Universiteit Leiden

Master Computer Science

Comparison between tensor networks and quantum machine learning on binary classification task.

Name:	Georgios Laskaris
Student ID:	S3142248
Date:	07/07/2023
Specialisation:	Data Science
1st supervisor:	Dr. F. Neukart
2nd supervisor:	Prof. Dr. T.H.W. Bäck

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The primary objective of this thesis is to conduct a comparative analysis between two Machine Learning approaches: Tensor Networks (TN) and Variational Quantum Classifiers (VQC). While both approaches share similarities in their representation, they diverge in the way they manipulate their trainable parameters. Thus, the aim is to evaluate and compare the expressibility and trainability of these approaches. By conducting this comparison, we can gain insights into potential areas where quantum advantage may be found. Currently, there is a limited understanding of why quantum computers can provide advantages in machine learning tasks. Therefore, this research aims to provide valuable intuition in this regard.

Our findings indicate that VQC exhibits advantages in terms of speed and accuracy when dealing with data, characterized by a small number of features. However, for high-dimensional data, TN surpasses VQC in overall classification accuracy. We believe that this disparity is primarily attributed to challenges encountered during the training of quantum circuits.

Acknowledgements

I would like to express my deepest gratitude to my thesis advisor, Dr. Florian Neukart, for his guidance and support throughout this research. His expertise and insightful feedback have been crucial in shaping the content and quality of this thesis.
I would also like to extend my appreciation to my second supervisor Prof. Dr. Thomas H.W. Baeck and the faculty members of the Computer Science Department (LIACS) at Leiden University. Since they provided me with a stimulating academic environment and access to necessary resources.
Special thanks go to Terra Quantum AG for their collaboration and support during this research. I am grateful to Michael Perelshtein and Artem Melnikov from Terra Quantum AG, since our fruitful discussions and interactions, had a crucial contribution to the development of my understanding of the subject matter.
Furthermore, I would like to express my heartfelt gratitude to my family, brother, and my girlfriend Chara, for their unconditional love, unwavering support, and belief in my abilities.

Contents

1 Introduction

The needs of modern societies tend to push the boundaries of technology more and more as people and technology itself evolve. In the past few decades, the field of Machine Learning has ascended to the surface of Computer science research interests and it expanded rapidly. The increasing amount of data, in combination with the more complex problems that we are asked to solve, leads to the demand for machine learning algorithms that are capable to handle large datasets and solve complex problems. The applications of machine learning are vast, from cybersecurity [1] and natural language processing (NLP) [2] to product recommendations [3], agriculture [4], and healthcare applications [5] among others [6].

Although societies evolve so rapidly, it is uncertain when classical computing, and by extension machine learning, will reach its limits in terms of resources and efficiency. Thus, there is an imperative need to develop the fields that seem the most promising to substitute classical machine learning. Such a field is quantum computing and quantum machine learning (QML). A Quantum computer is a universal computing device, that stores information in quantum bits (qubits) and utilizes different quantum mechanical properties, such as superposition, entanglement, and interference, in order to execute calculations [7–10].

QML is a broad field since it can be described as the combination of machine learning techniques with quantum computing at a given level. In other words, the quantum factor may appear in machine learning methods, under different forms. One form is generated data from quantum processes, or another can be to process classical data with a quantum computer [11]. In general, QML promises to surpass the capabilities of classical machine learning and classical computing. The key factor, that makes us believe in this statement, is the use of quantum mechanical principles in order to make the most efficient computations possible using qubits. The use and advancement of quantum machine learning and quantum technology in general, arise from the continuous growth of the data sizes that need to be processed. A good example is the use of QML [12–14], and quantum-inspired machine learning methods, such as tensor networks [15] in high-energy physics. In addition, more and more practical uses arise, that require the use of quantum computing. Such uses are the autonomous systems of self-driving cars and unmanned aerial vehicles [16] which utilize the image classification task [17–19] which can be solved efficiently with QML methods, such as a variational quantum circuit. The need for powerful computations and algorithms is evident in applications of self-driving cars, since it requires decision-making in real-time, along with fast adaptation to the environment. In these aspects, quantum machine learning can be proven the most efficient and accurate solution.

Another area of interest, that quantum machine learning can potentially revolutionize, is the field of drug discovery [20–22]. In that field, it is of the imperative need to identify the interaction of molecules with target proteins in order to discover a therapeutic effect. Of course, the complex nature of molecules does not allow us to calculate exact solutions, and in many cases, standard machine learning is not enough. It is proven that QML techniques, such as generative adversarial network (GAN) and convolutional neural network (CNN) are superior to their classical analog.

In the current thesis, we follow up on the work done in [23]. There, two models were compared in their performance on publicly available data. The two models were the classical stochastic gradient descent algorithm (SGD) and the quantum-inspired tensor networks (TN) algorithm, which utilizes the tensor train decomposition (TT-decomposition) and the Riemannian optimization algorithm, to execute certain tasks. It is shown that TN gives better performance than SGD, especially with non-random initialization of the weights. Here, we demonstrate the

trade-offs between the same TN model, with a type of parameterized quantum circuit (PQC), called variational quantum classifier (VQC). In practice, the two models are similar in the way they function. Their main difference is the way that manipulate the weights. TN represents the weights of the model as a high-dimensional tensor and VQC represents the weights (parameters) of the model, as rotation parameters of qubit gates (matrix representation).
The two models will be compared on their performance in solving a binary classification task using the UCI car dataset 2013 [24].
This work is structured as follows. In Sec. 2 we make a short but rather useful introduction to quantum computing. The elements presented in that chapter will be used later during the experimentation. In Sec. 3, we make another introduction, but this time to the vast world of machine learning. We give the basic concepts of classical machine learning, along with a quick introduction to quantum machine learning and variational quantum circuits which will be the base of the VQC model that we use. Furthermore, in Sec. 4, we finish with the preliminary knowledge by giving the necessary background on tensors and tensor networks. There, core concepts that are extensively used in the TN model are explained. Additionally, in Sec. 5, the implementation of the models is explained, along with their structure. In the second half of the same section, all the experimental results are presented along with an extensive analysis. Finally, in Sec. 6, we make a conclusion indicating the trade-off between the two tested models. Also, we draw some final inferences that came up from the experimental results. In Sec. 6.1, we refer to future work that can reach beyond this thesis and strengthen our results by verifying the experiments.

2 Quantum Computing.

The fundamental difference between a Classical and a Quantum computing is that the former utilizes quantum bits (qubits) instead of the Classical 0 and 1 bits. The qubits are often represented in *Dirac Notation* as $|0\rangle$ and $|1\rangle$, which is the representation of two-dimensional (column) vectors. The core difference between the qubits and classical bits is that qubits are capable to take every value between 0 and 1. The general state of a qubit can be written as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.1}$$

which is called a *Superposition* of states $|0\rangle$ and $|1\rangle$ [8]. The state $|\psi\rangle$ is, in reality, a two-dimensional complex vector space. The physical meaning of the complex values α and β, is that they can be used for the calculation of the probability of measuring the qubit as 0 or 1. That is

$$p_0 = |\alpha|^2, \quad p_1 = |\beta|^2 \tag{2.2}$$

where p_0 and p_1 are the probabilities to measure a qubit as 0 or 1 respectively.
Following the same strategy we can have systems of two or more qubits, that can be represented by a state $|\psi\rangle$ (see Bell state in section 2.2).
We can easily visualize a single qubit with the unit three-dimensional sphere, which in Quantum Computing and Quantum Information theory is called Bloch Sphere.

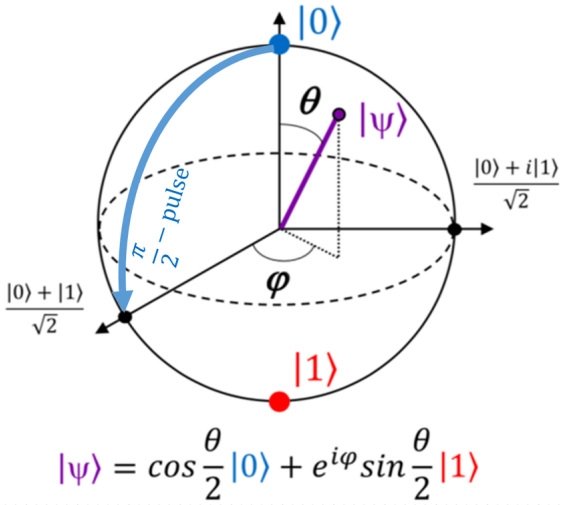


Figure 2.1: Visual representation of a single qubit state on the Bloch Sphere [25]

In order to rotate a qubit on the Bloch sphere, we use the Pauli matrices,

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

which are the generators of the rotations as we show in 2.1. Assuming the general state of a qubit in spherical coordinates (see in Fig.2.1), we can generally multiply the whole state with an additional term $e^{i\gamma}$. This term is called a global phase for the quantum state, and usually we tend to omit it. The rotation by a global phase has no practical impact in the quantum state, in contrast with the local phases (e.g. $\cos\left(\frac{\theta}{2}\right)$and $e^{i\phi}\sin\left(\frac{\theta}{2}\right)$ from Fig.2.1) that are indeed important for the quantum state.

2.1 Quantum gates.

Quantum gates are the quantum analogue of standard classical gates in classical computing. Quantum gates always follow some useful properties. The most important property is that quantum gates are square and unitary matrices [8]. This property secures the preservation of the normalization of the state under the application of a gate. Suppose a unitary matrix A. For A it is $A^\dagger A = I$. This property ensures that our procedures are reversible. Finally, the property of composition applies to quantum gates. If we were to apply a gate A and then a gate B on a quantum state, then the application of those two gates is equivalent with the application of their multiplication $B(A|\psi\rangle) = (B \cdot A)|\psi\rangle$
The general matrices which give the rotations of a qubit in the Bloch sphere are

$$R_x(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}, \tag{2.3}$$

$$R_y(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}, \tag{2.4}$$

$$R_z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix} \tag{2.5}$$

Notice that for $\theta = \pi$ we produce the Pauli matrices with a factor $-i$ at the front, so from (2.3), (2.4) and (2.5) we get

$$R_j(\pi) = -i\sigma_j, \quad \forall j \in \{x, y, z\}$$

Of course, there are not only single qubit gates. An important two-qubit gate is the $CNOT$ gate. This gate controls one qubit and if its value is 0 it does nothing, and if it is 1 it applies the NOT gate (i.e. the X gate). $CNOT$ gate is extensively used in quantum computing since it is the simplest gate which (in combination with the Hadamard gate), can entangle the qubits. Most of the structures used for the experiments utilize a layer of $CNOT$ gates at their core. Despite the entanglement, $CNOT$ gates are really useful when we want to construct more complex gates. An example is the $SWAP$ gate. This gate is applied on two qubits and it swaps their value.

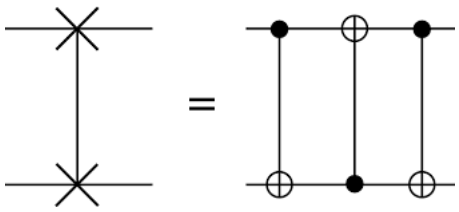


Figure 2.2: SWAP gate construction, using three CNOT gates [26].

It is simple to prove that those three $CNOT$ gates are equivalent to a $SWAP$ gate by calculating their product. We use the property of composition to prove that the product is indeed the application of each gate separately.

$$CNOT_{1\to2}CNOT_{2\to1}CNOT_{1\to2} = \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\cdot\begin{pmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{pmatrix}\cdot\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix} =$$

$$\begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix} = SWAP$$

So in total, will be

$$SWAP|00\rangle = \begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix}\cdot\begin{pmatrix}1\\0\\0\\0\end{pmatrix} = |00\rangle$$

$$SWAP|01\rangle = \begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix}\cdot\begin{pmatrix}0\\1\\0\\0\end{pmatrix} = |10\rangle$$

$$SWAP|10\rangle = \begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix}\cdot\begin{pmatrix}0\\0\\1\\0\end{pmatrix} = |01\rangle$$

$$SWAP|11\rangle = \begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix}\cdot\begin{pmatrix}0\\0\\0\\1\end{pmatrix} = |11\rangle$$

$SWAP$ indeed exchange the qubit states from 0 to 1 and vice versa.
Since each unitary matrix qualifies for a quantum gate, then there should be an infinite number of possible quantum gates. In order to make things more practical, we tend to use the so called universal quantum gate sets [27]. This is a family of sets of gates that are able to construct any gate (unitary matrix) within a finite number of gates from that set. There are too many of those universal sets, but the most popular are the rotation gates from (2.3), (2.4) and (2.5), plus the $CNOT$ gate. Another popular universal set of gates is the Clifford gates [28, 29], which are composed by the $CNOT$, H and S gates, plus the T gate which is originally not in the Clifford gates set. The S and T gates both represent rotations by $\frac{\pi}{2}$ and $\frac{\pi}{4}$ respectively. Finally, the Toffoli gate with the Hadamard gate compose a universal gate set [30]. It would be important to note here that Toffoli gate alone, is a universal gate for the classical reversible circuits [31]. The Toffoli is the equivalent of $CNOT$ for three qubits. It controls the first two and the third one.

2.2 Quantum circuits.

By definition, a quantum circuit is a collection of quantum gates that are connected with quantum wires [32]. Those structures, take a set of qubits as inputs (*Input register*) and after

the application of quantum gates, they return a set of output qubits (*Output register*). A trivial example of a quantum circuit is the bell state circuit.

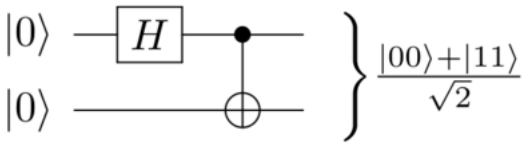


Figure 2.3: Bell state with input $|00\rangle$.

The Bell state depicted in Fig.2.3 has an all zero state as a two-qubit input register, a Hadamard gate H is applied on the first qubit, and then a $CNOT$ gate, with control on the first qubit and target on the second, is applied to the circuit. The output register, turns out to be a superposition between $|00\rangle$ and $|11\rangle$ states. Both those states can be derived with probability $1/2$ upon measurement.
We can prove that by doing the algebraic calculations. The first gate that acts on the input register is H, and then $CNOT$, so it will be

$$CNOT(H \otimes I)|00\rangle = CNOT(H|0\rangle \otimes I|0\rangle) = CNOT\left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle\right] =$$
$$\frac{1}{\sqrt{2}}CNOT(|00\rangle + |10\rangle)$$

Then the $CNOT$ acts as I on the second qubit if the first is 0 and as X gate if the first is 1.

$$CNOT(H \otimes I)|00\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

which is one of the desired Bell states.
Some of the major differences between classical and quantum circuits are, firstly, that in quantum circuits we cannot implement the Fan-In and Fan-out operations of the classical circuits. The Fan-In operation is when multiple bits are merged together into a single wire which is their bitwise OR. The problem with this operator in quantum circuits is that it is not reversible (has a non unitary representation), so we cannot allow it in quantum circuits. The Fan-Out operation is the inverse procedure, when multiple copies of a bit are produced. This procedure is actually forbidden by the the No Cloning Theorem [8].

Theorem 1 (No Cloning Theorem)**.** *It is impossible to create an independent and identical copy of an arbitrary unknown quantum state.*

Proof. Suppose two registers A and B. In register A there is a quantum state $|\psi\rangle$ which we want to copy to register B. The register B, can be in state $|0\rangle$ WLOG. Thus the total state of the quantum device, would be $|\psi\rangle \otimes |0\rangle$
We also suppose a unitary matrix C, which can copy the qubit states. Then, for two quantum states $|\psi\rangle$, $|\phi\rangle$ on A register, we would have

$$C(|\psi\rangle \otimes |0\rangle) = |\psi\rangle \otimes |\psi\rangle \tag{2.6}$$
$$C(|\phi\rangle \otimes |0\rangle) = |\phi\rangle \otimes |\phi\rangle \tag{2.7}$$

Beginning from the inner product of the two states $\langle|\phi\rangle|\psi\rangle$ and using (2.6), (2.7) and the identity

$$(A \otimes B) \cdot (C \otimes D) = A \cdot C \otimes B \cdot D$$

we get

$$\langle\phi|\psi\rangle = \langle\phi|\psi\rangle \otimes \langle 0|0\rangle = (\langle\phi| \otimes \langle 0|) \cdot (|\psi\rangle \otimes |0\rangle) = (\langle\phi| \otimes \langle\phi|)C^\dagger C(|\psi\rangle \otimes |\psi\rangle) = (\langle\phi| \otimes \langle\phi|) \cdot (|\psi\rangle \otimes |\psi\rangle) = \langle\phi|\psi\rangle \otimes \langle\phi|\psi\rangle = (\langle\phi|\psi\rangle)^2$$

But $\langle\phi|\psi\rangle = (\langle\phi|\psi\rangle)^2$, only if $\langle\phi|\psi\rangle = 0$ or $\langle\phi|\psi\rangle = 1$ (i.e. $|\psi\rangle$, $|\psi\rangle$ are orthogonal or $|\phi\rangle = |\psi\rangle$ respectively). This means, that such a unitary can clone only states that are orthogonal to one another, and thus there is no unitary that can clone any arbitrary quantum state in general. □

Despite Fan-In and Fan-Out, another major difference between Classical and Quantum circuits is that quantum circuits have to be acyclic. This means, there are no closed loops, and information transfer one-way from one part of the circuit to another.

2.2.1 Quantum measurement.

According to one of the quantum mechanical postulates, the measurement of a qubit is represented with a set of projectors, that describe an observable M, by $M = \sum_j jP_j$, where j is all possible outcomes after the measurement [8]. So if we measure a given state ϕ, the outcome will be j with probability $p_j = \langle\phi|P_j|\phi\rangle$. Where P_j is the projector for the outcome state j, with $\sum_j P_j = I$. Additionally, if the outcome j is obtained, the state after the measurement will be

$$\frac{P_j|\phi\rangle}{\sqrt{\langle\phi|P_j|\phi\rangle}} = \frac{P_j|\phi\rangle}{\sqrt{p_j}} \tag{2.8}$$

Finally, it is easy to calculate the expectation value of such observable M. That will be

$$E(M) = \sum_j jp_j = \sum_j j\langle\phi|P_j|\phi\rangle = \langle\phi|\sum_j jP_j|\phi\rangle = \langle\phi|M|\phi\rangle$$

3 Machine Learning.

Machine learning (ML) is a vast field of research for computer science. Despite its innumerable applications and its complexity, it is one of the most rapidly developed fields today. There are innumerable ML techniques, that can solve a wide variety of both scientific and practical problems. We tend to categorize those learning techniques as supervised learning (which is a paradigm where data are connected as pairs of input-output) [33] , unsupervised learning (an ML paradigm where patterns are identified in unstructured and unlabeled data) [34], reinforcement learning (a paradigm that uses rewards to make an agent solve various complicated problems) [35].

3.1 Supervised Binary Classification.

Binary classification problems are mainly characterized as a fundamental subcategory of supervised learning problems (although binary classification can also be applied for unsupervised learning problems [36]). It involves categorization of data into two different classes or labels. The purpose of the model is, to assign to each sample its corresponding class. Some example application of binary classification tasks are spam detection [37], applications on medical diagnosis [38], and others.
The two classes used in binary classification, are commonly represented by binary numbers (0 for one class and 1 for the other), although, it is not rare to use $+1$ and -1 to distinguish them. Since we are focusing more on supervised learning, each sample from the data is accompanied by its corresponding label. Thus, after our model makes its predictions, we are able to measure its accuracy using the true labels from our dataset.
To evaluate the performance of a binary classifier we can use a variety of metrics. The most common are accuracy, precision and recall of the model, the F1-score, the area under the curve (AUC) and the area under the ROC curve (AUC-ROC). The most simple and intuitive is the accuracy of the model, which counts the amount of predictions made by the model which agree with the true labels from the data. Among the named metrics, AUC and AUC-ROC are some excellent choices of metrics, since they are robust to unbalanced labels [23] due to their ability to distinguish between the classification classes for different classification thresholds.

3.2 Loss function and Optimizers.

The choice of the loss function is really important in ML. In principal, when we are working with model that depend on continuous parameters, we should choose a loss function that is continuous and differentiable with respect to those parameters [11].
One great candidate for a loss function in binary classification is the Mean Squared Error (MSE). Suppose the prediction of the model with some tunable parameters θ, on a sample x of data, as $\tilde{y_\theta} = \tilde{y}(x)_\theta$, then the true labels of the data as y. The MSE loss function would be

$$L(\tilde{y_\theta}, y) = \frac{(\tilde{y_\theta} - y)^2}{N} \tag{3.1}$$

where of course the loss function is calculated as the mean over all samples N of the dataset. Here, $\tilde{y_\theta} \in [0, 1]$ and $y \in \{0, 1\}$.
An alternative candidate for binary classification is binary cross-entropy loss function. This is given by

$$B(\tilde{y}_\theta, y) = -y \log(\tilde{y}_\theta) - (1-y)\log(1-\tilde{y}_\theta) \tag{3.2}$$

We will show that this loss functions converges to 0, when $\tilde{y}_\theta = y$ and diverges to ∞, when $\tilde{y}_\theta$ is the opposite label of y.
Suppose, without loss of generality, that $y = 1$, then if $\tilde{y}_\theta = y = 1$ it would be

$$B(\tilde{y}_\theta, y) = -\log(1) - 0 = 0$$

if $\tilde{y}_\theta = 0$

$$B(\tilde{y}_\theta, y) = -\log(0) - 0 \to -\infty$$

For the case that $y = 0$, then if $\tilde{y}_\theta = y = 0$

$$B(\tilde{y}_\theta, y) = 0 - \log(1) = 0$$

and if $\tilde{y}_\theta = 1$

$$B(\tilde{y}_\theta, y) = 0 - \log(0) \to -\infty$$

This proof can justify why binary cross-entropy is such a great candidate for binary classification problems, the reason is that it is differentiable and continuous to all values in $\mathbb{R}^+$
Despite the loss function, the choice of the gradient/optimizer for the minimization of the loss function is imperative. The gradient descent algorithm is used to a great extend in optimizers. Given a function $f(x) : \mathbb{R}^n \to R$, its gradient

$$-\nabla f(x) = -\left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(X)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n}\right) \tag{3.3}$$

denotes the direction where the $f(x)$ decreases. By moving iteratively, with a step α, towards that direction we can manage to find the minimum of the function $f(x)$, which is not other but the loss function we introduced above. Of course, by taking large steps, it is possible to get really far from the minimum with just a few iterations, and by taking too small steps, we would need many iterations to reach the minimum. As a result, the hyper-parameter tuning of that step (i.e. learning rate) plays a significant role on the overall performance of the model, and the execution time it requires. A common technique used in ML is the choice of a decaying learning rate α [39]. Among the many decaying learning rate methods, the one we chose to follow in this thesis is the exponential decaying learning rate method. To apply the method, we start from an initial relatively large learning rate (which helps reaching close to the minimum much faster, in case it is too far from our initial spot), then in every iteration, the learning rate is multiplied by a decaying rate $d \in (0, 1)$ so in total the learning rate follows this iterative formula

$$\alpha' = \alpha \cdot d^m \tag{3.4}$$

where α' is the learning rate at the end of the iterations, and m is the number of iterations.
In order to avoid all problematic landscape behaviours, as we mention in 3.3, we have to take into account different proposed optimizers that utilize the gradient descent algorithm. Some of the most important are the momentum gradient descent [40, 41], Nesterov's momentum optimizer [41], Adaptive gradient algorithm (Adagrad) [42], adaptive moment estimation (ADAM) [43] and Riemannian optimization [23, 44] for which we will discuss extensively in 4.2.1.

3.3 Learning Behaviour.

During training, both in classic and quantum machine learning (QML), it is possible to get some negative learning behaviours. In other words, the model might end up over-fitting or under-fitting to the data [45]. The problem of over-fitting to the data, has to do with the excess training of the model on the training set. This excess training can lead to a predictor model that has been optimized to predict exactly the labels on the data of the training set, but when a new batch of samples comes for classification, the model fails to predict the correct label. A visual example of this behaviour can be seen in Fig. 3.1. Overfitting can be spotted when analyzing the plot of the accuracy of the model, when it includes both training and test/validation sets. In case the training accuracy is increased significantly over the test accuracy, then this can be a strong indication of overfitting. Adding a regularization term in the loss function, sometimes can cure overfitting. Other heuristics are taking advantage of a larger dataset, use dropout terms and the use of early stopping by monitoring the validation set performance [11, 46].

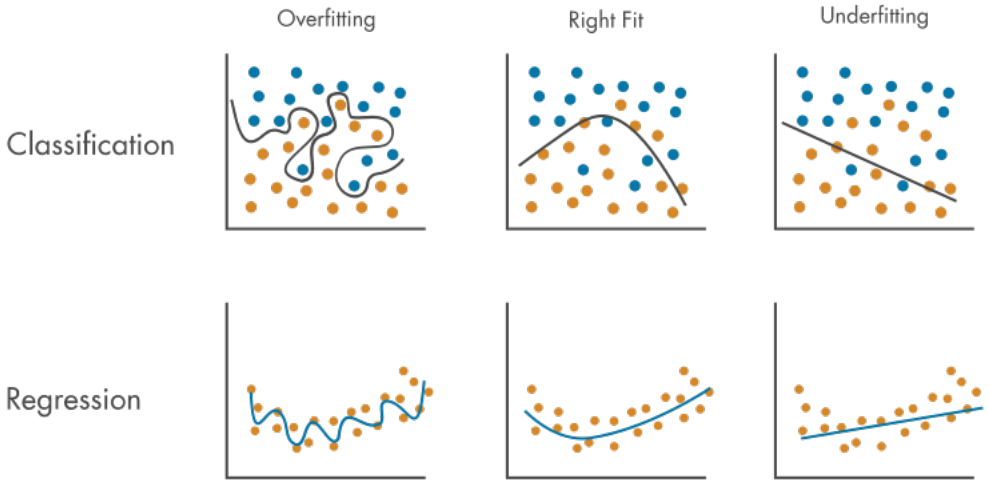


Figure 3.1: Learning behaviours on classification and regression problems. [47]

On the other hand, when the model is poorly trained on the data, then it tends to be quite simplistic and linear, which results to failure of prediction of the true labels of the data.
We can get a hint of underfitting to the data, once again in accuracy plots, when both the training and validation set accuracies are almost identical and really low in value. Of course, the low accuracy may arise from different factors, but underfitting can always be one of them.

3.3.1 Barren Plateaus

Despite of overfitting and underfitting to the data, there are also different kind of problems, especially associated with QML. One of those problems has to do with the loss function landscapes and the gradient of the model. That is the barren plateaus [11, 48, 49].
The barren plateaus, as can be seen from Fig. 3.2, are formed when the global minimum of the loss function becomes exponentially small with the problem size, leading to narrow-gorge [50]. The barren plateaus, may lead to destruction of quantum speed-ups and can result to more complex quantum structures that may resemble classical methods.

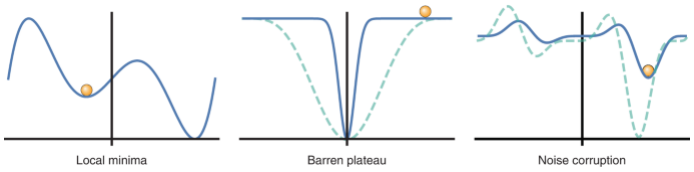


Figure 3.2: Different kinds of training problems regarding the gradient. In the first example there is the case of local minima, where the model needs bigger steps to get out of them and reach even lower values. In the second example we have the barren plateaus as discussed in the current section, and finally we have the case of noise corruption, which is evident only in QML where noise from the environment has large impact on the data and drifts the bias of the model [48].

3.4 Dimensionality reduction.

The dimensionality reduction of the data is generally a powerful tool of machine learning. It is applied directly to the data before they enter the network/circuit and the training is initialized. With dimensionality reduction techniques [51], we have a powerful tool towards curse of dimensionality [52]. The curse of dimensionality is related to the problems that arise when working with high-dimensional data. By high-dimensional data, we mean samples of data which consist of a large number of features. Do not forget that samples can be represented as points in a high-dimensional space, where the dimension is equal to the number of their features. When working with such data, we need exponentially more of them, to sufficiently train our model [53].
Other major problems that arise when we work with high-dimensional data are the computational power and resources required for the model to get trained and solve the problem efficiently. All those problems can be mitigated or even eliminated with some useful dimensionality reduction techniques.
Dimensionality reduction is used to project the data (which, as we discussed, are usually represented as points in an n-dimensional space, where n is the number of their features), to a feature space of a smaller dimension, with the solely purpose of giving prominence to their internal symmetries which will lead to better and most accurate solution of the relative problem (regression, classification, etc.).
Typically, the dimensionality reduction techniques used, are divided into linear and non-linear techniques. Some of the linear techniques are the principal component analysis (PCA) [11, 49, 54, 55], which we will extensively cover in section 3.4.1, the linear discriminant analysis (LDA) [55] and others. LDA's purpose is to maximize the variance of the data that belong on different classes and at the same time, minimize the scatter between samples of the same class.
Some examples of non-linear dimesnionality reduction techniques are the manifold learning methods, such as Laplacian eigenmaps [55, 56], isomap [55, 57], and others.

3.4.1 Principal Component Analysis.

Suppose we have a dataset of samples with n features. Then those samples/data can be represented as points in $\mathbb{R}^n$. The first principal direction of the dataset lies at the direction of the line, which by measurement of the mean squared error, best fits the data. Then the second principal direction is once again at the direction of the line that best fits the data, but this time we need it to be orthogonal to the first principle direction. The same procedure is

followed for the third principal direction, but this time it needs to be orthogonal to both first and second principle directions. We go on with the same algorithm for the remaining principle directions [11].
In total, the principal directions construct an orthonormal basis $\{v_1, v_2, \ldots, v_n\} \in \mathbb{R}^n$, where v_i is the i-th principle direction with elements $v_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$. So the original sample of n features, will be projected to a feature subspace of dimension m, with

$$\tilde{x}_i = \sum_{j=1}^{n} v_{ij} x_j$$

where $\tilde{x}_i$ is the i-th principal component and x_j is the j-th feature of the sample.

3.5 Quantum Machine Learning

Quantum Machine Learning (QML), can be characterized as the combination of Classical Machine Learning with quantum mechanics at some extent. The quantum mechanical element, can be inserted through different means. Some examples are the implementation of quantum data as qubits, the use of a parameterized quantum circuit (PQC) [58] for the training of the model, or even quantum algorithms, such as, Quantum Approximate Optimization Algorithm (QAOA) [59], Variational Quantum Eigensolver (VQE) [60] or Quantum Support Vector Machines (QSVM) [61], for the further improvement of the model training and optimization [48].

3.6 Variational quantum Circuits

A specific application of PQC is the variational quantum classifier (VQC) [49, 58, 62]. In general, a PQC is a type of quantum circuit that utilizes parameterized quantum gates in order to perform specific tasks. Then VQC uses the parameterized architecture of PQC with combination of classical optimization algorithms which fine-tune the parameters of the quantum gates, in order to minimize a loss function. The VQC is composed by two different parts, the encoder part and the variational part [58]. In the encoder part, a quantum circuit is used to encode the features of the samples into qubits. It is obvious that there are many different encoding techniques, such as Angle Encoding, Amplitude Encoding, wave-function encoding and others [11, 58, 63]. The choice of encoding is closely related to different kernel methods. These methods are used to project the data into a higher-dimensional feature space, where the same problem is usually easier to solve. So, the proper choice of a feature space is paramount for the solution of the problem.
For example, non-linear feature maps are able to project the data into a feature space where their relative distance is much different. In that way, it is possible that the samples of the data can be much easier distinguished from one another, and thus a binary classification among them, can be achieved with better accuracy. The inner product of two points (samples) in the feature space, characterizes the kernel similarity function [64].
The second part of the VQC is the variational or the parameterized part, which is a quantum circuit of a given structure, that consists of entangling layers and rotation gates on the qubits, with tunable parameters [65]. An important hyper-parameter of this model is the variational layers of the variational circuit. In order to give some depth to the variational circuit, which is crucial for its performance, we have to repeat the structure of $CNOT$ gates and rotations gates multiple times. Finally, measurements on one or more qubits are made, in order to end up with

the model's prediction of the class for the data. Further discussion on VQC implementation is demonstrated in 5.1.1.

4 Tensor Networks

In recent years, tensor networks have been extensively used as a potent tool for representing and manipulating high-dimensional data. Tensors, which can be characterized as multidimensional arrays, are included in a variety of scientific fields, such as physics, computer science (machine learning), and quantum computing (quantum information theory [66], QML, etc.). The current section provides an overview of tensor networks, giving emphasis on matrix product states (MPS). MPS is a specific structure of a tensor network (see 4.1.1). Tensor Train Decomposition (TT-decomposition) is mainly used for compressing and approximating high-dimensional tensors. We discuss the basic concepts and properties relevant to MPS and TT-decomposition, highlighting their performance when working with high-dimensional data structures.
Additionally, in the current section, we make a short introduction to Riemannian manifolds and how we can utilize their properties to introduce the Riemannian optimization algorithm. In particular, the Riemannian optimization algorithm introduced in 4.2.1 leverages the geometry of the underlying Riemannian manifold to perform optimization steps with respect to the geometry of the manifold, having as a result improved convergence rates and robustness.

4.1 Tensors

Tensors are mathematical objects that can generalize the idea of vectors and matrices. Tensors are better represented graphically as nodes with several edges sticking out of them. The number of their edges represents their rank (Fig. 4.1).

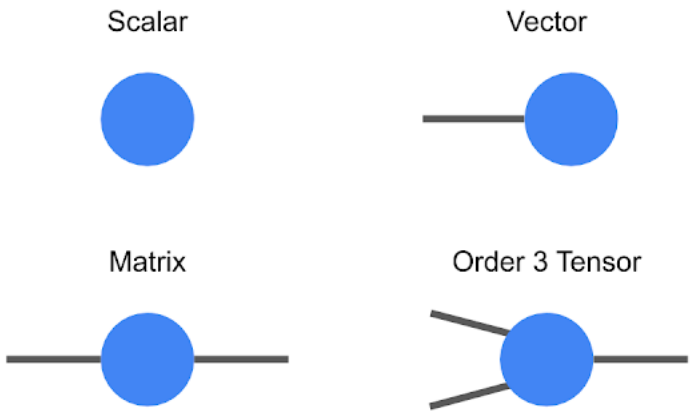


Figure 4.1: A rank-0 tensor (top-left) is a scalar, a rank-1 tensor (top-right) is a vector, a rank-2 tensor (bottom-left) is a matrix and (bottom-right) a rank-3 tensor can be considered as many matrices being one in front of the other, forming a geometrical cube of numbers.

In general, an r-rank tensor is an element of $\mathcal{C}^{d_1 \times d_2 \times \dots d_r}$, where $d_1, d_2 \dots d_r$, are the respective dimensions of each rank of the tensor.
A useful tensor operation is the contraction between two or more tensors. Contraction is the pairing that happens between a vector space and its dual space. So, for example, when we multiply two vectors $\langle \mathbf{x} | \mathbf{y} \rangle = \sum_i x_i \cdot y^i$, we get a scalar value. In practice, the vector $\mathbf{y}$ lies in the dual vector space of where $\mathbf{x}$ is. Moreover, the common index of the vectors in graph

notation is eliminated by the contraction operation, and thus, a node without edges (scalar), is left after the contraction. Another example is when we multiply a matrix A (rank-2 tensor), with a vector $\mathbf{x}$ (rank-1 tensor). A similar procedure follows, where graphically, all the common indices (edges) of the two tensors are eliminated, and the edges left, define the rank of the resulting tensor. In this example, the contraction $A\mathbf{x} = \mathbf{y}$ results in a rank-1 tensor $\mathbf{y}$ [67].
We should note here, that while we work with tensor networks, there are more than one ways to do the contraction between them. Usually, there are not all contractions the same. The order in which the contraction happens can play a huge role in the time and memory complexity of the procedure. The order in which the tensors are contracted is called bubbling [67, 68] and the optimization of it, is capable to reduce the complexity of the contraction problem for a tensor network significantly.
As happens with vectors, there are multiple norms for tensors too. The most common tensor norm used in quantum computing is the Frobenious norm, which is a generalization of norm-2 (Euclidean norm) of vectors. The Frobenious norm consists a matrix norm and a tensor norm as well [69]. So for a d-dimensional tensor $\mathcal{A}$ with entries $\alpha_{i_1 i_2 \dots i_d} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d}$ where $i_k \in \{1, 2, \dots, n_k\}$ and n_k is the size of k-th dimension of $\mathcal{A}$. Then, the Frobenious norm of $\mathcal{A}$ will be

$$||\mathcal{A}||_F = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \dots \sum_{i_d=1}^{n_d} \alpha_{i_1 i_2 \dots i_d}^2 \tag{4.1}$$

This norm is generally used as a measure of the similarity of two tensors. So for a second d-dimensional tensor $\mathcal{B}$, the norm $||\mathcal{A} - \mathcal{B}||_F$ will give a metric of the similarity between those two tensors. Of course, if $\mathcal{A} = \mathcal{B}$ then $||\mathcal{A} - \mathcal{B}||_F = 0$.

4.1.1 Matrix Product States

We can describe a tensor network as the diagrammatic representation of a collection of tensors. Tensor networks are usually used in many-body quantum systems, and one of the most well-studied families of tensor networks are Matrix product states or MPS [67, 70, 71].
An MPS or a tensor-train (TT) is a 1-dimensional architecture of a tensor network, where tensors are connected through one index called bond index (the dimension of which, is called a virtual dimension or **tensor train-rank** [67]) and have another index called visible index, sticking out of the tensor (the dimension of which, is called the physical dimension [67]) (Fig. 4.2). In general, the tensor train rank (TT-rank) r, can vary from bond index to bond index and it is regarded as a hyper-parameter of the model.
It appears to be two different variations of MPS. The first variation is the one described above and it is called an open boundary conditions MPS (OBC MPS) [72], and the second is when the last and the first tensors in the MPS are connected with a bond index. That variant is called periodic boundary conditions MPS (PBC MPS) [72].
An MPS is generally used to approximate a large tensor. Suppose a tensor with N indices of dimension D each, so such a tensor would have D^N parameters. With MPS model, we can approximate such a tensor with only NDm^2 parameters [73]. This number not only reduces the number of parameters by an exponential growth depending on N, to a linear dependence on N, but also can be further reduced, using different techniques and exploiting various constraints.

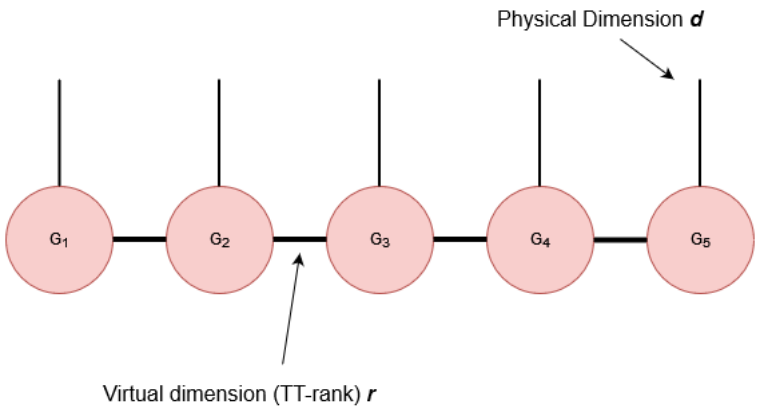


Figure 4.2: The structure and the dimensions of an MPS with open boundary conditions.

4.1.2 Singular Value Decomposition

The Singular Value Decomposition (SVD) [74–76], is one of the most important factorization techniques. It is applied to matrices, but it can generalize also to tensors as the so-called, high order singular value decomposition (HOSVD) [77, 78]. In SVD, we generally decompose a complex matrix $A \in \mathbb{C}^{n\times m}$. The SVD of such a matrix is $A = U\Sigma V^{\dagger}$, is guaranteed to exist, and it is unique [79]. $U \in \mathbb{C}^{n\times n}$ and $V \in \mathbb{C}^{m\times m}$ are square unitary matrices, with orthonormal columns. $\Sigma \in \mathbb{R}^{n\times m}$ is a diagonal matrix, with positive real values in its diagonal. Of course the symbol † is referring to the complex conjugate and transpose matrix of V.
As stated above, the idea of SVD generalizes to tensors. We are able to rewrite a tensor $\mathcal{A}$ of rank-$(n+m)$ into a matrix A, with elements $A_{i_n,j}$, where

$$j = 1 + \sum_{\substack{k=1 \\ k\neq n}}^{N} (i_k - 1) \prod_{\substack{m=1 \\ m\neq n}}^{k-1} I_m \tag{4.2}$$

and I_m the dimension of the i-th rank [77].

4.1.3 Tensor Train Decomposition

The tensor train decomposition or TT-decomposition is a way of manipulating multi-dimensional tensors as simple two-dimensional matrices or even approximating them with a small error ϵ. In the heart of TT-decomposition, lies the SVD, which is applied $d-1$ times on the desired d-dimensional tensor $\mathcal{A}$. This procedure returns d matrices, which can efficiently represent $\mathcal{A}$ [23, 78].
The general idea of TT-decomposition, is to approximate all entries of the tensor $\mathcal{A}$ by the product of d matrices $G_k(i_k)$ (called TT-cores [23]) of dimension $r_k \times r_{k-1}$ (with $r_0 = r_d = 1$ since we want the product to return a scalar value), within an error ϵ. The indices i_k, with $k \in \{1, 2 \ldots, d\}$ and $i_k \in \{1, 2, \ldots, n_k\}$, represent the dimension which enumerates over the k-th index of tensor $\mathcal{A}$, where n_k is the dimension of the k-th index of $\mathcal{A}$. In other words, we have

$$A_{i_1 i_2 \ldots i_d} = \prod_{k=1}^{d} G_k(i_k) \tag{4.3}$$

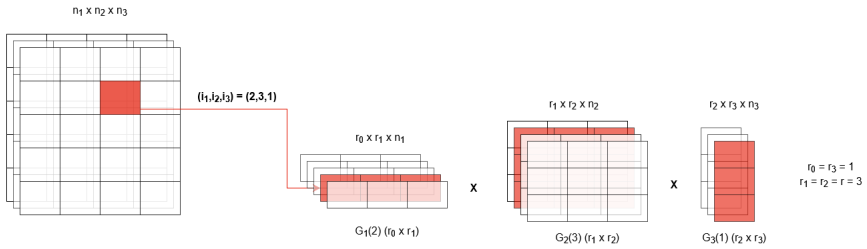


Figure 4.3: TT-decomposition example of getting an arbitrary element of a tensor, from the matrix product of the TT-cores $G_k(i_k)$ [23]. In the example, $r_1 = r_2 = r = 3$ has been used as the tensor rank and the element $\mathcal{A}_{2,3,1}$ is calculated.

Notice from Fig. 4.3, in order to calculate an entry of the tensor $\mathcal{A}$ (for example entry $\mathcal{A}_{2,3,1}$), we need to multiply the second matrix from the first TT-core (this corresponds to the first coordinate of the entry), with the third matrix of the second TT-core (which corresponds to the second coordinate of the entry) and finally, multiply with the first matrix of the final TT-core (which corresponds to the third coordinate of the entry).
As stated above, by applying the SVD multiple times, depending on the dimensionality of $\mathcal{A}$, we derive the desired TT-core tensors. For further analysis around the algorithm of TT-decomposition and some follow-up proofs of the methods used, please refer to [78].

4.2 Riemannian Manifolds

Riemannian geometry consists of a branch of differential geometry that includes and describes the Riemannian manifolds. Manifolds are topological spaces that locally resemble Euclidean spaces. A Riemannian Manifold $\mathcal{M}$ is a smooth Hausdorff and second countable manifold (by smooth we mean it is C^∞, infinitely times differentiable), equipped with a positive-definite smoothly varying inner product g metric, which can be used to determine an inner product on each point p of the tangent space $T_p\mathcal{M}$ of $\mathcal{M}$ [23, 44, 80].
In a given d-dimensional tensor $\mathcal{A}$, we can apply the TT-decomposition with a fixed rank $r_i = r,\ i \in \{1, 2, \ldots, d-1\}$ and of course $r_0 = r_d = 1$. Then the collection of all those possible tensors consist a Riemannian manifold.

$$\mathcal{M}_r = \{\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d} : \text{TT-rank}(\mathcal{A}) = r\} \tag{4.4}$$

by treating a tensor $\mathcal{A}$ as a point on the Riemannian manifold we can operate Riemannian optimization to approximate a different tensor $\mathcal{B}$ on the manifold, with $\mathcal{A}$.
The Riemannian optimization technique can be used in machine learning since we can consider tensors $\mathcal{X}$ and $\mathcal{W}$, where $\mathcal{X}$ is of rank 1 and consists of all the features that describe the data. $\mathcal{W}$ can be considered as a d-dimensional tensor (we parallelize it with $\mathcal{A}$ from the current and previous sections) which contains all tunable weights of the model. This tensor $\mathcal{W}$, is once again a d-dimensional tensor but its dimensions can be $n_1 = n_2 = \ldots = n_d = 2$, thus, its entries are in $\mathbb{R}^{2\times 2\times \ldots \times 2}$. This stems from the fat that the interactions of the features can be multiplied with the weights by the following equation

$$\hat{y}(\mathbf{x}) = \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} \ldots \sum_{i_d=0}^{1} \mathcal{W}_{i_1 i_2 \ldots i_d} \prod_{j=1}^{d} x_j^{i_j} \tag{4.5}$$

Where $\hat{y}(x)$ represents the prediction of the model [23]. In other words, the prediction is the product of a weight with the relative feature or product of features (as indicated by the indices

of the weight). When an entry of the weight tensor is multiplied with some features, then the position of all indices for which the indices become 1, symbolize the features that are multiplied by the weight. Respectively, the position of the indices of the weight for which the indices are 0 represent the features that are not multiplied with the weight. A simple example, would be on two dimensions, where we would have

$$\hat{y}(\mathbf{x}) = \mathcal{W}_{00} + \mathcal{W}_{10}x_1 + \mathcal{W}_{01}x_2 + \mathcal{W}_{11}x_1x_2 \tag{4.6}$$

In (4.6), notice in the first term, that the weight $\mathcal{W}_{00}$, since it has both 0 to its indices, it was not multiplied with either first or second feature. On the contrary, in the last term, the weight $\mathcal{W}_{11}$ was multiplied by both first and second features. Additionally, see in the second term that since the first index of the weight $\mathcal{W}_{10}$ is 1 and the second is 0, then the weight is multiplied only with the first feature and not with the second.
The problem we want to solve, is the minimization of loss. The loss function is

$$minimize \quad L(\mathcal{W}) \tag{4.7}$$

with a constant TT-rank r in TT-cores of $\mathcal{W}$. In (4.7) the loss can be expressed as

$$L(\mathcal{W}) = \sum_{s=1}^{N} MSE(\hat{y}(\mathbf{x}^{(s)}), y^{(s)}) + \frac{\lambda}{2}||\mathcal{W}||_F \tag{4.8}$$

where the term $\frac{\lambda}{2}||\mathcal{W}||_F$ is the L_2 regularization term, with λ being the regularization parameter and $||\mathcal{W}||_F$ the Frobenius norm of the weight tensor $\mathcal{W}$. The $MSE(\hat{y}(\mathbf{x}), y) : \mathbb{R}^2 \to \mathbb{R}$ is the mean squared error or the squared loss of the predictions $\hat{y}(\mathbf{x})$ towards true values y, and N being the total number of samples in the data.

4.2.1 Riemannian Optimization

Since all tensors such as $\mathcal{W}$ can be written as a manifold $\mathcal{M}_r$, then we can utilize the Riemannian optimization algorithm [23, 78], to train the entries of the weight tensor $\mathcal{W}$ on predicting as accurate as possible the class for each sample in the data. The steps of the Riemannian optimization algorithm are the following.
First, one has to calculate the gradient of the loss function towards the weight tensor $\frac{\partial L}{\partial \mathcal{W}}$ and project it to the tangent space of $\mathcal{M}_r$ which is $T_{\mathcal{W}}\mathcal{M}_r$ at point $\mathcal{W}$. We define that projection as

$$\mathcal{P} = P_{T_{\mathcal{W}}\mathcal{M}_r}\left(\frac{\partial L}{\partial \mathcal{W}}\right) \tag{4.9}$$

Then we follow the direction of the projection $\mathcal{P}$ with a small learning step α. This step takes us out of the manifold and into the tangent space. That action has as a consequence an increase in the TT-rank. So, in order to get a slightly shifted weight tensor, that belongs to the manifold $\mathcal{M}_r$ we need to reduce (or round) the TT-rank back to r [78]. This reduction can be achieved by retracting the projection from the weight tensor by a small learning step α. In total, it would be $\mathcal{W} - \alpha\mathcal{P}$ to return to the manifold.
By repeating those three steps multiple times, we are able to minimize the loss function, by training the weight tensor, to predict the classes as accurately as possible.
This optimization algorithm has given remarkable performance results in comparison with other popular methods, such as Stochastic Gradient Descent (SGD) [23].

5 Experiments and Results.

In the current chapter, we describe and analyze the experiments that were conducted along with the implementation of the models and the techniques used for the proper execution of the experiments. In the end, all the results are presented and discussed in order to come up to a solid conclusion. All experiments regarding quantum circuits run on a simulator with a Tesla T4 GPU with 16 GDDR6 RAM. We noticed during experimentation that the use of an actual quantum computer would speed up the experiments significantly, since with the T4, the experiments regarding VQC were running for two consecutive weeks in total.

5.1 Implementation

Two classification models (namely VQC and TN) are tested and compared on their performance on a binary classification task. The VQC model used for the experiments is based on the model discussed in section 3.6. The TN model is using the TT-decomposition algorithm and the Riemannian optimization method that were discussed in Sec. 4.1.3 and Sec. 4.2.1 respectively. The first experiment was to record the performance for each individual model, while their main hyper-parameter changed. For VQC that is the number of variational layers, and for TN is the TT-rank, which is the dimension of the TT-cores in TT-decompostion algorithm. That performance is recorded for different numbers of qubits. We are able to test the models with a reduced number of features, with the help of PCA. A second experiment conducted was a comparison experiment between the models. Both models have been evaluated according to their validation accuracy with respect to the number of trainable parameters they use. So for a fixed number of qubits, each time, we calculated the accuracy of the models while the number of layers/TT-rank changes, and as a consequence, the number of trainable parameters change. The results were combined into the same plot, so a straight comparison between the model is more evident.
For both models, the same publicly available dataset has been used for the binary classification. This is the UCI car evaluation dataset of 2013 [24]. It originally consists of 1728 samples with mainly categorical attributes which are 6 in total. Converting the categorical features to binary with one-hot encoding we end up with 21 binary features in total. For the experiments, we used a splitting ratio of 80% between training and validation sets. Additionally, for the data pre-processing we used the dimensionality reduction technique of Principal Component Analysis, as explained in chapter 3.4.1. With PCA, we were able to run experiments with $2, 5, 10$ and 16 principal components/features of the data, as well as all 21 features. The reasoning behind this choice is to record the performance of the models, both when they train on high-dimensional data and low-dimensional data.
Originally, the UCI car dataset refers to a multi-classification problem, since it consists of four different classes ("unacc", "acc", "good", "vgood"), which refers to the acceptability of each car in the dataset. In order to convert it to a binary classification problem, we merged the three classes ("acc", "good", "vgood") into one class (namely "acc") which we later represented with $+1$ and left the other class ("unacc") as -1. The features of the car data have to do with their "buying price", "price of maintenance", "number of doors", "number of persons to carry", "the size of luggage boot" and the "estimated safety of the car".
For both models, their weights were initialized randomly, and the accuracy of the validation set has been used as a comparison metric between them.

5.1.1 Variational Quantum Classifier model

The implementation of The VQC required of course an encoding part and a variational part. For the encoding part, the cosine/sine encoding has been used [81, 82]. For this encoding, starting from an all 0 state, we apply Y-rotation single qubit gates on each qubit. So after the encoding part, each qubit would be in state

$$|0\rangle \rightarrow R_y|0\rangle = \cos\left(\frac{\pi}{2}x_k\right)|0\rangle + \sin\left(\frac{\pi}{2}x_k\right)|1\rangle \tag{5.1}$$

where $k \in \{1, 2, \ldots, N\}$ and N is the total number of features/qubits used for each sample. In that way, we introduce and pass information from the data to the quantum circuit.
After the encoding part, a variational architecture had to be established. For that variational architecture, we chose to follow the Noisy Intermediate Scale Quantum (NISQ) friendly architecture used in [83]. The main parameter of this architecture (except the weights of course) is the number of variational layers used. Each layer consists of $CNOT$ gates with control on all odd-numbered qubits, then Y-rotation single qubit gates applied on each individual qubit, $CNOT$ gates with control on every even-numbered qubit, and one more Y-rotation on every qubit. So in total, for every layer we will need to train $2N$ parameters ($+N$ for the 0-th layer, which consists of only Y-rotation gates on every qubit). In total, this model needs to train $2NL + N = N(2L + 1)$ parameters, where L is the number of layers used.
In Fig. 5.1 an example of the architecture of the VQC model has been illustrated. This five qubits example refers to a VQC model with two variational layers. For the prediction of the model, only the expectation value of only the first measurement is used. If the expectation value is under the predefined threshold we set, the sample is classified to class -1 ("unacc"), and if it is above the threshold to class $+1$ ("acc"). The threshold used throughout the experiments is set to 0.5.

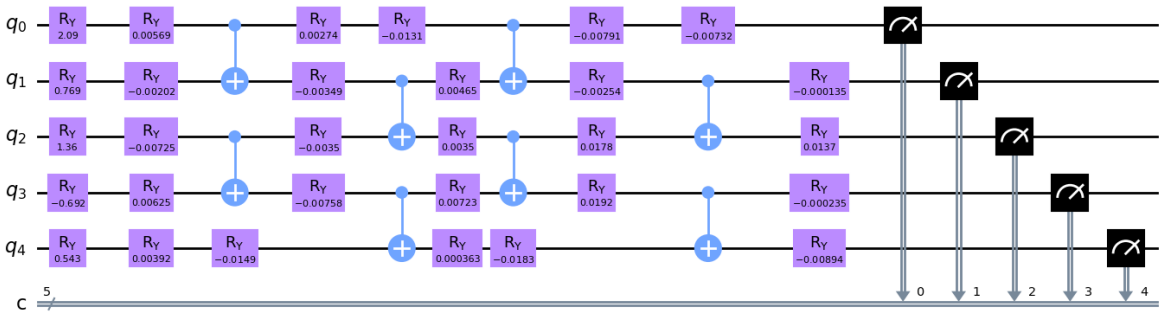


Figure 5.1: Example of the architecture used for the VQC model, with 5 qubits and 2 variational layers.

As a loss function for this model, we used the MSE function. For the implementation of the circuit, we used Pennylane (version = 0.30.0) library [84]. For the optimizer, we tried Standard Gradient Descent (SGD), Nesterov momentum optimizer, and Adaptive Momentum optimizer (Adam). But the one who gave the best performance among them was Adam so that one was chosen for all experiments with the VQC. During the training, a batch size of 32 samples was used, and a decaying learning rate was initialized from 0.1 and decayed with a decay factor of 0.95.

Finally, the latest feature of Pennylane was utilized. This is the Pennylane-lightning[GPU] device. This device let us use the NVidia GPU (CUDA) to run the simulation of the quantum circuit. With the help of this feature, a significant speed-up was achieved, especially, in comparison to a simple simulation on a CPU.

5.1.2 Tensor Network model

For the implementation of the Tensor Network (TN) model the implementation from [23] has been extensively used. In this model, the method of TT-decomposition has been utilized to manipulate the weights tensor easier and with better efficiency (especially when working with high-dimensional data). The encoding of the data follows the equation (5.2)

$$\mathcal{X}_{i_1 i_2 \dots i_d} = \prod_{k=1}^{d} x_k^{i_k} \tag{5.2}$$

where, $\mathcal{X}_{i_1 i_2 \dots i_d}$ represents the tensor which includes the features of the samples, and $x_k^{i_k}$ is the k-th feature of the sample. Once again, the indices $i_j,\ j \in \{1, 2, \dots, d\}$ are $i_j \in [0, 1]$.
This model is trained with the Riemannian optimization algorithm and uses the logistic loss as a loss function.
In TN, the main hyper-parameter is the TT-rank used for TT-decomposition. The total number of parameters that need to be trained in this model is $2Nr^2$, where N is the number of features used, and r the TT-rank. Notice, the number of parameters in TN scales faster in comparison with VQC, since the TN parameters, instead of having a linear dependence on r (as happened with VQC and the number of layers), depend on the square of the TT-rank.
For the implementation of the TN model, the ttpy library, which is a python implementation of the TT-Toolbox library [85], has extensively been used, in coordination with other libraries including mathematical tools, such as numpy [86], scikit-learn [87] and others.
As discussed in Sec. 4.2 the predictor of this model is based on the simple linear product between the features tensor $\mathcal{X}$ and the weights tensor $\mathcal{W}$. So in total, we can rephrase (4.5) as

$$\hat{y}(\mathbf{x}) = \langle \mathcal{X}, \mathcal{W} \rangle \tag{5.3}$$

There is no need for a separate bias term since this is integrated in $\mathcal{W}$ as $\mathcal{W}_{00\dots0}$ (for a two-dimensional example, see Sec. 4.2).

5.2 Results

In order to compare those two models (TN and VQC), we established a common ground between them. In other words, we tested their performance in the same experimental environment. That includes the same dataset, with the same split ratio between the training and validation set. Their performance was recorded according to the number of parameters they use. Despite that, we made individual experiments for every model, where their performance (accuracy of the validation set) was registered while changing the TT-rank for the TN and the number of variational layers for the VQC.
For the experiments with the TN model, the implementation from [23] has been utilized. To that implementation, the PCA dimensionality reduction technique has been added, along with all the necessary functions, to receive the desired plots that derive from the results.

Those plots are the accuracy of the validation set that TN achieves while TT-rank increases for $r \in \{2, 3, \ldots, 20\}$ and the validation accuracy it achieves depending on the number of trainable parameters.

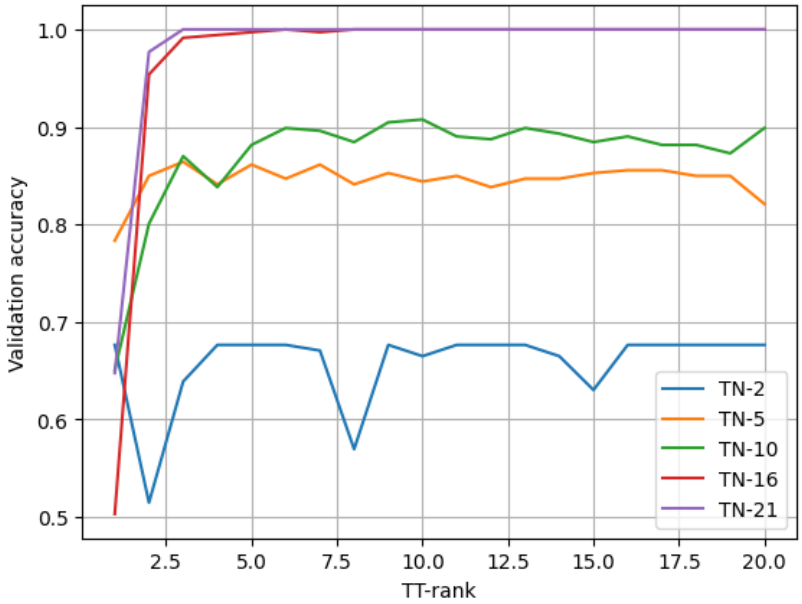
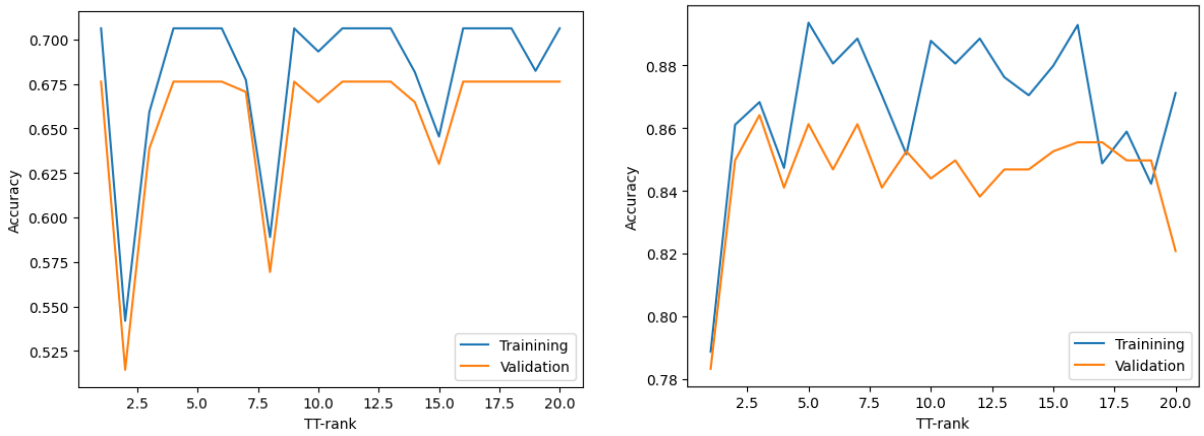


Figure 5.2: Plot of the validation accuracy achieved from the tensor network model, with TT-rank that runs from 2 up to 20. The experiment was conducted for different number of features using the principal component analysis (PCA). The number of components used is 2, 5, 10, 16, and 21 which is the number of features without PCA.

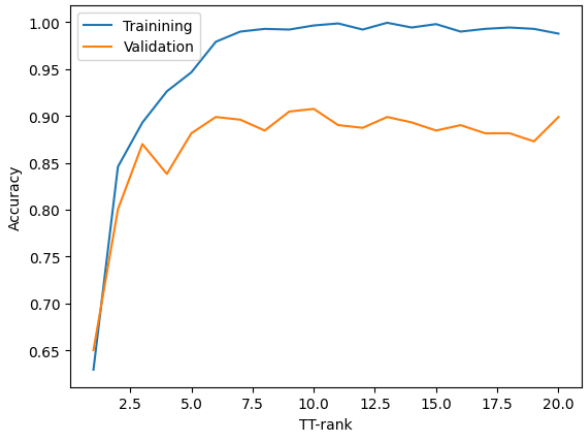
As we notice from Fig. 5.2, an increase in the TT-rank indeed increases the performance of the model. This increase is rapid for TT-ranks up to $r = 4$ or $r = 5$. With further increase in the TT-rank, notice the accuracy on the validation set is not improved significantly, but rather it converged to a final value. In parallel, the validation accuracy shows a significant increase as the number of principal components increases.
For the case of the two principal components, observe a slightly unstable performance for the model. It gets its best accuracy at $\sim 68\%$ with sudden drops for specific ranks. The reason behind this, is probably that there is not much information stored in the two principal components, in order for the model to sufficiently classify the data. This becomes more obvious when we focus on the behavior of the TN when experiments run, with more principal components. For example, with 5 principal components, its accuracy reaches $\sim 85\%$ and oscillates around that value for different TT-ranks. Adding more components (10) the accuracy slightly increases to $\sim 90\%$ and, once again, slightly oscillates around that value. Furthermore, by adding more and more principal components, it seems that enough information is stored in the encoding of the data, that the model can sufficiently get trained to achieve perfect classification of the data with 100% accuracy on the validation set. This is achieved earlier for the 21 principle components, at TT-rank $r = 3$, and later for the 16 principle components at TT-rank $r = 6$. In general, it seems that a TT-rank of 5 is sufficient to produce the best performance for the TN model, for any number of features.
In Fig. 5.3, the plots of the accuracy for the training and validation sets are illustrated as the TT-rank increases. In Fig. 5.3a, notice, once again, an unstable behavior from TN, with its accuracy dropping and increasing rapidly for specific values of the TT-rank. In Fig. 5.3b

the training accuracy seems to be much larger than the validation, especially for TT-ranks between 5 and 17. This difference is $\sim 5\%$ on average, so there is not a strong indication for over-fitting to the data. In Fig. 5.3c, the accuracy for both training and validation is much smoother as the TT-rank increases. This is an indicator that the TN model is robust to high-dimensional data. Additionally, there is a small indication of overfitting, since from TT-rank ≥ 4, the training accuracy becomes much larger than the validation accuracy. Although, further experiments need to be conducted, in order to validate that assumption. Finally, for the 16 principal components of Fig. 5.3d, perfect classification is achieved for TT-rank ≥ 5. Both training and validation sets converge to 100% accuracy and there is no indication of instability or over-fitting to the data. Additionally, the same plot for the 21 principal components has been produced, but it is not included, since it is almost identical to Fig. 5.3d. The only minor difference between them is that the perfect classification is achieved with a smaller TT-rank.

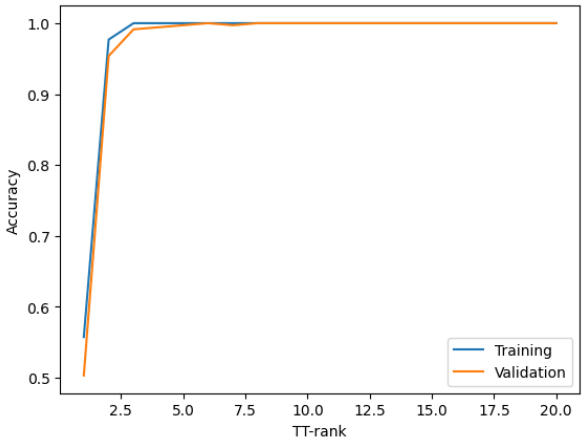


(a) 2 principal components

(b) 5 principal components



(c) 10 principal components



(d) 16 principal components

Figure 5.3: Plots of the accuracy of the tensor network model, with respect to the tested TT-ranks. In these plots, both the validation and training accuracies are depicted, which can give us useful information about the behavior of the model during training.

Similar plots were produced for the VQC model. In Fig. 5.4, we present the plot of the validation accuracy with respect to the number of variational layers, as tested in the experiments. At first glance, notice some major differences from the TN model. First of all, the accuracy of the

model does not increase as the number of qubits increases. This is anticipated, since with an increase in the number of qubits, an increase in the complexity of the quantum circuit follows, which is responsible for worse accuracy. On the other hand, it has been noticed from the experiments that the VQC model requires a lot of optimization time, especially when working with a large number of qubits (≥ 10). As a consequence, it is possible that by training VQC with a higher number of epochs, and with better tuned hyper-parameters, higher accuracy can be achieved. This is of course an assumption that still needs to be tested.

Following Fig. 5.4 notice that the VQC experiment with 2 qubits gives the worst performance among the rest. That peaks at $\sim 73\%$ with a high number of layers. In comparison to TN model, VQC has a slightly better accuracy but still not the overall best. Surprisingly, with 5 qubits, we manage to get the best performance out of VQC with $\sim 91\%$ accuracy on the validation set. Then, the 10 qubits did not perform as well, with a validation accuracy lower than 5 qubits. 10 qubit VQC barely reached $\sim 86\%$, which is still a great performance, in comparison with TN model (which was slightly better). For the 16 qubits, the performance was on average the same as 10 qubits, but it happened to have some large peaks (for 7 variational layers at $\sim 90\%$ validation accuracy and for 15 layers at $\sim 87\%$ validation accuracy) and some low valleys which were under the 10 qubit performance for a specific number of variational layers. Since the complexity of the quantum circuit for 16 qubits is large, the computational power of the GPU used, was not enough for us to train the model for more epochs. In total, $30 - 40$ epochs were used for VQC training. Increasing the number of epochs, might change the experimental lines of Fig. 5.4, so they can converge to some final values, and provide us a more clear understanding of the learning behavior of VQC, as the number of qubits increases. Finally, the experiment with 21 qubits was not included in the plot, since during the training, the maximum accuracy we received was exactly 70.2% for all numbers of variational layers tested. The reason for this result (despite falling on a barren plateau), is the encoding we used for the data. The original features of the data (after one-hot encoding) are binary. Thus, the cosine/sine implementation of them always gave one output state $|0\rangle$ or $|1\rangle$ with probability 1, and thus, the variational part, would be initialized with an input state, that was not a superposition of $|0\rangle$ and $|1\rangle$ (as happened with the other number of qubits under the PCA), but only a state $|i_1 i_2 \dots i_N\rangle = |i_1\rangle \otimes |i_2\rangle \otimes \dots \otimes |i_N\rangle$, where $i_j \in \{0, 1\}$ and $j \in \{1, 2, \dots, N\}$, with N being the total number of qubits.

In Fig. 5.5 the respective plots of the VQC model, regarding the accuracy of both training and validation sets with the number of variational layers, are presented. For all subplots of this figure, it seems that the validation accuracy follows faithfully the training accuracy for all variational layers tested. This means there is no sign of over-fitting or under-fitting to the data, and the gate parameters of the quantum circuit are trained to sufficiently classify "unseen" data.

In general, as the variational layers increase, the accuracy of the model seems to increase as well. The same happens with the TN model and the TT-rank. Some ambiguity might arise from Fig. 5.5c, where for specific numbers of layers, the accuracy of VQC oscillated and dropped for a large number of layers. This unintuitive result might mitigate if the model is trained for more epochs. The reason behind this behavior might also arise from random factors during the optimization process (decaying learning rate, etc.) or even a barren plateau.

Additionally, in order to examine if the VQC model trains as expected and minimizes the loss function, we were able to plot Fig. 5.6. Those are the plots of minimization of the loss function during the training, from randomly chosen experiments (one with 5 qubits and 15 variational layers and one for 16 qubits and 5 variational layers). For both experiments, see the mean

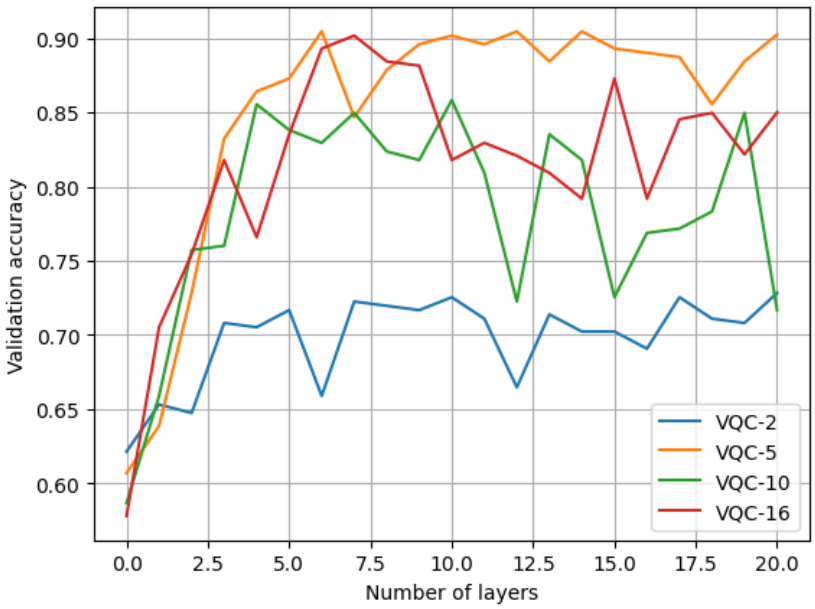


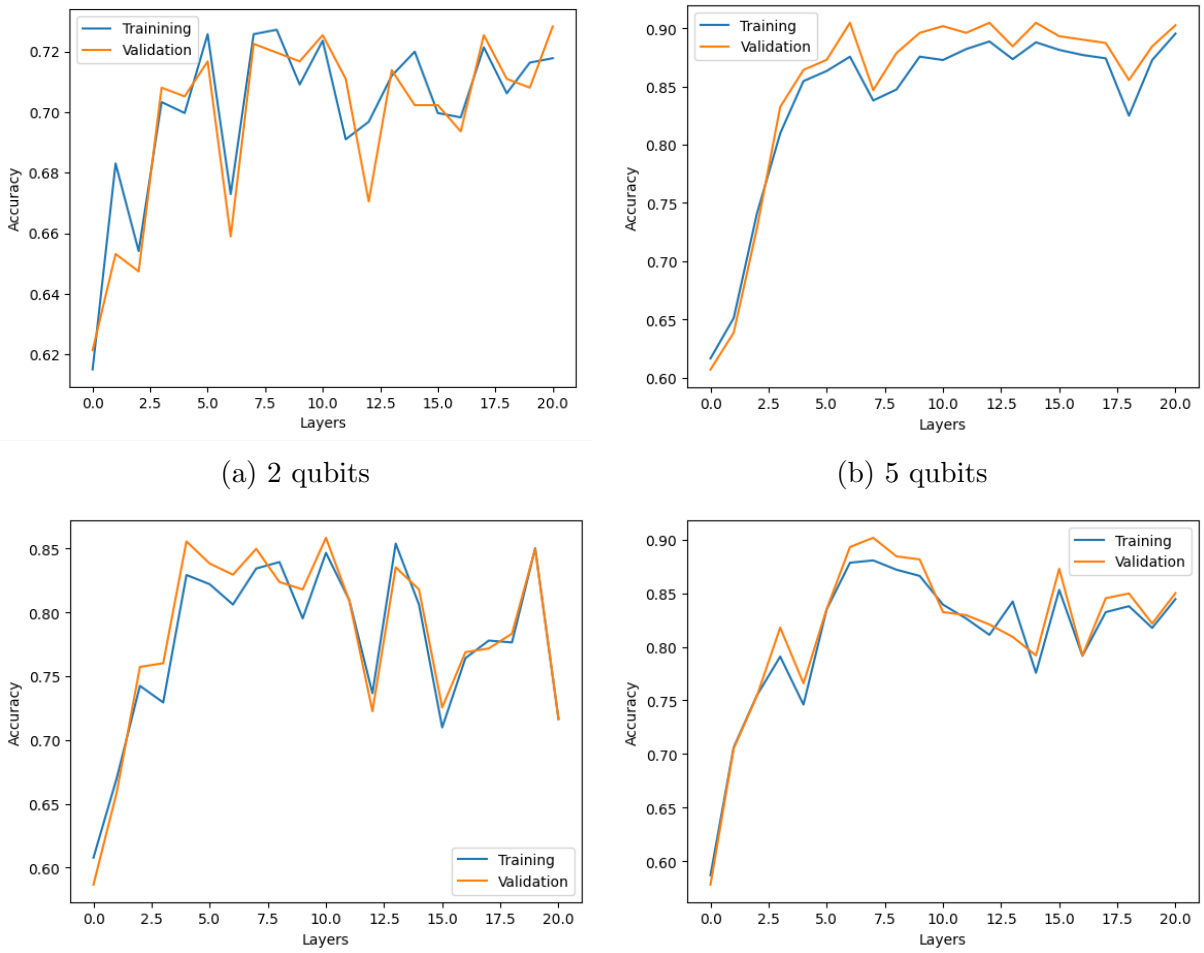
Figure 5.4: Plot of the validation accuracy for the Variational Quantum Classifier with the number of variational layers, for different numbers of qubits (2, 5, 10, 16)

square loss decreases from an initial value ~ 1.60 to almost 0.20 for the 5 qubits and 0.35 for the 16 qubits. This result indicates, that VQC indeed learns to classify the data correctly and it improves its performance, epoch by epoch. Moreover, in Fig. 5.6c and Fig. 5.6d, we can tell with certainty that there is not any abnormal training behavior for VQC. The validation accuracy follows the training accuracy, meaning there is no over-fitting or under-fitting to the data. Of course, as the epochs progress, we see an increase in the accuracy of the model. The accuracy is increased steeply for early epochs, but for later epochs, the accuracy almost converges to $\sim 90\%$ for the 5 qubits and $\sim 81\%$ for the 16 qubits.

Of course, the most important part is to compare the VQC and TN models, to see which has the best performance and when. Also, we examine how well they handle high-dimensional data and did an overall comparison between them.

To compare the models under a common denominator, we chose to plot their validation accuracy with respect to the number of tunable parameters they use. Such plots were prepared, having in mind that the number of parameters of the VQC is $p_{VQC} = N(2L + 1)$ and of the TN is $p_{TN} = 2Nr^2$, where N is the number of qubits/principle components of the model, L the number of variational layers and r the TT-rank used for the TT-decomposition in TN model. Since the parameters of the TN model depend on the square of the TT-rank, we expect that the TN model has significantly more tunable parameters in comparison with VQC (for the same number of qubits and number of layers or TT-rank). For that reason, in the plots, only results that reached 1000 trainable parameters in TN have been used. VQC never reached that many parameters, in any experiments tested. The most parameters reached by VQC are 861 when run for 21 qubits and 20 variational layers. For the aforementioned plots, a fixed number of qubits has been used each time.

In Fig. 5.7 the Validation accuracy of both models is presented, with the number of trainable parameters they used. This is the only plot where we did not depict up to 1000 trainable parameters, but rather up to 100. This happens since the VQC with 2 qubits and 20 variational layers has only $2(40 + 1) = 82$ trainable parameters and the accuracy of TN has converged to



(a) 2 qubits

(b) 5 qubits

(c) 10 qubits

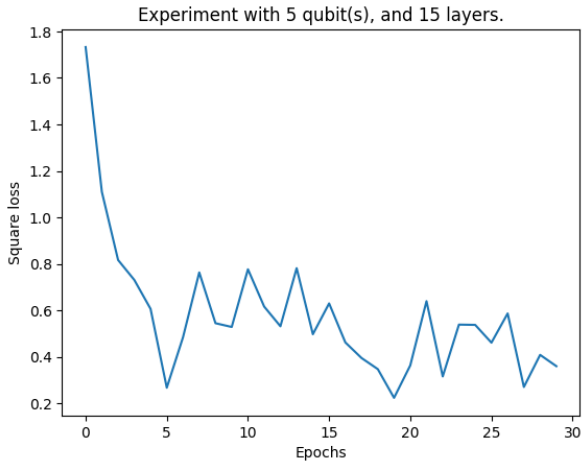
(d) 16 qubits

Figure 5.5: Plots of the accuracy of the variational quantum classifier model, with respect to the tested number of layers (from 0 to 20). In these plots, both the validation and training accuracies are depicted, which can give us useful information about the behavior of the model during the training.

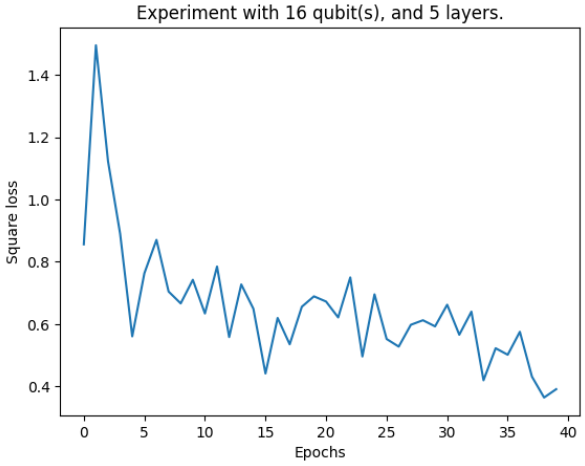
the same value for the remaining parameters.

A general note that we should make at this point is that for the TN model, notice that the validation accuracy lines are much smoother than those of the VQC. This is expected when we examine the scaling of the parameters for the two models. For example, between 0 and 1000 parameters, there are only a few experiments with a fixed number of qubits for TN. On the contrary, since the parameters of VQC scale slower with the depth of the circuit, we get much more experimental points.

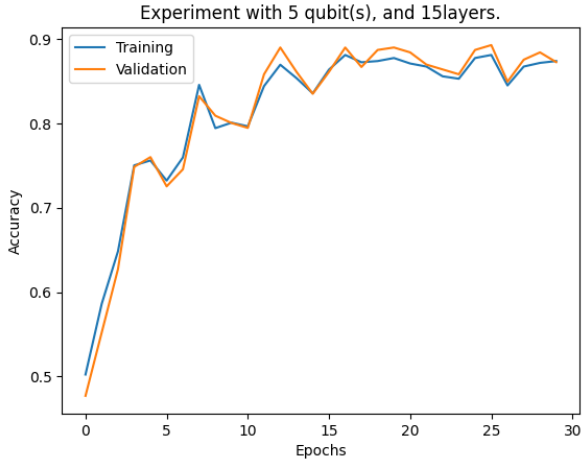
Notice in Fig. 5.7 that both models do not achieve great accuracy. Especially, the TN model performs consistently under 70% accuracy and the VQC model achieves to climb up to 74%. Thus, for 2 qubits, VQC has better performance, even though the overall performance of both models is not great. From this plot, we get an indication that VQC can potentially get even better accuracy with more variational layers than 20, in contrast with TN which seems to converge at 67% validation accuracy. Overall, in order to completely compare the two models, we need to follow the same procedure with experiments with different numbers of qubits or principal components.



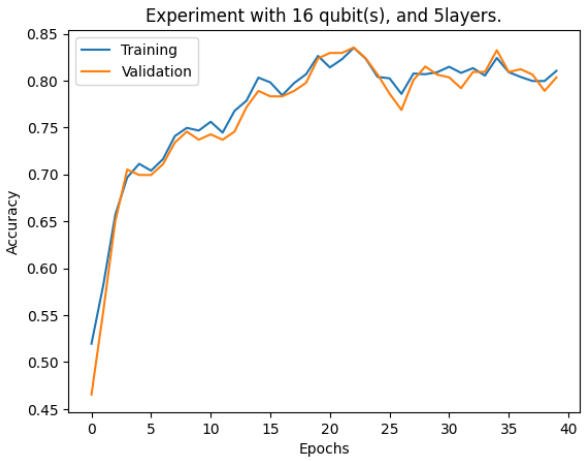
(a) Loss with the number of epochs for 5 qubits and 15 variational layers.



(b) Loss with the number of epochs for 16 qubits and 5 variational layers.



(c) Training and validation accuracies with the number of epochs for 5 qubits and 15 variational layers.



(d) Training and validation accuracies with the number of epochs for 16 qubits and 5 variational layers.

Figure 5.6: Plots from the training of the variational quantum classifier. They are chosen randomly from all the experiments, and depict the minimization of loss during training for 5,16 qubits, and 15,5 variational layers, respectively. Also, the accuracy plots of the training and validation sets during the training are presented for the same number of qubits and variational layers.

In Fig. 5.8, we have the respective plot for 5 qubits. Especially, in Fig. 5.8a, we see up to 1000 trainable parameters, and in Fig. 5.8b we zoom to the region up to 210 trainable parameters. clearly, for 5 qubits, VQC performs much better than TN, achieving a maximum validation accuracy of $\sim 91\%$ with only 65 trainable parameters or 6 variational layers. On the other hand, TN reaches almost 87% accuracy at its peak. In general, its performance oscillates around $85 - 86\%$ which is close to the performance of VQC. As the trainable parameters increase, we notice that TN converges to 85% validation accuracy, in contrast with VQC which oscillates around 90%.
For the respective plots in Fig. 5.9 with 10 qubits, notice that TN gains ground over VQC

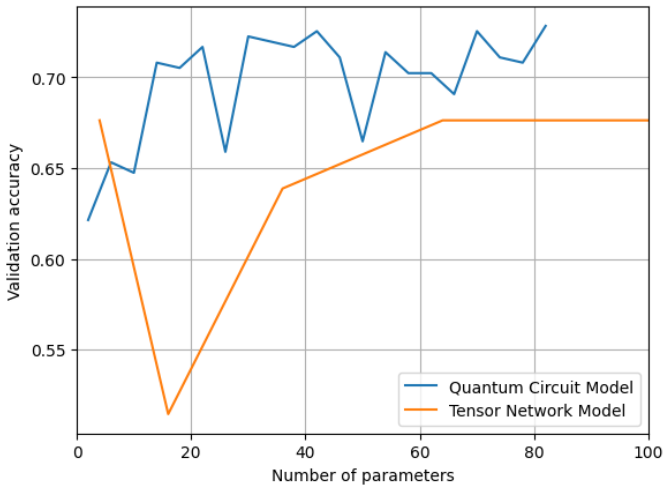
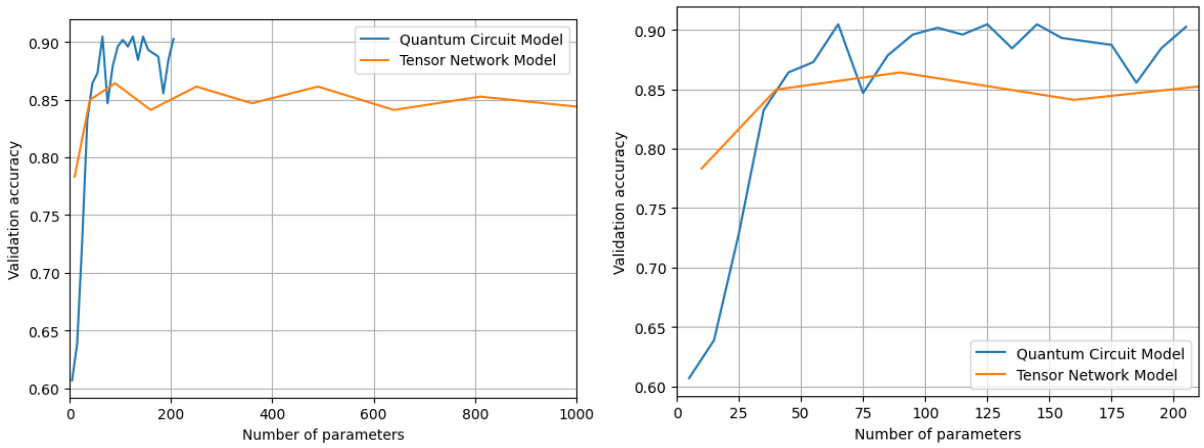


Figure 5.7: The validation accuracy achieved by TN and VQC model with 2 qubits with dependency on the number of trainable parameters used by the models.

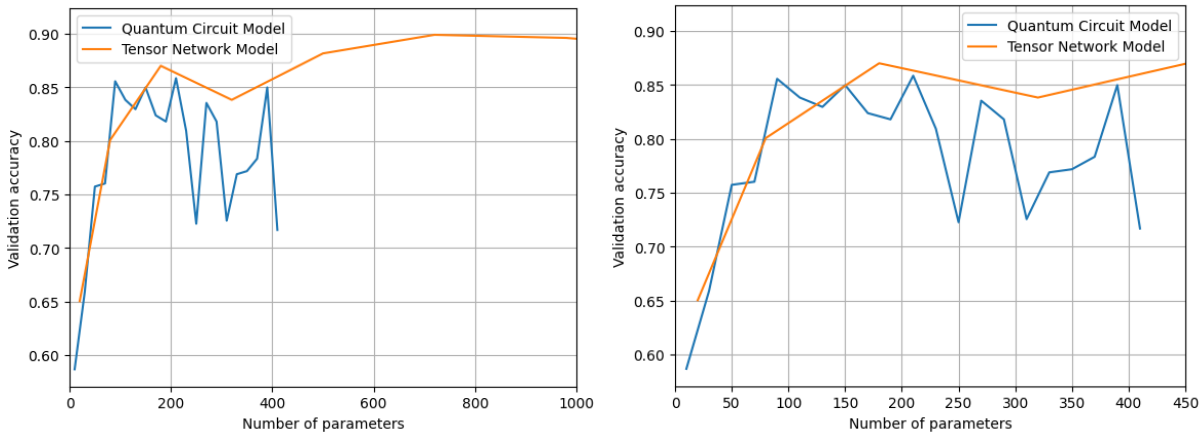


(a) Full plot up to 1000 parameters

(b) Zoomed plot to 210 parameters

Figure 5.8: The validation accuracy achieved by tensor network and variational quantum classifier models with 5 qubits, with dependency on the number of trainable parameters used by the models.

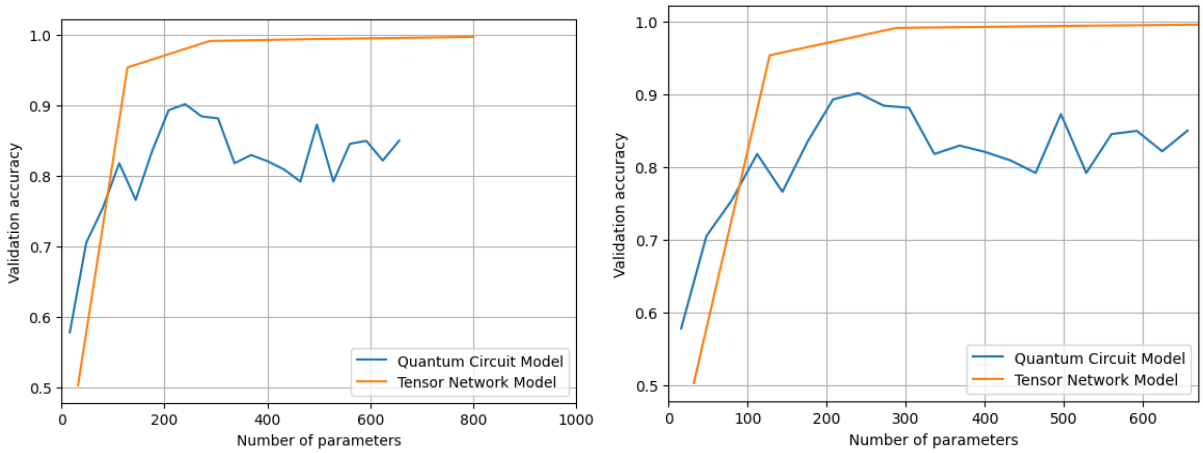
considering it achieves better accuracy than VQC from 150 trainable parameters and above. In a lower number of parameters (or layers) VQC is more steady, increasing its performance almost at the same rate as TN. After $200 - 220$ parameters, it becomes more unstable, with sudden drops and rises in its validation accuracy. In general, TN keeps increasing its performance, with the number of parameters, and seems to max out and converge to 90% validation accuracy, which is comparable with the performance of VQC at 5 qubits.
Finally, for 16 qubits in Fig. 5.10, once again TN seems to be better than VQC, regarding the validation accuracy. TN manages to do perfect classification, converging to 100% accuracy as the number of parameters increases, and VQC shows a more stable behavior in comparison with the 10 qubit experiment. Additionally, VQC reaches 90% validation accuracy at around 240 trainable parameters, which is much better than the performance it had with 10 qubits. As early as 100 parameters, TN starts to become more accurate than VQC without VQC getting



(a) Full plot up to 1000 parameters (b) Zoomed plot to 450 parameters

Figure 5.9: The validation accuracy achieved by tensor network and variational quantum classifier models with 10 qubits, with dependency on the number of trainable parameters used by the models.

over TN at any number of layers above that. VQC does not seem to converge to specific accuracy as the layers/parameters increase but in general, it oscillates around 85% validation accuracy.



(a) Full plot up to 1000 parameters (b) Zoomed plot to 680 parameters

Figure 5.10: The validation accuracy achieved by tensor network and variational quantum classifier models with 16 qubits, with dependency on the number of trainable parameters used by the models.

As stated above, the VQC for 21 qubits could be trained to reach only a single maximum accuracy at 70.2% which seems to be an encoding fault or that the training falls into a barren plateau (which is a much weaker assumption). For that reason, we did not include the respective plots for 21 qubits. TN with 21 features has the same behavior as the 16 principal components TN, but it achieves the perfect classification with slightly fewer parameters (i.e. smaller TT-rank).

5.3 Discussion

Based on Sec. 5, we are not able to claim that one model is clearly better than the other. Accuracy depends on computational power and time. Also, the dimensionality of the data plays a significant role when we want to choose between the models. If the only consideration is the overall accuracy, then TN models seem to be the best choice, especially, if the problem requires high-dimensional data. TT-decomposition enables us to manipulate the high-dimensional weight tensor much faster than other methods. Riemannian optimization seems to be an excellent fit as an optimizer to TN since it outperforms SGD [23] and manages to achieve perfect classification with only TT-rank $r \leq 5$. A drawback of TN is that requires significantly more trainable parameters compared to VQC. Too many parameters, insert more complexity into the model and as a result, tend to slow it down.

On the other hand, VQC equipped with a dimensionality reduction technique (such as PCA), or used on a problem with low-dimensional data, might be a better choice than TN. For example, 5 qubits VQC can reach $\sim 91\%$ validation accuracy on UCI car dataset. We observed that VQC achieved maximum accuracy after a small number of epochs. This is especially promising to reduce the training time of VQCs with a larger number of qubits.

There are indications that VQC can achieve even better accuracy with more qubits. See in Fig. 5.4 with 16 qubits, it is capable to outperform the 5 qubits for $6 - 8$ layers. So with hyper-parameter tuning or more training, VQC might also be suitable for high-dimensional data. However, because training time for the 16 qubits VQC requires much computation time, we did not investigate this possibility as much as it seems to deserve.

Overall, by examining the plots, notice that VQC is consistently the model that achieves the best validation accuracy for a small number of variational layers or low TT-rank. This result strengthens our previous finding about VQC, being preferable when we need to train a model within a limited time frame.

6 Conclusion

To sum up, both models give great accuracy on the binary classification task of the UCI car dataset of 2013. All in all, the tensor network model gave the best overall accuracy, especially, when working on high-dimensional data, achieving perfect classification. Its training, required more trainable parameters, a complex code implementation, and occasionally more computational time in comparison with VQC. On the contrary, VQC performed better on low-dimensional data, where it achieved a maximum validation accuracy of $\sim 91\%$.
Comparatively, for high-dimensional data, the optimization of the quantum circuit of VQC takes significantly more time than the training of a TN. But in low-dimensional data, VQC training times drop remarkably. In addition, observe in Fig. 5.6c and Fig. 5.6d, that just a few epochs are required for VQC, to get trained sufficiently. As a result, the training time for VQC drops even more.

6.1 Future work.

For future work, there are plenty of experiments that can be done to verify the findings of this thesis and to extend the research on tensor networks and variational quantum classifiers in general.
A minor experiment is to test the performance of models while training them for more epochs. In that way, we can validate that their accuracy has indeed converged, and can not be further improved by additional training.
Some major future work is to fix the encoding of VQC, to sufficiently train the model for 21 qubits. Moreover, the experiments conducted with TN used polynomial encoding for the data. So another future work, is to change the encoding of TN to cosine/sine encoding and compare the results.
Of course, we need to know if the models can be generalized to other datasets. So we could try executing exactly the same experiments using different publicly available datasets, with different data sizes and feature dimensionality. Could the models generalize to solve efficiently, a multi-class classification problem?
The optimization of hyper-parameters or hyper-parameter tuning can always produce better results. While implementing the models and training them, we tried to optimize the hyper-parameters. That does not mean they are in the best possible state. So for future work, the hyper-parameter tuning might produce significant improvement in the overall performance of the models. An important example of such hyper-parameter tuning is to choose the number of principal components for PCA that best capture the variance of the dataset. This can be done by producing the scree plot [88], which its bend will indicate the number of significant components.
A final experiment that can be done as future work, is to include a classical machine learning method. In that way, we can compare how well the TN and VQC models behave, in comparison with a classical model, such as Stochastic Gradient Descent (SGD).

Bibliography

[1] Anand Handa, Ashu Sharma, and Sandeep K Shukla. "Machine learning in cybersecurity: A review". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9.4 (2019), e1306.

[2] Wahab Khan et al. "A survey on the state-of-the-art machine learning models in the context of NLP". In: *Kuwait journal of Science* 43.4 (2016).

[3] JS Shyam Mohan et al. "Product recommendation systems based on customer reviews using machine learning techniques". In: *Data Intelligence and Cognitive Informatics: Proceedings of ICDICI 2020*. Springer. 2021, pp. 267–286.

[4] Konstantinos G Liakos et al. "Machine learning in agriculture: A review". In: *Sensors* 18.8 (2018), p. 2674.

[5] Sachi Nandan Mohanty et al. *Machine Learning for Healthcare Applications*. John Wiley & Sons, 2021.

[6] Iqbal H Sarker. "Machine learning: Algorithms, real-world applications and research directions". In: *SN computer science* 2.3 (2021), p. 160.

[7] Roman Rietsche et al. "Quantum computing". In: *Electronic Markets* (2022), pp. 1–12.

[8] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th. USA: Cambridge University Press, 2011. ISBN: 1107002176.

[9] Andreas Bayerstadler et al. "Industry quantum computing applications". In: *EPJ Quantum Technology* 8.1 (2021), p. 25.

[10] Alexey Pyrkov et al. "Quantum computing for near-term applications in generative chemistry and drug discovery". In: *Drug Discovery Today* (2023), p. 103675.

[11] Alberto Di Meglio Elías F. Combarro Samuel González-Castillo. *A Practical Guide to Quantum Machine Learning and Quantum Optimization*. Packt Publishing, 2023.

[12] Wen Guan et al. "Quantum machine learning in high energy physics". In: *Machine Learning: Science and Technology* 2.1 (2021), p. 011003.

[13] Kapil K Sharma. "Quantum machine learning and its supremacy in high energy physics". In: *Modern Physics Letters A* 36.02 (2021), p. 2030024.

[14] Alessio Gianelle et al. "Quantum Machine Learning for b-jet charge identification". In: *arXiv preprint arXiv:2202.13943* (2022).

[15] Timo Felser et al. "Quantum-inspired machine learning on high-energy physics data". In: *npj Quantum Information* 7.1 (2021), p. 111.

[16] Qingyuan Song et al. "Research on quantum cognition in autonomous driving". In: *Scientific reports* 12.1 (2022), p. 300.

[17] Arsenii Senokosov et al. "Quantum machine learning for image classification". In: *arXiv preprint arXiv:2304.09224* (2023).

[18] Yunqian Wang et al. "Development of variational quantum deep neural networks for image recognition". In: *Neurocomputing* 501 (2022), pp. 566–582.

[19] Rui Huang, Xiaoqing Tan, and Qingshan Xu. "Variational quantum tensor networks classifiers". In: *Neurocomputing* 452 (2021), pp. 89–98.

[20] Junde Li et al. "Drug discovery approaches using quantum machine learning". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2021, pp. 1356–1359.

[21] Alexey Pyrkov et al. "Quantum computing for near-term applications in generative chemistry and drug discovery". In: *Drug Discovery Today* (2023), p. 103675.

[22] Maria Avramouli et al. "Unlocking the Potential of Quantum Machine Learning to Advance Drug Discovery". In: *Electronics* 12.11 (2023), p. 2402.

[23] Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. "Exponential machines". In: *arXiv preprint arXiv:1605.03795* (2016).

[24] Marko Bohanec. *Car Evaluation*. UCI Machine Learning Repository. DOI:https://doi.org/10.24432/C5JP48. 1997.

[25] Farzan Jazaeri et al. "A review on quantum computing: From qubits to front-end electronics and cryogenic MOSFET physics". In: *2019 MIXDES-26th International Conference" Mixed Design of Integrated Circuits and Systems"*. IEEE. 2019, pp. 15–25.

[26] Alkım B Bozkurt and Serdar Kocaman. "Linear optical deterministic and reconfigurable SWAP gate". In: *Quantum Information Processing* 20 (2021), pp. 1–12.

[27] Jean-Luc Brylinski and Ranee Brylinski. "Universal quantum gates". In: *Mathematics of quantum computation*. Chapman and Hall/CRC, 2002, pp. 117–134.

[28] Sergey Bravyi and Alexei Kitaev. "Universal quantum computation with ideal Clifford gates and noisy ancillas". In: *Physical Review A* 71.2 (2005), p. 022316.

[29] Tomas Jochym-O'Connor et al. "The robustness of magic state distillation against errors in Clifford gates". In: *arXiv preprint arXiv:1205.6715* (2012).

[30] Dorit Aharonov. "A simple proof that Toffoli and Hadamard are quantum universal". In: *arXiv preprint quant-ph/0301040* (2003).

[31] Konstantin Sakharovskiy. "Universal Quantum Gates". In: (2022).

[32] D.C. Marinescu. *Classical and Quantum Information*. Elsevier Science, 2011. ISBN: 9780123838759. URL: https://books.google.nl/books?id=He-L5CLq1PUC.

[33] Bing Liu and Bing Liu. *Supervised learning*. Springer, 2011.

[34] Salim Dridi. "Supervised Learning-A Systematic Literature Review". In: (2021).

[35] Kai Arulkumaran et al. "A brief survey of deep reinforcement learning". In: *arXiv preprint arXiv:1708.05866* (2017).

[36] Nan Lu et al. "Binary classification from multiple unlabeled datasets via surrogate set classification". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 7134–7144.

[37] Mohd Fadzil Abdul Kadir et al. "Spam detection using machine learning based binary classifier". In: *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)* 26.1 (2022), pp. 310–317.

[38] Fahad B Mostafa and Easin Hasan. "Machine Learning Approaches for Inferring Liver Diseases and Detecting Blood Donors from Medical Diagnosis". In: *medRxiv* (2021), pp. 2021–04.

[39] Kaichao You et al. "How does learning rate decay help modern neural networks?" In: *arXiv preprint arXiv:1908.01878* (2019).

[40] Goran Nakerst, John Brennan, and Masudul Haque. "Gradient descent with momentum—to accelerate or to super-accelerate?" In: *arXiv preprint arXiv:2001.06472* (2020).

[41] Aleksandar Botev, Guy Lever, and David Barber. "Nesterov's accelerated gradient and momentum as approximations to regularised update descent". In: *2017 International joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 1899–1903.

[42] Agnes Lydia and Sagayaraj Francis. "Adagrad—an optimizer for stochastic gradient descent". In: *Int. J. Inf. Comput. Sci* 6.5 (2019), pp. 566–568.

[43] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[44] Gary Bécigneul and Octavian-Eugen Ganea. "Riemannian adaptive optimization methods". In: *arXiv preprint arXiv:1810.00760* (2018).

[45] Will Koehrsen. "Overfitting vs. underfitting: A complete example". In: *Towards Data Science* 405 (2018).

[46] Xue Ying. "An overview of overfitting and its solutions". In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019, p. 022022.

[47] *Avoid overfitting machine learning models*. `https://nl.mathworks.com/discovery/overfitting.html`. Accessed: 2023-06-04.

[48] M Cerezo et al. "Challenges and opportunities in quantum machine learning". In: *Nature Computational Science* 2.9 (2022), pp. 567–576.

[49] Marco Cerezo et al. "Variational quantum algorithms". In: *Nature Reviews Physics* 3.9 (2021), pp. 625–644.

[50] Andrew Arrasmith et al. "Equivalence of quantum barren plateaus to cost concentration and narrow gorges". In: *Quantum Science and Technology* 7.4 (2022), p. 045015.

[51] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. "A survey of dimensionality reduction techniques". In: *arXiv preprint arXiv:1403.2877* (2014).

[52] Naveen Venkat. "The curse of dimensionality: Inside out". In: *Pilani (IN): Birla Institute of Technology and Science, Pilani, Department of Computer Science and Information Systems* 10 (2018).

[53] Eamonn J Keogh and Abdullah Mueen. "Curse of dimensionality." In: *Encyclopedia of machine learning and data mining* 2017 (2017), pp. 314–315.

[54] Sidharth Prasad Mishra et al. "Multivariate statistical data analysis-principal component analysis (PCA)". In: *International Journal of Livestock Research* 7.5 (2017), pp. 60–78.

[55] Jin-Min Liang et al. "Variational quantum algorithms for dimensionality reduction and classification". In: *Physical Review A* 101.3 (2020), p. 032323.

[56] Mikhail Belkin and Partha Niyogi. “Laplacian eigenmaps for dimensionality reduction and data representation”. In: *Neural computation* 15.6 (2003), pp. 1373–1396.

[57] Eysan Mehrbani and Mohammad Hossein Kahaei. “Low-rank isomap algorithm”. In: *IET Signal Processing* 16.5 (2022), pp. 528–545.

[58] Marcello Benedetti et al. “Parameterized quantum circuits as machine learning models”. In: *Quantum Science and Technology* 4.4 (2019), p. 043001.

[59] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A quantum approximate optimization algorithm”. In: *arXiv preprint arXiv:1411.4028* (2014).

[60] Jules Tilly et al. “The variational quantum eigensolver: a review of methods and best practices”. In: *Physics Reports* 986 (2022), pp. 1–128.

[61] Rui Zhang et al. “Quantum support vector machine without iteration”. In: *Information Sciences* 635 (2023), pp. 25–41.

[62] Elies M Gil Fuster. “Variational Quantum Classifier”. In: (2019).

[63] Ryan LaRose and Brian Coyle. “Robust data encodings for quantum classifiers”. In: *Physical Review A* 102.3 (2020), p. 032420.

[64] Zhao Kang, Chong Peng, and Qiang Cheng. “Kernel-driven similarity learning”. In: *Neurocomputing* 267 (2017), pp. 210–219.

[65] Qingfeng Lan. “Variational quantum soft actor-critic”. In: *arXiv preprint arXiv:2112.11921* (2021).

[66] Benoıt Collins and Ion Nechita. “Random matrix techniques in quantum information theory”. In: *Journal of Mathematical Physics* 57.1 (2016).

[67] Jacob C Bridgeman and Christopher T Chubb. “Hand-waving and interpretive dance: an introductory course on tensor networks”. In: *Journal of physics A: Mathematical and theoretical* 50.22 (2017), p. 223001.

[68] Itai Arad and Zeph Landau. “Quantum computation and the evaluation of tensor networks”. In: *SIAM Journal on Computing* 39.7 (2010), pp. 3089–3121.

[69] Liqun Qi et al. “Tensor norm, cubic power and Gelfand limit”. In: *arXiv preprint arXiv:1909.10942* (2019).

[70] Román Orús. “A practical introduction to tensor networks: Matrix product states and projected entangled pair states”. In: *Annals of physics* 349 (2014), pp. 117–158.

[71] David Perez-Garcia et al. “Matrix product state representations”. In: *arXiv preprint quant-ph/0608197* (2006).

[72] Frank Verstraete, Diego Porras, and J Ignacio Cirac. “Density matrix renormalization group and periodic boundary conditions: A quantum information perspective”. In: *Physical review letters* 93.22 (2004), p. 227205.

[73] Yiqing Zhou, E Miles Stoudenmire, and Xavier Waintal. “What limits the simulation of quantum computers?” In: *Physical Review X* 10.4 (2020), p. 041038.

[74] Kirk Baker. “Singular value decomposition tutorial”. In: *The Ohio State University* 24 (2005), p. 22.

[75] ER Henry and J Hofrichter. "[8] Singular value decomposition: Application to analysis of experimental data". In: *Methods in enzymology*. Vol. 210. Elsevier, 1992, pp. 129–192.

[76] Gilbert W Stewart. "On the early history of the singular value decomposition". In: *SIAM review* 35.4 (1993), pp. 551–566.

[77] Tamara G Kolda and Brett W Bader. "Tensor decompositions and applications". In: *SIAM review* 51.3 (2009), pp. 455–500.

[78] Ivan V Oseledets. "Tensor-train decomposition". In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317.

[79] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.

[80] John M Lee. *Riemannian manifolds: an introduction to curvature*. Vol. 176. Springer Science & Business Media, 2006.

[81] E Miles Stoudenmire and David J Schwab. "Supervised learning with quantum-inspired tensor networks. arXiv 2016". In: *arXiv preprint arXiv:1605.05775* ().

[82] Samuel Yen-Chi Chen et al. "Hybrid quantum-classical classifier based on tensor network and variational quantum circuit". In: *arXiv preprint arXiv:2011.14651* (2020).

[83] Artem A Melnikov et al. "Quantum state preparation using tensor networks". In: *Quantum Science and Technology* (2023).

[84] Ville Bergholm et al. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2022. arXiv: 1811.04968 [quant-ph].

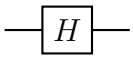
[85] Ivan Oseledets. *TT-Toolbox*. https://github.com/oseledets/TT-Toolbox. Retrieved June 16, 2023. 2023.

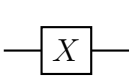
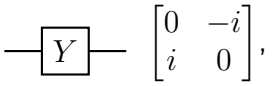
[86] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

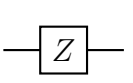
[87] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

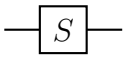
[88] Gibbs Y Kanyongo. "Determining the correct number of components to extract from a principal components analysis: a Monte Carlo study of the accuracy of the scree plot". In: *Journal of modern applied statistical methods* 4.1 (2005), p. 13.

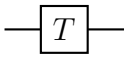
Quantum gates glossary

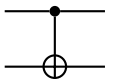
Hadamard  $\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$

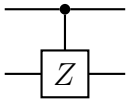
Pauli-X  $\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$, Pauli-Y  $\begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}$,

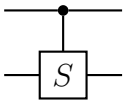
Pauli-Z  $\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$

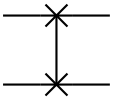
Phase  $\begin{bmatrix}1 & 0\\ 0 & i\end{bmatrix}$

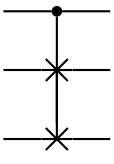
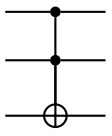
$\frac{\pi}{4}$-rotation  $\begin{bmatrix}1 & 0\\ 0 & e^{i\frac{\pi}{4}}\end{bmatrix}$

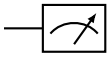
Controlled-NOT  $\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$

Controlled-Z  $\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & -1\end{bmatrix}$

Controlled-phase  $\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & i\end{bmatrix}$

swap  $\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$

Fredkin  $\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}$, Toffoli  $\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\end{bmatrix}$

Measurement  Projection of the measured qubit onto $|0\rangle$ and $|1\rangle$.

qubit — Quantum wire, carrying a single qubit from left to right.

classical bit = Wire carrying a classical bit from left to right.

n qubits $\not{}^{n}$ Quantum wire carrying n qubits from left to right.