Universiteit Leiden

Master Computer Science

Near-term quantum algorithms for regression, overfitting analysis and regularization

Name:	Luuk Visser
Student ID:	s1707264
Date:	31/07/2022
Specialisation:	Artificial Intelligence
1st supervisor:	Dr. Vedran Dunjko
2nd supervisor:	Dr. Alfons Laarman
Daily supervisors:	Casper Gyurik, MSc
	Charles Moussa, MSc

Master's Thesis in Computer Science

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

Abstract

Parametrized quantum circuits can serve as supervised learning models executed on near-term quantum computers. In previous work these models have been applied to classification and bivariate regression problems. However, their performance on real-world regression datasets has yet to be evaluated. Furthermore, very little is known about the generalization performance of these models or regularization methods to improve this performance. In this thesis we evaluate these models on a real-world multivariate regression dataset and obtain similar performance to many commonly used classical regression methods. Furthermore we train the models on artificially generated datasets and empirically show a decrease in generalization performance of circuits with repeated encoding of the data. We evaluate the effect of several regularization methods and show that these methods do not positively affect the generalization capacities of these circuits. Finally, we propose layer-cancelling parametrized quantum circuits and show these can be regularized successfully. This work therefore shows parametrized quantum circuits can be effectively applied to real-world regression problems and provides useful insights and methods to improve the generalization performance of these circuits.

Acknowledgements

This project has been carried out in the Applied Quantum Algorithms group. I would like to thank Vedran Dunjko for the opportunity to conduct this research project, as well as his frequent guidance and his flexible supervision. Furthermore, I want to thank Casper Gyurik and Charles Moussa for their many contributions during our meetings and their availability to answer my questions. Finally I would like to thank Alfons Laarman for his work as second supervisor.

Contents

1 Introduction

Regression problems deal with the task of finding the relationships between a single dependent variable and multiple independent variables. These problems have been studied for centuries and early mathematical models used to tackle regression problems can be traced back to the beginning of the 19th century. With the rise of computers in the 20th century more computationally expensive mathematical models have become prevalent, with artificial neural networks as primary example. These models have an increased expressivity, meaning they can generally be used to handle complex learning problems more accurately. Besides the developments in regression algorithms, the success of modern regression models can largely be attributed to the continuing increase in available computational power.

This increase in computational power is slowing down however, as the transistor size in modern computers is approaching physical limits [1]. Furthermore, there are multiple classes of problems which are believed to be impossible to solve efficiently on a classical computer. An example of such a problem is the simulation of quantum mechanical systems. The mathematical description of a quantum mechanical system generally grows exponentially in the size of the system. Because of this exponential growth large quantum mechanical systems become intractable to simulate on classical computers. This intractability guided physicists to the idea of using a new model of computation based on quantum mechanics.

This idea has started a wide branch of research based on quantum computing. Besides the large potential of efficiently simulating physical systems, this research has lead to many more algorithms with a variety of potential applications. Clever usage of quantum mechanical phenomena in these algorithms has in many cases led to a significant improvement in time complexity compared to the corresponding classical algorithm. The most well-known example is Shor's algorithm [2], which factors integers in polynomial time while the fastest known classical algorithm can only do this in sub-exponential time [3]. In the current era however, state-of-the-art quantum computers are relatively small-scale and often have to deal with noise, likely causing many of the potential applications (like Shor's algorithm) to be many years away. Therefore much of the current research is focused on algorithms which are not crucially affected by noise and use a lower amount of qubits, and can therefore obtain good results on near-term devices [4].

One class of currently researched quantum algorithms uses so-called parametrized quantum circuits (PQCs). With the help of classical computers to optimize the parameters, these circuits can serve as machine learning models for both classification and regression problems. PQCs map data into a high-dimensional feature space, which can be intractable to simulate on a classical computer [5]. The increased complexity of these models poses an interesting open question whether or not these feature spaces are useful for classical machine learning tasks.

Additionally, from classical machine learning we know that an increase in model complexity can come with problems of overfitting, where the performance of the model on test instances is much worse than its performance on the training instances. These problems are tackled using regularization techniques, which limit the complexity of the trained models to increase the test performance of the model. The increased complexity provided by parametrized quantum circuits gives rise to the open question to what extent overfitting can occur within these models.

In this thesis, the learning capabilities of the PQC are tested on a classical regression dataset through simulations on classical computers. Furthermore, we investigate the presence of overfitting in PQC-based regression models. Finally, we experiment with several regularization techniques on the models with overfitting behaviour.

2 Preliminaries

2.1 Quantum computing

2.1.1 Qubits/quantum states

In classical computers, information is stored using binary digits, or bits. Each bit is described by its state, which can be either 0 or 1. By combining multiple bits, larger pieces of information can be represented by the resulting bit string.

In quantum computers, information is represented by quantum bits, or qubits. A qubit is a quantum-mechanical system which can take a similar form as classical bits (denoted using so-called bra-ket notation as $|0\rangle$ and $|1\rangle$), as well as a linear combination (or superposition) of these two states. Mathematically speaking, the state of a qubit $|\psi\rangle$ can be described by a two-dimensional complex unit vector [6]. This description can be specified in different bases. Using the most commonly used basis ($\{|0\rangle, |1\rangle\}$), also known as the computational basis, a qubit can therefore be written in the form

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \qquad \text{where} \qquad \alpha, \beta \in \mathbb{C} : |\alpha|^2 + |\beta|^2 = 1 \tag{1}$$

When α and β are both nonzero, we say that the state is in a superposition over the states $|0\rangle$ and $|1\rangle$.

Similarly to a single qubit, a system with multiple qubits can be represented by a state vector. More specifically, a state vector with n qubits is a unit vector in the 2^n-dimensional complex Hilbert space.[1] In this vector space, the computational basis is defined as the set of possible bitstrings of length n. As an example, the combined state of two qubits can be written as

$$|\psi\rangle = \alpha |00\rangle + \beta |01\rangle + \gamma |10\rangle + \delta |11\rangle = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \quad \text{where} \quad \alpha, \beta, \gamma, \delta \in \mathbb{C} : |\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1 \tag{2}$$

Generally, when we have two separate systems described by the states $|\psi_a\rangle$ and $|\psi_b\rangle$ respectively, their states can be combined using the tensor product with the resulting combined state equal to $|\psi\rangle = |\psi_a\rangle \otimes |\psi_b\rangle \equiv |\psi_a\rangle |\psi_b\rangle$.

2.1.2 Computations

Quantum states are evolved by unitary transformations. In quantum computers, these transformations are applied using quantum gates. These gates can act on single qubits or on multiple qubits

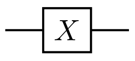
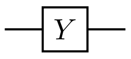
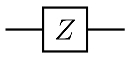
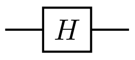
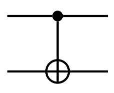
[1]Technically, this is a description of so-called pure states. Qubits can also be in mixed states, which are statistical ensembles of pure states. Throughout this thesis we will not encounter mixed states and therefore only refer to pure states.

at the same time. Quantum gates can be combined into a quantum circuit, which serves as the model for quantum computations.

Circuits are often represented mathematically by unitary matrices in the computational basis. The resulting state after running a quantum circuit with unitary transformation U on a system of qubits with initial state $|\psi\rangle$ is given by $U|\psi\rangle$.

A full computation is usually illustrated using a circuit diagram. Such a diagram shows the initial state of the qubits and connects these to gates using wires from left to right to illustrate the order of the gates applied to each specific qubit. Table 1 depicts the circuit symbols of some of the commonly used gates, along with their matrix representation in the computational basis.

Table 1. Common quantum gates with corresponding matrix representations in the computational basis. The first four gates are applied to a single wire each, indicating that these gates are single-qubit gates. The CNOT gate is applied to two wires and is therefore a two-qubit gate. An operation is applied to the second qubit (the target) conditional on the state of the first qubit (the control).

Name	Gate	Matrix representation
Pauli-X		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Controlled Not (CNOT)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Besides the fixed gates in Table 1, there are three common gates that each depend on a parameter. These can be found in Table 2. As we will see in Section 2.1.4, each of the operations performed by these parametrized operators can be geometrically represented as a qubit rotation by an angle θ around a specific axis. When used in circuits, changing the parameters of these gates allows the computation performed by the circuit to be tuned.

2.1.3 Measurements

As described in previous sections, we can use unitary operators to transform quantum states. Unlike in classical computing, where we can observe the value of each bit at any given time and then perform more calculations with it afterwards, in quantum computing such observations can only be performed using measurements which can permanently influence the state of the system. Therefore, in most quantum computations these measurements take place at the end of the computation to

Table 2. Parametrized quantum gates. Each gate represents a qubit rotation by an angle θ around its specific axis (x, y, or z).

Name	Gate	Matrix representation
$R_x(\theta)$	$-\boxed{R_x(\theta)}-$	$e^{-iX\theta/2} = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$
$R_y(\theta)$	$-\boxed{R_y(\theta)}-$	$e^{-iY\theta/2} = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$
$R_z(\theta)$	$-\boxed{R_z(\theta)}-$	$e^{-iZ\theta/2} = \begin{bmatrix} \exp(-i\theta/2) & 0 \\ 0 & \exp(i\theta/2) \end{bmatrix}$

extract classical information from the quantum system. The circuit symbol used for a measurement on a single qubit is depicted in Figure 1.

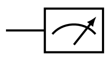


Figure 1. Measurement symbol

To be able to describe how measurements work, we write the complex inner product between two states $|x\rangle, |y\rangle \in \mathbb{C}^{2^n}$ as

$$\langle x|y\rangle := |x\rangle^\dagger |y\rangle = \sum_i \bar{x}_i y_i$$

Here, † denotes Hermitian conjugation.

The possible outcomes of a measurement depend on $|\psi\rangle$, the state of the system before the measurement is performed, and the so-called observable which is measured, O. This observable acts as an Hermitian operator. Each of its eigenvalues $\{\lambda_i\}$ represents a possible outcome from the measurement and its corresponding eigenvector λ_i represents the state of the system after obtaining this outcome from the measurement. Because an observable is Hermitian, it can be described using its spectral decomposition:

$$O = \sum_i \lambda_i |\lambda_i\rangle \langle\lambda_i|$$

where $\langle\lambda_i| = |\lambda_i\rangle^\dagger$ and acts as a functional on $|\psi\rangle$, such that

$$O|\psi\rangle = \sum_i \lambda_i |\lambda_i\rangle \langle\lambda_i|\psi\rangle = \sum_i \lambda_i \langle\lambda_i|\psi\rangle |\lambda_i\rangle$$

The probability of obtaining the outcome λ_i from the measurement is given by the Born rule

$$p(\lambda_i) = \big|\langle\lambda_i|\psi\rangle\big|^2$$

From this expression, the expected outcome of the measurement can be derived to be

$$E(O, |\psi\rangle) = \langle\psi|O|\psi\rangle$$

On quantum computers, this expected outcome can be estimated through sampling by executing the circuit multiple times and measuring each final state.

An important result of the way in which measurements work is the physical irrelevance of the global phase, the phase of the entire system. To illustrate this, we can add any global phase θ to a state $|\phi\rangle$ such that $|\psi\rangle = e^{i\theta}|\phi\rangle$.

For the measurement of observable O on this system, the probability of measuring any eigenvalue λ_i on the state $|\psi\rangle$ is therefore

$$p(\lambda_i, |\psi\rangle) = \big|\langle\lambda_i|\psi\rangle\big|^2 = \big|e^{i\theta}\langle\lambda_i|\phi\rangle\big|^2 = \big|e^{i\theta}\big|^2\big|\langle\lambda_i|\phi\rangle\big|^2 = \big|\langle\lambda_i|\phi\rangle\big|^2 = p(\lambda_i, |\phi\rangle)$$

The global phase of a state therefore has no effect on the probabilities of the measurement outcomes.

2.1.4 Bloch sphere representation

The general description of a single-qubit state (eq. 1) uses two complex numbers, which means four real numbers are needed to describe the state of the system. However, the insignificance of the global phase allows one of the complex coefficients to be fixed to a real positive value without changing the physical state being represented. Using only three real numbers a, b and ϕ, the state of a single qubit can therefore be described as

$$|\psi\rangle = a|0\rangle + be^{i\phi}|1\rangle \qquad \text{where} \qquad a, b \in \mathbb{R}_{\geq 0} : a^2 + b^2 = 1$$

Furthermore, due to the unit vector-constraint on the state of the qubit we can represent the state as a point on a unit sphere in three dimensions. The real values a and b can be parametrized using this constraint, yielding

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}e^{i\phi}|1\rangle \qquad \text{where} \qquad 0 \leq \theta \leq \pi \text{ and } 0 \leq \phi \leq 2\pi$$

The sphere representing all single-qubit states is called the Bloch sphere. Its visual representation is depicted in Figure 2. In this representation the x, y, and z-axes are defined by the eigenvector corresponding to the positive eigenvalue of the Pauli-X, Pauli-Y and Pauli-Z matrix respectively.

2.1.5 Entanglement

The final gate in Table 1 depicts a two-qubit gate known as the Controlled-NOT (CNOT) gate. If the control qubit is in the state $|0\rangle$, this gate has no effect on the system. If the control qubit is in the state $|1\rangle$, the gate effectively applies the Pauli-X gate to the target qubit, i.e. $|0\rangle$ is transformed to $|1\rangle$ and vice-versa.

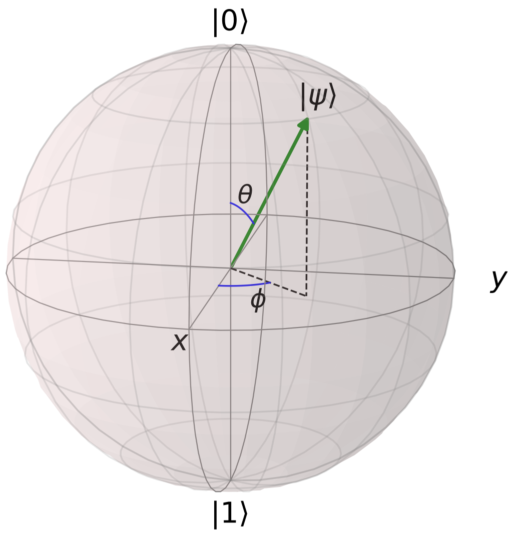


Figure 2. Bloch-sphere representing all possible single-qubit states. The z-axis points in the direction of $|0\rangle$. The position of the state $|\psi\rangle$ on the sphere is determined by the angles θ and ϕ.

An interesting phenomenon happens when the state of the control qubit is in a superposition. As example, let us take $|\psi_0\rangle = H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ as the control qubit and $|\psi_1\rangle = |0\rangle$ as the target qubit. The combined state of this two-qubit system is

$$|\psi\rangle = |\psi_0\rangle \otimes |\psi_1\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$$

We then apply the CNOT gate to this system, as shown in Figure 3.

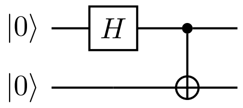


Figure 3. Circuit containing a CNOT gate where the state of the control qubit is in a superposition.

The final state after running this circuit is

$$|\psi_f\rangle = \text{CNOT}\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

The state of the this two-qubit system $|\psi_f\rangle$ is such that we can no longer describe the state of each qubit separately, i.e. $\nexists\,|\psi_a\rangle, |\psi_b\rangle \in \mathbb{C}^2 : |\psi_f\rangle = |\psi_a\rangle \otimes |\psi_b\rangle$. The states of the two qubits are therefore entangled. This phenomenon has interesting consequences when we perform measurements. If we measure the control qubit, there is a probability of $\frac{1}{2}$ of measuring $|0\rangle$ and the state of the system collapsing to $|00\rangle$, and an equal probability of measuring $|1\rangle$ and the state collapsing to $|11\rangle$. If we subsequently measure the target qubit, this will give us the same outcome as the measurement of the first qubit. Entangled systems therefore introduce correlations into the measurement of multiple qubits.

2.2 Regression

2.2.1 Problem statement

In regression, one aims to predict a real-valued target variable using given independent variables by learning the function f mapping the independent variables to the target variable. This is done through supervised learning, where a dataset of N instances $D = \{\boldsymbol{X_i}, y_i\}_{i=1}^{N}$ is used. Here $\boldsymbol{X_i} \in \mathbb{R}^m$ is a vector of m features, which are independent variables. $y_i \in \mathbb{R}$ is a target, which is a dependent variable that depends on $\boldsymbol{X_i}$ as well as an error term $\epsilon_i \in \mathbb{R}$ such that

$$y_i = f(\boldsymbol{X_i}) + \epsilon_i$$

Each instance $(\boldsymbol{X_i}, y_i)$ of the dataset can be used to train a regression model. This mathematical model acts as a function $\hat{f}_D(\boldsymbol{x})$ with $\boldsymbol{x}$ as a feature vector. The goal of training this model is to have $\hat{f}_D$ represent the function f as accurately as possible, so that in a situation where the value of the target variable is unknown, the model can use the independent variables $\boldsymbol{x}$ to estimate its value.

Regression models generally are trained by making the output $\hat{f}_D(\boldsymbol{x})$ depend on a set of parameters $\boldsymbol{\theta}$ and optimizing these parameters such that they minimize the error of the predictions made by the model on the instances in the dataset. Two commonly used error measures (also known as loss functions) are the mean squared error (MSE) and the root mean squared error (RMSE). When calculated on a set of n instances, with $\boldsymbol{y}$ as the vector of target values in the dataset and $\boldsymbol{\hat{y}}$ as the vector of target values predicted by the model, these error measures are given by:

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

2.2.2 Classical models

The most well-known class of regression models is the class of multiple linear regression models. These models assume there are one or more linear relations between the independent variables and the target variable, i.e. the output function of the model on a vector of independent variables $\boldsymbol{x}$ is

$$\hat{f}_D(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^{n}\theta_j x_j$$

Here, the assumed linear relations are described by the parameters $\boldsymbol{\theta}$, which are often optimized using the so-called least-squares method. Using this method on a dataset X with n instances containing m features each, the parameters $\boldsymbol{\theta}$ which minimize the mean-squared error are related to $\boldsymbol{y}$ through the Moore-Penrose inverse of X, i.e.

$$\boldsymbol{\theta} = (X^T X)^{-1} X^T \boldsymbol{y}$$

where $\boldsymbol{y}$ is the vector of target values in the dataset and X is a $n \times m$-matrix such that X_{ij} denotes the j-th feature of the i-th instance in the dataset.

Another commonly used class of models is the class of artificial neural networks, which can be used for a wide range of machine learning tasks including regression tasks. The number of parameters used in these models can range from a few parameters to billions of parameters. Artificial neural networks can therefore be much more complex than linear regression models, so they are more suitable for datasets with non-linear relations.

2.2.3 Overfitting

As discussed earlier, regression models are used to estimate an unknown target variable y from a given feature vector $\boldsymbol{x}$, where $y = f(\boldsymbol{x}) + \epsilon$. The goal of training regression models is then to use the training dataset $D = \{\boldsymbol{X_i}, y_i\}_{i=1}^{N}$ to find a function $\hat{f}_D$ which represents the function f as closely as possible. However, as each y_i is known while each $f(\boldsymbol{X_i})$ is not, the task of minimizing the error on the training dataset means representing each y_i as accurately as possible and therefore also implicitly taking into account the error terms ϵ_i. When $\hat{f}_D$ conforms too closely to the training dataset, it will therefore not accurately represent the function f, which in turn means the performance of the model on instances outside of the training dataset may be much worse. This phenomenon is a common problem in supervised learning methods called overfitting. Overfitting usually happens when the model used is capable of representing functions which are more complex than the target function f. An example in which this happens is depicted in Figure 4.

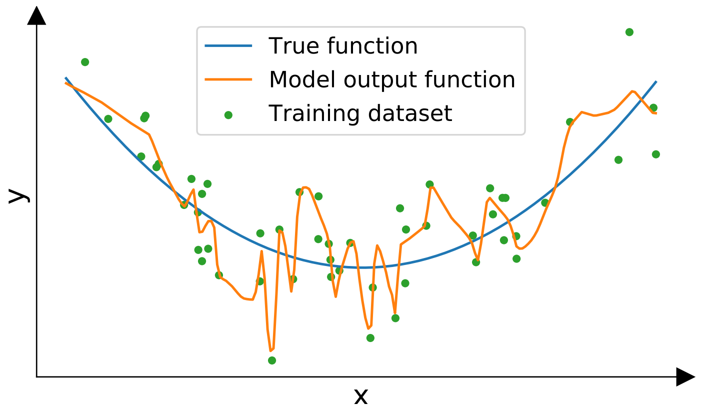


Figure 4. Example of a model suffering from overfitting. The model output function $\hat{f}_D$ represents the data points much more accurately than the true function f and therefore often diverts from f.

Because of the possibility of overfitting, a proper evaluation of the trained model cannot be performed on data used for training. Therefore datasets are usually split up into two datasets: one for training and one for validation. The validation dataset is used to evaluate the performance of the model on unseen data not used in training. In this way, different models can be compared using their performance on the validation dataset.

A disadvantage of using a validation dataset for evaluation, is the fact that the evaluation now depends on what part of the dataset is used for validation. To reduce the variance this introduces into the evaluation metric, a method called k-fold cross-validation can be used. With this method, the dataset is split into k groups. Each group is used once as validation dataset while a model is trained on the other $k-1$ groups. After training k models, the performance on their respective validation datasets is averaged to obtain the cross-validated performance.

2.2.4 Bias-variance trade-off

When trying to prevent overfitting, one has to deal with the so-called bias-variance trade-off. Both the bias and the variance are part of the expected prediction error that a model will make on an instance $(\boldsymbol{x}, y) \notin D$, where D denotes the training dataset. To define these two errors, we define $P(\boldsymbol{x}, y)$ as the distribution from which each instance in D has been drawn, i.e. $D \sim P^N$. We then define the expected validation error on an unseen instance $(\boldsymbol{x}, y)$ as

$$\mathbb{E}_D\left[(\hat{f}_D(\boldsymbol{x}) - y)^2\right] := \mathbb{E}_{D\sim P^N}\left[(\hat{f}_D(\boldsymbol{x}) - y)^2\right]$$

It can be shown that

$$\mathbb{E}_D\left[(\hat{f}_D(\boldsymbol{x}) - y)^2\right] = \underbrace{\mathbb{E}_D\left[\left(\hat{f}_D(\boldsymbol{x}) - \mathbb{E}_D[\hat{f}_D(\boldsymbol{x})]\right)^2\right]}_{\text{Variance}} + \underbrace{\left(\mathbb{E}_D[\hat{f}_D(\boldsymbol{x})] - f(\boldsymbol{x})\right)^2}_{\text{Bias}^2} + \underbrace{\left(y - f(\boldsymbol{x})\right)^2}_{\text{Noise}}$$

In this equation, the noise represents the error term ϵ which is added to $f(\boldsymbol{x})$ to form the instance $(\boldsymbol{x}, y)$. This term is independent of the model, and therefore forms a lower bound on the expected validation error.

The bias represents the error caused by assumptions made within the design of the model. A high bias means these assumptions generally tend to result in a model for which its output value $\hat{f}_D(\boldsymbol{x})$ is very different from $f(\boldsymbol{x})$. This is usually the case when a model is not complex enough to capture the dependence of the true function f on the feature vector $\boldsymbol{x}$. Bias generally decreases as more complex models are used.

Finally, the variance represents the degree to which $\hat{f}_D(\boldsymbol{x})$ fluctuates depending on what dataset is sampled. When the variance is high, these fluctuations tend to result in models for which the error achieved when evaluated at $\boldsymbol{x}$ is very dependant on the training data. This is often the case when the model is complex enough to accurately represent the noise in the training data, and therefore no longer accurately represents the true function f, i.e. the model is susceptible to overfitting. Variance generally increases as more complex models are used.

Because bias decreases when more complex models are used while variance decreases as less complex models are used, decreasing the expected validation error of a regression model generally equates to finding the right balance in this bias-variance trade-off. This is illustrated in Figure 5.

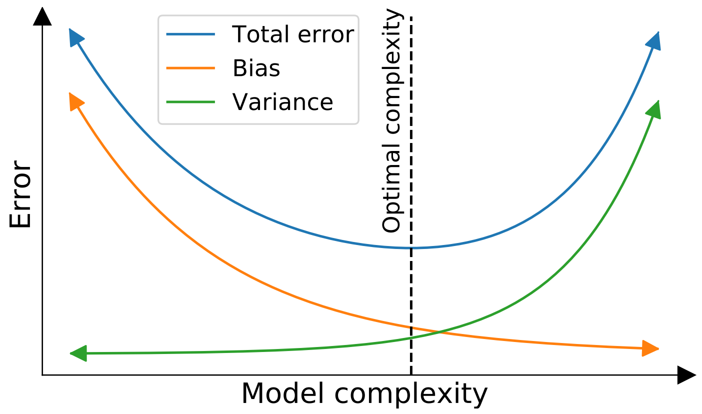


Figure 5. Influence of the model complexity on the bias, variance and the resulting total validation error. To achieve the lowest possible total error, it is necessary to use a model with the right complexity.

2.2.5 Regularization

Overfitting can be reduced using so-called regularization methods which limit the complexity of the model. These methods increase the bias of the model and decrease its variance. Therefore regularization methods can be used to achieve a better balance between these two quantities. Two very common regularization methods are L1 and L2-regularization, which add an additional term to the loss function which is optimized. This loss function will therefore be a sum of the mean squared error and a regularization penalty.

With L1-regularization, the resulting loss function is given by

$$L = \mathrm{MSE} + \lambda \left|\left|\boldsymbol{\theta}\right|\right|_1 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{p} |\theta_i|$$

where θ_i is the i-th parameter in the model and λ is a hyperparameter used to weight the regularization penalty. This form of regularization penalizes any increase of the model parameters, therefore leading to a sparse model where the parameters tend towards zero.

L2-regularization works similarly to the L1-regularization, but with a slightly different penalty. Instead of using the L1 norm of the parameter vector $\boldsymbol{\theta}$, we use the L2 norm so that the loss function becomes

$$L = \mathrm{MSE} + \lambda \left|\left|\boldsymbol{\theta}\right|\right|_2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{p} \theta_i^2$$

This type of regularization has a different effect from L1-regularization, as it mainly penalizes the large parameter values and leaves the smaller parameter values relatively unchanged.

2.3 Parametrized quantum circuits

On quantum computers, one way to create a regression model is through parametrized quantum circuits (PQCs). These circuits make use of parametrized gates which can be tuned to change the possible measurement outcomes and probabilities. PQCs are part of a hybrid quantum-classical algorithm, where the model is evaluated through the parametrized circuit on a quantum computer, and the parameters of the model $\boldsymbol{\theta}$ are optimized on a classical computer. The model can produce predictions by sampling the expected measurement outcome through multiple runs of the same circuit. The original structure of a PQC as proposed in [7] is depicted in Figure 6a. As shown, the

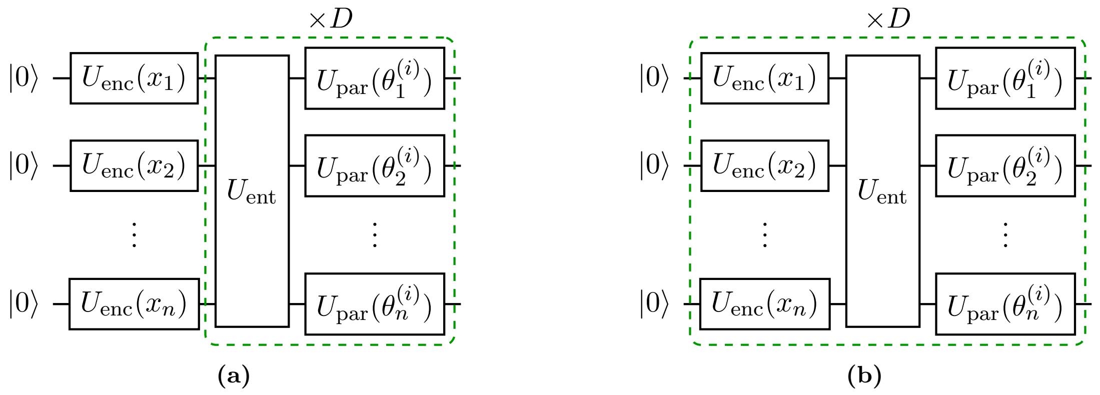


Figure 6. Circuit layout without data reuploading (a) and with data reuploading (b). In either circuit the highlighted area is repeated D times, where D denotes the depth of the circuit. Here U_{enc} represents a single-qubit encoding operation, U_{ent} represents an n-qubit entangling operation, and U_{par} denotes a single-qubit parametrized operation.

circuit is built from encoding, entangling and parametrized components. Each encoding component depends on the feature x_i and therefore embeds x_i into the state of the n-qubit system. Each entangling component uses multi-qubit gates (e.g. the CNOT gate) to introduce entanglement into the system. Finally, the parametrized gates embed the parameters $\boldsymbol{\theta}$ into the system. The gates used in each of these components can vary widely and are usually adjusted to the specific learning problem.

An alternate PQC structure is depicted in Figure 6b, where the encoding components have become a part of the repeated section. With this method, known as data reuploading [8], the feature vector $\boldsymbol{x}$ is encoded into the circuit multiple times.

The parameters of the circuit $\boldsymbol{\theta}$ are optimized on a classical computer using gradient descent. The most common method to calculate the gradient of the expected measurement output with respect to each parameter is the parameter-shift rule [7, 9]. With this method the parametrized quantum circuit can be used to calculate these gradients. For a circuit with p parameters, this method requires estimation of $2p$ expected outcomes of the circuit.

3 Related work

3.1 Development of parametrized quantum circuits

The first experiments and developments with hybrid quantum-classical optimization algorithms were not aimed specifically at machine learning tasks, but rather at the task of state preparation. More specifically, they were developed to be used in conjunction with quantum algorithms like quantum phase estimation (QPE) and quantum expectation estimation (QEE) [10]. QPE and QEE require an approximation of the ground state of a given Hamiltonian, i.e. the lowest energy state of a quantum mechanical system. Earlier methods to obtain this approximation, like adiabatic evolution [11], are difficult to perform on near-term quantum devices because they require long coherent evolutions. This motivated early work on hybrid quantum-classical optimization algorithms in order to replace these earlier methods with more feasible algorithms for near-term devices [12, 10].

3.2 Parametrized quantum circuits as machine learning models

Early work on PQCs as supervised learning models has been performed by Mitarai et al. [7] In their work, they successfully train PQCs on datasets generated using univariate functions, as well as classification datasets.

Chen et al. [13] investigated the overfitting capabilities in a regression context as well as a classification context. Specifically, they experiment with the PQCs used in [7], which do not use data reuploading. They found that for increasing circuit depth, the increase in expressive power diminishes. In their experimental results they therefore do not observe significant overfitting.

In the field of quantum support-vector machines, Park et al. [14] used visualizations of the decision boundary to demonstrate overfitting. This decision boundary was smoothened significantly with the use of L2-regularization.

Gyurik et al. [15] proved that depending on the ansatz used, the complexity of quantum linear classifiers can be tuned by varying the rank of the observable. Furthermore the authors showed that better generalization performance can be achieved by tuning the Frobenius norm of the observable.

Pérez-Salinas et al. [8] introduced the concept of data reuploading to parametrized quantum circuits. Instead of only encoding the data into the circuit at the start of the circuit, the data is encoded into the circuit multiple times. Schuld et al. [16] showed that data reuploading significantly increases the expressivity of PQCs.

In their research performed simultaneously with the research described in this thesis, Kobayashi et al. [17] successfully applied a regularization method to their PQC-based models trained on bivariate regression datasets. This method randomly removes entangling gates during each training iteration, improving the generalization performance of the model.

3.3 Contributions

In this thesis, research is performed on the use of parametrized quantum circuits for multivariate regression problems. Specifically, three main contributions are provided. Firstly, we show that parametrized quantum circuits can be applied to real-world multivariate regression datasets and can obtain a similar performance to many classical regression methods. Secondly, we use artificially generated datasets to show that parametrized quantum circuits with data reuploading can suffer from severe overfitting which increases as deeper circuits are used. Finally, we evaluate the effect of several regularization techniques and provide an adapted PQC-architecture which can be regularized through cancelling of circuit layers.

4 Methodology

4.1 Classical simulation

Quantum computers are still in early stages of development and currently mainly used for very specific experiments. Our experiments are therefore simulated on classical computers using the TensorFlow Quantum-package [18]. This package is developed to help the development of machine learning algorithms for quantum computers. This software uses the Cirq-package [19] as a basis to create quantum circuits, and incorporates this with classical optimization algorithms to create quantum machine learning algorithms.

Classical simulation of PQCs has advantages and disadvantages. The time complexity scales exponentially in the number of qubits, so the number of qubits is very limited in a classical simulation. On the other hand, classical simulation allows access to the final state of the circuit without using measurements. Therefore classical simulators can directly calculate the expected measurement outcome, compared to quantum computers only being able to sample this outcome through measurements.

On quantum computers, determining the gradient of the expected measurement output with respect to each parameter is usually performed using the earlier described parameter-shift rule. For classical simulation of PQCs, there is a faster method to calculate the gradients known as adjoint differentiation [20]. This method requires one forward pass through the circuit to determine the final state, and one backwards pass through the circuit to calculate all gradients. Because all our experiments are simulated classically, this iterative method is used.

Using adjoint differentiation, a variety of gradient-based optimization algorithms can be used to tune the parameters of the circuit. For all experiments the Adam [21] optimizer was chosen. This optimizer uses stochastic gradient descent with an adaptive learning rate. The adaptation depends on the first-order and second-order moments of the past gradients, and decay rates are used to decrease the significance of older gradients. The adaptive learning rate reduces the importance of the initial learning rate and therefore significantly reduces the amount of hyperparameter tuning needed.

4.2 Parametrized quantum circuit components

For the experiments performed in this work, we use PQCs without data reuploading (depicted earlier in Figure 6a) as well as PQCs with data reuploading (depicted earlier in Figure 6b).

4.2.1 Encoding components

The feature-vector $\boldsymbol{x}$ is encoded into the circuit by applying rotational gates (as defined earlier in Table 2) to multiple qubits. This encoding, proposed in [7], has been extended for usage with multiple features such that

$$U_{\text{enc}}(x_j) = R_z(\cos^{-1} x_j^2) R_y(\sin^{-1} x_j)$$

The domain of this encoding is $[-1, 1]$ and the range when applied to an input state $|0\rangle$ is illustrated in Figure 7. For all experiments, the number of qubits is chosen as either once or twice the number

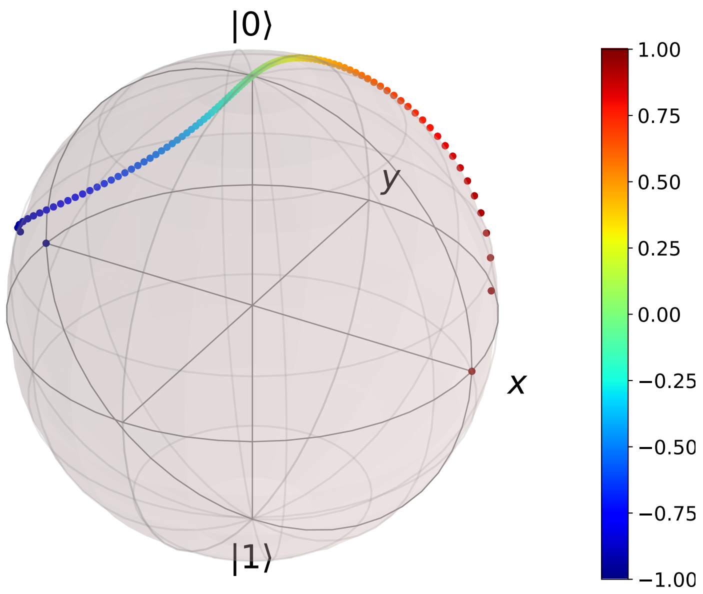


Figure 7. Bloch-representation of qubit encoding $R_z(\cos^{-1} x_j^2) R_y(\sin^{-1} x_j)\,|0\rangle$, with x_j ranging from -1 (dark blue) to 1 (dark red). The input variable x_j here is not to be confused with the x-axis.

of features in the dataset. In both cases, the j-th feature is encoded into the j-th qubit. In the second case this encoding is repeated for the second half of the qubits.

4.2.2 Entangling components

The entangling layer is composed out of CNOT gates. In our experiments we use two different layouts for the entangling layer, i.e. ring and full connectivity. These are illustrated in Figure 8.

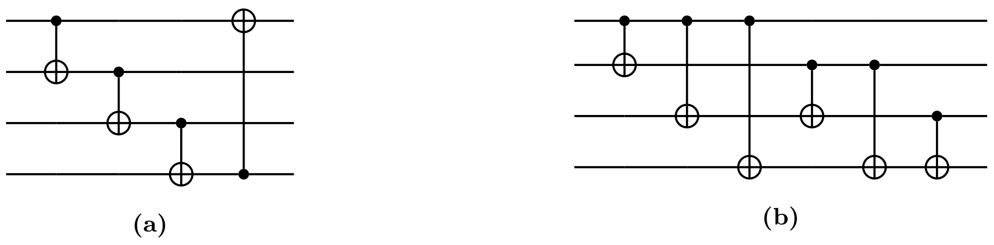


Figure 8. CNOT entanglement with ring (a) and full (b) connectivity. With ring connectivity only neighbouring qubits are connected (including the last and first qubit). Full connectivity means all pairs of qubits are connected unidirectionally, with the order of the qubits determining the control (first) and target (second) qubits.

4.2.3 Parametrized components

For the parametrized layer, the unitary operator acting on the j-th qubit in the i-th parametrized layer is given by

$$U_{\mathrm{par}}(\boldsymbol{\theta}_j^{(i)}) = R_y(\theta_{j1}^{(i)}) R_z(\theta_{j2}^{(i)})$$

All parameters are initialized to uniformly random values in the interval $[-\pi, \pi]$.

4.2.4 Measurement

Two measurement observables are used in the majority of the experiments. The first observable is denoted by Z_0, which measures Z on the first qubit. The second observable is $Z^{\otimes N}$, which measures Z on all qubits.

4.3 Grid search on real-world dataset

To evaluate the performance of parametrized quantum circuits (PQCs) on real-world regression datasets, we train many different PQCs on the Combined Cycle Power Plant (CCPP) dataset [22]. This dataset was collected from a power plant with the goal of predicting its electrical power output from the ambient temperature, atmospheric pressure, relative humidity, and the exhaust steam pressure.

The goal of the first experiment is to find the parametrized quantum circuits which work best on the CCPP dataset and to compare this performance with classical regression methods. In order to do that we perform a grid search of multiple circuits with multiple hyperparameters. The hyperparameters used in this grid search are described in Table 3. All features and targets in the dataset are rescaled to the interval [-1,1]. This is necessary because the features are encoded into qubit rotations and the domain of the measurement is limited by the observable. The errors obtained from the model are rescaled back accordingly for comparison to the methods used in the original paper.

Table 3. Hyperparameters used in hyperparameter grid search for parametrized quantum circuits trained on the CCPP dataset.

Hyperparameter	Values
Entanglement connectivity	$\{\text{ring}, \text{full}\}$
Measurement observable	Z_0
Number of qubits	$\{4, 8\}$
Circuit depth	$\{1, 3, 5, 7, 9\}$
Learning rate	$\{0.003, 0.01, 0.03\}$
Batch size	$\{32, 64\}$
Epochs	200

4.4 Artificial datasets

Our second set experiments is designed to observe overfitting of parametrized quantum circuits and to identify in what conditions overfitting occurs. For this experiment, artificial datasets are generated using randomly initialized PQCs. More specifically, for each instance of the dataset four uniformly random features are generated within the interval [-1, 1]. The target corresponding to these features is determined by the expected outcome of measuring an observable after running a randomly initialized PQC of depth 2. Other than the depth of the generator PQC, all other circuit properties (e.g. measurement observable, data reuploading) are set to be the same as the PQC that is trained on this generated dataset. The generated datasets consist of 1000 instances each. The hyperparameters used for the models which are trained are described in Table 4.

Table 4. Hyperparameters used in experiments with artificially generated datasets.

Hyperparameter	Values
Data reuploading	$\{\text{with}, \text{without}\}$
Entanglement connectivity	$\{\text{ring}, \text{full}\}$
Measurement observable	$\{Z_0, Z^{\otimes N}\}$
Number of qubits	4
Circuit depth	$\{2, 4, 6, 10, 13, 16, 20\}$
Learning rate	0.005
Batch size	64
Epochs	200

The PQCs used as models in this experiment vary in depth. The use of higher-depth models means highly complex models are used to fit relatively simple functions generated by low-depth models. Models which are not prone to overfitting, will have a similar training and validation error, even at higher circuit depths. On the other hand, we expect to see a large difference between these two errors for high-depth circuits which are susceptible to overfitting.

For models which do not show overfitting, we perform three separate experiments to test their resistance to overfitting. Firstly, Gaussian noise is added to the target variables in the generated datasets. As discussed in Section 2.2.3, errors added to the target variables be a source of overfitting. Secondly, the size of the training dataset is varied, as the use of small training datasets can cause overfitting. Finally, we experiment with a different observable, which is specified in matrix form in the computational basis by $\text{diag}(-1, 0, 1, 2, ..., 2^n - 2)$, where n is the number of qubits in the circuit. This observable has n distinct eigenvalues and has a larger Frobenius norm than the Z_0 and $Z^{\otimes n}$ observables. This is motivated by [15], in which these properties of the observable were shown to control the complexity of the model.

4.5 Regularization methods

To determine whether or not overfitting PQC-based models can be successfully regularized, we experiment with several regularization techniques.

4.5.1 L1 and L2-regularization

In the first of these experiments we use L1 and L2-regularization, introduced earlier in Section 2.2.5. Because we are working with PQCs where the parameters are rotational angles and are therefore periodic, the angles are wrapped to the $[-\pi, \pi]$ domain immediately after every training update. This leaves the output of the PQC unchanged while ensuring the regularization penalty is based on the true magnitude of each rotation.

4.5.2 Parameter dropout procedure

Next, we use a different method from L1-regularization to enforce a fraction of the model parameters to be zero, which we will call 'parameter dropout'. With this method, the model is fully trained multiple times. After each full training iteration, the smallest parameter is fixed to zero for the remainder of the experiment, essentially dropping the parametrized rotations corresponding to that parameter. For this experiment, two different procedures are tried. In the first procedure, the parameters are randomly reinitialized after each full training iteration except for the parameters which are fixed to zero. In the second procedure, the values of the parameters are preserved between each full training iteration, with the newly fixed parameters as the only change to be made between two iterations. This experiment is performed with models using a depth of 16 layers.

4.5.3 Layer-cancelling procedure

In our final regularization method, we change the structure of our PQCs to allow them to perform the exact same operations as the lower-depth PQCs used to generate the datasets. The unitary corresponding to the i-th repeated layer of a PQC using data reuploading can be denoted as

$$U_{\text{layer}}(\boldsymbol{x}, \boldsymbol{\theta}^{(i)}) = U_{\text{enc}}(\boldsymbol{x}) U_{\text{ent}} U_{\text{par}}(\boldsymbol{\theta}^{(i)})$$

where U_{enc} is the unitary corresponding to the entire encoding layer, U_{ent} is the unitary corresponding to the entire entangling layer and U_{par} is the unitary corresponding to the entire parametrized layer, which is a function of the i-th vector of parameters $\boldsymbol{\theta}^{(i)}$ applied in the i-th parametrized layer. Our first modification to this repeated layer is the addition of an altered entangling layer such that a full layer becomes

$$U_{\text{layer}}(\boldsymbol{x}, \boldsymbol{\theta}^{(i)}) = U_{\text{enc}}(\boldsymbol{x}) U_{\text{ent}} U_{\text{par}}(\boldsymbol{\theta}^{(i)}) U_{\text{ent}}^{\dagger} \tag{3}$$

The use of this new layer allows the circuit to cancel out the entangling layer. To see this, remember that $U_{\text{par}}(\boldsymbol{\theta}^{(i)})$ only consists of single-qubit rotations using $\boldsymbol{\theta}^{(i)}$ as input angles. Therefore

$$U_{\text{par}}(\mathbf{0}) = I$$

and thus

$$\begin{aligned} U_{\text{layer}}(\boldsymbol{x}, \mathbf{0}) &= U_{\text{enc}}(\boldsymbol{x}) U_{\text{ent}} U_{\text{par}}(\mathbf{0}) U_{\text{ent}}^{\dagger} \\ &= U_{\text{enc}}(\boldsymbol{x}) U_{\text{ent}} U_{\text{ent}}^{\dagger} \\ &= U_{\text{enc}}(\boldsymbol{x}) \end{aligned}$$

So with this first modification, which we will call 'entanglement-cancelling', a full layer can in theory reduce to a single encoding layer.

In the second modification we use two full layers, each using the entanglement-cancelling method described above. The second full layer is modified such that U_{ent} is replaced with its Hermitian conjugate $U_{\text{ent}}^{\dagger}$. Because our original entangling layer consists of parametrized rotational gates, its Hermitian conjugate can be implemented by reversing the order of the gates and flipping the sign of the input angles.

Within the full circuit, the layers alternate between using U_{ent} and $U_{\text{ent}}^{\dagger}$. Therefore two subsequent layers U_1 and U_2 can be described as

$$U_1(\boldsymbol{x}, \boldsymbol{\theta}^{(i)}) U_2(\boldsymbol{x}, \boldsymbol{\theta}^{(i+1)}) = (U_{\text{enc}}(\boldsymbol{x}) U_{\text{ent}} U_{\text{par}}(\boldsymbol{\theta}^{(i)}) U_{\text{ent}}^{\dagger})(U_{\text{enc}}^{\dagger}(\boldsymbol{x}) U_{\text{ent}} U_{\text{par}}(\boldsymbol{\theta}^{(i+1)}) U_{\text{ent}}^{\dagger})$$

With the first and this second modification combined, which we will call 'layer-cancelling', the encoding circuits of two subsequent layers can cancel each other out if the first parameter vector $\boldsymbol{\theta}^{(i)}$ is set to $\mathbf{0}$. This gives

$$\begin{aligned} U_1(\boldsymbol{x}, \mathbf{0}) U_2(\boldsymbol{x}, \boldsymbol{\theta}^{(i+1)}) &= (U_{\text{enc}}(\boldsymbol{x}) U_{\text{ent}} U_{\text{par}}(\mathbf{0}) U_{\text{ent}}^{\dagger})(U_{\text{enc}}^{\dagger}(\boldsymbol{x}) U_{\text{ent}} U_{\text{par}}(\boldsymbol{\theta}^{(i+1)}) U_{\text{ent}}^{\dagger}) \\ &= U_{\text{enc}}(\boldsymbol{x}) U_{\text{enc}}^{\dagger} U_{\text{ent}} U_{\text{par}}(\boldsymbol{\theta}^{(i+1)}) U_{\text{ent}}^{\dagger} \\ &= U_{\text{ent}} U_{\text{par}}(\boldsymbol{\theta}^{(i+1)}) U_{\text{ent}}^{\dagger} \end{aligned}$$

Therefore effectively the two layers act together as a single layer without any encoding gates. In the special case where the second parameter vector $\boldsymbol{\theta}^{(i+1)}$ is also set to $\mathbf{0}$, the unitary matrix of both layers combined reduces to identity and the effective depth of the circuit is reduced by two layers.

We experiment with layer-cancelling in combination with several degrees of L1-regularization to guide the optimization procedure towards parameter vectors containing elements equal to zero.

5 Results

5.1 Classical dataset performance

To compare the performance of parametrized quantum circuits to classical regression methods, we perform a grid search of different circuits with multiple hyperparameters. For this grid search, the Combined Cycle Power Plant dataset is used. In the original paper on this dataset by Tufekci, a large variety of classical regression methods were applied to the dataset. The authors used 5 x 2 cross-validation to evaluate their models, i.e. shuffling the dataset 5 times and performing two-fold cross-validation on each of those 5 datasets. With methods as linear regression and the multilayer perceptron (MLP) the authors reached a RMSE of 4.561 and 5.399 respectively. The best performance was obtained with the bagging REP tree method, which obtained a RMSE of 3.779. With the use of genetic algorithms to tune the structure of their multilayer perceptron, Lorencin et al. [23] improved the performance of the MLP to a RMSE of 4.305.

Our models are evaluated using 5 x 2 cross-validation on the same shuffled datasets used in the original CCPP paper. We compare circuits by choosing the batch size and learning rate with which it obtained the best cross-validation score. The results of training PQCs on the CCPP dataset are shown in Table 5. The results show that one of the two largest PQCs in the experiment, with 8 qubits, a depth of 9 and ring connectivity in the entangling layer, obtains the best performance. With a RMSE of 4.255 this model is outperformed by the multilayer perceptrons created by Lorencin et al., but it performs better than the multilayer perceptron used in the original paper by Tufekci.

Additionally, we observe only small differences in performance among all models with a depth of at least 3. The models using a depth of 1 perform substantially worse than all other models.

Table 5. Performance of different parametrized quantum circuits on the CCPP dataset. The best performing learning rate and batch size combination has been selected for each set of circuit hyperparameters, with the achieved performance shown in the last column.

Qubits	Depth	Connectivity	Learning rate	Batch size	RMSE performance
4	1	full	0.003	64	16.991
		ring	0.030	32	16.908
	3	full	0.010	32	4.489
		ring	0.010	64	4.633
	5	full	0.003	32	4.474
		ring	0.030	64	4.370
	7	full	0.010	32	4.502
		ring	0.010	64	4.442
	9	full	0.030	64	4.608
		ring	0.030	64	4.592
8	1	full	0.003	32	16.995
		ring	0.030	32	16.803
	3	full	0.010	64	4.315
		ring	0.010	64	4.673
	5	full	0.010	32	4.290
		ring	0.030	64	4.546
	7	full	0.010	64	4.346
		ring	0.010	64	4.281
	9	full	0.003	64	4.519
		ring	**0.003**	**64**	**4.255**

5.2 Overfitting capabilities

As the PQCs used in the previous experiment obtained a relatively good validation error, for this experiment we use artificially generated datasets to observe overfitting and to identify when overfitting occurs. These datasets are generated using low-depth PQCs and are used to train different circuit architectures at different depths. For this experiment 5 x 2 cross-validation is used. A new dataset is generated for each of the five repetitions and the final score is an average of the ten runs performed. Figure 9 shows the learning performance of the PQCs without and with the use of data reuploading on data generated with PQCs of depth 2. Regardless of the observable we observe different behaviour depending on the use of data reuploading. The set of models using only a single encoding layer show an improvement in both the training and the validation error as deeper models are used. This behaviour is not present when we look at the set of models using data reuploading. These models only show an improvement in the training error as deeper models are used, while the validation error increases as a function of the model depth used. In other words, as models of

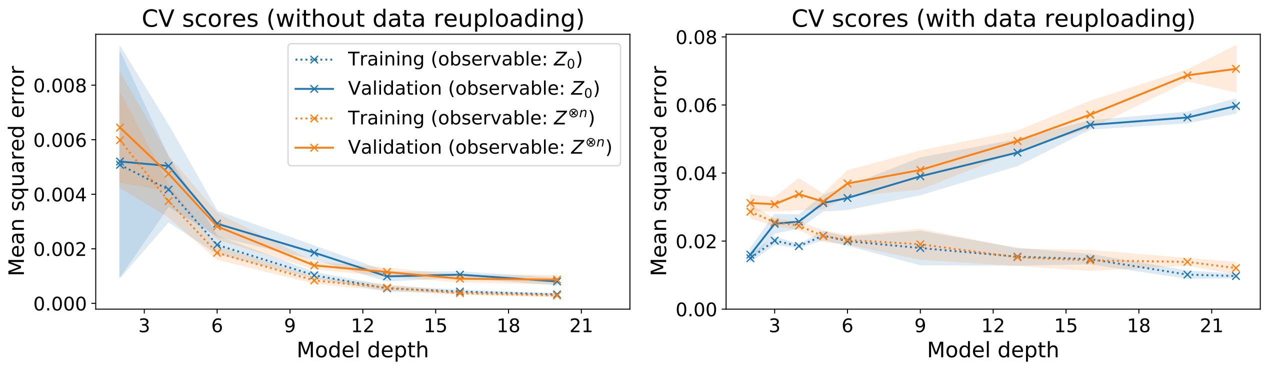


Figure 9. Cross-validated performance of models without and with the use of data reuploading on both the training and validation set. Note that each unique model is trained on datasets generated with a low-depth version of that model. The legend applies to both plots.

higher depth and therefore more expressive power are used, their performance becomes worse on unseen data. Therefore the models using data reuploading are a good example of overfitting in this experiment.

We perform additional experiments to see if we can observe a similar form of overfitting in models without data reuploading as we have seen in models with data reuploading. In these three experiments we add noise to the dataset, vary the size of the training dataset and change the observable measured respectively. Again, every experiment is performed with models of different depths and lower-depth PQCs are used to generate the dataset.

We start by analyzing the behaviour of the PQC-based models when different amounts of Gaussian noise are added to the targets in these datasets. The results of this experiment can be found in Figure 10.

Again we see that both the training and validation error decrease when deeper models are used. When we add a large amount of noise ($\sigma = 0.1$), there is a small gap between the training and validation error. However, this gap does not increase as the depth of the models used increases. Therefore, within this experiment the addition of Gaussian noise does not cause deeper models to overfit.

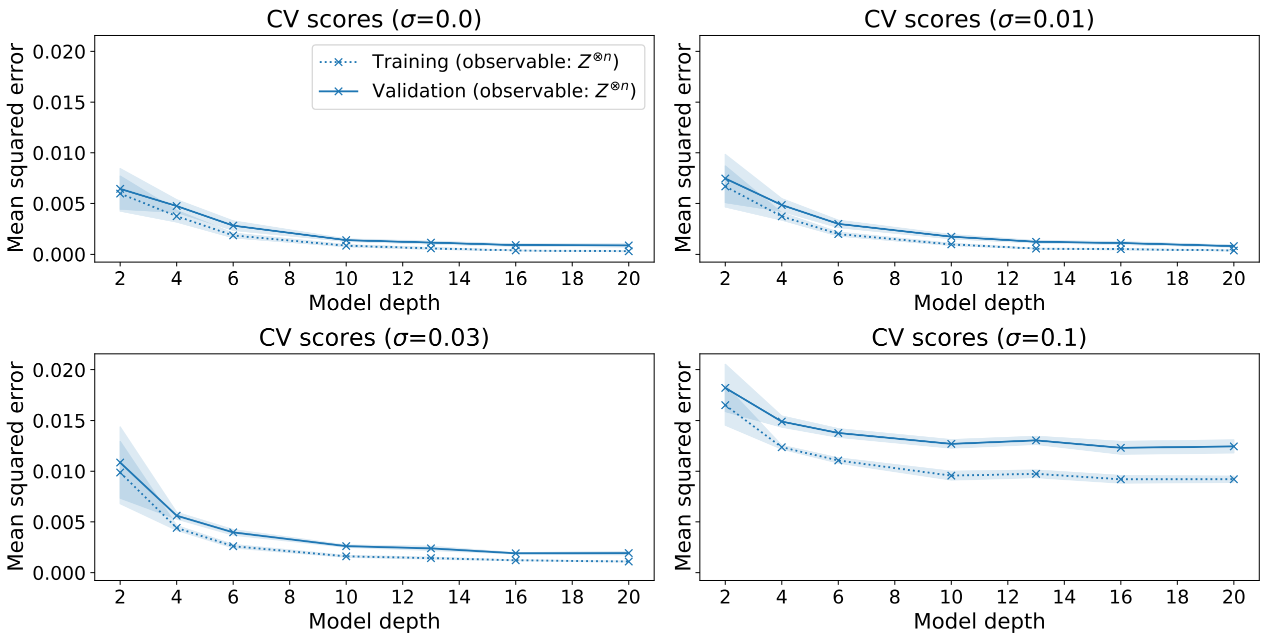


Figure 10. Cross-validated performance of models without data reuploading on both the training and validation set with different amounts of noise added to the targets in the dataset.

For the second experiment to investigate the overfitting capabilities of PQCs using a single encoding layer, we perform the same experiment without noise and instead vary the size of the training dataset. The size of the validation dataset is preserved to maintain the accuracy of the evaluation. The results of this experiment can be found in Figure 11. Using smaller training set sizes, we observe a larger gap between the training and validation error achieved at higher depths. However, this gap does not increase as a function of the depth and the validation error decreases as deeper PQCs are used. Therefore this cannot be qualified as the type of overfitting observed in earlier in Figure 9 with PQCs using data reuploading.

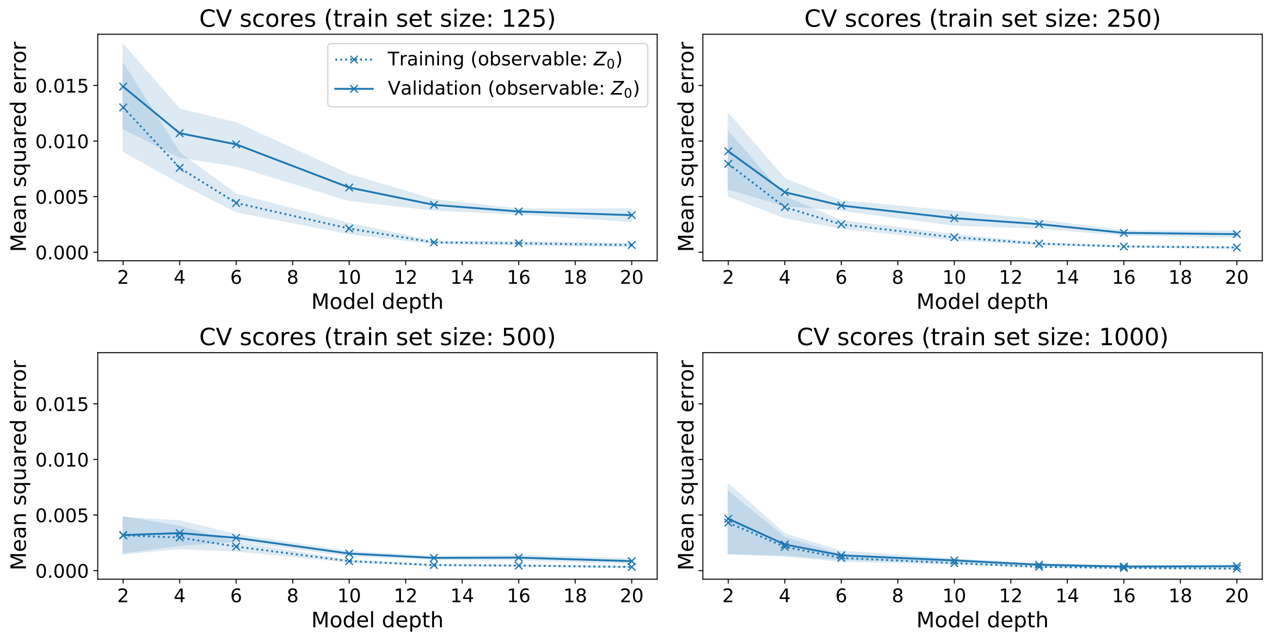


Figure 11. Cross-validated performance of models without data reuploading on both the training and validation set, trained on datasets of varying size.

Finally, we experiment with an observable which is specified in matrix form in the computational basis by $\mathrm{diag}(-1, 0, 1, 2, ..., 2^n - 2)$, where n is the number of qubits in the circuit. The results are depicted in Figure 12.

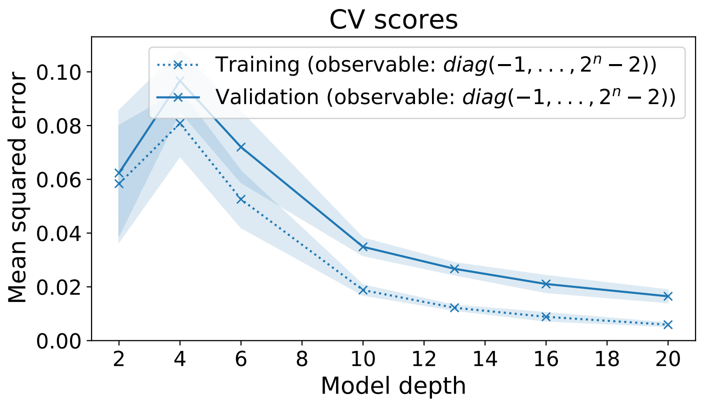


Figure 12. Cross-validated performance of models without data reuploading using the $\mathrm{diag}(-1, 0, 1, 2, ..., 2^n - 2)$ observable, on both the training and validation set.

Similarly to the previous two experiments, we observe a constant gap between the training and validation error, with both errors decreasing as the model depth is increased.

5.3 Regularization

In the previous experiments (Figure 9) we observed a large amount of overfitting when training models using data reuploading on our generated datasets. Because the amount of overfitting in these models increased as deeper circuits were used, these results provide an implicit regularization method in which the depth is limited to reduce the amount of overfitting. Using this method, selecting a circuit with a depth of 2 will result in the best validation error.

The use of lower-depth models is therefore a useful regularization method but does not provide ways to improve the validation error obtained by deeper models. In the final experiments we therefore investigate several regularization methods to reduce the amount of overfitting in these deeper models.

5.3.1 Loss function penalties

In the first of these experiments we apply L1 and L2-regularization and observe the effect on the validation error. The results from applying L1-regularization can be found in Figure 13.

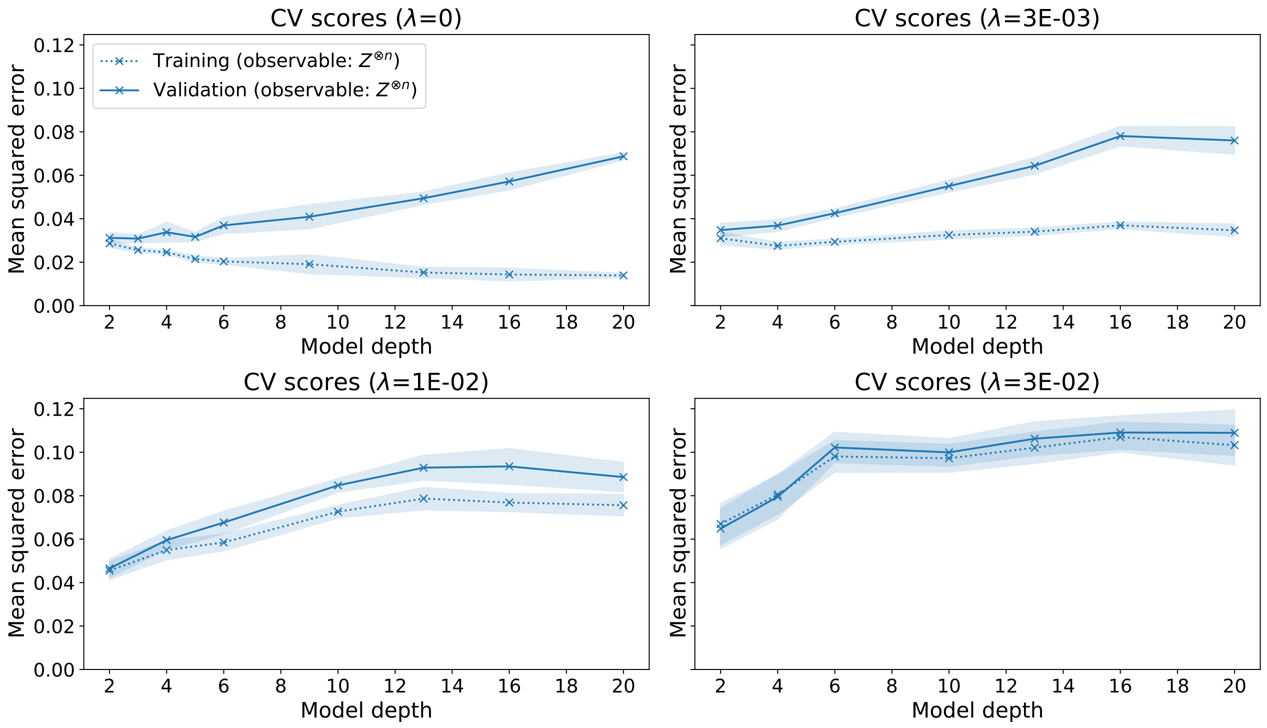


Figure 13. Cross-validated performance of models using data reuploading with various degrees of L1-regularization applied, controlled with the hyperparameter λ.

We observe that as λ increases, the gap at higher model depths between the training and validation scores decreases, so the amount of overfitting is successfully reduced. Unfortunately the validation error increases as λ increases, meaning that the reduction in overfitting does not have the desired result of decreasing the validation error.

The results from applying L2-regularization can be found in Figure 14.

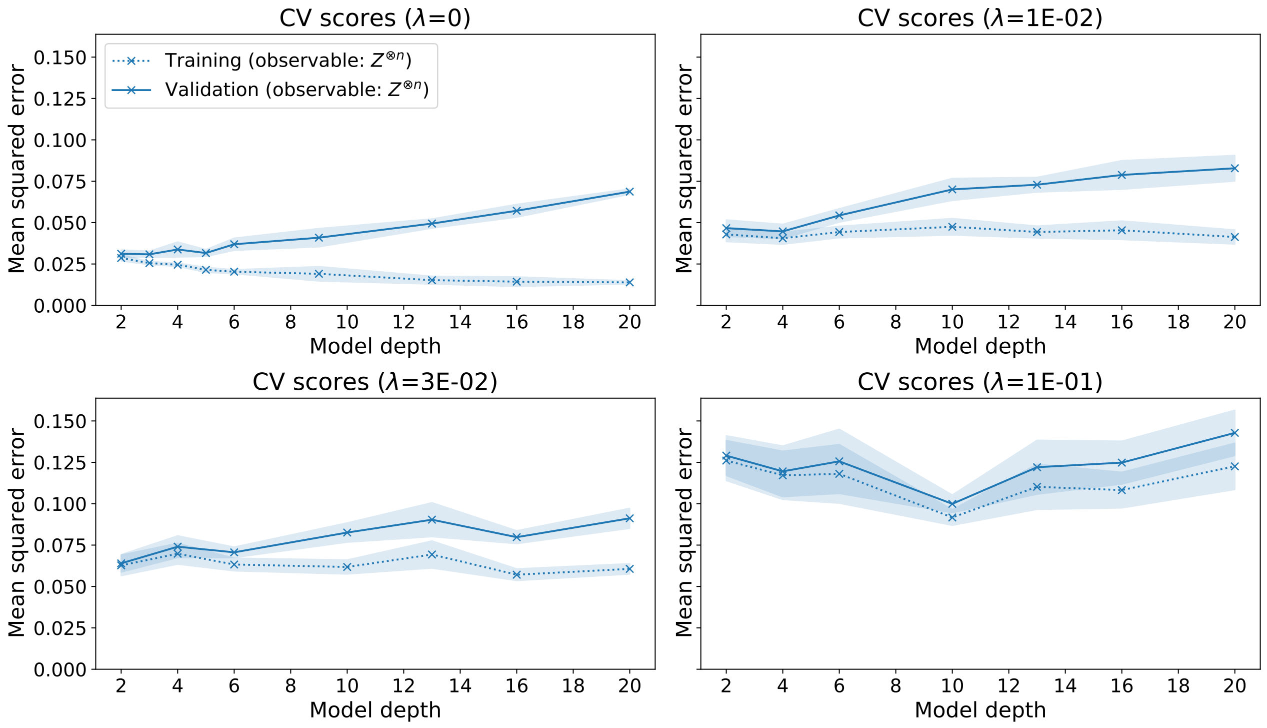


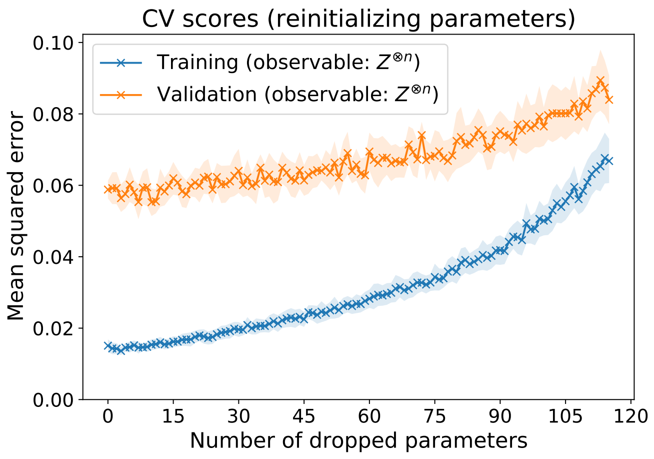
Figure 14. Cross-validated performance of models using data reuploading with various degrees of L2-regularization applied, controlled with the hyperparameter λ.

These results are quite similar to the earlier results using L1-regularization. The degree of overfitting at higher model depths is reduced as λ is increased, but the validation error is also increased. Therefore L1 and L2-regularization do not produce the desired results of improving the generalization performance.

5.3.2 Parameter dropout

In our next experiment we apply parameter dropout, where the model is fully trained multiple times and after each iteration the smallest parameter is fixed to zero. We observe the training and validation performance as more parameters are fixed to zero. Figure 15 illustrates the results of the experiment.

In this experiment similar results can be observed as in the experiment with L1 and L2-



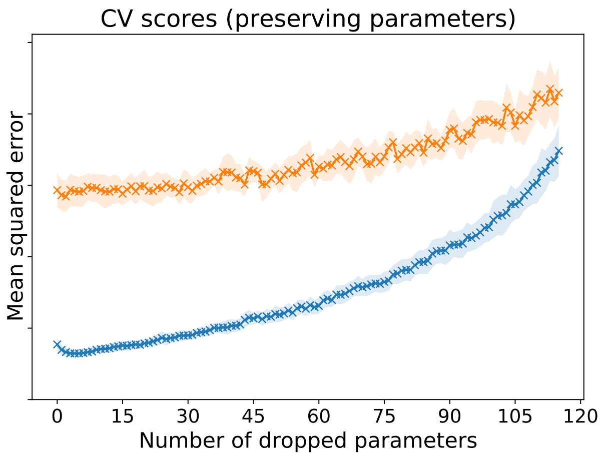


Figure 15. Cross-validated performance of models trained repeatedly, fixing the parameter corresponding to the smallest rotation to zero at each full training iteration. In the left figure, non-fixed parameters are randomly reinitialized after each full training iteration. In the right figure, non-fixed parameters are preserved in between iterations.

regularization. Although the overall gap between validation and training performance is reduced as more parameters are fixed to zero, the validation error increases as well, regardless of whether or not the remaining parameters are preserved in between training iterations.

5.3.3 Layer-cancelling

In our final method to attempt to regularize our model, we change the structure of our PQCs so that the layers in the circuit cancel out when the parameters in the layer are set to zero. This allows the PQC to perform the exact same operations as the lower-depth PQCs used to generate the datasets. We experiment with layer-cancelling PQCs in combination with several degrees of L1-regularization. The results can be found in Figure 16.

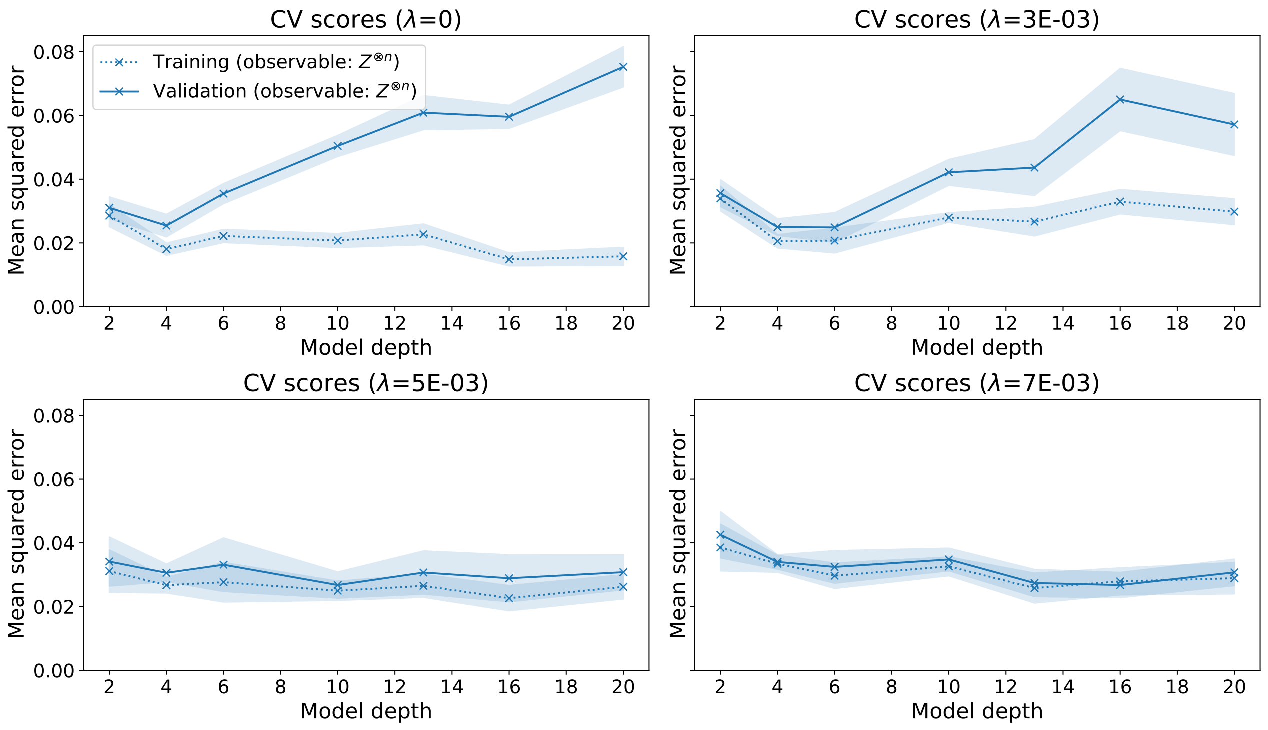


Figure 16. Cross-validated performance of layer-cancelling PQCs, with various degrees of L1-regularization applied, controlled by the hyperparameter λ.

These results show that applying L1 regularization to this modified model significantly improves the validation error of higher-depth models. Because of this, the higher-depth models are able to obtain similar validation performance as the lower-depth models. The higher-depth models do not provide a significant improvement in performance over lower-depth models however.

6 Discussion

In our first experiment we saw that parametrized quantum circuits with a depth as low as three layers can compete with many classical regression methods when applied to the Combined Cycle Power Plant dataset. There is a small side note to this result. As described in Section 5.1, all features and targets in the dataset are rescaled to the interval $[-1, 1]$, which means data from both the training and test set is used to determine the scaling factors. A more valid future approach would determine the scaling factors from the training set only, rescale the dataset using these scaling factors and finally clamp any outliers to the range $[-1, 1]$.

In our second experiment we observed overfitting of higher-depth parametrized quantum circuits on artificially generated datasets. This overfitting only occurred when data reuploading was used within the circuits. This experimental result can be related to the theoretical work by Schuld et al. [16], where the authors showed that repeated use of encoding layers expands the frequency spectrum of the functions the models can represent and therefore increases the expressivity of the models. Because our datasets were generated using lower-depth models with a smaller frequency spectrum, the use of higher-depth model means that frequencies outside of the generator models spectrum can be used by the trained model to improve the training performance while reducing the generalization performance.

This reason would explain why the first regularization experiments did not produce the desired results. Both the L1 and L2-regularization as well as the parameter dropout procedure were purely focused on restricting the parametrized gates in the circuits. These methods restrict how the accessible frequencies can be used by the model, but they do not restrict which frequencies can be used by the model. This is a possible reason why these methods do not sufficiently reduce the expressivity of the model. More experiments are needed to demonstrate this however.

In our final experiment with layer-cancelling parametrized quantum circuits, we observed a large reduction in overfitting when L1 regularization was applied. Although the higher-depth models did not gain any advantage over lower-depth models, this result does suggest that reducing the effectiveness of the encoding layers can reduce the amount of overfitting taking place. Therefore future research on regularization of PQCs applied to other datasets should investigate reducing the expressivity of the circuit not only through regularizing the parametrized layers, but also through reducing the effectiveness of the encoding layers. Fine-tuning the layer-cancelling PQCs could be one way to further improve the generalization performance of deeper circuits. Another interesting follow-up experiment would be to add trainable weight parameters to the encoding gates, so that the angle of each rotation performed by an encoding gate is scaled by the gates corresponding weight. This modification itself increases the expressivity of the model as more parameters are introduced. On the other hand, it creates a more direct opportunity to reduce the effectiveness of the encoding layers by applying L1 and/or L2 regularization to these scaling weights.

In conclusion, the experiments performed in this work show that parametrized quantum circuits can be successfully used in models applied to real-world multivariate regression problems. Furthermore, we have seen that the increase in model expressivity from the use of data reuploading can have a negative impact on the generalization performance of the model. We have proposed parametrized quantum circuits which can successfully be regularized. Finally, our experiments with regularization point the direction of future research towards restriction of the encoding layers in the circuits.

References

[1] J. Shalf, "The future of computing beyond Moore's Law," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2166, p. 20190061, 2020.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[3] J. P. Buhler, H. W. Lenstra, and C. Pomerance, "Factoring integers with the number field sieve," in *The development of the number field sieve* (A. K. Lenstra and H. W. Lenstra, eds.), (Berlin, Heidelberg), pp. 50–94, Springer Berlin Heidelberg, 1993.

[4] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, aug 2018.

[5] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, pp. 209–212, mar 2019.

[6] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information.* Cambridge University Press, 2012.

[7] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *Physical Review A*, vol. 98, no. 3, pp. 1–3, 2018.

[8] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre, "Data re-uploading for a universal quantum classifier," *Quantum*, vol. 4, 2020.

[9] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, "Evaluating analytic gradients on quantum hardware," *Physical Review A*, vol. 99, no. 3, pp. 1–8, 2019.

[10] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, pp. 1–20, 2016.

[11] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, and M. Head-Gordon, "Chemistry: Simulated quantum computation of molecular energies," *Science*, vol. 309, no. 5741, pp. 1704–1707, 2005.

[12] A. Peruzzo, J. McClean, P. Shadbolt, M. H. Yung, X. Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, no. May, 2014.

[13] C.-C. Chen, M. Watabe, K. Shiba, M. Sogabe, K. Sakamoto, and T. Sogabe, "On the Expressibility and Overfitting of Quantum Circuit Learning," *ACM Transactions on Quantum Computing*, vol. 2, no. 2, pp. 1–24, 2021.

[14] J.-E. Park, B. Quanz, S. Wood, H. Higgins, and R. Harishankar, “Practical application improvement to Quantum SVM: theory to practice,” 2020.

[15] C. Gyurik, D. van Vreumingen, and V. Dunjko, “Structural risk minimization for quantum linear classifiers,” 2021.

[16] M. Schuld, R. Sweke, and J. J. Meyer, “Effect of data encoding on the expressive power of variational quantum-machine-learning models,” *Physical Review A*, vol. 103, no. 3, pp. 1–16, 2021.

[17] M. Kobayashi, K. Nakaji, and N. Yamamoto, “Overfitting in quantum machine learning and entangling dropout,” pp. 1–6, 2022.

[18] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa, E. Peters, O. Lockwood, A. Skolik, S. Jerbi, V. Dunjko, M. Leib, M. Streif, D. Von Dollen, H. Chen, S. Cao, R. Wiersema, H.-Y. Huang, J. R. McClean, R. Babbush, S. Boixo, D. Bacon, A. K. Ho, H. Neven, and M. Mohseni, “TensorFlow Quantum: A Software Framework for Quantum Machine Learning,” 2020.

[19] Cirq Developers, “Cirq,” aug 2021.

[20] T. Jones and J. Gacon, “Efficient calculation of gradients in classical simulations of variational quantum algorithms,” 2020.

[21] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[22] P. Tüfekci, “Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods,” *International Journal of Electrical Power and Energy Systems*, vol. 60, pp. 126–140, 2014.

[23] I. Lorencin, N. Anđelić, V. Mrzljak, and Z. Car, “Genetic algorithm approach to design of multi-layer perceptron for combined cycle power plant electrical power output estimation,” *Energies*, vol. 12, no. 22, 2019.