VSB TECHNICAL UNIVERSITY OF OSTRAVA

Quantum Gate Neural Networks

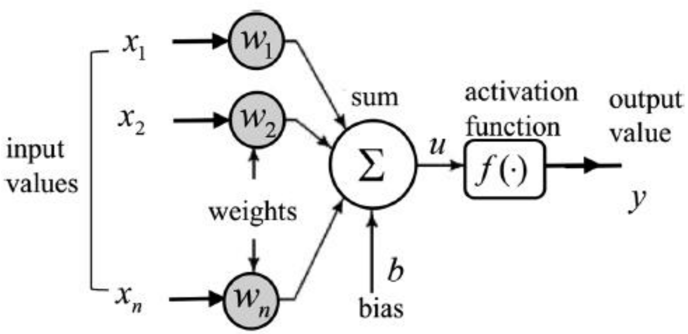
A Walkthrough From Quantum-Inspired To Quantum Domain

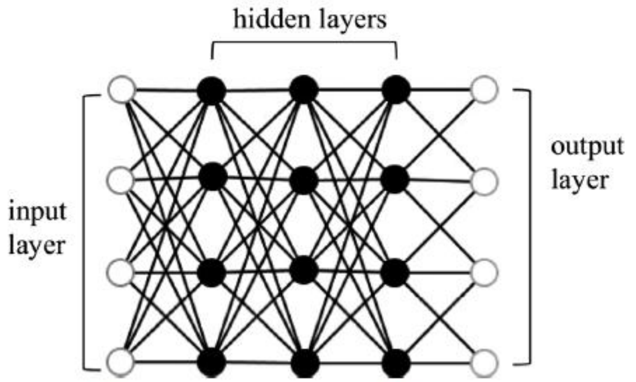
Dr. Siddhartha Bhattacharyya
VSB Technical University of Ostrava
Ostrava, Czech Republic
dr.siddhartha.bhattacharyya@gmail.com

Agenda

- Classical Neural Networks
- Quantum-Inspired Neural Networks
- TensorFlow Quantum
- Cirq
- Parameterized Quantum Circuits (PQCs)
- Quantum Neural Network (QNN) Training Model
- High-Level PQCs
- Optimization of Parameters
- Quantum Variational Autoencoder
- Advantages of QNN

Classical Neural Networks



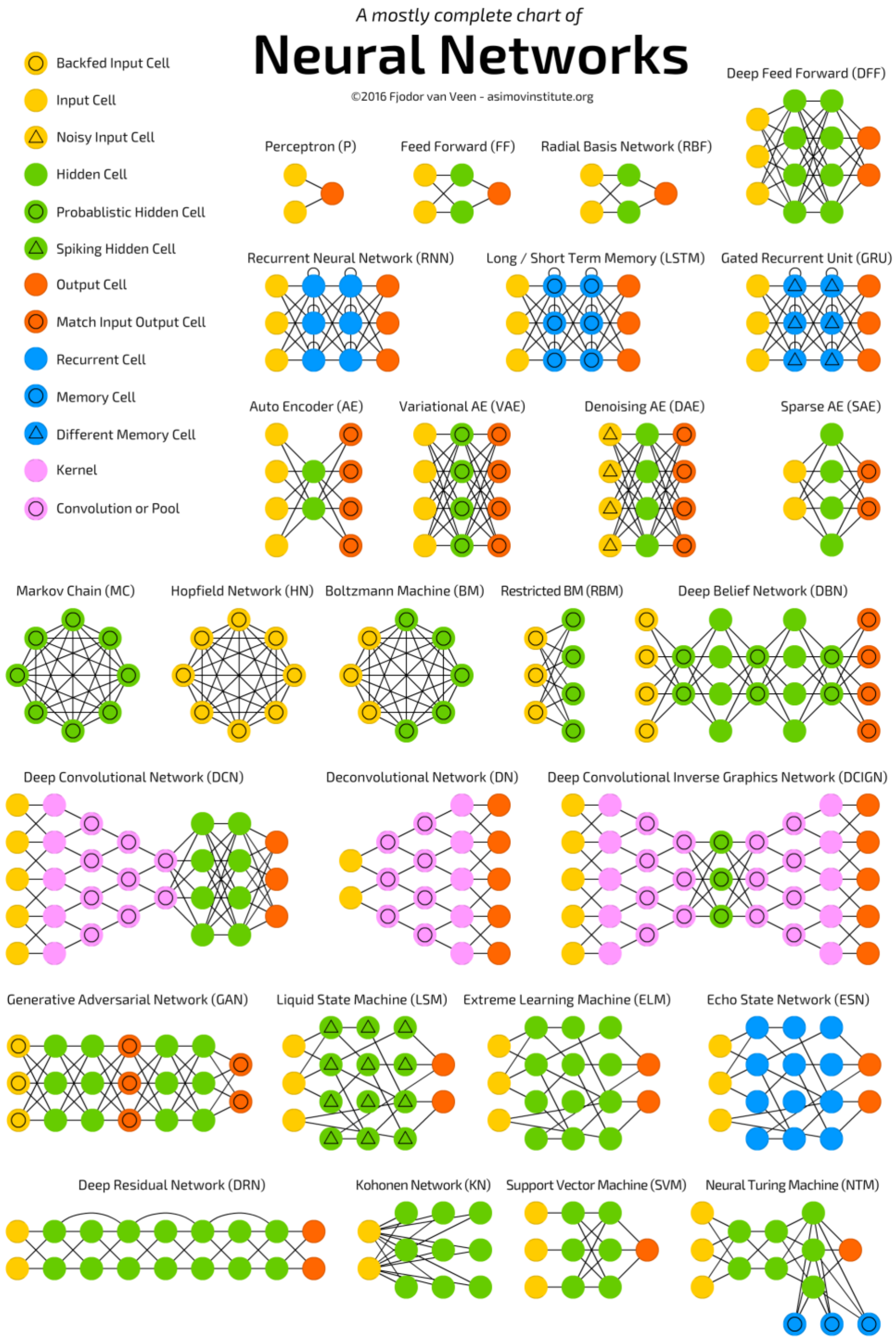


	Function
Logistic function	$f(x) = \frac{1}{1+e^{-x}}$
tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
cos	$\cos(x)$
Softmax[a)]	$\sigma(\mathbf{x})_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$
Rectified linear unit	$\mathrm{ReLU}(x) = \max\{0, x\}$
Exponential linear unit	$\mathrm{ELU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$
Softplus	$\mathrm{SP}(x) = \ln(e^x + 1)$

$$C(\Omega) := C(y(x_i), y_i) = \frac{1}{2N} \sum_{i=1}^{N} \| y(x_i) - y_i \|^2$$

$$w_{ij} \rightarrow w'_{ij} = w_{ij} - \frac{\eta}{N'} \sum_{i=1}^{N'} \frac{\partial C(X_i)}{\partial w_{ij}}$$

$$b_i \rightarrow b'_i = b_i - \frac{\eta}{N'} \sum_{i=1}^{N'} \frac{\partial C(X_i)}{\partial b_i}$$


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Quantum-inspired Soft Computing

- Quantum computing offers:
  - Massively increased computational efficiency
  - Potential for bridging brain and mind
  - Increasing relevance as computer technology develops into nanotechnology
  - Wave function, superposition (coherence), measurement (decoherence), entanglement, and unitary transformations

Quantum-inspired Soft Computing

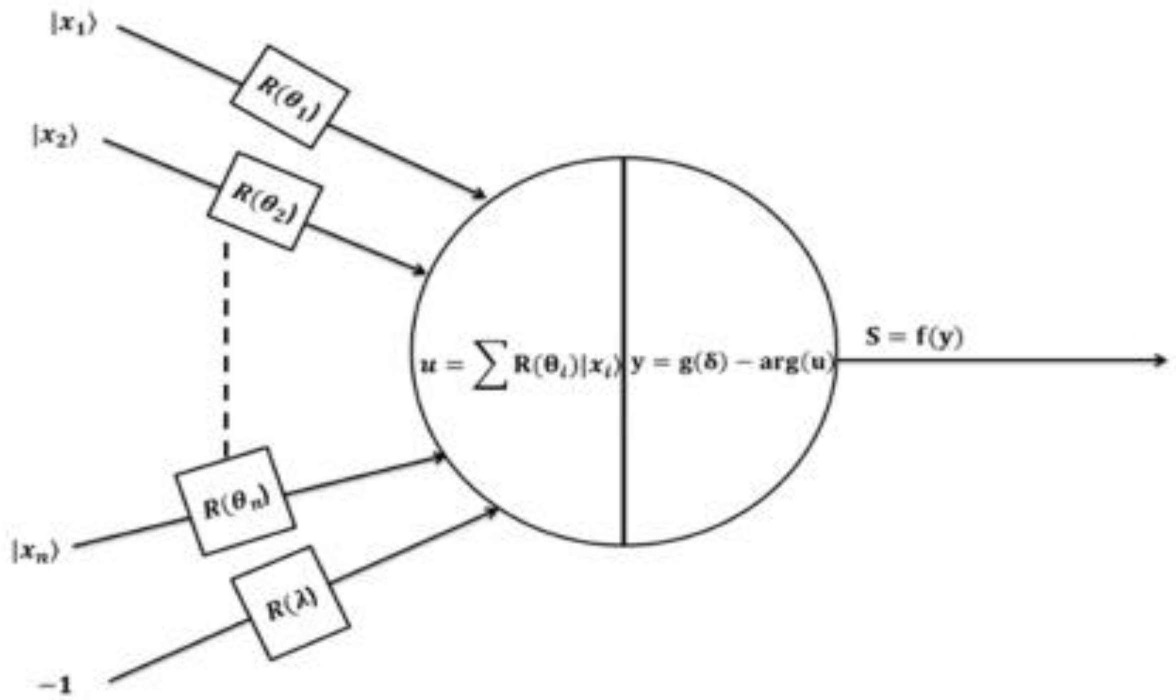
- Quantum computation is a linear theory
- ANN is, however, a non-linear approach involving
  - processing elements (neurons),
  - transformation performed by the elements (a nonlinear mapping),
  - interconnection structure,
  - network dynamics
    - learning rule that governs the modification of interaction strengths
  - massive parallel, distributed processing of information

Quantum-inspired Soft Computing

- In order to establish such correspondence is a major challenge in the development of a model of QNN (Quantum Neural Network)

- Many researchers use their own analogies in establishing a connection between quantum mechanics and neural networks (Kanabov 2000)

A Quantum Neuron Model



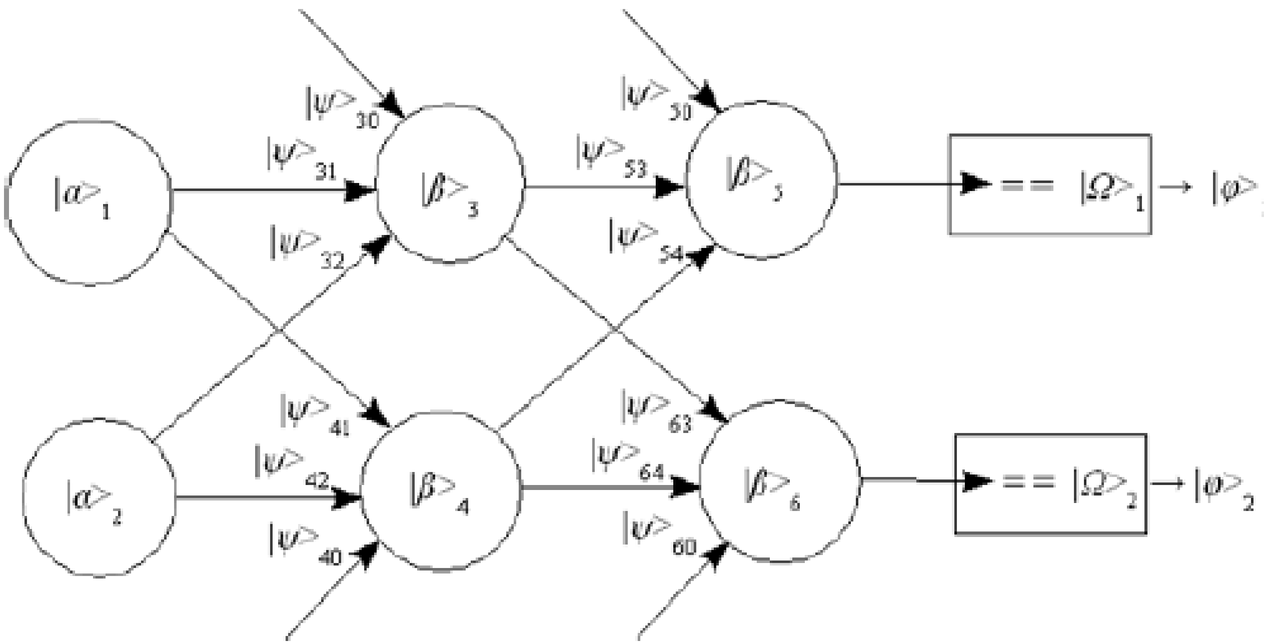
A **Q**uantum **N**euron (QN) [basic building block of a **Q**uantum **B**ackpropagation **N**eural **N**etwork (**QBPNN**)]

Principle of Operation

- Here, $|x_i\rangle$ are the input *qubits,* $R(\theta_i)$ are the rotation gates and λ is the input bias
- We define $f(x) = e^{ix}$ where, $i = \sqrt{-1}$
- Thus, $$u = \sum_{i=1}^{n} R(\theta_i)|x_i > -f(\lambda)$$
- Hence, $$y = \frac{\pi}{2} g(\delta) - \arg(u)$$

  where, $g(\delta) = \dfrac{1}{1 + e^{-\delta}}$ is the sigmoidal function
- Now $S = f(y) = e^{iy}$ is the neuron output

QBPNN Architecture



A Three Layer **Q**uantum **B**ackpropagation **N**eural **N**etwork (**QBPNN**)

QBPNN Architecture

- Analogous to the classical multilayer perceptron architecture
  - One input layer of QNs (represented by superposition state of *qubits*) which accepts inputs in the form of *qubits*
  - Several hidden layers each with varying number of QNs
  - One output layer of QNs
  - Connection weights are represented by single *qubit* rotation gates

Principle of Operation

- If the inputs are real values (x_i) scaled in the [0 . . . 1] range, then it can be transformed into *qubit* $|x_i\rangle$ and fed to the input layer as

$$|x_i > f(\frac{\pi}{2}x_i) = e^{i\frac{\pi}{2}x_i}$$

- If the neuron is an output layer neuron, the real output is evaluated as

$$O = |Im(y)|^2 = (\sin y)^2$$

Quantum BP Algorithm

- Here, in order to express quantum states, we connect the probability amplitude of $|0\rangle$ to the real part and that of $|1\rangle$ to the imaginary part
- We have a representation of the quantum state that uses complex numbers
- Three kinds of parameters exist in this neuron model: the phase parameter of the weight connection θ, the threshold λ, and the reversal parameter δ

Network Error

- The network error is represented as

$$E_{total} = \frac{1}{2}\sum_{p}^{P}\sum_{n}^{N}(t_{n,p} - output_{n,p})^2$$

- Here, P is the number of learning patterns. $t_{n,p}$ is a target signal for the n^{th} neuron, and $output_{n,p}$ means an $output_n$ at the p^{th} pattern. The error gradient is given by

$$\frac{\partial E}{\partial \theta_{jk}} = t_{n,p} - output_{n,p}\frac{\partial output}{\partial \theta_{jk}}$$

Network Error Adjustment

- Thus, the weight update equation takes the form

$$\theta_{jk}^{new} = \theta_{jk}^{old} - \eta \frac{\partial E}{\partial \theta_{jk}^{old}}$$

for hidden-output layer interconnections, where η is the learning coefficient

- Similarly,

$$\theta_{ij}^{new} = \theta_{ij}^{old} - \eta \frac{\partial E}{\partial \theta_{ij}^{old}}$$

for input-hidden layer interconnections

λ-δ Adjustment

$$\left.\begin{aligned} \lambda_k^{new} &= \lambda_k^{old} - \eta \frac{\partial E}{\partial \lambda_k^{old}} \\ \delta_k^{new} &= \delta_k^{old} - \eta \frac{\partial E}{\partial \delta_k^{old}} \end{aligned}\right\} \text{For output layer}$$

$$\left.\begin{aligned} \lambda_j^{new} &= \lambda_j^{old} - \eta \frac{\partial E}{\partial \lambda_j^{old}} \\ \delta_j^{new} &= \delta_j^{old} - \eta \frac{\partial E}{\partial \delta_j^{old}} \end{aligned}\right\} \text{For hidden layer}$$

Simulations

- The QBPNN architecture is compared with its classical counterpart - the classical MLP for the restoration of images from blurred and noisy perspectives
- A 49-8-1 architecture (49 input nodes, 8 hidden nodes, and 1 output node) is considered for each pixel in a 7×7 neighborhood, ensuring the best balance in terms of convergence ratio and signal-to-noise ratio (SNR)

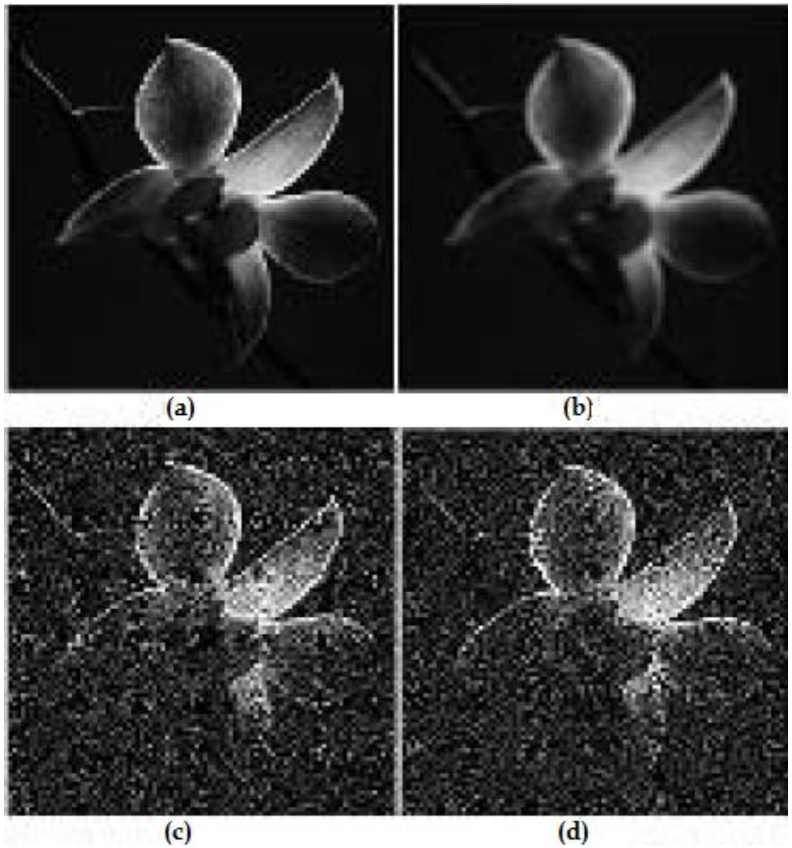
Simulations

- Two real-life gray-level images, viz., Lena and Tulip images, are considered.
- The training images are of 512×512 dimensions with 8-bit gray level quantization – a training set comprising 2,62,144 training patterns.
- Both networks are trained on two kinds of problems with varying parameters and a range of learning rates.
  - Image sharpening
  - Noise filtering (Gaussian and uniform)

Training Images

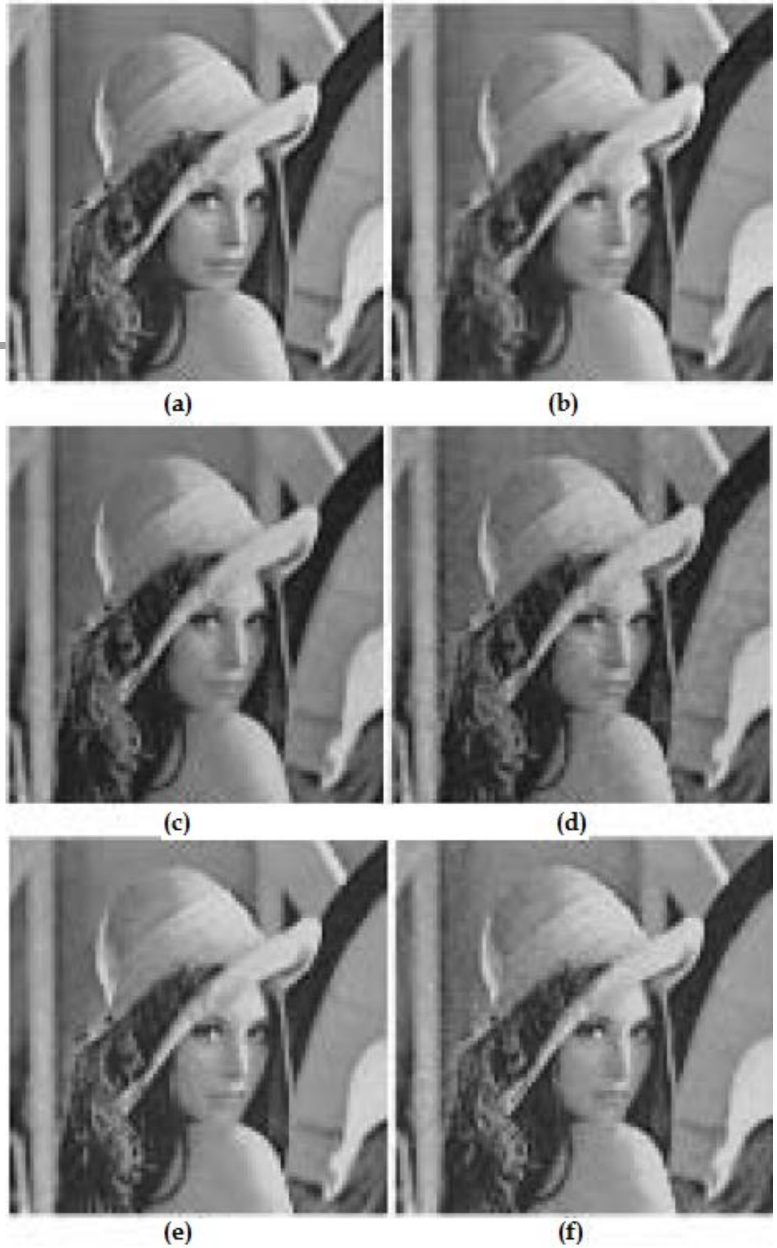
Training Lena images (a) original image (b) lens blur of radius=15 pixels (c) Gaussian noise (σ=16) affected image (d) uniform noise (σ =25%) affected image

Training Images



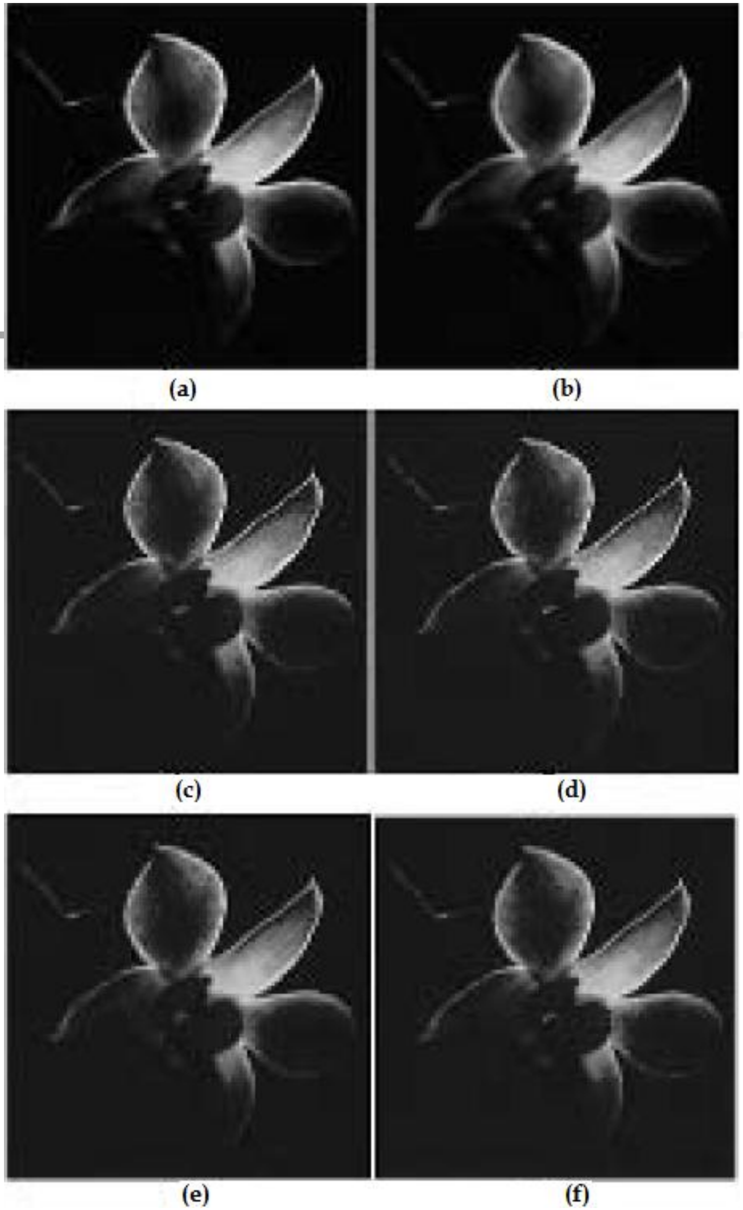
Training Tulip images (a) original image (b) lens blur of radius=15 pixels (c) Gaussian noise (σ=16) affected image (d) uniform noise (σ =25%) affected image

OUTPUT LENA IMAGES AFTER 5,000 EPOCHS



Filtered Lena images (a) QBPNN output for lens blur (b) MLP output for lens blur (c) QBPNN output for Gaussian noise (d) MLP output for Gaussian noise (e) QBPNN output for uniform noise (f) MLP output for uniform noise

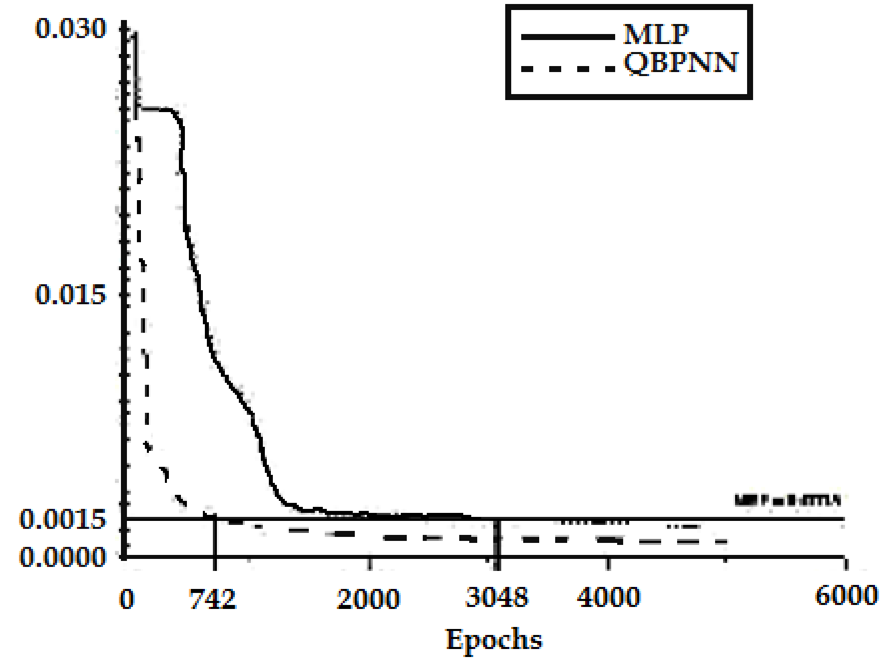
OUTPUT TULIP IMAGES AFTER 5,000 EPOCHS



Tulip Lena images (a) QBPNN output for lens blur (b) MLP output for lens blur (c) QBPNN output for Gaussian noise (d) MLP output for Gaussian noise (e) QBPNN output for uniform noise (f) MLP output for uniform noise

Network Convergence

- The convergence conditions of the networks are decided based on MSE and convergence ratio
- The convergence ratio of both networks was established for 50 trials for each parameter
- To adjudge the efficacy of the networks, it has been decided that the MSE of the output has to be less than 0.0015 in less than 5,000 epochs
- It was found that the MLP had the highest convergence ratio (~0.92) for learning rates in the neighborhood of 0.025, while the QBPNN had the highest convergence ratio (~0.98) for learning rates in the neighborhood of 0.06
- QBPNN converges faster (in 742 epochs) than MLP (3,048 epochs)



Convergence curves of QBPNN and MLP

Output Image Quality

- The quality of the filtered images is determined by the Signal to Noise Ratio (SNR)
- Table I lists the attained SNR values after 5000 epochs by the QBPNN and MLP for the Lena and Tulip images.
- These results clearly demonstrate that the QBPNN has better generalization capabilities than the MLP.

Table I SNR values of Images after 5000 epochs

Input Image	MLP O/P SNR (DB)	QBPNN O/P SNR (DB)
Fig. 3(b)	20.103	22.364
Fig. 3(c)	23.897	24.280
Fig. 3(d)	19.293	20.373
Fig. 4(b)	22.860	23.600
Fig. 4(c)	25.030	26.490
Fig. 4(d)	25.293	26.370

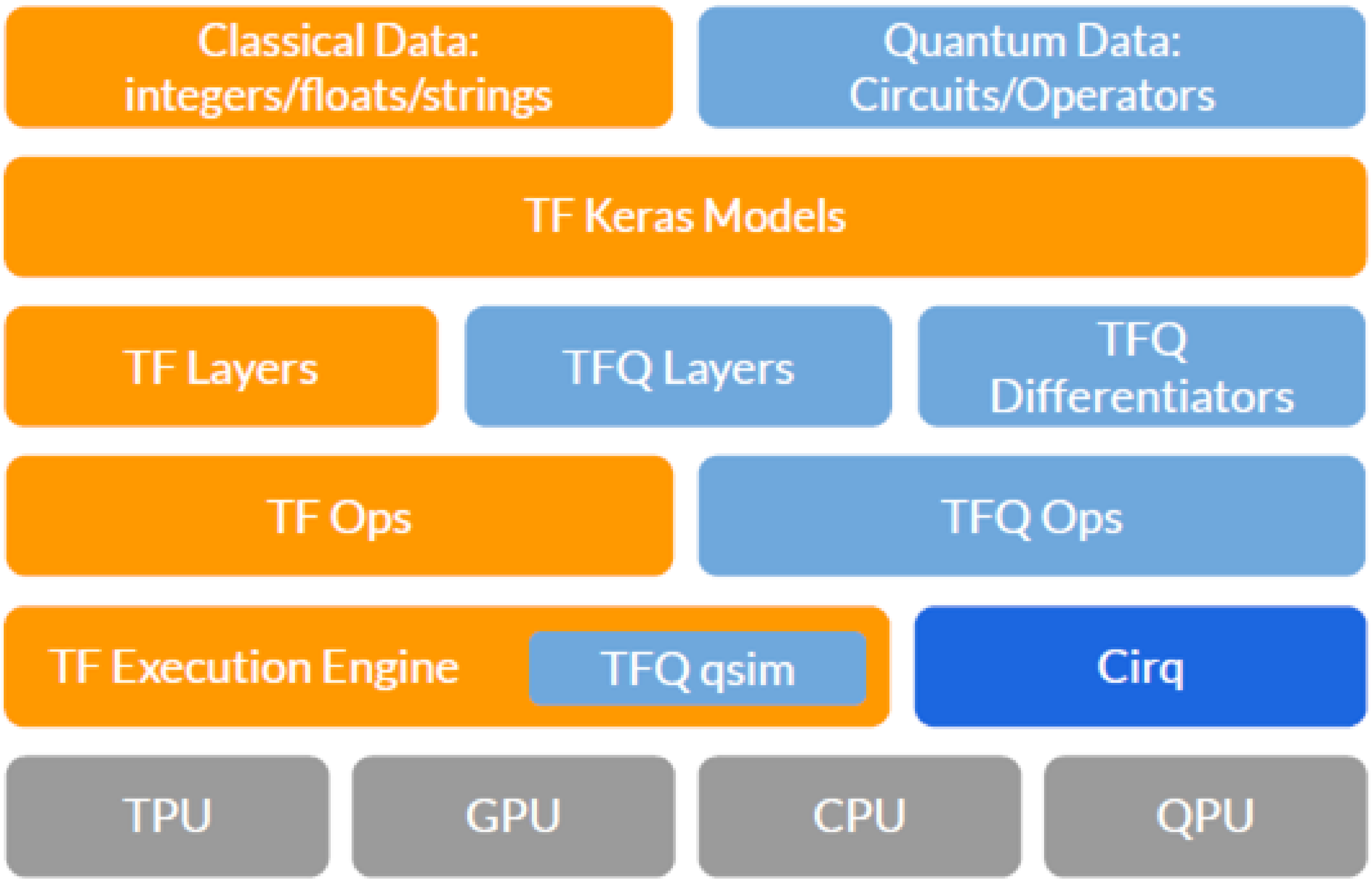
Remarks

- A generalized QBPNN model for restoration of images from blurred and noisy perspectives is presented
- Results demonstrate the efficacy of QBPNN over its classical counterpart in terms of convergence ratio and convergence speed

TensorFlow Quantum

- TensorFlow Quantum (TFQ) is a Python framework for quantum machine learning
- As an application framework, TFQ allows quantum algorithm researchers and ML application researchers to leverage Google's quantum computing frameworks, all from within TensorFlow
- TensorFlow Quantum focuses on *quantum data* and building *hybrid quantum-classical models*
- It provides tools to interleave quantum algorithms and logic designed with TensorFlow

TensorFlow Quantum Framework



TensorFlow is Based on Cirq

- As mentioned previously Cirq is an open-source framework for invoking quantum circuits on near term devices
- It contains the basic structures, such as qubits, gates, circuits, and measurement operators, that are required for specifying quantum computations
- User-specified quantum computations can then be executed in simulation or on real hardware
- Cirq also contains substantial machinery that helps users design efficient algorithms for Noisy, Intermediate Scale, Quantum (NISQ) machines, such as compilers and schedulers

An Open-source Framework for Programming QC

- Cirq is a Python software library for writing, manipulating, and optimizing quantum circuits, and then running them on quantum computers and quantum simulators
- Cirq provides useful abstractions for dealing with today's noisy intermediate-scale quantum computers, where details of the hardware are vital to achieving state-of-the-art results

Cirq

https://quantumai.google/cirq

Cirq Code Example

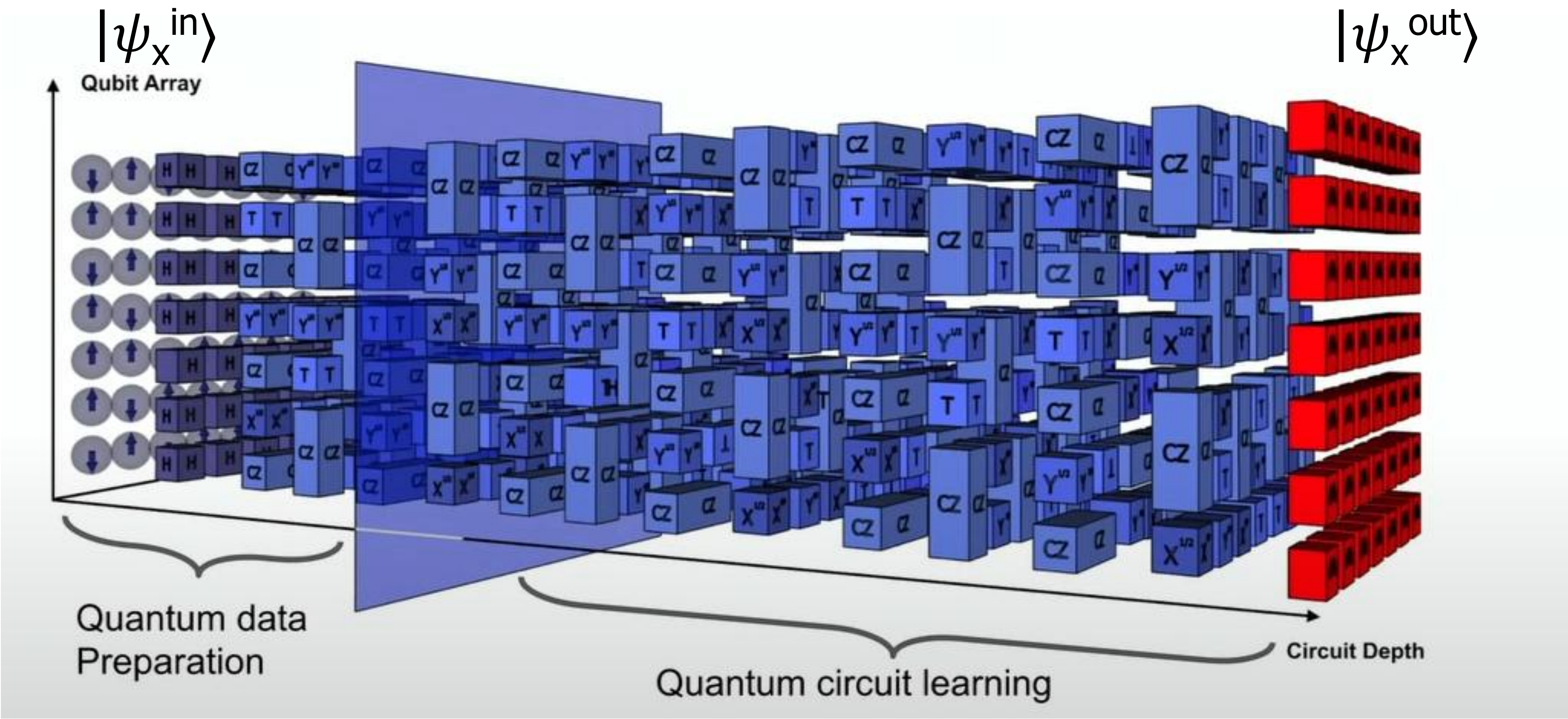
- import cirq

- # Pick a qubit.
- qubit = cirq.GridQubit(0, 0)

- # Create a circuit
- circuit = cirq.Circuit(
-     cirq.X(qubit)**0.5, # Square root of NOT.
-     cirq.measure(qubit, key='m') # Measurement.
-)
- print("Circuit:")
- print(circuit)

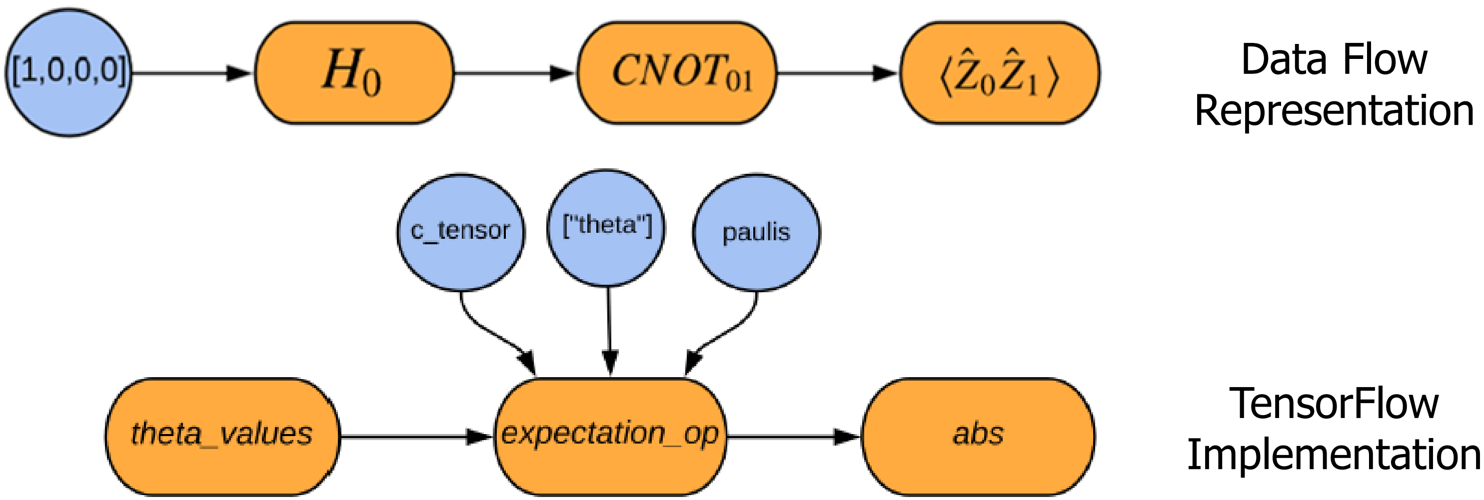
- # Simulate the circuit several times.
- simulator = cirq.Simulator()
- result = simulator.run(circuit, repetitions=20)
- print("Results:")
- print(result)

Parameterized Quantum Circuits (PQCs) or Quantum Neural Networks (QNNs)



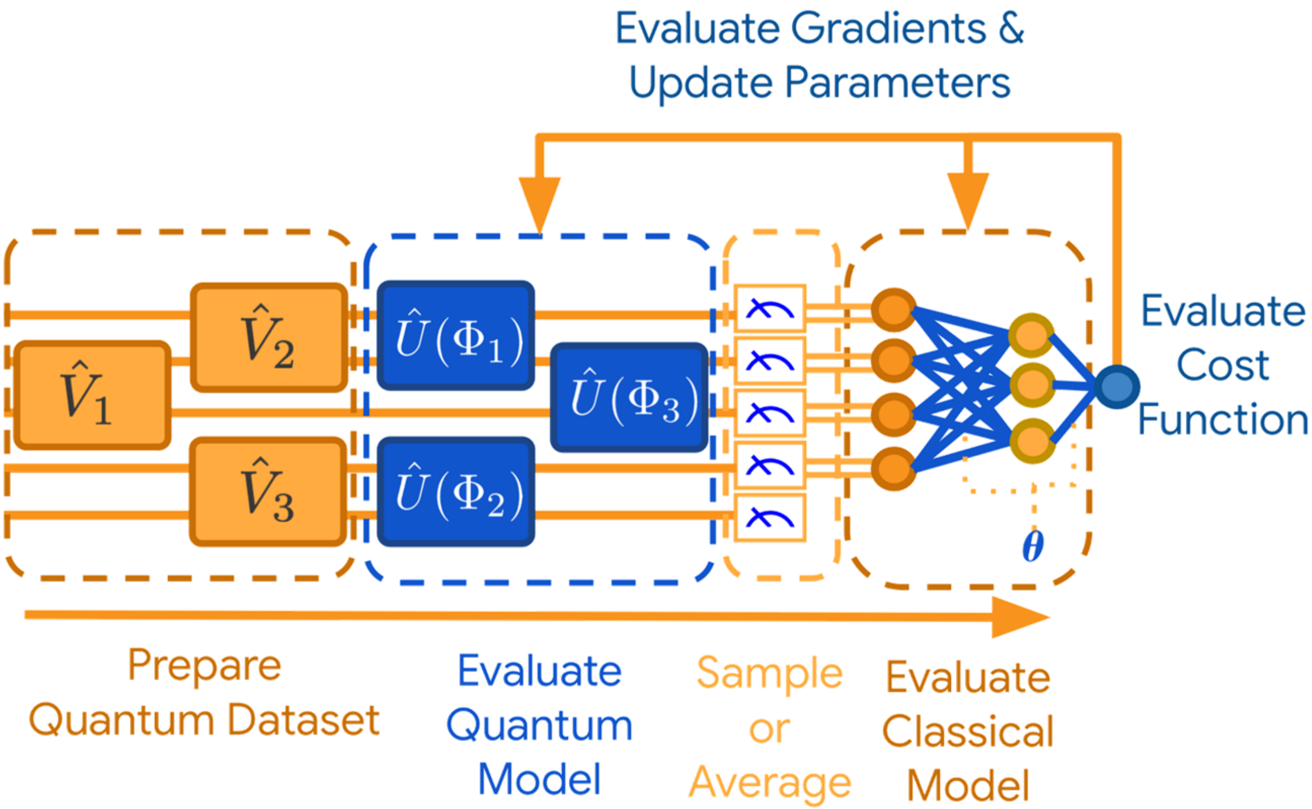
Beer, K., Bondarenko, D., Farrelly, T. et al. Training deep quantum neural networks. Nature Communication 11, 808 (2020).

TensorFlow Version of Bell State Preparation and Measurement



- Quantum Computations as Tensors (tfq.convert_to_tensor)
- Composing Quantum Models (tfq.layers.AddCircuit)
- Sampling and Expectation Values (tfq.layers.Sample)
- Differentiating Quantum Circuits (tfq.differentiators.Differentiator)
- Gate Fusion with qsim

QNN Training Model Used in TQF



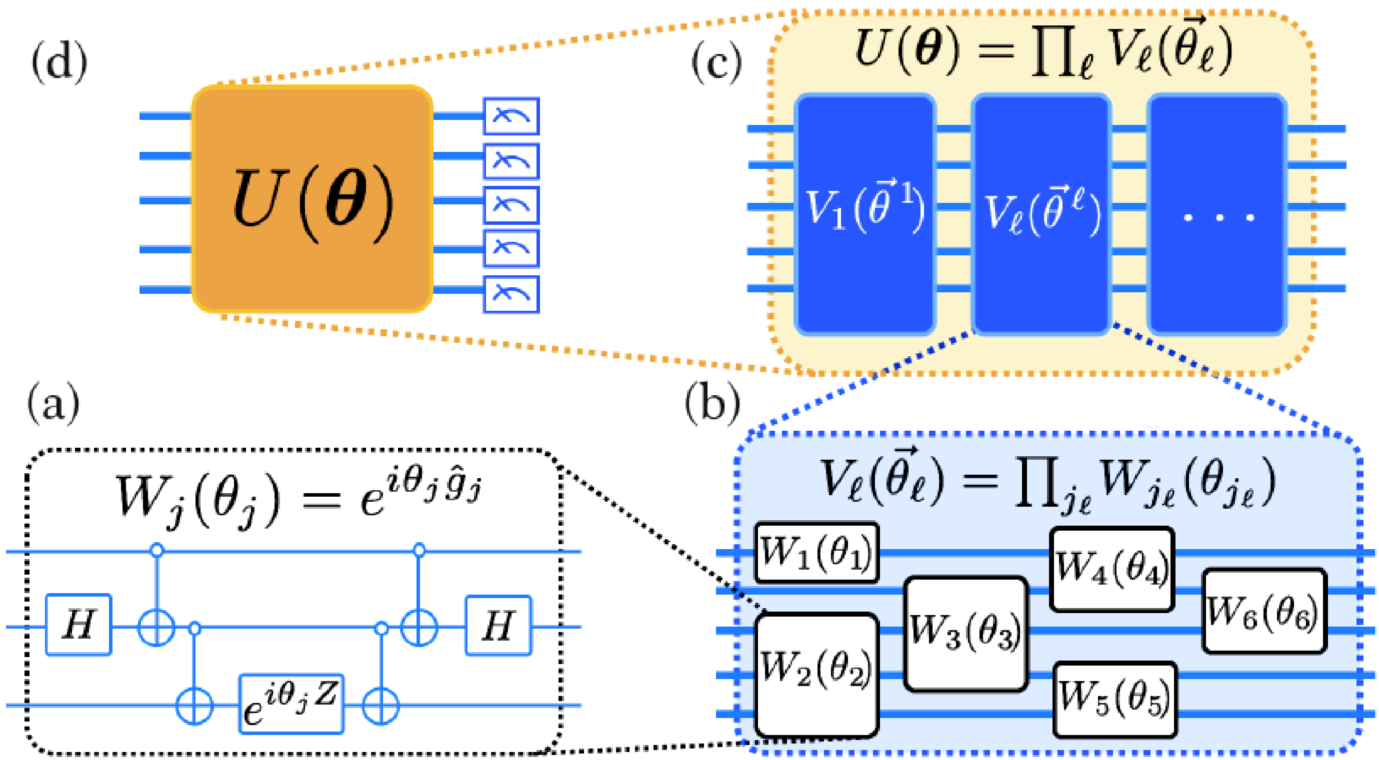
Φ_n represents the quantum model parameters and θ represents the classical model parameters

QNN Training

1. Prepare Quantum Dataset
2. Evaluate Quantum Model
3. Sample or Average
4. Evaluate Classical Model
5. Evaluate Cost Function
6. Evaluate Gradients & Update Parameters

To support gradient descent, TFQ exposes derivatives of quantum operations to the TensorFlow backpropagation machinery via the *tfq.differentiators.Differentiator* interface

High-level Multilayer Parameterized Quantum Circuits



Broughton et al., TensorFlow Quantum: A Software Framework for Quantum Machine Learning, March 9, 2020

PQC Model -I

A Quantum Neural Network can generally be written as a product of layers of unitary matrix in the form:

$$\hat{U}(\boldsymbol{\theta}) = \prod_{\ell=1}^{L} \hat{V}^{\ell} \hat{U}^{\ell}(\boldsymbol{\theta}^{\ell}),$$

where the ℓ^{th} layer of the QNN consists of the product of $\hat{V}^{\ell}$, a non-parametric unitary, and $\hat{U}^{\ell}(\boldsymbol{\theta}^{\ell})$ a unitary with variational parameters (note the superscripts here represent indices rather than exponents).

PQC Model -II

The multiparameter unitary matrices of a given layer can also be represented as a global unitary matrix

$$\hat{U}^{\ell}(\boldsymbol{\theta}^{\ell}) \equiv \bigotimes_{j=1}^{M_\ell} \hat{U}_j^{\ell}(\theta_j^{\ell}).$$

Finally, each of these unitaries $\hat{U}_j^{\ell}$ can be expressed as the exponential of some generator $\hat{g}_{j\ell}$, which itself can be any Hermitian operator on n qubits (thus expressible as a linear combination of n-qubit Pauli's),

$$\hat{U}_j^{\ell}(\theta_j^{\ell}) = e^{-i\theta_j^{\ell}\hat{g}_j^{\ell}}, \quad \boxed{\hat{g}_j^{\ell} = \sum_{k=1}^{K_{j\ell}} \beta_k^{j\ell}\hat{P}_k,}$$

PQC Model -III

One can simply decompose the unitary into a product of exponentials of each term

$$\hat{U}_j^{\ell}(\theta_j^{\ell}) = \prod_k e^{-i\theta_j^{\ell}\beta_k^{j\ell}\hat{P}_k}. \quad (5)$$

$$\boxed{\hat{U}_j^{\ell}(\theta_j^{\ell}) = \prod_k \left[\cos(\theta_j^{\ell}\beta_k^{j\ell})\hat{I} - i\sin(\theta_j^{\ell}\beta_k^{j\ell})\hat{P}_k\right]}$$

Sampling and Expectations

To optimize the parameters, we need a cost function to optimize. In the case of standard variational quantum algorithms, this cost function is most often chosen to be the expectation value of a cost Hamiltonian

$$f(\boldsymbol{\theta}) = \langle \hat{H} \rangle_{\boldsymbol{\theta}} \equiv \langle \Psi_0 | \hat{U}^\dagger(\boldsymbol{\theta}) \hat{H} \hat{U}(\boldsymbol{\theta}) | \Psi_0 \rangle \qquad (6)$$

where $|\Psi_0\rangle$ is the input state to the parameterized circuit.

In general, the cost Hamiltonian can be expressed as a linear combination of operators, e.g. in the form

$$\hat{H} = \sum_{k=1}^{N} \alpha_k \hat{h}_k \equiv \boldsymbol{\alpha} \cdot \hat{\boldsymbol{h}},$$

Sampling and Expectations

$$f(\boldsymbol{\theta}) = \langle \hat{H} \rangle_{\boldsymbol{\theta}} = \sum_{k=1}^{N} \alpha_k \langle \hat{h}_k \rangle_{\boldsymbol{\theta}} \equiv \boldsymbol{\alpha} \cdot \boldsymbol{h}_{\boldsymbol{\theta}}$$

where we introduced the vector of expectations $\boldsymbol{h}_{\boldsymbol{\theta}} \equiv \langle \hat{\boldsymbol{h}} \rangle_{\boldsymbol{\theta}}$. In the case of non-commuting terms, the various expectation values $\langle \hat{h}_k \rangle_{\boldsymbol{\theta}}$ are estimated over separate runs.

Gradients of Quantum Neural Networks

For a parameter of interest θ_j^ℓ appearing in a layer ℓ in a parametric circuit in the form (5), consider the change of variables $\eta_k^{j\ell} \equiv \theta_j^\ell \beta_k^{j\ell}$, then from the chain rule of calculus [90], we have

$$\frac{\partial f}{\partial \theta_j^\ell} = \sum_k \frac{\partial f}{\partial \eta_k^{j\ell}} \frac{\partial \eta_k^{j\ell}}{\partial \theta_j^\ell} = \sum_k \beta_k^{j\ell} \frac{\partial f}{\partial \eta_k}. \tag{11}$$

Gradient Using Parameter Shift Methods

As can be shown from this form, the analytic derivative of the expectation value:

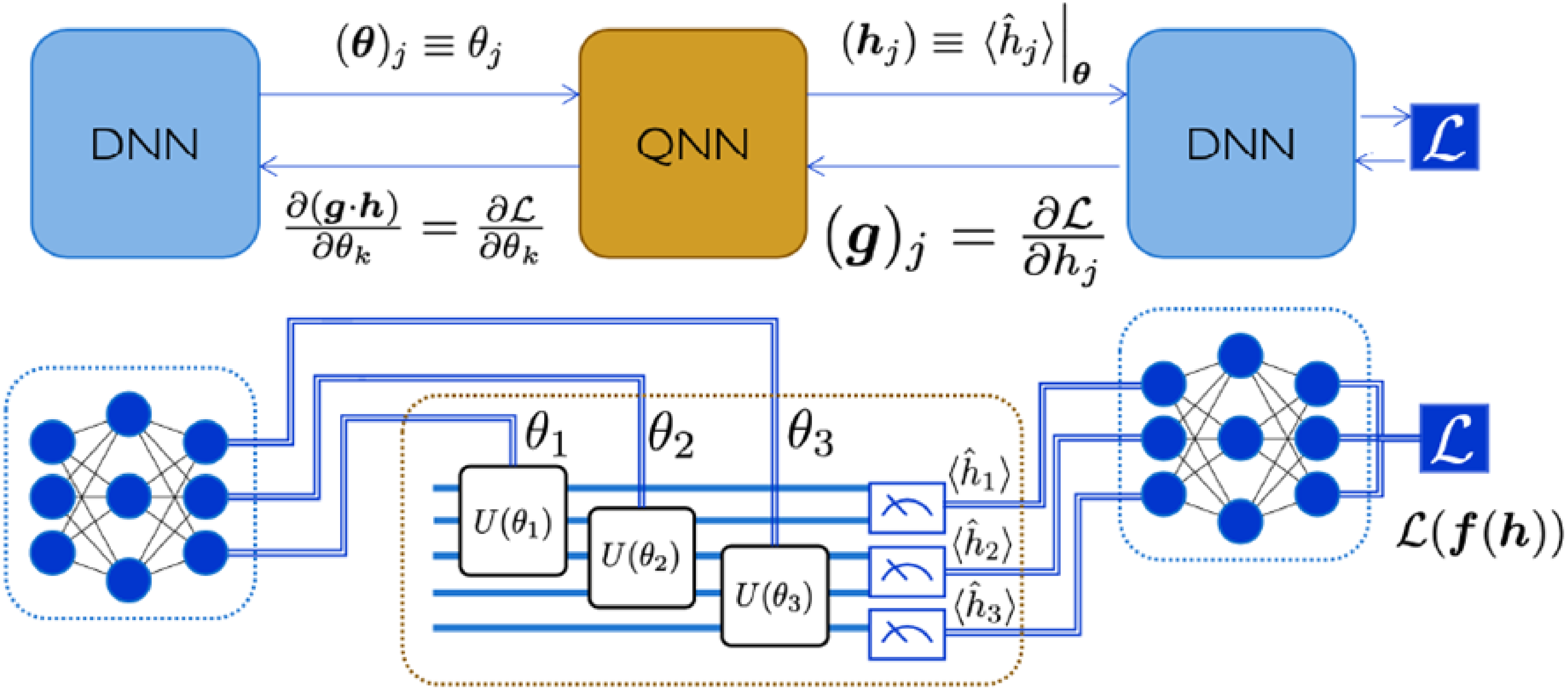
$$f(\boldsymbol{\eta}) \equiv \langle\Psi_0|\hat{U}^\dagger(\boldsymbol{\eta})\hat{H}\hat{U}(\boldsymbol{\eta})|\Psi_0\rangle$$

is:

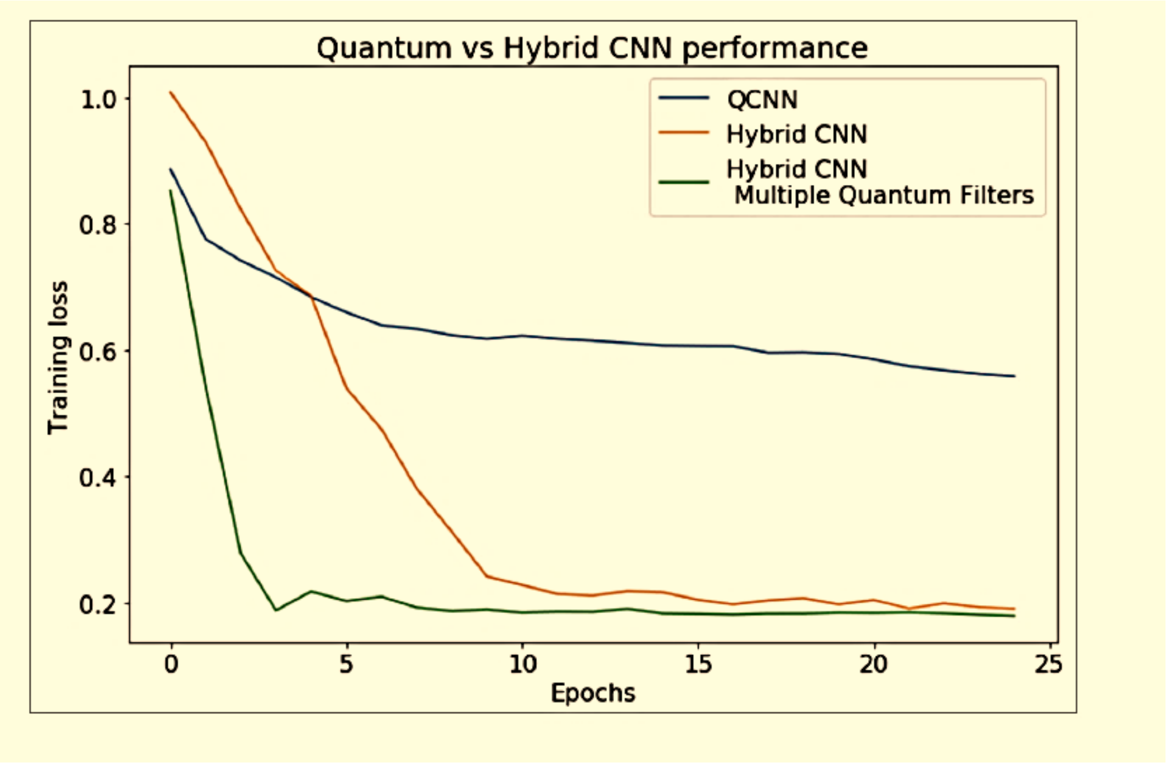
$$\frac{\partial}{\partial\eta_k^{j\ell}} f(\boldsymbol{\eta}) = f(\boldsymbol{\eta} + \tfrac{\pi}{4}\boldsymbol{\Delta}_k^{j\ell}) - f(\boldsymbol{\eta} - \tfrac{\pi}{4}\boldsymbol{\Delta}_k^{j\ell})$$

where $\boldsymbol{\Delta}_k^{j\ell}$ is a vector representing unit-norm perturbation of the variable $\eta_k^{j\ell}$ in the positive direction.

Example of Inference and Hybrid Backpropagation

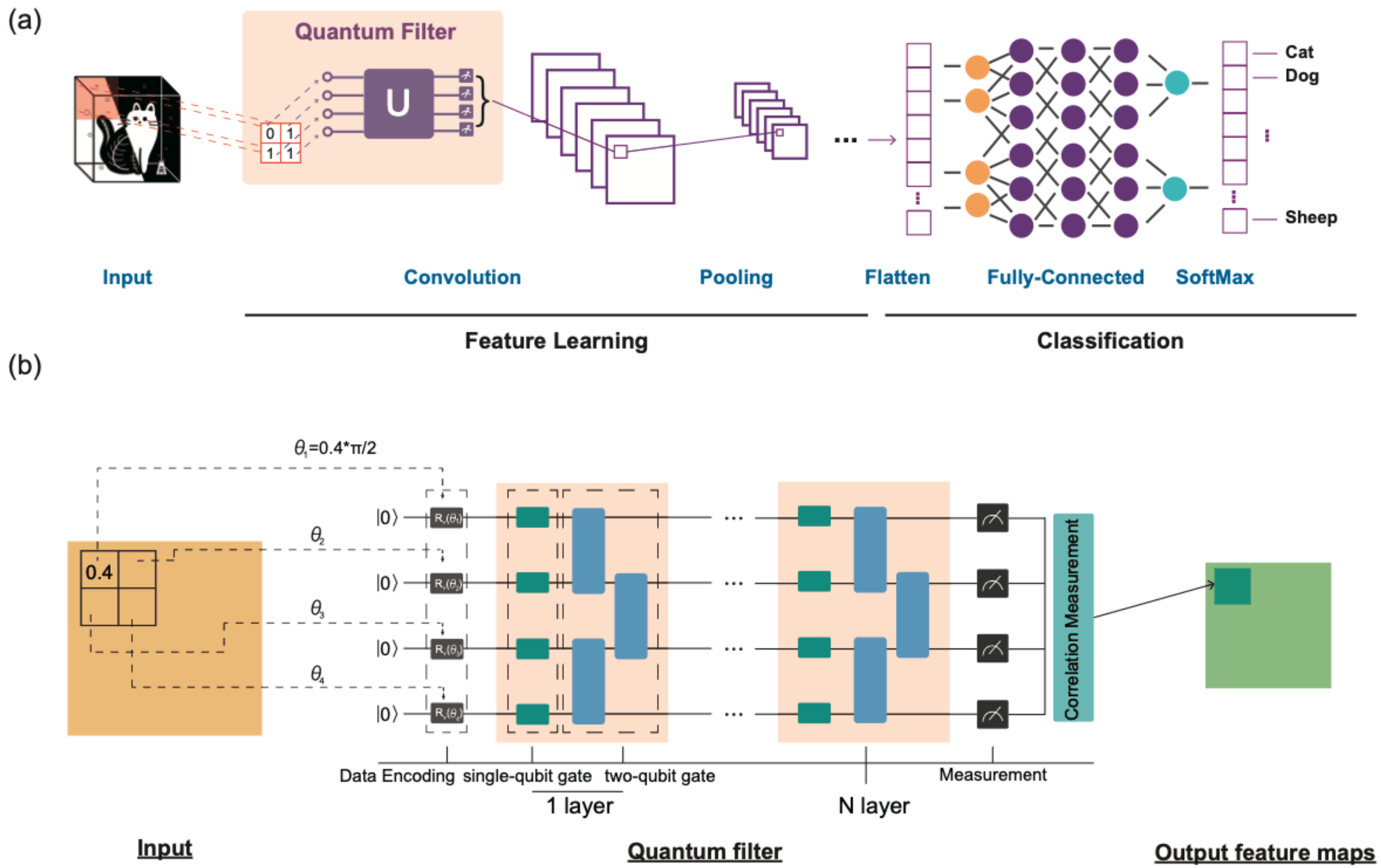


QNN vs Hybrid CNN

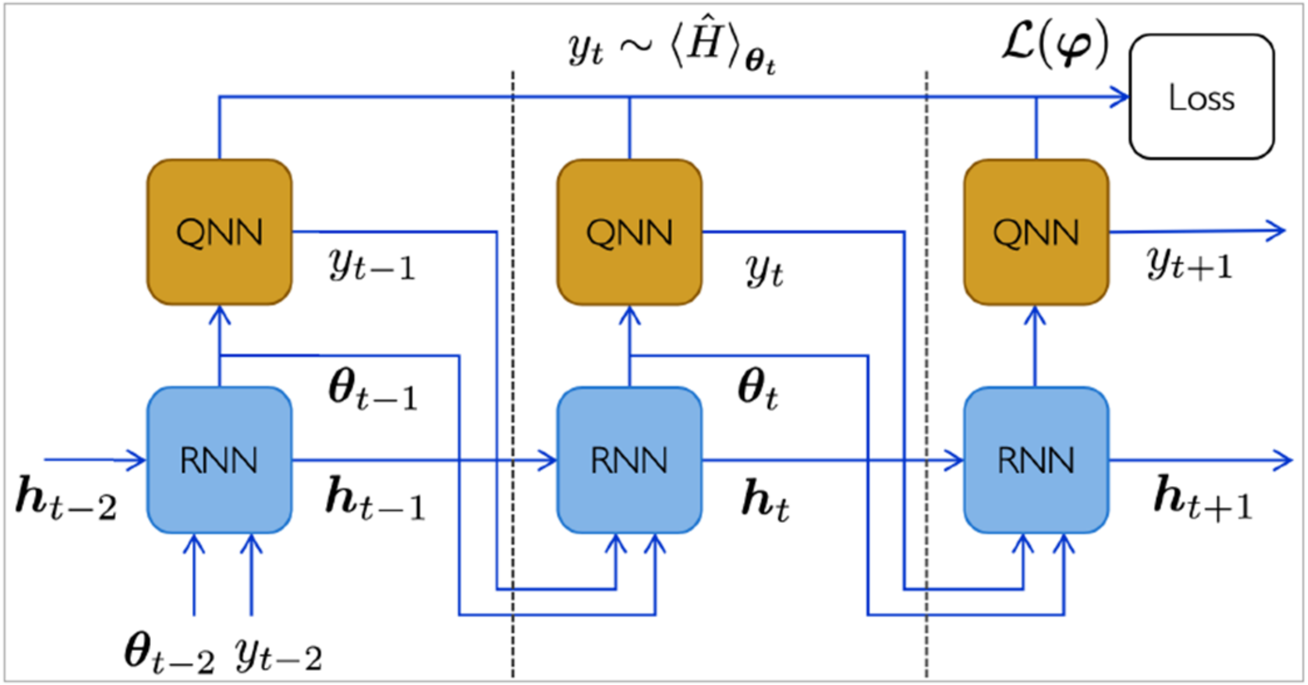


Mean squared error loss as a function of training epoch for three different hybrid classifiers. We find that the purely quantum classifier trains the slowest, while the hybrid architecture with multiple quantum filters trains the fastest

Quantum CNN for Images



Optimization of Variational Quantum Using RNN



https://pennylane.ai/qml/demos/learning2learn.html

Optimization of Variational Quantum Using RNN

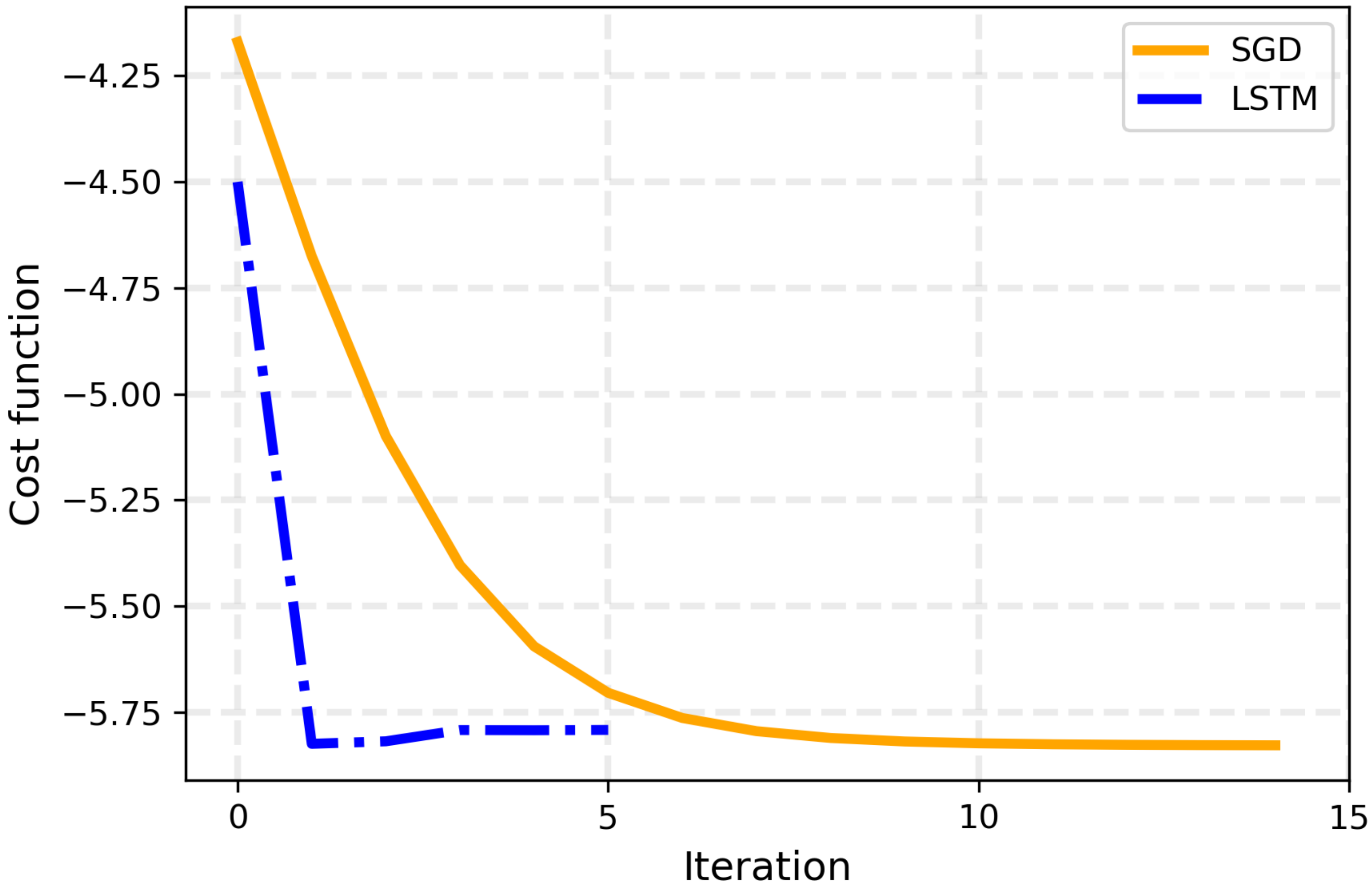
- Given parameters $\boldsymbol{\theta}_{t-1}$ of the variational quantum circuit, the cost function y_{t-1} , and the hidden state of the classical network $\mathbf{h}_{t-1}$ at the previous time step, the recurrent neural network proposes a new guess for the parameters $\boldsymbol{\theta}_t$, which are then fed into the quantum computer to evaluate the cost function y_t
- By repeating this cycle a few times, and by training the weights of the recurrent neural network to minimize the loss function y_t , a good initialization heuristic is found for the parameters $\boldsymbol{\theta}$ of the variational quantum circuit

- At a given iteration, the RNN receives as input the previous cost function y_t evaluated on the quantum computer, where y_t is the estimate of $\langle \mathbf{H} \rangle_t$, as well as the parameters $\boldsymbol{\theta}_t$ for which the variational circuit was evaluated
- The RNN at this time step also receives information stored in its internal hidden state from the previous time step $\mathbf{h}_t$
- The RNN itself has trainable parameters ϕ , and hence it applies the parametrized mapping:

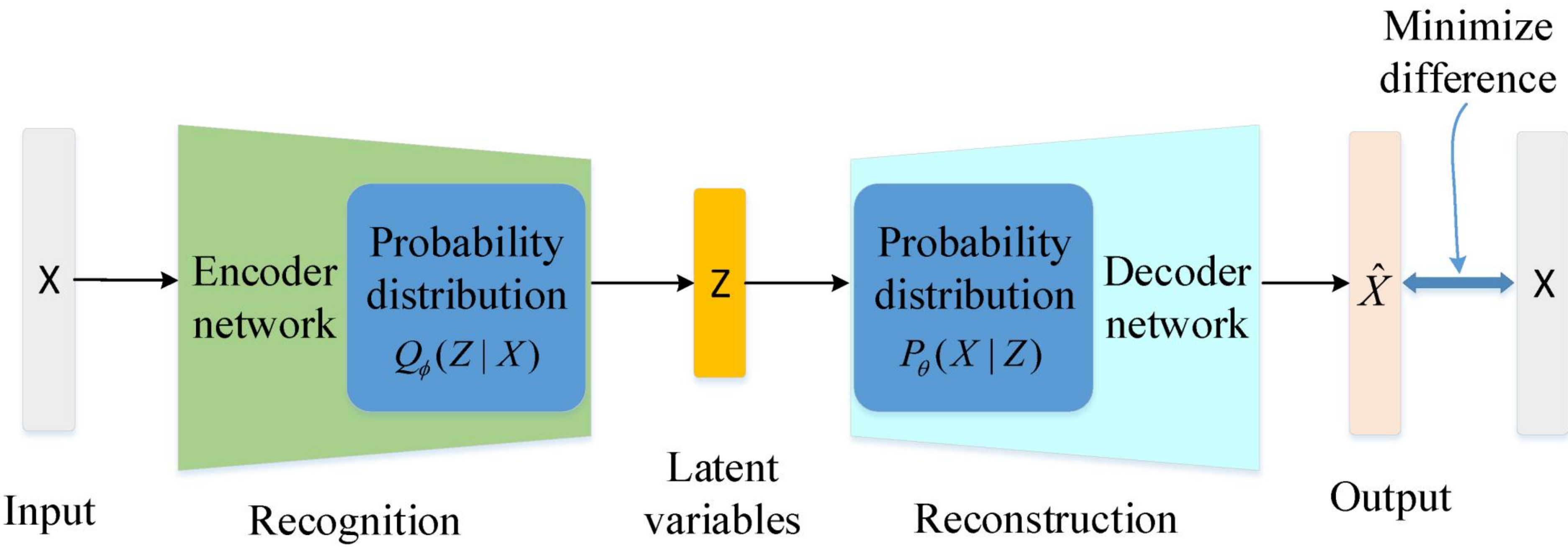
$$\boldsymbol{h}_{t+1}, \boldsymbol{\theta}_{t+1} = \mathrm{RNN}_{\phi}(\boldsymbol{h}_t, \boldsymbol{\theta}_t, y_t),$$

- which generates a new suggestion for the variational parameters as well as a new internal state. Upon training the weights ϕ the RNN eventually learns a good heuristic to suggest optimal parameters for the quantum circuit.

Quantum Optimization vs standard Stochastic Gradient Descent (SGD)

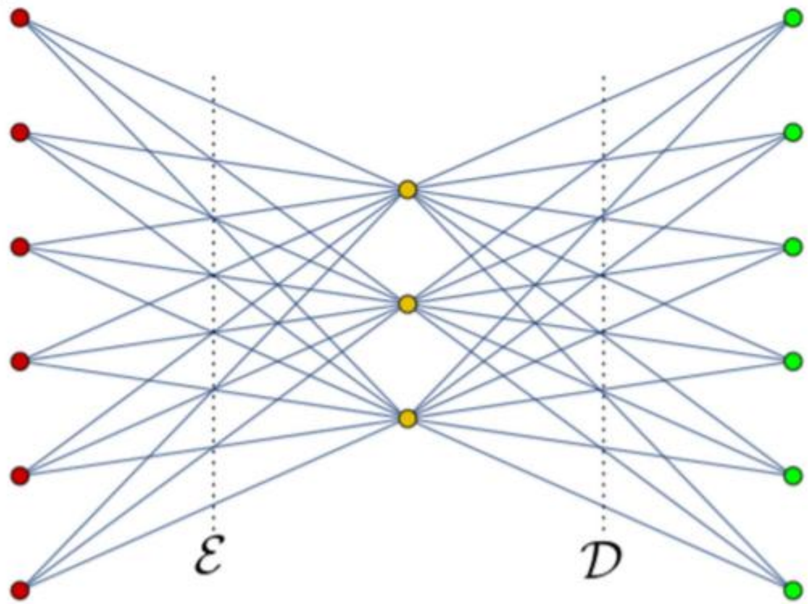


Classical Auto-Encoder

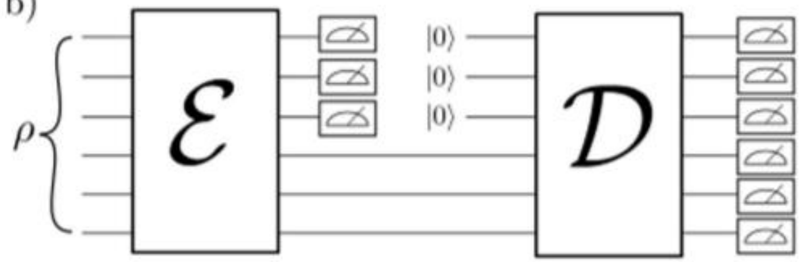


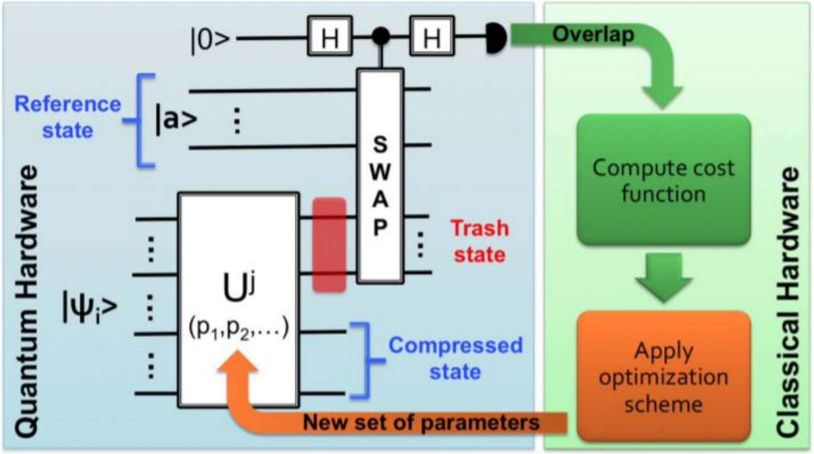
Quantum Variational Autoencoder

a)



b)





Other QNN Models

- Quantum M-P Neural Network
- Quantum Competitive Neural Network
- Quantum Dot Neural Network
- Quantum Cellular Neural Network
- Qubit Neural Network
- Quantum Associative Neural Network

Remarks

- QNNs demonstrates remarkable capabilities including:
  - The ability to generalize
  - Tolerance to noisy training data
  - An absence of a barren plateau in the cost function landscape

Advantages of QNN

- Compared to classical neural networks, quantum neural networks have the following advantages:
  - Exponential memory capacity
  - Higher performance for lower number of hidden neurons
  - Faster learning
  - Single layer network solutions of linearly inseparable problems
  - Faster processing speed
- **These advantages of quantum neural networks are indeed compelling**

Important Reads

[1] R. C. Gonzales and R. E. Woods, *Digital Image Processing*, 2nd ed., pp. 220-278, Prentice Hall, 2002.

[2] L. J. Spreeuwers, “Image Filtering with Neural Networks: Applications and Performance Evaluation”, PhD Thesis, University of Twente, ISBN 90-9005555-X, 1992.

[3] S. Bhattacharyya, P. Dutta, and U. Maulik, “Binary object extraction using bi-directional self-organizing neural network (BDSONN) architecture with fuzzy context sensitive thresholding”, *International Journal of Pattern Analysis and Applications*, Springer-Verlag, London, pp. 345-360, 2007.

[4] P. W. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring”, in *Proc. 35th Annual Symp. Foundations of Computer Science*, New Mexico: IEEE Computer Society Press, pp. 124-134, 1994.

[5] L. K. Grover, “A fast quantum mechanical algorithm for database search”, in *Proc. 28th Annual ACM Symp. Theory of Computing*, Pennsylvania, pp. 212-221, 1996.

[6] R. Penrose, *The Emperor's New Mind*, UK: Oxford University Press, 1999.

[7] N. Matsui, N. Kouda, and H. Nishimura, “Neural networks based on QBP and its performance”, in *Proc. IEEE-INNS-ENNS Int'l Jt. Conf. Neural Networks*, pp. 247-252, 2000.

Important Reads

[8] L. Panchi and L. Shiyong, “Learning algorithm and application of quantum BP neural networks based on universal quantum gates,” *Journal of Systems Engineering and Electronics*, vol. 19, no. 1, pp. 167-174, 2008.

[9] J. S. Bell, *Speakable and Unspeakable in Quantum Mechanics*, UK: Cambridge University Press, 1987.

[10] D. McMahon, *Quantum Computing Explained*, Wiley Interscience, 2008.

[11] D. Ventura and T. Martinez, “Quantum Associative Memory with Exponential Capacity”, Proc. *Int'l Jt. Conf. Neural Networks*, pp. 509-513, May 1998.

[12] J. L. Mitrpanont and A. Srisuphab, “The realization of quantum complex-valued backpropagation neural network for pattern recognition problem”, in *Proc. 9th Int'l Conf. Neural Information Processing (ICONIPO2)*, vol. 1, pp. 462-466, 2002.

[13] C. R. B. Azevedo and T. A. E. Ferreira, “The Application of Qubit Neural Networks for Time Series Forecasting with Automatic Phase Adjustment Mechanism”, *Anais do XXVII Congresso da SBC*, pp.11121121, 2007.

[14] F. Rosenblatt, *Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms*, Washington DC: Spartan Books, 1961.

[15] N. Kouda, N. Matsui, and N. Haruhiko, “Image compression by layered quantum neural networks”, *Neural Processing Letters*, vol. 16, pp. 67-80, 2002.

Thank you