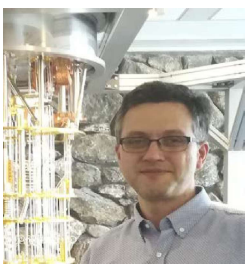
QML + Q FINANCE (SEMINAR +PRACTICE) ON-LINE 30 DE MAYO DE 2025

Guillermo Botella Juan, Ginés Carrascal de las Heras
gbotella@ucm.es; gines_carrascal@es.ibm.es

Schedule: Coarse Grain (Seminar + Practice)

- Guillermo Botella Juan
- Full Professor (UCM) Computer Architecture and Automation Dept
- "Arquitectura y Programación de Computadores cuánticos" Coordinator and Lecturer (Optative subject degree)

- Ginés Carrascal de la Heras
- Part-Time Professor at UFV and IBM employee
- Quantum Ambassador at IBM
- Performing the PhD regarding Quantum Finance Computing at UCM



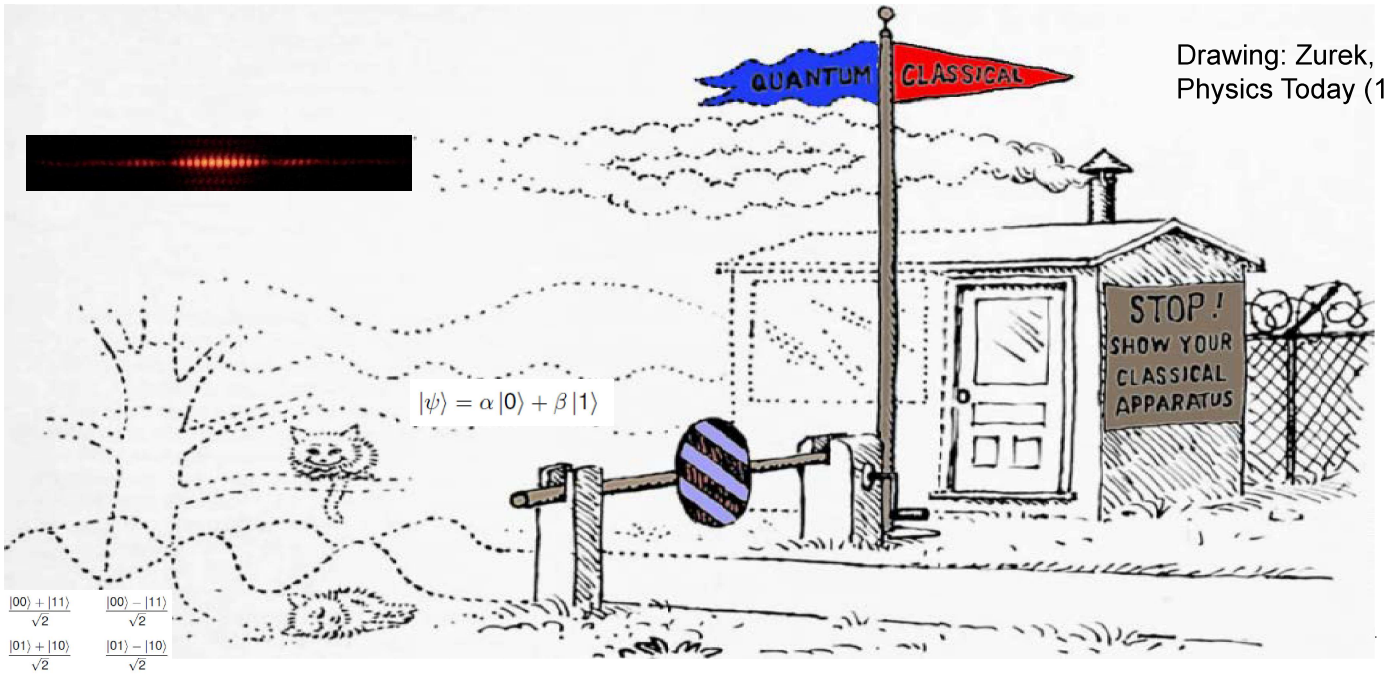
Schedule: Coarse Grain (Seminar + Practice)

- First Session: Q Machine Learning

  (Source of Pennylane content: https://pennylane.ai/qml/)

- Second Session: Q Finance
  - Preliminary Concepts: Finance using QC
  - Trading Optimization
  - Risk Profiling

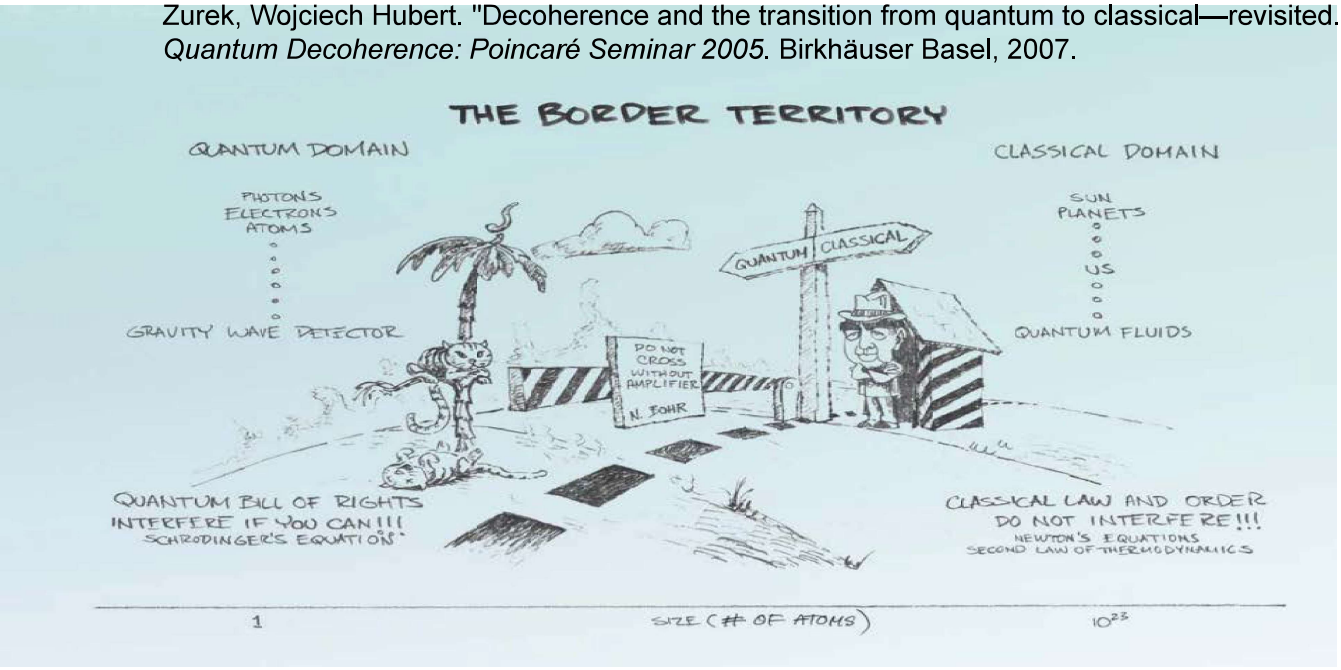
Q Machine Learning + Q Finance Computing



Drawing: Zurek,
Physics Today (1991)

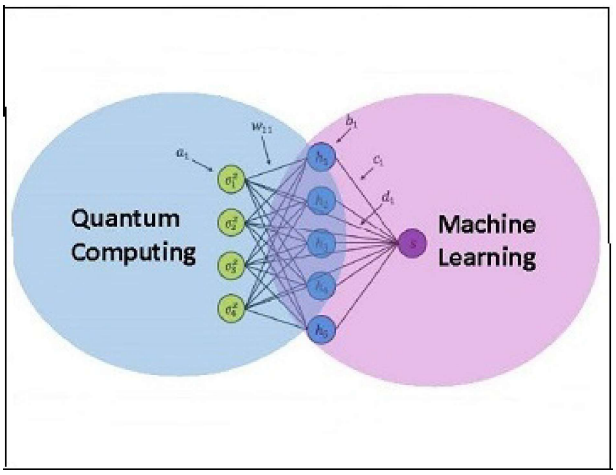
Q Machine Learning + Q Finance Computing

Zurek, Wojciech Hubert. "Decoherence and the transition from quantum to classical—revisited." *Quantum Decoherence: Poincaré Seminar 2005*. Birkhäuser Basel, 2007.



Schedule

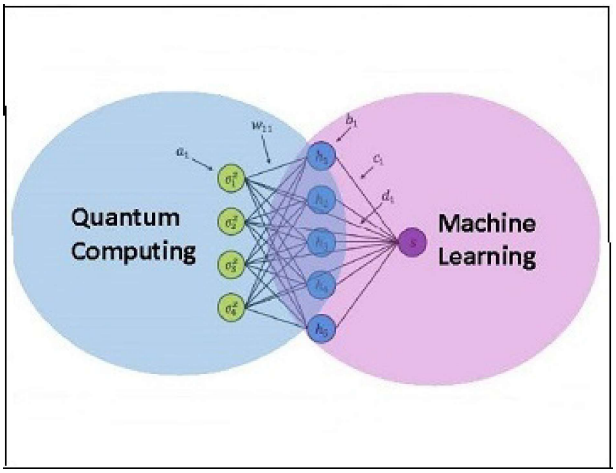
- ❑ **QML: General Concepts**
  - ❑ From ML to QML
  - ❑ Training Quantum Circuits
  - ❑ Tools → Pennylane use case
  - ❑ Brief summary of QML
- ❑ Example: Classic Classifiers
- ❑ Constructing Models
- ❑ Example: Quantum Classifier



Source: https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

Schedule

- **QML: General Concepts**
  - **From ML to QML**
    - **Example of ML (Financial)**



Source: https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

QML: Introduction

- **What can Machine Learning do for us ? (Finance example!)**
  - We imagine bank and loan requests
  - Before, several experts who had their own heuristics (accumulated experience) were consulted
  - Machine Learning → data analysis method that automates the creation of analytical models (really a changing in paradigm!)
    - E.g: Given customer ***x*** with a given set of these entries, he returned loan
    - E.g: Given customer ***y*** with a given set of these entries, he did not repay loan
    - We need a lot of data!
  - ML → Get patterns between all these inputs and outputs that automatically tell me if the loan is going to be repaid or not

QML: Introduction

- **Quantum Machine Learning (Finance)**
  - Steps we have to take:
    - Data preparation (e.g., cleaning, normalization, and selection)
    - Model development (e.g., feature engineering and algorithm selection)
    - Error estimation (e.g., cross-validation and accuracy metrics)

QML: Introduction

- **Quantum Machine Learning (Finance)**
  - **Data preparation**
    - Obtain data of sufficient quality and of sufficient quantity
      - E.g: Customer data, medical data ...
    - Once acquired, the data must be cleaned (make it understandable by the model)
      - E.g: Fields that are not filled in completely
      - E.g: Dimensionality reduction → we can discard some data
        - Take into account we have “small” QC
      - E.g: Model that works with data in a specific format (numeric)

QML: Introduction

❑ Quantum Machine Learning (Finance)

❑ Model Development

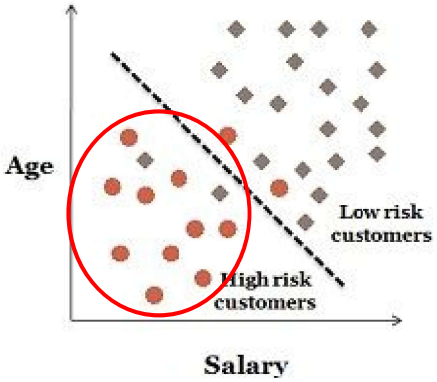
- ❑ Model as a function that takes as input the data of a client and as output the fact of granting the loan or not (for example a normalized score of type 0..1 or 1 ..- 1, etc..)
- ❑ Models are usually defined with a set of parameters, let's see the simplest of all using the linear classification
- ❑ We intuitively label the salary in the abscissa and the age in the ordinate and we see those who have repaid the loan and those who have not
- ❑ We will have a line, in a hyperplane that will divide the space between the sets that do not repay the loan and those that do repay the loan

QML: Introduction

❑ Quantum Machine Learning (Finance)

❑ Example 1: Bank and loan requests

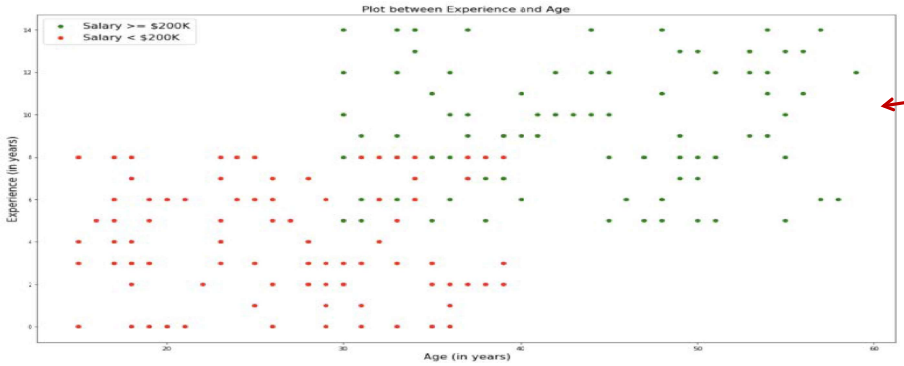
- ❑ We use a linear classification model, looking both sides of the line



QML: Introduction

❑ Quantum Machine Learning (Finance)

❑ Example 2: HR department



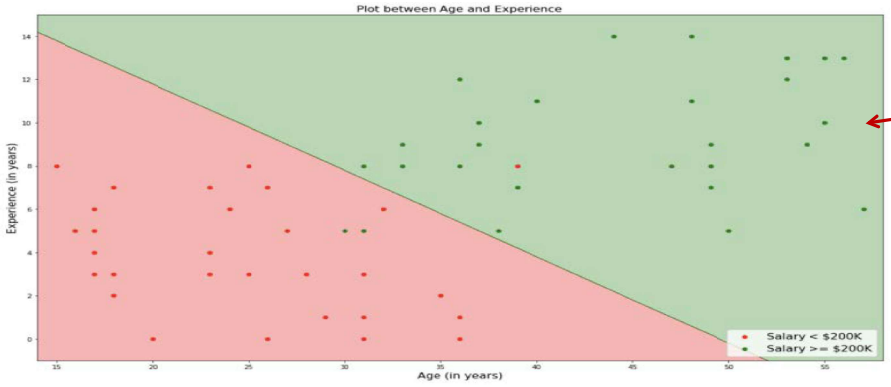
The GREEN points represent salary more than or equal to $200K and RED points represent salary less than $200K

SOURCE: https://towardsdatascience.com/machine-learning-classifier-evaluation-using-roc-and-cap-curves-7db60fe6b716

QML: Introduction

❑ Quantum Machine Learning (Finance)

❑ Example 2: HR department

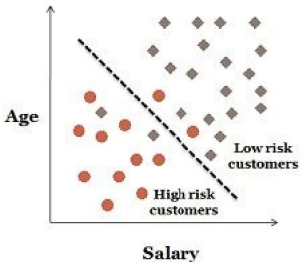


The GREEN points represent salary more than or equal to $200K and RED points represent salary less than $200K

SOURCE: https://towardsdatascience.com/machine-learning-classifier-evaluation-using-roc-and-cap-curves-7db60fe6b716

QML: Introduction

- **Quantum Machine Learning (Finance)**
  - **Model Development** → y = ax+ b
  - *a* is the slope and *b* is the ordinate at origin
  - Defining f(x,y) = y – ax –b
    - The points located on the line satisfy that f (x, y) = 0
    - The points located on one side of the hyperplane will satisfy that f (x, y) > 0
    - The points located on the other side of the hyperplane will satisfy that f (x, y) < 0

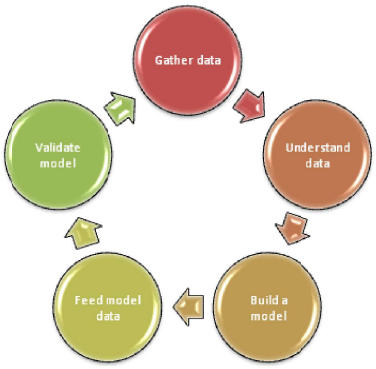


QML: Introduction

- **Quantum Machine Learning (Finance)**
  - **Model Development** → y = ax+ b
  - The classification function returned 1 or -1 depending on whether the loan is returned or not!
  - One further step. Apply sign function → sg (f (x,y))
    - Since given a new entry it returns 1 if it belongs to one set or -1 if it belongs to another (discrete classification)
  - We have to determine the parameters, because there are a lot of possible lines that we can create with a and b

QML: Introduction

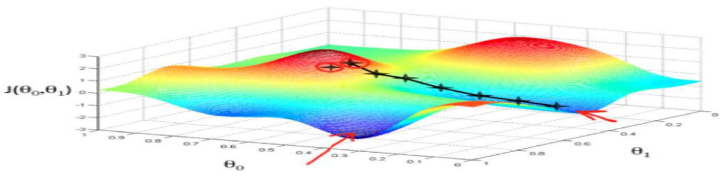
- **Quantum Machine Learning (Finance)**
  - **Error Estimation**
    - It will indicate how well we are adjusting the parameters
    - Adjust the parameters so that the final model is gradually refined with the minimum error
    - Model ←→ Error ←→ Parameters Fitting



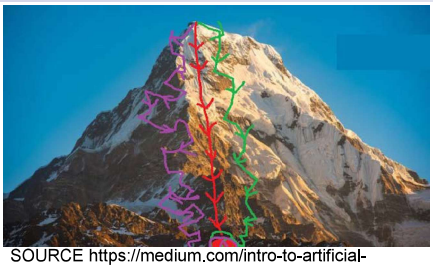
SOURCE https://www.kdnuggets.com/2019/01/fine-tune-machine-learning-models-forecasting.html

QML: Introduction

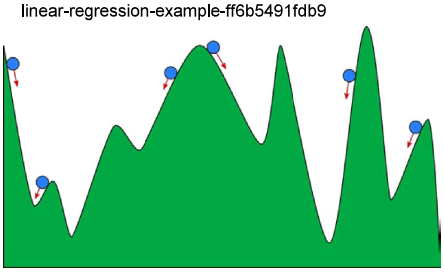
- **Quantum Machine Learning (Finance)**
  - **Error Estimation**
    - Ideal situation when the error function is differentiable!
      - Derivative is a vector that indicates the direction of maximum slope
      - We are doing iterations in order to reduce the error



    - In many procedures in QC we will not be able to “derive” and we will have to use other types of methods



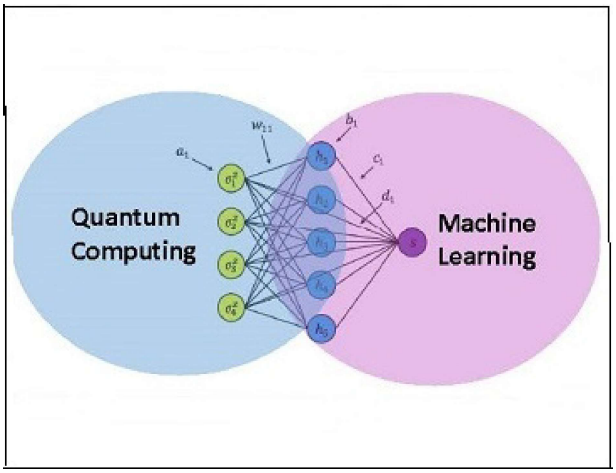
SOURCE https://medium.com/intro-to-artificial-intelligence/gradient-descent-algorithm-explained-with-linear-regression-example-ff6b5491fdb9



SOURCE https://towardsdatascience.com/machine-learning-101-an-intuitive-introduction-to-gradient-descent-366b77b52645

Schedule

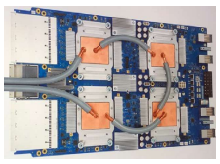
- ❑ **QML: General Concepts**
  - ❑ **From ML to QML**
    - ❑ **Example of ML (Financial)**
    - ❑ **Reasons for success!**

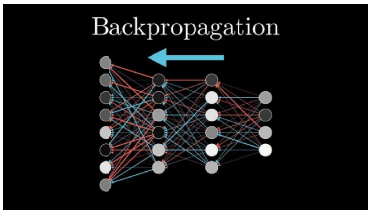


Source:
https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

QML: Introduction

- ❑ **Thus, ML successes nowadays**
  - ❑ Why is ML successful today?
    - ❑ Hardware advancements (GPUs, TPUs,…)
    - ❑ Workhorse algorithms
      (backpropagation, stochastic gradient descent)
    - ❑ Specialized, user-friendly software





TensorFlow

PyTorch

QML: Introduction

❑ ML successes nowadays

- ❑ Hardware advancements (GPUs, TPUs)

Large-scale Deep Unsupervised Learning using Graphics Processors

Rajat Raina RAJATR@CS.STANFORD.EDU
Anand Madhavan MANAND@STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford CA 94305 USA

*Rajat Raina, Anand Madhavan, and Andrew Y. Ng. 2009. Large-scale deep unsupervised learning using graphics processors. In Proceedings of the 26th Annual International Conference on Machine Learning (**ICML '09**). Association for Computing Machinery, New York, NY, USA, 873–880. DOI:https://doi.org/10.1145/1553374.1553486*

In this paper, we suggest massively parallel methods to help resolve these problems. We argue that modern graphics processors far surpass the computational capabilities of multicore CPUs, and have the potential to revolutionize the applicability of deep unsupervised learning methods. We develop general principles for massively parallelizing unsupervised learning tasks using graphics processors. We show that these principles can be applied to successfully scaling up learning algorithms for both DBNs and sparse coding. Our implementation of DBN learning is up to 70 times faster than a dual-core CPU implementation for large models. For example, we are able to reduce the time required to learn a four layer DBN with 100 million free parameters from several weeks to around a single day. For sparse coding, we develop a simple, inherently parallel algorithm, that leads to a 5 to 15-fold speedup over previous methods.

QML: Introduction

❑ ML successes nowadays

- ❑ Hardware advancements (GPUs, TPUs)

TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings

Industrial Product*

Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson
Google, Mountain View, CA

{jouppi,gkurian,lsheng,pcma,rahulnagarajan,lnai,nishantpatil,suvinay,aswing,btowles,cliffy,zhoux,zongweiz}@google.com and pattrsn@cs.berkeley.edu

ABSTRACT

In response to innovations in machine learning (ML) models, production workloads changed radically and rapidly. TPU v4 is the fifth Google domain specific architecture (DSA) and its third supercomputer for such ML models. Optical circuit switches (OCSes) dynamically reconfigure its interconnect topology to improve scale, availability, utilization, modularity, deployment, security, power, and performance; users can pick a twisted 3D torus topology if desired. Much cheaper, lower power, and faster than Infiniband, OCSes and underlying optical components are <5% of system cost and <3% of system power. Each TPU v4 includes SparseCores, dataflow processors that accelerate models that rely on embeddings by 5x–7x yet use only 5% of die area and power. Deployed since 2020, TPU v4 outperforms TPU v3 by 2.1x and improves performance/Watt by 2.7x. The TPU v4 supercomputer is 4x larger at 4096 chips and thus nearly 10x faster overall, which along with OCS flexibility and availability allows a large language model to train at an average of ~60% of peak FLOPS/second. For similar sized systems, it is ~4.3x–4.5x faster than the Graphcore IPU Bow and is 1.2x–1.7x faster and uses 1.3x–1.9x less power than the Nvidia A100. TPU v4s inside the energy-optimized warehouse scale computers of Google Cloud use ~2.6x less energy and produce ~20x less CO2e than contemporary DSAs in typical on-premise data centers.

CCS CONCEPTS

• **Computer systems organization → Architectures → Other architectures → Neural networks**

KEYWORDS

Machine learning, domain specific architecture, TPU, GPU, IPU, supercomputer, optical interconnect, reconfigurable, embeddings, large language model, power usage effectiveness, warehouse scale computer, carbon emissions, energy, CO2 equivalent emissions

*This paper is part of the Industry Track of ISCA 2023's program.

QML: Introduction

- **ML successes nowadays**
  - Hardware advancements (GPUs, TPUs)

A Survey: Collaborative Hardware and Software Design in the Era of Large Language Models

Cong Guo, Feng Cheng, Zhixu Du, James Kiessling, Jonathan Ku, Shiyu Li, Ziru Li, Mingyuan Ma, Tergel Molom-Ochir, Benjamin Morris, Haoxuan Shan, Jingwei Sun, Yitu Wang, Chiyue Wei, Xueying Wu, Yuhao Wu, Hao Frank Yang, Jingyang Zhang, Junyao Zhang, Qilin Zheng, Guanglei Zhou, Hai Li, *Fellow, IEEE*, and Yiran Chen, *Fellow, IEEE*

Digital Object Identifier 10.1109/MCAS.2024.3476008
Date of current version: 7 February 2025

Abstract

The rapid development of large language models (LLMs) has significantly transformed the field of artificial intelligence, demonstrating remarkable capabilities in natural language processing and moving towards multi-modal functionality. These models are increasingly integrated into diverse applications, impacting both research and industry. However, their development and deployment present substantial challenges, including the need for extensive computational resources, high energy consumption, and complex software optimizations. Unlike traditional deep learning systems, LLMs require unique optimization strategies for training and inference, focusing on system-level efficiency. This paper surveys hardware and software co-design approaches specifically tailored to address the unique characteristics and constraints of large language models. This survey analyzes the challenges and impacts of LLMs on hardware and algorithm research, exploring algorithm optimization, hardware design, and system-level innovations. It aims to provide a comprehensive understanding of the trade-offs and considerations in LLM-centric computing systems, guiding future advancements in AI. Finally, we summarize the existing efforts in this space and outline future directions toward realizing production-grade co-design methodologies for the next generation of large language models and AI systems.

Index Terms—Large language model, hardware-software co-design.

QML: Introduction

- **ML successes nowadays**
  - Hardware advancements (GPUs, TPUs)

A Survey: Collaborative Hardware and Software Design in the Era of Large Language Models

Cong Guo, Feng Cheng, Zhixu Du, James Kiessling, Jonathan Ku, Shiyu Li, Ziru Li, Mingyuan Ma, Tergel Molom-Ochir, Benjamin Morris, Haoxuan Shan, Jingwei Sun, Yitu Wang, Chiyue Wei, Xueying Wu, Yuhao Wu, Hao Frank Yang, Jingyang Zhang, Junyao Zhang, Qilin Zheng, Guanglei Zhou, Hai Li, *Fellow, IEEE*, and Yiran Chen, *Fellow, IEEE*

Digital Object Identifier 10.1109/MCAS.2024.3476008
Date of current version: 7 February 2025

Table 2.
LLM accelerators for inference.

Name	Platform	Model	Energy efficiency (TOPS/W)	Quantization/ Sparsity	Year
TranCIM [181]	ASIC tapeout 28nm	BERT	20.5 (INT8)	Sparsity	2022
DFX [182]	FPGA	GPT-2			2022
X-Former [183]	PIM simulator 32 nm	BERT	13.44 (INT8)	-	2022
DOTA [184]	ASIC 22nm/simulator	GPT-2	-	Quantization/ Sparsity	2022
SPRINT [185]	PIM simulator 32 nm	GPT-2/BERT	19.6x	Sparsity	2022
TransPIM [186]	PIM 65nm/simulator	GPT-2/BERT	666.6x RTX 2080Ti	-	2022
Mokey [160]	ASIC 65nm/simulator	BERT	9x GOBO (FP16)	Quantization	2022
LeOPArd [187]	ASIC 65nm/simulator	GPT-2/BERT	3x SpAtten	Quantization/ Sparsity	2022
STP [188]	ASIC tapeout 12 nm	BERT	18.1 (FP4)	Quantization	2023
HAIMA [189]	PIM simulator 45 nm	BERT	-		2023
TF-MVP [190]	ASIC 28nm	BERT/GPT-2	0.48 (FP16)	Sparsity	2023
TiC-SAT [191]	gem5-X	BERT	-	-	2023
Transformer-OPU [192]	FPGA	BERT	-	-	2023
FACT [193]	ASIC 28nm	BERT	94.88x V100	Quantization/ Sparsity	2023
TaskFusion [194]	ASIC 22nm/simulator	BERT	19.83x Jetson Nano	Sparsity	2023
OliVe [161]	ASIC 22nm/simulator	BERT/GPT-2/OPT	4 x GOBO	Quantization	2023
C-Transformer [195]	ASIC tapeout 28 nm	GPT-2	33.4 (INT8)	-	2024
SpecPIM [196]	PIM simulator	LLaMA/OPT	6.67x A100 (FP16)	-	2024
ASADI [197]	PIM simulator	BERT/GPT-2	- (FP32)	Sparsity	2024
AttAcc [198]	PIM simulator	LLaMA/GPT-3	2.67x DGX A100 (FP16)	-	2024
NeuPIMs [199]	PIM simulator 22nm	GPT-3	-	-	2024

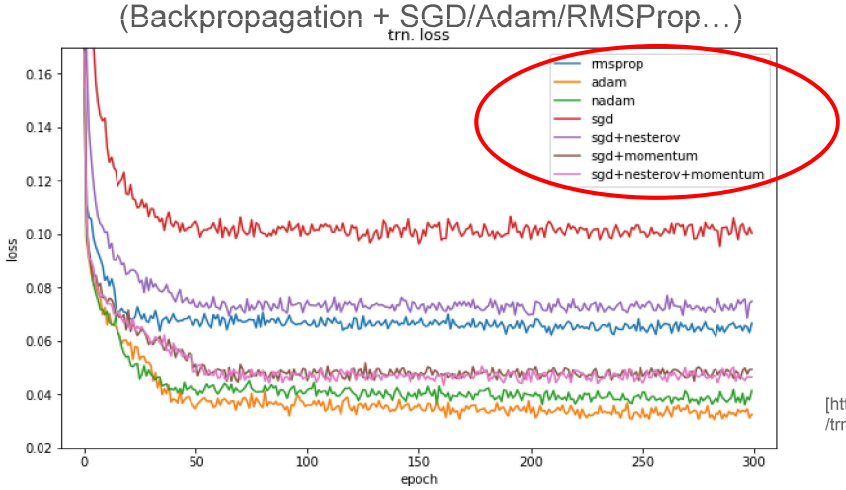
FIRST QUARTER 2025 1531-636X/25©2025IEEE YSTEMS MAG.

QML: Introduction

❑ ML successes nowadays

❑ Workhorse algorithms: Gradient calculation and Optimization

(Backpropagation + SGD/Adam/RMSProp…)

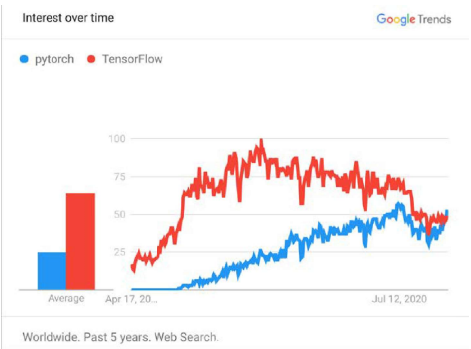


[https://shaoanlu.files.wordpress.com/2017/05/trn_loss.png]

QML: Introduction

❑ ML successes nowadays

❑ Specialized, user-friendly software

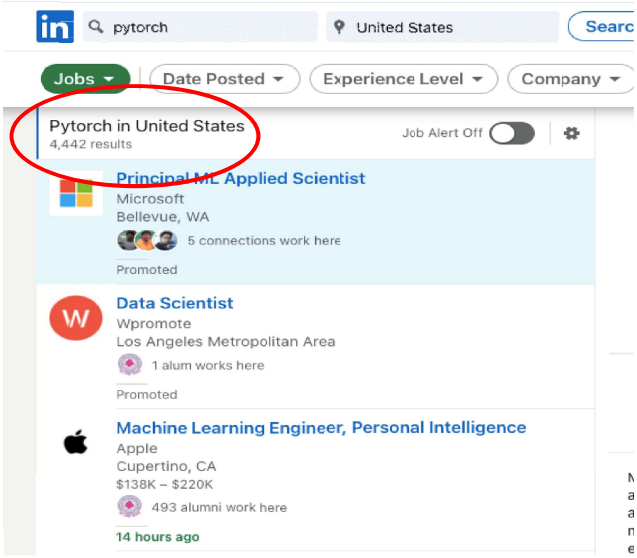


PyTorch

TensorFlow

QML: Introduction

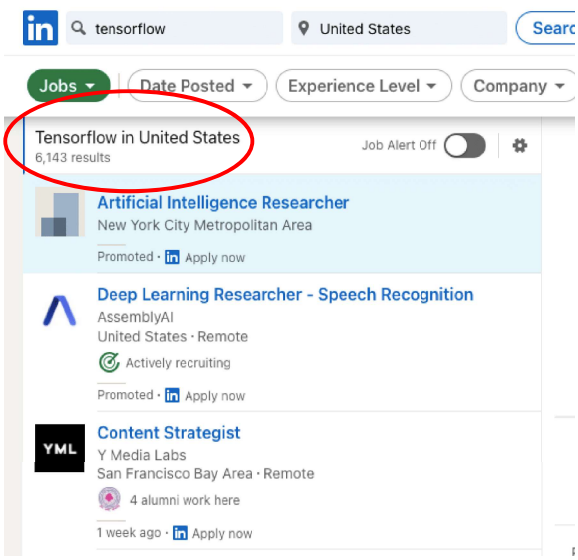
❑ ML successes nowadays



PyTorch vs TensorFlow 2023-A Head-to-Head Comparison

PyTorch vs TensorFlow 2023-A Head-to-Head Comparison of the similarities and differences of the top deep learning frameworks.

Last Updated: 24 Apr 2023



QML: Introduction

❑ ML successes nowadays

- ❑ Specialized, user-friendly software

PyTorch vs. TensorFlow: Full Overview 2025 Guide

Lazy Programmer
February 21, 2025

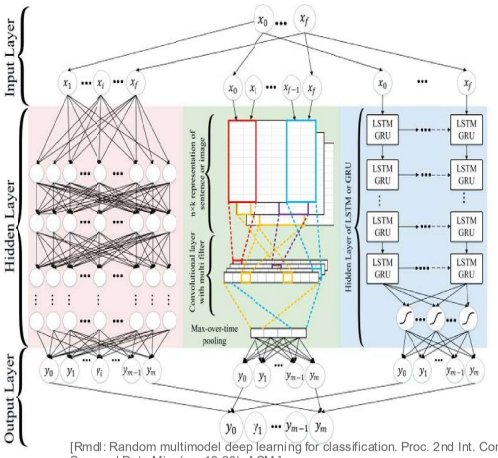
PyTorch

TensorFlow

Feature	PyTorch	TensorFlow
CNN Performance	Good	Excellent
RNN Performance	Excellent	Good
BERT Models	Superior	Good
Large-scale Training	Good	Excellent
GPU Utilization	Excellent	Excellent
TPU Support	Limited	Native

QML: Introduction

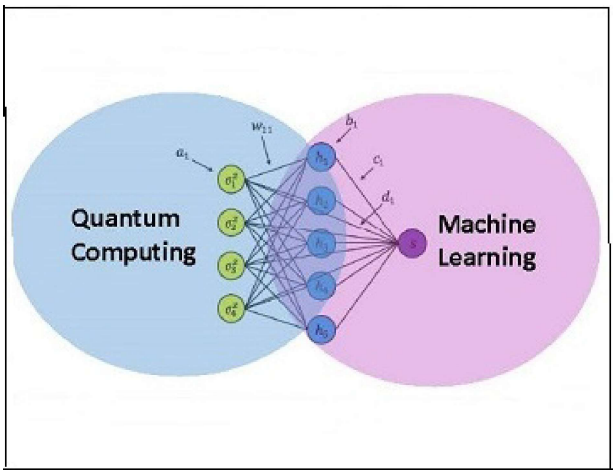
- **Other insights from ML**
  - Why is ML successful?
    - Rapid iteration
    - Compose and reuse component
    - Crowdsourced innovation
    - Shared code (including trained weights)
    - New ways of thinking



[Rmdl: Random multimodel deep learning for classification. Proc. 2nd Int. Conf. Info. Sys. and Data Min. (pp. 19-28). ACM.]

Schedule

- **QML: General Concepts**
  - **From ML to QML**
    - **Example of ML (Financial)**
    - **Reasons for success!**
    - **QC and ML?**

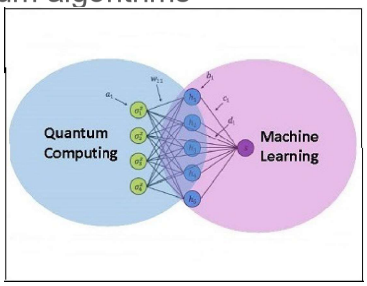


Source: https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

QML: Introduction

❑ Quantum Machine Learning

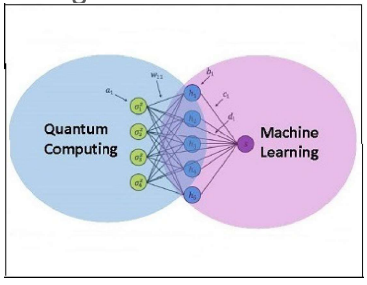
- ❑ Research area that explores the interplay of ideas from QC & ML
  - ❑ We might want to find out whether quantum computers can speed up the time it takes to train or evaluate a ML model. This could lead to exponential or polynomial speedups in certain supervised or generative tasks.
  - ❑ On the other hand, we can leverage techniques from ML to help us discover quantum error-correcting codes, estimate the properties of quantum systems, or develop new quantum algorithms



QML: Introduction

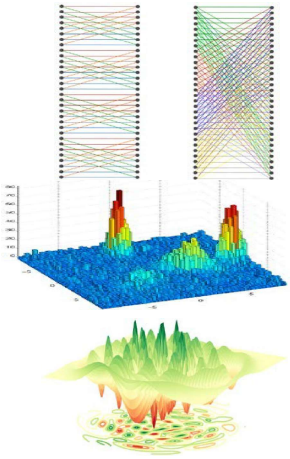
❑ Quantum Machine Learning

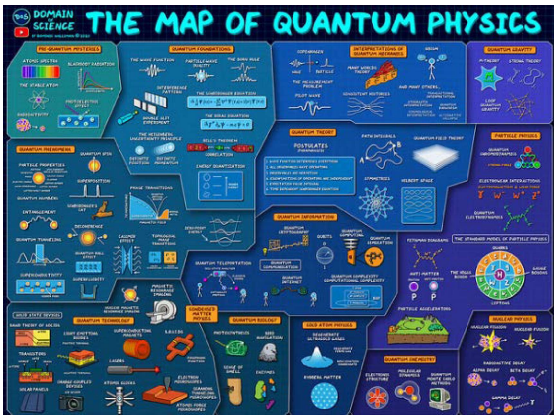
- ❑ Research area that explores the interplay of ideas from QC & ML
  - ❑ We might want to find out whether quantum computers can speed up the time it takes to train or evaluate a ML model. This could lead to exponential or polynomial speedups in certain supervised or generative tasks.
  - ❑ On the other hand, we can leverage techniques from ML to help us discover quantum error-correcting codes, estimate the properties of quantum systems, or develop new quantum algorithms



QML: Introduction

- **Quantum Computers perform well at:**
  - Quantum Physics
  - Linear Algebra
  - Sampling
  - Optimization

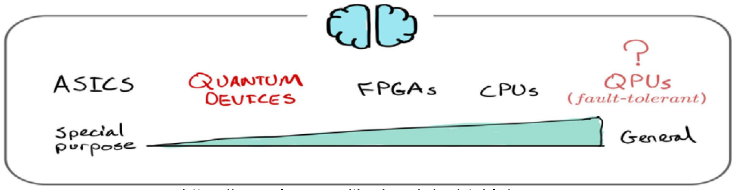




https://store.dftba.com/collections/domain-of-science. Dominic Walliman. 2020

QML: Introduction

- **Quantum Machine Learning**
  - Machine learning on near-term quantum devices
    - Some research focuses on ideal, universal quantum computers ("fault-tolerant QPUs") which are still years away. But there is rapidly-growing interest in quantum machine learning on near-term quantum devices
    - We can understand these devices as special-purpose hardware like ASICs and FPGAs, which are more limited in their functionality → May train intractable models until now



https://pennylane.readthedocs.io/en/stable/

QML: Introduction

- **Quantum Machine Learning**
  - We can adapt machine learning tools to help understand and build quantum computers
  - Use automatic differentiation to tune circuits

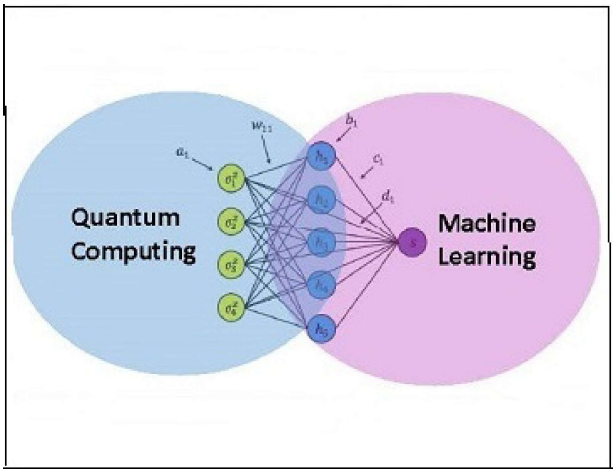
  - Discover new algorithms and error-correction strategies
  - Unearth new physics?

QML: Introduction

- **QML nowadays!**
  - Everyone can easily explore and run QML algorithms
  - Models are widely shared (or reimplemented by others)
  - Reusable circuit blocks, embeddings, pre-/post-processing
  - Exciting QML results every single week
  - Feedback loop →new ideas we can't predict today

Schedule

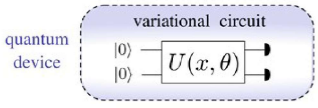
- **QML: General Concepts**
  - From ML to QML
  - Training Quantum Circuits
    - Concepts:
      - Variational Circuits
      - Training Quantum Circuits
      - Quantum Nodes
      - Hybrid Computation

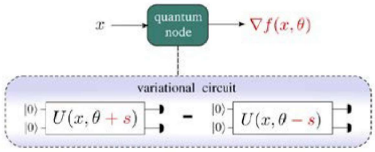


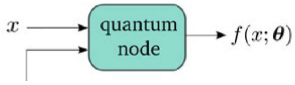
Source: https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

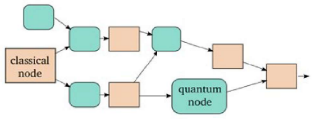
QML: Training Quantum Circuits

- **Concepts:**
  - Variational circuits
  - Training quantum circuits
  - Quantum nodes
  - Hybrid computation



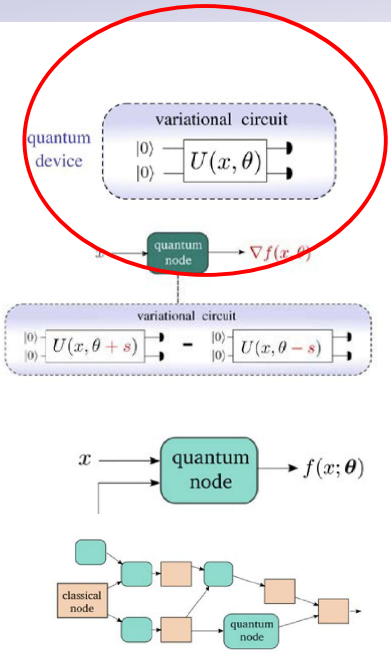






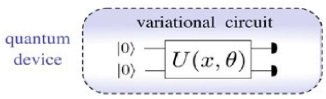
QML: Training Quantum Circuits

- **Concepts:**
  - **Variational circuits**
  - Training quantum circuits
  - Quantum nodes
  - Hybrid computation



QML: Training Quantum Circuits

- **Variational circuits**

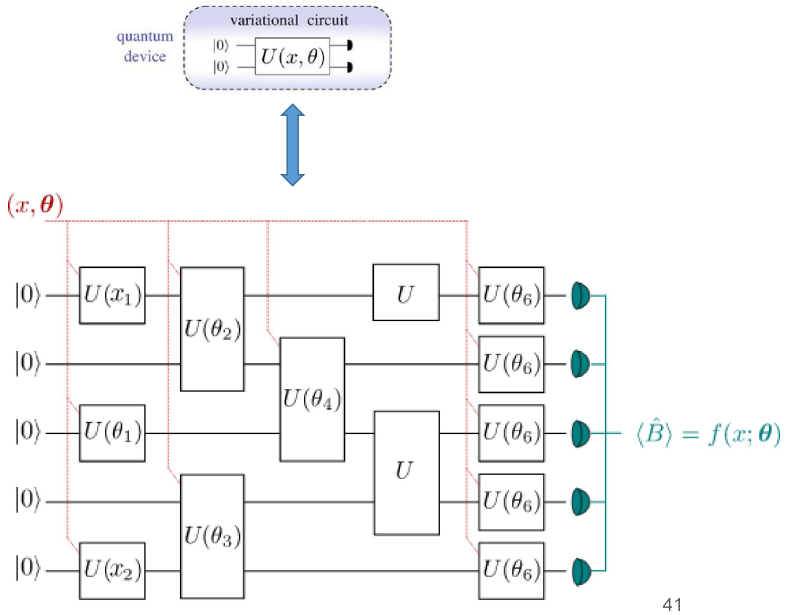


- Parametrized quantum circuits (algorithms) that depend on free parameters. Like standard quantum circuits, they consist of three ingredients:
  - Preparation of a fixed initial state (eg, the vacuum state or the zero state)
  - A quantum circuit *U(θ)*, parameterized by a set of free parameters *θ*
  - Measurement of an observable B at the output. This observable may be made up from local observables for each wire in the circuit, or just a subset of wires

QML: Training Quantum Circuits

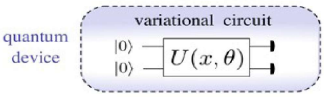
❑ Variational circuits

- ❑ Preparation of a fixed initial state
- ❑ Quantum circuit; input data and free parameters are used as gate arguments
- ❑ Measurement of fixed observable

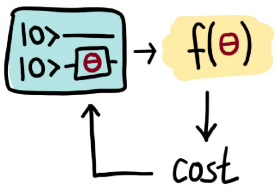


QML: Training Quantum Circuits

❑ Variational circuits

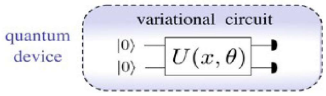


- ❑ Typically, the expectation values $f(\theta)=\langle 0|U^{\dagger}(\theta)BU(\theta)|0\rangle$ of one or more such circuits — possibly with some classical post-processing — define a scalar cost for a given task
- ❑ The free parameters $\theta=(\theta_1,\theta_2,...)$ of the circuit(s) are tuned to optimize this cost function



QML: Training Quantum Circuits

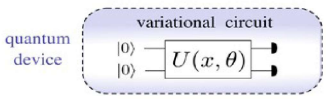
- **Variational circuits** →



- Trained by a classical optimization algorithm that queries to the QC
- Optimization is iterative and searches out better candidates for the parameters with every step
- Quantum Circuits for near-term quantum devices. Such devices can only run short gate sequences, since without fault tolerance every gate increases the error in the output
- Usually, a quantum algorithm is decomposed into a set of elementary operations, which are in turn implemented by the quantum hardware

QML: Training Quantum Circuits

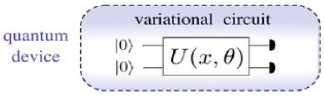
- **Variational circuits** →



- The intriguing idea of variational circuit for near-term devices is to merge this two-step procedure into a single step by “learning” the circuit on the noisy device for a given task
- This way, the “natural” tunable gates of a device can be used to formulate the algorithm, without the detour via a fixed elementary gate set. Furthermore, systematic errors can automatically be corrected during optimization

QML: Training Quantum Circuits

- **Variational circuits**



  - Main QML method for near-term (NISQ) devices
  - Ideas earlier began in simpler specialized forms:
    - Variational Quantum Eigensolver (VQE)
    - Quantum Alternated Operator Ansatz (QAOA)
  - Natural extension to other circuits and tasks (e.g., quantum classifier)
  - Nowadays many, many proposals

QML: Training Quantum Circuits

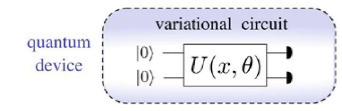
- **VQE** is used primarily to find the ground state energy of a system, which involves determining the lowest eigenvalue of its Hamiltonian
- **Hamiltonian:**
  - Consider a simple Hamiltonian for a two-qubit system: $H=\sigma z^{(1)}\otimes\sigma z^{(2)}$
  - This Hamiltonian measures the Z-spin correlation between two qubits
- **Initial State:**
  - Start with both qubits in the ground state $|00\rangle$
- **Parameterized Circuit:**
  - Apply $Ry(\theta)$ gates to both qubits to create superpositions: $|\psi(\theta)\rangle=(Ry(\theta_1)\otimes Ry(\theta_2))|00\rangle$
  - Apply a CNOT gate to entangle the qubits: $|\psi\text{entangled}(\theta)\rangle=\text{CNOT}|\psi(\theta)\rangle$
- **Objective Function:**
  - The objective is to minimize the expectation value of the Hamiltonian: $\langle H\rangle=\langle\psi\text{entangled}(\theta)|H|\psi\text{entangled}(\theta)\rangle$
- **Calculation:**
  - For $H=\sigma z^{(1)}\otimes\sigma z^{(2)}$, the expectation value depends on the probabilities of the qubits being in the same or opposite states: $\langle H\rangle=P_{00}+P_{11}-P_{01}-P_{10}$ where Pij is the probability of measuring the state $|ij\rangle$
- **Optimization:**
  - Adjust θ_1 and θ_2 to minimize $\langle H\rangle$

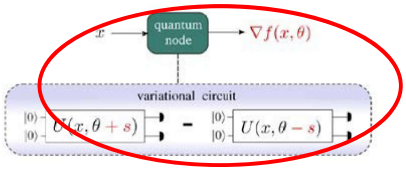
QML: Training Quantum Circuits

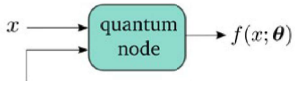
- **QAOA** is designed for solving optimization problems, typically represented by a cost function encoded in a Hamiltonian
- **Problem:**
  - Consider a simple Max-Cut problem for a graph with two vertices connected by one edge
- **Hamiltonian:**
  - The Hamiltonian for this problem can be represented as: $H_C=1/2(I-\sigma z^{(1)}\otimes\sigma z^{(2)})$ where each term in H_C contributes positively when the qubits (vertices) are in different states (representing a cut in the graph)
- **Initial State:**
  - Start with a superposition of all possible states: $|\psi 0\rangle=H^{\otimes 2}|00\rangle$
- **QAOA Circuit:**
  - **Mixing Unitary *U*(*B*,*β*)**: Apply *Rx*(*β*) gates to both qubits, where *β* is the mixing angle
- **Problem Unitary *U*(*C*,*γ*)**: Apply a phase based on the cost Hamiltonian H_C, which involves controlled-Z operations conditioned on the graph's edges and a phase shift by *γ*
- **Calculation:**
  - The goal is to maximize the expectation value of H_C : $\langle H_C\rangle=\langle\psi(\beta,\gamma)|H_C|\psi(\beta,\gamma)\rangle$
- **Optimization:** Adjust β and γ to maximize $\langle H_C\rangle$

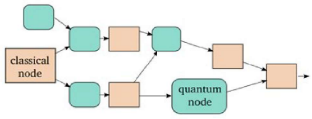
QML: Training Quantum Circuits

- **Concepts:**
  - Variational circuits
  - **Training quantum circuits**
  - Quantum nodes
  - Hybrid computation



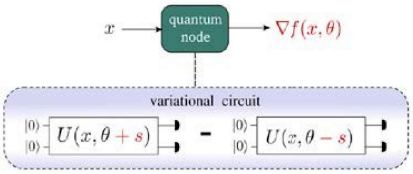






QML: Training Quantum Circuits

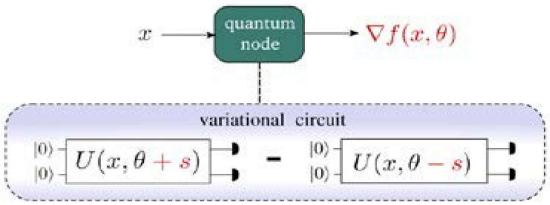
- **Differentiable Computation**



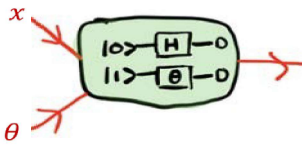
- Deep learning began with NNs, but nowadays the models are much richer
- Attention mechanisms, external memory, neural differential equations, etc.
- Key insight: computation is end-to-end differentiable
- Program only the structure of computation ("build the model")
- Use optimization (e.g., gradient descent) to fine-tune parameters ("train the model")

QML: Training Quantum Circuits

- **Differentiable Computation**



- Quantum computing is also differentiable
- Gates are controlled by parameters (eg, rotation angle, squeezing amplitude)
- Expectation values depend smoothly on gate parameters
- Can we train quantum circuits?

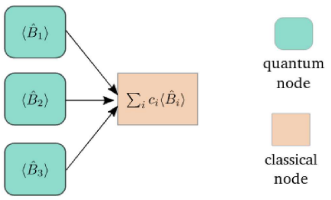


$$f(x,\theta) = \langle \hat{B} \rangle = \langle U^\dagger(\theta) \hat{B} U(\theta) \rangle$$

QML: Training Quantum Circuits

- **How to train Quantum Circuits?** Two ways
  - Simulator- based:
    - Build simulation inside existing classical library
    - Can leverage existing optimization & ML tools
    - Great for small circuits, but not scalable

  - Hardware - based:
    - No access to quantum information;
      - Only have measurements & expectation values
    - Needs to work as hw becomes more powerful and cannot be simulated



QML: Training Quantum Circuits

- **Gradient of Quantum circuit** ∇f
  - Training strategy: use gradient descent algorithms
  - Need to compute gradients of variational circuit outputs with respect to their gate parameters
  - How can we compute gradients of quantum circuits when even simulating their output is classically intractable?

Mitarai, K., Negoro, M., Kitagawa, M., & Fujii, K. (2018). Quantum circuit learning. *Physical Review A*, *98*(3), 032309.

Schuld, M., Bergholm, V., Gogolin, C., Izaac, J., & Killoran, N. (2019). Evaluating analytic gradients on quantum hardware. *Physical Review A*, *99*(3), 032331.

QML: Training Quantum Circuits

- **The parameter shift method**
- **Idea**

$$\nabla f$$

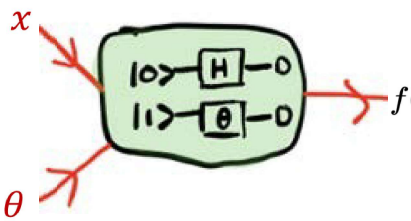
$$f(\theta) = \sin\theta \Rightarrow \partial_\theta f(\theta) = \cos\theta$$

$$\partial_\theta f = \frac{1}{\sqrt{2}}[f(\theta + \pi/4) - f(\theta - \pi/4)]$$

$$\cos\theta = \frac{\sin(\theta+\pi/4) - \sin(\theta-\pi/4)}{\sqrt{2}}$$

QML: Training Quantum Circuits

- **The parameter shift method**
  - Main insight: Use the same quantum hardware to evaluate its own gradients!
  - The gradient of a circuit can be computed by the same circuit, with shifted parameters
  - The parameters *c* and *s* depend on the specific function. Crucially, *s* is large



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

QML: Training Quantum Circuits

- **It is not finite difference method**

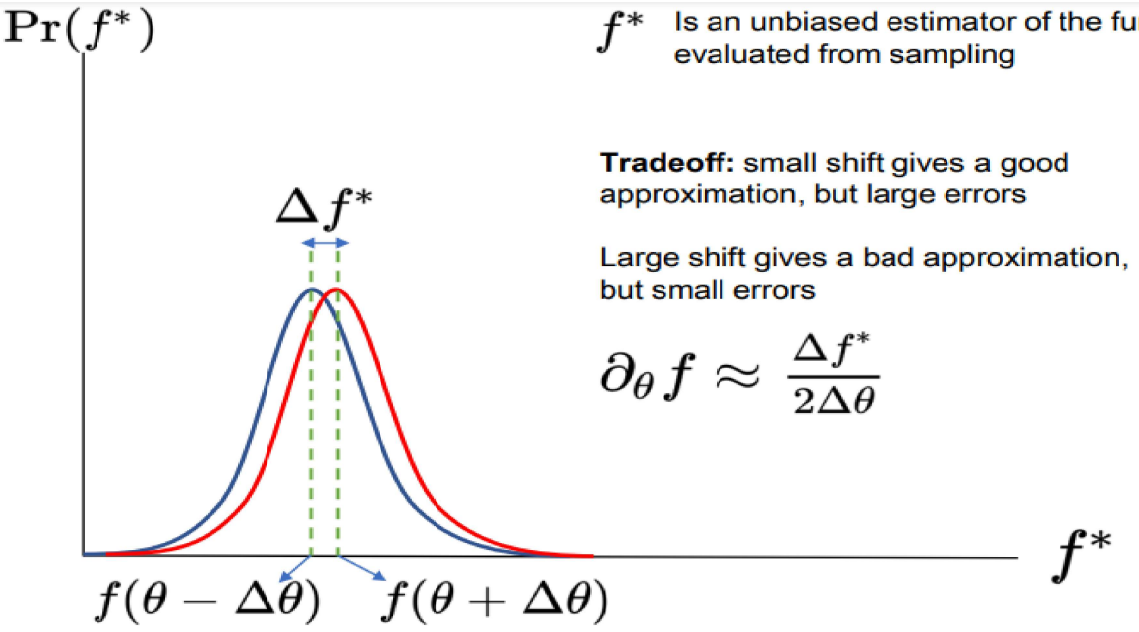
$$\partial_\theta f(\theta) = c[f(\theta + s) - f(\theta - s)]$$

- Exact
- Shift is specific to each gate – in general, we use a large shift
- Using the structure of quantum circuits, can derive analytic recipes

$$\partial_\theta f(\theta) = \frac{f(\theta+\Delta\theta) - f(\theta-\Delta\theta)}{2\Delta\theta}$$

- Only an approximation
- Requires that shift is small
- Known to give rise to numerical issues
- For NISQ devices, small shifts could lead to the resulting difference being swamped by noise

QML: Training Quantum Circuits



f^* Is an unbiased estimator of the function, evaluated from sampling

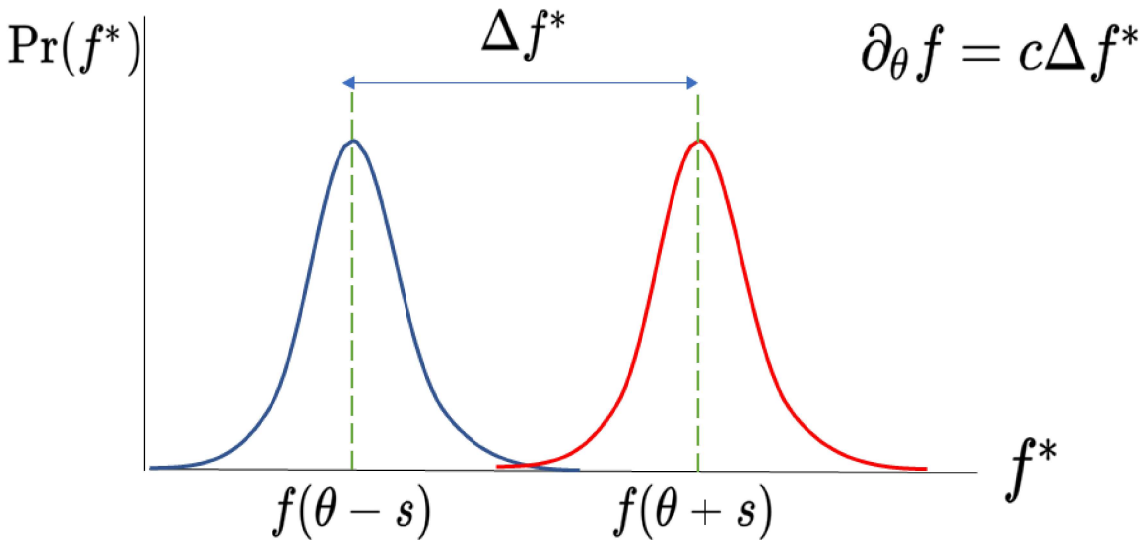
Tradeoff: small shift gives a good approximation, but large errors

Large shift gives a bad approximation, but small errors

$$\partial_\theta f \approx \frac{\Delta f^*}{2\Delta\theta}$$

QML: Training Quantum Circuits

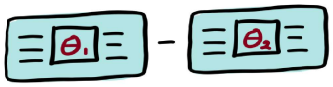
- Using the structure of quantum circuits, can derive analytic recipes



QML: Training Quantum Circuits

- **Exact and analytic recipes**

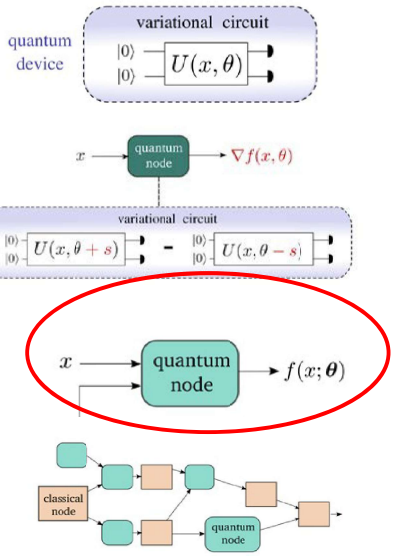
$\nabla_\theta f = f(\theta_1) - f(\theta_2)$



Gate $\mathcal{G}$	Heisenberg representation $M^{\mathcal{G}}$	Partial derivatives of $M^{\mathcal{G}}$
Phase rotation $R(\phi)$	$M^R(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$	$\partial_\phi M^R(\phi) = \frac{1}{2}\left(M^R(\phi + \frac{\pi}{2}) - M^R(\phi - \frac{\pi}{2})\right)$
Displacement $D(r,\phi)$	$M^D(r,\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 2r\cos\phi & 1 & 0 \\ 2r\sin\phi & 0 & 1 \end{pmatrix}$	$\partial_r M^D(r,\phi) = \frac{1}{2s}\left(M^D(r+s,\phi) - M^D(r-s,\phi)\right),\ s \in \mathbb{R}$ $\partial_\phi M^D(r,\phi) = \frac{1}{2}\left(M^D(r,\phi+\frac{\pi}{2}) - M^D(r,\phi-\frac{\pi}{2})\right)$
Squeezing[a] $S(r)$	$M^S(r) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \mathrm{e}^{-r} & 0 \\ 0 & 0 & \mathrm{e}^{r} \end{pmatrix}$	$\partial_r M^S(r) = \frac{1}{2\sinh(s)}\left(M^S(r+s) - M^S(r-s)\right),\ s \in \mathbb{R}$
Beamsplitter $B(\theta,\phi)$	$M^B(\theta,\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & -\alpha & -\beta \\ 0 & 0 & \cos\theta & \beta & -\alpha \\ 0 & \alpha & -\beta & \cos\theta & 0 \\ 0 & \beta & \alpha & 0 & \cos\theta \end{pmatrix}$	$\partial_\theta M^B(\theta,\phi) = \frac{1}{2}\left(M^B(\theta+\frac{\pi}{2},\phi) - M^B(\theta-\frac{\pi}{2},\phi)\right)$ $\partial_\phi M^B(\theta,\phi) = \frac{1}{2}\left(M^B(\theta,\phi+\frac{\pi}{2}) - M^B(\theta,\phi-\frac{\pi}{2})\right)$ $\alpha = \cos\phi\sin\theta,\quad \beta = \sin\phi\sin\theta$

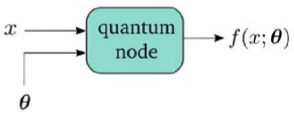
QML: Training Quantum Circuits

- **Concepts:**
  - Variational circuits
  - Training quantum circuits
  - **Quantum nodes**
  - Hybrid computation

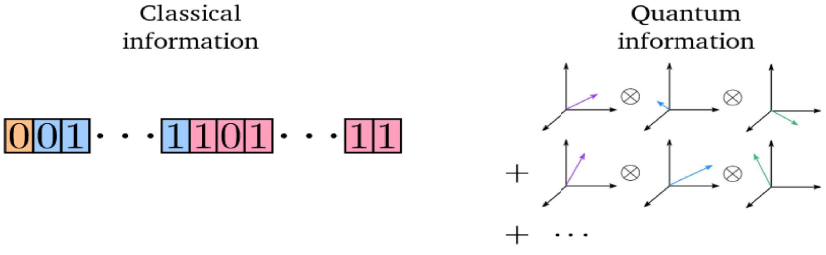


QML: Training Quantum Circuits

- **Quantum Nodes**

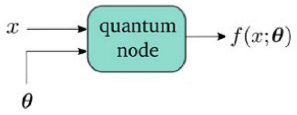


  - Classical and quantum information are distinct
  - A classical processor can't access quantum information inside a circuit

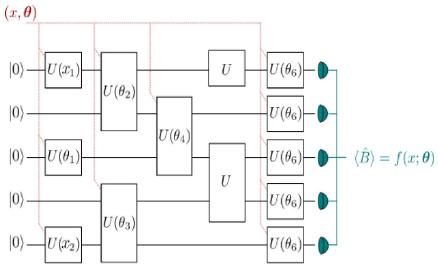


QML: Training Quantum Circuits

- **Quantum Nodes**

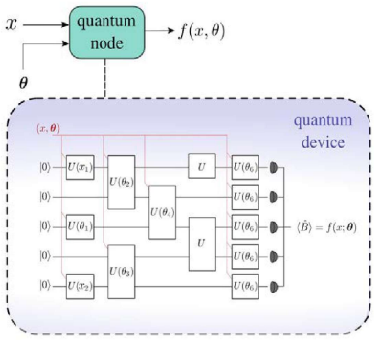


  - However, a variational circuit:
    - takes classical information as input (gate parameters)
    - produces classical information as output (expectation values)
  - Transforms classical data to:
    - classical data
    - function itself may be classically intractable



QML: Training Quantum Circuits

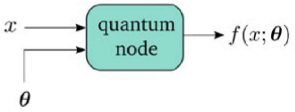
- **QNode: common interface for quantum and classical devices**



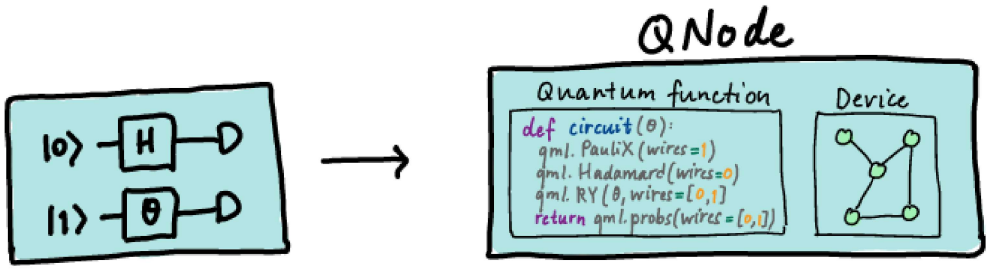
  - Classical device sees a callable parameterized function
  - Quantum device sees fine-grained circuit details
  - A quantum node consists of a quantum function (such as a variational circuit), and a device on which it executes.

QML: Training Quantum Circuits

- **Quantum Nodes**



  - A quantum node consists of a quantum function (such as a variational circuit), and a device on which it executes.

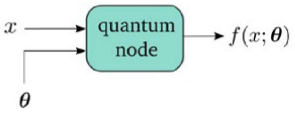


SOURCE
https://pennylane.ai/qml/glossary/quantum_node.html

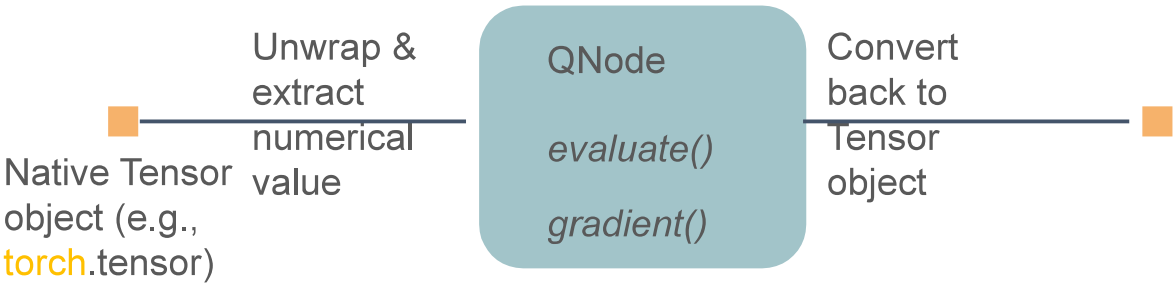
QML: Training Quantum Circuits

- **QNode**

enables interface between quantum computers and classical ML libraries

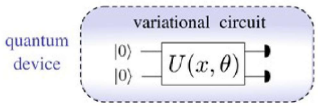


  - NumPy Autograd, Tensor Flow, Pytorch

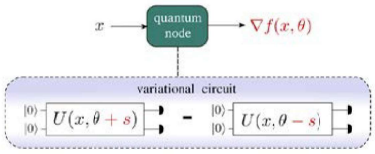


QML: Training Quantum Circuits

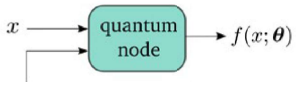
- **Concepts**
  - Variational circuits



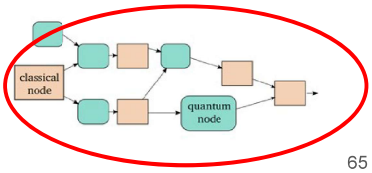
  - Training quantum circuits



  - Quantum nodes

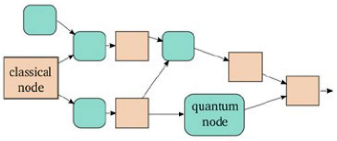


  - **Hybrid computation**

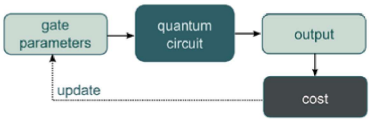


QML: Training Quantum Circuits

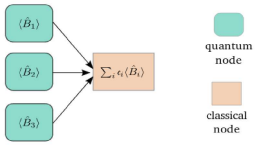
- **Quantum circuit can be just one step of a larger computation**



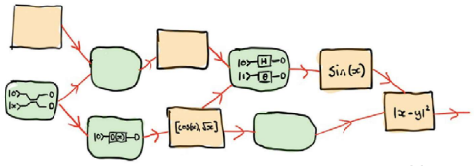
  - Classical optimization loop



  - Pre-/post-process quantum circuit outputs

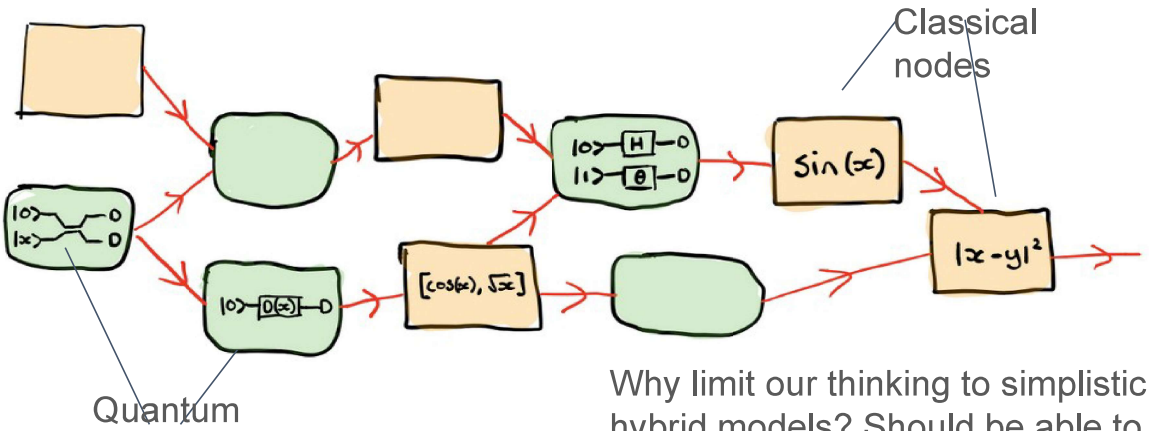


  - Arbitrarily structured hybrid computations



QML: Training Quantum Circuits

- Quantum and classical nodes can be combined into an arbitrary directed acyclic graph (DAG)
- No loops, information flows from each node to its successors, and no cycles (loops) are created

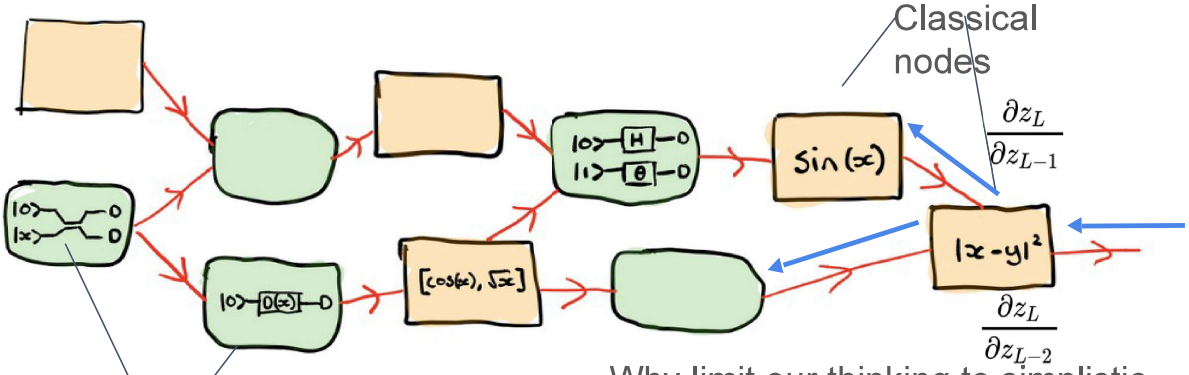


SOURCE
https://pennylane.ai/qml/glossary/hybrid_computation.html

Why limit our thinking to simplistic hybrid models? Should be able to compose arbitrarily

QML: Training Quantum Circuits

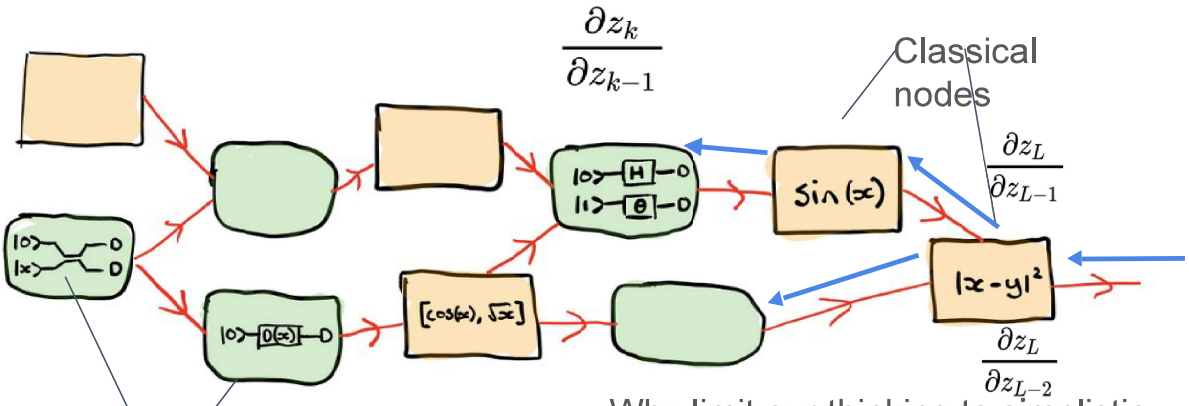
- Backpropagation steps backwards through computational graph
- Computes gradient at each step and aggregates (chain rule)



Why limit our thinking to simplistic hybrid models? Should be able to compose arbitrarily

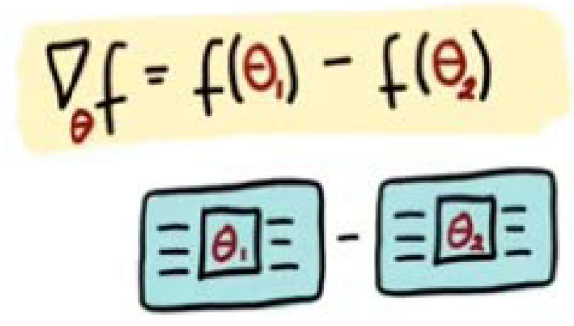
QML: Training Quantum Circuits

- When we hit a quantum node, use parameter shift method
- Can't use backpropagation inside a QNode, but can backpropagate through it
- End-to-end differentiable

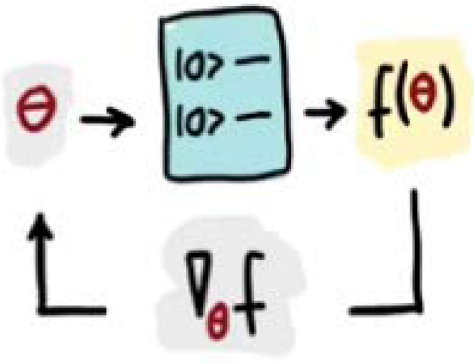


Why limit our thinking to simplistic hybrid models? Should be able to compose arbitrarily

QML: Training Quantum Circuits



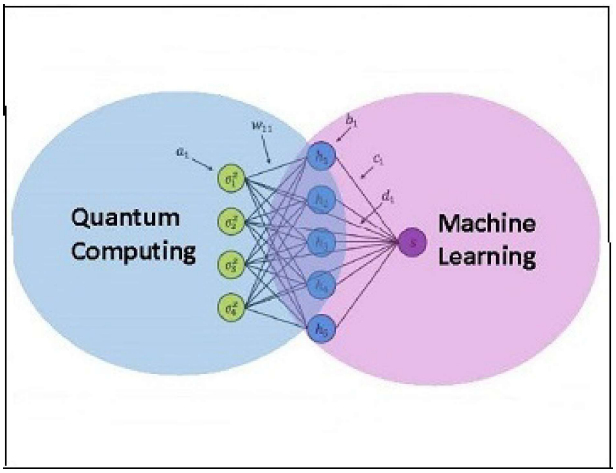
Parameter-shift rule



Train via gradient descent

Schedule

- ❑ **QML: General Concepts**
  - ❑ From ML to QML
  - ❑ Training Quantum Circuits
  - ❑ Tools → Pennylane use case
    - ❑Tools we will use



Source:
https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

QML: Tools

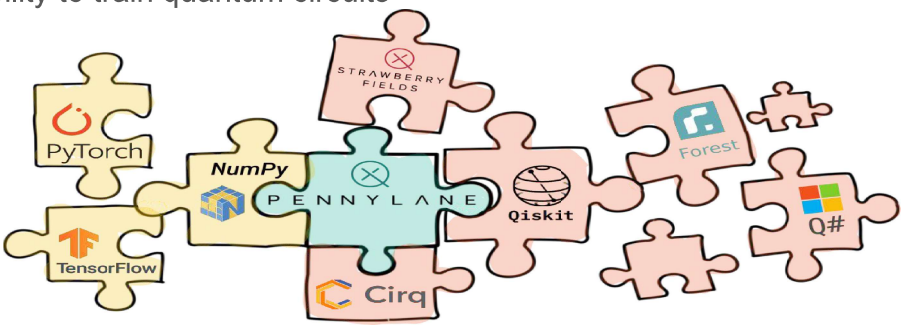
- ❑ **Quantum Machine Learning (Finance)**
  - ❑ Tools we will use to the examples
  - ❑ Colab notebook
    - ❑ Colaboratory (Google "colab") for machine learning education and research
    - ❑ Jupyter notebook environment that requires no setup to use and runs entirely in the cloud

QML: Tools

❑ Quantum Machine Learning (Finance)

❑ Pennylane

- ❑ Open-source software framework (Apache License) built around the concept of quantum differentiable programming
- ❑ It integrates classical ML libraries with quantum simulators and hardware, giving the capability to train quantum circuits



QML: Tools

❑ Quantum Machine Learning (Finance)

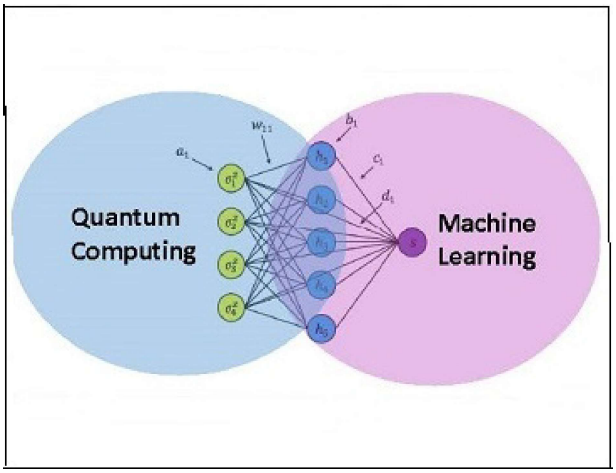
❑ Different available frameworks:

Quantum computation	Classical computation: CPU-only	Classical computation: GPU-ready
Simulator	Cirq Q#	Strawberry Fields QMLT QCGPU QuantumFlow
QPU-ready	Forest/pyQuil Qiskit ProjectQ XACC	PennyLane

Existing Libraries and their integration

Schedule

- **QML: General Concepts**
  - From ML to QML
  - Training Quantum Circuits
  - Tools → Pennylane use case
  - Brief summary of QML

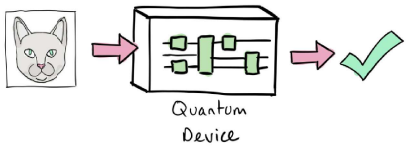


Source:
https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

QML: Brief summary of QML

- **Resuming: Quantum Machine Learning**
  - Using QC as Neural Networks

    - In the modern viewpoint, QC can be used and trained like neural networks. We can systematically adapt the physical control parameters, such as an electromagnetic field strength or a laser pulse frequency, to solve a problem
    - For example, a trained circuit can be used to classify the content of images, by encoding the image into the physical state of the device and taking measurements



QML: **Brief summary of QML**

❑ Resuming: Quantum Machine Learning

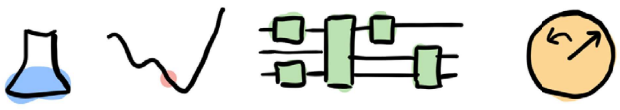
❑ The bigger picture: differentiable programming

- ❑ But the story is bigger than just using QC to tackle ML problems. Quantum circuits are differentiable, and a QC itself can compute the change in control parameters needed to become better at a given task
- ❑ Differentiable programming is the very basis of deep learning, implemented in software libraries such as TensorFlow and PyTorch
- ❑ Differentiable programming is more than deep learning: it is a programming paradigm where the algorithms are not hand-coded, but learned

QML: **Brief summary of QML**

❑ Resuming: Quantum Machine Learning

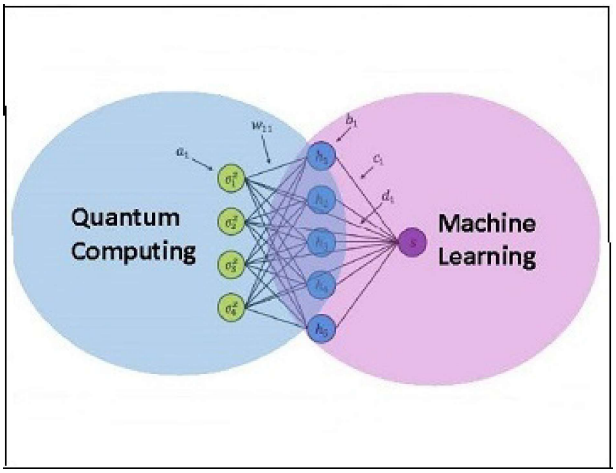
❑ The bigger picture: differentiable programming



- ❑ Similarly, the idea of training QC is larger than QML. Trainable quantum circuits can be leveraged in other fields like quantum chemistry or quantum optimization
- ❑ It can help in a variety of applications such as the design of quantum algorithms, the discovery of quantum error correction schemes, and the understanding of physical systems

QML: Classic Classifier

- QML: General Concepts
- **Example: Classic Classifier**
- Constructing Models
- Example: Quantum Classifier



Source:
https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

QML: Classic Classifier

- **Quantum Machine Learning (Finance)**
  - **First example: classic classifier**

```
import pennylane as qml
from pennylane import numpy as np
import matplotlib.pyplot as plt
```

- The pennylane library is imported, focused on QML. Although we are going to do the classic classifier first, this same library works for us
- Numpy is imported from pennylane instead of the traditional library because the one from pennylane is numpy extended to work with ML
- In classical models, it uses Autograd, while in quantum models, PennyLane internally applies the parameter-shift rule when applicable

QML: Classic Classifier

❑ Quantum Machine Learning (Finance)

❑ First example: classic classifier

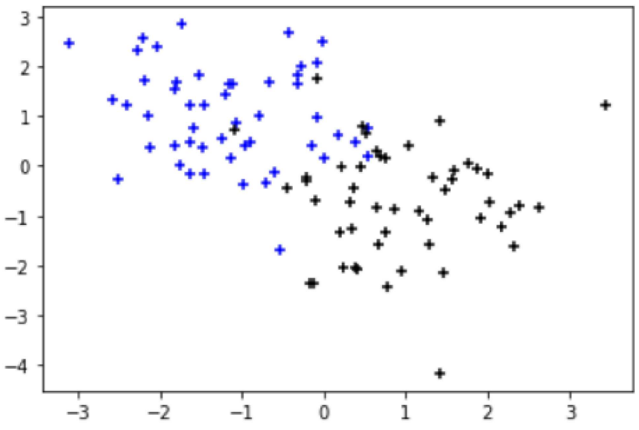
```
## generate the data
items = 100
class1 = np.array([[np.random.normal(loc=-1), np.random.normal(loc=1)] for index in range(items//2)])
class2 = np.array([[np.random.normal(loc=1), np.random.normal(loc=-1)] for index in range(items//2)])

plt.scatter(class1[:,0], class1[:,1], c='b', marker='+') ## class1 in blue
plt.scatter(class2[:,0], class2[:,1], c='k', marker='+') ## class2 in black
plt.show()
```

- ❑ A sample of 100 elements will be created that will correspond to two different classes and that will have to be classified
- ❑ To do this, 50 of these elements follow a normal distribution between -1 and 1. While the other 50 elements follow a normal distribution between 1 and -1
- ❑ plt.scatter allows us to draw the result of the generated elements

QML: Classic Classifier

❑ Quantum Machine Learning (Finance)

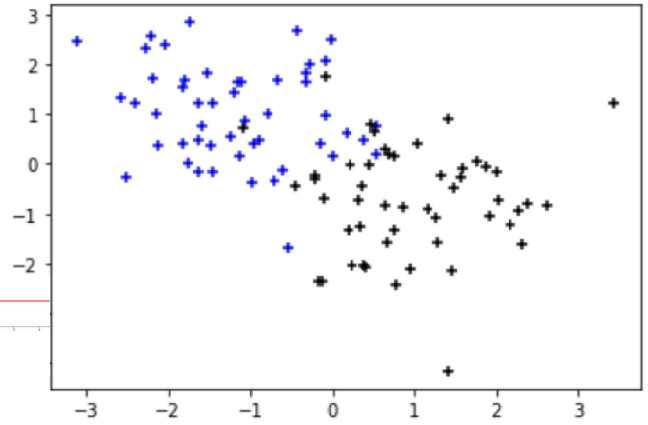
❑ First example: classic classifier



- ❑ The result of the generated elements
- ❑ Once the classes are created, we join the characteristics (feature) and the value (-1 for the blue points, and 1 for the black points)
- ❑ By joining them, the data set that we are going to classify is formed

QML: Classic Classifier

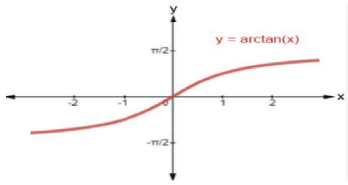
- Quantum Machine Learning (Finance)
  - First example: classic classifier



```
def model(x, w):
    return np.sign(x[0]*w[0] + x[1]*w[1] + w[2])
```

Differentiable!

```
def model(x, w):
    return np.arctan(x[0]*w[0] + x[1]*w[1] + w[2]) / (np.pi / 2)
```



QML: Classic Classifier

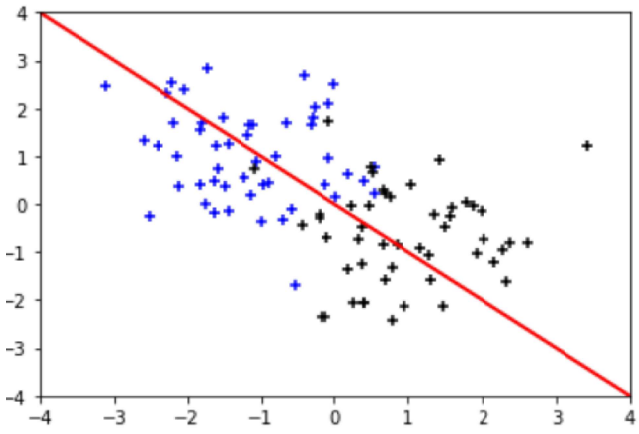
- Quantum Machine Learning (Finance)
  - First example: classic classifier

```
w = np.array([1, 1, 0], requires_grad = True)

def draw_line(x, w):
    if w[1] == 0:
        w[1] = 0.0001
    return -(w[0] * x  + w[2]) / w[1]

def plot_model(w):
    x1, x2 = -4, 4
    plt.plot([x1, x2], [draw_line(x1,w),draw_line(x2,w)], 'r', linewidth=2)
    plt.scatter(class1[:,0], class1[:,1], c='b', marker='+')
    plt.scatter(class2[:,0], class2[:,1], c='k', marker='+')
    plt.axis((-4,4,-4,4))
    plt.show()

plot_model(w)
```

- An array with weights is created and the functions are created to draw the model and the separation line between the classes

QML: Classic Classifier

- **Quantum Machine Learning (Finance)**
  - **First example: classic classifier**
    - At this point, the result shows that the data is not classified
    - Plot_model draws a line between points -4 and 4 (with plot) and draws the points of each class (with scatter)



QML: Classic Classifier

- **Quantum Machine Learning (Finance)**
  - **First example: classic classifier**

```
def average_error(weight, data):
    error = 0
    for x,y in data:
        distance = model(x,weight) - y
        error += (distance) ** 2
    return error/len(data)
```

    - The method for calculating the error is created
    - To do this, all the data of the classes are traversed. Distance between the output that the model gives and the correct result that the model should return is measured
    - If the model classifies well, the distance is 0, otherwise the distance is 2, a value to which the power operator is applied and the error rate increases
    - Finally, the mean of the calculated error is returned

QML: Classic Classifier

- **Quantum Machine Learning (Finance)**
  - **First example: classic classifier**

```
weight = [1,1,0]
print("error", average_error(weight, data))
```

```
error 1.2631248528976216
```

```
gradient = qml.grad(average_error, argnum=0)
```

- What is the mean error with the indicated line and the initial weights? The value of error taken is 1.26
- Given the error, we must know how to modify the weights in order to reduce the error. For this we use a differentiable function
- For this we use qml.grad, which derives the mean error based on the argument 0, in this case, it derives based on weight. We want to shift weight parameter to reduce error

QML: Classic Classifier

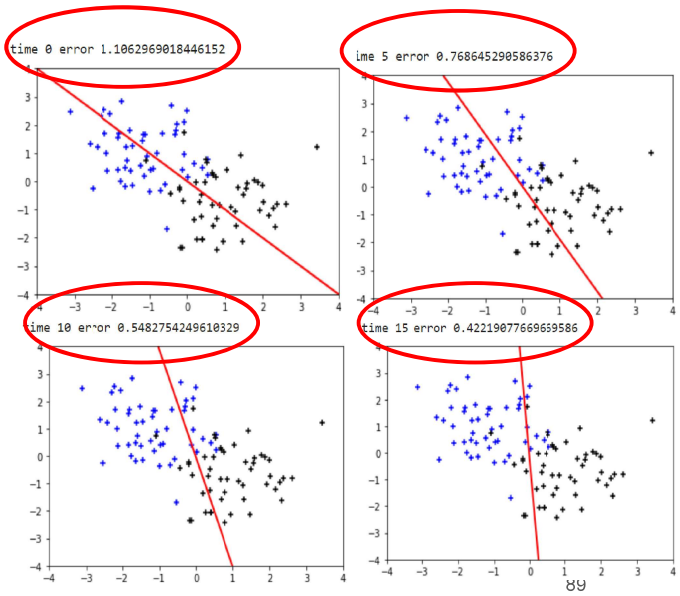
- **Quantum Machine Learning (Finance)**
  - **First example: classic classifier**

```
weight = np.array([1, 1, 0], requires_grad = True)

step = 0.1
for time in range(101):
    weight = weight - step*gradient(weight, data)
    if time % 5 == 0:
        print("time", time, "error", average_error(weight, data))
        plot_model(weight)
```

- We start from the initial value of weight [1,1,0] and we are going to train the model for 101 iterations to help us reduce the error
- In each iteration we are going to apply a step of size 0.1 to adjust the error. In each iteration, the weights are modified according to the value indicated by the gradient

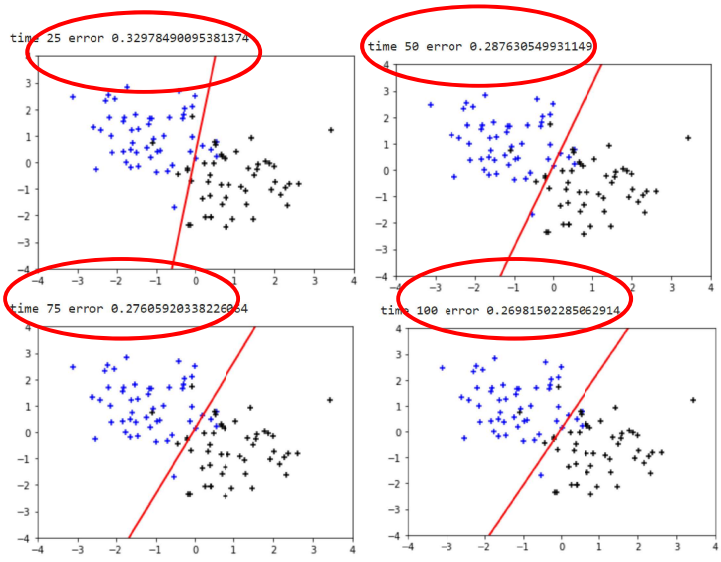
QML: Classic Classifier

- **Quantum Machine Learning (Finance)**
  - **First example: classic classifier**
    - If the step is smaller, it takes longer to reduce the error, while if the step is larger, the reduction of the error is faster in the first iterations
    - Error evolves during the training phases of the model. In the first steps the error goes down fast



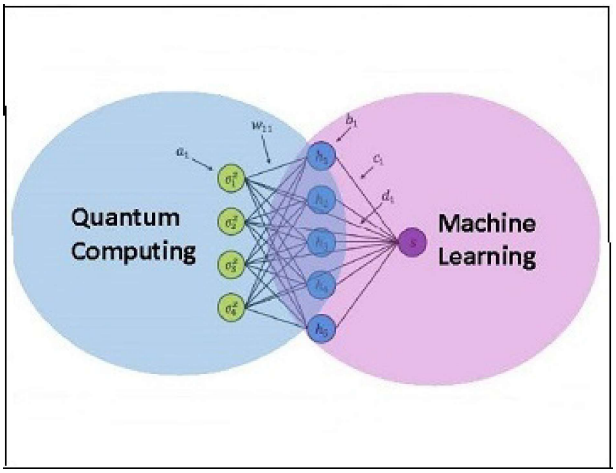
QML: Classic Classifier

- **Quantum Machine Learning (Finance)**
  - **First example: classic classifier**
    - Error evolves during the training phases of the model. in the last phases the pace is slower, refining the result



QML: Coding data + Constructing models

- ❑ QML: General Concepts
- ❑ Example: Classic Classifier
- ❑ **Constructing Models**
- ❑ Example: Quantum Classifier



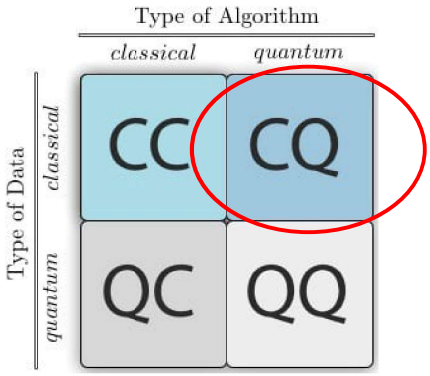
Source:
https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

QML: Coding data + Constructing models

- ❑ **Quantum Machine Learning**
  - ❑ Steps we have to take (remember):
    - ❑ Data preparation
    - ❑ Model development
    - ❑ Error estimation

QML: Coding data + Constructing models

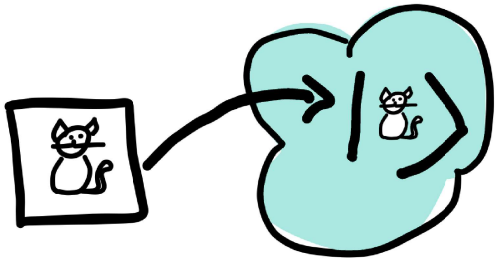
- **Quantum Machine Learning**
  - Data Preparation
    - Focusing on the data preparation part, any ML model receives a set of input data and processes it to release an output one
    - 4 approaches to QML categorized by whether the system under study is classic or quantum, and whether the information processing device is classical or quantum
    - We will work with the situation that we find in the real world, both input and output with classical data



SOURCE https://medium.com/voice-tech-podcast/demystifying-quantum-machine-learning-e1029715884b

QML: Coding data + Constructing models

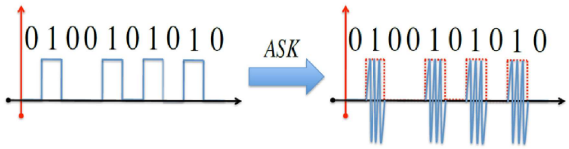
- **Quantum Machine Learning**
  - Data Preparation → Embedding
    - Starting with the classical data that we have (input) and converting it into a quantum state to use
    - This process is the so-called Embedding, let see three different ways to perform it
    - Let's imagine a bank who wants to know a clients's solvency (0-10) according the last 4 evaluations. He has obtained a score of 4, 6, 8, 9 over 10 (max rate)



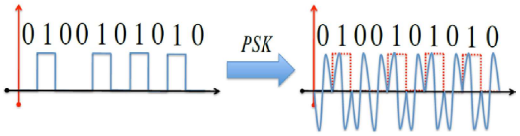
SOURCE https://pennylane.ai/qml/glossary/quantum_embedding.html

QML: Coding data + Constructing models

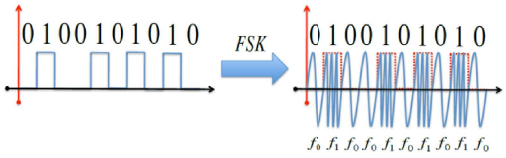
- **Quantum Machine Learning**
  - Remember analog signal modulation
    - Amplitude



    - Phase



    - Frequency



QML: Coding data + Constructing models

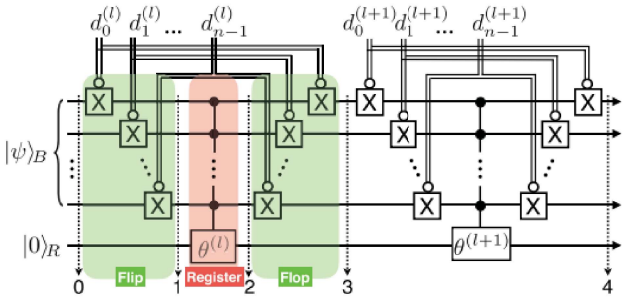
- **Quantum Machine Learning**
  - Data Preparation
    - How can we code the data of the solvency?
    - Using the computational basis → QRAM
    - Using Phase Coding
    - Using Amplitude Coding

QML: Coding data + Constructing models

❑ Quantum Machine Learning

❑ Data Preparation

- ❑ First alternative→ using a QRAM
- ❑ We see that we need 2 qbits for the indices (evaluations) and another 4 qbits for the values of 0-10 → Total of 6 qbits (I.e Third evaluation the client obtained an average of score = 9)
- ❑ Drawbacks!: Difficult to insert non-integer numbers (I.e example “4.9”)



Park, D.K., Petruccione, F. & Rhee, JK.K. Circuit-Based Quantum Random Access Memory for Classical Data. *Sci Rep* **9**, 3949 (2019). https://doi.org/10.1038/s41598-019-40439-3

QML: Coding data + Constructing models

QRAM: A Survey and Critique

Samuel Jaques* and Arthur G. Rattew†
Department of Materials, University of Oxford, Parks Road, Oxford OX1 3PH, United Kingdom

Quantum random-access memory (QRAM) is a mechanism to access data (quantum or classical) based on addresses which are themselves a quantum state. QRAM has a long and controversial history, and here we survey and expand arguments and constructions for and against.

We use two primary categories of QRAM from the literature: (1) active, which requires external intervention and control for each QRAM query (e.g. the error-corrected circuit model), and (2) passive, which requires no external input or energy once the query is initiated. In the active model, there is a powerful opportunity cost argument: in many applications, one could repurpose the control hardware for the qubits in the QRAM (or the qubits themselves) to run an extremely parallel classical algorithm to achieve the same results just as fast. Escaping these constraints requires ballistic computation with passive memory, which creates an array of dubious physical assumptions, which we examine in detail. Considering these details, in everything we could find, all non-circuit QRAM proposals fall short in one aspect or another. We apply these arguments in detail to quantum linear algebra and prove that most asymptotic quantum advantage disappears with active QRAM systems, with some nuance related to the architectural assumptions.

In summary, we conclude that *cheap, asymptotically scalable* passive QRAM is unlikely with existing proposals, due to fundamental limitations that we highlight. We hope that our results will help guide research into QRAM technologies that attempt to circumvent or mitigate these limitations. Finally, circuit-based QRAM still helps in many applications, and so we additionally provide a survey of state-of-the-art techniques as a resource for algorithm designers using QRAM.

I. INTRODUCTION

Computers are machines to process data, and so access to data is a critical function. Quantum computers are no exception. Specialized memory-access devices (e.g., RAM) are now commonplace in classical computers because they are so useful. It thus seems intuitive that future quantum computers will have similar devices.

Unfortunately for quantum computers, it's not clear that the analogy holds. Memory access, classical or quantum, requires a number of gates that grows proportional to the memory size. For a classical computer, manufacturing gates is generally a fixed cost and we care more about the runtime (though this assumption starts to break down with large scale, high-performance computing). In contrast, gates in most quantum technologies are active interventions, requiring energy or computational power to enact.

The foundational work on QRAM, [GLM08b], attempts to break from this model and imagines passive components to enact the memory, such that after a signal is sent into the device, it propagates without external

subject-specific works. Here we attempt to survey and collect all such results.

Similarly, there is a broad literature of attempts to construct QRAM (in theory or experiment) and many published criticisms of it. This paper aims to unify the existing criticisms and provide new ones, and consider all existing QRAM proposals in light of these criticisms.

A. Summary

To frame this paper, we first summarize existing arguments for QRAM and try to explain, conceptually, how to think about the nature of a QRAM device.

Classically we are used to “gates” being static physical components that data propagates through, just as depicted in circuit diagrams. Applying that intuition to quantum computing is unjustified. In contrast, the memory peripheral framework of [JS19] models quantum computing like Figure 1, where data (in the form of qubits) are static physical components, and gates are operations

CONTENTS

QML: Coding data + Constructing models

There is no large-scale functional QRAM hardware. All of them are proposals or simulations, no physical QRAM exists yet in IBM, IonQ, Rigetti, nor in superconducting nor photonic platforms

For instance, this approach is based on polynomial encoding of binary strings. It achieves an exponential improvement in T-depth and reduces T-count compared to bucket-brigade QRAM

Practical implementation is not yet feasible, as it would require trillions of physical qubits and full fault-tolerant quantum hardware

In Quantum Machine Learning today, alternative embeddings (Angle, Amplitude, Phase) are used instead of physical QRAM implementations

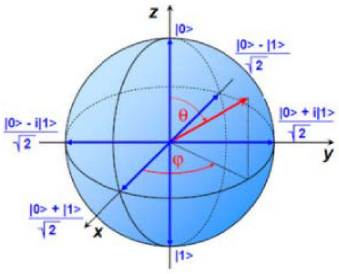
QML: Coding data + Constructing models

- **Quantum Machine Learning**
  - Data Preparation
    - Second Alternative → Phase Coding
    - We normalize the scores and encode them in rotations (for example 5/10 would mean a rotation of 0.5 radians)
    - Although the explanation will be made through the ***z*** axis (visually it is easier to understand), by agreement the *Ry* gate (rotation over Y axis) is usually used

$$\hat{R}_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$\hat{R}_y(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$\hat{R}_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$



QML: Coding data + Constructing models

- **Quantum Machine Learning**
  - Data Preparation
    - Third Alternative → Amplitude Coding
    - Data is encoded into the amplitudes of a quantum state.
    - A normalized classical N-dimensional datapoint x is represented by the amplitudes of a n-qubit quantum state $|\psi_x\rangle$. $N=2^n$, x_i is the i-th element of x, and $|i\rangle$ is the i-th computational basis state

$$|\psi_x\rangle = \sum_{i=1}^{N} x_i |i\rangle$$

QML: Coding data + Constructing models

- **Quantum Machine Learning**
  - Data Preparation
    - Third Alternative → Amplitude Coding
    - Cnorm is the normalization constant; this vector must be normalized $|\alpha|^2=1$
    - The input dataset can now be represented in the computational basis as $|\mathcal{D}\rangle = \sum_{i=1}^{2^n} \alpha_i |i\rangle$
    - where α_i are the elements of the amplitude vector α and $|i\rangle$ are the computational basis states. The number of amplitudes to be encoded is N×M
    - As a system of n qubits provides 2^n amplitudes, amplitude embedding requires $n \geq \log_2(NM)$ qubits. Example N=1000 data and M=10 features → 14 qubits

QML: Coding data + Constructing models

❑ Quantum Machine Learning

❑ Data Preparation

- ❑ Third Alternative → Amplitude Coding
- ❑ For example, let's say we want to encode the four-dimensional floating-point array x=(1.0,0.0,−5.5,0.0) using amplitude embedding
- ❑ The first step is to normalize it, $x_{norm} = \frac{1}{\sqrt{31.25}}(1.0, 0.0, -5.5, 0.0)$
- ❑ $|\psi_{x_{norm}}\rangle = \frac{1}{\sqrt{31.25}}[|00\rangle - 5.5|10\rangle]$
- ❑Let's consider the classical dataset D mentioned above. Its amplitude embedding can be easily understood if we concatenate all the input examples x(m) together into one vector, i.e.,

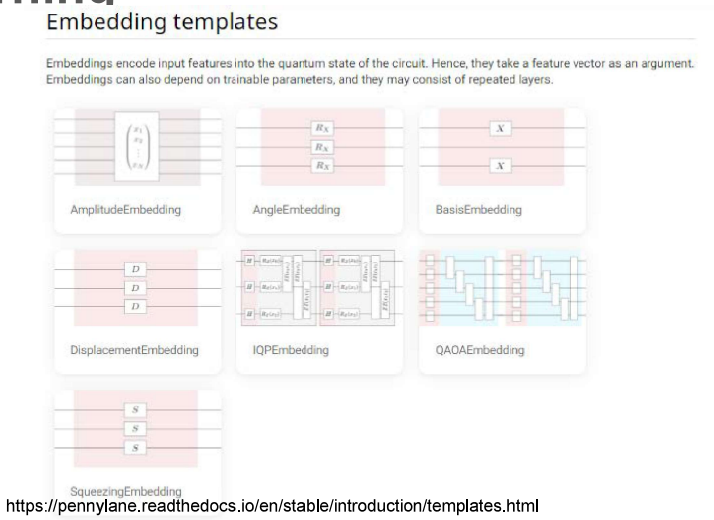
$$\alpha = C_{norm}\{x_1^{(1)}, \ldots, x_N^{(1)}, x_1^{(2)}, \ldots, x_N^{(2)}, \ldots, x_1^{(M)}, \ldots, x_N^{(M)}\}$$

QML: Coding data + Constructing models

❑ Quantum Machine Learning

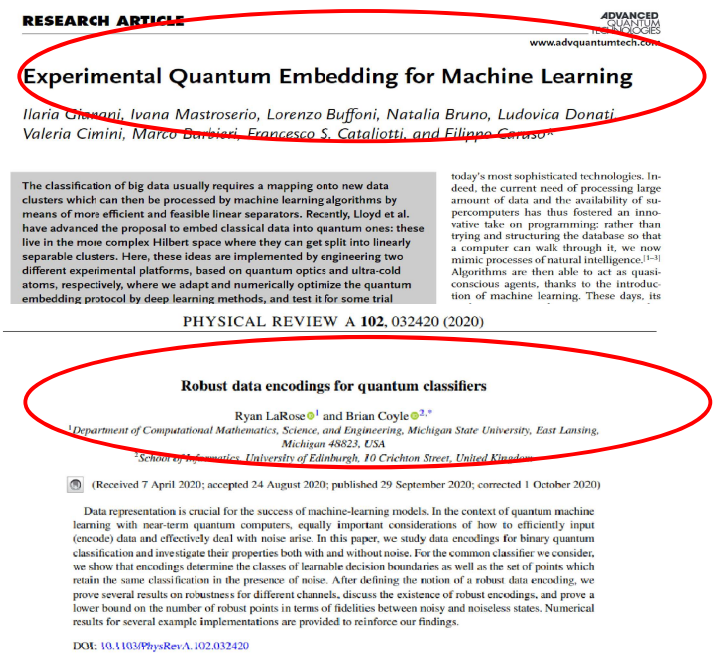
❑ Data Preparation

- ❑ Templates of embedded Coding
- ❑ (Pennylane)



https://pennylane.readthedocs.io/en/stable/introduction/templates.html

QML: Coding data + Constructing models



RESEARCH ARTICLE

ADVANCED QUANTUM TECHNOLOGIES
www.advquantumtech.com

Experimental Quantum Embedding for Machine Learning

*Ilaria Gianni, Ivana Mastroserio, Lorenzo Buffoni, Natalia Bruno, Ludovica Donati, Valeria Cimini, Marco Barbieri, Francesco S. Cataliotti, and Filippo Caruso**

The classification of big data usually requires a mapping onto new data clusters which can then be processed by machine learning algorithms by means of more efficient and feasible linear separators. Recently, Lloyd et al. have advanced the proposal to embed classical data into quantum ones: these live in the more complex Hilbert space where they can get split into linearly separable clusters. Here, these ideas are implemented by engineering two different experimental platforms, based on quantum optics and ultra-cold atoms, respectively, where we adapt and numerically optimize the quantum embedding protocol by deep learning methods, and test it for some trial

today's most sophisticated technologies. Indeed, the current need of processing large amount of data and the availability of supercomputers has thus fostered an innovative take on programming: rather than trying and structuring the database so that a computer can walk through it, we now mimic processes of natural intelligence.[1–3] Algorithms are then able to act as quasi-conscious agents, thanks to the introduction of machine learning. These days, its

PHYSICAL REVIEW A **102**, 032420 (2020)

Robust data encodings for quantum classifiers

Ryan LaRose[1] and Brian Coyle[2,*]

[1]*Department of Computational Mathematics, Science, and Engineering, Michigan State University, East Lansing, Michigan 48823, USA*

[2]*School of Informatics, University of Edinburgh, 10 Crichton Street, United Kingdom*

(Received 7 April 2020; accepted 24 August 2020; published 29 September 2020; corrected 1 October 2020)

Data representation is crucial for the success of machine-learning models. In the context of quantum machine learning with near-term quantum computers, equally important considerations of how to efficiently input (encode) data and effectively deal with noise arise. In this paper, we study data encodings for binary quantum classification and investigate their properties both with and without noise. For the common classifier we consider, we show that encodings determine the classes of learnable decision boundaries as well as the set of points which retain the same classification in the presence of noise. After defining the notion of a robust data encoding, we prove several results on robustness for different channels, discuss the existence of robust encodings, and prove a lower bound on the number of robust points in terms of fidelities between noisy and noiseless states. Numerical results for several example implementations are provided to reinforce our findings.

DOI: 10.1103/PhysRevA.102.032420

Quantum embeddings for machine learning

Seth Lloyd,[1,2] Maria Schuld,[2] Aroosa Ijaz,[2] Josh Izaac,[2] and Nathan Killoran[2]

[1]*Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA*

[2]*Xanadu, Toronto, Canada*

(Dated: July 3, 2022)

Quantum classifiers are trainable quantum circuits used as machine learning models. The first part of the circuit implements a quantum feature map that encodes classical inputs into quantum states, embedding the data in a high-dimensional Hilbert space; the second part of the circuit executes a quantum measurement interpreted as the output of the model. Usually, the measurement is trained to distinguish quantum-embedded data. We propose to instead train the first part of the circuit—the embedding—with the objective of maximally separating data classes in Hilbert space, a strategy we call *quantum metric learning*. As a result, the measurement minimizing a linear classification loss is already known and depends on the metric used: for embeddings separating data using the ℓ_1 or trace distance, this is the Helstrøm measurement, while for the ℓ_2 or Hilbert-Schmidt distance, it is a simple overlap measurement. This approach provides a powerful analytic framework for quantum machine learning and eliminates a major component in current models, freeing up more precious resources to best leverage the capabilities of near-term quantum information processors.

0 Feb 2020

Machine learning is a potential application for near-

a. Training the embedding b. Classification

IEEE Access

Multidisciplinary | Rapid Review | Open Access Journal

Received March 11, 2022, accepted April 7, 2022, date of publication April 14, 2022, date of current version April 22, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3167398

Quantum Embedding Search for Quantum Machine Learning

NAM NGUYEN, (Student Member, IEEE), AND KWANG-CHENG CHEN, (Fellow, IEEE)

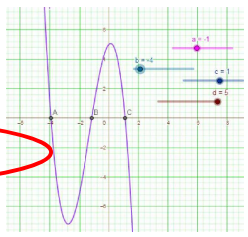
Department of Electrical Engineering, University of South Florida, Tampa, FL 33620, USA

Corresponding author: Kwang-Cheng Chen (kwangcheng@usf.edu)

This work was supported in part by the University of South Florida (USF) Quantum Initiatives.

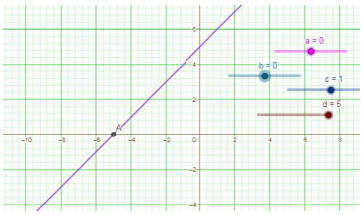
QML: Coding data + Constructing models

- **Quantum Machine Learning**
  - Model Development
    - Suppose that within the classification we want to create an algorithm that distinguishes between creditworthy clients and not
    - When we program the linear model, we are making the assumption that the objective function is represented by a line
      - We had to adapt the parameters to determine which is the line that interests us
      - Many times the line is not a good approximation to the desired objective function
      - We can test it with a cubic classifier (polynomial) $y = ax^3 + bx^2 + cx + d$



QML: Coding data + Constructing models

- **Quantum Machine Learning**
  - Model Development
    - What happens if a = 0 and b=0 ?
    - in that case the model would be equal to the definition of a linear model
    - **y = cx +d**
    - Linear model is a subset of the cubic model, the line is contemplated in both models
    - The expressiveness of the cubic model is greater than the line (the commitment to gain expressibility must be assessed at the cost of introducing more parameters → training effort)



QML: Coding data + Constructing models

- **Quantum Machine Learning**
  - Model Development
    - Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms

arXiv:1905.10876 **[quant-ph]**

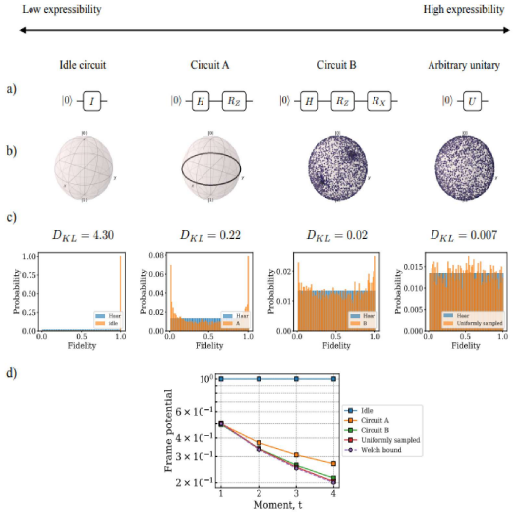
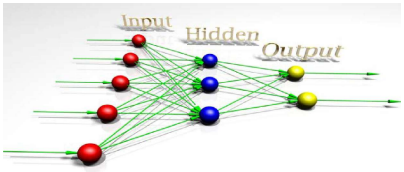


Figure 1: Quantifying expressibility for single-qubit circuits. (a) Circuit diagrams are shown for the four types of circuits. (b) For each circuit, 1000 sample pairs of circuit parameter vectors were uniformly drawn, corresponding to 2000 parameterized states that were plotted on the Bloch sphere using `QuTiP` [32]. (c) Histograms of estimated fidelities are shown, overlaid with fidelities of the Haar-distributed ensemble, with the computed KL divergences reported above the histograms. (d) The frame potential estimates for the first four moments are plotted for each circuit, with the Haar values (Welch bounds) plotted using a purple dotted line.

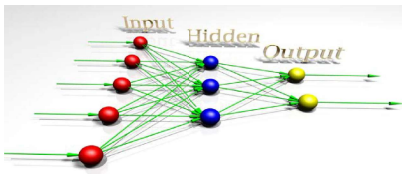
QML: Coding data + Constructing models

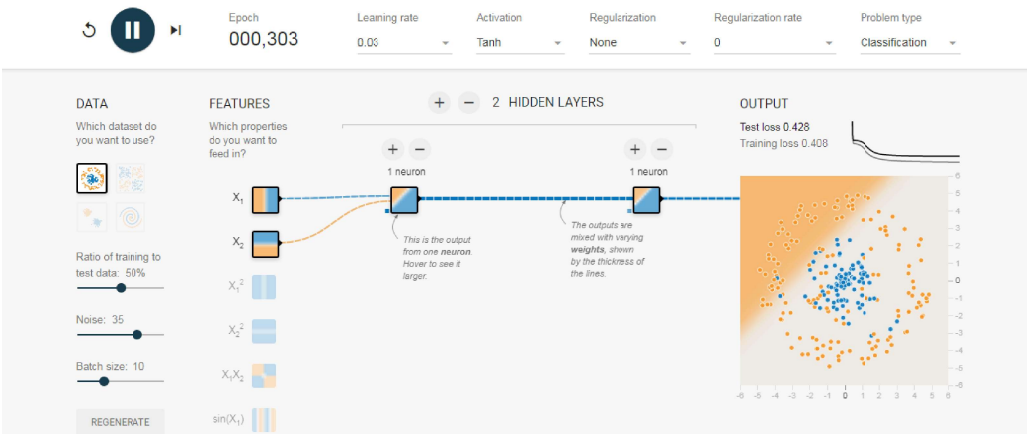
- **Quantum Machine Learning**
  - Model Development
    - It is interesting to understand the structure that each model will generate
    - For example a neural network represents a set of linear classifiers (with a non-linear element at the end)
    - Let's see the Tensor Flow playground



QML: Coding data + Constructing models

- **Quantum Machine Learning**
  - Model Development → 1 hidden neuron

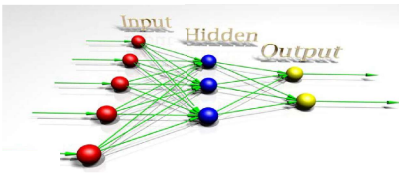


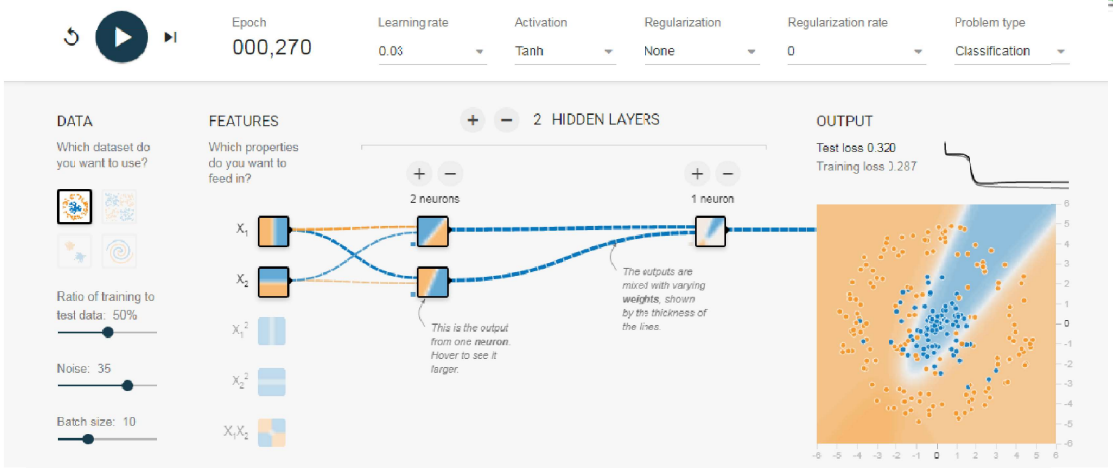


QML: Coding data + Constructing models

❑ Quantum Machine Learning

❑ Model Development → 2 hidden neurons

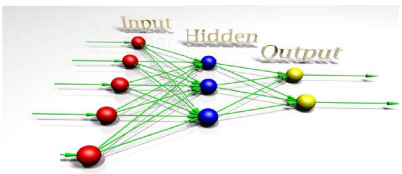


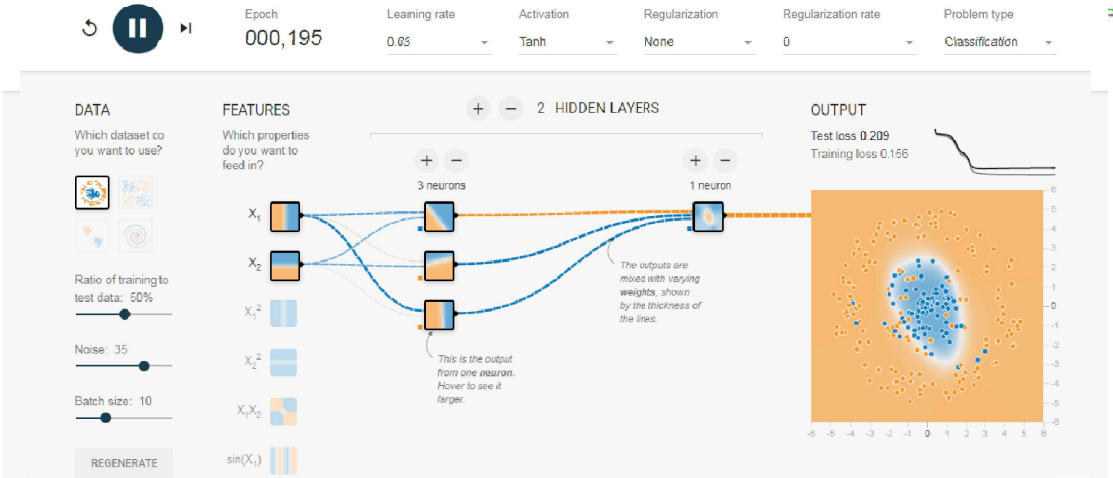


QML: Coding data + Constructing models

❑ Quantum Machine Learning

❑ Model Development → 3 hidden neurons



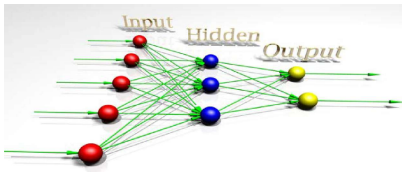


QML: Coding data + Constructing models

❑ Quantum Machine Learning

❑ Quantum Model Development

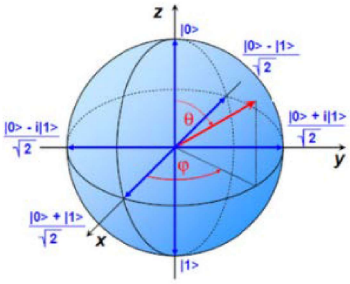
❑ We need a parameterizable function, in order to vary. What are the ones we know in QC? The Rx, Ry, Rz gates



$$\hat{R}_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$\hat{R}_y(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

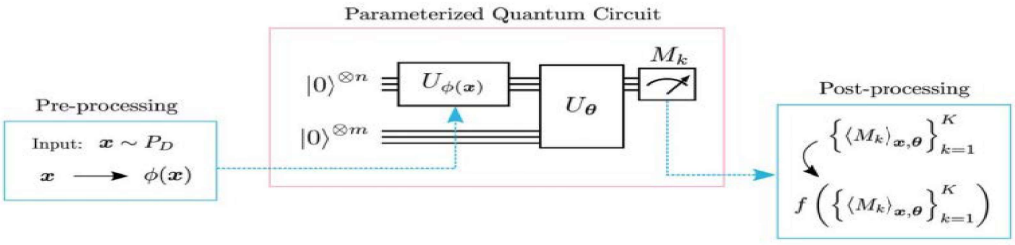
$$\hat{R}_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$



QML: Coding data + Constructing models

❑ Quantum Machine Learning

❑ Quantum Model Development



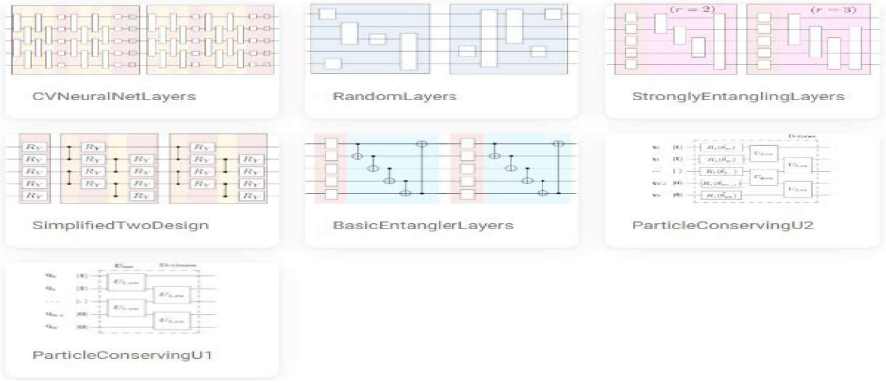
❑ Data vector is sampled from the dataset distribution, x. The pre-processing scheme maps it to the vector Φ(x) that parameterizes the encoder circuit $U_{\Phi(x)}$.

❑ A variational (parameterized) circuit U_Θ, acts on the state prepared by the encoder circuit producing the state $| U_\Theta U_{\Phi(x)}>$

QML: Coding data + Constructing models

❑ Quantum Machine Learning

❑Quantum Model Development

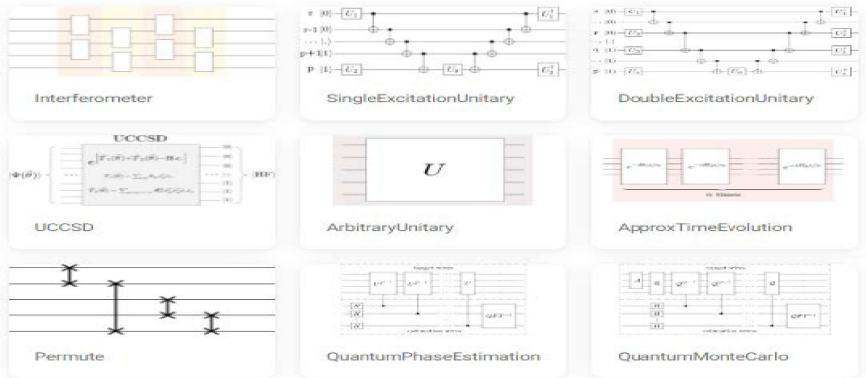


❑ Layer architectures define sequences of trainable gates that are repeated like the layers in a neural network. Note arbitrary templates or operations can also be layered

QML: Coding data + Constructing models

❑ Quantum Machine Learning

❑ Quantum Model Development



❑ Subroutines are sequences of (possibly trainable) gates that do not fulfill the conditions of other templates

QML: Coding data + Constructing models

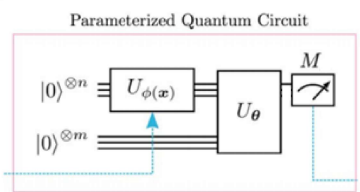
- **Quantum Machine Learning**
  - Quantum Model Development

    - We are interested in having deterministic functions:
      - Same input data & same parameters → same output values
    - Using quantum circuit, instead of 0 or 1 we will have the expected value of Pauli gate z

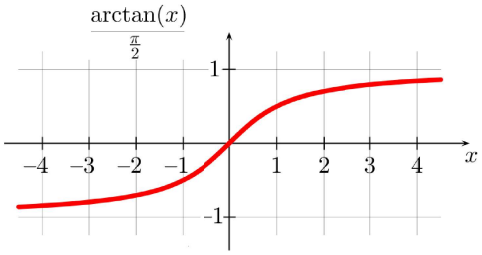
$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

QML: Coding data + Constructing models

- **Quantum Machine Learning**
  - Quantum Model Development



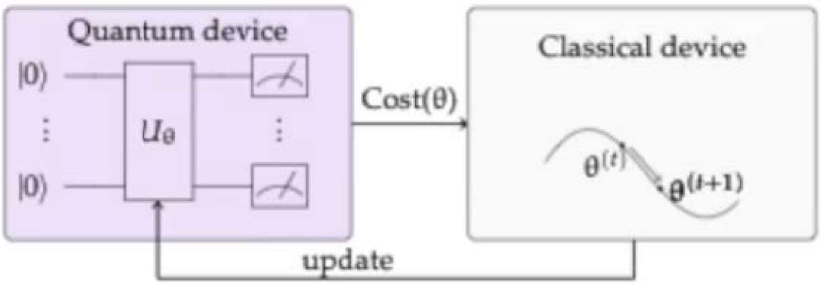
"0" means "X" class
"1" means "Y" class



- We run the circuit several times releasing the value: E{0} – E{1} which can be modeled using arc tan function
- We have reached up a continuous model! → We can measure errors and fit parameters

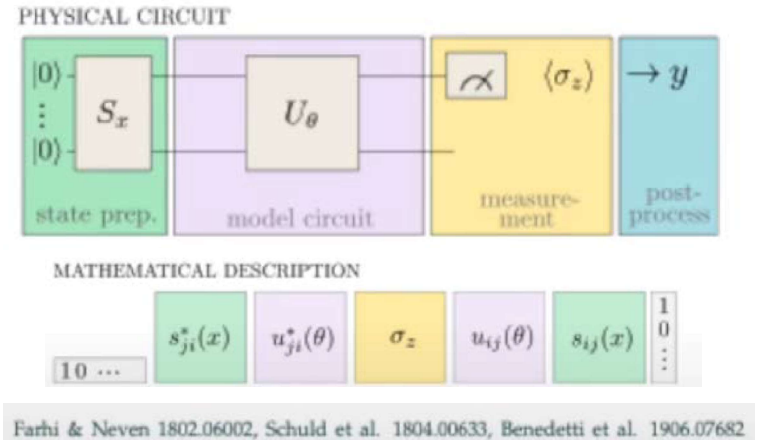
QML: Coding data + Constructing models

❑ **Resuming: Quantum circuits such as models**



QML: Coding data + Constructing models

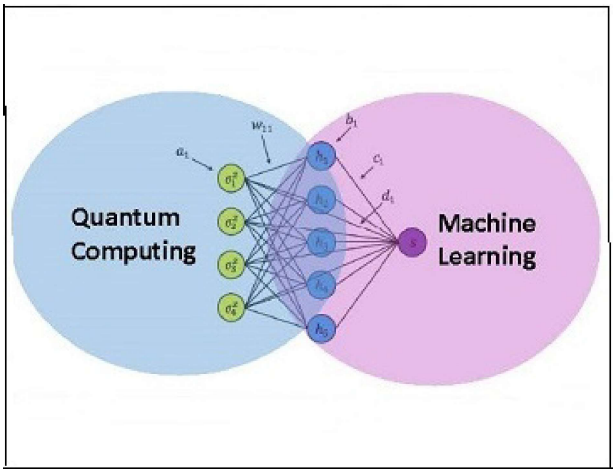
❑ **Resuming: Quantum circuits such as models**



Farhi & Neven 1802.06002, Schuld et al. 1804.00633, Benedetti et al. 1906.07682

QML: Quantum Classifier

- ❑ QML: General Concepts
- ❑ Example: Classic Classifier
- ❑ Constructing Models
- ❑ **Example:Quantum Classifier**



Source:
https://www.chem.purdue.edu/kais/research/Quantum%20Machine%20Learning.html

QML: Quantum Classifier

- ❑ **Quantum Machine Learning (Finance)**
  - ❑ **Second example: classic + quantum classifier**

```
import pennylane as qml
from pennylane import numpy as np
import matplotlib.pyplot as plt
```

- ❑ The pennylane library is imported. We remember that it is the library that will allow us to develop quantum machine learning in a simple way
- ❑ Numpy is imported from pennylane instead of the traditional library because the one from pennylane is numpy extended to work with machine learning

QML: Quantum Classifier

❑ Quantum Machine Learning (Finance)

❑ Second example: classic + quantum classifier

```
##generate the data
items = 200
class1 = np.array([[np.random.normal(loc=-2), np.random.normal(loc=2)] for i in range(items//4)])
class2 = np.array([[np.random.normal(loc=2), np.random.normal(loc=-2)] for i in range(items//4)])
class3 = np.array([[np.random.normal(loc=2), np.random.normal(loc=2)] for i in range(items//4)])
class4 = np.array([[np.random.normal(loc=-2), np.random.normal(loc=-2)] for i in range(items//4)])

plt.scatter(class1[:,0], class1[:,1],  c='b', marker='+')
plt.scatter(class2[:,0], class2[:,1],  c='b', marker='+')
plt.scatter(class3[:,0], class3[:,1],  c='r', marker='+')
plt.scatter(class4[:,0], class4[:,1],  c='r', marker='+')
plt.show()
```

- ❑ A sample of 200 elements will be created that will correspond to 4 different classes and that will have to be classified

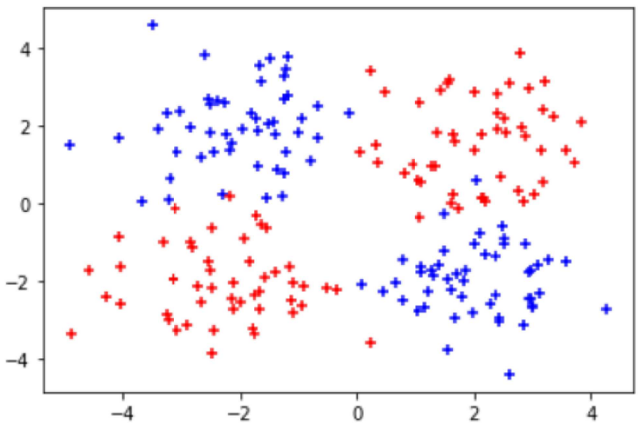
QML: Quantum Classifier

❑ Quantum Machine Learning (Finance)

❑ Second example: classic + quantum classifier

- ❑ The result of the generated elements

- ❑ Normal distributions (4):
  - ❑ 50 of these elements between -2 and 2
  - ❑ 50 elements between 2 and -2
  - ❑ 50 elements between 2 and 2
  - ❑ 50 elements between -2 and -2



QML: Quantum Classifier

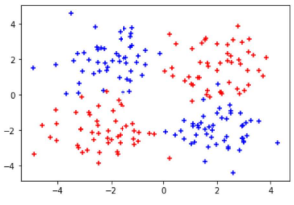
- **Quantum Machine Learning (Finance)**
  - **Second example: classic + quantum classifier**

```
feature = np.concatenate([class1, class2,class3,class4], axis=0)
name = np.concatenate([-np.ones(items // 2), np.ones(items//2)], axis=0)
data = list(zip(feature, name))
```

- Once the classes are created, we join the characteristics (feature) and the value (-1 for the blue points, and 1 for the red points)
- By joining them, the data set that we are going to classify is formed

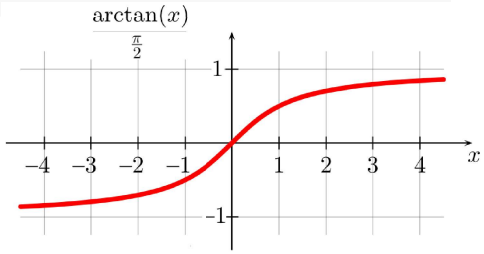
QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: classic + quantum classifier**



```
def model(x, w):
    return np.arctan(x[0]*w[0] + x[1]*w[1] + w[2]) / (np.pi / 2)
```

- We create the linear model.
- We used an arctangent function so that all the values were between -pi / 2 and pi / 2 (and normalized by pi / 2) finish between -1 and 1.



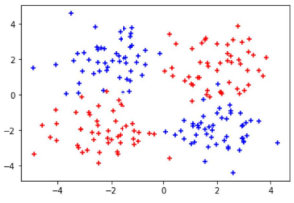
QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: classic + quantum classifier**

```
def z_function(x, w):
    return model(x,w)

limit = np.pi
x = np.arange(-limit, limit, 0.2)
y = np.arange(-limit, limit, 0.2)

def plot_model(f):
    Z = np.zeros([len(x), len(y)])
    for index, i in enumerate(x):
        for index2, j in enumerate(y):
            Z[index,index2] = f([i, j],weight)

    im = plt.imshow(Z, extent=(-limit, limit, -limit, limit), interpolation='bilinear', cmap="cividis")
    plt.scatter(class1[:,0], class1[:,1], c='b', marker='+')
    plt.scatter(class2[:,0], class2[:,1], c='b', marker='+')
    plt.scatter(class3[:,0], class3[:,1], c='r', marker='+')
    plt.scatter(class4[:,0], class4[:,1], c='r', marker='+')
    plt.axis((-limit,limit,-limit,limit))
    plt.colorbar(im);

    plt.show()
```


QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: classic+ quantum classifier**



- We need to calculate the error: It will be the mean of the calculation of all individual errors
- We apply the gradient function

```
def average_error(weight, data):
    error = 0
    for x,y in data:
        distance = model(x,weight) - y
        error += (distance) ** 2
    return error/len(data)
```

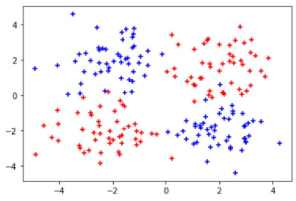
```
gradient = qml.grad(average_error, argnum=0)
```

QML: Quantum Classifier

❑ Quantum Machine Learning (Finance)

❑ Second example: classic+ quantum classifier

```
weight = np.array([-1., -0.2, 0.,0.5], requires_grad = True)

step = 0.1
for time in range(101):
    weight = weight - step*gradient(weight,data)
    if time % 5 == 0:
        print("time", time, "error", average_error(weight,data))
        plot_model(model)
```



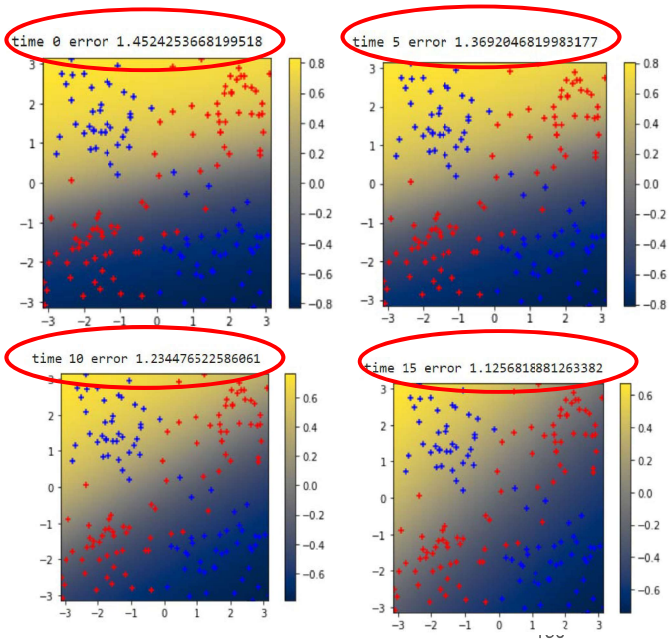
- ❑ We start from the initial value of weight [-1, -0.2,0, 0.5] and we are going to train the model for 100 iterations to help us reduce the error
- ❑ Each iterations means (size step of 0.1). Weights are modified according to the value indicated by the gradient

QML: Quantum Classifier

❑ Quantum Machine Learning (Finance)

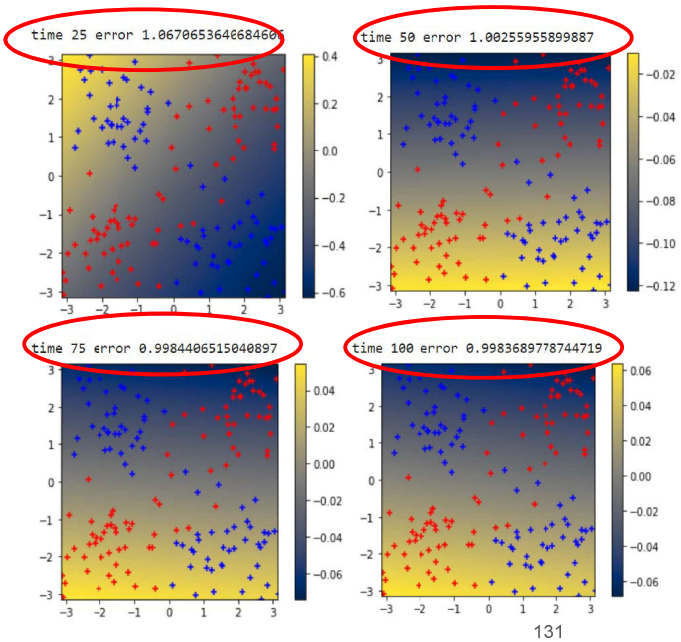
❑ Second example: classic+ quantum classifier

- ❑ If the step is smaller, it takes longer to reduce the error, while if the step is larger, the reduction of the error is faster in the first iterations
- ❑ Error evolves during the training phases of the model. In the first steps the error goes down fast



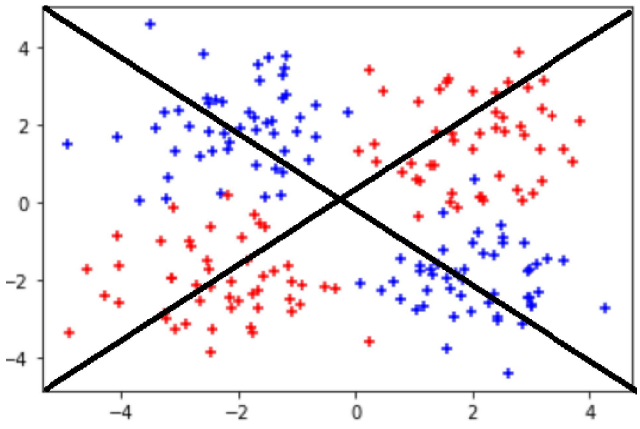
QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: classic+ quantum classifier**
    - If the step is smaller, it takes longer to reduce the error, while if the step is larger, the reduction of the error is faster in the first iterations
    - Error evolves during the training phases of the model. In the first steps the error goes down fast



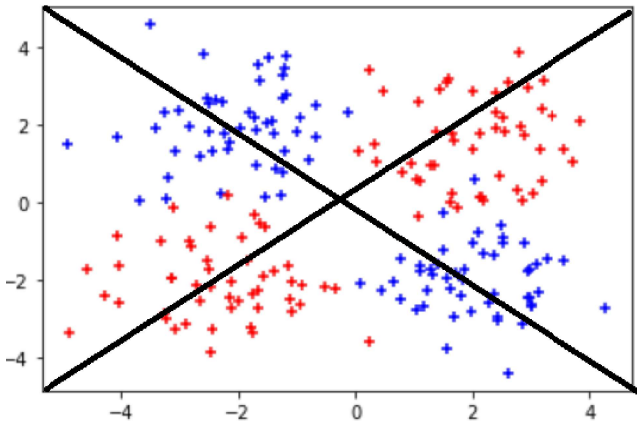
QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: classic+ quantum classifier**
    - We are applying a linear classification model to a dataset that is not linearly separable. No matter how much we train, we are not going to get any results
    - This dataset makes difficult training the model we have seen before



QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: classic+ quantum classifier**
    - What next?
    - Lets use a quantum classifier!



QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: just quantum classifier**

It takes some input (x) data and some output (w) parameters

We are encoding embedding through angle (phase)

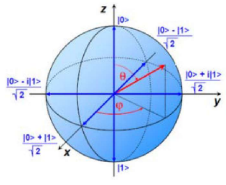
```
def model(x, w):

    qml.RX(x[0], wires = 0)
    qml.RX(x[1], wires = 1)

    qml.RX(w[0], wires = 0)
    qml.RX(w[1], wires = 1)
    qml.CNOT(wires = [1,0])
```

- Apply a Rx gate with angle x[0] at qbit_0
- Apply a Rx gate with angle x[1] at qbit_1

$$\hat{R}_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$\hat{R}_y(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

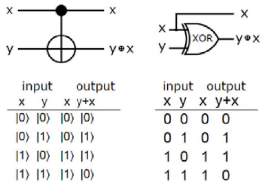
$$\hat{R}_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$



QML: Quantum Classifier

❑ Quantum Machine Learning (Finance)

❑ Second example: quantum classifier



input		output	
x	y	x	y+x
|0⟩	|0⟩	|0⟩	|0⟩
|0⟩	|1⟩	|0⟩	|1⟩
|1⟩	|0⟩	|1⟩	|1⟩
|1⟩	|1⟩	|1⟩	|0⟩

input		output	
x	y	x	y+x
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

We are only going to measure with respect to qbit 0 and the cnot connects us qbits 1 and qbit 0

```
def model(x, w):

    qml.RX(x[0], wires = 0)
    qml.RX(x[1], wires = 1)

    qml.RX(w[0], wires = 0)
    qml.RX(w[1], wires = 1)
    qml.CNOT(wires = [1,0])
```

- ❑ The information from qbit_1 goes into the final result

QML: Quantum Classifier

❑ Quantum Machine Learning (Finance)

❑ Second example: quantum classifier

- ❑ We just have defined our model
- ❑ Where this model will be executed?

A service needs stablished (using 2 qbits), we can use real backends or simulator such IBM computers… by default is the penyline simulator

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

@qml.qnode(device)
def model(x, w):

    qml.RX(x[0], wires = 0)
    qml.RX(x[1], wires = 1)

    qml.RX(w[0], wires = 0)
    qml.RX(w[1], wires = 1)
    qml.CNOT(wires = [1,0])


    return qml.expval(qml.PauliZ(wires = 0))
```

QML: Quantum Classifier

❑ Quantum Machine Learning (Finance)

❑ Second example: quantum classifier

- ❑ We just have defined our model
- ❑ Where this model will be executed?

A service needs to be encapsulated by the concept of node (model and service under same object)

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

@qml.qnode(device)
def model(x, w):

    qml.RX(x[0], wires = 0)
    qml.RX(x[1], wires = 1)

    qml.RX(w[0], wires = 0)
    qml.RX(w[1], wires = 1)
    qml.CNOT(wires = [1,0])


    return qml.expval(qml.PauliZ(wires = 0))
```

QML: Quantum Classifier

❑ Quantum Machine Learning (Finance)

❑ Second example: quantum classifier

- ❑ We have calculating the gradient under error function, which is instantiating to a circuit

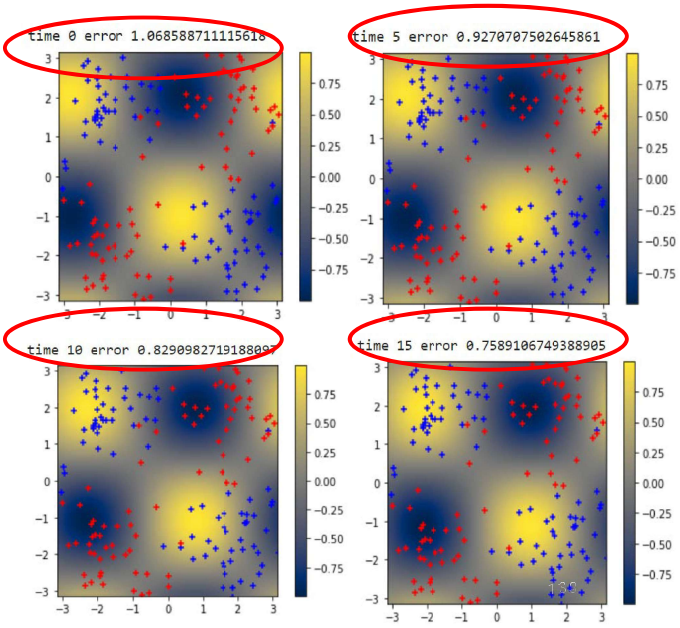
```
def average_error(weight, data):
    error = 0
    for x,y in data:
        distance = model(x,weight) - y
        error += (distance) ** 2
    return error/len(data)
```

```
gradient = qml.grad(average_error, argnum=0)
```

- ❑ In short → We have making the derivative of a quantum circuit!
- ❑ It will be performed in an automatic way (exact value, powerful finding)!

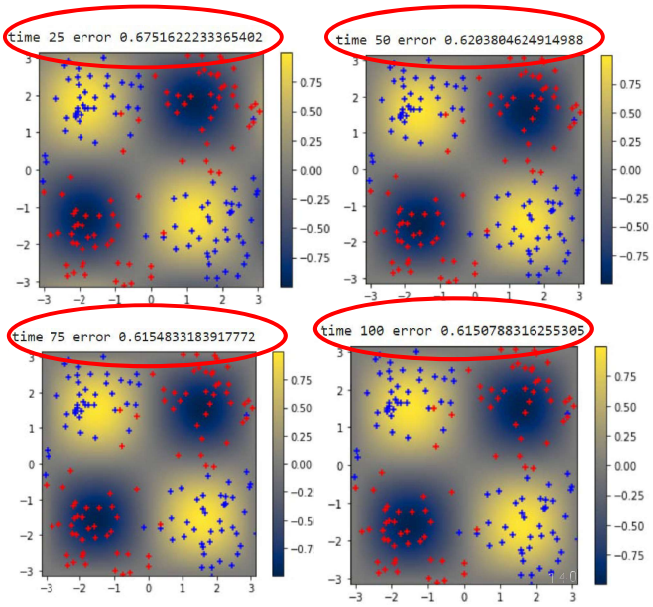
QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: quantum classifier**
    - It is about adjusting the blue points in the yellow wells and the red points in the blue wells
    - Error evolves during the training phases of the model. In the first steps the error goes down fast



QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: quantum classifier**
    - It is about adjusting the blue points in the yellow wells and the red points in the blue wells
    - Error evolves during the training phases of the model. In the first steps the error goes down fast



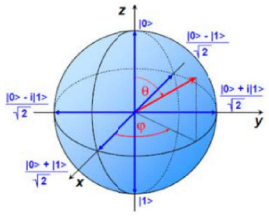
QML: Quantum Classifier

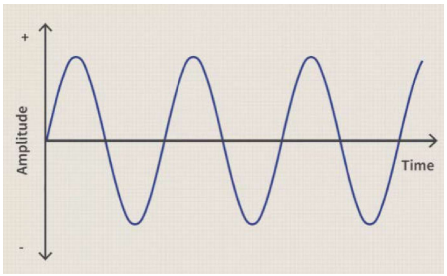
- **Quantum Machine Learning (Finance)**
  - **Second example: quantum classifier**

$$\hat{R}_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$\hat{R}_y(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$\hat{R}_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$



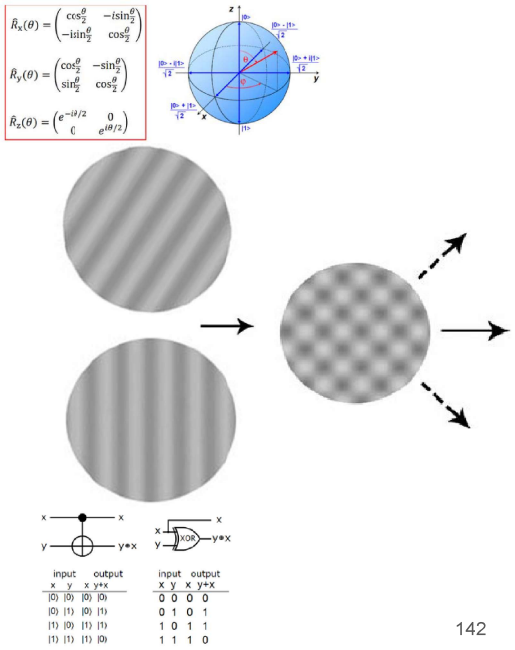


- The periodic function that goes from 1 to -1 models the expected values of the qbit that is rotating along R gate

QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: quantum classifier**

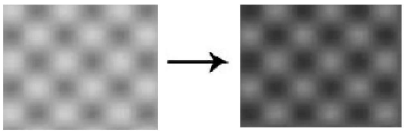
- We have two waves: (first wave → rotation), (second wave → cnot effect)
- The periodic function that goes from 1 to -1 models the expected values of the qbit that is rotating along R gate



QML: Quantum Classifier

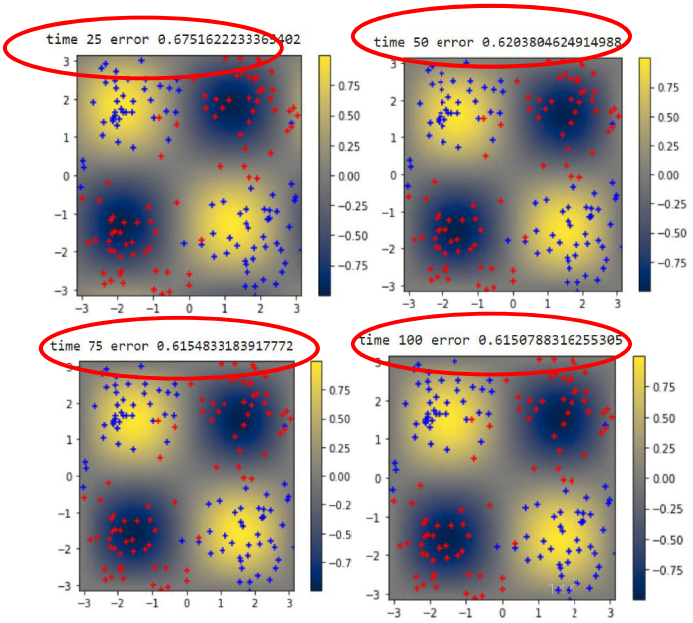
- **Quantum Machine Learning (Finance)**
  - **Second example: quantum classifier**
    - We multiply the sine grating by W_0 parameter → Displacement of the pattern

```
qml.RX(w[0], wires = 0)
qml.RX(w[1], wires = 1)
```



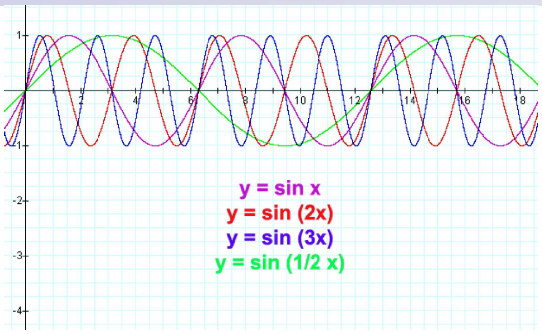
QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: quantum classifier**
    - Well size adjusts to the size of the point sets, but this does not always have to be the case
    - Would be great to be able to change the size of the wells
    - Lets modify the model a little bit



QML: Quantum Classifier

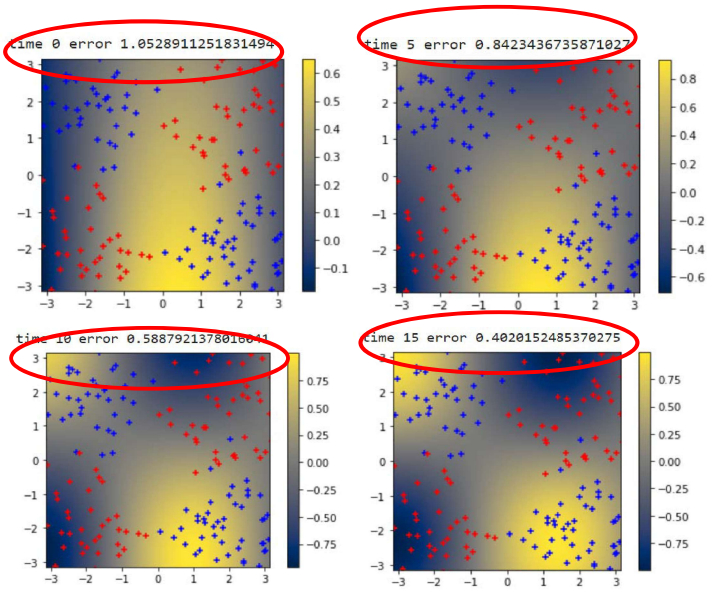
- **Quantum Machine Learning (Finance)**
  - **Second example: quantum classifier**
    - The idea is to change the period→ how will take the value (making the wells bigger), for this we multiply the input values by parameters
    - These modifications are outside the Pennylane templates




```
qml.RX(x[0]*w[2], wires = 0)
qml.RX(x[1]*w[3], wires = 1)
```

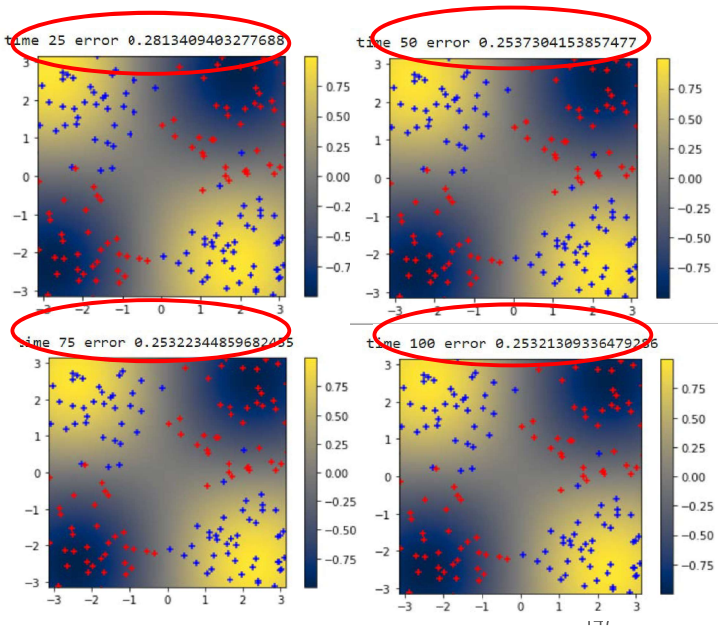

QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: quantum classifier**
    - The model improves with this mod Much better performance is obtained



QML: Quantum Classifier

- **Quantum Machine Learning (Finance)**
  - **Second example: quantum classifier**
    - The model improves with this mod Much better performance is obtained



QML: Quantum Classifier

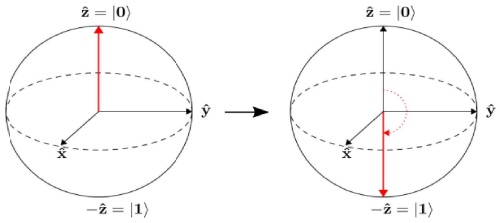
- **Quantum Machine Learning (Finance)**
  - **Pennylane**
    - https://pennylane.ai/qml/demos/tutorial_qubit_rotation.html
    - To find out more, visit the PennyLane Documentation, or check out the gallery of hands-on quantum machine learning demonstrations

QML: Quantum Classifier

❑ Quantum Machine Learning (Finance)

❑ Pennylane

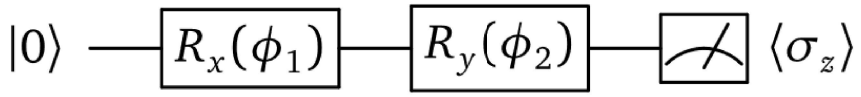
- ❑ The PennyLane version of “hello world”
- ❑ Goal is to build a single-qubit circuit that rotates a qubit to a desired pure state



QML: (“Hello world”): Purpose

❑ Quantum Machine Learning (Finance)

❑ Pennylane



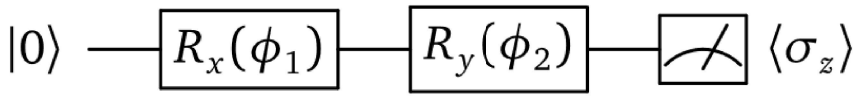
$$|\psi\rangle = R_y(\phi_2)R_x(\phi_1)|0\rangle \qquad \langle\psi \mid \sigma_z \mid \psi\rangle$$

$$R_y(\phi_2) = e^{-i\phi_2\sigma_y/2} = \begin{bmatrix} \cos\frac{\phi_2}{2} & -\sin\frac{\phi_2}{2} \\ \sin\frac{\phi_2}{2} & \cos\frac{\phi_2}{2} \end{bmatrix} \quad R_x(\phi_1) = e^{-i\phi_1\sigma_x/2} = \begin{bmatrix} \cos\frac{\phi_1}{2} & -i\sin\frac{\phi_1}{2} \\ -i\sin\frac{\phi_1}{2} & \cos\frac{\phi_1}{2} \end{bmatrix} \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

QML: (“Hello world”): Purpose

❑ Quantum Machine Learning (Finance)

❑ Pennylane



- ❑ Exact expectation value

$$\langle \psi \mid \sigma_z \mid \psi \rangle = \langle 0 \mid R_x(\phi_1)^\dagger R_y(\phi_2)^\dagger \sigma_z R_y(\phi_2) R_x(\phi_1) \mid 0 \rangle = \cos(\phi_1)\cos(\phi_2)$$

- ❑ Depending on the circuit parameters ϕ_1 and ϕ_2, the output expectation lies between 1 (if $|\psi\rangle=|0\rangle$) and −1 (if $|\psi\rangle=|1\rangle$)

QML: (“Hello world”): Import Libraries

❑ Quantum Machine Learning (Finance)

❑ Pennylane

- ❑ Important: NumPy must be imported from PennyLane to ensure compatibility with automatic differentiation
- ❑ Basically, this allows you to use NumPy as usual

```
import pennylane as qml
from pennylane import numpy as np
```

- ❑ You can also use PyTorch or TensorFlow instead of NumPy

QML: (“Hello world”): Create Device

- Device name
- Wires are subsystems (because they are represented as wires in a circuit diagram)

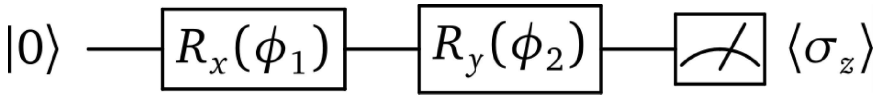
```
dev = qml.device('default.qubit', wires=1)
```

- Any computational object that can apply quantum operations and return a measurement result is called a quantum device
- In PennyLane, a device could be a hardware device (such as the Rigetti QPU, via the PennyLane-Forest plugin), or a software simulator (such as Strawberry Fields, via the PennyLane-SF plugin)

QML: (“Hello world”): Create a node

- Gate parameters
- Python decorator
- Wire the gate acts on
- Expectation value output

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



- QNodes are quantum functions, described by a quantum circuit. They are bound to a particular quantum device, which is used to evaluate expectation values of this circuit

QML: (“Hello world”): Cost function

```
def cost(params):
    expval = circuit(params)
    return np.abs(expval - (-1)) ** 2
```

- Can define any differentiable NumPy function from the output of the qnode
- In this case, we want the expectation value of the circuit to be -1

QML: (“Hello world”): Initial Parameters

```
params = np.random.normal(size=(2,))
circuit(params)
```

Dimension of params

- We can evaluate the circuit at any value of params
- In this case, there are two rotation angles, which we initialize randomly from the standard normal distribution
- When any qnode is evaluated, PennyLane calls the device itself to obtain the result

QML: (“Hello world”): Optimize the circuit

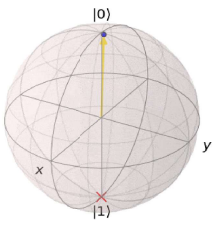
```
opt = qml.AdamOptimizer()
steps = 300

for i in range(steps):
    params = opt.step(cost, params)

print('Circuit output:', circuit(params))
print('Final parameters:', params)
```

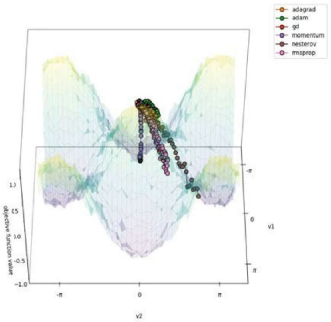
Improves parameters by gradient descent

- We can choose from a wide variety of gradient-based optimizers. In this case we select the Adam optimizer
- The parameters are optimized one step at a time for a total of 300 steps, then printed

QML: (“Hello world”): Putting all together

```
import pennylane as qml
from pennylane import numpy as np

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

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

def cost(params):
  expval = circuit(params)
  return np.abs(expval - (-1)) ** 2

params = np.random.normal(size=(2,))

opt = qml.AdamOptimizer()
steps = 300

for i in range(steps):
    params = opt.step(cost, params)

print('Circuit output:', circuit(params))
print('Final parameters:', params)
```





QML: (“Hello world”): Classical Interfaces

```python
import pennylane as qml
from pennylane import numpy as np

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

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

def cost(params):
  expval = circuit(params)
  return np.abs(expval - (-1)) ** 2

params = np.random.normal(size=(2,))

opt = qml.AdamOptimizer()
steps = 300

for i in range(steps):
  params = opt.step(cost, params)
```

TensorFlow

```python
import pennylane as qml
import torch
from torch.autograd import Variable

qpu = qml.device('forest.qpu', device='Aspen-1-2Q-B')

@qml.qnode(dev, interface='torch')
def circuit(phi1, phi2):
  qml.RX(phi1, wires=0)
  qml.RY(phi2, wires=0)
  return qml.expval(qml.PauliZ(0))

def cost(phi1, phi2):
  expval = circuit(phi1, phi2)
  return torch.abs(expval - (-1)) ** 2

phi1 = Variable(torch.tensor(1.), requires_grad=True)
phi2 = Variable(torch.tensor(0.05), requiers_grad=True)
opt = torch.optim.Adam([phi1, phi2], lr=0.1)

steps = 300

for i in range(steps):
  opt.zero_grad()
  loss = cost(phi, theta)
  loss.backward()
  opt.step()
```

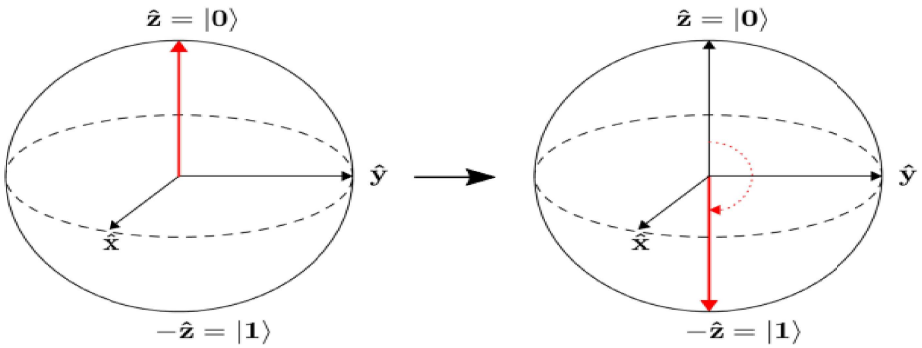
PyTorch

QML: (“Hello world”): Classical Interfaces

```python
# initialise the optimizer
opt = qml.GradientDescentOptimizer(stepsize=0.4)

# set the number of steps
steps = 100
# set the initial parameter values
params = init_params

for i in range(steps):
    # update the circuit parameters
    params = opt.step(cost, params)

    if (i + 1) % 5 == 0:
        print("Cost after step {:5d}: {: .7f}".format(i + 1, cost(params)))

print("Optimized rotation angles: {}".format(params))
```

```
Cost after step     5:  0.9961778
Cost after step    10:  0.8974944
Cost after step    15:  0.1440490
Cost after step    20: -0.1536720
Cost after step    25: -0.9152496
Cost after step    30: -0.9994046
Cost after step    35: -0.9999964
Cost after step    40: -1.0000000
Cost after step    45: -1.0000000
Cost after step    50: -1.0000000
Cost after step    55: -1.0000000
Cost after step    60: -1.0000000
Cost after step    65: -1.0000000
Cost after step    70: -1.0000000
Cost after step    75: -1.0000000
Cost after step    80: -1.0000000
Cost after step    85: -1.0000000
Cost after step    90: -1.0000000
Cost after step    95: -1.0000000
Cost after step   100: -1.0000000
Optimized rotation angles: [7.15266381e-18 3.14159265e+00]
```

QML: (“Hello world”): Classical Interfaces



- We can see that the optimization converges after approximately 40 steps.
- Substituting this into the theoretical result $\langle\psi|\sigma z|\psi\rangle=\cos\phi_1\cos\phi_2$, we can verify that this is indeed one possible value of the circuit parameters that produces $\langle\psi|\sigma z|\psi\rangle=-1$, resulting in the qubit being rotated to the state $|1\rangle$

QML: Resumee

- Deep learning began with neural networks, but nowadays the models are much richer
- Similarly, quantum machine learning can be much richer than current models

- Variational circuits: strong foundation for near-term QML
- Compute gradients of quantum circuits using “parameter shift” method
- Train quantum circuits the same as neural networks
- QNode abstraction enables highly flexible hybrid classical- quantum computation
- Speed up progress via ease-of-implementation, reusability, sharing code/models, rapid iteration

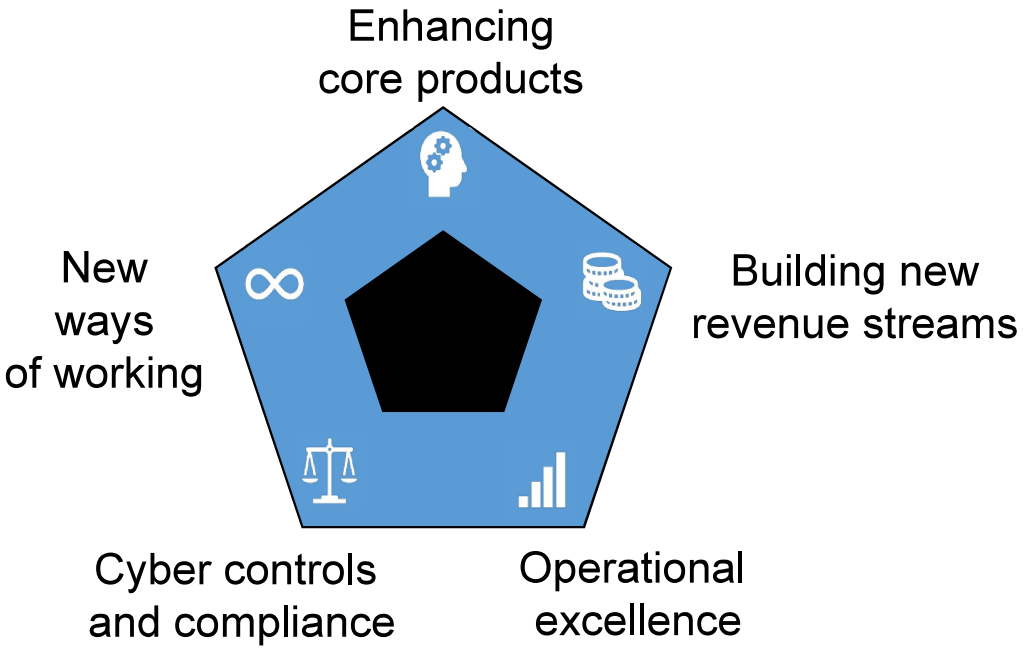
- https://github.com/XanaduAI/pennylane
- https://pennylane.readthedocs.io
- https://pennylane.ai

Why Should finnancial institutions start their journey to quantum now?

Disruptive forces are shaping the banking and financial industry

- ❑ Changing customer behaviour and expectations
- ❑ Heightened regulatory compliance burden
- ❑ Shareholder demands for improved returns
- ❑ Disruption from the FinTechs
- ❑ Expanded risks for security and fraud
- ❑ Digital disruption from Tech giants

Financial institutions are responding by

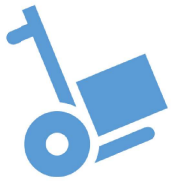


Technology innovations driving banking industry transformations

- ❑ AI Everywhere
  - ❑ Retail Banking
  - ❑ Commercial
  - ❑ Financial Markets
  - ❑ Business Process Automation

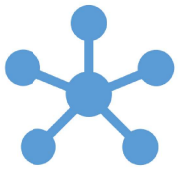
Technology innovations driving banking industry transformations

- Banking as a platform
  - Platform Economy
  - APIs
  - Open Banking and Ecosystems



Technology innovations driving banking industry transformations

- Blockchain and new collaborations
  - Trade Finance
  - Digital Identity
  - Shared KYC
  - Cross-border Payments



Technology innovations driving banking industry transformations

- Cloud as a new technology platform
  - Enterprise Cloud
  - Business Solutions as a Service

Technology innovations driving banking industry transformations

- Deeper insight and redefining computing
  - Big Data and Analytics
  - Artificial Intelligence
  - Security
  - Neuromorphic systems
  - Quantum computing

Finance is a promising environment for quantum

- Many problems in optimization are NP-hard, simulations are **massively time and resource consuming** and parts of supporting ML technologies are classical not solvable
- With quantum these challenges can be addressed, so a new level of insight and products will be reachable, ensuring **new revenue streams**

Now is the time to build your intellectual property in the quantum space

- Develop your **specific and proprietary algorithms** to enhance your products and get ahead of the competition
- Although current devices are **NISQ** systems, size and quality will improve. Current devices are similar to those expected to provide “advantage” in coming years

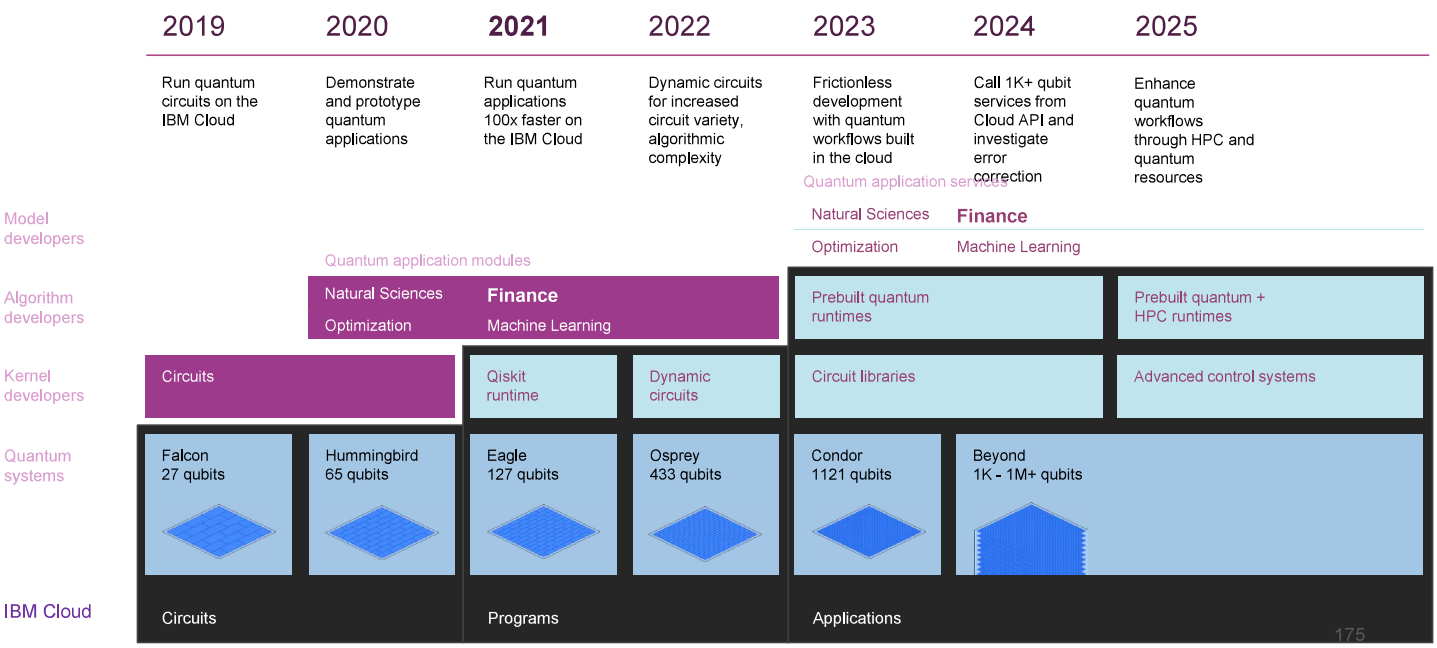
Talent people are necessary, and talents are scarce

- Bringing current challenges into a quantum code is a complex project
- It is a completely **new way of programming and thinking** and there is only limited number of people availailable with suitable multidisciplinary skill mix
- The demand is high and the war for talents has already begun

Fast-follower strategy will fail for quantum!

- Short and midterm there will be no simple commercial offerings available that can be adopted. Without the necessary skills and talents a later jump onto the train will be extremely heavy
- Further integrating such a new system into an existing and regulated IT Infrastructure and application environment will take many years

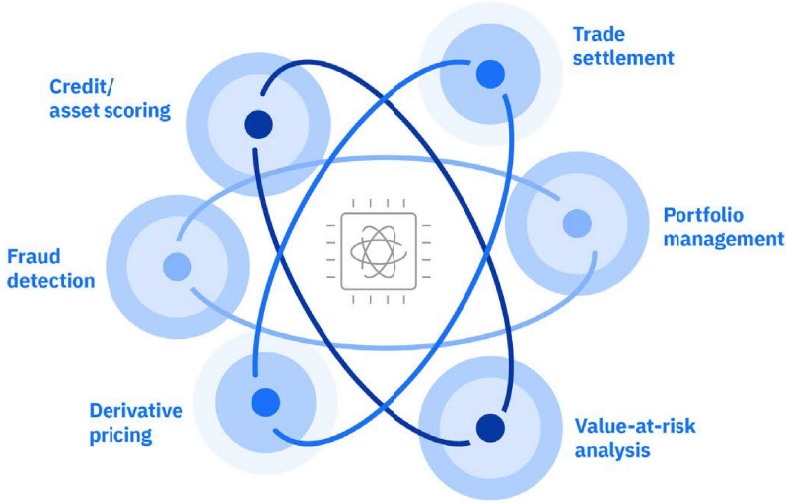
Quantum Computing – Present and future (IBM Timeline)



Application Focus Areas of Quantum Computing in Finance

Six potential uses of quantum computing that are computationally challenging in financial services trading

- **Trding Optimisation**
  - **Trade settlement:** Improving the process of transferring securities due to an asset trade
  - **Portfolio management:** Enhancing investment decisions to optimize returns and decrease risk
- **Risk profiling**
  - **Value-at-risk analysis:** Strengthening risk mitigation by developing more accurate models
  - **Derivative pricing:** Improving the pricing of financial assets using simulations
- **Targeting and Prediction**
  - **Fraud detection:** Enhancing the detection of irregular patterns to flag fraudulent transactions
  - **Credit/asset scoring:** Strengthening the statistical analysis that segments customer financial solvency and bond ratings



Trading Optimization

- Complexity in financial markets trading activity is skyrocketing
- Simulate large numbers of potential scenarios and investment options

- **Quantum computing may**
  - Enable greater investment portfolio diversification,
  - Rebalance portfolio investments more precisely based on market conditions
  - More cost-effectively streamline trading settlement processes

Trading Optimization

- Application Focus Area Examples
  - Portfolio Optimization
  - Index Tracking
  - Transaction / Security Settlement

Trading Optimization

- Optimization
  - Optimization like “Portfolio Optimization” is an NP-hard problem, often solved by sequential approximation methods
  - Transaction settlement is a difficult optimization problem due to a combination of both the legal constraints and the additional optionality

Trading Optimization

- Optimization
  - Classical
    - Linear / non-Linear / Quadratic Programming (Quadratic Unconstrained Binary Optimization - QUBO)
    - Mean-Variance Method
  - Quantum
    - Variational Quantum Eigensolver (VQE)
    - Quantum Approximation Optimization Algorithm (QAOA)

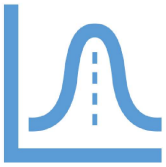
Targeting & Prediction

- Customers demand personalised products and services,
- Fraud detection systems remain highly challenged

- **Quantum computing may**
  - Prove superior in finding patterns, performing classifications, and
  - Making predictions that are not possible today because of the challenges of complex data structures

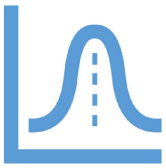
Targeting & Prediction

- Application Focus Area Examples
  - Risk Analysis (VaR, CVaR)
  - Credit Risk
  - Pricing of Financial Assets



Targeting & Prediction

- Simulation
  - Often solved by stochastic approach (Monte Carlo) used to simulate the effect of uncertainties affecting financial objects, speed of convergence dependent on number of samples means complex and long running calculations



Targeting & Prediction

- Simulation
  - Classical
    - Black-Scholes-Merton model
    - Monte Carlo Simulation
  - Quantum
    - Quantum Amplitude Estimation (QAE)

Risk Profiling

- Balance risk, hedge positions more effectively
- Perform a wider range of stress tests
- Liquidity management, derivatives pricing

- **Quantum computing may**
  - Process a much larger array of risk-management stress scenarios

Risk Profiling

- Application Focus Area Examples
  - Credit Scoring
  - Fraud
  - Anti Money Laundry

Risk Profiling

- Machine Learning
  - Find kernel functions for data transformation
  - High dimensional feature space make Hyperplane optimizations complex
  - Complex training of Boltzmann machines, require evaluation of normalization factor for loss function derivatives

Risk Profiling

- Machine Learning
  - Classical
    - Regression
    - Boltzmann Machines
    - Support Vector Machine
    - Kernel functions (polynomal, Gaussian, hyperbolic tangent)
  - Quantum
    - Quantum Support Vector Machine (QSVM)
    - Harrow-Hassidim-Lloyd (HHL)
    - Quantum Boltzmann Machine (QBM)
    - Quantum Variational Classifier (QVC)
    - Quantum Kernel Estimator (QKE)

When are the use cases in finance potentially achievable?

- Near-term quantum computers in 3-5 years?
- General purpose/error corrected machines in 10 years?
- What will the ‘quantum advantage’ be?

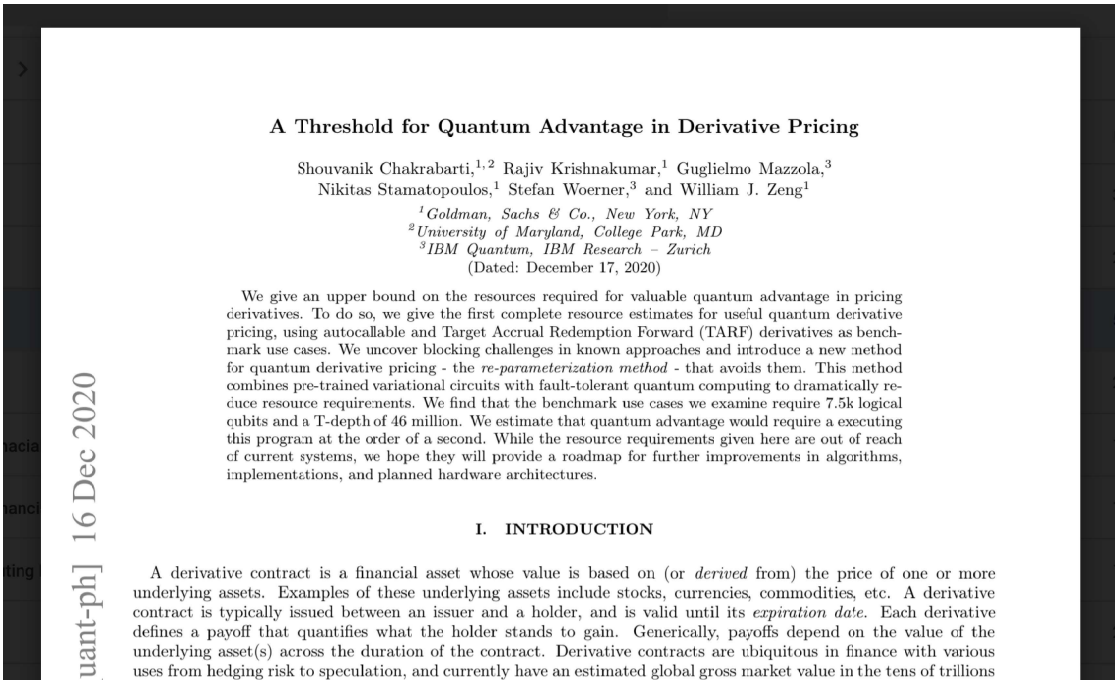
Optimization: A Point of View

- Quantum algorithm techniques are being used to approximate solutions to optimization problems, but quantum advantage is unclear and has yet to be demonstrated

Simulation: A Point of View

- A recent publication by Goldman Sachs and IBM about derivative pricing with QAE expresses a requirement for the first fault-tolerant systems such that these kinds of simulation workloads running in real world sizes can perform adequately on quantum
- On such systems quantum may show quadratic speed-up (albeit very challenging) in comparison to classic Monte Carlo simulations, therefore potentially enabling **large simulations of derivative or option pricing, or credit risk analysis might become possible in real-time**

Simulation: A Point of View



uant-ph] 16 Dec 2020

A Threshold for Quantum Advantage in Derivative Pricing

Shouvanik Chakrabarti,[1,2] Rajiv Krishnakumar,[1] Guglielmo Mazzola,[3]
Nikitas Stamatopoulos,[1] Stefan Woerner,[3] and William J. Zeng[1]

[1] *Goldman, Sachs & Co., New York, NY*
[2] *University of Maryland, College Park, MD*
[3] *IBM Quantum, IBM Research – Zurich*

(Dated: December 17, 2020)

We give an upper bound on the resources required for valuable quantum advantage in pricing derivatives. To do so, we give the first complete resource estimates for useful quantum derivative pricing, using autocallable and Target Accrual Redemption Forward (TARF) derivatives as benchmark use cases. We uncover blocking challenges in known approaches and introduce a new method for quantum derivative pricing - the *re-parameterization method* - that avoids them. This method combines pre-trained variational circuits with fault-tolerant quantum computing to dramatically reduce resource requirements. We find that the benchmark use cases we examine require 7.5k logical qubits and a T-depth of 46 million. We estimate that quantum advantage would require a executing this program at the order of a second. While the resource requirements given here are out of reach of current systems, we hope they will provide a roadmap for further improvements in algorithms, implementations, and planned hardware architectures.

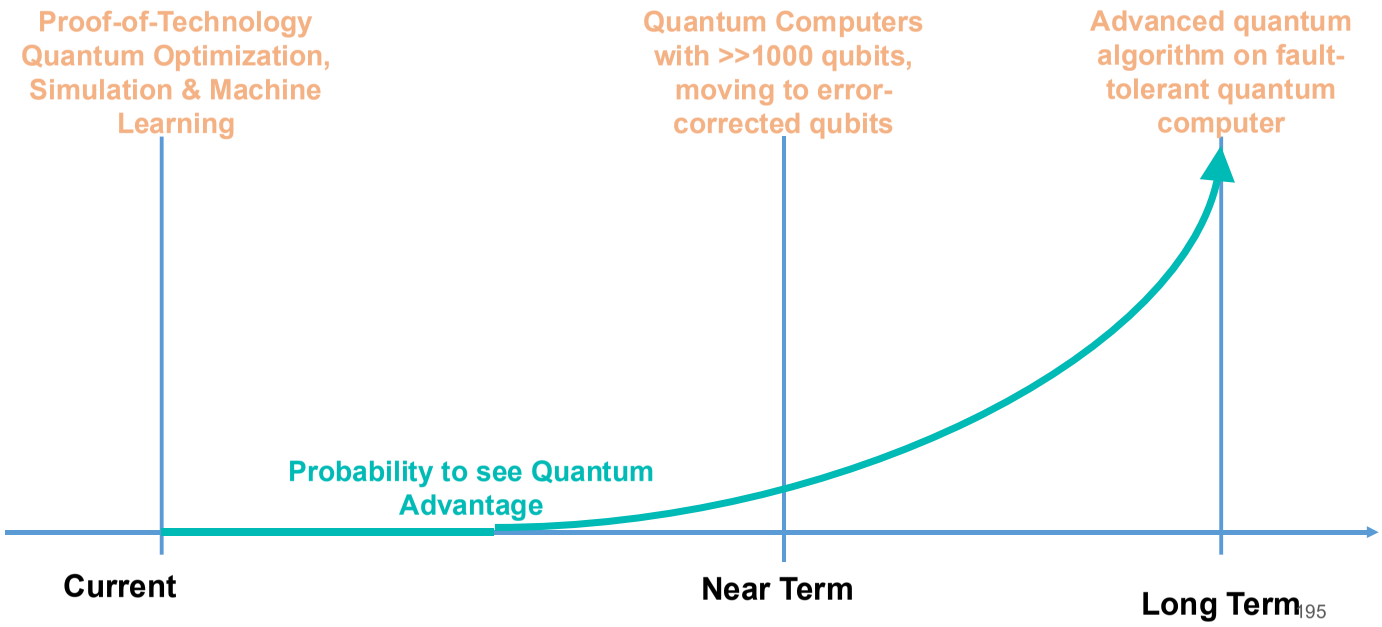
I. INTRODUCTION

A derivative contract is a financial asset whose value is based on (or *derived* from) the price of one or more underlying assets. Examples of these underlying assets include stocks, currencies, commodities, etc. A derivative contract is typically issued between an issuer and a holder, and is valid until its *expiration date*. Each derivative defines a payoff that quantifies what the holder stands to gain. Generically, payoffs depend on the value of the underlying asset(s) across the duration of the contract. Derivative contracts are ubiquitous in finance with various uses from hedging risk to speculation, and currently have an estimated global gross market value in the tens of trillions

Machine Learning: A Point of View

- An optional learning approach in ML for showing quantum advantage would be a hybrid, i.e., combining both quantum and classical machine learning technology
- Early papers about QML (e.g., regarding QSVM) show potential for a quantum approach to provide good results in high speed up and accuracy, in comparison to classical approaches on near-term quantum computers
- This makes the ML area a good candidate for early success of quantum techniques, to potentially provide speed-ups and improved accuracy compared to classical approaches, even on near-term quantum computing
- Classical ML techniques (Bayesian Optimization, Graph Theory, Time Series Forecasting etc.) are already under consideration or examination in different kinds of financial use cases (price forecasting, credit scoring, fraud detection, AML)
- But it has to be considered that classical ML techniques still have their challenges, e.g. limited explainability which cannot be solved by quantum neither

Quantum Algorithm in Banking - Future Outlook



Quantum for Finance – Proved Quantum Algorithms

Quantum Algorithm	Description	Impact
VQE - Variational Quantum Eigensolver	Use energy states to calculate the function of the variables to optimize	Procedure to assign compute-intensive functions to quantum and those of controls to classical
QAOA - Quantum Approximate Optimization	Optimize combinatorial style problems to find solutions with complex constraints	Simplify analysis clauses for constraints and provide robust optimization in complex scenarios
QAE - Quantum Amplitude Estimator	Create simulation scenarios by estimating an unknown property, Monte Carlo style	Handle random distributions directly, instead of only sampling, to solve dynamic problems quadratically speeding up simulations
QSVM - Quantum Support Vector Machines	Supervised machine learning for high dimensional problem sets	Map data to quantum-enhanced feature space to enable separation and better separate data points to achieve more accuracy
HHL – Harrow-Hassidim-Lloyd	Estimate the resulting measurement of large linear systems	Solve high dimensional problems speeding up exponentially calculations
QSDP - Quantum Semidefinite Programming	Optimize a linear objective over a set of positive semidefinite matrices	Estimate quantum system states with less measurements to exponentially speedup in terms of dimension and constraints
VQBM – Variational Quantum Boltzmann Machines	Facilitates near-term compatible QBM training with gradients of the actual loss function	Using arbitrary parameterized Hamiltonian which do not have to be fully visible but also may include hidden units
QVC/QKE – Quantum Variational Classifier / Kernel Estimator	Generates a seperating hyperplane in / estimate the kernel function of the quantum feature space	Overcome limitations when the feature space becomes large and the kernel functions become computationally expensive to estimate

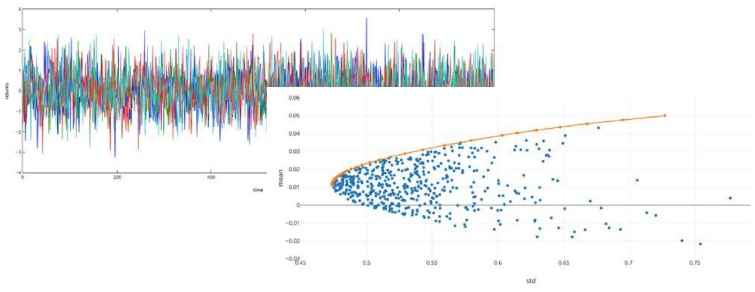
Use Cases Deep Dive: Portfolio Optimization

Portfolio Optimization

Markowitz Portfolio Optimization:

1. Assume different assets, each with a return series
2. Sample returns from a normal distribution.
3. Generate the mean returns and volatility for may random portfolios

Plotting those you will observe that they form a characteristic parabolic shape called the ‘Markowitz bullet‘ with the boundaries being called the ‘efficient frontier‘ (lowest variance for a given expected).



Quantum alternative:

Hybrid quantum/classical variational algorithms:

Minimize the expectation of the problem Hamiltonian for a parameterized trial quantum state. The expectation is estimated as the sample mean of a set of measurement outcomes, while the parameters of the trial state are optimized classically

VQE - Variational Quantum Eigensolver

P. K. Barkoutsos, G. Nannicini, A. Robert, I. Tavernelli, and S. Woerner, Quantum 4, 256 (2020).

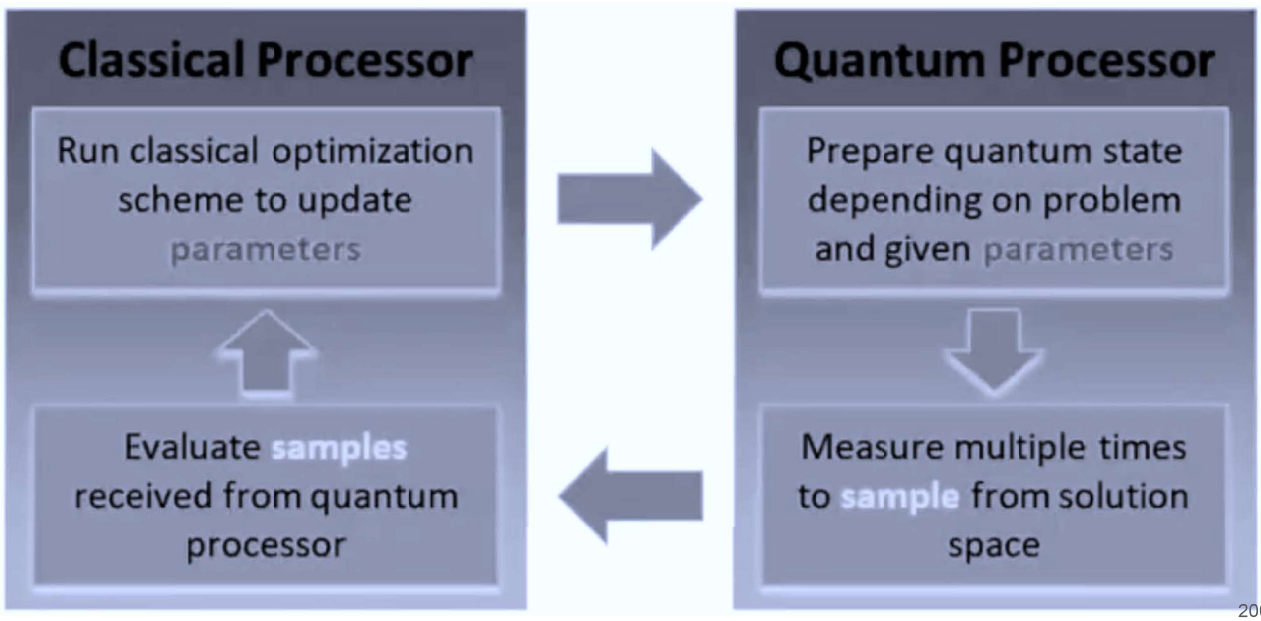
Variational Quantum Eigensolver (VQE)

- Use ideas from quantum chemistry to compute ground state
  - Hybrid Quantum/ Classical Optimization Algorithms
- Replace $|\psi\rangle$ by $|\psi(\theta)\rangle$ using a variational form and optimize over θ:

$$\min_\theta \langle\psi(\theta)|\, H|\psi(\theta)\rangle$$

- For all variational algorithms:
  - Heuristics
  - Speed-up unclear

Hybrid Quantum / Classical Optimization Algorithm



Quadratic Unconstrained Binary Optimization (QUBO)

Binary Optimization: $x_i \in \{0, 1\}$

Quadratic: $x_i * x_j$

Unconstrained:

$$\min_{x \in \{0,1\}^n} x^T Q x + c^T x$$

→Linear equality constraints via quadratic penalty terms:

$$\left(\mathrm{a}^{\mathrm{T}} x - b\right)^2$$

Mapping of QUBO to Ising Hamiltonian

1. Set $x_i = \frac{z_i+1}{2}$ for $z_i \in \{-1, +1\}$ → $\tilde{c}, \tilde{Q}$

2. Replace z_i by $\sigma_Z^i = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

→ Ising Hamiltonian H with ground state (minimal energy state) corresponding to optimal solution of original optimization problem:

$$H = \sum_i \tilde{c}_i \sigma_Z^i + \sum_{i,j} \tilde{Q}_{ij} \sigma_Z^i \otimes \sigma_Z^j$$

→ Find ground state:

$$\min_{|\psi\rangle_n} \langle \psi | H | \psi \rangle$$

Portfolio Optimization Problem Definition

solve the following mean-variance portfolio optimization problem for n assets:

$$\min_{x\in\{0,1\}^n} qx^T\Sigma x - \mu^T x$$
$$\text{subject to: } 1^T x = B$$

where we use the following notation:

- $x\in\{0,1\}^n$ denotes the vector of binary decision variables, which indicate which assets to pick ($x[i]$=1) and which not to pick ($x[i]$=0)
- $\mu\in\mathbb{R}n$ defines the expected returns for the assets
- $\Sigma\in\mathbb{R}^{n\times n}$ specifies the covariances between the assets
- q>0 controls the risk appetite of the decision maker
- and B denotes the budget, i.e. the number of assets to be selected out of n

We assume the following simplifications:

- all assets have the same price (normalized to 1)
- the full budget B has to be spent, i.e. one has to select exactly B assets

The equality constraint 1^Tx=B is mapped to a penalty term $(1^Tx-B)^2$ which is scaled by a parameter and subtracted from the objective function
The resulting problem can be mapped to a Hamiltonian whose ground state corresponds to the optimal solution

Portfolio Optimization: DOCPlex Model

```
# create docplex model
mdl = Model('portfolio_optimization')
x = mdl.binary_var_list('x{}'.format(i) for i in range(n))
objective = mdl.sum([mu[i]*x[i] for i in range(n)])
objective -= q * mdl.sum([sigma[i,j]*x[i]*x[j] for i in range(n) for j in range(n)])
mdl.maximize(objective)
mdl.add_constraint(mdl.sum(x[i] for i in range(n)) == budget)

# case to
qp = QuadraticProgram()
qp.from_docplex(mdl)
```

Portfolio Optimization: QUBO

```
# we convert the problem to an unconstrained problem for further analysis,
# otherwise this would not be necessary as the MinimumEigenSolver would do this
# translation automatically
linear2penalty = LinearEqualityToPenalty(penalty=penalty)
qp = linear2penalty.convert(qp)
```


Portfolio Optimization: Pauli matrices

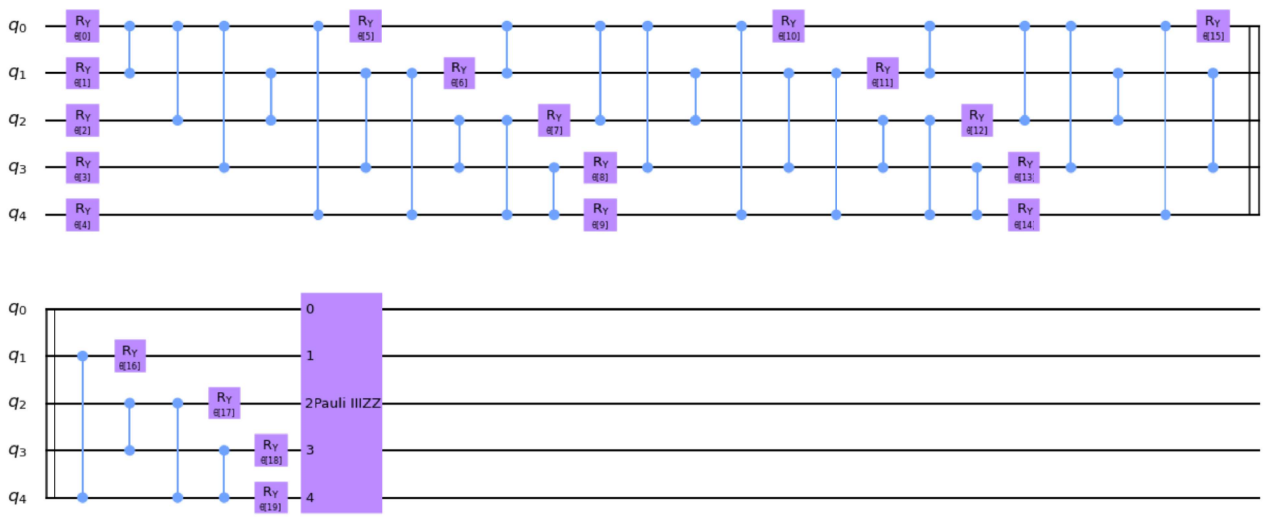
```
print(qubitOp.print_details())
```

```
IIIIZ   (-2.501436237409633+0j)
IIIZI   (-2.5015544569006765+0j)
IIIZZ   (2.500002832156184+0j)
IIZII   (-2.50067622106303+0j)
IIZIZ   (2.5000030337775563+0j)
IIZZI   (2.5000027410029517+0j)
IZIII   (-2.5011019470358145+0j)
IZIIZ   (2.500005219136618+0j)
IZIZI   (2.4999962702229923+0j)
IZZII   (2.5000078344509085+0j)
ZIIII   (-2.498742387687216+0j)
ZIIIZ   (2.499995289515744+0j)
ZIIZI   (2.4999964091976126+0j)
ZIZII   (2.5000164894710513+0j)
ZZIII   (2.5000225277800694+0j)
```

Portfolio Optimization: Optimizer

```
# set classical optimizer
maxiter = 100
optimizer = COBYLA(maxiter=maxiter)

# set variational ansatz
ansatz = RealAmplitudes(n, reps=1)
m = ansatz.num_parameters

# set backend
backend_name = 'qasm_simulator'  # use this for QASM simulator
# backend_name = 'statevector_simulator'  # use this for statevector simlator
backend = Aer.get_backend(backend_name)
```

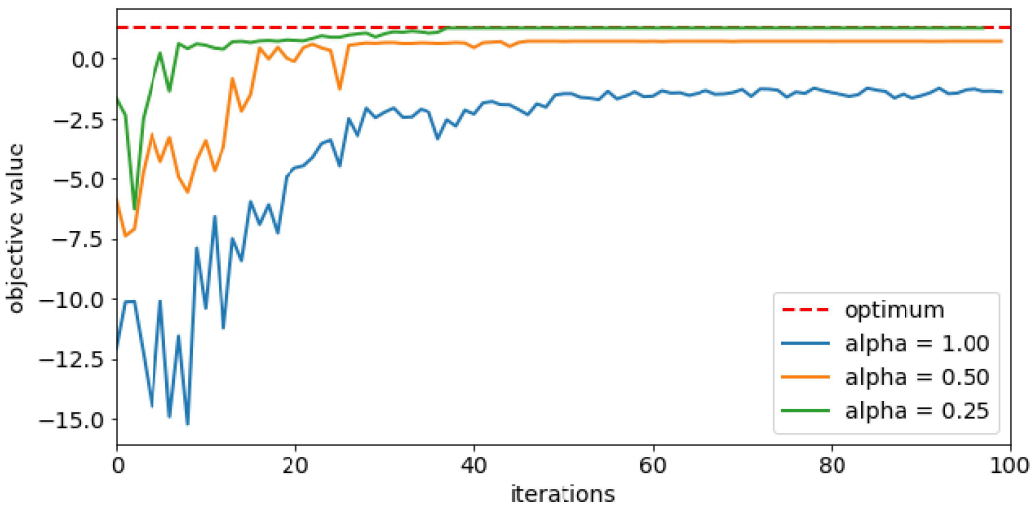

Portfolio Optimization: Ansatz



Portfolio Optimization: CVaR Expectation

```
# initialize CVaR_alpha objective
cvar_exp = CVaRExpectation(alpha, PauliExpectation())

# initialize VQE using CVaR
vqe = VQE(expectation=cvar_exp, optimizer=optimizer, ansatz=ansatz, quantum_instance=backend,
```


Portfolio Optimization: Convergence



Use Cases Deep Dive: Derivatives Pricing

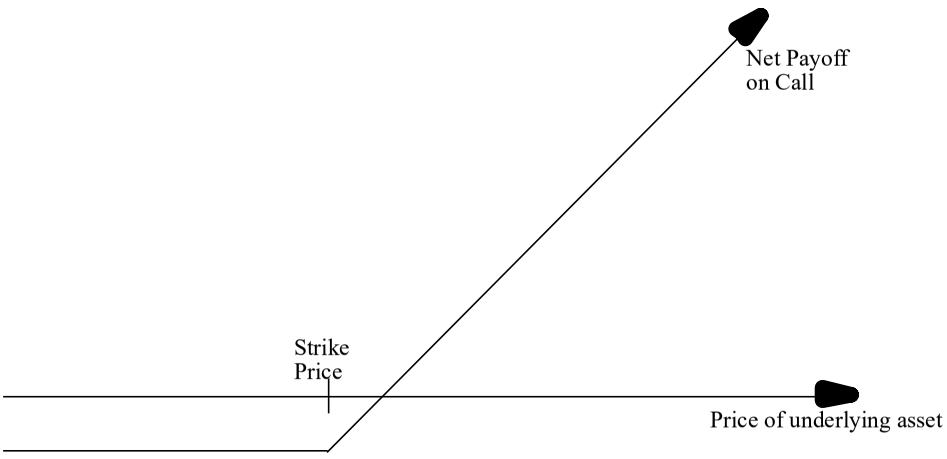
What is an option?

- An option provides the holder with the **right** to buy or sell a specified quantity of an underlying asset at a fixed price (called a strike price or an exercise price) at or before the expiration date of the option
- Since it is a right and **not an obligation**, the holder can choose not to exercise the right and allow the option to expire
- There are two types of options - **call** options (right to buy) and **put** options (right to sell)

Call Options

- A call option gives the buyer of the option the right to buy the underlying asset at a fixed price (strike price or K) at any time prior to the expiration date of the option. The buyer pays a price for this right
- At expiration,
  - If the value of the underlying asset (S) > Strike Price(K)
    - Buyer makes the difference: S - K
  - If the value of the underlying asset (S) < Strike Price (K)
    - Buyer does not exercise
- More generally,
  - the value of a call increases as the value of the underlying asset increases
  - the value of a call decreases as the value of the underlying asset decreases

Payoff Diagram on a Call



Determinants of option value

- Variables Relating to Underlying Asset
  - **Value of Underlying Asset:** as this value increases, the right to buy at a fixed price (calls) will become more valuable and the right to sell at a fixed price (puts) will become less valuable
  - **Variance in that value:** as the variance increases, both calls and puts will become more valuable because all options have limited downside and depend upon price volatility for upside
  - **Expected dividends on the asset:** which are likely to reduce the price appreciation component of the asset, reducing the value of calls and increasing the value of puts
- Variables Relating to Option
  - **Strike Price of Options:** the right to buy (sell) at a fixed price becomes more (less) valuable at a lower price
  - **Life of the Option:** both calls and puts benefit from a longer life
- Level of Interest Rates
  - As rates increase, the right to buy (sell) at a fixed price in the future becomes more (less) valuable

A Summary of the Determinants of Option Value

Factor	*Call Value*
Increase in Stock Price	Increases
Increase in Strike Price	Decreases
Increase in variance of underlying asset	Increases
Increase in time to expiration	Increases
Increase in interest rates	Increases
Increase in dividends paid	Decreases

Creating a replicating portfolio

- The objective in creating a replicating portfolio is to use a combination of risk-free borrowing and the underlying asset to create the same cashflows as the option being valued
  - Call = Borrowing + Buying Δ of the Underlying Stock
  - The number of shares bought or sold is called the **option delta**
- The principles of arbitrage then apply, and the value of the option must be equal to the value of the replicating portfolio

The Limiting Distributions….

- As the time interval is shortened, the limiting distribution, as t -> 0, can take one of two forms
  - If as t -> 0, **price changes become smaller**, the limiting distribution is the normal distribution and the **price process is a continuous one**
  - If as t->0, **price changes remain large**, the limiting distribution is the Poisson distribution, i.e., a **distribution that allows for price jumps**
- **The Black-Scholes model** applies when the **limiting distribution is the normal distribution** , and explicitly assumes that the price process is continuous and that there are no jumps in asset prices

The Black-Scholes Model

- The version of the model presented by Black and Scholes was designed to value European options, which were dividend-protected
- The value of a call option in the Black-Scholes model can be written as a function of the following variables:

  S = Current value of the underlying asset

  E = Strike price of the option

  t = Life to expiration of the option

  r = Riskless interest rate corresponding to the life of the option

  σ^2 = Variance in the ln(value) of the underlying asset

Black-Scholes Formula

- The call option payoff is

$$\text{Payoff}(S) = \max(S - E, 0)$$

- The underlying stock follows

$$\frac{dS}{S} = \mu dt + \sigma dz$$

- The is also a money market security that pays the real interest rate

$$rdt$$

Black-Scholes Formula

Portfolio Construction:

$$\Delta S = \mu S \Delta t + \sigma S \Delta z \Rightarrow \Delta f = \left(\frac{\partial f}{\partial S} \mu S + \frac{\partial f}{\partial t} + \frac{1}{2} \frac{\partial^2 f}{\partial S^2} \sigma^2 S^2 \right) \Delta t + \frac{\partial f}{\partial S} \sigma S \Delta z$$

$$\begin{cases} -1 : derivative \\ +\dfrac{\partial f}{\partial S} : share \end{cases} \Rightarrow \Pi = -f + \frac{\partial f}{\partial S} S \Rightarrow \Delta \Pi = -\Delta f + \frac{\partial f}{\partial S} \Delta S$$

$$\left.\begin{array}{l} \Delta \Pi = \left(-\dfrac{\partial f}{\partial t} - \dfrac{1}{2} \dfrac{\partial^2 f}{\partial S^2} \sigma^2 S^2 \right) \Delta t \\ \Delta \Pi = r \Pi \Delta t \end{array}\right\} \Rightarrow \left(\frac{\partial f}{\partial t} + \frac{1}{2} \frac{\partial^2 f}{\partial S^2} \sigma^2 S^2 \right) \Delta t = r \left(f - \frac{\partial f}{\partial S} S \right) \Delta t$$

$$\Rightarrow \quad \boxed{\frac{\partial f}{\partial t} + rS \frac{\partial f}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = rf}$$

Black-Scholes Formula

Risk Neutral Pricing:

$$\text{Call option value } = SN(d_1) - Ee^{-r(T-t)} N(d_2),$$

where

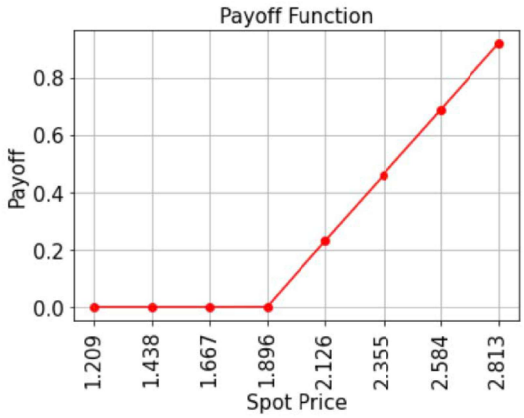
$$d_1 = \frac{\log(S/E) + (r + \frac{1}{2})(T-t)}{\sigma \sqrt{T-t}},$$

and

$$d_2 = \frac{\log(S/E) + (r - \frac{1}{2})(T-t)}{\sigma \sqrt{T-t}}.$$

Option Pricing using Quantum Computers

- Uncertainty model
- Payoff Function



Uncertainty Model

- We construct a circuit factory to load a log-normal random distribution into a quantum state. The distribution is truncated to a given interval [low,high] and discretized using 2^n grid points, where n denotes the number of qubits used. The unitary operator corresponding to the circuit factory implements the following:

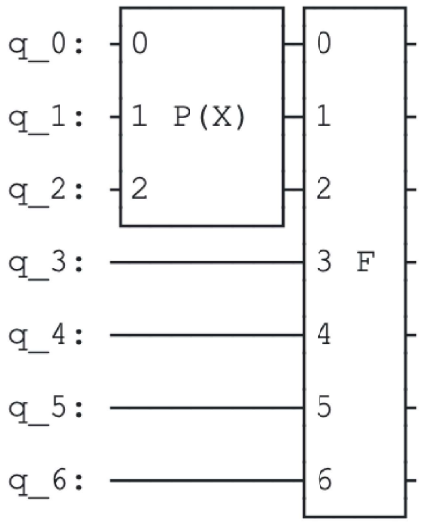
$$|0\rangle_n \mapsto |\psi\rangle_n = \sum_{i=0}^{2^n-1} \sqrt{p_i}|i\rangle_n$$

- where p_i denote the probabilities corresponding to the truncated and discretized distribution and where i is mapped to the right interval using the affine map:

$$\{0, \dots, 2^n - 1\} \ni i \mapsto \frac{\text{high} - \text{low}}{2^n - 1} * i + \text{low} \in [\text{low}, \text{high}]$$

Payoff Function

- The payoff function equals zero as long as the spot price at maturity S_T is less than the strike price E and then increases linearly
- The implementation uses a comparator, that flips an ancilla qubit from $|0\rangle$ to $|1\rangle$ if $S_T \geq E$, and this ancilla is used to control the linear part of the payoff function



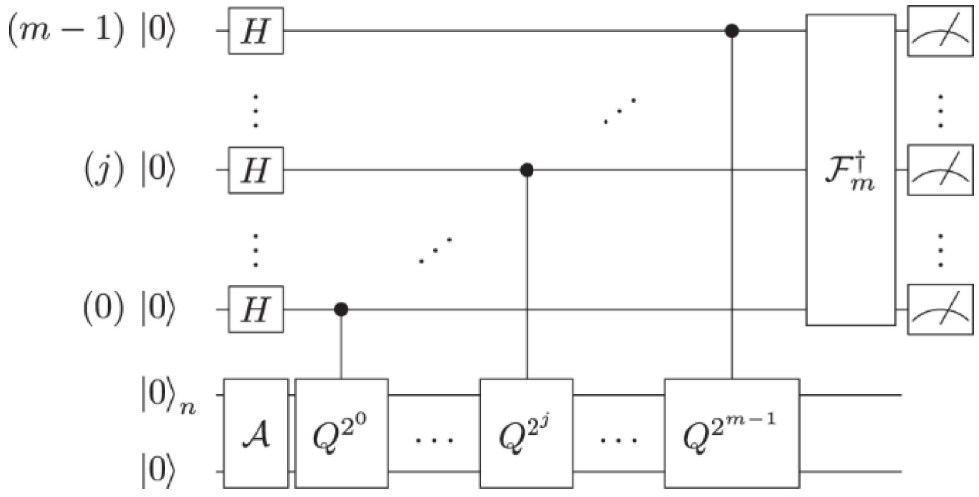
Evaluate Expected Payoff

```
# set target precision and confidence level
epsilon = 0.01
alpha = 0.05

# construct amplitude estimation
ae = IterativeAmplitudeEstimation(epsilon=epsilon, alpha=alpha,
                                  state_preparation=european_call,
                                  objective_qubits=[3],
                                  post_processing=european_call_objective.post_processing)
```

Iterative Quantum Amplitude Estimation

- A variant of *Quantum Amplitude Estimation (QAE)*, called *Iterative QAE* (IQAE), which does not rely on *Quantum Phase Estimation* (QPE) but is only based on *Grover's Algorithm*, which reduces the required number of qubits and gates
- Requires iterative queries to the quantum computer to achieve the quadratic speedup and cannot be parallelized

Canonical QAE



Iterative Quantum Amplitude Estimation

- At the heart of the techniques to estimate the amplitude without QFT and controlled Grover operators is a post processing function that combines the results of running Grover circuits with various number of iterations

Practical Sesion: Codes involved (Code I)

- **Code I:** (Getting Started with Qiskit) (0.43.0 of June 2023)
- https://qiskit.org/documentation/tutorials/circuits/1_getting_started_with_qiskit.html
- **Learn Qiskit Basics**: Start by familiarizing yourself with the Qiskit quantum computing library. Understand its capabilities to simulate arbitrary quantum processes.
- **Explore Optimization Problems**: Use Qiskit to solve a variety of optimization problems. Experiment with different problems to understand the practical performance of Qiskit.
- **Implement Quantum Algorithms**: Try implementing quantum and hybrid quantum-classical variational algorithms. These can be particularly useful for solving combinatorial optimization problems.
- **Study Quantum Risk Analysis and Machine Learning**: Use Qiskit to delve into advanced topics like quantum risk analysis and quantum machine learning. These areas can offer more efficient solutions than traditional methods.

Practical Sesion: Codes involved (Code II)

- **Code II: (Quantum Neural Networks)** (Qiskit 0.43.0 of June 2023)
- https://qiskit.org/ecosystem/machine-learning/tutorials/01_neural_networks.html
- **Understand the Basics:** Quantum Neural Networks (QNNs) are a truly quantum analogue of classical neurons, forming quantum feedforward neural networks capable of universal quantum computation. They are often trained using a parameterized quantum circuit with a classical optimization loop.
- **Overcome Challenges:** Training QNNs can be challenging due to the exponential dimension of Hilbert space and the gradient estimation complexity. Understanding these challenges is crucial for successful implementation.
- **Explore Different Types of QNNs:** There are different types of QNNs such as continuous-variable quantum neural networks, which can encode highly nonlinear transformations while remaining completely unitary.
- **Apply QNNs in Real-World Scenarios:** QNNs have potential applications in various fields, including quantum simulation, optimization, and machine learning. They can be used to gain novel insights into complex quantum-chemical systems.

Practical Sesion: Codes involved (Code III)

- **Code III: (Quantum Neural Networks)** (Qiskit 0.43.0 of June 2023)
- https://qiskit.org/ecosystem/machine-learning/tutorials/05_torch_connector.html
- **Data Loaders:** Utilize torchvision API to load subsets of the MNIST dataset and define torch DataLoaders for training and testing. vvv
- **QNN and Hybrid Model:** Leverage the power of TorchConnector to embed a quantum neural network layer (EstimatorQNN) into a torch module.
- **Training:** Define the model, optimizer, and loss function. Initiate training and save the trained model for future inference.
- **Evaluation:** Recreate the model and load the state from the saved file. You can train a model on real hardware available on the cloud and then use a simulator for inference, or vice versa.

Practical Sesion: Codes involved (Code IV)

- **Code IV: (Quantum Support Vector Machine)** (Qiskit 0.43.0 of June 2023)
- https://qiskit.org/ecosystem/machine-learning/tutorials/03_quantum_kernel.html
- **Quantum Kernel Definition**: The tutorial uses the FidelityQuantumKernel class with a ZZFeatureMap to define quantum kernels. This setup is used for both classification and clustering tasks.
- **Classification with SVC and QSVC**: The tutorial demonstrates how to use the SVC algorithm with a custom quantum kernel as a callable function and a precomputed quantum kernel matrix for classification. It also shows how to train classifiers with the QSVC algorithm from Qiskit ML
- **Clustering with Spectral Clustering Model**: The tutorial uses the SpectralClustering algorithm from scikit-learn with a precomputed quantum kernel matrix for clustering. The kernel matrix is visualized and the clustering score is calculated using normalized mutual information.
- **Kernel Principal Component Analysis (KPCA)**: The tutorial demonstrates how to use KPCA to transform the original features into a new space where the classification task can be performed with a simpler model. It shows how to use a quantum kernel with scikit-learn's KernelPCA algorithm to transform the dataset.

Practical Sesion: Codes involved (Code V)

- **Code V: (Portfolio Optimization)** (Qiskit 0.43.0 of June 2023)
- https://qiskit.org/ecosystem/finance/tutorials/01_portfolio_optimization.html
- **Problem Definition**: The tutorial uses the PortfolioOptimization application from Qiskit Finance to define a mean-variance portfolio optimization problem with a given set of parameters.
- **Classical Solution**: The tutorial first solves the problem classically using the NumPyMinimumEigensolver and the MinimumEigenOptimizer from Qiskit's optimization module.
- **Quantum Solution with SamplingVQE**: The tutorial then solves the problem using the Sampling Variational Quantum Eigensolver (SamplingVQE). It demonstrates how to specify the optimizer and variational form to be used.
- **Quantum Solution with QAOA**: Finally, the tutorial solves the problem using the Quantum Approximate Optimization Algorithm (QAOA). This is another variational algorithm that uses an internal variational form created based on the problem.

Practical Sesion: Codes involved (Code VI)

- **Code VI: (Getting real Financial Data)** (Qiskit 0.43.0 of June 2023)
- https://qiskit.org/ecosystem/finance/tutorials/11_time_series.html
- **Generating and Loading Data**: The tutorial demonstrates how to generate pseudo-Random time-series data and download actual stock-market time series from common providers.
- **Data Processing**: Once the data are loaded, various algorithms can be run to aggregate the data. This includes computing the covariance matrix or a variant, which would consider alternative time-series similarity measures based on dynamic time warping (DTW).
- **Accessing Real-Time Data**: The tutorial provides information on how to access real-time data from NASDAQ and Exchange Data International (EDI) using access tokens. It also shows how to access Yahoo Finance Data without a token.
- **Visualizing Data**: The tutorial includes code for visualizing the underlying evolution of stock prices and the covariance matrix.
- **Further Applications**: The tutorial suggests further applications of the data in portfolio optimization or portfolio diversification.