Quantum Generalization of Natural Gradient Descent

Aradhita Sharma
EEE 606 Final Report
Electrical Engineering Department
Oncampus, MS, Arizona State University
ashar314@asu.edu

***Abstract*—The Gradient Descent optimization algorithm is used to find a function's minima. This algorithm is commonly used to train machine learning models and neural networks. The quantum generalization of this algorithm (Quantum Gradient Descent) provides the benefit of exponential speedup for problems with high-dimensional inputs. Quantum Geometric Tensor (QGT) is used to determine the optimization dynamics with respect to Quantum Information Geometry. This paper presents a comparison of gradient descent optimization performance in the classical and quantum domain. Furthermore, binary classification using neural network and quantum variational classifier are performed to compare the two approaches.**
Keywords— Gradient descent, optimization, machine learning, quantum generalization, Quantum gradient descent, exponential speedup, quantum geometric tensor (QGT), quantum information theory, binary classification, quantum variational classifier.

I. Introduction

Gradient descent is the most popular adaptive optimization algorithm used in machine learning and deep learning technology [2]. Gradient descent is the core foundation of deep neural network. It considers the first-order derivative to adapt the parameters of the system that minimize the error between predicted functions and empirical data. Various methods have been presented in the domain of gradient descent optimization such as RMSprop and Adam to enhance the neural network training performance. However, training massive datasets takes long training time per iteration, which is not time efficient. Therefore, research is being done to look out for ways to make the training time efficient [10].

Nowadays, quantum computation has become an emerging technology to integrate within machine learning to achieve quantum advantage [3]. It refers to the computational advantage that quantum computation can achieve for certain problems which are too complex to be solved by classical computation. Numerous quantum computing implementation approaches have taken optimization into account. Quantum circuits are being constructed for obtaining a quantum minimum searching algorithm and a quantum approximate optimization algorithm [4], [5]. The literature on variational quantum circuits has presented a number of optimization techniques for obtaining the best variational parameters, including derivative-free (zeroth-order) techniques like Nelder-Mead, finite-differencing [6], and SPSA [7]. Work has been done to evaluate the first-order derivative quantum algorithms.

In this work, an attempt is being made to develop a quantum algorithm for replicating the iterative optimization of gradient descent in quantum computing where the effect of geometry on the dynamics of variational algorithms has been examined [8]. The space of quantum states is a complex projective space that possesses Fubini-Study metric tensor. By computing the block-diagonal approximation to the quantum geometric tensor of the circuit, a quantum circuit for quantum gradient descent is constructed which outperforms vanilla gradient descent in terms of the number of iterations required to reach the convergence [1]. Furthermore, the implementation of this gradient descent algorithm is done to construct quantum binary classification system and is compared with classical binary classification system.

II. Theoretical Background

A. Gradient Descent Algorithm

Gradient is a measurement of how steep a function's slope is. For gradient descent algorithm to work on a function, the function should be differentiable having a derivative for each point and convex in nature having a local minimum point. For a function $f(x)$ to be convex, the second derivative should be always greater than zero.

$$\frac{d^2 f(x)}{dx^2} > 0$$

The primary goal of gradient descent is to find the convex function's minima value, where the slope of the function is zero. The following is the optimal gradient descent definition for a function's local minimum or maximum.

- For getting the local minimum of the function, move towards the negative gradient, that is, move away from the gradient of function at the current value.
- For getting the local maximum of the function, move towards the positive gradient, that is, move towards the gradient of function at the current value.

For optimization, function's minimum value is calculated, which is getting the local minimum of the function by moving towards the negative gradient as shown in the Fig 1. This is known as a method of gradient descent or steepest descent.

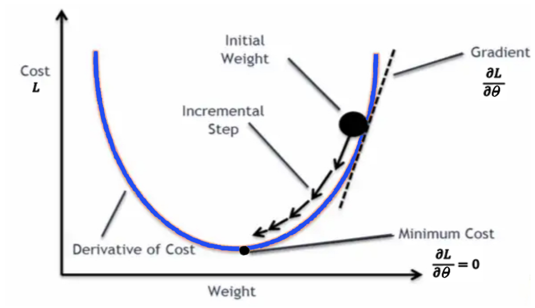


Fig. 1. Gradient descent algorithm.

The primary goal of gradient descent is to reduce the cost convex function through iterative parameter updates, where the cost function depends on these parameters. This error function (E) is defined as a measure of difference between true value and the predicted value given as

$$E = Y(true) - Y(predicted)$$

The mean square of this error function is calculated to obtain the cost function (L) shown in Fig 1 as

$$MSE = L(\theta) = mean(E^2(\theta))$$

where, the loss function depends on the variable parameters θ. Gradient descent algorithm is used to calculate next value of θ iteratively using the current value of θ and gradient of the value's cost function until the cost function is minimized. The iterative equation is written as

$$\theta(t+1) = \theta(t) + \eta * \frac{dL(\theta)}{d\theta}$$

where η is the learning rate or step size taken to reach the minimum of the cost function.
A lot of variations of gradient descent algorithm have been proposed to make this algorithm more efficient. In this project, vanilla gradient descent and ADAM optimization algorithms are implemented for the conducting the classical computations of gradient descent.

B. Vanilla Gradient Descent Optimization

Vanilla gradient descent is also known as batch gradient descent. This is the most basic variation of the gradient descent method. This algorithm computes the gradient of the cost function for the entire data. Here, the whole input is given to calculate the error first, and then the gradient is calculated and weights are updated.

$$\theta(t+1) = \theta(t) - \eta * \nabla L(\theta)$$

where, ∇ is the gradient of the cost function L. Since the whole data is given, this method results in stable gradient and stable convergence as a big advantage. However, because, for every iteration, the whole data is processed again and again, this optimization algorithm is slower [13].

C. ADAM Optimization

ADAM optimizer's name is derived from Adaptive Moment Estimation. This algorithm is s stochastic gradient descent method, where stochastic means random. It randomly picks one data point from the whole dataset at each iteration to reduce the computations. This is much more computationally efficient than vanilla gradient descent.

This technique calculates the adaptive learning rate for every parameter using momentum. Momentum is a method which accelerates the gradient descent in the relevant direction and decays the step size. When the gradients of two dimensions point in the same directions, the momentum term increases, and when the gradient changes the directions, the momentum term is reduced. As a result, oscillation is diminished and convergence becomes faster. [13] ADAM optimization keeps an exponentially decaying average of past gradient's mean (first moment m_t) and variance (second moment v_t), and makes the learning rate adaptive in the training phase using β_1 and β_2 values as the decay factors. The adaptive equation is given as

$$\theta(t+1) = \theta(t) - \frac{\eta}{\sqrt{v_t} + \epsilon} * m_t)$$

$$m_t = \beta_1 * m(t-1) + (1-\beta_1) * g_t$$

$$v_t = \beta_2 * v(t-1) + (1-\beta_2) * g_t^2$$

where, β_1, β_2 are the exponential decay rate with respect to first and second moments of the gradient, g_t is the previous gradient, m_t is the mean of the gradient as first moment, and v_t is the variance of the gradient as second moment. [13]

D. Natural Gradient Descent Optimization

Optimization in gradient descent is dependent to the Euclidean geometry. Gradient descent calculates the gradient based on Euclidean metric of the parameter space. Therefore, gradient descents have problems with badly scaled data, since the gradient will vary a lot in this situation.
This situation is explained in figure 2, two gaussians are considered which are solely parameterized by its mean, with varying variance in both images. The distance between the gaussians are same according to Euclidean metric (represented as red line), however, when analyzing the distributions of the gaussians, the distance is different for the first and second image. Therefore, Euclidean metric of the parameter space is not considering the distributions realised by the parameters. this information can be taken into account by finding the KL divergence of the two distributions. The first image has lower KL divergence, because of the greater overlap between the two gaussians as compared to the second image. This example is taken from [14].

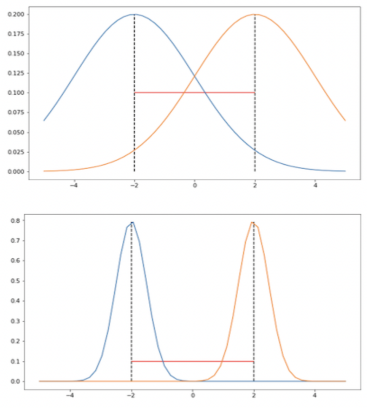


Fig. 2. Demonstration of two distributions for distribution space [14].

Therefore, the probability distribution, called as distribution space should be considered for steepest descent direction. Therefore, KL divergence is used to measure the difference between two models, the previous one and the updated one. KL divergence is the distance between two probabilistic distributions θ, θ' , named as distribution space, and steepest descent direction should be in that space.

$$D(\theta||\theta') = \sum(\theta(i) * log\frac{\theta(i)}{\theta'(i)}$$

1) Fisher Information Matrix: Since KL divergence is roughly analogous to a distance measure between distributions, this means Fisher information serves as a local distance metric between distributions, transforming the steepest descent in Euclidean parameter space to the steepest descent in the distribution space. Fisher information matrix is the second order derivative of KL divergence D.

$$F_\theta = \nabla^{2}_{\theta}{}' D(\theta'||\theta)$$

The inverse of this matrix, F inverse is multiplied with the gradient of loss function to make this natural gradient, and the whole process as natural gradient descent algorithm. The adaptive equation is given as

$$\theta(t+1) = \theta(t) - \eta * F^{-1} * \nabla L(\theta)$$

where F is the Fisher information matrix [15]. However, the calculation of the Fisher matrix and its inverse becomes computationally hard in classical domain. This fisher matrix can be calculated much faster in quantum domain.

E. Quantum Natural Gradient Descent Optimization

The analogous to natural gradient in quantum domain is quantum natural gradient descent. Just as the fisher information matrix, the term in quantum domain is fubini study metric tensor. When this metric tensor is restricted to the quantum states defining the parametric family, the real part of tensor gives the Quantum Geometric Tensor (QGT). This calculates the divergence between two distributions by computing the expectation values of the hermitian of a quantum state.

1) Quantum Circuit Representation of Quantum Geometric Tensor: The approximation of QGT as diagonal form is represented as :

$$\mathbf{QGT} = \begin{array}{c} \\ \theta_1 \\ \theta_2 \\ \\ \theta_L \end{array} \begin{array}{c} \begin{array}{cccc} \theta_1 & \theta_2 & \cdots & \theta_L \end{array} \\ \begin{pmatrix} G^1 & 0 & \cdots & 0 \\ 0 & G^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & G^L \end{pmatrix} \end{array}$$

where θ_l parameterizes the l^{th} layer of the quantum circuit. QGT's block diagonal approximation is similar to the Fubini-Study metric tensor's block diagonal approximation.

$$g_{ij}^{\ l} = Re(G_{ij}^{\ l}) = G_{ij}^{\ l}$$

Therefore, the Fubini-Study metric tensor can be calculated using the Hermitian observable's expectation values, and these expectation values are the values of the subcircuits of the full quantum circuit which is measurable. [9]

These obtained $g_{ij}^{\ l}$ values corresponds to the values of the Fubini-Study metric tensor which is the quantum analogous of Fisher Information matrix. Therefore, the adaptive equation for quantum natural gradient descent becomes

$$\theta(t+1) = \theta(t) - \eta * g^{+} * \nabla L(\theta)$$

where g^{+} is the pseudo inverse of g, Fubini-Study metric tensor. Quantum Natural gradient (QNG) optimizer calculated the Fubini-Study metric tensor $g(\theta)$ at each iteration and updates the parameter θ values using the above equation.

III. Implementation

In this project, gradient descent algorithms are implemented and analysed for binary classification systems in classical and quantum domain. Firstly, for an adaptive system, three algorithms namely, vanilla gradient descent, ADAM, and quantum natural gradient descent are implemented for optimization of the system parameters, and their cost convergence curve are compared. Secondly, two binary classification systems are constructed using classical neural network and quantum variational classifier to compare classical and quantum optimizations. The implementation is described in the upcoming subsections.

A. Quantum Variational Circuits

In quantum computations, we can implement adaptive algorithms in variational quantum circuits, which are also called adaptable quantum circuits, in which the inputs are encoded and there are variational parameters as well, which are represented as quantum information.

As seen in Fig 3, the input value is x and the variational parameters θ are encoded forming the unitary transformation as $U(x, \theta)$. James, et al [1] have created a random variational circuit, for H2 molecule for finding the energy value with some

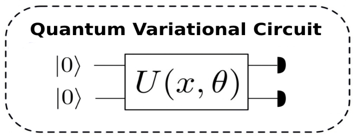


Fig. 3. Variational Quantum Circuit Representation.

initial parameters, which formed the variational circuit shown in the Fig 4. In this project, same H2 molecule variational circuit is designed to compared the loss convergence for gradient descent and quantum natural gradient descent.

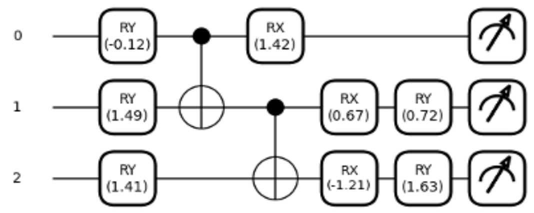


Fig. 4. Variational Quantum Circuit for H2 molecule

B. *Gradient Descent Optimization Implementations*

For implementing the quantum simulations, **Pennylane** library of python is used, which is an open-source software framework for quantum machine learnig [16]. For comparing the cost convergence of classical and quantum gradient descents, the H2 molecule variational circuit is chosen as the system with adaptive parameters which needs to be adapted. Circuit using 2 qubits and 3 qubits are designed for comparison of performance with different number of qubits. For 500 iterations, with 0.01 learning/adaptive rate, VGD, ADAM and QNG optimizers are compared. initial parameters are given input to variational circuit, updated parameters are calculated using different gradient descent optimizers and cost history is saved to further analysis, for 500 iterations. This cost history convergence is plotted to see the convergence rates using different optimizers.

C. *Classical Binary Classification*

For this project, dataset is collected from the University of California Irvine (UCI) machine learning repository [11]. This IRIS dataset is about classification of IRIS plants, which consists of 2 classes of IRIS plants, with 4 input features about the measurement of petal length and width, sepal length and width.

Classical Binary classification can be implemented using a neural network model, where the 4 input features are given to input layers, which are then computed with the weights associated with hidden layers, such that the output of output layer, that is the predicted output is as accurate as the desired output, as shown in Fig 5. These weights are updated after every iteration using gradient descent optimization.

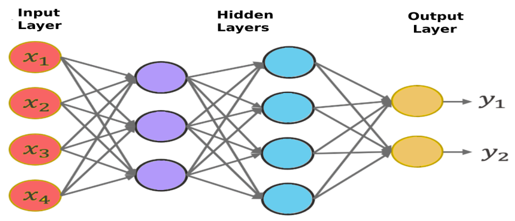


Fig. 5. Classical Binary classification model.

D. *Quantum Binary Classification*

Quantum classification model is also known as quantum variational classifier. A quantum circuit is designed which behaves similarly to a traditional machine learning algorithm. As seen in the Fig 6, the input data features (4 feature values) are encoded as quantum states represented as U(x). The second part consists of variational quantum circuit, which includes the adaptive parameters which are updated after every iteration, and the obtained measurement results are optmized using the gradient descent algorithms by evaluating the cost functions in classical domain, to be classified as the desired output class.

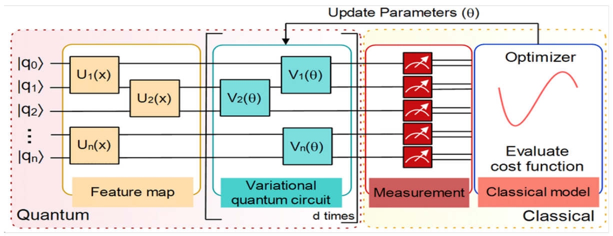


Fig. 6. Quantum Binary classification model [12].

IV. Results

A. *Gradient Descent Comparisons*

For the H2 molecule variational circuit, with random initial parameters, the gradient descent optimization techniques is compared for varying number of qubits, which is the circuit depth. The cost convergence curve using various optimizers is shown in Fig 7.

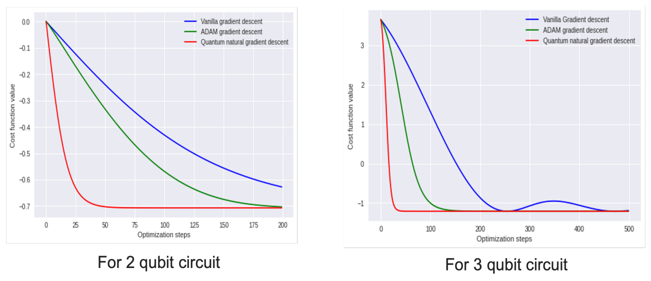


Fig. 7. Cost Convergence curves for a given quantum variational circuit with different optimizers.

In the first curve on the left side, for a variational circuit with 2 qubits only (2 nodes in each layer), it is seen that vanilla gradient descent optimization takes more number of iterations, the optimization steps to converge to a loss value. On comparison, ADAM optimizes faster as compared to the Vanilla gradient descent, because it considers the variable learning rate/step size. Whereas if the quantum natural gradient cost curve converges much faster with very few iterations compared to classical optimization algorithms. Quantum algorithm performs better since the information of probabilistic distributions is included. Comparing with the right graph of Fig 7, where the variational circuits are designed using 3 qubits, a fast convergence is seen as the number of qubits is increased. Therefore, ”Quantum Natural Gradient optimizer appears more significant with increasing number of qubits and retains its advantage with increasing circuit depth” [1].

B. Binary classification results

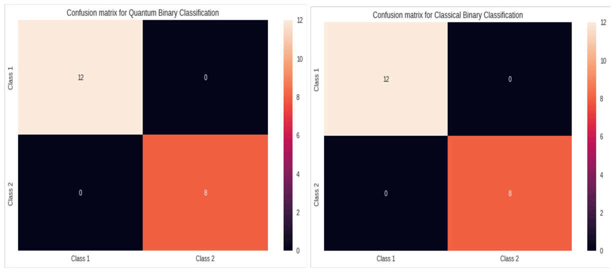


Fig. 8. Confusion Matrix for Binary Classification using classical and quantum computation.

For the classification analysis, binary classification has been performed using classical machine learning and quantum machine learning. It is seen that training using quantum variational classifier is much faster, as the loss function is reduced in a smaller number of epochs as compared to the epochs for loss function convergence using classical neural network model. However, for the testing samples, 100% accuracy is obtained using both the classification models with the confusion matrix shown in the Fig 8. Same results are obtained for both these classifications where 12 tests are classified correctly as class 1, and 8 tests for class 2 correctly.

V. Acknowledgement

I would like to express my gratitude to Professor Andreas Spanias for providing their guidance and suggestions throughout the course of the project. I would like to thank the center for Machine Learning and Intelligent Systems of University of California Irvine (UCI) for providing the IRIS Data Set in their Machine learning Repository to be publicly accessible.

VI. Conclusion

From this study, it is seen that Adam Optimization is more efficient than vanilla gradient descent. It has been said that quantum computing promises exponential speedup for some complex problems, and it is seen in this study that quantum natural gradient outperforms both the classical approaches, vanilla gradient descent and Adam optimization. As we increase the number of qubits, and number of layers of variational circuit, quantum natural gradient optimizer appears more significant, and retains its advantage with increasing circuit depth. It is seen that same classification results are obtained for both quantum and classical classification, where quantum classification model is optimized in less number of epochs (iterations) as compared to classical classification model.

References

[1] Stokes, James, Josh Izaac, Nathan Killoran, and Giuseppe Carleo. ”Quantum natural gradient.” Quantum 4 (2020): 269.

[2] X. Wang, L. Yan and Q. Zhang, ”Research on the Application of Gradient Descent Algorithm in Machine Learning,” 2021 International Conference on Computer Network, Electronic and Automation (ICCNEA), 2021, pp. 11-15, doi: 10.1109/ICCNEA53019.2021.00014.

[3] Ristè, D., da Silva, M.P., Ryan, C.A. et al. Demonstration of quantum advantage in machine learning. npj Quantum Inf 3, 16 (2017). https://doi.org/10.1038/s41534-017-0017-3

[4] Durr, Christoph, and Peter Hoyer. ”A quantum algorithm for finding the minimum.” arXiv preprint quant-ph/9607014 (1996).

[5] Farhi, Edward, and Aram W. Harrow. ”Quantum supremacy through the quantum approximate optimization algorithm.” arXiv preprint arXiv:1602.07674 (2016).

[6] Gian Giacomo Guerreschi and Mikhail Smelyanskiy. Practical optimization for hybrid quantumclassical algorithms. arXiv preprint arXiv:1701.01450, 2017.

[7] James C Spall et al. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control, 37(3):332–341, 1992. DOI: 10.1109/9.119632.

[8] Aram Harrow and John Napp. Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms. arXiv preprint arXiv:1901.05374, 2019.

[9] Naoki Yamamoto. “On the natural gradient for variational quantum eigensolver.” arXiv:1909.05074, 2019

[10] Zou, F., Shen, L., Jie, Z., Zhang, W., Liu, W. (2019). A sufficient condition for convergences of adam and rmsprop. In Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition (pp. 11127-11135).

[11] IRIS Dataset, UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets/iris

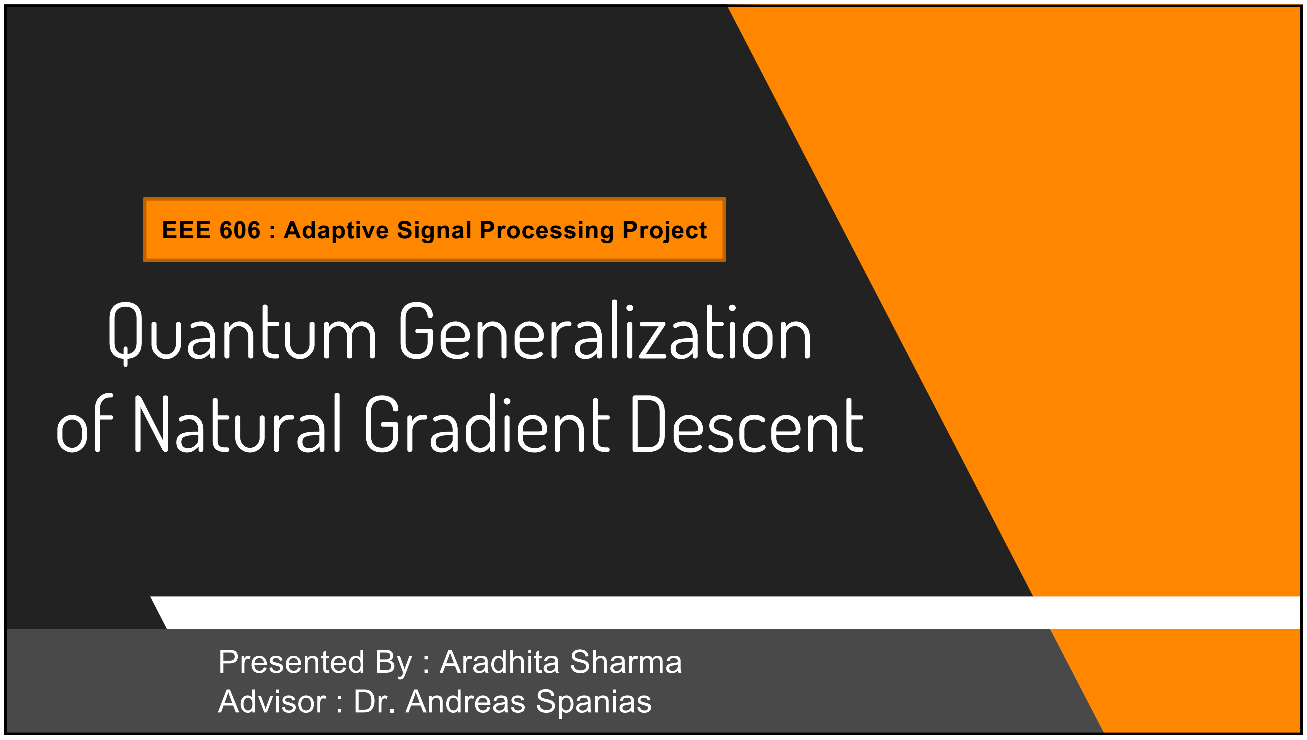
[12] Sen, Pinaki Bhatia, Amandeep. (2021). Variational Quantum Classifiers Through the Lens of the Hessian.

[13] Ruder, Sebastian. ”An overview of gradient descent optimization algorithms.” arXiv preprint arXiv:1609.04747 (2016).

[14] Pascanu, Razvan, and Yoshua Bengio. “Revisiting natural gradient for deep networks.” arXiv preprint arXiv:1301.3584 (2013).

[15] Ly, Alexander, et al. “A tutorial on Fisher information.” Journal of Mathematical Psychology 80 (2017): 40-55.

[16] pennylane.ai, Pennylane, https://pennylane.ai/

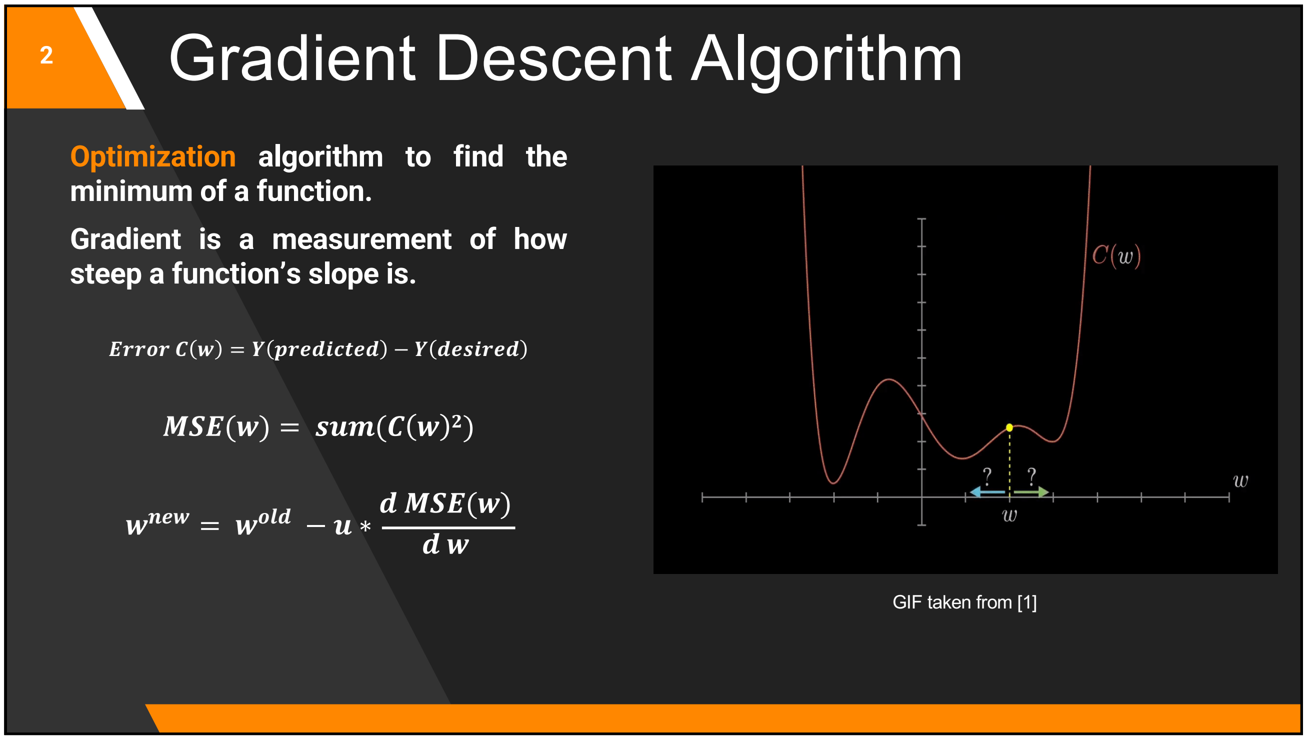


EEE 606 : Adaptive Signal Processing Project

Quantum Generalization of Natural Gradient Descent

Presented By : Aradhita Sharma
Advisor : Dr. Andreas Spanias

1



2

Gradient Descent Algorithm

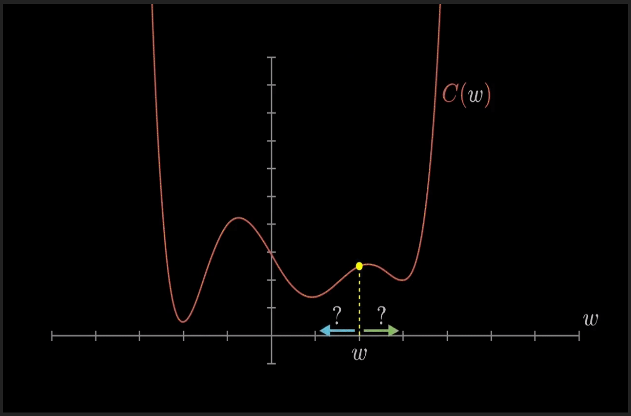
Optimization algorithm to find the minimum of a function.

Gradient is a measurement of how steep a function's slope is.

$$Error\ C(w) = Y(predicted) - Y(desired)$$

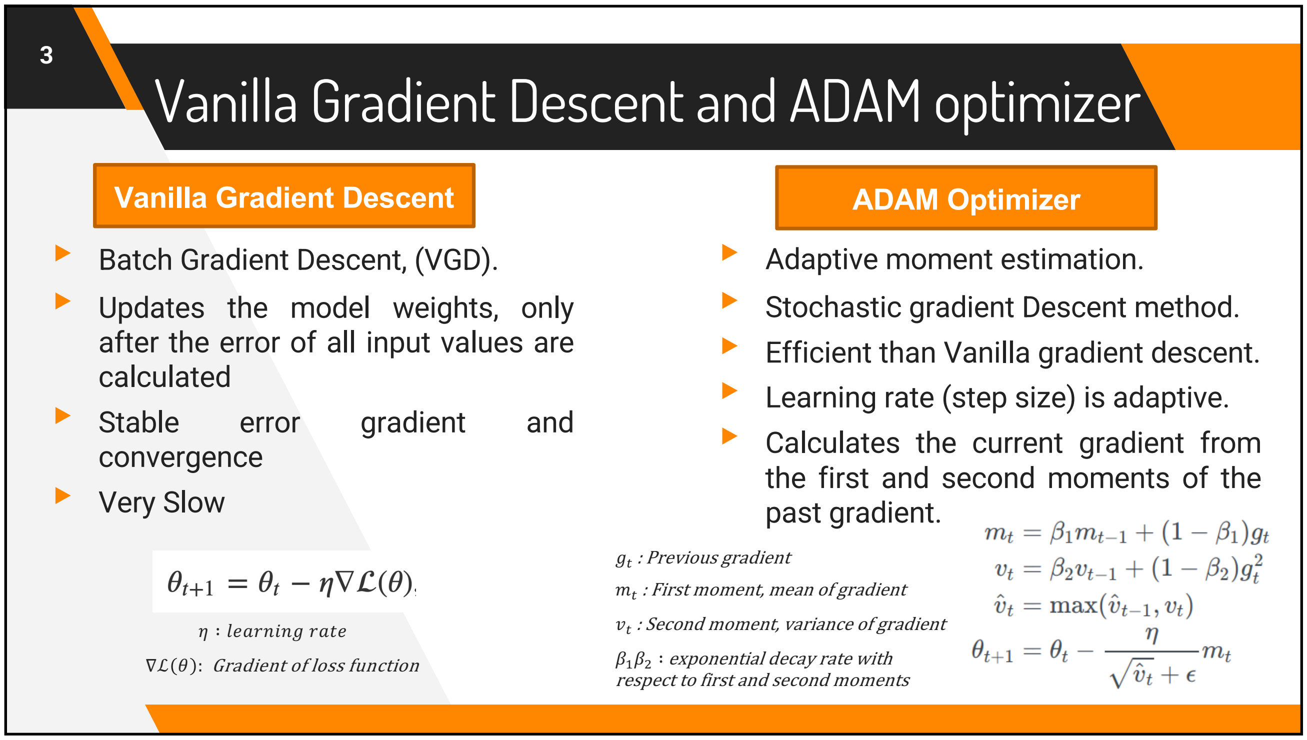
$$MSE(w) = sum(C(w)^2)$$

$$w^{new} = w^{old} - u * \frac{d\ MSE(w)}{d\ w}$$



GIF taken from [1]

2



3

Vanilla Gradient Descent and ADAM optimizer

Vanilla Gradient Descent

- Batch Gradient Descent, (VGD).
- Updates the model weights, only after the error of all input values are calculated
- Stable error gradient and convergence
- Very Slow

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta)$$

η : *learning rate*

$\nabla\mathcal{L}(\theta)$: *Gradient of loss function*

ADAM Optimizer

- Adaptive moment estimation.
- Stochastic gradient Descent method.
- Efficient than Vanilla gradient descent.
- Learning rate (step size) is adaptive.
- Calculates the current gradient from the first and second moments of the past gradient.

g_t : *Previous gradient*

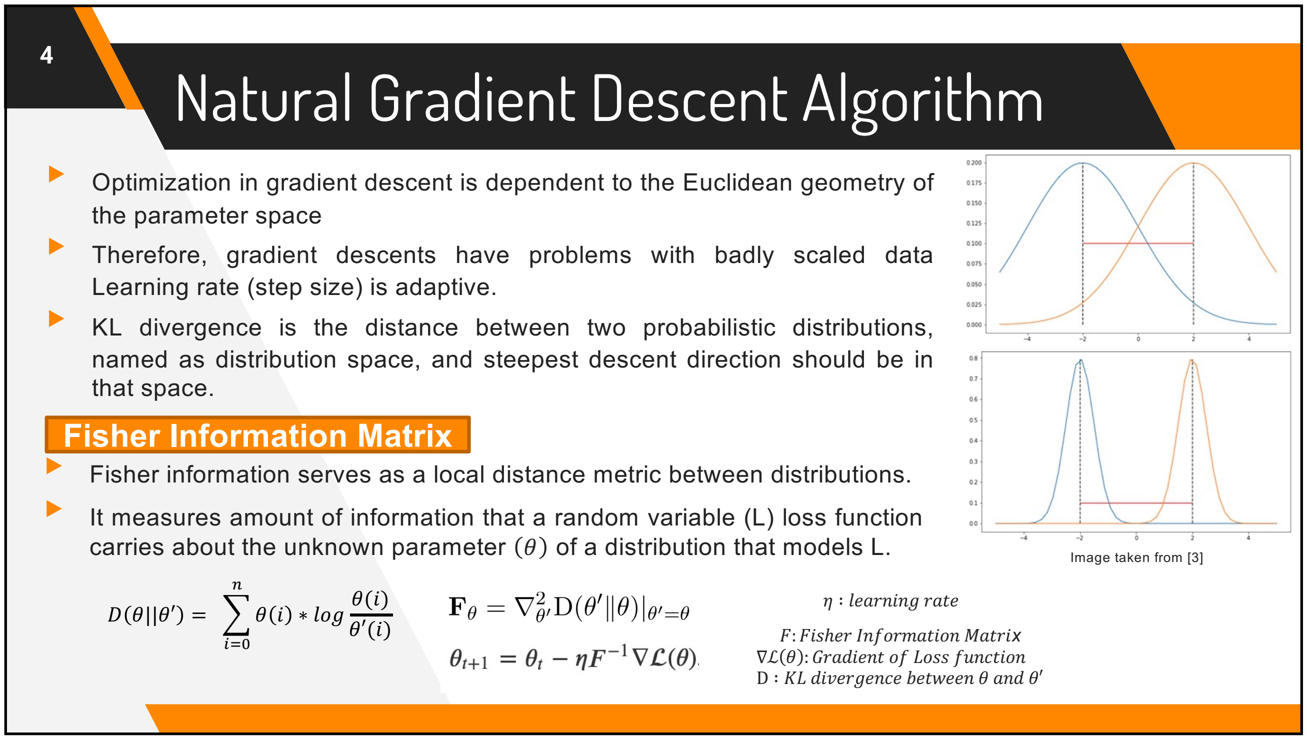
m_t : *First moment, mean of gradient*

v_t : *Second moment, variance of gradient*

$\beta_1\beta_2$: *exponential decay rate with respect to first and second moments*

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} m_t$$

3



4

Natural Gradient Descent Algorithm

- Optimization in gradient descent is dependent to the Euclidean geometry of the parameter space
- Therefore, gradient descents have problems with badly scaled data Learning rate (step size) is adaptive.
- KL divergence is the distance between two probabilistic distributions, named as distribution space, and steepest descent direction should be in that space.

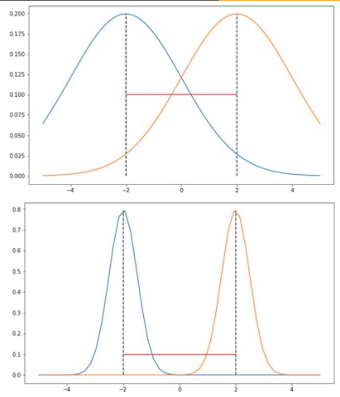


Image taken from [3]

Fisher Information Matrix

- Fisher information serves as a local distance metric between distributions.
- It measures amount of information that a random variable (L) loss function carries about the unknown parameter (θ) of a distribution that models L.

$$D(\theta||\theta') = \sum_{i=0}^{n} \theta(i) * log \frac{\theta(i)}{\theta'(i)}$$

$$\mathbf{F}_\theta = \nabla^2_{\theta'} \mathrm{D}(\theta' \| \theta)|_{\theta'=\theta}$$

$$\theta_{t+1} = \theta_t - \eta F^{-1} \nabla \mathcal{L}(\theta)$$

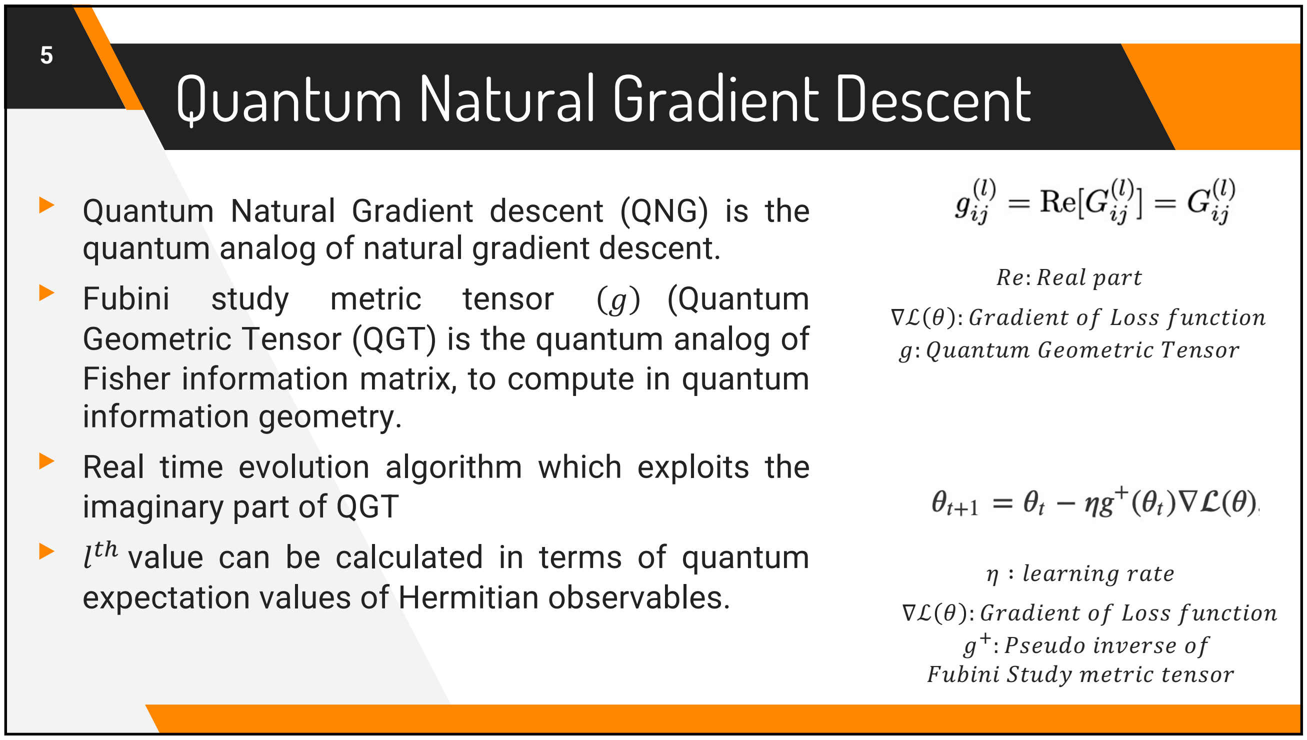
η : *learning rate*

F: *Fisher Information Matrix*

$\nabla\mathcal{L}(\theta)$: *Gradient of Loss function*

D : *KL divergence between* θ *and* θ'

4



5

Quantum Natural Gradient Descent

- Quantum Natural Gradient descent (QNG) is the quantum analog of natural gradient descent.
- Fubini study metric tensor (g) (Quantum Geometric Tensor (QGT) is the quantum analog of Fisher information matrix, to compute in quantum information geometry.
- Real time evolution algorithm which exploits the imaginary part of QGT
- l^{th} value can be calculated in terms of quantum expectation values of Hermitian observables.

$$g_{ij}^{(l)} = \mathrm{Re}[G_{ij}^{(l)}] = G_{ij}^{(l)}$$

$Re: Real\ part$

$\nabla\mathcal{L}(\theta): Gradient\ of\ Loss\ function$

$g: Quantum\ Geometric\ Tensor$

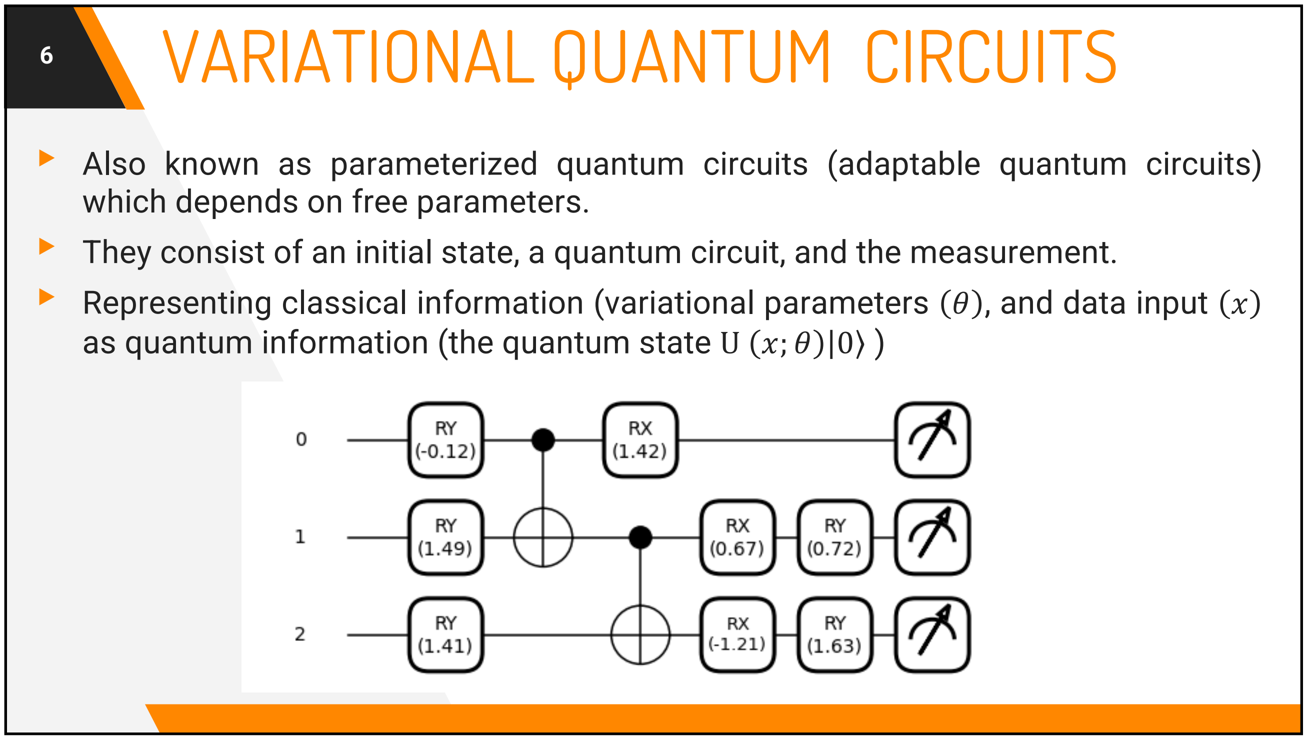
$$\theta_{t+1} = \theta_t - \eta g^{+}(\theta_t)\nabla\mathcal{L}(\theta)$$

$\eta : learning\ rate$

$\nabla\mathcal{L}(\theta): Gradient\ of\ Loss\ function$

$g^{+}: Pseudo\ inverse\ of$
$Fubini\ Study\ metric\ tensor$

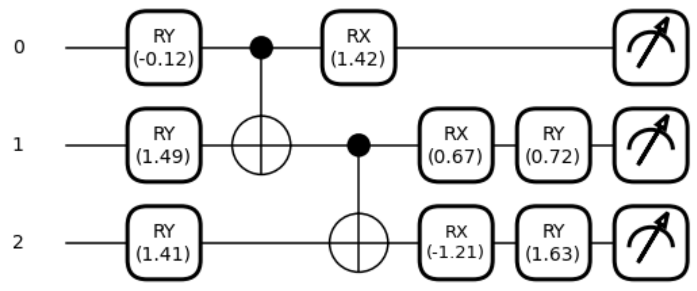
5



6

VARIATIONAL QUANTUM CIRCUITS

- Also known as parameterized quantum circuits (adaptable quantum circuits) which depends on free parameters.
- They consist of an initial state, a quantum circuit, and the measurement.
- Representing classical information (variational parameters (θ), and data input (x) as quantum information (the quantum state $\mathrm{U}\,(x;\theta)|0\rangle$)



6

7

Binary Classification Model

Classical Classification

IRIS dataset : of 3 classes of IRIS plants, out of which 2 are chosen, with 4 input features.

IRIS dataset : of 3 classes of IRIS plants, out of which 2 are chosen, with 4 input features.

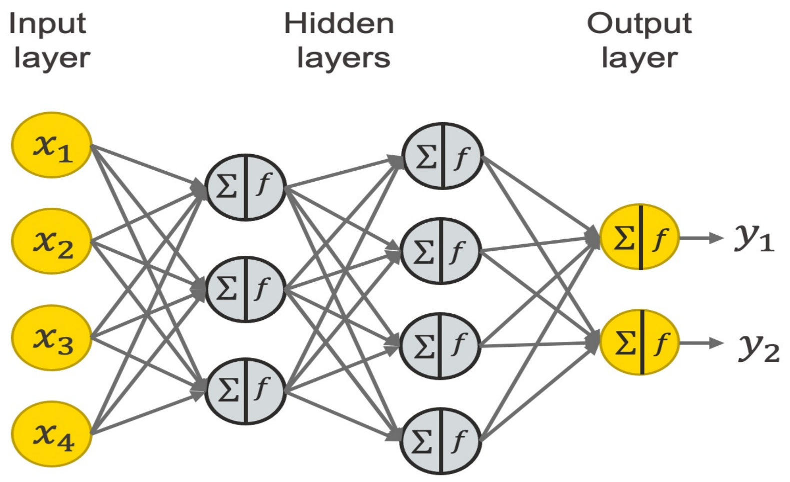


Image taken from [4]

7

8

Binary Classification Model

Quantum Classification

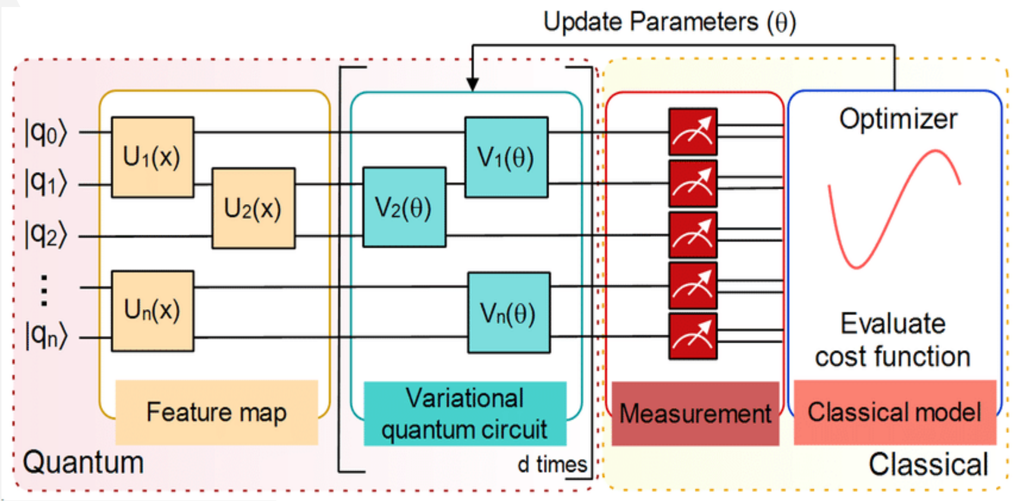
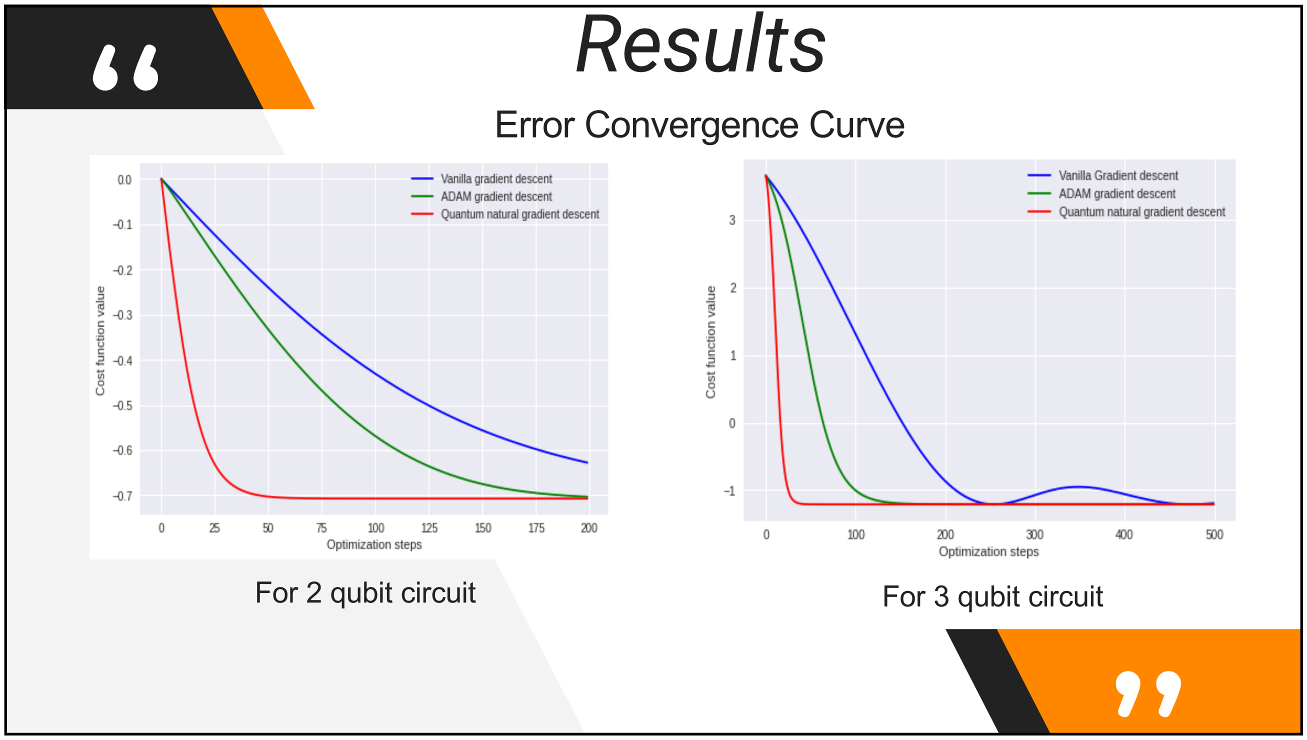


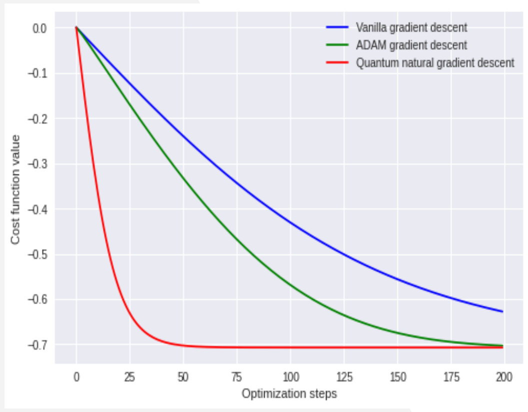
Image taken from [5]

8

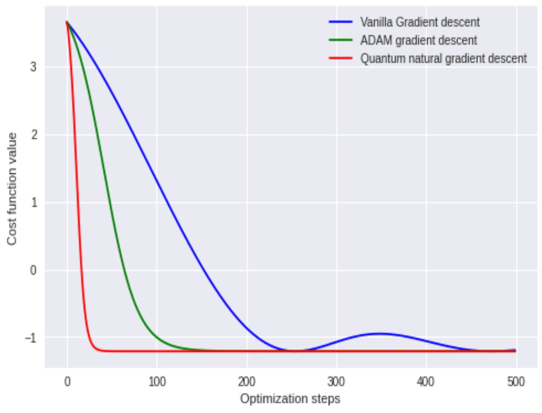


Results

Error Convergence Curve

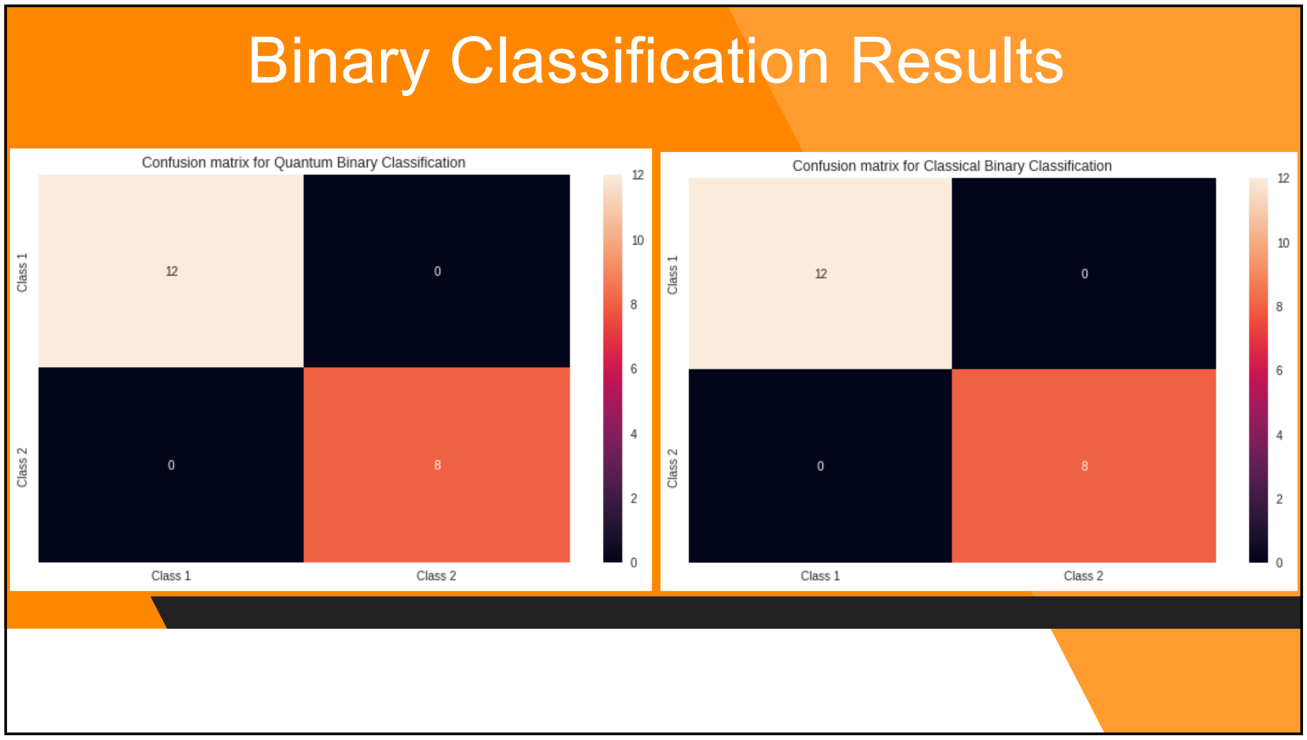


For 2 qubit circuit

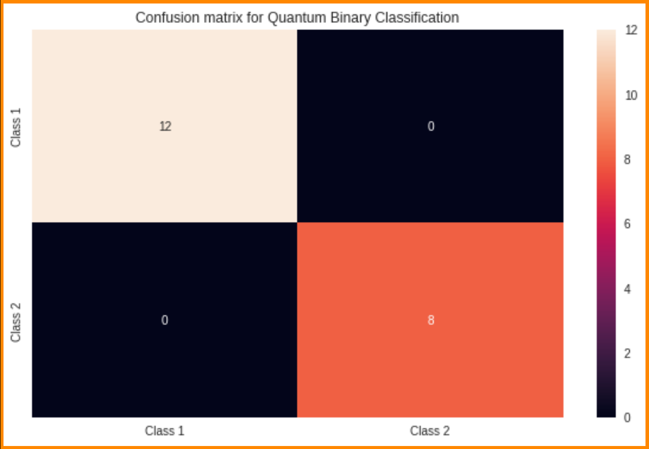


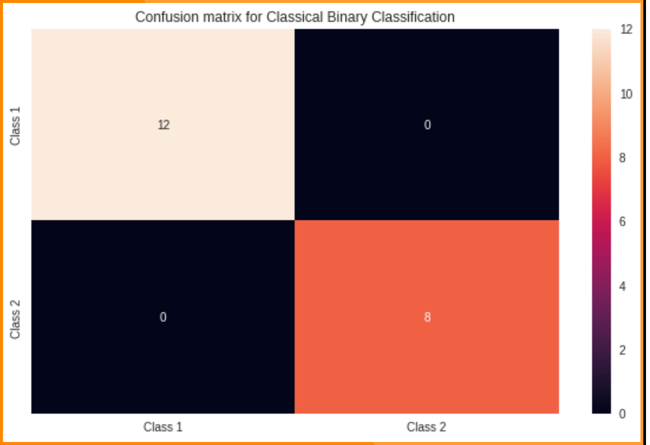
For 3 qubit circuit

9



Binary Classification Results





10

11

Conclusion

- Adam Optimization is more efficient than vanilla gradient descent.
- Quantum natural gradient outperforms both vanilla gradient descent and Adam optimization.
- Quantum natural gradient optimizer appears more significant with increasing number of qubits and retains its advantage with increasing circuit depth.
- Same classification results for both Quantum and Classical classification.

Google Colab : Code link

https://colab.research.google.com/drive/1OVQEEFCb4_rv4Laty4W-hdA7-KC_-k2E

11

12

References

[1] https://gfycat.com/jampackedfewcottontail

[2] https://preview.redd.it/m1iloe7s16a61.gif?format=mp4&s=c28d64e5ffe30014eb04916649187d5a945d79f4

[3] https://agustinus.kristia.de/techblog/2018/03/14/natural-gradient/

[4] https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks

[5] Sen, Pinaki & Bhatia, Amandeep. (2021). Variational Quantum Classifiers Through the Lens of the Hessian.

[6] Durr, Christoph, and Peter Hoyer. "A quantum algorithm for finding the minimum." *arXiv preprint quant-ph/9607014* (1996).

[7] Farhi, Edward, and Aram W. Harrow. "Quantum supremacy through the quantum approximate optimization algorithm." *arXiv preprint arXiv:1602.07674* (2016).

[8] Aram Harrow and John Napp. Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms. arXiv preprint arXiv:1901.05374, 2019.

[9] Naoki Yamamoto. “On the natural gradient for variational quantum eigensolver.” arXiv:1909.05074, 2019

[10] Stokes, James, Josh Izaac, Nathan Killoran, and Giuseppe Carleo. "Quantum natural gradient." *Quantum* 4 (2020): 269.

12

Thank you

13

Quantum Natural Gradient

James Stokes[1], Josh Izaac[2], Nathan Killoran[2], and Giuseppe Carleo[3]

[1]Center for Computational Quantum Physics and Center for Computational Mathematics, Flatiron Institute, New York, NY 10010 USA
[2]Xanadu, 777 Bay Street, Toronto, Canada
[3]Center for Computational Quantum Physics, Flatiron Institute, New York, NY 10010 USA

A quantum generalization of Natural Gradient Descent is presented as part of a general-purpose optimization framework for variational quantum circuits. The optimization dynamics is interpreted as moving in the steepest descent direction with respect to the Quantum Information Geometry, corresponding to the real part of the Quantum Geometric Tensor (QGT), also known as the Fubini-Study metric tensor. An efficient algorithm is presented for computing a block-diagonal approximation to the Fubini-Study metric tensor for parametrized quantum circuits, which may be of independent interest.

1 Introduction

Variational optimization of parametrized quantum circuits is an integral component for many hybrid quantum-classical algorithms, which are arguably the most promising applications of Noisy Intermediate-Scale Quantum (NISQ) computers [29]. Applications include the Variational Quantum Eigensolver (VQE) [27], Quantum Approximate Optimization Algorithm (QAOA) [10] and Quantum Neural Networks (QNNs) [9, 14, 30].

All the above are examples of stochastic optimization problems whereby one minimizes the expected value of a random cost function over a set of variational parameters, using noisy estimates of the cost and/or its gradient. In the quantum setting these estimates are obtained by repeated measurements of some Hermitian observables for a quantum state which depends on the variational parameters.

A variety of optimization methods have been proposed in the variational quantum circuit literature for determining optimal variational parameters, including derivative-free (zeroth-order) methods such as Nelder-Mead, finite-differencing [12] or SPSA [33]. Recently the possibility of exploiting direct access to first-order gradient information has been explored. Indeed quantum circuits have been designed to estimate such gradients with minimal overhead compared to objective function evaluations [31].

One motivation for exploiting first-order gradients is theoretical: in the convex case, the expected error in the objective function using the best known zeroth-order stochastic optimization algorithm scales polynomially with the dimension d of the parameter space, whereas Stochastic Gradient Descent (SGD) converges independently of d. Another motivation stems from the empirical success of stochastic gradient methods in training deep neural networks, which involve minimization of non-convex objective functions over high-dimensional parameter spaces.

The application of SGD to deep learning suffers from the caveat that successful optimization hinges on careful hyper-parameter tuning of the learning rate (step size) and other hyper-parameters such as Momentum. Indeed a vast literature has developed devoted to step size selection (see e.g. [15]). The difficulty of choosing a step size can be understood intuitively in the simple quadratic bowl approximation, where the optimal step size depends on the maximum eigenvalue of the Hessian, a quantity which is difficult to calculate in high dimensions. In practical applications the step size selection problem is overcome by using adaptive methods of stochastic optimization such as Adam [18] which have enjoyed wide adoption because of their ability to dynamically select a step size by maintaining a history of past gradients.

Independently of the improvements arising from historical averaging as in Momentum and Adam, it is natural to ask if the geometry of quantum states favors a particular optimization strategy. Indeed, it is well-known that the choice of optimization is intimately linked to the choice of geometry on the parameter space [26]. In the most well-known case of vanilla gradient descent, the relevant geometry corresponds to the l_2 geometry as can be seen from the following exact rewriting of the iterative update rule

$$\begin{aligned}\theta_{t+1} &:= \theta_t - \eta \nabla \mathcal{L}(\theta_t) \ , \qquad (1)\\ &= \arg\min_{\theta \in \mathbb{R}^d} \left[\langle \theta - \theta_t, \nabla \mathcal{L}(\theta_t) \rangle + \frac{1}{2\eta} \|\theta - \theta_t\|_2^2 \right] \ ,\end{aligned}$$

where $\mathcal{L}$ is the loss as a function of the variational parameters $\theta \in \mathbb{R}^d$ and η is the step size. Thus, vanilla gradient descent moves in the steepest descent direction with respect to the l_2 geometry.

In the deep learning literature, it has been argued that the l_2 geometry is poorly adapted to the space of weights of deep networks, due to their intrinsic parameter redundancy [26]. The Natural Gradient [1], in contrast, moves in the steepest descent direction with respect to the Information Geometry. This natural gradient descent is advantageous compared to the vanilla gradient because it is invariant under arbitrary re-parametrizations [1] and moreover possesses an approximate invariance with respect to over-parametrizations [22], which are typical for deep neural networks.

In a similar spirit, the quantum circuit literature has investigated the impact of geometry on dynamics of variational algorithms. In particular, it was shown that under the assumption of strong convexity, the l_2 geometry is sub-optimal in some situations compared to the l_1 geometry [13]. The intuitive argument put forth favoring the l_1 geometry is that some quantum state ansätze can be physically interpreted as a sequence of pulses of Hamiltonian evolution, starting from a fixed reference state. In this particular parametrization, each variational parameter can be interpreted as the duration of the corresponding pulse. This is not the only useful parametrization of quantum states, however, and it is thus desirable to find a descent direction which is not tied to any particular parametrization.

Ref. [13] leaves open the problem of finding the relevant geometry for general-purpose variational quantum algorithms, and this paper seeks to fill that void. The contributions of this papers are as follows:

- We point out that the demand of invariance with respect to arbitrary reparametrizations can be naturally fulfilled by introducing a Riemannian metric tensor on the space of quantum states, and that the implied descent direction is invariant with respect to reparametrizations by construction.
- We note that the space of quantum states is naturally equipped with a Riemannian metric, which differs from l_2 and l_1 geometries explored previously. In fact, in the absence of noise, the space of quantum states is a complex projective space, which possesses a unique unitarily-invariant metric tensor called the Fubini-Study metric tensor. When restricted to the submanifold of quantum states defining the parametric family, the Fubini-Study metric tensor emerges as the real part of a more general geometric quantity called the Quantum Geometric Tensor (QGT).
- We show that the resulting gradient descent algorithm is a direct quantum analogue of the Natural Gradient in the statistics literature, and reduces to it in a certain limit.
- We present quantum circuit construction which computes a block-diagonal approximation to the Quantum Geometric Tensor and show that a simple diagonal preconditioning scheme outperforms vanilla gradient descent in terms of number of iterates required to achieve convergence

2 Theory

2.1 Quantum Information Geometry

Consider the set of probability distributions on N elements $[N] = \{1, \ldots, N\}$; that is, the set of positive vectors $p \in \mathbb{R}^N$, $p \succeq 0$ which are normalized in the 1-norm $\|p\|_1 = 1$. The following function is easily shown to be a metric (Fisher-Rao metric) on the probability simplex Δ^{N-1},

$$d(p, q) = \arccos(\langle \sqrt{p}, \sqrt{q} \rangle) \ , \qquad (2)$$

where $\sqrt{p}$ and $\sqrt{q}$ denote the elementwise square root of the probability vectors in the probability simplex $p, q \in \Delta^{N-1}$.

Now consider a parametric family of strictly positive probability distributions $p_\theta \succ 0$ indexed by real parameters $\theta \in \mathbb{R}^d$. It can be shown that the infinitesimal squared line element between two members of the parametric family is given by

$$d^2(p_\theta, p_{\theta+\mathrm{d}\theta}) = \frac{1}{4} \sum_{(i,j)\in[d]^2} I_{ij}(\theta) \mathrm{d}\theta^i \mathrm{d}\theta^j \ , \tag{3}$$

where $I_{ij}(\theta)$ are the components of a Riemannian metric tensor (with possible degeneracies) called the Fisher Information Matrix. Letting $p_\theta(x)$ denote the component of the probability vector p_θ corresponding to $x \in [N]$ we have

$$I_{ij}(\theta) = \sum_{x\in[N]} p_\theta(x) \frac{\partial \log p_\theta(x)}{\partial \theta^i} \frac{\partial \log p_\theta(x)}{\partial \theta^j} \ . \tag{4}$$

Now consider a N-dimensional complex Hilbert space $\mathbb{C}^N$. Given a vector $\psi \in \mathbb{C}^N$ which is normalized in the 2-norm $\|\psi\|_2 = 1$, a pure quantum state is defined as the projection $P_\psi = |\psi\rangle\langle\psi| \in \mathbb{CP}^{N-1}$ onto the one-dimensional subspace spanned by the unit vector ψ. In direct analogy with the simplex, the following function is easily shown to be a metric (Fubini-Study metric) on the space of pure states:

$$d(P_\psi, P_\phi) = \arccos(|\langle\psi, \phi\rangle|) \ , \tag{5}$$

where $\psi, \phi \in \mathbb{C}^N$ are unit vectors. Letting ψ_θ denote a parametric family of unit vectors, the infinitesimal squared line element between two states defined by the parametric family is given by

$$d^2(P_{\psi_\theta}, P_{\psi_{\theta+\mathrm{d}\theta}}) = \sum_{(i,j)\in[d]^2} g_{ij}(\theta) \, \mathrm{d}\theta^i \mathrm{d}\theta^j \ , \tag{6}$$

where $g_{ij}(\theta) = \mathrm{Re}[G_{ij}(\theta)]$ is the Fubini-Study metric tensor, which can be expressed in terms of the following Quantum Geometric Tensor (see [3, 19, 34] for a review),

$$G_{ij}(\theta) = \left\langle \frac{\partial\psi_\theta}{\partial\theta^i}, \frac{\partial\psi_\theta}{\partial\theta^j} \right\rangle - \left\langle \frac{\partial\psi_\theta}{\partial\theta^i}, \psi_\theta \right\rangle \left\langle \psi_\theta, \frac{\partial\psi_\theta}{\partial\theta^j} \right\rangle \ . \tag{7}$$

Indeed if $\{|x\rangle : x \in [N]\}$ denotes an orthonormal basis for $\mathbb{C}^N$ then one can easily verify that for the family of unit vectors defined by

$$\psi_\theta = \sum_{x\in[N]} \sqrt{p_\theta(x)} \, |x\rangle \ , \tag{8}$$

we have $G_{ij}(\theta) = \frac{1}{4} I_{ij}(\theta)$. Clearly, not all quantum states are of this form due to the possibility of complex phases.

Finally, although we have posed the discussion in finite-dimensions, all of the above concepts carry over to infinite-dimensional Hilbert spaces by appropriately replacing sums by integrals.

2.2 Optimization problem

Consider a parametric family of unitary operators $U_\theta \in U(N)$ which are indexed by real parameters $\theta \in \mathbb{R}^d$. Given a fixed reference unit vector $|0\rangle \in \mathbb{C}^N$ and a Hermitian operator $H = H^\dagger$ acting on $\mathbb{C}^N$, we consider the following optimization problem

$$\min_{\theta\in\mathbb{R}^d} \mathcal{L}(\theta) \ , \qquad \mathcal{L}(\theta) = \frac{1}{2} \mathrm{tr}(P_{\psi_\theta} H) = \frac{1}{2} \langle \psi_\theta, H\psi_\theta \rangle \ , \tag{9}$$

where $\psi_\theta = U_\theta|0\rangle$ and $P_{\psi_\theta} \in \mathbb{CP}^{N-1}$ is the associated projector. In particular, note that ψ_θ is normalized since U_θ is unitary. Global optimization of the nonconvex objective function $\mathcal{L}(\theta)$ is impractical, so we instead propose to search for local optima by iterating the following discrete-time dynamical system,

$$\theta_{t+1} = \arg\min_{\theta\in\mathbb{R}^d} \left[\langle \theta - \theta_t, \nabla\mathcal{L}(\theta_t) \rangle + \frac{1}{2\eta} \|\theta - \theta_t\|^2_{g(\theta_t)} \right] \ , \tag{10}$$

where $g(\theta_t)$ is the symmetric matrix with (i, j) component $\mathrm{Re}[G_{ij}(\theta_t)]$ and we have introduced the following notation:

$$\|\theta - \theta_t\|^2_{g(\theta_t)} = \langle \theta - \theta_t, g(\theta_t)(\theta - \theta_t) \rangle \ . \tag{11}$$

The first-order optimality condition corresponding to (10) is

$$g(\theta_t)(\theta_{t+1} - \theta_t) = -\eta \nabla\mathcal{L}(\theta_t) \ . \tag{12}$$

A solution of the optimization problem (10) is thus provided by the following expression which involves the pseudo-inverse $g^+(\theta_t)$ of the metric tensor,

$$\theta_{t+1} = \theta_t - \eta \, g^+(\theta_t) \nabla\mathcal{L}(\theta_t) \ . \tag{13}$$

In practice, however, we avoid materializing the pseudo-inverse by directly solving the linear system (12) which is both more efficient and more numerically stable. In the continuous-time limit corresponding to vanishing step size $\eta \to 0$, the dynamics (10) is equivalent to imaginary-time evolution within the variational subspace according to the Hamiltonian H, as shown in the supplementary material.

2.3 Relationship with previous work

Quantum Natural Gradient optimization possesses important differences compared to its classical counterpart because of the form of the objective function. In classical statistical learning, the task is to minimize the relative entropy $D(p \,\|\, p_\theta)$ between the unknown data distribution p and the model distribution p_θ, parametrized by θ. Since the data distribution is unknown, the objective function is sometimes chosen to be an empirical estimate of the population negative-log-likelihood $\mathcal{L}$ of the model, $\mathcal{L}(\theta) = -\mathbb{E}_{x \sim p} \log p_\theta(x)$. Minimization of the empirical negative-log-likelihood asymptotically minimizes the relative entropy $D(p \,\|\, p_\theta)$. Under additional assumptions (reviewed in the supplementary material), the Fisher Information Matrix approximates the Hessian of $\mathcal{L}$ and the natural gradient can be viewed as an approximate second-order method. In the quantum optimization problem however, there is no direct relationship between the quantum Fisher Information and the curvature of the objective, and the quantum natural gradient is more naturally interpreted as constrained imaginary-time evolution.

In the variational quantum Monte Carlo literature, the Stochastic Reconfiguration algorithm [32] and the time-dependent variational Monte Carlo [4, 5] have been developed for imaginary and real-time evolution, respectively. These algorithms evolve variational states ψ_θ by classically sampling from the Born probability distribution. In the quantum computing literature, an associated real-time evolution algorithm which exploits the imaginary part $\mathrm{Im}[G_{ij}(\theta)]$ of the Quantum Geometric Tensor (7) has been developed in [21] and subsequently demonstrated on quantum hardware in [6]. For details on the geometry of the time-dependent variational principle we refer the reader to [20, Proposition 2.4]. Variational imaginary-time evolution on hybrid quantum-classical devices has been previously investigated in [16, 17, 23]. In these works, the choice of optimization geometry can be shown to correspond to the unit sphere $\mathbb{S}^{N-1} = \{\psi \in \mathbb{C}^N : \|\psi\|_2 = 1\}$, rather than the complex projective space $\mathbb{CP}^{N-1}$ utilized in this paper. Recently, Ref. [36] appeared which considers general evolution of variational density matrices in both real and imaginary time, from a different perspective. By restricting their proposal to pure state projectors (elements of $\mathbb{CP}^{N-1}$) they find an algorithm equivalent to ours.

2.4 Parametric family

In a digital quantum computer the Hilbert space dimension $N = 2^n$ is exponential in the number of qubits $n \in \mathbb{N}$ and the Hilbert space has a natural tensor product decomposition into two-dimensional factors $\mathbb{C}^N = \mathbb{C}^{2^n} = (\mathbb{C}^2)^{\otimes n}$. A parametric family of unitaries relevant to variational quantum algorithms consists of decompositions into products of $L \geq 1$ non-commuting layers of unitaries. Specifically, assume that the variational parameter vector is of the form $\theta = \boldsymbol{\theta}_1 \oplus \cdots \oplus \boldsymbol{\theta}_L \in \mathbb{R}^d$ where $\oplus$ denotes the direct sum (concatenation) and consider a unitary operator acting on n qubits of the following form

$$U_L(\theta) := V_L(\boldsymbol{\theta}_L) W_L \cdots V_1(\boldsymbol{\theta}_1) W_1 \ , \tag{14}$$

where $V_l(\boldsymbol{\theta}_l)$ and W_l are parametric and non-parametric unitary operators, respectively. In particular, all parametric gates within a given layer are assumed to commute. For later convenience, we introduce the following notation for representing subcircuits between layers $l_1 \leq l_2$

$$U_{[l_1:l_2]} := V_{l_2} W_{l_2} \cdots V_{l_1} W_{l_1} \ , \tag{15}$$

so that, for example

$$U_L(\theta) = U_{(l:L]} V_l W_l U_{[1:l)} \ , \tag{16}$$

where $(l{:}L] = [l-1{:}L]$ and $[1{:}l) = [1{:}l-1]$. Moreover, we define the following convenience state for each layer $l \in [L]$:

$$\psi_l := U_{[1:l]} |0\rangle \ . \tag{17}$$

2.5 Quantum Circuit Representation of Quantum Geometric Tensor

Computing the Quantum Geometric Tensor corresponding to a parametrized quantum circuit of the form (14) is a challenging task. In this section we will show, nevertheless, that block-diagonal components of the tensor can be efficiently computed on a quantum computer, producing an approximation to the QGT of the following block-diagonal form:

$$\begin{array}{c|cccc}
 & \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \cdots & \boldsymbol{\theta}_L \\
\boldsymbol{\theta}_1 & \boxed{G^{(1)}} & \mathbf{0} & \cdots & \mathbf{0} \\
\boldsymbol{\theta}_2 & \mathbf{0} & \boxed{G^{(2)}} & \cdots & \mathbf{0} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\boldsymbol{\theta}_L & \mathbf{0} & \mathbf{0} & \cdots & \boxed{G^{(L)}}
\end{array} . \tag{18}$$

Consider the lth layer of the circuit parametrized by $\boldsymbol{\theta}_l$ and let ∂_i and ∂_j denote the partial derivative operators acting with respect to any pair of components of $\boldsymbol{\theta}_l$ (not necessarily distinct). For each layer $l \in [L]$ there exist Hermitian generator matrices K_i and K_j such that,

$$\partial_i V_l(\boldsymbol{\theta}_l) = -\mathrm{i}K_i V_l(\boldsymbol{\theta}_l) \ , \tag{19}$$
$$\partial_j V_l(\boldsymbol{\theta}_l) = -\mathrm{i}K_j V_l(\boldsymbol{\theta}_l) \ , \tag{20}$$

where for notational clarity we have dropped the layer index l from the Hermitian generator K_j, despite the fact that the generators can vary between layers. For simplicity we assume that for all distinct parameters $i \neq j$ within a layer we have $\partial_i K_j = 0$ (this can also serve as the defining property of a layer). Then the commutativity of the partial derivative operators combined with unitarity of $V_l(\boldsymbol{\theta}_l)$ implies that $[K_i, K_j] = 0$.

Using (16), (19) and (20) we compute

$$\partial_j U_L(\theta) = U_{(l:L]}\partial_j V_l(\boldsymbol{\theta}_l) W_l U_{[1:l)} \ , \tag{21}$$
$$= U_{(l:L]}(-\mathrm{i}K_j) V_l(\boldsymbol{\theta}_l) W_l U_{[1:l)} \ , \tag{22}$$
$$= U_{(l:L]}(-\mathrm{i}K_j) U_{[1:l]} \ . \tag{23}$$

Similarly, we have

$$\partial_i U_L(\theta)^\dagger = U^\dagger_{[1:l]}(\mathrm{i}K_i^\dagger) U^\dagger_{(l:L]} \ . \tag{24}$$

It follows from unitarity of the subcircuit $U_{(l:L]}$ and Hermiticity of the generator K_i that

$$\langle \partial_i \psi_\theta | \partial_j \psi_\theta \rangle = \langle \psi_l | K_i K_j | \psi_l \rangle \ . \tag{25}$$

Similarly, the so-called Berry connection is given by

$$\mathrm{i}\langle \psi_\theta | \partial_j \psi_\theta \rangle = \langle \psi_l | K_j | \psi_l \rangle \ . \tag{26}$$

Combining these expressions we obtain the following form for the lth block of the QGT,

$$G^{(l)}_{ij} = \langle \psi_l | K_i K_j | \psi_l \rangle - \langle \psi_l | K_i | \psi_l \rangle \langle \psi_l | K_j | \psi_l \rangle \ . \tag{27}$$

The operator $K_i K_j$ is Hermitian since $[K_i, K_j] = 0$ and thus the block-diagonal approximation of the QGT coincides with the block-diagonal approximation of the Fubini-Study metric tensor,

$$g^{(l)}_{ij} = \mathrm{Re}[G^{(l)}_{ij}] = G^{(l)}_{ij} \ . \tag{28}$$

The preceding calculation demonstrates the following key facts:

1. The lth block of the Fubini-Study metric tensor can be evaluated in terms of quantum expectation values of Hermitian observables.
2. The states ψ_l defining the quantum expectation values are prepared by subcircuits of the full quantum circuit and are thus experimentally realizable.

2.6 Observables

Having identified the states for which the quantum expectation values are to be evaluated, we now turn to characterizing the Hermitian observables defining the quantum measurement.

For simplicity of exposition we focus on one of the most common parametric families encountered in the literature, which consists of tensor products of single-qubit Pauli rotations,

$$V_l(\boldsymbol{\theta}_l) = \bigotimes_{k=1}^{n} R_{P_{l,k}}(\boldsymbol{\theta}_{l,k}) \ . \tag{29}$$

The rotation gates are given by

$$R_{P_{l,k}}(\boldsymbol{\theta}_{l,k}) = \exp\left[-\mathrm{i}\frac{\boldsymbol{\theta}_{l,k}}{2} P_{l,k}\right] \ , \tag{30}$$

where $\boldsymbol{\theta}_{l,k} \in [0, 2\pi)$, and $P_{l,k} \in \{\sigma_x, \sigma_y, \sigma_z\}$ denotes the Pauli matrix which acts on qubit k of layer l. The expressive power of this class of circuits was recently investigated in [8]. In this case the generators are easily shown to be

$$K_i = \frac{1}{2}\mathbb{1}^{[1,i)} \otimes P_{l,i} \otimes \mathbb{1}^{(i,n]} \ , \tag{31}$$

where $\mathbb{1}^{[1,i)} = \bigotimes_{1\leq j<i} \mathbb{1}$. These operators evidently satisfy $[K_i, K_j] = 0$. Since $P_{l,i}^2 = \mathbb{1}$ as a result of the Pauli algebra, it follows that the lth block of the QGT requires the evaluation of the quantum expectation value $\langle \psi_l | \hat{A} | \psi_l \rangle$ where $\hat{A} \in S_l$ belongs to the following set of operators

$$S_l = \{P_{l,i} \mid 1 \leq i \leq n\} \cup \{P_{l,i}P_{l,j} \mid 1 \leq i < j \leq n\}. \tag{32}$$

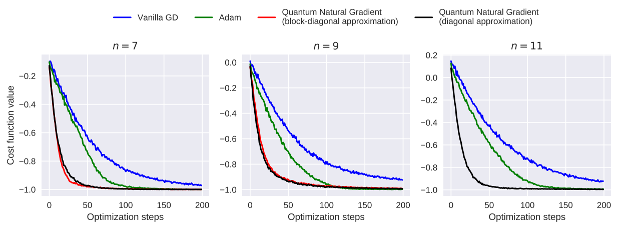


Figure 1: The cost function value for $n = 7, 9, 11$ qubits and $l = 5$ layers as a function of training iteration for four different optimization dynamics. 8192 shots (samples) are used per required expectation value during optimization.

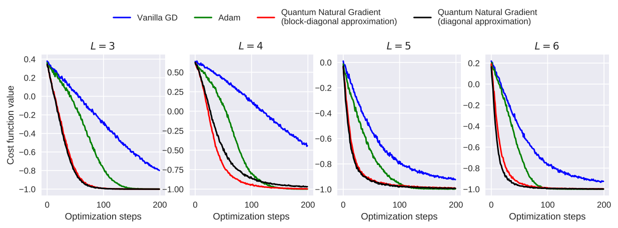


Figure 2: The cost function value for $n = 9$ qubits and $l = 3, 4, 5, 6$ layers as a function of training iteration for four different optimization dynamics. 8192 shots (samples) are used per required expectation value during optimization.

Furthermore, since every operator in S_l commutes, this implies that the number of state preparations is reduced from the naive counting $|S_l| = n(n+1)/2$ to just a single measurement.

3 Numerical Experiments

In order to assess the performance of the Quantum Natural Gradient optimizer, we present in this section numerical experiments comparing the analytical complexity of QNG, assuming oracle access to local data including gradient and Fubini-Study tensor information. These numerical experiments suggest improved oracle complexity compared to existing optimization techniques such as vanilla gradient and Adam optimization. Although the oracle model of complexity is unrealistic because it ignores the added per-iteration complexity of querying the oracle, we provide additional experiments in Sec. A.5 of the supplementary material which demonstrate that the advantage persists when optimizers are compared in terms of both wall time and number of required quantum evaluations. These experiments were performed with the open-source quantum machine learning software library PennyLane [2, 31]. New functionality was added for efficiently computing the block-diagonal $g_{ij}^{(l)}$ and diagonal g_{ii} approximations of the Fubini-Study metric tensor for arbitrary n-qubit parametrized quantum circuits on quantum hardware.

This process involves the following steps:

1. **Represent the circuit as a directed acyclic graph (DAG)**. This allows the parametrized layer structure to be programmatically extracted. Gates which have no dependence on each other (e.g., because they act on different wires) can be grouped together into the same layer.

2. **Determine observables**. For each layer l consisting of m parameters, the generators K_i for each parametrized gate are determined, and a subcircuit preparing ψ_l constructed.

3. **Calculate the lth block of the Fubini-Study metric tensor.**

   (a) **Entire block**: The unitary operation which rotates ψ_l into the shared eigenbasis of $\{K_i | 1 \leq i \leq m\} \cup \{K_i K_j | 1 \leq i, j \leq m\}$ is calculated and applied to the subcircuit, and all qubits measured in the Pauli-Z basis. Classical post-processing is performed to determine $\langle\psi_l|K_i K_j|\psi_l\rangle$, $\langle\psi_l|K_i|\psi_l\rangle$, and $\langle\psi_l|K_j|\psi_l\rangle$ for all $1 \leq i, j \leq m$, and subsequently $g_{ij}^{(l)}$.

   (b) **Diagonal approximation**: The variance $\langle K_i^2\rangle - \langle K_i\rangle^2$ is computed for all $1 \leq i \leq m$, and subsequently the diagonal approximation to the block-diagonal, $g_{ii}^{(l)}$.

Thus, to evaluate the block-diagonal approximation of the Fubini-Study metric tensor on quantum hardware, a single quantum evaluation is performed for each layer in the parametrized quantum circuit. Finally, a Quantum Natural Gradient optimizer was implemented in PennyLane (see [35] for full source code). This optimizer computes the block-diagonal metric tensor $g(\theta)$ at each optimization step (L quantum evaluations), as well as the analytic gradient of the objective function $\nabla\mathcal{L}(\theta)$ via the parameter shift rule [25] ($2d$ quantum evaluations), and updates the parameter values by classically solving the linear system (12). As a result, each optimization step requires $2d + L$ quantum evaluations.

For numerical verification, we considered the circuit of [24], which consists of an initial fixed layer of $R_y(\pi/4)$ gates acting on n qubits, followed by L layers of parametrized Pauli rotations interwoven with 1D ladders of controlled-Z gates, and target Hermitian observable chosen to be the same two-Pauli operator $Z_1 Z_2$ acting on the first and second qubit which has a ground state energy of -1. Starting from the same random initialization of Ref. [24], we optimize the parametrized Pauli rotation gates using vanilla gradient descent, the Adam optimizer, and the Quantum Natural Gradient optimizer, with both the block-diagonal and diagonal approximations. The results are shown in Fig. 1 for $n = 7, 9, 11$ qubits, $L = 5$ layers, and with the optimization performed using 8192 samples per expectation value. In all cases the vanilla gradient descent fails to find the minimum of the objective function, while the Quantum Natural Gradient descent finds the minimum in a small number of iterations, in both block-diagonal and strictly diagonal approximation. In addition, we present a comparison with the Adam optimizer which is a non-local averaging method. In this particular circuit, Adam is capable of finding the minimum but requires a larger number of iterations than the Quantum Natural Gradient. Furthermore, the improvement afforded by the Quantum Natural Gradient optimizer appears more significant with increasing qubit number. Note that for $n = 11$, we do not include the block-diagonal approximation, due to the increased classical overhead associated with numerically computing the shared eigenbasis for each parametrized layer. However, this overhead can likely be negated by implementing the techniques of [7] and [11].

To investigate the effects of variable circuit depth, the numerical experiment was repeated with $n = 9$ qubits, and parametric quantum circuits with $L = 3, 4, 5, 6$ layers. The results are shown in Fig. 2, highlighting that the Quantum Natural Gradient optimizer retains its advantage with increasing circuit depth.

4 Discussion

It is instructive to compare our proposal with existing preconditioning schemes such as Adam. Unlike Adam, which involves some kind of historical averaging, the preconditioning matrix suggested by quantum information geometry does not depend on the specific choice of loss function (Hermitian observable). It is instead a reflection of the local geometry of the quantum state space. In view of these differences it is natural to expect that the benefits provided by the Quantum Natural Gradient are complementary to those of existing stochastic optimization methods such as Adam. It is therefore of interest to perform a detailed ablative study combining these methods, which we leave to future work.

Finally, this paper only considered the relevant geometry for idealized systems described by pure quantum states. In near-term noisy devices it may be of interest to study the relevant geometry for density matrices. The most promising candidate is the Bures metric, which possesses a number of desirable features. In particular, it is the only monotone metric which reduces to both the Fubini-Study metric for pure states and the Fisher information matrix for classical mixtures [28].

A Supplementary Material

In this appendix we employ the Einstein summation convention where summation over repeated indices is implied.

A.1 Real and imaginary parts of Quantum Geometric Tensor

Partially differentiating both sides of the normalization condition $1 = \|\psi_\theta\|^2$ with respect to θ^i gives

$$\left\langle \psi_\theta, \frac{\partial \psi_\theta}{\partial \theta^i} \right\rangle + \left\langle \frac{\partial \psi_\theta}{\partial \theta^i}, \psi_\theta \right\rangle = 0 \ . \tag{33}$$

Partially differentiating again with respect to θ^j gives

$$\left\langle \psi_\theta, \frac{\partial^2 \psi_\theta}{\partial\theta^i \partial\theta^j} \right\rangle + \left\langle \frac{\partial^2 \psi_\theta}{\partial\theta^i \theta^j}, \psi_\theta \right\rangle + \left\langle \frac{\partial \psi_\theta}{\partial\theta^i}, \frac{\partial \psi_\theta}{\partial\theta^j} \right\rangle + \left\langle \frac{\partial \psi_\theta}{\partial\theta^j}, \frac{\partial \psi_\theta}{\partial\theta^i} \right\rangle = 0 \ . \tag{34}$$

Consider the wavefunction in a neighborhood $\theta + \delta\theta$ of $\theta \in \mathbb{R}^d$. Taylor expanding in the displacement vector $\delta\theta$ we obtain,

$$\psi_{\theta+\delta\theta} = \psi_\theta + \frac{\partial \psi_\theta}{\partial\theta^i}\delta\theta^i + \frac{1}{2}\frac{\partial^2 \psi_\theta}{\partial\theta^i \partial\theta^j}\delta\theta^i\delta\theta^j + \cdots \ . \tag{35}$$

Taking the inner product with ψ_θ gives

$$\langle \psi_\theta, \psi_{\theta+\delta\theta} \rangle = 1 + \left\langle \psi_\theta, \frac{\partial \psi_\theta}{\partial\theta^i} \right\rangle \delta\theta^i + \frac{1}{2}\left\langle \psi_\theta, \frac{\partial^2 \psi_\theta}{\partial\theta^i \partial\theta^j} \right\rangle \delta\theta^i\delta\theta^j + \cdots \ . \tag{36}$$

It follows that the fidelity between ψ_θ and $\psi_{\theta+\delta\theta}$ is given to quadratic order in the displacement $\delta\theta$ by,

$$\begin{aligned} |\langle \psi_\theta, \psi_{\theta+\delta\theta} \rangle|^2 &= \langle \psi_\theta, \psi_{\theta+\delta\theta} \rangle \langle \psi_{\theta+\delta\theta}, \psi_\theta \rangle & (37) \\ &= 1 + \left[\left\langle \psi_\theta, \frac{\partial \psi_\theta}{\partial\theta^i} \right\rangle + \left\langle \frac{\partial \psi_\theta}{\partial\theta^i}, \psi_\theta \right\rangle \right] \delta\theta^i + \left[\left\langle \frac{\partial \psi_\theta}{\partial\theta^i}, \psi_\theta \right\rangle \left\langle \psi_\theta, \frac{\partial \psi_\theta}{\partial\theta^j} \right\rangle + \right. \\ &\quad \left. + \frac{1}{2}\left\langle \psi_\theta, \frac{\partial^2 \psi_\theta}{\partial\theta^i \partial\theta^j} \right\rangle + \frac{1}{2}\left\langle \frac{\partial^2 \psi_\theta}{\partial\theta^i \theta^j}, \psi_\theta \right\rangle \right] \delta\theta^i\delta\theta^j + \cdots \ , \\ &= 1 + \left[\left\langle \frac{\partial \psi_\theta}{\partial\theta^i}, \psi_\theta \right\rangle \left\langle \psi_\theta, \frac{\partial \psi_\theta}{\partial\theta^j} \right\rangle - \frac{1}{2}\left(\left\langle \frac{\partial \psi_\theta}{\partial\theta^i}, \frac{\partial \psi_\theta}{\partial\theta^j} \right\rangle + \left\langle \frac{\partial \psi_\theta}{\partial\theta^j}, \frac{\partial \psi_\theta}{\partial\theta^i} \right\rangle \right) \right] \delta\theta^i\delta\theta^j + \cdots \ , & (38) \end{aligned}$$

where we have used (33) and (34). Now use the approximation

$$d^2(P_\psi, P_\phi) = \arccos^2(|\langle \psi, \phi \rangle|) = 1 - |\langle \psi, \phi \rangle|^2 + O((1 - |\langle \psi, \phi \rangle|^2)^2) \ . \tag{39}$$

It follows that the infinitesimal squared distance is given by,

$$d^2(P_{\psi_\theta}, P_{\psi_{\theta+\mathrm{d}\theta}}) = \left[\frac{1}{2}\left(\left\langle \frac{\partial \psi_\theta}{\partial\theta^i}, \frac{\partial \psi_\theta}{\partial\theta^j} \right\rangle + \left\langle \frac{\partial \psi_\theta}{\partial\theta^j}, \frac{\partial \psi_\theta}{\partial\theta^i} \right\rangle \right) - \left\langle \frac{\partial \psi_\theta}{\partial\theta^i}, \psi_\theta \right\rangle \left\langle \psi_\theta, \frac{\partial \psi_\theta}{\partial\theta^j} \right\rangle \right] \mathrm{d}\theta^i \mathrm{d}\theta^j \ . \tag{40}$$

The term multiplying $\frac{1}{2}$ on the right-hand side of (40) is manifestly real. The term multiplying -1 is also real because of (33) which implies

$$\mathrm{Re}\left[\left\langle \psi_\theta, \frac{\partial \psi_\theta}{\partial\theta^i} \right\rangle \right] = 0 \ . \tag{41}$$

It follows that the metric tensor is given by the real part of the QGT,

$$\begin{aligned} d^2(P_{\psi_\theta}, P_{\psi_{\theta+\mathrm{d}\theta}}) &= \mathrm{Re}\left[\left\langle \frac{\partial \psi_\theta}{\partial\theta^i}, \frac{\partial \psi_\theta}{\partial\theta^j} \right\rangle - \left\langle \frac{\partial \psi_\theta}{\partial\theta^i}, \psi_\theta \right\rangle \left\langle \psi_\theta, \frac{\partial \psi_\theta}{\partial\theta^j} \right\rangle \right] \mathrm{d}\theta^i \mathrm{d}\theta^j \ , & (42) \\ &= \mathrm{Re}\left[G_{ij}(\theta) \right] \mathrm{d}\theta^i \mathrm{d}\theta^j \ . & (43) \end{aligned}$$

For completeness, the imaginary part of the QGT is given by

$$\mathrm{Im}[G_{ij}(\theta)] = -\frac{1}{2}\left[\frac{\partial}{\partial\theta_i}A_j(\theta) - \frac{\partial}{\partial\theta_j}A_i(\theta) \right] \ , \tag{44}$$

where $A_i(\theta)$ is the Berry connection,

$$A_i(\theta) = \mathrm{i}\left\langle \psi_\theta, \frac{\partial \psi_\theta}{\partial\theta^i} \right\rangle \ . \tag{45}$$

A.2 Relationship with imaginary-time evolution

Consider the imaginary-time evolution operator $e^{-H\delta\tau}$ generated by the Hermitian operator H where $\delta\tau \in \mathbb{R}$. Let $P_{\psi_\theta} = |\psi_\theta\rangle\langle\psi_\theta|$ denote the projector onto the one-dimensional subspace spanned by the unit vector ψ_θ and let $\bar{\psi}_\theta = e^{-H\delta\tau}\psi_\theta$. Then the projected imaginary-time evolution is defined by,

$$\arg\min_{\delta\theta\in\mathbb{R}^d} \left\|\bar{\psi}_\theta - P_{\psi_{\theta+\delta\theta}}\bar{\psi}_\theta\right\|^2 = \arg\max_{\delta\theta\in\mathbb{R}^d} \left|\left\langle\bar{\psi}_\theta, \psi_{\theta+\delta\theta}\right\rangle\right|^2 \quad , \tag{46}$$

where we used the normalization of $\psi_{\theta+\delta\theta}$. We have

$$\langle\bar{\psi}_\theta, \psi_{\theta+\delta\theta}\rangle = \langle\bar{\psi}_\theta, \psi_\theta\rangle + \left\langle\bar{\psi}_\theta, \frac{\partial\psi_\theta}{\partial\theta^i}\right\rangle\delta\theta^i + \frac{1}{2}\left\langle\bar{\psi}_\theta, \frac{\partial^2\psi_\theta}{\partial\theta^i\partial\theta^j}\right\rangle\delta\theta^i\delta\theta^j + \cdots \quad . \tag{47}$$

So Taylor expanding $|\langle\bar{\psi}_\theta, \psi_{\theta+\delta\theta}\rangle|^2$ to quadratic order in the displacement gives,

$$\begin{aligned}|\langle\bar{\psi}_\theta, \psi_{\theta+\delta\theta}\rangle|^2 = |\langle\bar{\psi}_\theta, \psi_\theta\rangle|^2 + \left[\langle\psi_\theta, \bar{\psi}_\theta\rangle\left\langle\bar{\psi}_\theta, \frac{\partial\psi_\theta}{\partial\theta^i}\right\rangle + \left\langle\frac{\partial\psi_\theta}{\partial\theta^i}, \bar{\psi}_\theta\right\rangle\langle\bar{\psi}_\theta, \psi_\theta\rangle\right]\delta\theta^i + \\ + \left[\left\langle\frac{\partial\psi_\theta}{\partial\theta^i}, \bar{\psi}_\theta\right\rangle\left\langle\bar{\psi}_\theta, \frac{\partial\psi_\theta}{\partial\theta^j}\right\rangle + \frac{1}{2}\langle\psi_\theta, \bar{\psi}_\theta\rangle\left\langle\bar{\psi}_\theta, \frac{\partial^2\psi_\theta}{\partial\theta^i\partial\theta^j}\right\rangle + \frac{1}{2}\left\langle\frac{\partial^2\psi_\theta}{\partial\theta^i\theta^j}, \bar{\psi}_\theta\right\rangle\langle\bar{\psi}_\theta, \psi_\theta\rangle\right]\delta\theta^i\delta\theta^j + \cdots \quad .\end{aligned} \tag{48}$$

Expanding the exponential $e^{-H\delta\tau}$ in $\delta\tau$ and neglecting cubic-order terms in the multi-variable Taylor expansion in $\delta\theta$ and $\delta\tau$,

$$|\langle\bar{\psi}_\theta, \psi_{\theta+\delta\theta}\rangle|^2 = |\langle\bar{\psi}_\theta, \psi_\theta\rangle|^2 - \left[\left\langle\frac{\partial\psi_\theta}{\partial\theta^i}, H\psi_\theta\right\rangle + \left\langle H\psi_\theta, \frac{\partial\psi_\theta}{\partial\theta^i}\right\rangle\right]\delta\theta^i\delta\tau - \mathrm{Re}[G_{ij}(\theta)]\delta\theta^i\delta\theta^j + \cdots \quad , \tag{49}$$

where we have made use of (33) and (34). The first-order optimality condition $0 = \frac{\partial}{\partial\delta\theta^i}|\langle\bar{\psi}_\theta, \psi_{\theta+\delta\theta}\rangle|^2$, at lowest order in $\delta\theta$ and $\delta\tau$, thus gives

$$\begin{aligned} 0 &= -\left[\left\langle\frac{\partial\psi_\theta}{\partial\theta^i}, H\psi_\theta\right\rangle + \left\langle H\psi_\theta, \frac{\partial\psi_\theta}{\partial\theta^i}\right\rangle\right]\delta\tau - 2\,\mathrm{Re}[G_{ij}(\theta)]\delta\theta^j + \cdots \quad , & (50)\\ &= -\frac{1}{2}\left[\left\langle\frac{\partial\psi_\theta}{\partial\theta^i}, H\psi_\theta\right\rangle + \left\langle\psi_\theta, H\frac{\partial\psi_\theta}{\partial\theta^i}\right\rangle\right]\delta\tau - \mathrm{Re}[G_{ij}(\theta)]\delta\theta^j \cdots \quad , & (51)\\ &= -\frac{\partial}{\partial\theta^i}\mathcal{L}(\theta)\delta\tau \quad - \mathrm{Re}[G_{ij}(\theta)]\delta\theta^j + \cdots , & (52)\end{aligned}$$

where $\mathcal{L}(\theta) = \frac{1}{2}\langle\psi_\theta, H\psi_\theta\rangle$ and we have used $H = H^\dagger$. In the limit $\delta\tau \to 0$ we obtain the following system of ordinary differential equations,

$$g(\theta(\tau))\dot{\theta}(\tau) = -\nabla\mathcal{L}(\theta(\tau)) \quad . \tag{53}$$

A.3 Relationship with curvature of objective

Let $p_\theta \succ 0$ be a parametric family probability distributions over $[N]$, indexed by $\theta \in \mathbb{R}^d$. Differentiating both sides of the expression $1 = \mathbb{E}_{x\sim p_\theta}[1]$ we find the identity

$$0 = \mathop{\mathbb{E}}_{x\sim p_\theta}\left[\frac{\partial\log p_\theta(x)}{\partial\theta^i}\right] \quad , \tag{54}$$

and differentiating once again gives

$$0 = \mathop{\mathbb{E}}_{x\sim p_\theta}\left[\frac{\partial\log p_\theta(x)}{\partial\theta^i}\frac{\partial\log p_\theta(x)}{\partial\theta^j} + \frac{\partial^2\log p_\theta(x)}{\partial\theta^i\partial\theta^j}\right] \quad . \tag{55}$$

The Fisher Information Matrix can thus be expressed as

$$I_{ij}(\theta) = -\mathop{\mathbb{E}}_{x\sim p_\theta}\left[\frac{\partial^2 \log p_\theta(x)}{\partial\theta^i\partial\theta^j}\right] \ . \tag{56}$$

Now suppose that $p \succ 0$ is an unknown probability vector. Recall that the relative entropy between p and p_θ can be expressed as

$$D(p \,\|\, p_\theta) = \mathcal{L}(\theta) - S(p) \ , \tag{57}$$

where $S(p)$ is the entropy of p and $\mathcal{L}(\theta)$ is the population loss given by of expected negative-log-likelihood of the model,

$$\mathcal{L}(\theta) = -\mathop{\mathbb{E}}_{x\sim p} \log p_\theta(x) \ . \tag{58}$$

The Hessian of the loss is given by

$$\frac{\partial^2\mathcal{L}(\theta)}{\partial\theta^i\partial\theta^j} = -\mathop{\mathbb{E}}_{x\sim p}\left[\frac{\partial^2 \log p_\theta(x)}{\partial\theta^i\partial\theta^j}\right] \ . \tag{59}$$

Introducing the shorthand $f_{ij}(x) = -\frac{\partial^2 \log p_\theta(x)}{\partial\theta^i\partial\theta^j}$ and using Hölder's inequality we obtain

$$\left|\frac{\partial^2\mathcal{L}(\theta)}{\partial\theta^i\partial\theta^j} - I_{ij}(\theta)\right| = \left|\mathop{\mathbb{E}}_{x\sim p}[f_{ij}(x)] - \mathop{\mathbb{E}}_{x\sim p_\theta}[f_{ij}(x)]\right| \ , \tag{60}$$

$$= |\langle p - p_\theta, f_{ij}\rangle| \ , \tag{61}$$

$$\leq \|p - p_\theta\|_1 \|f_{ij}\|_\infty \ . \tag{62}$$

Finally, using Pinsker's inequality $D(p \,\|\, p_\theta) \geq \frac{1}{2}\|p - p_\theta\|_1^2$ we obtain

$$\left|\frac{\partial^2\mathcal{L}(\theta)}{\partial\theta^i\partial\theta^j} - I_{ij}(\theta)\right| \leq \max_{x\in[N]}\left|\frac{\partial^2 \log p_\theta(x)}{\partial\theta^i\partial\theta^j}\right| \sqrt{2[\mathcal{L}(\theta) - S(p)]} \ . \tag{63}$$

Thus the error in approximation is controlled by the loss deficit $\mathcal{L}(\theta) - S(p) \geq 0$ and the curvature of the likelihood function.

A.4 Relationship with classical Fisher information

Let $\{|x\rangle : x \in [N]\}$ be an orthonormal basis for $\mathbb{C}^N$ and suppose $p_\theta(x)$ is a parametric family of probability distributions on the finite set $[N]$. Define the following parametric family of quantum states

$$\psi_\theta = \sum_{x\in[N]} \sqrt{p_\theta(x)}|x\rangle \ . \tag{64}$$

Then by the chain rule

$$\frac{\partial\psi_\theta}{\partial\theta^i} = \frac{1}{2}\sum_{x\in[N]} \frac{1}{\sqrt{p_\theta(x)}}\frac{\partial p_\theta(x)}{\partial\theta^i}|x\rangle \ . \tag{65}$$

Thus the Berry connection for this family of states vanishes

$$\left\langle \psi_\theta, \frac{\partial\psi_\theta}{\partial\theta^i}\right\rangle = \frac{1}{2}\sum_{x\in[N]}\sum_{x'\in[N]} \frac{\sqrt{p_\theta(x')}}{\sqrt{p_\theta(x)}}\frac{\partial p_\theta(x)}{\partial\theta^i}\langle x'|x\rangle \ , \tag{66}$$

$$= \frac{1}{2}\sum_{x\in[N]}\frac{\partial p_\theta(x)}{\partial\theta^i} \ , \tag{67}$$

$$= \frac{1}{2}\frac{\partial}{\partial\theta^i}\sum_{x\in[N]} p_\theta(x) \ , \tag{68}$$

$$= \frac{1}{2}\frac{\partial}{\partial\theta^i}1 \ , \tag{69}$$

$$= 0 \ , \tag{70}$$

where we used the orthonormality of the basis $\langle x'|x\rangle = \delta_{xx'}$. The QGT is thus given by

$$G_{ij}(\theta) = \left\langle \frac{\partial\psi_\theta}{\partial\theta^i}, \frac{\partial\psi_\theta}{\partial\theta^j} \right\rangle \ , \tag{71}$$

$$= \frac{1}{4} \sum_{x\in[N]} \sum_{x'\in[N]} \frac{1}{\sqrt{p_\theta(x)p_\theta(x')}} \frac{\partial p_\theta(x)}{\partial\theta^i} \frac{\partial p_\theta(x')}{\partial\theta^j} \langle x'|x\rangle \ , \tag{72}$$

$$= \frac{1}{4} \sum_{x\in[N]} \frac{1}{p_\theta(x)} \frac{\partial p_\theta(x)}{\partial\theta^i} \frac{\partial p_\theta(x)}{\partial\theta^j} \ , \tag{73}$$

$$= \frac{1}{4} \sum_{x\in[N]} p_\theta(x) \frac{\partial \log p_\theta(x)}{\partial\theta^i} \frac{\partial \log p_\theta(x)}{\partial\theta^j} \ , \tag{74}$$

$$= \frac{1}{4} I_{ij}(\theta) \ . \tag{75}$$

A.5 Additional experiments and figures

In the following section, we present some additional plots comparing the optimization dynamics of the Quantum Natural Gradient to various other optimization strategies, including gradient descent-based (standard or vanilla gradient descent, Adam) and gradient-free (COBYLA, Nelder-Mead) strategies. In addition, we include in this comparison a version of the Adam optimizer modified to use the natural gradient in its parameter update step. While it remains difficult to make direct comparisons between the (non-local) Adam optimizer and the Quantum Natural Gradient optimizer, it is instructive to compare the behaviour of the Adam optimizer when using the natural gradient as opposed to the standard gradient.

The results of these additional experiments are shown in Fig. 3, highlighting each optimization strategy for fixed number of shots and increasing circuit depth, and Fig. 4, for fixed circuit depth but varying number of samples used to compute circuit expectation values. In both experiments, the same circuit architecture is used as in Sec. 3. Here, we compare the progress of each optimization strategy against the number of iterations, total computational wall time (note that this includes the wall time required to perform all quantum simulations), and number of quantum evaluations. In particular, we note that:

- The Quantum Natural Gradient continues to outperform both vanilla gradient descent and Adam optimization.
- The diagonal approximation and the block diagonal approximation to the Quantum Geometric Tensor provide comparable results when used with the Quantum Natural Gradient, however the diagonal approximation results in significantly reduced overall wall time—comparable to vanilla gradient descent—due to the decrease in classical processing overhead.
- Comparison with gradient-free techniques is more difficult; within the same number of iterations, both gradient-free techniques failed to find the local minimum. However, COBYLA and Nelder-Mead required significantly fewer number of quantum evaluations over these iterations.
- The inclusion of the natural gradient within the Adam optimizer parameter update step appears to provide some benefit, with the modified Adam optimizer converging to the local minimum in fewer iterations than the standard Adam optimizer.

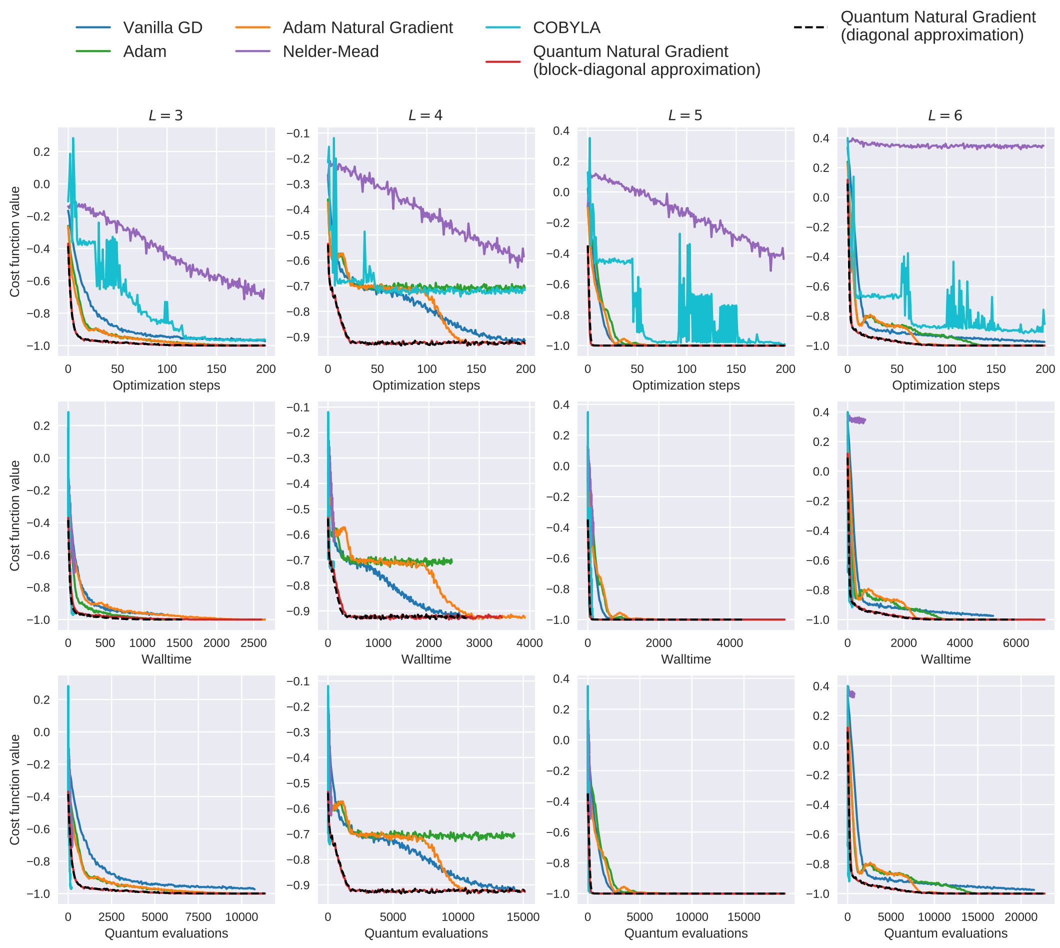


Figure 3: The cost function value for $n = 9$ qubits and $l = 3, 4, 5, 6$ layers as a function of training iteration (top), wall time (middle), and number of quantum evaluations (bottom) for various optimization techniques; vanilla gradient descent (blue), Adam (green), Adam modified to use the natural gradient (orange), Nelder-Mead (purple), COBYLA (cyan), the Quantum Natural Gradient (block-diagonal approximation) (red), and the Quantum Natural Gradient (diagonal approximation) (black, dashed). 8192 shots (samples) are used per required expectation value during optimization, with a learning rate of 0.01 where applicable.

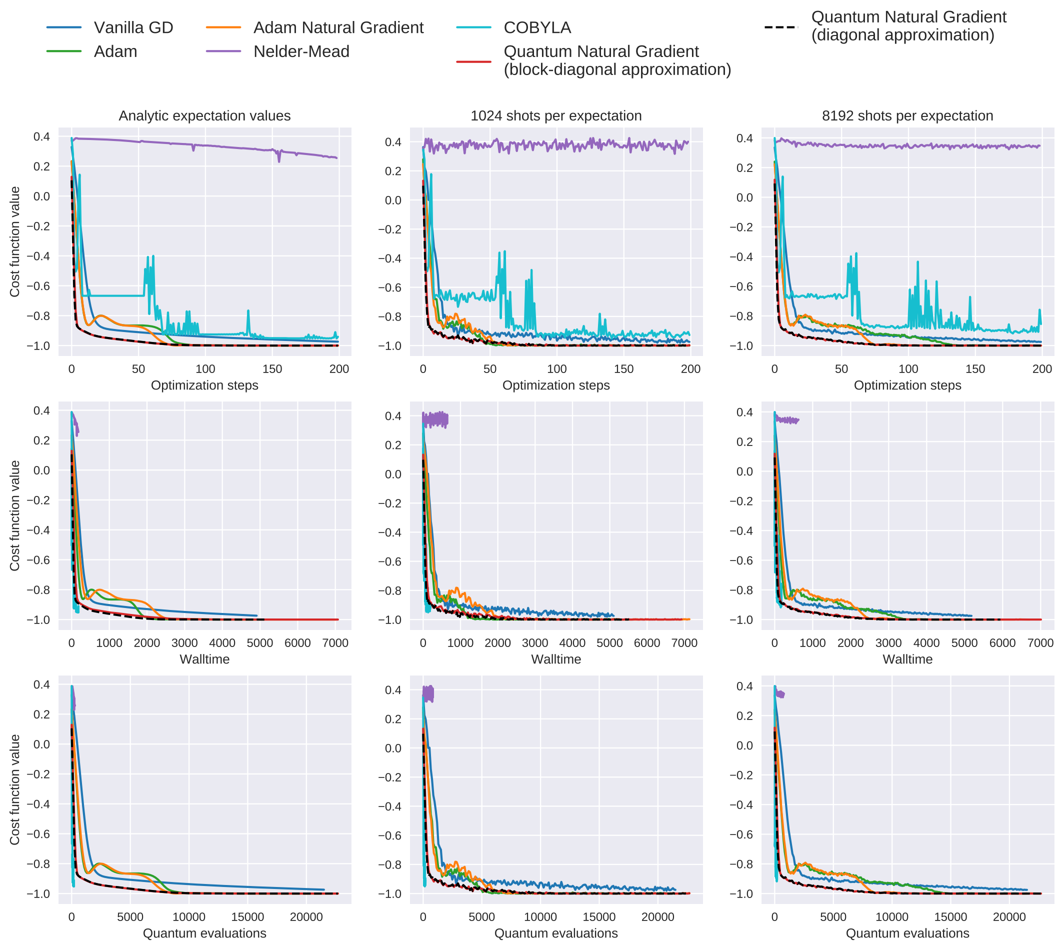


Figure 4: The cost function value for $n = 9$ qubits, $l = 6$ layers as a function of training iteration (top), wall time (middle), and number of quantum evaluations (bottom) for various optimization techniques; vanilla gradient descent (blue), Adam (green), Adam modified to use the natural gradient (orange), Nelder-Mead (purple), COBYLA (cyan), the Quantum Natural Gradient (block-diagonal approximation) (red), and the Quantum Natural Gradient (diagonal approximation) (black, dashed). The optimization is performed using analytic expectation values (left), 1024 shots (samples) per expectation value (center), and 8192 shots per expectation value (right). The learning rate is 0.01 where applicable.

References

[1] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2): 251–276, 1998. DOI: 10.1162/089976698300017746.

[2] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M. Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, Keri McKiernan, Johannes Jakob Meyer, Zeyue Niu, Antal Szàva, and Nathan Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

[3] Marin Bukov, Dries Sels, and Anatoli Polkovnikov. Geometric speed limit of accessible many-body state preparation. *Physical Review X*, 9(1):011034, 2019. DOI: 10.1103/PhysRevX.9.011034.

[4] Giuseppe Carleo, Federico Becca, Marco Schiró, and Michele Fabrizio. Localization and glassy dynamics of many-body quantum systems. *Scientific reports*, 2:243, 2012. DOI: 10.1038/srep00243.

[5] Giuseppe Carleo, Federico Becca, Laurent Sanchez-Palencia, Sandro Sorella, and Michele Fabrizio. Light-cone effect and supersonic correlations in one-and two-dimensional bosonic superfluids. *Physical Review A*, 89(3):031602, 2014. DOI: 10.1103/PhysRevA.89.031602.

[6] Ming-Cheng Chen, Ming Gong, Xiao-Si Xu, Xiao Yuan, Jian-Wen Wang, Can Wang, Chong Ying, Jin Lin, Yu Xu, Yulin Wu, et al. Demonstration of adiabatic variational quantum computing with a superconducting quantum coprocessor. *arXiv preprint arXiv:1905.03150*, 2019.

[7] Ophelia Crawford, Barnaby van Straaten, Daochen Wang, Thomas Parks, Earl Campbell, and Stephen Brierley. Efficient quantum measurement of pauli operators. *arXiv preprint arXiv:1908.06942*, 2019.

[8] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, and Dacheng Tao. The expressive power of parameterized quantum circuits. *arXiv preprint arXiv:1810.11922*, 2018.

[9] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*, 2018.

[10] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[11] Pranav Gokhale, Olivia Angiuli, Yongshan Ding, Kaiwen Gui, Teague Tomesh, Martin Suchara, Margaret Martonosi, and Frederic T Chong. Minimizing state preparations in variational quantum eigensolver by partitioning into commuting families. *arXiv preprint arXiv:1907.13623*, 2019.

[12] Gian Giacomo Guerreschi and Mikhail Smelyanskiy. Practical optimization for hybrid quantum-classical algorithms. *arXiv preprint arXiv:1701.01450*, 2017.

[13] Aram Harrow and John Napp. Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms. *arXiv preprint arXiv:1901.05374*, 2019.

[14] William James Huggins, Piyush Patil, Bradley Mitchell, K Birgitta Whaley, and Miles Stoudenmire. Towards quantum machine learning with tensor networks. *Quantum Science and Technology*, 4:024001, 2018. DOI: 10.1088/2058-9565/aaea94.

[15] Stanislaw Jastrzebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*, 2017.

[16] Tyson Jones and Simon C Benjamin. Quantum compilation and circuit optimisation via energy dissipation. *arXiv preprint arXiv:1811.03147*, 2018.

[17] Tyson Jones, Suguru Endo, Sam McArdle, Xiao Yuan, and Simon C Benjamin. Variational quantum algorithms for discovering hamiltonian spectra. *Physical Review A*, 99(6):062304, 2019. DOI: 10.1103/PhysRevA.99.062304.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Michael Kolodrubetz, Dries Sels, Pankaj Mehta, and Anatoli Polkovnikov. Geometry and non-adiabatic response in quantum and classical systems. *Physics Reports*, 697:1–87, 2017. DOI: 10.1016/j.physrep.2017.07.001.

[20] PH Kramer and Marcos Saraceno. *Geometry of the time-dependent variational principle in quantum mechanics*. Springer, 1981. DOI: 10.1007/3-540-10271-X_317.

[21] Ying Li and Simon C Benjamin. Efficient variational quantum simulator incorporating active error minimization. *Physical Review X*, 7(2):021050, 2017. DOI: 10.1103/PhysRevX.7.021050.

[22] Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-rao metric, geometry, and complexity of neural networks. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 888–896, 2019.

[23] Sam McArdle, Tyson Jones, Suguru Endo, Ying Li, Simon C Benjamin, and Xiao Yuan. Variational ansatz-based quantum simulation of imaginary time evolution. *npj Quantum Information*, 5(1):1–6, 2019. DOI: 10.1038/s41534-019-0187-2.

[24] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature communications*, 9(1): 4812, 2018. DOI: 10.1038/s41467-018-07090-4.

[25] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018. DOI: 10.1103/PhysRevA.98.032309.

[26] Behnam Neyshabur, Ruslan R Salakhutdinov, and Nati Srebro. Path-SGD: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2422–2430, 2015.

[27] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:4213, 2014. DOI: 10.1038/ncomms5213.

[28] Dénes Petz. Information-geometry of quantum states. In *Quantum Probability Communications: Volume X*, pages 135–157. World Scientific, 1998. DOI: 10.1142/9789812816054_0006.

[29] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, 2018. DOI: 10.22331/q-2018-08-06-79.

[30] Maria Schuld, Alex Bocharov, Krysta Svore, and Nathan Wiebe. Circuit-centric quantum classifiers. *arXiv preprint arXiv:1804.00633*, 2018. DOI: 10.1103/PhysRevA.101.032308.

[31] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3):032331, 2019. DOI: 10.1103/PhysRevA.99.032331.

[32] Sandro Sorella, Michele Casula, and Dario Rocca. Weak binding between two aromatic rings: Feeling the van der waals attraction by quantum monte carlo methods. *The Journal of Chemical Physics*, 127(1):014105, 2007. DOI: 10.1063/1.2746035.

[33] James C Spall et al. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992. DOI: 10.1109/9.119632.

[34] F Wilczek and A Shapere. Geometric phases in physics. *Geometric Phases In Physics. Series: Advanced Series in Mathematical Physics, ISBN: 978-9971-5-0621-6. WORLD SCIENTIFIC, Edited by F Wilczek and A Shapere, vol. 5*, 5, 1989. DOI: 10.1142/0613.

[35] Xanadu Quantum Technologies. PennyLane source code. `https://github.com/XanaduAI/pennylane`, 2019. [Online; accessed 3-Mar-2020].

[36] Xiao Yuan, Suguru Endo, Qi Zhao, Ying Li, and Simon C Benjamin. Theory of variational quantum simulation. *Quantum*, 3:191, 2019. DOI: 10.22331/q-2019-10-07-191.

EEE606 Project : SAMPLE CODE

Submitted by : Aradhita Sharma

Python language is used for this project.
Import necessary libraries (matplotlib, pennylane, sklearn, seaborn, pandas).
Develop two variational quantum circuits (adaptive system) with 2 and 3 qubits using the functions : Cost_fn_2qubits() and cost_fn_3qubits() , which returns the cost using qml.expval() function.
Qml.draw() for drawing the variational circuit.

<u>PART 1 : Convergence for different optimizations :</u>

```
Optimizers = [VGD, ADAM, QNG]
For i in optimizers :
    For j in range(0,500) # 500 iteration steps
        • Give initial params to cost_fn
        • Obtain the updated params by evaluating the cost_fn gradient optimizers(i)
        • Append the cost_fn error value to separate cost arrays for different optimizers
```

Plot the cost_fn convergence curves for different optimizers.

<u>PART 2 : Binary Classification :</u>

Import dataset from sklearn library and split the dataset into training and testing data (80-20%).

<u>Quantum binary Classification :</u>

- Construct quantum circuit with angle encoding of input values, and adaptive parameters as entangled layers, which is the variational classifier which returns +1 and -1 as two different class outputs.
- Split the data into 5 batches.
- Run the above convergence for 5 epochs with adam optimization.
- Test the model weights on testing data, and obtain the accuracy and confusion matrix.

<u>Classical binary Classification :</u>

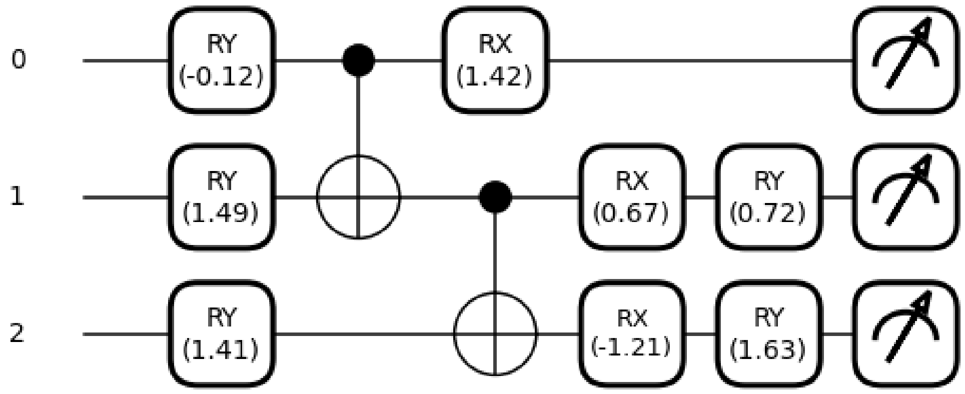
- Input values are passed through the model with 4 inputs and 1 binary output,
- Sigmoid function is the activation function.
- Weights are updated for 5 epochs with Vanilla gradient descent optimization (calculating the gradient of sigmoid).
- Test the model weights on testing data, and obtain the accuracy and confusion matrix.

```
!pip install pennylane

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

dev_2qubits = qml.device("default.qubit", wires=2)
@qml.qnode(dev_2qubits)
def cost_fn_2qubits(params):
    # |psi_0>: state preparation
    qml.RY(np.pi / 4, wires=0)
    qml.RY(np.pi / 3, wires=1)
    # V0(theta0, theta1): Parametrized layer 0
    qml.RZ(params[0], wires=0)
    qml.RZ(params[1], wires=1)
    # W1: non-parametrized gates
    qml.CNOT(wires=[0, 1])
    # V_1(theta2, theta3): Parametrized layer 1
    qml.RY(params[2], wires=0)
    qml.RX(params[3], wires=1)
    # W2: non-parametrized gates
    qml.CNOT(wires=[0, 1])
    return qml.expval(qml.PauliY(0))
params = np.array([0.432, -0.123, 0.543, 0.233])

dev_3qubits = qml.device("default.qubit", wires=3)
def circuit_3qubits(params, wires=0): # circuit for error, with
respect to b
    # |psi_0>: state preparation
    # V0(theta0, theta1): Parametrized layer 0
    qml.RY(params[0], wires=0)
    qml.RY(params[1], wires=1)
    qml.RY(params[2], wires=2)
    # W1: non-parametrized gates
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    # V_1(theta2, theta3): Parametrized layer 1
    qml.RX(params[3], wires=0)
    qml.RX(params[4], wires=1)
    qml.RX(params[5], wires=2)
    # W2: non-parametrized gates
    qml.RY(params[6], wires=1)
    qml.RY(params[7], wires=2)

coeffs_3qubits =[-0.124, 1.489, 1.409, 1.417, 0.671, -1.207, 0.717,
1.630 ];
init_params = np.array([0,0,0,0,0,0,0,0], requires_grad=True)
obs_3qubits = [qml.PauliZ(0),
qml.PauliZ(1),qml.PauliZ(2),qml.PauliZ(0),qml.PauliZ(1),
qml.PauliZ(2),qml.PauliX(0),qml.PauliX(1) ]
```

```
H_3qubits = qml.Hamiltonian(coeffs_3qubits, obs_3qubits)

@qml.qnode(dev_3qubits)
def cost_fn_3qubits(params):
    circuit_3qubits(params)
    return qml.expval(H_3qubits)


dev_3qubits = qml.device("default.qubit", wires=3)
@qml.qnode(dev_3qubits)
def circuit_3qubits(params, wires=3): # circuit for error, with
respect to b
    # |psi_0>: state preparation
    # V0(theta0, theta1): Parametrized layer 0
    qml.RY(params[0], wires=0)
    qml.RY(params[1], wires=1)
    qml.RY(params[2], wires=2)
    # W1: non-parametrized gates
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    # V_1(theta2, theta3): Parametrized layer 1
    qml.RX(params[3], wires=0)
    qml.RX(params[4], wires=1)
    qml.RX(params[5], wires=2)
    # W2: non-parametrized gates
    qml.RY(params[6], wires=1)
    qml.RY(params[7], wires=2)
    return qml.expval(H_3qubits)

params=[-0.124, 1.489, 1.409, 1.417, 0.671, -1.207, 0.717, 1.630 ];

fig, ax = qml.draw_mpl(circuit_3qubits, decimals=2)(params)
plt.show()
```

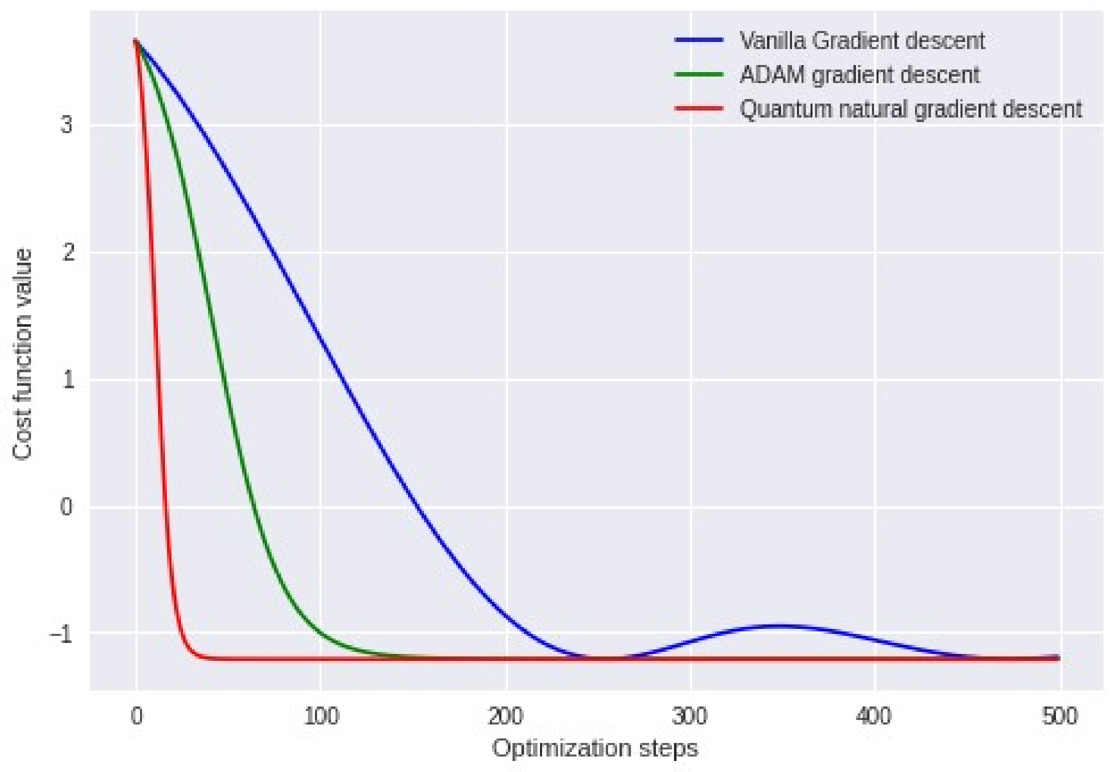
```
/usr/lib/python3.8/_collections_abc.py:832:
MatplotlibDeprecationWarning:
The datapath rcparam was deprecated in Matplotlib 3.2.1 and will be
removed two minor releases later.
  self[key] = other[key]
/usr/lib/python3.8/_collections_abc.py:832:
MatplotlibDeprecationWarning:
The savefig.frameon rcparam was deprecated in Matplotlib 3.1 and will
be removed in 3.3.
  self[key] = other[key]
/usr/lib/python3.8/_collections_abc.py:832:
MatplotlibDeprecationWarning:
The text.latex.unicode rcparam was deprecated in Matplotlib 3.0 and
will be removed in 3.2.
  self[key] = other[key]
/usr/lib/python3.8/_collections_abc.py:832:
```

```
MatplotlibDeprecationWarning:
The verbose.fileo rcparam was deprecated in Matplotlib 3.1 and will be
removed in 3.3.
  self[key] = other[key]
/usr/lib/python3.8/_collections_abc.py:832:
MatplotlibDeprecationWarning:
The verbose.level rcparam was deprecated in Matplotlib 3.1 and will be
removed in 3.3.
  self[key] = other[key]
```



```python
def convergence(opt, cost_fn, init_params,  max_iterations,
step_size):
  params = init_params
  param_history = [params]
  cost_history = []
  if(opt == "VGD"):
    opt = qml.GradientDescentOptimizer(stepsize=step_size)
  elif(opt == "QNG"):
    opt = qml.QNGOptimizer(stepsize=step_size, approx="block-diag")
  elif(opt == "ADAM"):
    opt = qml.AdamOptimizer(stepsize=step_size, beta1=0.99,
beta2=0.99, eps=1e-08)

  for n in range(max_iterations):
     params, prev_energy = opt.step_and_cost(cost_fn, params)
     param_history.append(params)
     cost_history.append(prev_energy)
     energy = cost_fn(params)
  return cost_history, param_history
```

```
max_iterations = 500
step_size = 0.01
'''
QNGcost, QNGparams = convergence("QNG", cost_fn_2qubits, init_params,
max_iterations, step_size)
VGDcost, VGDparams = convergence("VGD", cost_fn_2qubits, init_params,
max_iterations, step_size)
ADAMcost, ADAMparams = convergence("ADAM", cost_fn_2qubits,
init_params,  max_iterations, step_size)
'''
QNGcost, QNGparams = convergence("QNG", cost_fn_3qubits, init_params,
max_iterations, step_size)
VGDcost, VGDparams = convergence("VGD", cost_fn_3qubits, init_params,
max_iterations, step_size)
ADAMcost, ADAMparams = convergence("ADAM", cost_fn_3qubits,
init_params,  max_iterations, step_size)

plt.style.use("seaborn")
plt.plot(VGDcost, "b", label="Vanilla Gradient descent")
plt.plot(ADAMcost, "g", label="ADAM gradient descent")
plt.plot(QNGcost, "r", label="Quantum natural gradient descent")

plt.ylabel("Cost function value")
plt.xlabel("Optimization steps")
plt.legend()
plt.show()
```



PART 2 : Binary Classification

```
from itertools import chain
from sklearn import datasets
from sklearn.utils import shuffle
from sklearn.preprocessing import minmax_scale
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics

import pennylane as qml
from pennylane import numpy as np
from pennylane.templates.embeddings import AngleEmbedding
from pennylane.templates.layers import StronglyEntanglingLayers
from pennylane.optimize import GradientDescentOptimizer
import seaborn
# load the dataset
iris = datasets.load_iris()
# shuffle the data
X, y = shuffle(iris.data, iris.target, random_state=0)
# select only 2 first classes from the data
X = X[y<=1]
y = y[y<=1]
# normalize data
X = minmax_scale(X, feature_range=(0, np.pi))
# split data into train+validation and test
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, 
test_size=0.2)
# split into train and validation
X_train, X_validation, y_train, y_validation = 
train_test_split(X_train_val, y_train_val, test_size=0.20)
print(np.shape(X_train), y_train)
```

```
(64, 4) [0 0 0 1 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 
1 0 0 1 1 1
 1 1 1 0 1 0 0 1 0 1 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0]
```

Quantum Binary classification

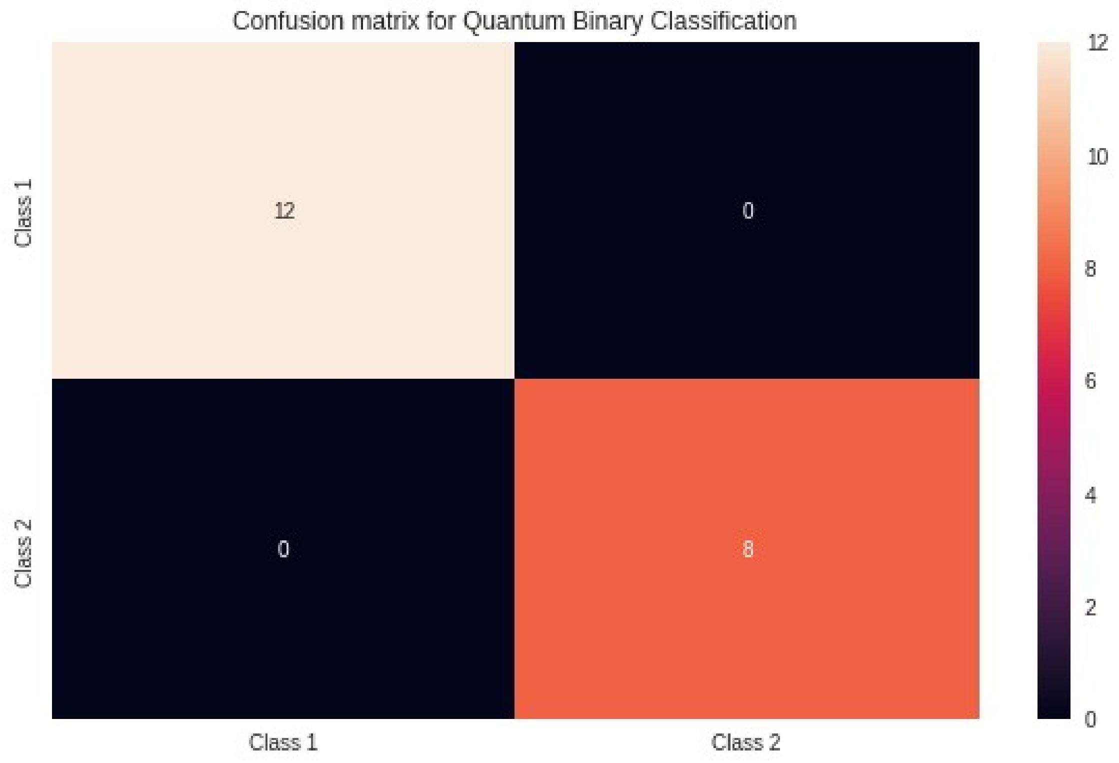
```
#--------- Quantum Part
# number of qubits is equal to the number of features
n_qubits = X.shape[1]
# quantum device handle
dev = qml.device("default.qubit", wires=n_qubits)
# quantum circuit
@qml.qnode(dev)
def circuit(weights, x=None):
    AngleEmbedding(x, wires = range(n_qubits))
    StronglyEntanglingLayers(weights, wires = range(n_qubits))
    return qml.expval(qml.PauliZ(0))
# variational quantum classifier
def variational_classifier(theta, x=None):
    weights = theta[0]
    bias = theta[1]
    return circuit(weights, x=x) + bias
# cost function
def cost(theta, X, expectations):
    e_predicted = \
        np.array([variational_classifier(theta, x=x) for x in X])
    loss = np.mean((e_predicted - expectations)**2)
    return loss
# number of quantum layers
n_layers = 3
n_wires = n_qubits
# convert classes to expectations: 0 to -1, 1 to +1
e_train = np.empty_like(y_train)
e_train[y_train == 0] = -1
e_train[y_train == 1] = +1
# select learning batch size
batch_size = 5
# calculate numbe of batches
batches = len(X_train) // batch_size
# select number of epochs
n_epochs = 5
# draw random quantum node weights
param_shape = StronglyEntanglingLayers.shape(n_layers=n_layers,
n_wires=n_wires)
init_params = np.random.uniform(low=0, high=2*np.pi, size=param_shape,
requires_grad=True)
theta_weights = init_params
theta_bias = 0.0
theta_init = (theta_weights, theta_bias) # initial weights
# train the variational classifier
theta = theta_init
pennylane_opt = GradientDescentOptimizer() # build the optimizer
object

# split training data into batches
X_batches = np.array_split(np.arange(len(X_train)), batches)
cost_q = []
for it, batch_index in enumerate(chain(*(n_epochs * [X_batches]))):
    # Update the weights by one optimizer step
    batch_cost = \
        lambda theta: cost(theta, X_train[batch_index],
e_train[batch_index])
    theta = pennylane_opt.step(batch_cost, theta)
    cost_q.append(cost(theta, X_train[batch_index],
e_train[batch_index]))
    # use X_validation and y_validation to decide whether to stop
# end of learning loop

# convert expectations to classes
```

```
expectations = np.array([variational_classifier(theta, x=x) for x in
X_test])
prob_class_one = (expectations + 1.0) / 2.0
y_pred = (prob_class_one >= 0.5)

print(metrics.accuracy_score(y_test, y_pred))

cf_matrix = metrics.confusion_matrix(y_test, y_pred)
index = ["Class 1", "Class 2"]
columns = ["Class 1", "Class 2"]
import pandas as pd
cm_df = pd.DataFrame(cf_matrix ,columns,index)
plt.figure(figsize=(10,6))
plt.title("Confusion matrix for Quantum Binary Classification")
seaborn.heatmap(cm_df, annot=True)
```

```
/usr/local/lib/python3.7/dist-packages/pennylane/_grad.py:108:
UserWarning: Attempted to differentiate a function with no trainable
parameters. If this is unintended, please add trainable parameters via
the 'requires_grad' attribute or 'argnum' keyword.
  "Attempted to differentiate a function with no trainable parameters.
"
```

```
1.0
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd639419dd0>
```



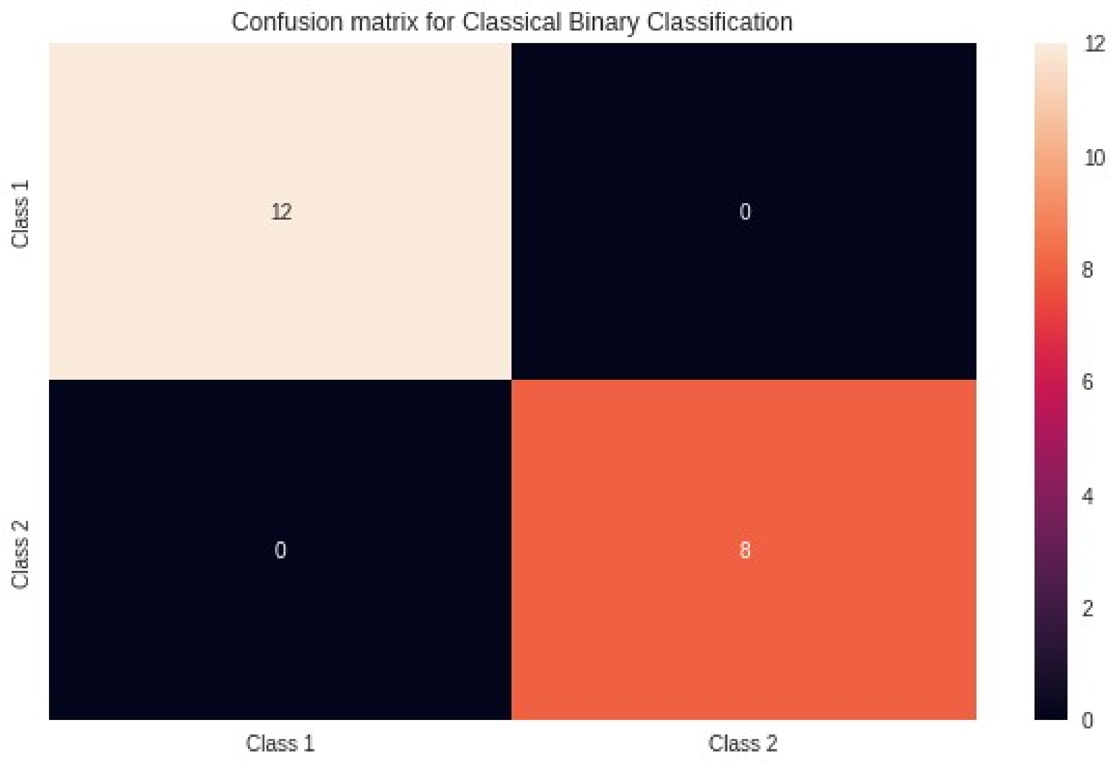
Classical binary classification

```
class NeuralNetwork():

  def __init__(self):
    np.random.seed(1)
    self.synaptic_weights = 2*np.random.random((4,1)) - 1

  def sigmoid(self, x):# The Sigmoid function, an S shaped curve,
normalizes the weighted sum of the inputs between 0 and 1.
    return 1 /(1+np.exp(-x))

  def sigmoid_derivative(self,x):  # The derivative of the Sigmoid
function.# The gradient of the Sigmoid curve
    return x*(1-x)

    # The training phase adjusts the weights each time to reduce the
error
  def train(self, training_inputs, training_outputs,
training_iterations):
    cost = []
    for iteration in range(training_iterations):
      output= self.think(training_inputs)
      # Calculate the error
      error = training_outputs - output
      cost.append(np.mean(error**2))
      # Adjustments refers to the backpropagation process
      adjustments = np.dot(training_inputs.T,
error*self.sigmoid_derivative(output))
      # Adjust the weights.
      self.synaptic_weights += adjustments
    return cost

  def think(self, inputs):
    # Pass inputs through our neural network (our single neuron).
    inputs = inputs.astype(float)
    output = self.sigmoid(np.dot(inputs, self.synaptic_weights))
    return output


neural_network = NeuralNetwork()

# The training set.each consisting of 4 input values and 1 output
value.
training_inputs = X_train
training_outputs = np.reshape(y_train, (len(y_train), 1))
cost = neural_network.train(training_inputs, training_outputs, 5)
expectations = neural_network.think(X_test)
```

```
#If the score is higher than 0.5 then it's a 1 otherwise a 0
def getResult(score):
    if score < 0.5:
        return 0
    elif score >= 0.5:
        return 1

#Apply function on predicted dataframe
y_pred = []
for i in range(len(expectations)):
  y_pred.append( getResult(expectations[i]) )

#Evaluate model performance
print(metrics.accuracy_score(y_test, y_pred))

cf_matrix = metrics.confusion_matrix(y_test, y_pred)
index = ["Class 1", "Class 2"]
columns = ["Class 1", "Class 2"]
import pandas as pd
cm_df = pd.DataFrame(cf_matrix ,columns,index)
plt.figure(figsize=(10,6))
plt.title("Confusion matrix for Classical Binary Classification")
seaborn.heatmap(cm_df, annot=True)
```

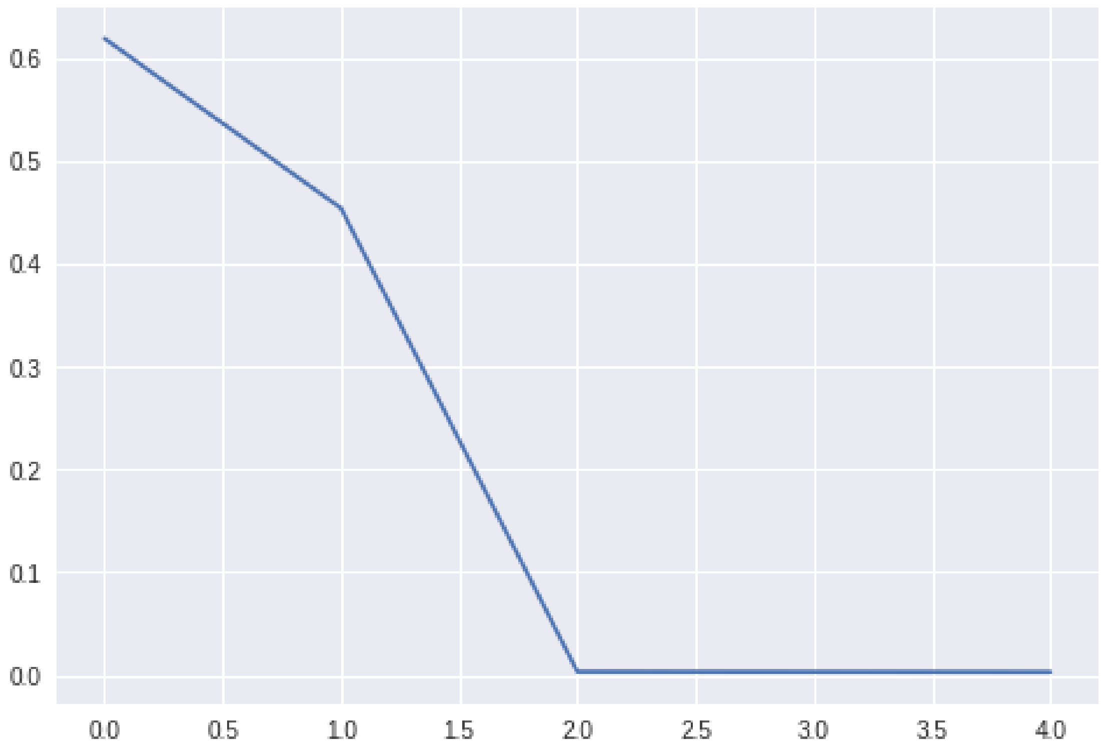
```
1.0
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd63c869e90>
```



```
plt.plot(cost)
```

```
[<matplotlib.lines.Line2D at 0x7fd6391b4750>]
```


0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0