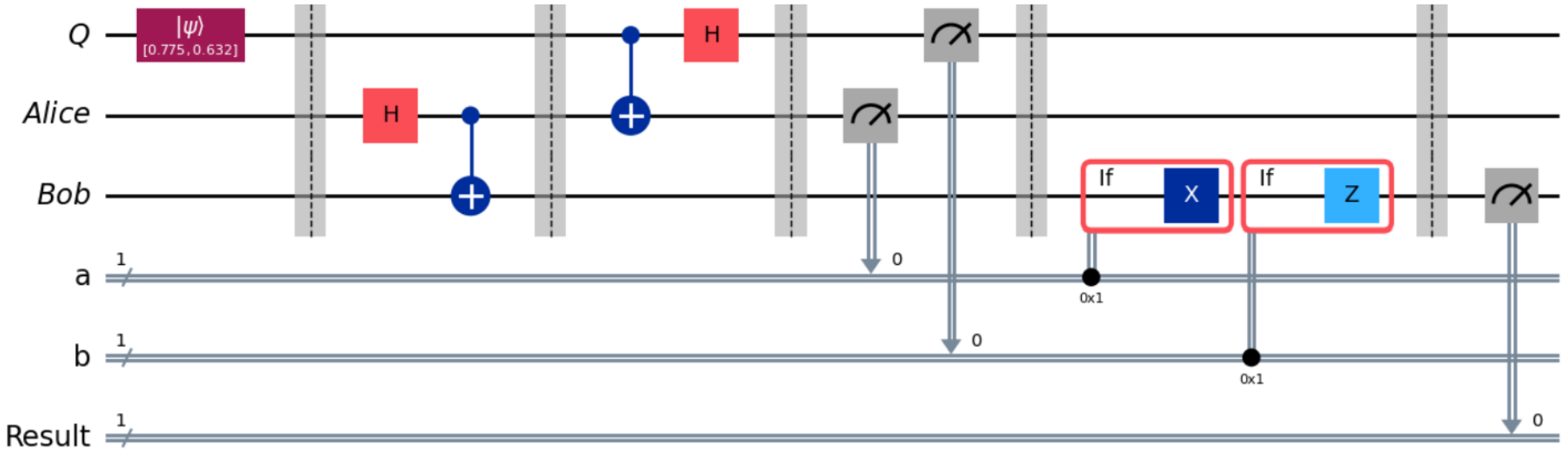
In Linear Algebra, an invertible complex square matrix U is **unitary** if its matrix inverse U^{-1} equals its conjugate transpose U^*, that is, if

$$U^*U = UU^* = I$$

where I is the identity matrix

Simple Example of a Dynamic Circuit

The “Quantum Teleportation Algorithm”



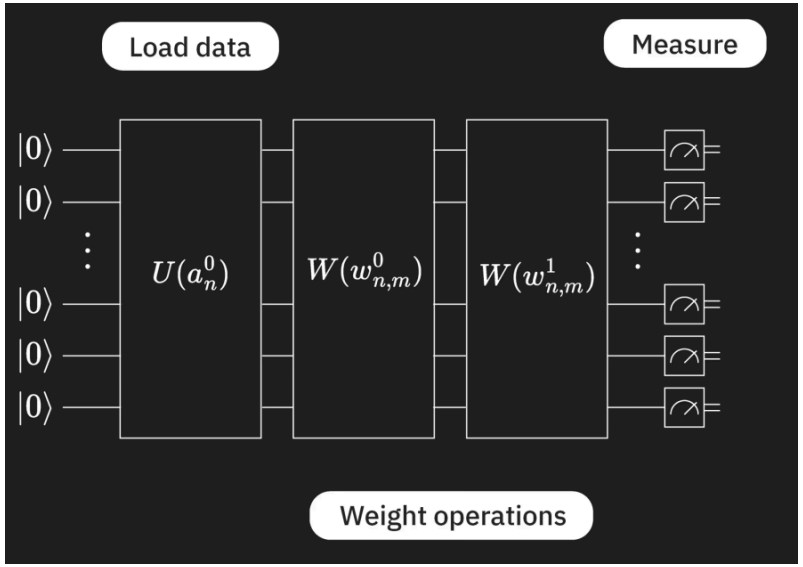
Operations performed by Bob

- I if ab = 00
- Z if ab = 01
- X if ab = 10
- ZX if ab = 11

Variational Quantum Classifiers and Neural Networks

Quantum Neural Network

- We first encode the input data with the **Unitary Layer U**
- We apply Quantum Circuits corresponding to **weights between layers W**
- We finally add a layer of **Measurement**



Some considerations

- The **Data Loading** and **Weighting** are *Linear Operations*.
- The measurements are **Non-Linear**.
- So as in classical neural networks we have **both** Linear and Non-Linear components.
- The weight circuits have variational parameters so there is still a **Classical Minimization** to be carried out

$$f_{QNN}(x) = \langle 0|U^{\dagger}(X)W^{\dagger}OWU(x)|0\rangle$$

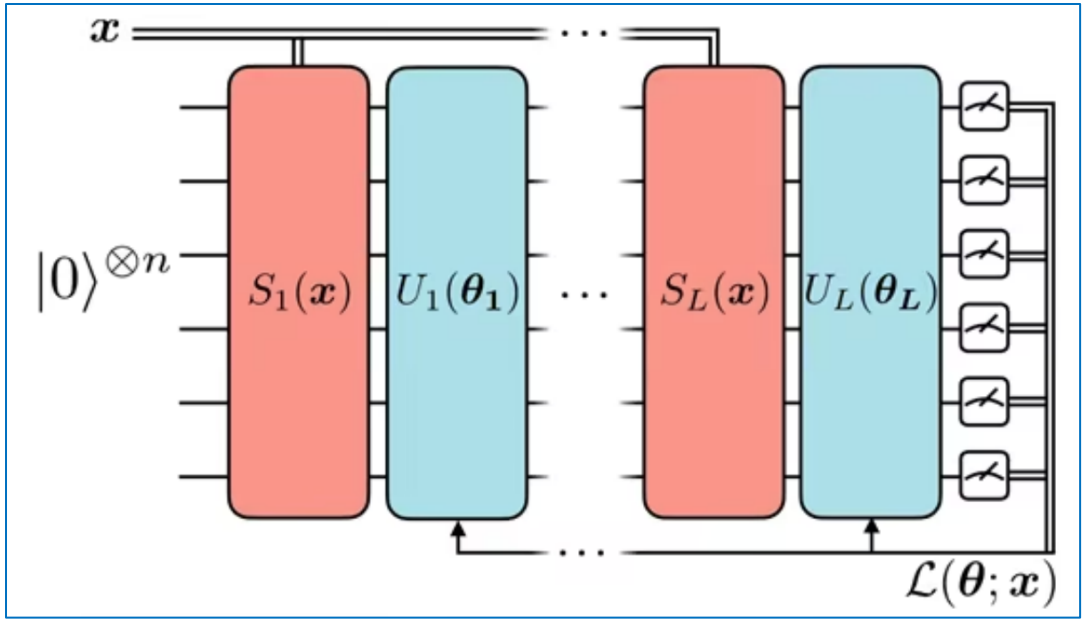
Variational Quantum Classifiers and Neural Networks

Generalizations

- With the help of **multiple data re-uploading**, a single Qubit provides sufficient computational capabilities to construct a **Universal Quantum Classifier** when assisted with a classical routine.
- **Data re-uploading is a technique that we can use to enhance the expressiveness and representational power of the model**
- It allows your Quantum Neural Network to approximate complex functions

S_i = Feature Map

U_i= Ansatz



Variational Quantum Classifiers and Neural Networks
Generalizations

- In the classical Neural Network information flows from left to right starting with the input and ending with the output.
- In the right direction we will do back-propagation to train the model.

- In this **Quantum Neural Network** construction, we see that the **Data Encoding Unitary Block repeats itself** between the **Variational Unitary Blocks** with the trainable parameters.
- We call this strategy data **Re-Uploading**

- This is studied in the paper by *Salinas, Cervera-Lierte, Gil-Fuster, and Latorre*

Data re-uploading for a universal quantum classifier

Adrián Pérez-Salinas[1,2], Alba Cervera-Lierta[1,2], Elies Gil-Fuster[3], and José I. Latorre[1,2,4,5]

[1]Barcelona Supercomputing Center
[2]Institut de Ciències del Cosmos, Universitat de Barcelona, Barcelona, Spain
[3]Dept. Física Quàntica i Astrofísica, Universitat de Barcelona, Barcelona, Spain.
[4]Nikhef Theory Group, Science Park 105, 1098 XG Amsterdam, The Netherlands.
[5]Center for Quantum Technologies, National University of Singapore, Singapore.

Published: 2020-02-06, volume 4, page 226

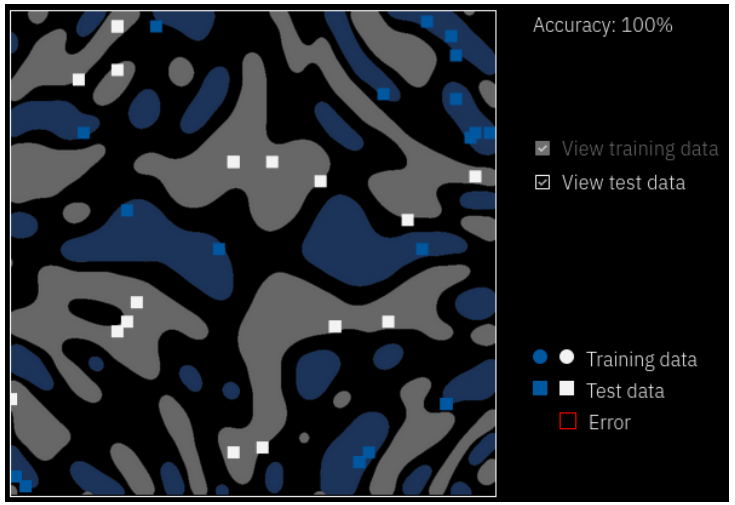
https://quantum-journal.org/papers/q-2020-02-06-226/

Near-term QML Algorithms – Key Flavours

Quantum Neural Networks

- Classification, Regression
- Variational Quantum Circuits
- Benefits: model expressiveness, resilience to Barren Plateaus

Key paper: Abbas et al, *Nature Comp. Sci.* 1, pp 403–409 (2021)

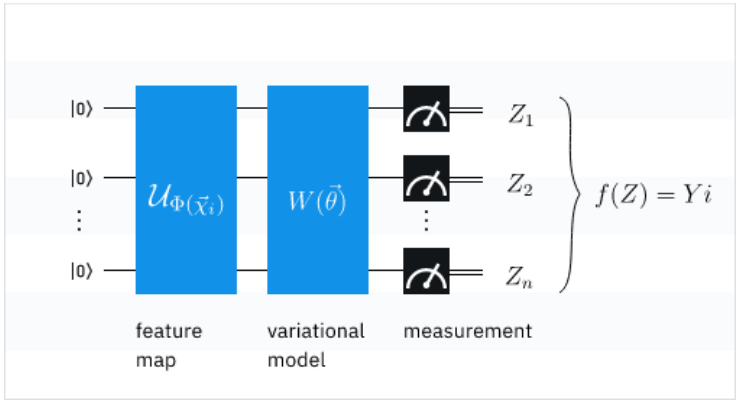


https://www.nature.com/articles/s41586-019-0980-2

Quantum SVM

- Classification, Regression
- Quantum Kernel Estimation method
- Benefit: Quantum Feature Space

Key paper: Havlíček et al, *Nature* 567, pp 209–212 (2019)

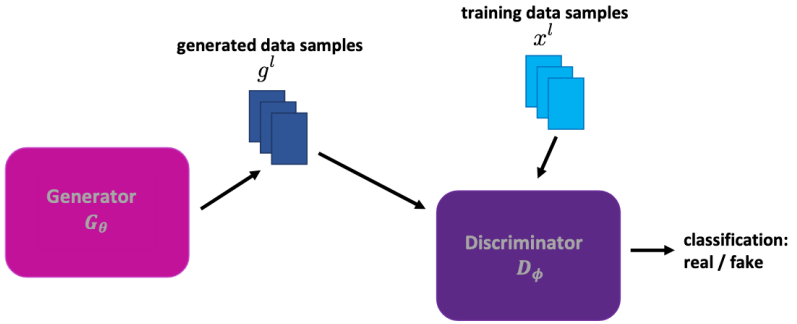


https://arxiv.org/abs/2011.00027

Quantum GANs

- Data Generation
- Quantum and Classical Neural Networks
- Benefits: efficient data sampling

Key paper: Zoufal et al, *npj Quant. Info*. 5, no: 103 (2019)



https://export.arxiv.org/abs/1904.00043

References

[1] "**Reinforcement Learning: An Introduction**" , Richard S. Sutton and Richard G. Barto, MIT Press, Second Edition, Cambridge, MA, 2018

http://incompleteideas.net/book/RLbook2020.pdf

[2] "**Pattern Recognition and Machine Learning**" , Christopher M. Bishop, Springer, 2006

https://link.springer.com/book/9780387310732

[3] "**Foundations of Machine Learning**" , Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar, MIT Press, Second Edition, 2018.

https://cs.nyu.edu/~mohri/mlbook/

Review of Relevant Machine Learning Models

End of Topic 2

Data Encoding

Topic 3

Introduction

- To use a Quantum Algorithm, Classical Data must be brought into a Quantum Circuit.
- This encoding process is a critical part of QML.
- This is referred to as **Data Encoding** or **Data Loading**.
- Transferring classical data to a Quantum Computer is called also **Feature Mapping.**
- **Qiskit has built-in Feature Maps**.
- These Feature Maps typically include rotation layers and entangling layers
- These layers extend the state to many dimensions in the Hilbert space.
- We will show a high **variety** of possible encoding strategies so you can compare and contrast them and see what is possible

Example

- The famous `efficient_su2` with a full **entangling scheme** is much more likely to capture Quantum Features of data than methods that use Product States as Feature Map.
- But this does not mean `efficient_su2` is sufficiently well matched to your data
- There is also a **balancing act** with **circuit depth** since many Feature Maps which fully **entangled** Qubits yield very deep circuits - **too deep** to get usable results on today's Quantum Computers

- PS. `efficient_su2` was originally developed as Ansatz, but it is also used as Feature Map.

Notation

- $X = \{\vec{x}^{(j)} \mid j \in [M]\}$
  - $\vec{x}^{(j)} = (\vec{x}_1^{(j)}, \ldots, \vec{x}_N^{(j)}) \in \mathbb{R}^N$ is a single vector from the full set X
  - A dataset is a set of $\boldsymbol{M}$ **data vectors**
  - Each vector is $\boldsymbol{N}$ **−dimensional**
- Feature Mapping of data vector $\vec{x}$
  - Machine Learning in general: $\Phi(\vec{x})$
  - Unitary Circuit implementation: $U(\vec{x})$

Normalization

- In Classical Machine Learning, training data features are often normalized to improve model performance
- A common approach is "*min-max normalization*"
- All feature values fall in the unit interval: $x'^{(i)}_k \in [0, 1]$ for all $i \in [M], k \in [N]$

$$x_k^{'(i)} = \frac{x_k^{(i)} - \min\{x_k^{(j)} \mid \vec{x}^{(j)} \in [\mathrm{X}]\}}{\max\{x_k^{(j)} \mid \vec{x}^{(j)} \in [\mathrm{X}]\} - \min\{x_k^{(j)} \mid \vec{x}^{(j)} \in [\mathrm{X}]\}}$$

- **This is a normalization imposed by <u>human experts</u>**
- Normalization is also a fundamental concept in **Quantum Computing and Quantum Mechanics** but slightly different from min-max normalization:
  - The length of a State Vector ψ is 1 : $\big|\big||\psi|\big|\big| = \sqrt{\langle\psi|\psi\rangle} = 1$

$a\ |0\rangle + b\ |1\rangle$
$|a|^2 + |b|^2 = 1$

- The Quantum State is normalized by dividing by this so-called "2-norm": $|\psi\rangle \rightarrow ||\ \psi\ ||^{-1}\ |\psi\rangle$
- **This is a normalization imposed by a fundamental property <u>of Quantum States</u>**

Methods of Encoding

- Basis Encoding
- Amplitude Encoding
- Angle Encoding
- Phase Encoding
- Dense Angle Encoding

Built-In Feature Maps (using Angle and Phase Encoding):

- Efficient SU2
- Z Feature Map
- ZZ Feature Map
- Pauli Feature Map

These BIFMs differ from each other in **Depth, #Qubits, Degree of Entanglement**

We will discuss each of them in the subsequent slides

- We use as example a classical dataset X_{ex} consisting of

$$\mathrm{X_{ex}} = \{(4,8,5),(9,8,6),(2,9,2),(5,7,0),(3,7,5)\}$$

  - $M = 5$ data vectors
  - Each with $N = 3$ features

- For example: $\vec{x}_1^{(4)} = 5$

Basis Encoding

Number of Qubits
= ***Number of bits*** *required to represent the* ***maximum feature value***
= *3*

- Encodes a classical **P-Bit** string into a computational basis state of a **P-Qubit** system
- A single Data Vector of the set is described as a **Superposition** of all Computational Basis States that describe the Features of that Vector
- Example: vector $\vec{x}^{(4)} = (5, 7, 0)$
  - Example for one single feature in one single vector:
    - $\vec{x}_1^{(1)} = 5$ can be represented as (101)
    - A 3-Qubit system brings us to the state $|101\rangle$
  - Equivalently
    - Encode 7 as $|111\rangle$
    - Encode 0 as $|000\rangle$
  - As the Quantum State must have norm = 1, we need to divide by $\sqrt{3}$
  - Conclusion: $|\vec{x}^{(4)}\rangle = \frac{1}{\sqrt{3}}(\,|101\rangle + |111\rangle + |000\rangle\,)$
- General formula:

$$|x^{(j)}\rangle = \frac{1}{\sqrt{N}} \sum_{k=1}^{N} |x_k^{(j)}\rangle$$

Advantage:
- Simple to understand

Disadvantages:
- The state vectors can become quite sparse.
- Schemes to implement it are not Qubit efficient.

Amplitude Encoding

- Encodes Data into amplitudes of **Quantum States**
- Example: vector $\vec{x}^{(1)} = (4, 8, 5)$
  - Normalization:
    - The value of α becomes

$$\sum_{i=1}^{N} \left| x_i^{(1)} \right|^2 = 4^2 + 8^2 + 5^2 = 105 = |\alpha|^2 \rightarrow \alpha = \sqrt{105}$$

  - Conclusion:
    - We get a 2-Qubit Quantum State

$$|\psi(\vec{x}^{(1)})\rangle = \frac{1}{\sqrt{105}}(4|00\rangle + 8|01\rangle + 5|10\rangle + 0|11\rangle)$$

- General Formula:

$$|\psi_x^{(j)}\rangle = \frac{1}{\alpha} \sum_{i=1}^{N} x_i^{(j)} |i\rangle \qquad \sum_{i=1}^{N} \left| x_i^{(j)} \right|^2 = |\alpha|^2$$

- A classical N-dimensional data vector is represented as amplitudes of a n-Qubit Quantum State with $|i\rangle$ being the i^{th} computational basis state

- *Each feature in a data vector is stored as the amplitude of a DIFFERENT Quantum State*
- ***N features***
- *Required #Qubits:* $n \geq log_2(N)$

Advantage:
- Less Qubits needed

Disadvantages:
- Algorithms must operate on the amplitudes
- Preparation and Measurement of Quantum States tend NOT to be efficient

Amplitude Encoding "Pros" and "Cons" with O-notation

Benefit

- **Only $log_2(N)$ Qubits** are required for an N-dimensional or N-feature data vector

Disadvantages

- Generally, an inefficient procedure that requires arbitrary **State Preparation**, which is **exponential in the number of CNOT gates**
- Stated differently:
  - The State Preparation has a polynomial runtime complexity of $O(N)$ in the number of dimensions, where $N = 2^n$, and n is the number of Qubits
  - Amplitude encoding "***provides an exponential saving in space at the cost of an exponential increase in time***"[1]
- However, more "modest" runtime increases to $O(logN) = O(logn)$are achievable in certain cases[(2)]

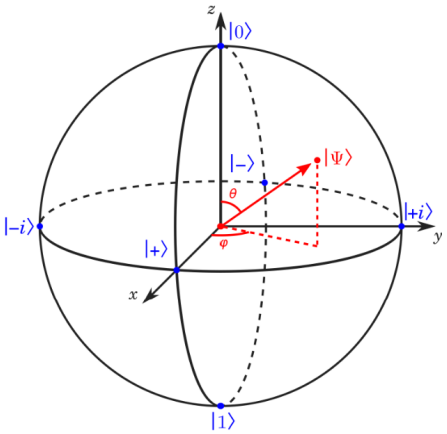
For an end-to-end quantum speedup, the data loading runtime complexity needs to be considered !

(1) https://arxiv.org/abs/2003.01695
(2) https://arxiv.org/abs/quant-ph/0208112

Angle Encoding (1/2)

- Angle Encoding is of interest in many QML models using Pauli Feature Maps such as **QSVMs** and **VQCs**, among others
- It refers to a rotation θ away from the z-axis accomplished by an R_X gate or an R_Y gate but it could be any rotation or combination of rotations
- $\boldsymbol{R_Y}$ **is most common in the literature**
- Let us apply it to a single Qubit:

$$|\vec{x}_k^{(j)}\rangle = R_Y(\theta = \vec{x}_k^{(j)})|0\rangle = \cos\left(\frac{\vec{x}_k^{(j)}}{2}\right)|0\rangle + \sin\left(\frac{\vec{x}_k^{(j)}}{2}\right)|1\rangle.$$



- Here, we encoded the k^{th} feature from the j^{th} data vector in a dataset
- Alternatively, we can also apply an R_X gate, with complex relative phases
- The differences of **Angle Encoding** from the previous two methods:
  - Each feature is mapped to a corresponding Qubit, leaving the **Qubits in a product state**
  - One numerical value is encoded at a time (and not a whole set of features from a data point)
  - For N data features, we need N Qubits
  - The **resulting circuit has a constant depth** (1 before transpilation)

Angle Encoding (2/2)

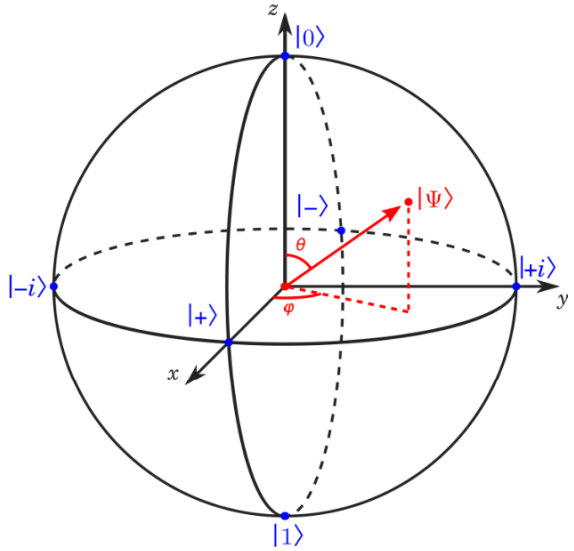
- The constant depth **makes Angle Encoding amenable to current Quantum Hardware**
- **It is recommended to rescale the data so that** $x_k^{(j)} \in (0, 2\pi]$

- Let us encode N features into the rotation angles of n Qubits
  - Data vector: $\vec{x}^{\,j} = (x_1, \ldots, x_N)$
  - Products state:

$$|\vec{x}^{(j)}\rangle = \bigotimes_{k=1}^{N} \cos(\vec{x}_k^{(j)})|0\rangle + \sin(\vec{x}_k^{(j)})|1\rangle$$

  - Equivalent expression:

$$|\vec{x}^{(j)}\rangle = \bigotimes_{k=1}^{N} R_Y(2\vec{x}_k^{(j)})|0\rangle.$$



Phase Encoding

- Similar to Angle Encoding
- The phase of a Qubit is a real-valued angle ϕ about the $z-axis$ from the $+x-axis$
- **Data are mapped with a Phase Rotation:**
  - $P(\phi) = e^{i\phi/2} R_z(\phi)$ with $\phi \in (0, 2\pi]$
- **It is recommended again to rescale the data so that $x_k^{(j)} \in (0, 2\pi]$**
- What is the practical approach?
  - The Qubit is initialized in $|0\rangle$
  - A Hadamard Gate is applied: $H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
  - A relative phase proportional to the data value is applied:

$$|\vec{x}_k^{(j)}\rangle = P(\phi = \vec{x}_k^{(j)})|+\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{i\vec{x}_k^{(j)}}|1\rangle\right)$$

- The circuit has a depth 2, making it efficient for encoding
- For N features, there are n Qubits and this is the Product State:

$$|\vec{x}^{(j)}\rangle = \bigotimes_{k=1}^{N} P_k(\phi = \vec{x}_k^{(j)})|+\rangle^{\otimes N} = \frac{1}{\sqrt{2^N}} \bigotimes_{k=1}^{N} \left(|0\rangle + e^{i\vec{x}_k^{(j)}}|1\rangle\right)$$

Phase encoding is used in many Quantum Feature Maps, such as Z, ZZ and Pauli Feature Maps

Dense Angle Encoding or DAE

- Combination of Angle Encoding and Phase Encoding
- Allows **2 Features to be encoded in a 1 Qubit:**
  - One angle with a $\boldsymbol{Y-axis}$ **rotation** angle and the other with a $\boldsymbol{Z-axis}$ **rotation** angle
- Two features are encoded as follows:

$$\vec{x}_k^{(j)}, \vec{x}_\ell^{(j)} \rightarrow \theta, \phi$$

$$|\vec{x}_k^{(j)}, \vec{x}_\ell^{(j)}\rangle = R_Z(\phi = \vec{x}_\ell^{(j)}) R_Y(\theta = \vec{x}_k^{(j)})|0\rangle = \cos\left(\frac{\vec{x}_k^{(j)}}{2}\right)|0\rangle + e^{i\vec{x}_\ell^{(j)}} \sin\left(\frac{\vec{x}_k^{(j)}}{2}\right)|1\rangle$$

- DAE can be extended to more Features:

$$\vec{x} = (x_1, \ldots, x_N)$$

For N data features, we need $\frac{N}{2}$ Qubits

$$|\vec{x}\rangle = \bigotimes_{k=1}^{N/2} \cos(x_{2k-1})|0\rangle + e^{ix_{2k}} \sin(x_{2k-1})|1\rangle$$

- **General Qubit Encoding**: generalize DAE to arbitrary functions instead of $\boldsymbol{\cos(x)}$ **and** $\boldsymbol{\sin(x)}$

Encoding with Built-In Feature Maps

- Angle Encoding, Phase Encoding, and Dense Encoding **prepared product states** with 1 or 2 Features encoded on each Qubit
- This is different from Basis Encoding and Amplitude Encoding, in that those methods make use of **entangled states**: There is not a 1:1 correspondence between Data Feature and Qubit

- **Generally, methods that encode in product states yield shallower circuits**
- Methods that use entanglement and associate a feature with a state (rather than a qubit) result in **deeper circuits**, but can **store more Features per Qubit** on average

- But encoding **need not be entirely** in product states or entirely in entangled states
- Indeed, many encoding schemes built into Qiskit allow encoding both before and after an **entanglement layer**, as opposed to just at the beginning. **This is known as "data reuploading“**

- Maximizing data loading for a given number of Qubits is not the only consideration.
- In many cases, **Circuit Depth** may be an even more important consideration than **Qubit Count**.

Efficient SU2

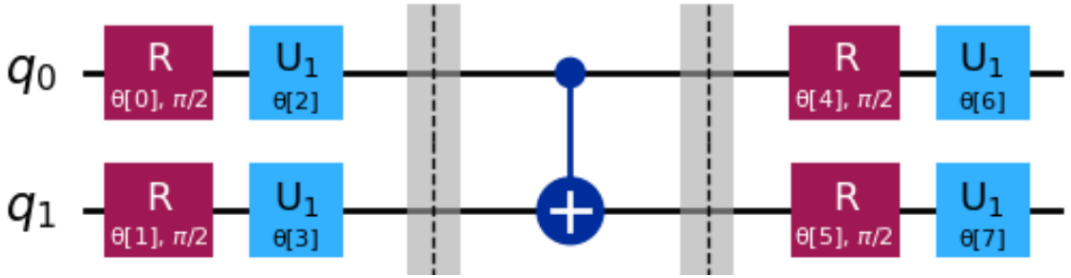
- Impressively, this Circuit can for example encode **8 Features on 2 Qubits** !
- Example 1:

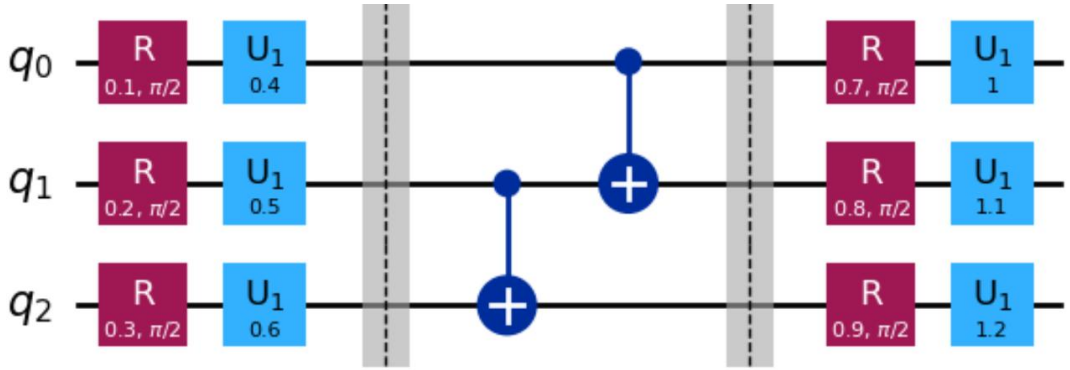
```
circuit =
efficient_su2(num_qubits=2,reps=1,
insert_barriers=true)
```

- By increasing the **number of Qubits** and increasing the number **of Repetitions** (of Entangling and Rotation Layers), one can encode much more data
- Example 2:

  Data Vector with **12 Features on 3 Qubits**

- Trade-Off: Circuits with **more Qubits or more Repetitions** may **store more parameters**, but do so with **greater Circuit Depth**





Z Feature Map (ZFM)

- The ZFM can be interpreted as a **natural extension of Phase Encoding**
- ZFM consists of alternating layers of Hadamard Gates and Phase Gates

$$\mathcal{U}_{\mathrm{ZFM}}(\vec{x})|0\rangle^{\otimes N} = |\phi(\vec{x})\rangle$$

- $|0\rangle^{\otimes N}$ is the N-Qubit Ground State
- The Circuit Unitary for the full ZFM Circuit with a single repetition:

$$\mathcal{U}_{\mathrm{ZFM}} = \left(P(\vec{x}_1) \otimes \ldots P(\vec{x}_k) \otimes \ldots P(\vec{x}_N) H^{\otimes N}\right) = \left(\bigotimes_{k=1}^{N} P(\vec{x}_k)\right) H^{\otimes N}$$

- The Circuit Unitary for the full ZFM Circuit with a r repetitions:

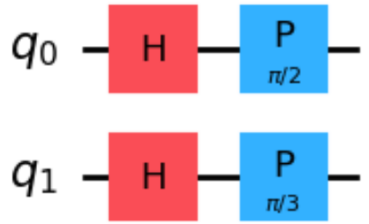
$$\mathcal{U}_{\mathrm{ZFM}}^{(r)}(\vec{x}) = \prod_{s=1}^{r} \left[\left(\bigotimes_{k=1}^{N} P(\vec{x}_k)\right) H^{\otimes N}\right]$$

- **In all repetitions, the Data Features x_k are mapped to the Phase Gates in the same way**

Simple Example:

2 Qubits

Data Vector $\vec{x} = (\frac{\pi}{2}, \frac{\pi}{3})$



ZZ Feature Map (ZZFM)

- The ZZFM extends the ZFM with the inclusion of 2-Qubit **Entangling Gates**
- These gates are instances of the **ZZ-Rotation Gate $R_{ZZ}(\theta)$**
- the ZZ-Rotation Gate $R_{ZZ}(\theta)$ is maximally entangling for $\theta = \frac{\pi}{2}$
- The ZZFM is conjectured to be **generally expensive to compute on a Classical Computer** unlike the ZFM
- The ZZFM consists of a **Hadamard Gate, the Phase Gate, the ZZ-Gate**
- For a single repetition:

$$\mathcal{U}_{\mathrm{ZZFM}}(\vec{x}) = U_{ZZ}(\vec{x})\left(P(\vec{x}_1)\otimes \ldots P(\vec{x}_k)\otimes \ldots P(\vec{x}_N)H^{\otimes N}\right) = U_{ZZ}(\vec{x})\left(\bigotimes_{k=1}^{N} P(\vec{x}_k)\right)H^{\otimes N}$$

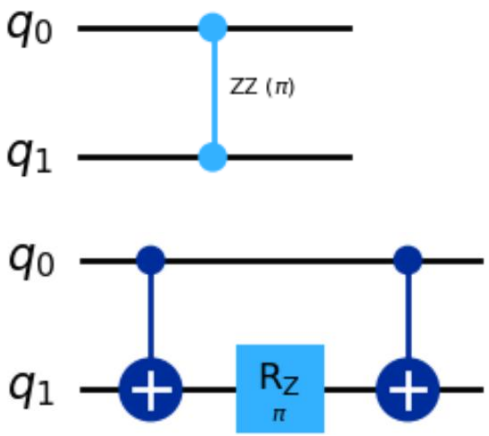
- Suppose the R_{ZZ} gate is to applied on Qubits p and q
- The argument of R_{ZZ} is then

$$\theta_{q,p} \rightarrow \phi(\vec{x}_q, \vec{x}_p) = 2(\pi - \vec{x}_q)(\pi - \vec{x}_p).$$

- For r repetitions:
- Entanglement schemes:
  `linear, circular, full`

$$\mathcal{U}^{(r)}_{\mathrm{ZZFM}}(\vec{x}) = \prod_{s=1}^{r}\left[U_{ZZ}(\vec{x})\left(\bigotimes_{k=1}^{N} P(\vec{x}_k)\right)H^{\otimes N}\right].$$

Simple Example:
2 Qubits with $\theta = \pi$



Pauli Feature Map (PFM)

- PFM is the **generalization of the ZFM and ZZFM** using **arbitrary Pauli Gates**
- For r repetitions for encoding the vector $\vec{x}$:

$$\mathcal{U}_{\mathrm{PFM}}(\vec{x}) = \prod_{s=1}^{r} U(\vec{x}) H^{\otimes n}.$$

- $U(\vec{x})$ is generalized as follows:

$$U(\vec{x}) = \exp\left(i \sum_{S \in \mathcal{I}} \phi_S(\vec{x}) \prod_{i \in S} \sigma_i \right)$$

  - ✓ σ_i is a Pauli operator: $\boldsymbol{\sigma_i \in X, Y, Z, I}$
  - ✓ I is the set of all Qubit connectivities as determined by the Feature Map, **including** the set of Qubits acted on by single-qubit gates.
  - ✓ Example: if a $P - Gate$ acts on Qubit 0 and an $R_{ZZ} - Gate$ acts upon Qubit 2 and 3, then $\{\{0\},\{2,3\}\} \epsilon\, I$
  - ✓ S runs through all elements of that set I
  - ✓ Here, $\phi_s(\vec{x})$ can involve 1-Qubit Gates or 2-Qubit Gates

$$\phi_S(\vec{x}) = \begin{cases} x_i & \text{if } S = \{i\} \text{ (single-qubit)} \\ \prod_{j \in S} (\pi - x_j) & \text{if } |S| \geq 2 \text{ (multi-qubit)} \end{cases}$$

Conclusions

- Generally, **Pauli** and **ZZ Feature Maps** will result *in greater circuit depth* and higher numbers of *2-Qubit gates* than **efficient_su2** and **Z Feature Maps**.
- Because the Feature Maps built into Qiskit are widely applicable, ***we will often not need to design our own, especially in the learning phase.***
- However, **experts in QML** will likely return to the subject of designing their own feature mapping, as they tackle two complicated challenges:
  1. Modern hardware: **the presence of noise** and the large overhead of error-correcting code mean that present-day applications will need to consider things like **hardware efficiency** and minimizing two-qubit gate depth.
  2. Mappings that fit the problem at hand:
     - It is one thing to say that the `zz_feature_map`, for example, is *difficult to simulate classically*, and therefore interesting.
     - It is quite another thing for the `zz_feature_map` **to be ideally suited to your Machine Learning task or data set**.

The performance of different parameterized Quantum Circuits on different types of data is an active area of investigation.

Hardware-Efficient Feature Mapping

- Takes into account constraints of **real Quantum Computers** to reduce noise and errors

Main strategy:

- **Minimize depth of Circuit** so **that noise** and **decoherence** have less time to corrupt the computation

- **Considerations:**
  - ✓ The depth of the logical circuit may be much lower that the depth of the **transpiled circuit**
  - ✓ Gates in the Logical Circuit must be replaced by **Native Hardware Gates**
    - Example: `ibm_torino: CZ, ID, RZ, SX, X`
  - ✓ **SWAP Gates** are added to move Qubit States between Qubits to **enable coupling**
  - ✓ The higher the number of **repetitions** of the Feature Map, **the deeper** the circuit
  - ✓ `Full` Entanglement is deeper than `Linear` Entanglement

Some Review before the Hands-On Labs

$$U(\theta,\phi,\lambda) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\phi+\lambda)}\cos\left(\frac{\theta}{2}\right) \end{pmatrix} \quad = \quad U3(\theta,\phi,\lambda)$$

$$U(\theta,\phi,\lambda) \equiv R_z(\phi)\,R_y(\theta)\,R_z(\lambda)$$

Special cases

Gate	Parameters
Pauli-X	$U(\pi,0,\pi)$
Pauli-Y	$U(\pi,\frac{\pi}{2},\frac{\pi}{2})$
Pauli-Z	$U(0,0,\pi)$
Hadamard	$U(\frac{\pi}{2},0,\pi)$

$$U2(\phi,\lambda) = U3\left(\frac{\pi}{2},\phi,\lambda\right) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\phi+\lambda)} \end{pmatrix}$$

$$U1(\lambda) = U3(0,0,\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$$

$$U1(\lambda) = R_z(\lambda)$$

$$P(\lambda) = R_z(\lambda)$$

Exercise 2

- Practice Data Encoding Techniques

Data Encoding

End of Topic 3

Topics of 24-Nov-2025

1. Introduction
2. Recap of Machine Learning
3. Data Encoding

Quick Recap of Lecture 1

Access to Real Quantum Devices

4. **Quantum Kernel methods and Support Vector Machines**

Lab 1: Kernel Matrix - Simulator

Lab 2: Single Entry in Kernel Matrix – Real Device

5. **Variational Quantum Classifiers and Neural Networks**

Lab 3: QNN for Image Classification - Simulator

Quick Recap of Lecture 1

Intermezzo

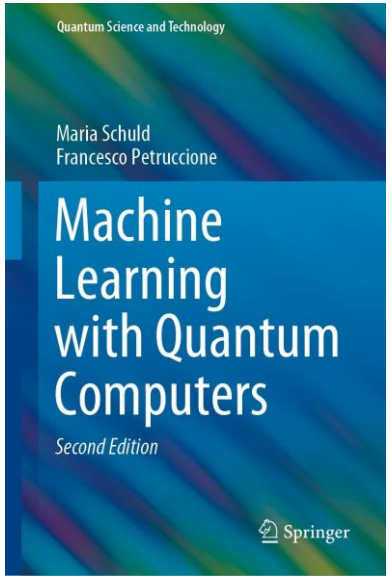
Positioning Quantum Machine Learning (QML)

Type of Data
Data Generating System

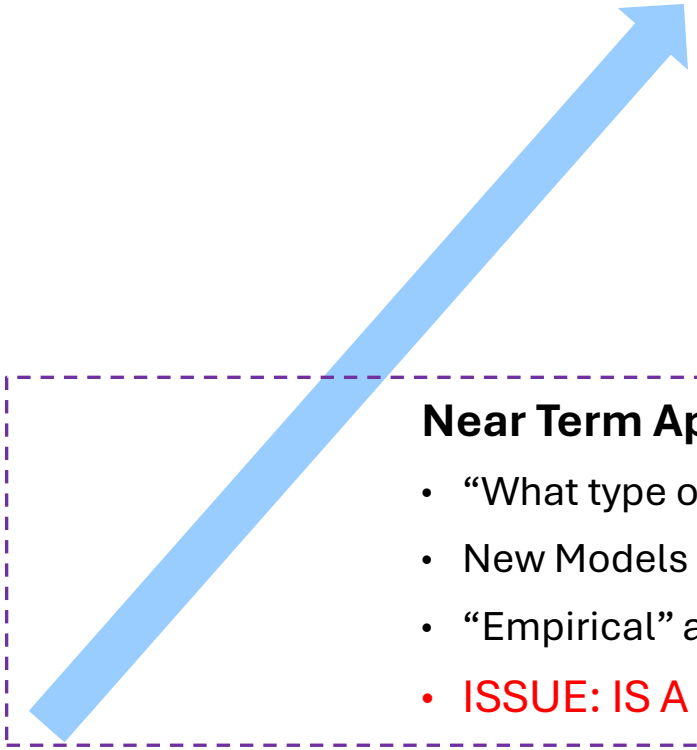
Type of Algorithm
Processing Device

Quantum \ Classical	Classical	Quantum
Classical	CC	**CQ**
Quantum	QC	QQ

Maria Schuld, Francesco Petruccione
Machine Learning with Quantum Computers", Page 7



Near-Term and Fault-Tolerant Methods



Emerging Approach - Third Wave

- Based on deeper understanding of ML Potential of Quantum phenomena
- Combine the CML knowledge with Quantum Information Theory
- Train Quantum Models we cannot simulate any more …
- **QUANTUM INSPIRED METHODS**

> 2021

Near Term Approach - Second Wave

- "What type of ML Model fits the physical characteristics of a small-scale Quantum Device?"
- New Models and Algorithms derived
- "Empirical" and "Heuristic" mindset
- ISSUE: IS A CLASSICAL ML MINDSET SUFFICIENT… ?

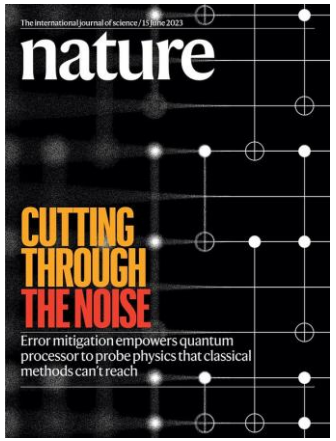
> 2017

Long Term Benefit – First Wave

- Assumes Fault-Tolerant Quantum Computers
- Rather "Academic" and "Mathematical" mindset
- ISSUE: WE ONLY HAVE NISQ TODAY …

> 2013

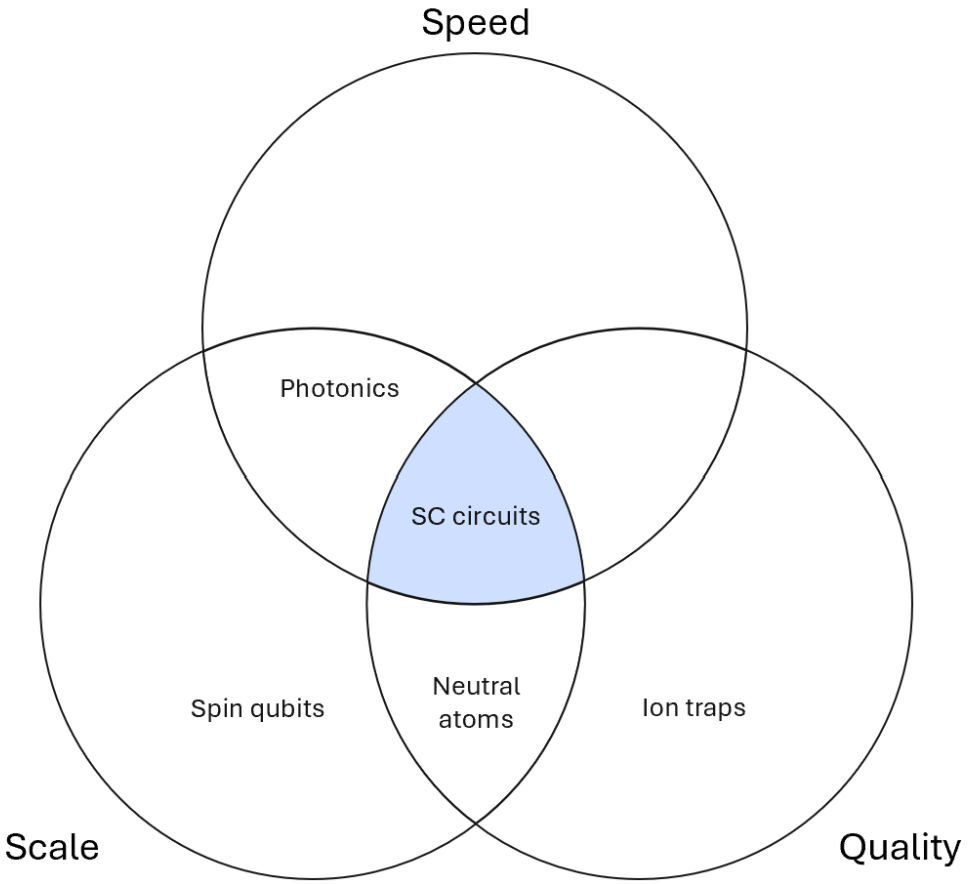
Quantum Error Mitigation, Error Suppression, Error Correction



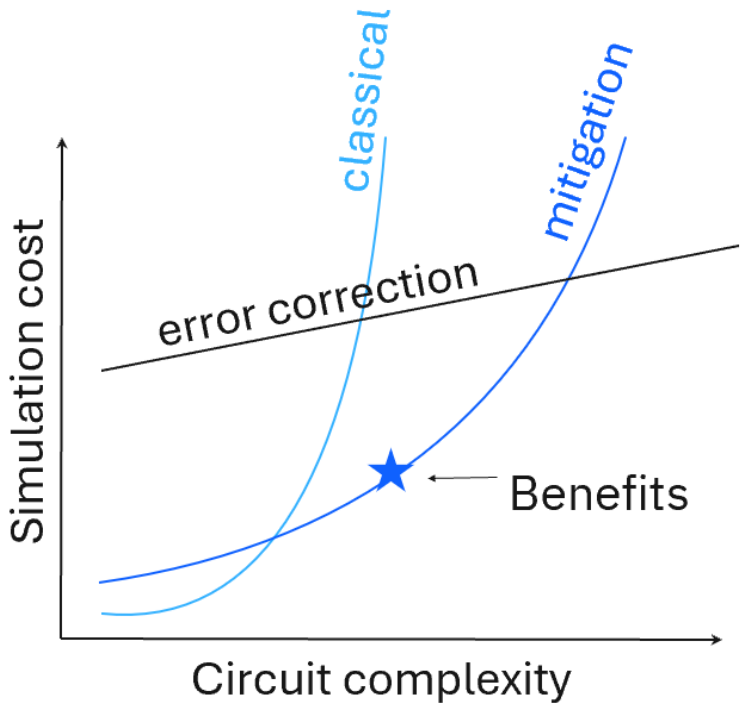
Dynamic Decoupling
Gate Twirling
Twirled Readout Extinction
Probabilistic Error Cancelation
Zero-Noise Extrapolation

Gross Code (qLDPC)

Quantum Computing Hardware Implementations

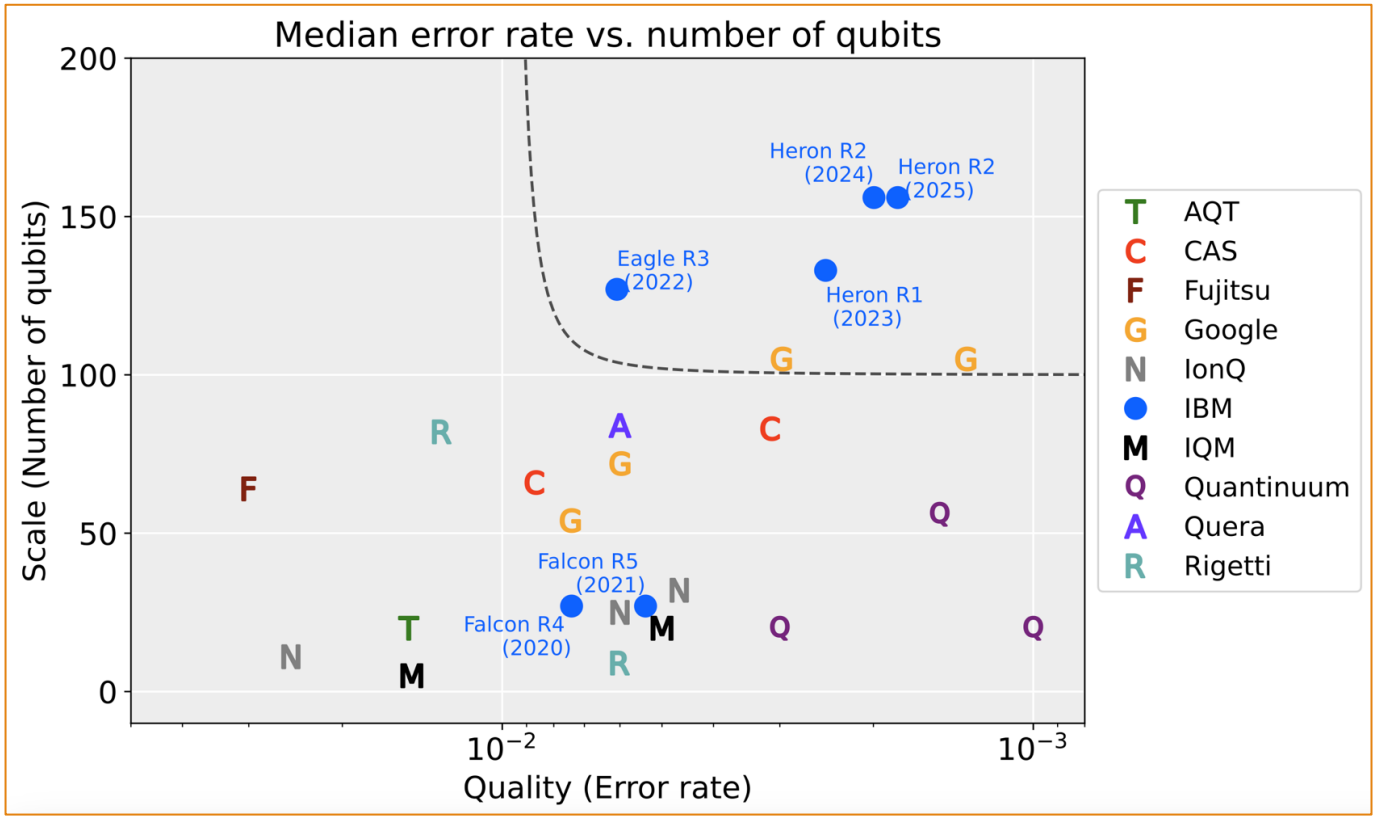


Topological Qubits ???



The Limits of Quantum Simulators

Quantum Simulator
Quantum Fake Device
Quantum Real Device



Utility scale defined as proven to be able to run a circuit larger than can be simulated classically by brute-force. System size and error rates sourced from publicly available system data sheets and published research papers.

Superconducting vs Ions

Speed

400x-2,000x

faster

@ 500 shots & 56 Qubits

Cost

1,200x-70,000x

cheaper

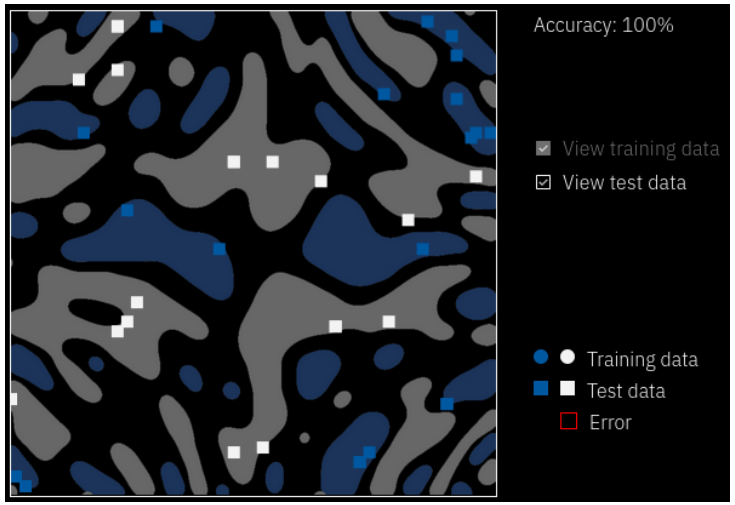
@ 500 shots & 56 Qubits

Near-term QML Algorithms – Key Flavours

Quantum SVM

- Classification, Regression
- Quantum Kernel Estimation method
- Benefit: Quantum Feature Space

Key paper: Havlíček et al, *Nature* 567, pp 209–212 (2019)

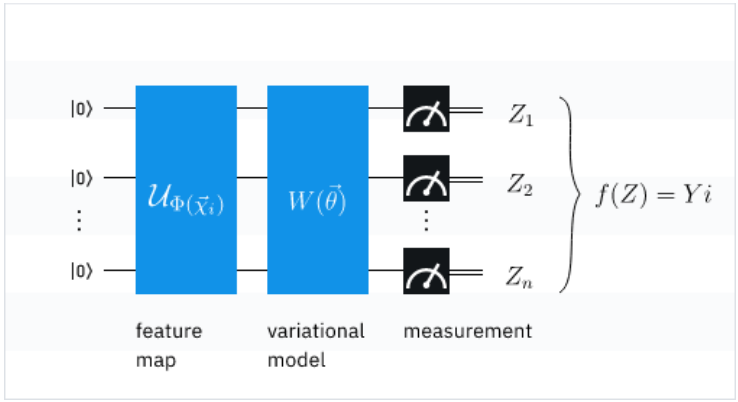


https://www.nature.com/articles/s41586-019-0980-2

Quantum Neural Networks

- Classification, Regression
- Variational Quantum Circuits
- Benefits: model expressiveness, resilience to Barren Plateaus

Key paper: Abbas et al, *Nature Comp. Sci.* 1, pp 403–409 (2021)

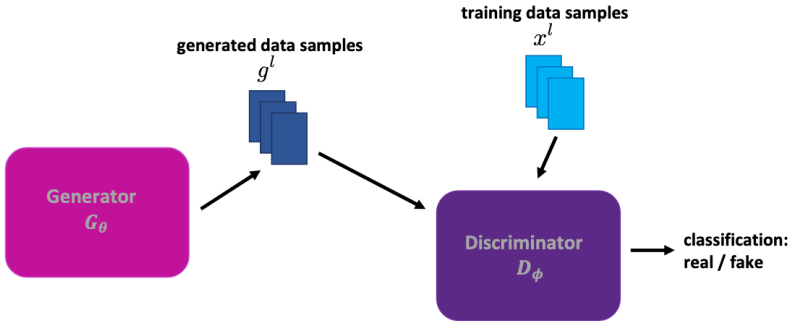


https://arxiv.org/abs/2011.00027

Quantum GANs

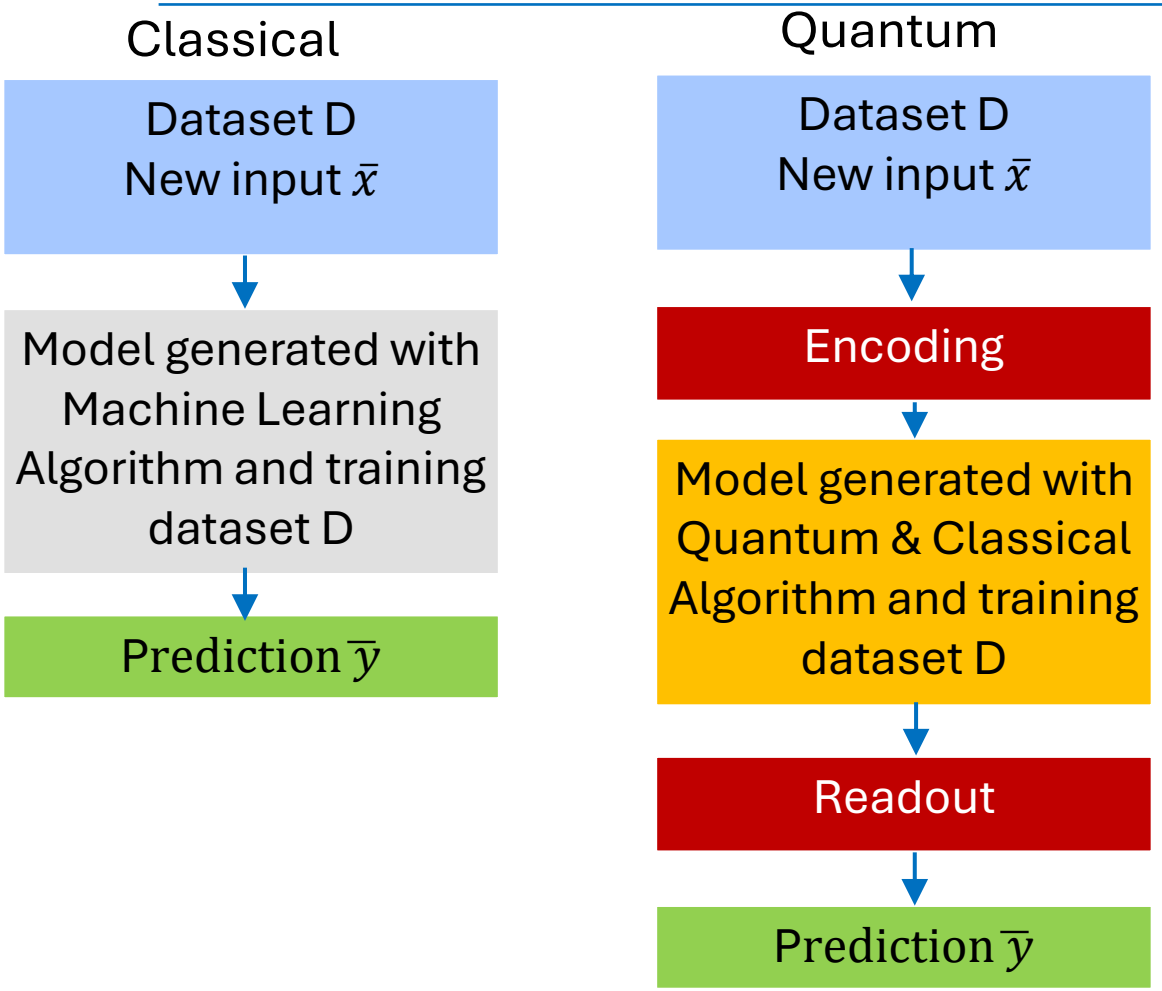
- Data Generation
- Quantum and Classical Neural Networks
- Benefits: efficient data sampling

Key paper: Zoufal et al, *npj Quant. Info*. 5, no: 103 (2019)



https://export.arxiv.org/abs/1904.00043

The Key Challenge in QML



Data Encoding

Sample encoding methods

- Basis encoding
- Amplitude encoding
- Angle encoding
- Phase encoding
- Dense Angle Encoding

- *ZFeatureMap*
- *ZZFeatureMap*
- *PauliFeatureMap*
- *Efficient SU2*

M. Schuld, F. Petruccione: Supervised Learning with Quantum Computers
M. Schuld, N. Killoran: arxiv 1803.07128

Encoding Data - Summary

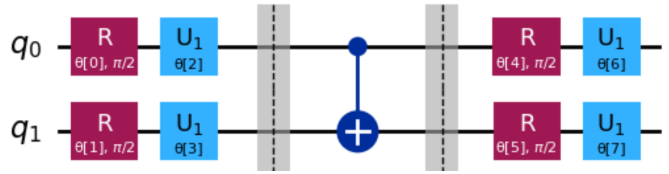
M = Number of features
n = number of bits per feature
$N = \lceil log_2 n \rceil$

Example: $[6, 8, 0]$: 6 km. walk/day, 8 hrs. sleep/day ➔ no cancer

- **Basis Encoding**
  Encode each *n*-bit feature into *n* qubits
  $x = (b_{n-1}, \dots, b_1, b_0) \rightarrow |x\rangle = |b_{n-1}, \dots, b, b_0\rangle$
  $\frac{\sqrt{3}}{3}(|0110\rangle + |1000\rangle + |0000\rangle)$ ➔ 4 Qubits
- **Amplitude Encoding**
  Encode into quantum state amplitudes
  $$x = \begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \end{bmatrix} \rightarrow |\psi_x\rangle = \sum_{j=0}^{N-1} x_j |j\rangle$$
  $(\frac{3}{5}|00\rangle + \frac{4}{5}|01\rangle + 0\,|10\rangle + 0\,|10\rangle)$ ➔ 2 Qubits
- **Angle Encoding (and Phase Encoding)**
  Encode values into qubit rotation angles
  $$|x\rangle = \bigotimes^{N} \cos(x_i)\,|0\rangle + \sin(x_i)\,|1\rangle$$
  $(\cos\frac{6\pi}{7}\,|0\rangle + \sin\frac{6\pi}{7}\,|1\rangle) \otimes (\cos\frac{8\pi}{7}\,|0\rangle + \sin\frac{8\pi}{7}\,|1\rangle) \otimes$
  $(\cos 0\,|0\rangle + \sin 0\,|1\rangle)$ ➔ 3 Qubits

- **Arbitrary Encoding (Feature Map)**
  Encode N features in constant-depth circuit with n qubits
  $$x = \begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \end{bmatrix} \rightarrow |\psi_x\rangle = \mathcal{U}_{\Phi(x)}|0\rangle$$

  - **EfficientSU2**



  - **ZFeatureMap**

$$\mathcal{U}_{\mathrm{ZFM}}^{(r)}(\vec{x}) = \prod_{s=1}^{r}\left[\left(\bigotimes_{k=1}^{N} P(\vec{x}_k)\right) H^{\otimes N}\right]$$

  - **ZZFeatureMap**

$$\mathcal{U}_{\mathrm{ZZFM}}^{(r)}(\vec{x}) = \prod_{s=1}^{r}\left[U_{ZZ}(\vec{x})\left(\bigotimes_{k=1}^{N} P(\vec{x}_k)\right) H^{\otimes N}\right]$$

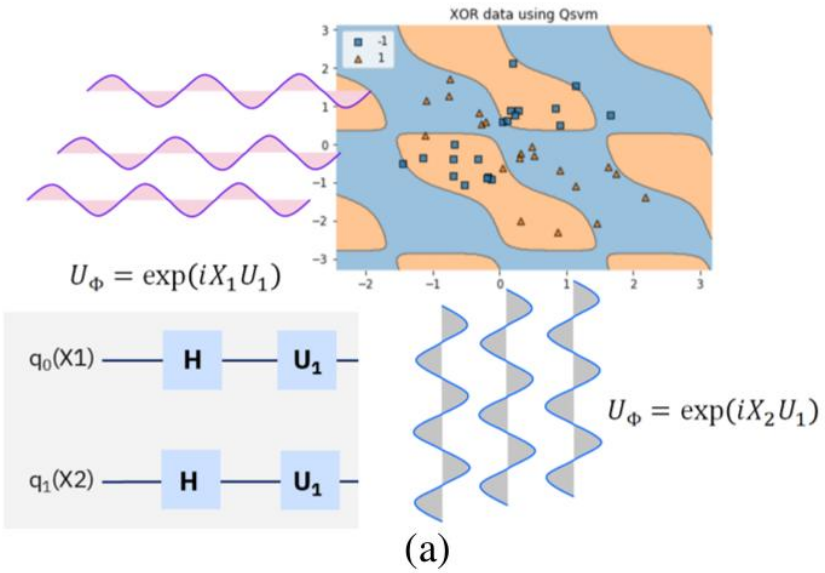
  - **PauliFeatureMap**

$$U(\vec{x}) = \exp\left(i\sum_{S\in\mathcal{I}} \phi_S(\vec{x}) \prod_{i\in S} \sigma_i\right)$$

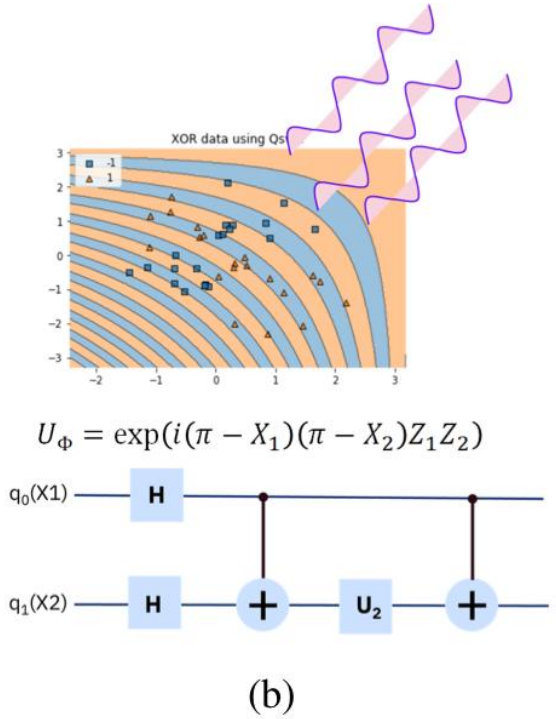
$$\phi_S(\vec{x}) = \begin{cases} x_i & \text{if } S = \{i\} \text{ (single-qubit)} \\ \prod_{j\in S}(\pi - x_j) & \text{if } |S| \geq 2 \text{ (multi-qubit)} \end{cases}$$

The impact of Entanglement in the Quantum Feature Map *(Outcome)*

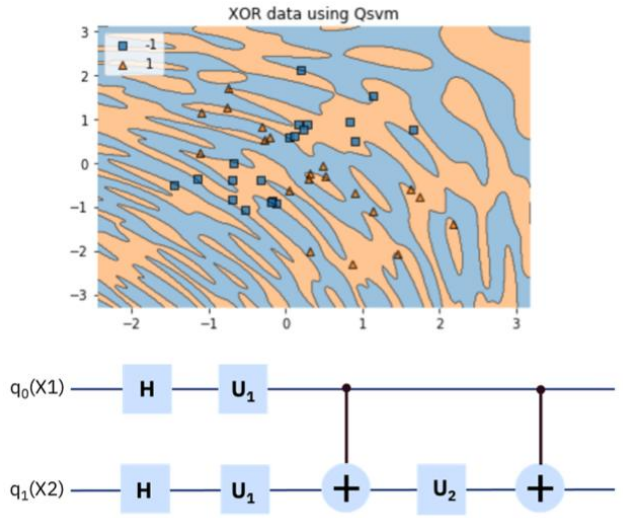
Simple rotations



Entangled unitary rotations



Simple and Entangled unitary rotations



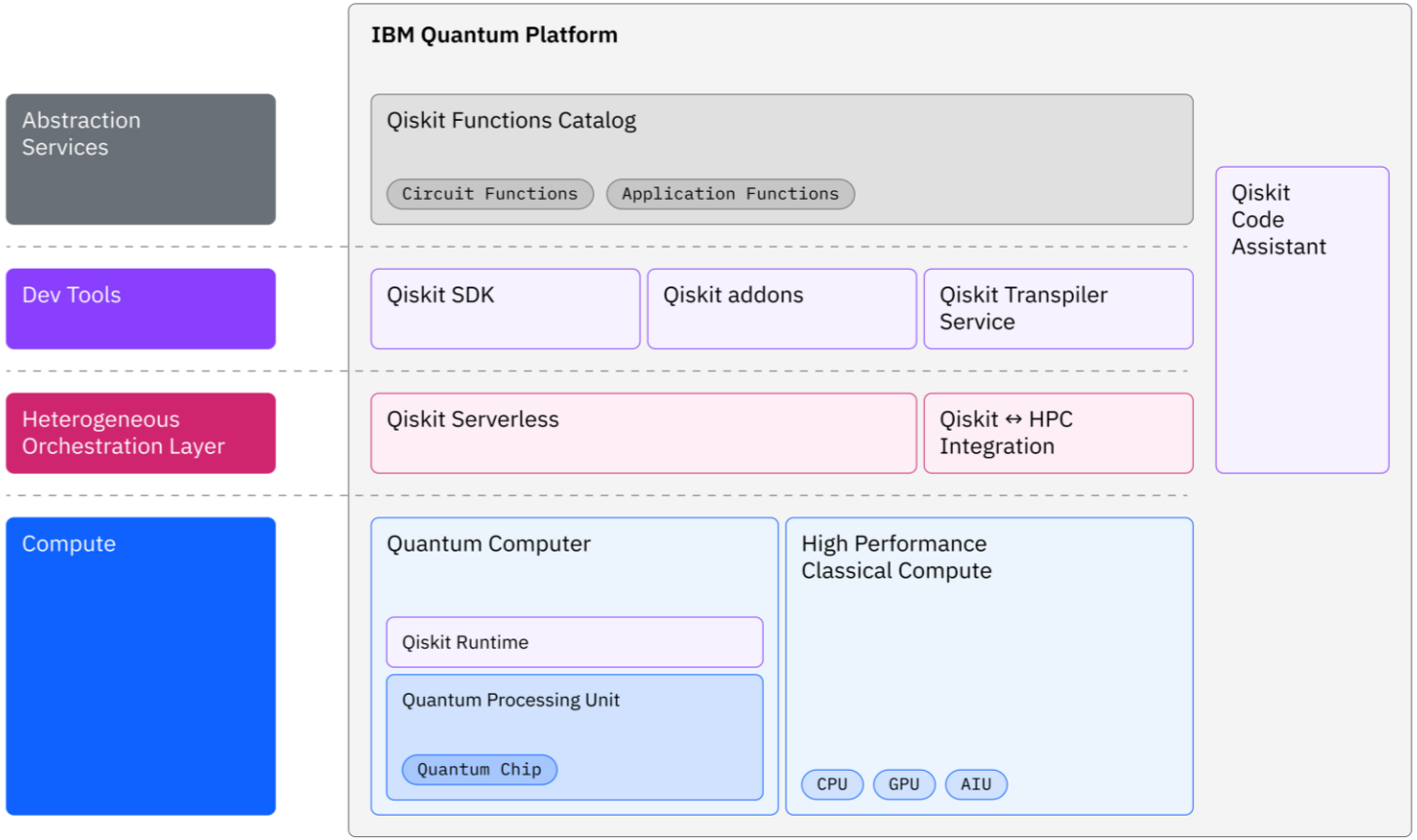
Park et al, arXiv: 2012.07725v1

https://arxiv.org/abs/2012.07725

Qiskit Stack

The **Lingua Franca** of Quantum Computing: write once and execute quantum circuits on **10+** different hardware providers

IBM Quantum
AQT
IQM
Azure Quantum
Alice & Bob
IonQ
Amazon Braket
Superstaq
Quantinuum
Rigetti

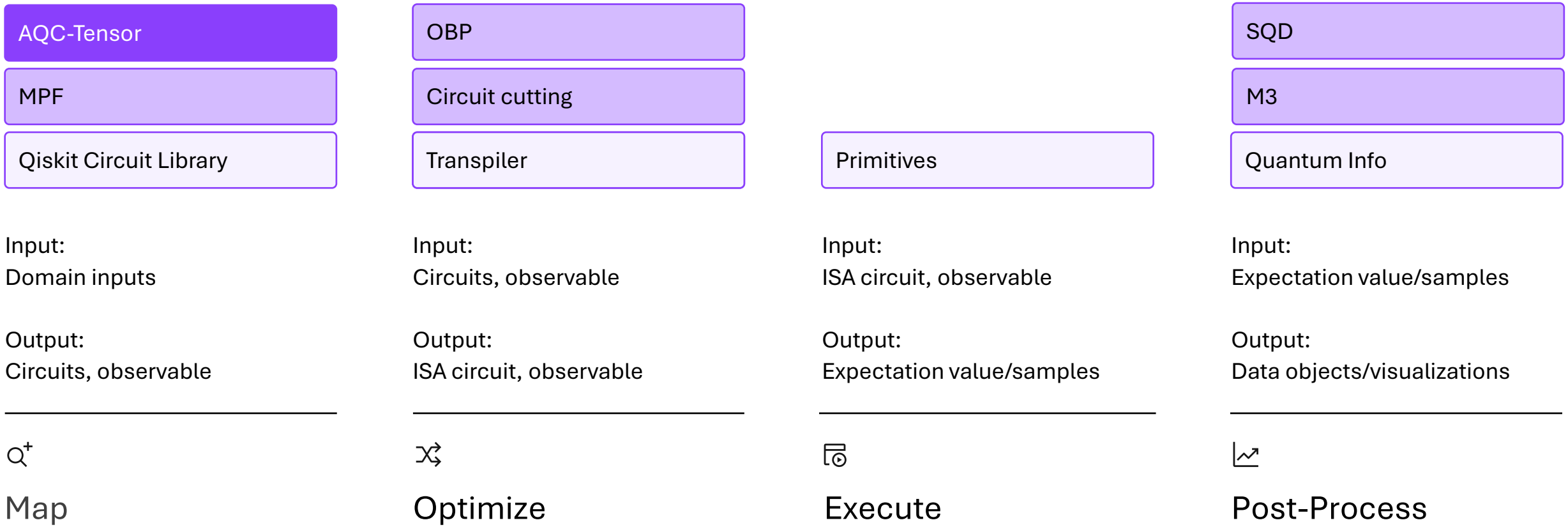


Qiskit 2.0 is the first release to introduce a C API beyond the usual Python interface, marking the beginning of multi-language support.

Qiskit Addons built on the Qiskit SDK

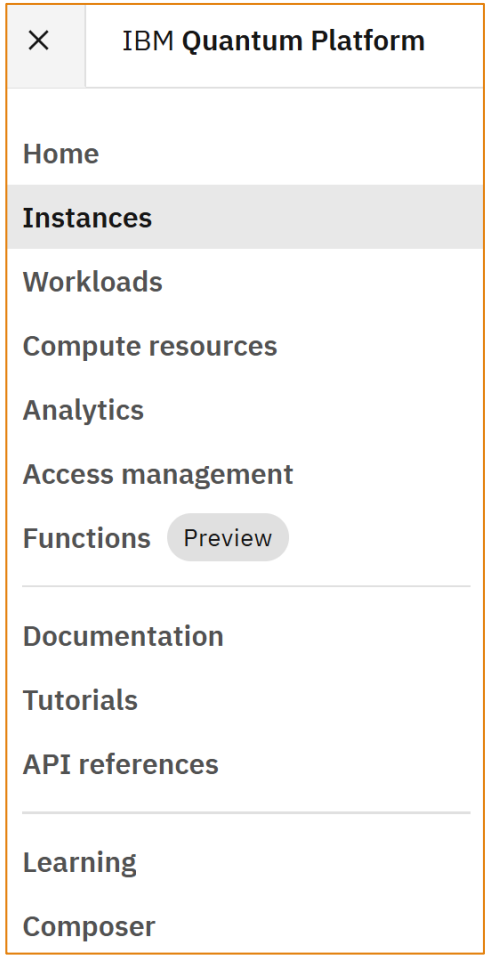
A collection of research capabilities developed as modular tools that can plug into a workflow to design new algorithms at the utility scale

- Multi-Product Formulas (MPF)
- Approximate Quantum Compilation (AQC-Tensor)
- Operator backpropagation (OBP)
- Sample-based Quantum Diagonalization (SQD)
- Matrix Free Measurement Mitigation (M3)



Access to Real Quantum Computers

- Access to IBM Quantum via IBM Cloud
- Request a Feature Code - 4251d476576d457e42c4a0ab2c47a1fc
  - ✓ `uni.lu` Academic Account needed
  - ✓ Web Site: https://www.ibm.com/academic/
- Quantum Platform: https://quantum.cloud.ibm.com/instances
- 10 QPU minutes per month



Quick Recap of Lecture 1

End of Intermezzo

Quantum Kernels

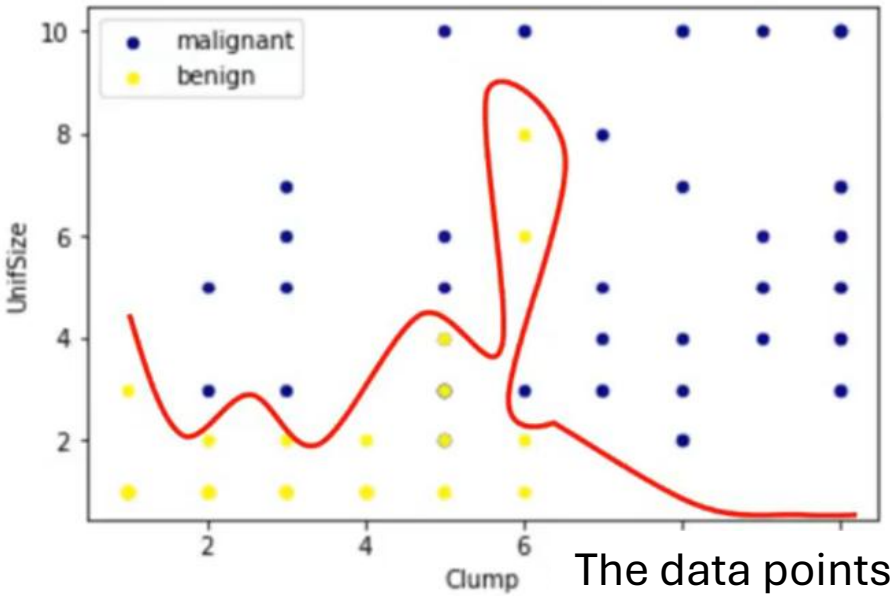
Topic 4

What is a Support Vector Machine (SVM)?

An SVM is a supervised algorithms that classifies cases by finding a separator

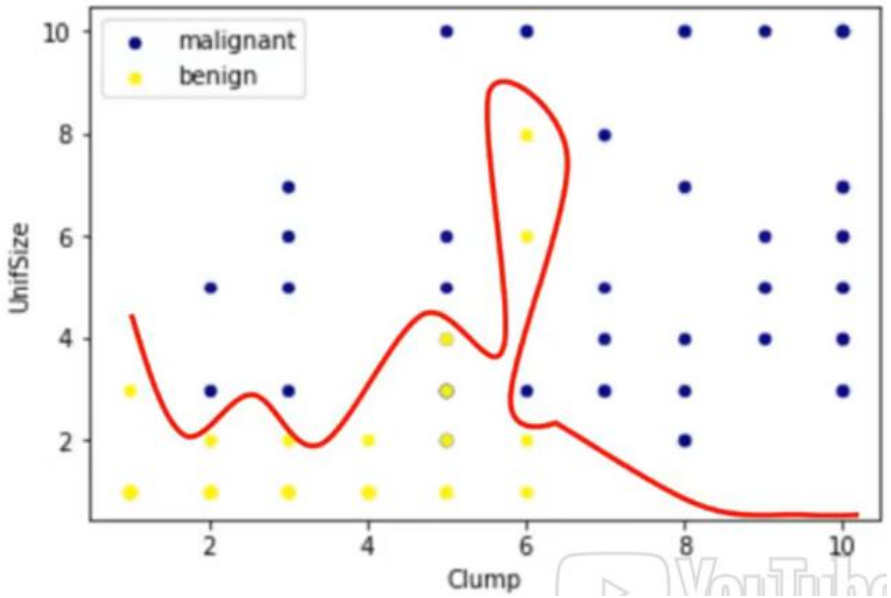
1. Map data to a higher dimensional feature space - if necessary
2. Find a separator

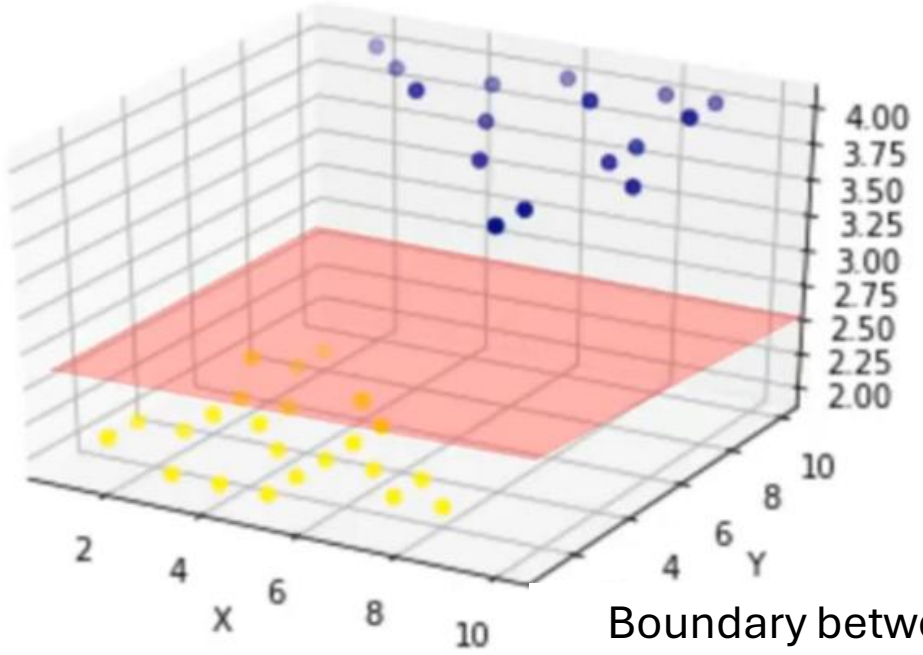
Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
5	1	1	1	2	1	3	1	1	benign
5	4	4	5	7	10	3	2	1	benign
3	1	1	1	2	2	3	1	1	malignant
6	8	8	1	3	4	3	7	1	benign
4	1	1	3	2	1	3	1	1	benign
8	10	10	8	7	10		7	1	malignant
1	1	1	1	2	10	3	1	1	benign
2	1	2	H	2	1	3	1	1	benign
2	1	1	1	2	1	1	1	5	benign
4	2	1	1	2	1	2	1	1	benign



The data points are not linearly separable

SVM Approach

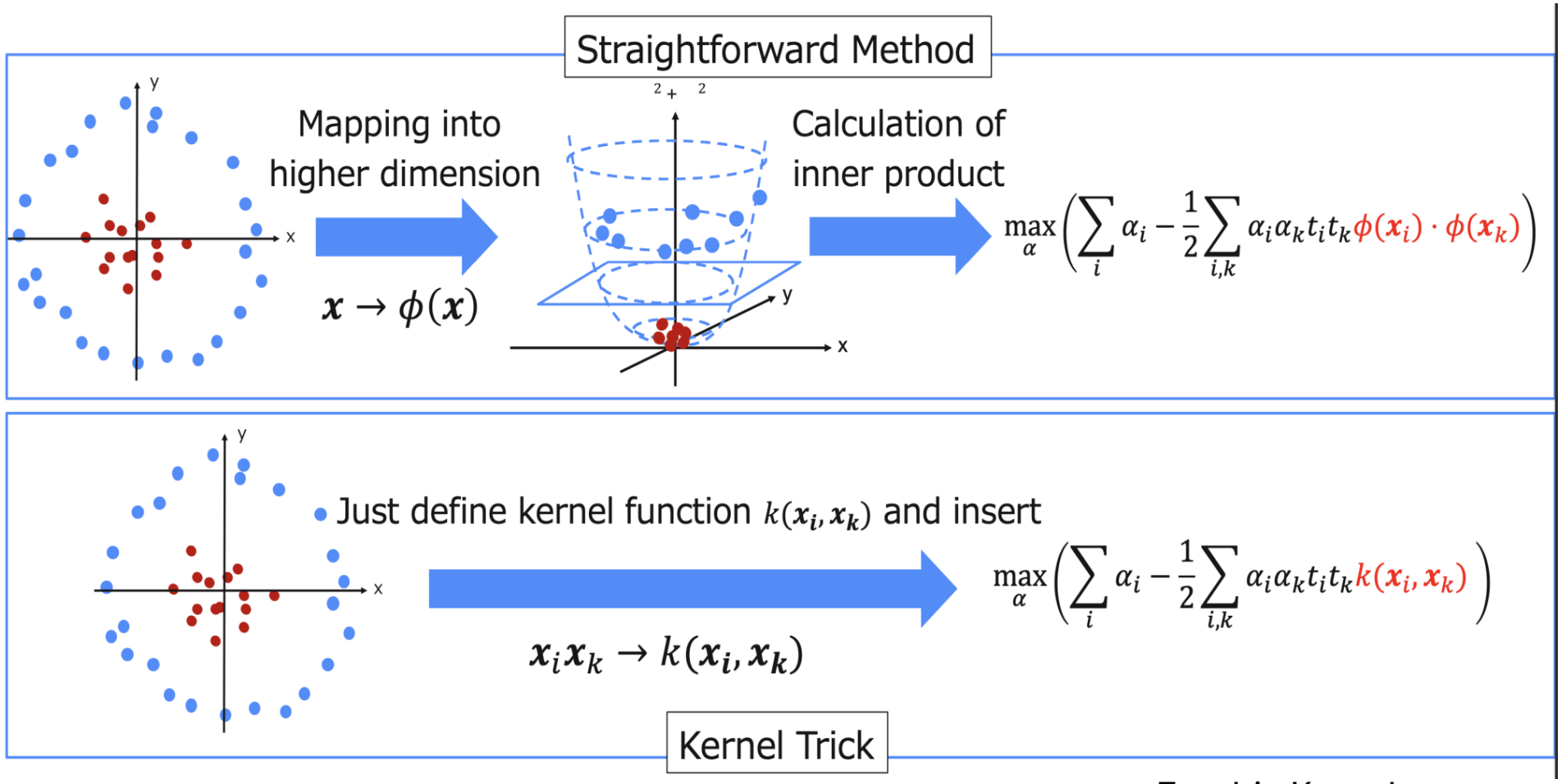




Boundary between categories with a Hyperplane allows to classify unknown cases

- Q1: How do we transform the data in such a way that a separator can be drawn as a hyperplane?
- Q2: How can we find the best optimized hyperplane?

The SVM “Kernel Trick”



For this Kernel,
Quantum Computing is applied

Mapping data into a higher dimensional space is called “**Kernelling**”

There are different mapping or transformation functions

Most of them are implemented in Data Science Programming languages

Introduction to Quantum Kernels

- The "**Quantum Kernel Method**" refers to any method that *uses Quantum Computers to estimate a Kernel Matrix or its individual entries*
- The **Kernel Function K** takes vectors in the feature-mapped space as arguments and returns their inner product

$$K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \text{ with } K(x, y) = \langle \Phi(x) | \Phi(y) \rangle$$

$\Phi(\vec{x})$ is a mapping from $\vec{x} \in \mathbb{R}^d$ to $\Phi(\vec{x}) \in \mathbb{R}^{d'}$, where usually $d' > d$

In the method of **Quantum Kernels**, the Feature Mapping is done by a **Quantum Circuit**, and the Kernel is estimated using **Measurements on that Circuit** and the Relative Measurement Probabilities

- We are interested in **Feature Maps Φ** for which the Kernel Function is **easy to evaluate**
- This often means finding a Kernel Function for which the **inner product in the feature-mapped space can be written *in terms of the original data vectors, without having to ever construct $\Phi(x)$ and $\Phi(y)$***

Qiskit Patterns
Step 1 – Map Classical Inputs to a Quantum Problem

- Input: Training dataset
- Output: Abstract circuit for calculating a **Kernel Matrix Entry**

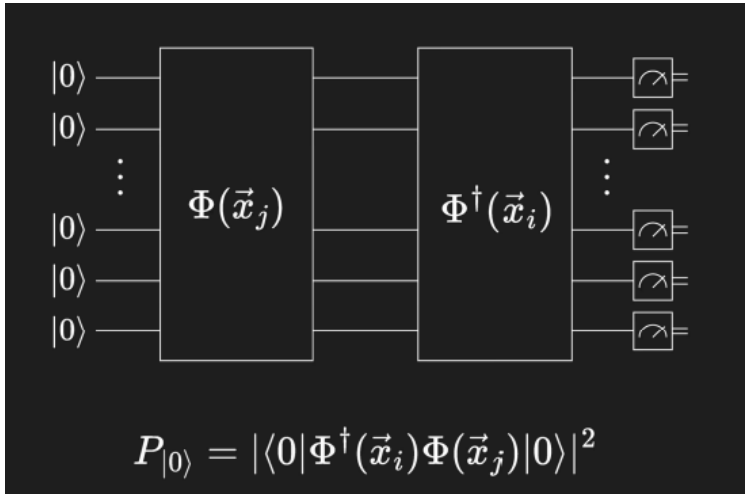
- Given the dataset, the starting point is to encode the data into a **Quantum Circuit**
- We select a data dependent Quantum Circuit and map **two data vectors** into the Feature Hilbert Space of States of our Quantum Computer
- We can construct **our own circuit** to encode your data, or you can use a **pre-made Feature Map** like `zz_feature_map`

- To calculate a **single** Quantum Kernel Matrix element, we will want to encode two different points, so we can estimate **their inner product**
- A full Quantum Kernel Matrix will involve an **iteration of many such inner products** between mapped data vectors
- Qiskit: combine unitaries U_1 and U_2 using `unitary_overlap`

unitary_overlap is a tool for comparing how close two Quantum Operations (unitaries) are to each other.

Qiskit Patterns
Step 1 (Cont.)

For the task of generating a Kernel Matrix, we are particularly interested in the probability of measuring the state $|0\rangle^{\otimes N}$, in which all Qubits are in the state $|0\rangle$



Why ?

- Suppose the encoding of two data vectors is as follows:

$$|\psi(\vec{x}_i)\rangle = \Phi(\vec{x}_i)|0\rangle^{\otimes N}$$

$$|\psi(\vec{x}_j)\rangle = \Phi(\vec{x}_j)|0\rangle^{\otimes N}.$$

- Then, the desired Quantum Kernel Entry is the inner product

$$\langle\psi(\vec{x}_j)|\psi(\vec{x}_i)\rangle = \langle 0|^{\otimes N}\Phi^\dagger(\vec{x}_j)\Phi(\vec{x}_i)|0\rangle^{\otimes N}$$

- The probability to measure the state $|0\rangle$ is

$$P_0 = |\langle 0|^{\otimes N}\Phi^\dagger(\vec{x}_j)\Phi(\vec{x}_i)|0\rangle^{\otimes N}|^2.$$

- The Measurement Layer of our circuit will return **measurement probabilities** (or so-called "*Quasi-Probabilities*", if certain error mitigation methods are used

Qiskit Patterns
Step 2 – Optimize the Problem for Quantum Execution

- Input: Abstract circuit, not optimized for a particular backend
- Output: Target Circuit and observable, **optimized for the selected QPU**

- We use the `generate_preset_pass_manager` function from Qiskit to specify an optimization routine for our circuit with respect to the real Quantum Computer on which we plan to run the experiment.

  ✓ The `optimization_level=3` provides the **highest level of optimization**.
  ✓ In this context, "optimization" refers to **optimizing the implementation** of the circuit on a **real Quantum Computer**

- `Generate_preset_pass_manager` is used to quickly generate a preset pass manager
- Preset pass managers are the default pass managers used by the `transpile()` function
- This function provides a convenient and simple method to construct a standalone `PassManager` object

Qiskit Patterns
Step 2 – Optimize the Problem for Quantum Execution (Cont.)

- Input: Abstract circuit, not optimized for a particular backend
- Output: Target Circuit and observable, **optimized for the selected QPU**

- Optimization includes considerations like **selecting physical Qubits** to correspond to Qubits in the abstract quantum **circuit that will minimize depth** or selecting **physical qubits with the lowest available error rates**

- *This is not related to optimization of the Machine Learning problem (as in classical optimizers like COBYLA*)

- You have to **optimize the circuit more than once**, since **each pair of points** involved in a matrix element produces a **different circuit** to be measured.

- Qiskit: `least_busy` backend

Qiskit Patterns
Step 3 - Execute using Qiskit Runtime Primitives

- Input: Target circuit
- Output: Probability distribution

- Use the **Sampler Primitive** from Qiskit Runtime to reconstruct a probability distribution of states yielded from sampling the circuit

- This may be referred to as a "**quasi-probability distribution**“
  - ✓ This term is applicable where **noise is an issue** and when extra steps are introduced, **such as in error mitigation**
  - ✓ In such cases, **the sum of all probabilities may not exactly equal 1**

- Since we optimized the circuit for the backend in Step 2, we can avoid doing Transpilation on the Runtime server by setting `skip_transpilation=True` and passing the optimized circuit to the Sampler

Sampler
- Runs Quantum Circuits and returns **quasi-probability distributions** over **measurement outcomes**
- For tasks where you need shot-based sampling

Estimator
- Evaluates **expectation values** of observables given **a quantum state** prepared by a circuit.
- When you need numerical values like $\langle\psi|0|\psi\rangle$ for Optimization, Variational Algorithms, or Cost-Function evaluation

Qiskit Patterns
Step 4 - Post-Process, return result in Classical Format

- Input: Probability distribution
- Output: A single Kernel Matrix Element, or a Kernel Matrix if repeating

- Calculate the probability of measuring on the Quantum Circuit
- **Populate the Kernel Matrix** in the position corresponding to the two data vectors used
- To fill out the entire kernel matrix, we need to run a **quantum experiment for each entry**
- Once we have a Kernel Matrix, **we can use it in many Classical Machine Learning** Algorithms that accept pre-calculated kernels
    - For example: `qml_svc = SVC(kernel="precomputed")`
    - Next, use `SVC.fit` and `SVC.score`
- We can then use classical workstreams to apply our model on our **Test Data**, and get an **Accuracy Score**

- Depending on our satisfaction with our accuracy score, we may need to revisit aspects of our calculation, such as our **Feature Map**

Considerations

- Using **z_feature_map** results in low 2-Qubit Transpiled Depth
- Using **zz_feature_map** results in higher 2-Qubit Transpiled Depth
- Too deep circuits result in a lot of noise
  - ✓ A 2-Qubit Transpiled Depth of 10 or fewer should be no problem
  - ✓ A 2-Qubit Transpiled Depth of 50-60 will require **Advanced Error Mitigation**

- The Symmetry of the Kernel Matrix can be leveraged to reduce the number of calculations by ½

- The Diagonal Elements can be simply set to 1, as they should be in the absence of noise

- The process must be done for the **Training and Test Data**

Considerations

- In very large complex circuits, the pass manager can return different transpiled circuits each time it runs → **it is useful to transpile a few times and take the shallowest circuit**

- **The dimension of the Kernel Matrix is dictated by the number of points in the Training Data, not the number of Features**
- → **small datasets with large numbers of features can still yield effective classification**

F1	F2	F3	.	.	.	.	.																		.	.	.	L
																												0
																												1
																												1

- **There is still much work to be done to characterize different Data Types and Data Relationships to see where Quantum Kernels will be most useful**

Areas of Investigation

- **How robust is the Accuracy**? Does it hold **for *broad types of data*** or just this specific training data? **No over-fitting?**
- What **structure in your data** makes you suspect that a Quantum Feature map is useful?
- How is the Accuracy affected by **increasing/decreasing the amount of Training Data**?
- What Feature Maps can you use and how do the **results vary with Feature Maps**?
- How are the Accuracy and running time affected by **increasing the number of Features**?
- **How to scale** Kernel Estimation up to **more and more Features?**
- Which **trends**, if any, do we expect to hold on **Real Quantum Computers**?

Review

Calculating a Quantum Kernel involves …

- Calculating Kernel Matrix entries, using **pairs of training data points**
- Encoding the data and mapping it via a **Feature Mapping**
- **Optimizing your circuit** for running on real Quantum Computers / backends

The Quantum Kernel can then be used in Classical Machine Learning algorithms

Some key things to keep in mind when using Quantum Kernels:

- Is the **dataset likely to benefit** from Quantum Kernel Methods?
- Try **different Feature Maps** and Entanglement Schemes.
- Is the **Circuit Depth** acceptable?
- Try running a **Pass Manager multiple times** and use the smallest depth circuit you can get.

Quantum Kernel Methods are potentially powerful tools, given a proper match between Datasets with **Quantum-Amenable Features** and a **suitable Feature Map**

Useful Paper: https://www.nature.com/articles/s41567-021-01287-z

Quantum Kernels

End of Topic 4

Quantum Variational Circuits & Quantum Neural Networks

Topic 5

Introduction to Quantum Neural Networks

- **Quantum Neural Networks (QNN) are a subset of Variational Quantum Classifiers (VQC) or Parametrized Quantum Circuits (PQC)**
- It *is not reasonable* to expect a Quantum Speed-Up for simple cases where Classical Algorithms are so efficient already
- Example:
  - Training data consists of **70 images** containing horizontal and vertical stripes
  - Test data consists of **30 images** containing also horizontal and vertical stripes
  - Our goal is to label unseen images into one of two categories depending on the orientation of their line
  - Full row or column: pixel value of $\frac{\pi}{2}$
  - Remaining pixels get a random value in $\left[0, \frac{\pi}{4}\right]$, representing noise in the data
  - Labels are $+1$ (image contains vertical line) and -1 (image contains horizontal line)

This can be accomplished with a VQC but it does not outperform a Classical Algorithm

Map the Problem to a Quantum Circuit – Step 1

- **Load Training Data**
  - By hand: Basis, Amplitude, Angle, Phase, Dense Angle
  - Feature Map
- **Construct an Ansatz**
  - Containing Rotation and Entanglement Layers that are appropriate for the problem
- **Monitor Circuit Depth to ensure quality results on real Quantum Computers**

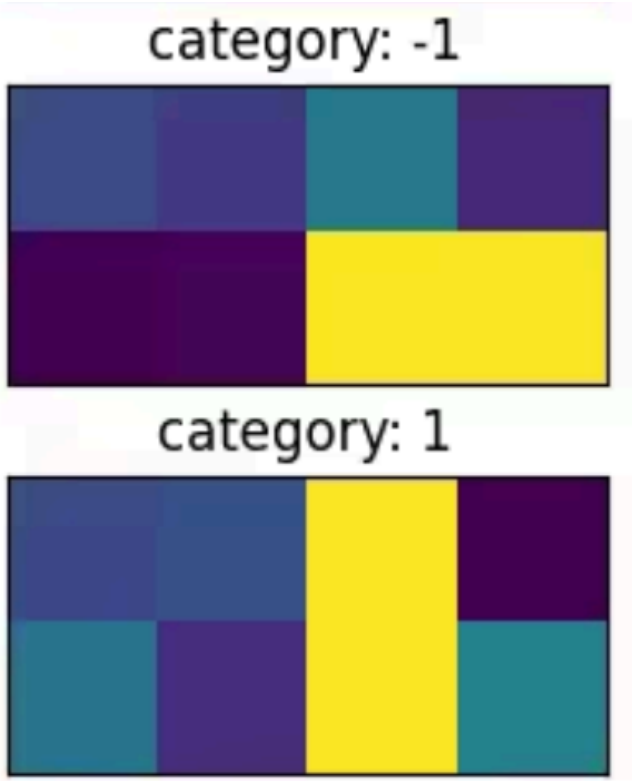
Variational Quantum Classifier – Step 1

- We map problem to Quantum Circuit

<u>Goal:</u> find a function f with parameters θ that maps a data vector (or image) to the correct category: $f_\theta(\vec{x}) \rightarrow \pm 1$

$$f_\theta(\vec{x}) = \langle 0|U^\dagger(\vec{x})W^\dagger(\theta)OW(\theta)U(\vec{x})|0\rangle$$

- $U(\vec{x})$ is **the encoding circuit** – E.g. **Feature Map**
- $W(\theta)$ is a variational or trainable circuit block
  - ✓ Also named the "**Ansatz**"
- θ is the **set of parameters to be trained**, that yield the best classification of images by the Quantum Circuit
- O is some **Observable to be estimated with the Estimator Primitive**
- We could have **multiple Variational and/or Encoding Layers** in any order, that is technically motivated



Observable

- The **Observable** is used in the **Cost function**
- We obtain an **Expectation Value** for the Observable using the **Estimator**
- Convolutional Layers are often used
  - ✓ A Convolutional Layer combines information onto fewer Qubits
  - ✓ As such, Measurements are only needed on a subset of the Qubits in the Circuit
- Often, **a Z Operator is included for each Measured Qubit**
- A Measurement of a Z can yield 2 possible outcomes: the **Eigenvalues of Z are $+1$ and -1**
- **As such, we often Measure in Pauli Z basis**

Loss Function

- Calculates the difference between **predicted** and **real values** of the labels
- General example of Loss Function: **Mean Squared Error** (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

n = number of data points

y_i = actual or true value

$\hat{y}_i$ = predicted value

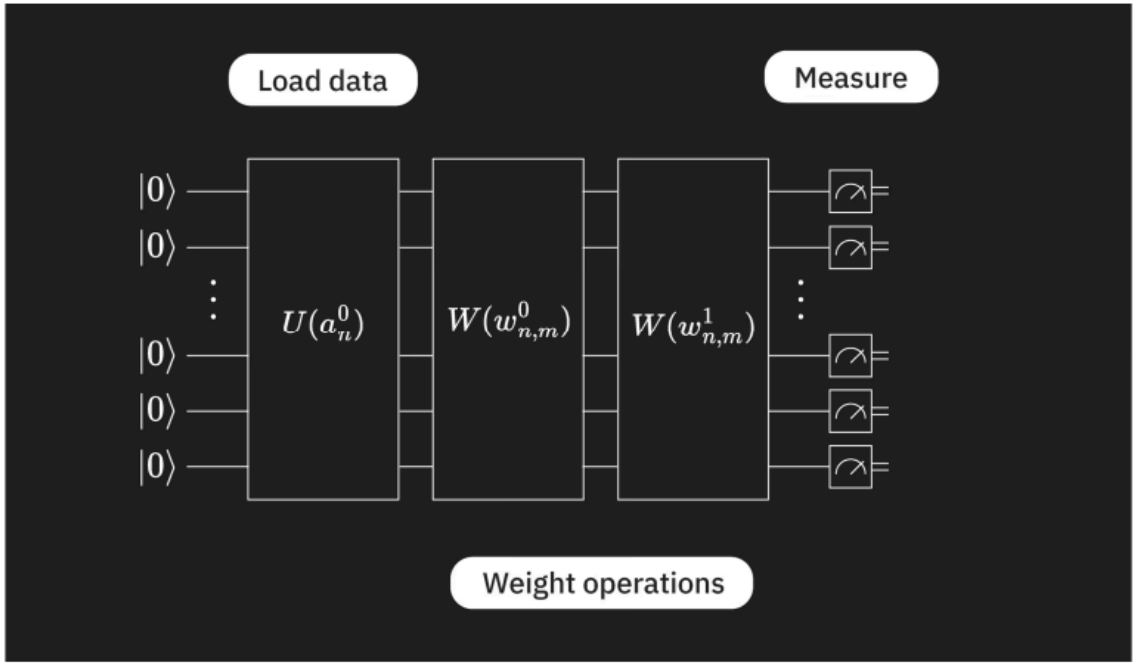
- Here, the **Loss Function is a function of the variable parameters or weights**
  - ✓ Used by Classical Optimizer
  - ✓ Example: **COBYLA**
- The Classical Optimizer will train the model by sampling different weights and **attempting to lower the output of the Loss Function**

COBYLA (Constrained Optimization BY Linear Approximations)

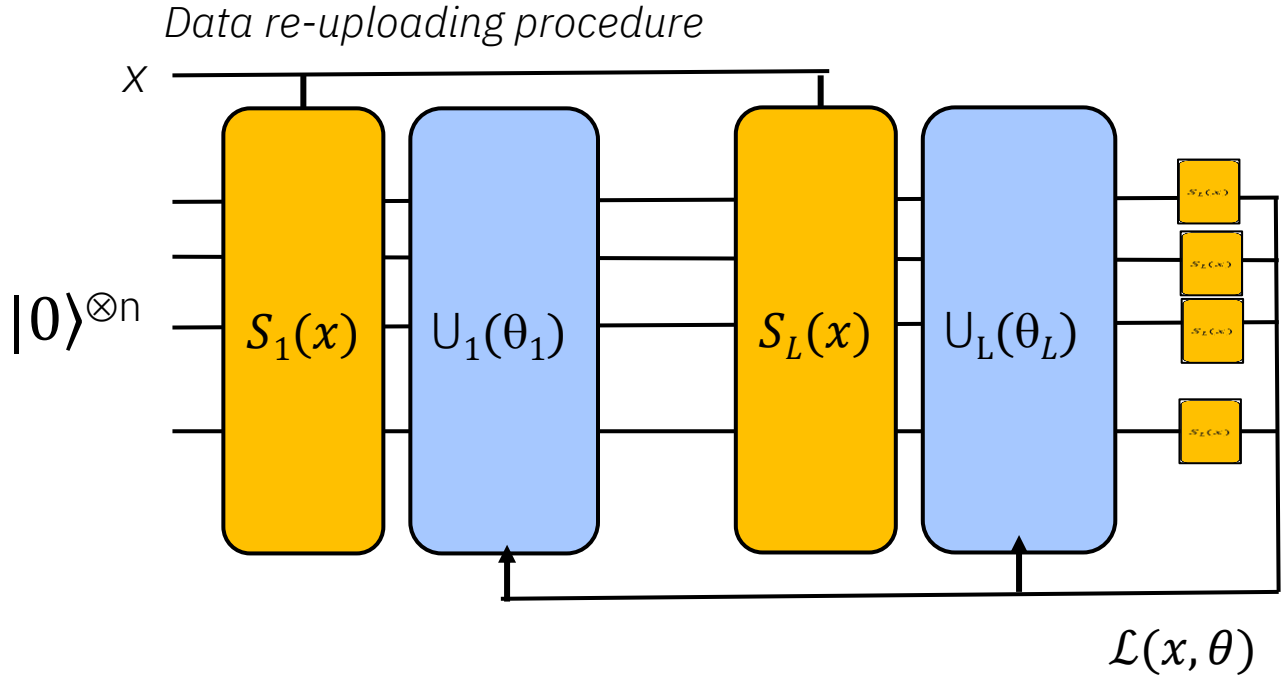
- **Gradient-free** optimization algorithm that solves nonlinear constrained problems by approximating the objective function with linear models.
- It's often used in **Quantum Computing and ML** when derivatives are hard to compute.
- Alternatives include Nelder–Mead, Powell, L-BFGS-B, and SLSQP

The Parameters are Variable “Weights”

- $U(\vec{x})$ loads input values (for example from an Image)
- $W(\theta)$ represents layer of variable weights



- $S_i(\vec{x})$ loads input values (for example from an Image)
- $U_i(\theta_i)$ represents layer of variable weights



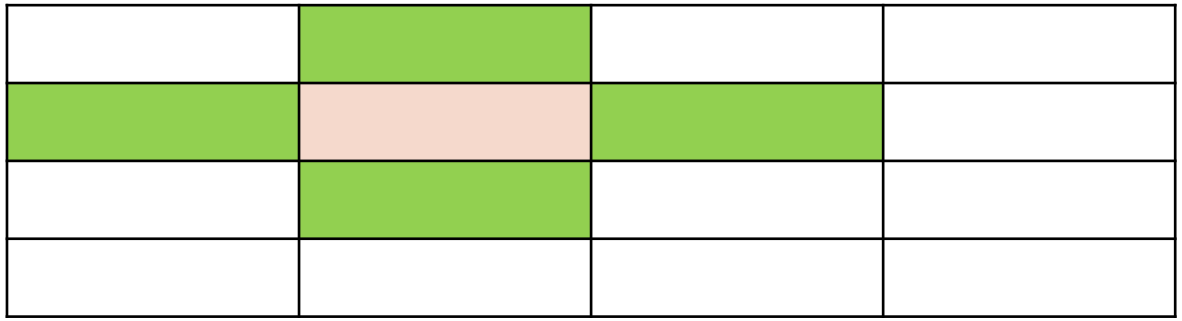
Selecting the Ansatz - Considerations

Hardware

- Ansatz that is excessively deep will not produce good results on a Noisy Device
- Transpilation increases the depth even more
- Entangling very distant Qubits can increase depth substantially due to SWAP Gates

Problem

- Use information about the problem that can guide the Ansatz
- E.g.: for horizontal/vertical line discovery with pixels, think about **CNOT gates** to correlate the Qubits that correspond to **adjacent pixels**



Selecting the Ansatz – Considerations (Cont.)

Number of Parameters

- **Each independently parametrized Quantum Gate** in the Circuit increases the space to be (Classically) Optimized and this results in slower convergence

- With scaling up, **Barren Plateaus** can be encountered
  - ✓Phenomenon where **optimization landscape becomes exponentially flat** as the problem size increases
  - ✓This makes it **difficult to train the algorithm**
  - ✓Barent Plateaus are relevant to VQC, QNN, and other VQAs
  - ✓You can avoid its appearance by decreasing the number of parameters, another cost function and parameter initialization
  - ✓See next slide for more details

The Challenge of the Barren Plateaus

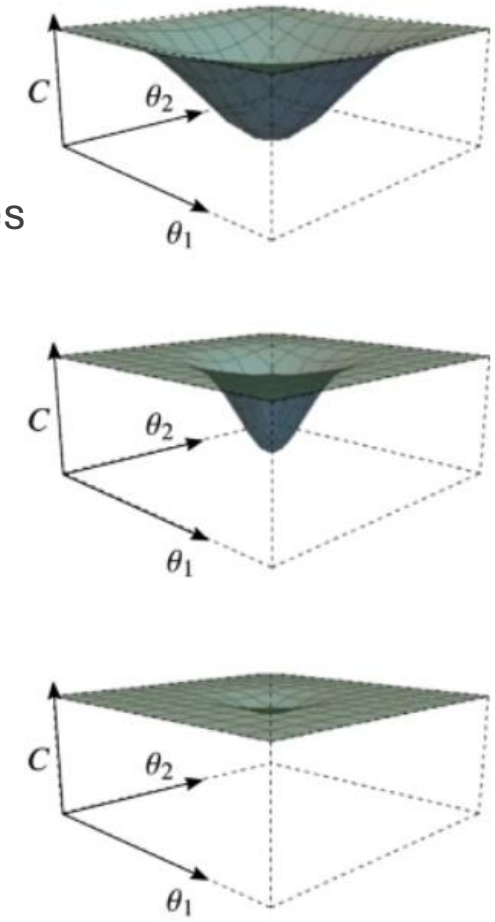
Key "Victims"

- **Noisy** Near-Term Quantum Computers
- **Highly expressive** Circuits and large Hilbert Spaces
- **Deep** Quantum Circuits
- Circuits with **global measurements**
- **High entanglement** entropy
- **Dissipative** Quantum Neural Networks

Convolutional QNNs are **resilient** to Barren Plateaus

[See: Pesah et al., Phys Rev X, 2021]

https://journals.aps.org/prx/pdf/10.1103/PhysRevX.11.041011



Some strategies to **avoid or mitigate Barren Plateaus**

- Initialize and train parameters in batches
- Reduce depth of circuit by randomly initializing a subset of the total number of parameters
- Introduce correlations between the parameters in multiple layers of the QNN to reduce the overall dimensionality of the parameter space
- Leverage Classical recurrent NNs to find good parameter initializing heuristics such that the network starts close to a minimum
- Choice of local cost functions and specific entanglements between hidden and visible units

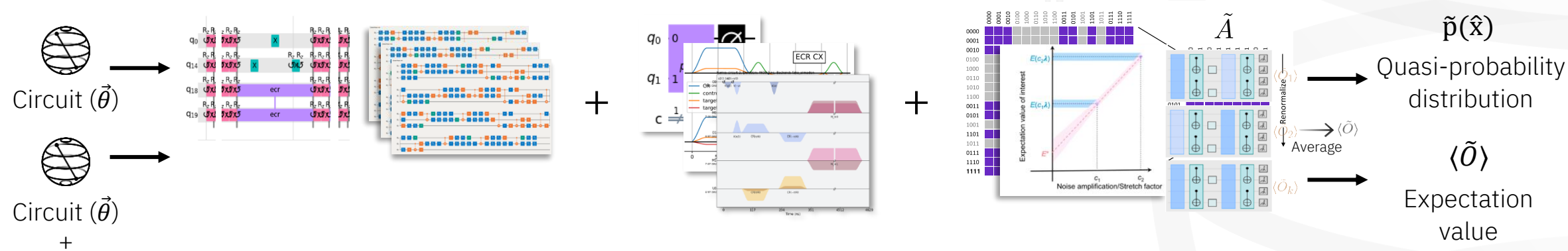
Optimization – Step 2

- The Pass Manager is used
  - It might make sense to run the Pass Manager several times and find the **best transpiled circuit, i.e. the shallowest circuit**
- We use the `least-busy backend`
- We can use **Error Mitigation** and **Error Suppression** techniques
  - ✓ Dynamical Decoupling
  - ✓ Gate Twirling
  - ✓ Twirled Readout Extinction
  - ✓ Zero Noise Extrapolation
  - ✓ Probabilistic Error Cancellation

Error Mitigation and Error Suppression

Qiskit Runtime

The engine for performant execution of Primitives powered by Error Suppression, Runtime Compilation , Error Mitigation.



Error suppression

Ex.
Dynamical Decoupling,
Pauli Twirling

Runtime compilation

Ex. Efficient parameter updating, time-scheduling and gate/operation parallelization, controller code generation

Error mitigation

Ex.
Zero-Noise Extrapolation (ZNE)
Probabilistic Error Cancellation (PEC)
Twirled Readout Error Extinction (TREX)

Dynamical Decoupling (DD) Insertion Pass

- Works on a scheduled, **physical circuit**
- ***It scans the circuit for idle periods of time and inserts a DD sequence of gates in those spots***
- These gates amount to the **Identity**, ***so do not alter the logical action of the circuit***
- But they have the effect of ***mitigating decoherence in those idle periods***

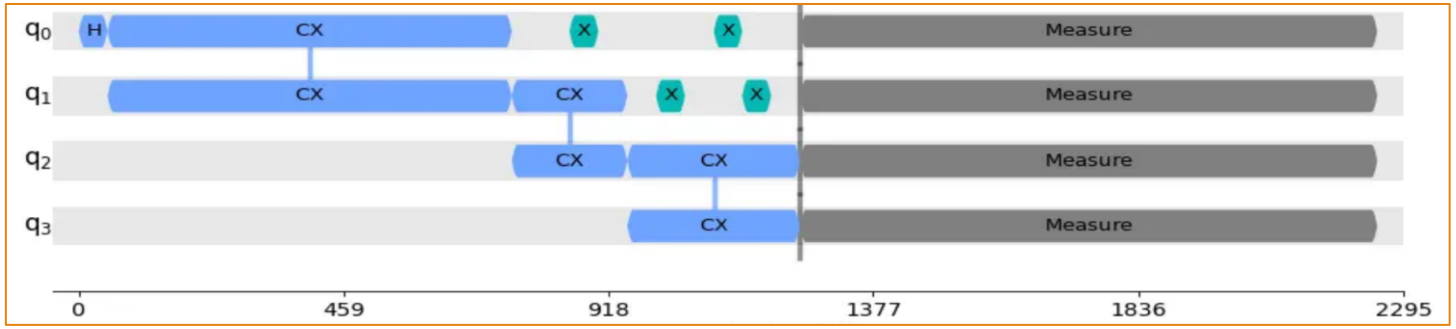
- This pass ensures that the inserted sequence **preserves the circuit logic exactly** (including global phase).

- **ALAPScheduleAnalysis**: schedules the stop time of instructions "as late as possible".

$$X.X = I$$

```
circ = QuantumCircuit(4)
circ.h(0)
circ.cx(0, 1)
circ.cx(1, 2)
circ.cx(2, 3)
circ.measure_all()
durations = InstructionDurations(
   [("h", 0, 50), ("cx", [0, 1], 700), ("reset", None, 10),
    ("cx", [1, 2], 200), ("cx", [2, 3], 300),
    ("x", None, 50), ("measure", None, 1000)])
dd_sequence = [XGate(), XGate()]
pm = PassManager([ALAPScheduleAnalysis(durations),
                  DynamicalDecoupling(durations, dd_sequence)])
circ_dd = pm.run(circ)
```

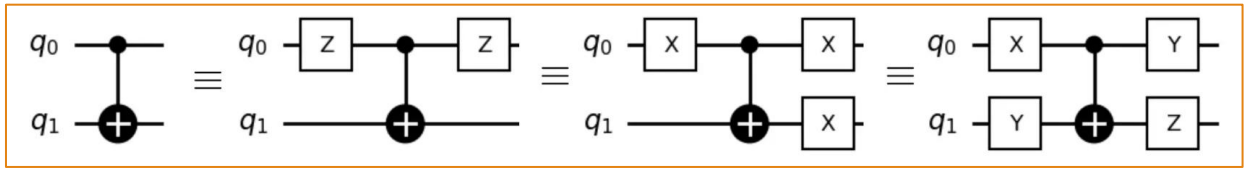
Sequence_types = XX, YXYX, XYXYYXYX



Gate Twirling and Pauli Twirling

- **Twirling**, also known as **randomized compiling**, is a widely used technique for **converting arbitrary noise channels into noise channels with more structure**
- Basically, it involves randomizing certain aspects of quantum operations to **make the noise more uniform and easier to handle.**
- It is a technique to "*symmetrize*" the noise
- **Pauli Twirling** is a special kind of twirling that uses **Pauli operations**
- Performed alone, **it can mitigate coherent noise**
- *Coherent noise tends to accumulate quadratically with the number of operations, whereas Pauli Noise accumulates linearly*.
- **Pauli Twirling is often combined with other error mitigation techniques** that **work better with Pauli noise than with arbitrary noise**

- Implemented by **sandwiching** a chosen set of gates **with randomly chosen single-qubit Pauli gates** in such a way that the ideal effect of the gate remains the same.
- The result is that a single circuit is replaced with a **random ensemble of circuits**, all with the same ideal effect.
- Samples are drawn from multiple random instances, rather than just a single one
- Since most of the errors come from 2-Qubit gates, this technique is **often applied exclusively to 2-Qubit gates**, for example…



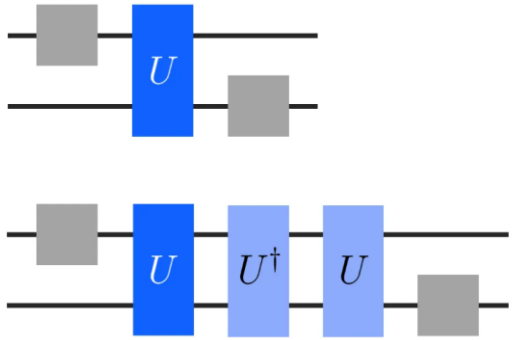
Zero Noise Extrapolation (ZNE)

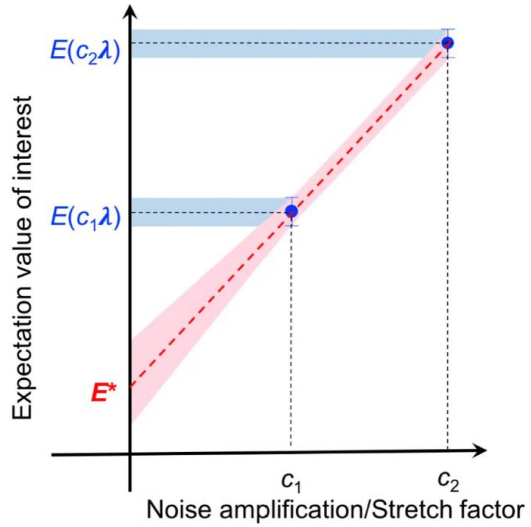
- ZNE is a technique for **mitigating errors in *estimating expectation values of observables***.
- It often improves results but does not guarantee to produce an unbiased result.

- ZNE consists of two stages:
  1. **Noise Amplification**: The original quantum circuit is executed multiple times at different noise rates.
  2. **Extrapolation**: The ideal result is estimated by extrapolating the noisy expectation value results to the zero-noise limit.

- **Qiskit Runtime** implements Noise Amplification by "**Digital Gate Folding**": 2-Qubit gates are replaced with equivalent sequences of the gate and its inverse
- Example: replacing a unitary U with $UU^{+}U$ would yield a Noise Amplification factor of 3
- For the extrapolation, you can choose from one of several functional forms, including a **linear fit or an exponential fit**

Digital Gate Folding





Probabilistic Error Cancellation (PEC)

- PEC is a technique for mitigating errors in ***estimating expectation values of observables***
- Unlike ZNE, it returns an unbiased estimate of the expectation value
- In PEC, **the effect of an ideal target circuit is expressed as a linear combination of noisy circuits that are actually implementable in practice**
- The output of the ideal circuit can then be reproduced by executing different noisy circuit instances drawn from a random ensemble
- However, it generally incurs a **greater overhead**
  1. The sampling overhead is a **multiplicative factor on the number of shots**
  2. It scales **quadratically with γ, which in turn scales exponentially with the depth of the circuit**

$$\mathcal{O}_{\text{ideal}} = \sum_i \eta_i \mathcal{O}_{noisy,i}$$

The coefficients η_i form a *quasi-probability distribution* because some of the coefficients are negative

$$\gamma = \sum_i |\eta_i| \geq 1.$$

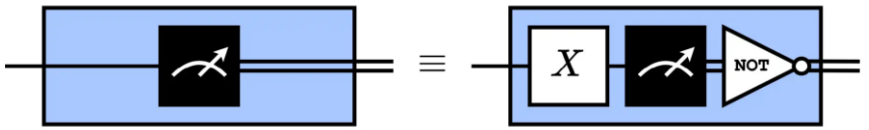
Measurement Error Mitigation

Twirled Readout Error Extinction (TREX)

- **TREX mitigates the effect of measurement errors for the estimation of *Pauli observable expectation values***
- It is based on the notion of **twirled measurements**
- Like in standard Gate Twirling, this sequence is **equivalent** to a plain measurement in the absence of noise
- In the presence of readout error, measurement twirling has the **effect of diagonalizing the Readout-Error Transfer Matrix, making it easy to invert**
- Estimating the Readout-Error Transfer Matrix **requires executing additional calibration circuits, which introduces a small overhead**

Twirled Measurements are accomplished by randomly substituting measurement gates by a sequence of

(1) a Pauli X gate
(2) a measurement
(3) classical bit flip



Execute using Qiskit Primitives – Step 3

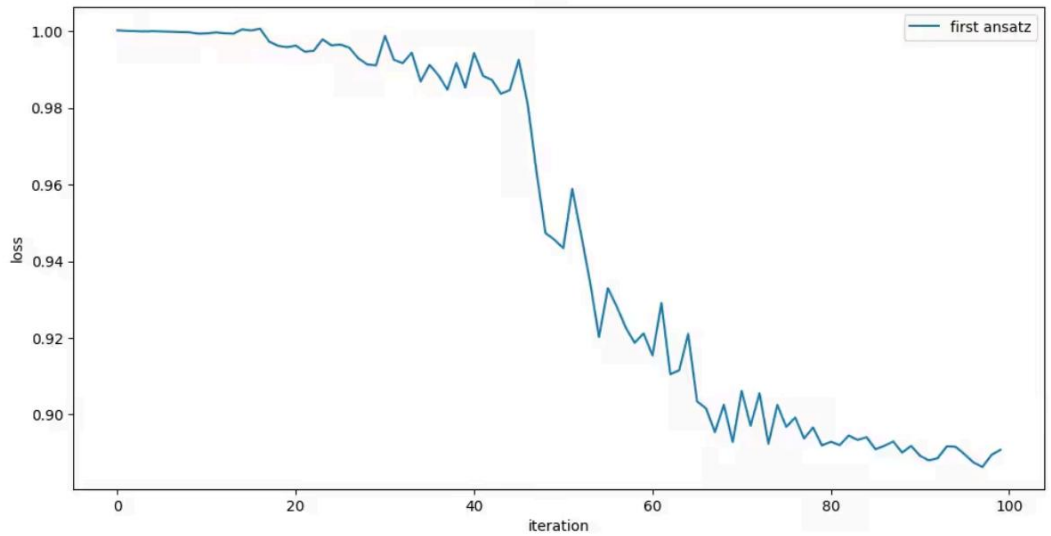
- Use the desired number of **epochs** to train the **Quantum Neural Network**
- Carry out preliminary trials on **Simulators** to optimize the Ansatz
- Execute on **Quantum Computer**
- An execution on a **Fake Device** can also be helpful

- An epoch in Machine Learning refers to **one complete pass through the entire training dataset** by the learning algorithm.
- During this pass, the model processes all data samples, calculates the loss, and updates its parameters (e.g., weights in neural networks) to minimize the error
- Training typically involves **multiple epochs** to iteratively improve the model's performance.

Post Process – Step 4

- Return the result in Classical Format
- Calculate the **Accuracy for the Training Data**
- Calculate the **Accuracy for the Test Data**
- Monitor **convergence** of Classical Optimization
- If the accuracy is below 75 %
  - Consider alternative Ansatz
  - Evaluate number of parameters
  - Assess the Entanglement method used
  - Improve mapping of problem to Quantum Circuit – Data Encoding

Output:



Quantum Variational Circuits & Quantum Neural Networks

End of Topic 5

Quantum Machine Learning

$\langle 10|Q\rangle$ for your attention!

Eric Michiels
Quantum Computational Scientist
Qiskit Advocate