Quantum Neural Networks

Boye Gravningen Sjo

Autumn 2022

TMA4500

Industrial Mathematics, Specialisation Project

Department of Mathematical Sciences

Faculty of Information Technology and Electrical Engineering

Norwegian University of Science and Technology

Preface

This report concludes the course *TMA4500 Industrial mathematics, specialisation project* whose purpose is two-fold. First, it is to acquaint the author-student to a specific topic, which is this case is quantum machine learning or more generally quantum computing. Secondly, it is to make the student familiar with the process of writing a semester-long thesis under the supervision of an expert, and so make them ready for the master's thesis next semester.

The initial aim of the project was to learn about variational quantum algorithms and the application thereof to machine learning. This lead to the investigation of quantum neural networks, which are mostly just interpretation of variational quantum algorithms through the lens of standard, classical neural network design. With the limited scope of this pre-master thesis, it was decided only to implement and test some simple models based on recent research papers.

I would like to thank my de-facto supervisor Franz Fuchs for not only taking time off his important work to supervise me, but also for showing enthusiasm and inviting me into this brave new world of quantum computing when I did not really know what I wanted to do. I also want to thank my co-supervisor, Alexander Stasik, for his help and guidance in the process of writing this thesis. Finally, I would like to thank my official supervisor, Gunnar Taraldsen, for allowing me to work with something I find truly exciting and for still providing feedback while I worked with something out of the ordinary.

All code used in this report will be made available on GitHub via `https://github.com/boyesjo/tma4500`.

Contents

Chapter 1

Machine learning

Machine learning lies at the intersection of statistics, computer science and optimisation. The central idea is to design an algorithm that uses data to solve a problem, and in so avoid explicitly programming a solution. With ever more data available and with ever more powerful computers, machine learning has become a powerful tool in many fields, solving problems previously thought intractable. Such algorithms or models can be used for a plethora of tasks, which are mainly divided into three main categories:

- *Supervised learning:* Given data with corresponding labels, find the relationship and try to assign correct labels to new, unseen data.
- *Unsupervised learning:* Given data, find some underlying structure, patterns, properties or relationships, such as clusters or outliers.
- *Reinforcement learning:* Given a set of rules or environment, such as a game, try different strategies and determine one that optimises some reward.

Only the first thereof will be explicitly considered in this thesis, though much of the theory and results can be extended to the latter two. This chapter is hardly a comprehensive introduction to machine learning, but rather a brief overview of the most important concepts and techniques, serving as a reference point for the later endeavours into quantum machine learning. Some familiarity with basic probability theory is assumed.

1.1 Supervised learning

For supervised learning, some data is assumed given. Mathematically, this data is described by a set of input-output pairs $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathcal{X}$ is a vector of features in the input domain $\mathcal{X}$ and $y_i \in \mathcal{Y}$ is the corresponding label in the output domain $\mathcal{Y}$. The data is assumed to be drawn from a joint distribution $p(\boldsymbol{x}, y)$. Given a new, unseen input $\boldsymbol{x}$, the goal predict its corresponding label y.

The input domain $\mathcal{X}$ is usually assumed to be the vector space $\mathbb{R}^N$ for some integer N, though some dimensions may be discrete and even binary. Some preprocessing is often

done to the data, such rescaling the features or using the data to calculate new features. While preprocessing can be essential to the performance of a machine learning model, it does not matter for this theoretical discussion; any preprocessed data can be assumed to be drawn from the distribution of the transformed variable.

The output domain $\mathcal{Y}$ is assumed to be single-dimensional, as a multidimensional prediction can be decomposed into multiple one-dimensional prediction problems. It may be finite, in which case the problem is called classification, or continuous, in which case it is called regression. Regression methods may easily be applied to classification problems trough simple thresholds.

1.1.1 Models

Deterministic models

In machine learning, models are typically deterministic and thus can be seen as functions $f : \mathcal{X} \to \mathcal{Y}$ that map inputs to outputs,

$$\hat{y} = f(\boldsymbol{x}), \tag{1.1}$$

where $\hat{y}$ is the predicted output.

A model belongs to some family, a set of models with similar properties, where families often are parametric. Parametric models are defined by a finite set of parameters $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_k\}$, where each parameter θ_i is a real number.

Probabilistic models

With probabilistic models, the model instead attempts to reconstruct the conditional probability distribution $p(y|\boldsymbol{x})$. In statistics, this distribution will often be of a known type which is parametrised by a function of the data and some parameters. For instance, in linear regression, the distribution is assumed to be Gaussian with mean $\boldsymbol{\theta}^\top \boldsymbol{x}$ and variance given by another parameter σ^2.

Alternatively, the conditional distributions may be thought of as being defined by

$$p(y|\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon, \tag{1.2}$$

where ε is a random variable and f a deterministic function. The ε can be seen as some inherent randomness or noise in the data, or as a result of lacking information. It is often assumed to be independent of the input.

Probabilistic models can be converted to deterministic models using a method to extract a single output from the distribution, such as the maximum a posteriori,

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} p(y|\boldsymbol{x}), \tag{1.3}$$

or the mean,

$$\hat{y} = \int_{\mathcal{Y}} y\, p(y|\boldsymbol{x}) \mathrm{d}y. \tag{1.4}$$

Probabilistic has the benefit of being able to provide a measure of uncertainty in the prediction, which is often useful. With simpler settings, statistical arguments can be made about the relationship between the input and output, which can be used to make inferences about the data. However, the simpler design of deterministic models makes them more suitable for computationally demanding tasks, such as deep learning.

1.1.2 Measuring and optimising performance

A loss function $L(y, \hat{y}) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ is used to measure the performance of a model by comparing a predicted output $\hat{y}$ to the true output y, which then is used to optimise the model. By convention, the loss should be positive, and lower values indicate better predictions, zero indicating exact agreement. Its main purpose is to optimise the model, so it need not be the most useful metric of performance. It is usually chosen to be differentiable, as this allows for the use of gradient-based optimisation. Therefore, something like $I(\hat{y} = y)$ is not used as a loss (though often as a performance metric). Instead, losses like the square loss,

$$L(y, \hat{y}) = (y - \hat{y})^2, \tag{1.5}$$

often is used for regression problems and the cross-entropy,

$$L(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}), \tag{1.6}$$

for classification problems. If the model is probabilistic and assumed to be a particular type of distribution, its structure can be used to define a loss. Maximum likelihood estimation can be used, switching only the sign to conform with machine learning conventions, leading to the negative log-likelihood loss.

With some loss, one can consider the risk, defined as the expected loss over the joint distribution of the data,

$$R(f) = \mathrm{E}(L(y, f(\boldsymbol{x}))) = \iint_{\mathcal{X} \times \mathcal{Y}} L(y, f(\boldsymbol{x}))\, p(\boldsymbol{x}, y)\, \mathrm{d}\boldsymbol{x}\, \mathrm{d}y, \tag{1.7}$$

and wish to minimise it. Of course, this is only possible if the joint distribution $p(\boldsymbol{x}, y)$ is known, so it is in practice replaced by the empirical risk, assuming independent and equally likely samples,

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i)). \tag{1.8}$$

This can be assumed to converge to the true risk as $n \to \infty$ by the law of large numbers. More data is certainly better, but it may be expensive to collect, and the deviance between the empirical and true risk may not decrease monotonically with n. It is therefore desirable to find models and training methods that minimise the risk with as few samples as possible.

For instance, square loss leads to the mean squared error (MSE),

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x}_i))^2. \tag{1.9}$$

These empirical risks that depend on the whole data set defines cost functions that are used to train the model.

For a chosen parametric model family, the supervised learning problem can be formulated as an optimisation problem to find the best parameters $\boldsymbol{\theta}^*$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}; \boldsymbol{\mathcal{D}}), \tag{1.10}$$

where C is the cost function. Since the loss function and models are often designed to be differentiable, giving differentiable costs, this can be solved using gradient descent methods. In the simplest case, that means that the parameters are updated until some convergence criterion is met by

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}^{(t)}, \mathcal{D}), \tag{1.11}$$

where $\eta > 0$ is a learning rate and $\boldsymbol{\theta}^{(t)}$ the parameters at iteration t. In practice, more sophisticated gradient methods are often used, such as the Adam optimiser [2].

The process of optimising the cost function is called training. As the model improves, it is said to learn.

1.1.3 Generalisation: the bias-variance trade-off

The goal of supervised learning is to find a model that performs well on unseen data. Optimising the cost function on the training data will lead to a model that performs well on the training data, but may not generalise well to unseen data. Therefore, the model is evaluated on a test set, which is not used for training.

There is a constant struggle between having models with lots of parameters and great expressive power versus simpler models with fewer parameters. The former are more likely to overfit the data, while the latter are more likely to underfit the data. This is known as the bias-variance trade-off. The main goal is of course to generalise, that is to have a model that truly captures the underlying properties of the data and subsequently performs well on data that it has not seen before.

There may be mismatch between the true risk in eq. (1.7) and the empirical risk in eq. (1.8). Minimising the empirical risk may not be the same as minimising the true risk. Intrinsically, with a probabilistic model, there is some uncertainty or noise that is inherent to the data. Consequently, one must choose a model that is flexible enough to capture the underlying structure of the data, but not so flexible that it captures the noise. When a model is too simple to capture the underlying structure, it is said to have high bias or be underfitted, while a model complex enough to capture the noise is said to have high variance or be overfitted. An overfitted model will have small or no errors on the data used for training, but may be wildly inaccurate on new data. This is captured in fig. 1.1.

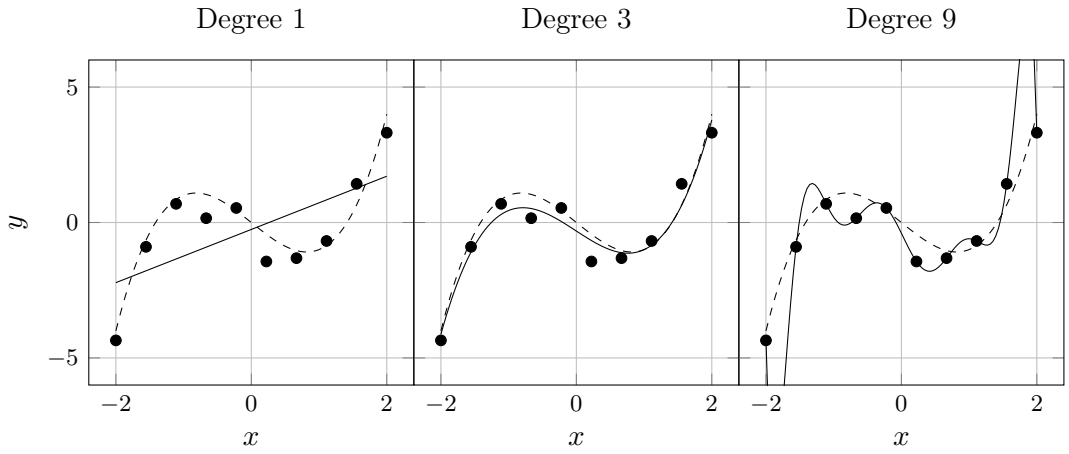

Figure 1.1: A simple example of overfitting and underfitting. Ten data points were generated by $x^3 - 2x$ plus some Gaussian noise with 0.4 standard deviation, shown in the figure as dots. The solid lines denote models fitted using squared loss, being respectively polynomials of degree 1, 3 and 9, while the dashed line is the true function. The model with degree 1 is underfitted, while the model with degree 9 is overfitted — it perfectly fits all samples, but greatly deviates from the true function elsewhere. However, the 'correct' cubic polynomial lies much closer to the true function.

1.2 Neural networks

Modern machine learning owes much of its popularity to the success of artificial neural networks, or if the context is clear, just neural networks (NNs). With easier access to larger datasets, more powerful hardware (in particular GPUs or even dedicated TPUs) and the backpropagation algorithm, NNs have become able to solve problems far too complicated for traditional methods.

Though state-of-the neural networks can contain billions of parameters, training them remains feasible. Modern hardware is of course paramount, but also backpropagation is crucial. Neural networks are trained using gradient methods, and with backpropagation, the gradient can be computed efficiently.

How such large models avoid overfitting is not entirely clear. Seemingly contradicting the bias-variance trade-off, the double descent phenomenon appears as model complexity increases or as more gradient descent iterations are done. This is a phenomenon where, after some point, the variance no longer increase with model complexity. Instead, it decreases and converges toward some value. This can be seen in fig. 1.2, where convolutional networks[1] were tested with different layer sizes. Unlike statistical methods, once past this complexity threshold, there is little reason not to increase the performance by expanding the model other than the increased computational cost. Double descent have been shown to appear for a variety of models [3].

[1] See section 1.2.1.

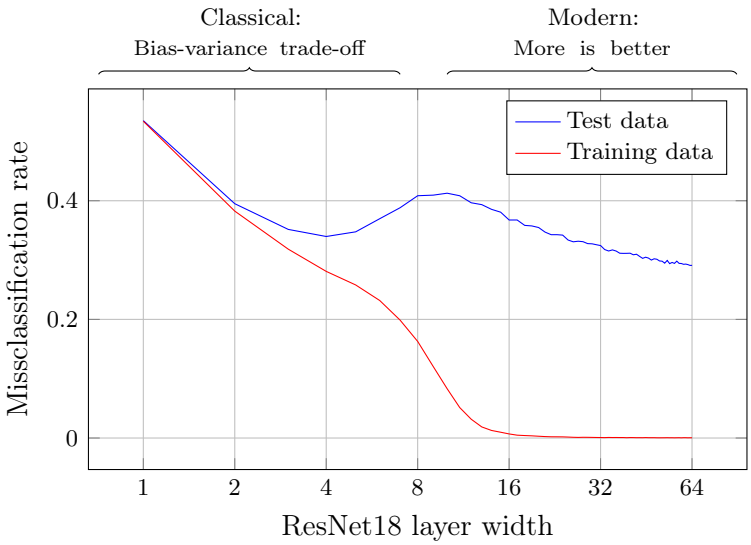

Figure 1.2: Double descent phenomenon. As model complexity increases, the train error decreases steadily, but the test error reaches a minimum and after which the test error starts to increase again. However, past a certain point, the test error decreases again. This defines two regimes: the first classical regime, where the bias-variance trade-off applies, and model complexity must remain low to avoid overfitting, compared to the modern ML regime, where more complexity is always better. The data is from [3], where ResNet18 models were used for image classification with added label noise.

With the great size and complexity, interpretability is sacrificed. The models are deterministic and often black boxes; it is difficult to understand why they make the predictions they do. Luckily, there have been some developments, perhaps most notably the universal approximation theorem. It states that a neural network with a single hidden layer can approximate any continuous function to arbitrary precision, given enough neurons[2]. This gives some credence to the idea that NNs of other structures could be used to approximate intricate functions.

1.2.1 Architectures

Dense feed-forward neural networks

The basic neural network is a dense feed-forward neural network, with a typical structure shown in fig. 1.3. There can be more or fewer nodes in each layer, and as many or few hidden layers as desired.

In the input layer, the data is fed in with each input node corresponding to a feature. Then, in the hidden layers, the data is transformed by a series of linear transformations

[2]In addition to some easily met requirements regarding the activation function.

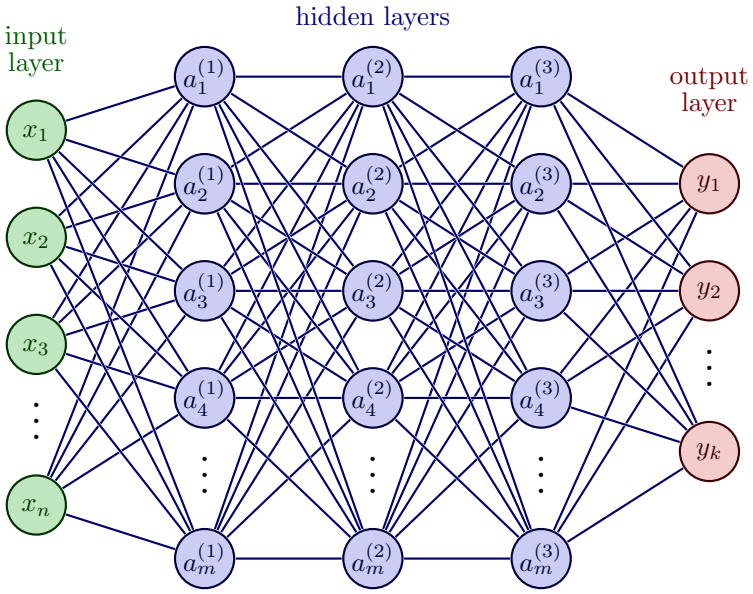

Figure 1.3: Typical structure of a dense feed-forward neural network. Here it has three hidden layers of constant size m, but the number of layers and the size of each layer can be chosen arbitrarily. Being dense means that each node in one layer is connected to all nodes in the next layer, while being feed-forward means that the connections are unidirectional. From [4].

and non-linear activation functions. These can be expressed as

$$a_i^{(j)} = \sigma\left(b_i^{(j)} + \sum_{n=1}^{m} w_{i,n}^{(j)} a_n^{(j-1)}\right), \tag{1.12}$$

where j is the layer number, i is the node number and w and b are the weights and biases of the network — parameters to be optimised.

The activation function σ is a non-linear function, such as the sigmoid function, the hyperbolic tangent or the rectified linear unit (ReLU). Non-linearity is needed for the network not to collapse to one great linear transformation. Some commonly used activation functions are listed in table 1.1.

The output layer is similar to the hidden layers, though perhaps with different activation functions and fewer nodes. For instance, if the goal is to classify the data into k classes, the output layer could have k nodes with an activation function that ensures the sum of the outputs is one. In that way, the output can be interpreted as probabilities of the data belonging to the respective classes.

Models being dense mean that each node in one layer depends on all nodes in the previous layer. It is feed-forward, because the data flows in one direction; the perceptrons in a layer depends only on those in the former.

Table 1.1: Common activation functions. Usually, they are applied element-wise to the output of a linear transformation. However, in the case of the softmax function, it depends on the whole layer.

Name	**Definition**	**Typical use case**
Identity	$\sigma(x_i) = x_i$	Regression output
Sigmoid	$\sigma(x_i) = 1/(1 + e^{-x_i})$	Hidden layers, binary classification output
Hyperbolic tangent	$\sigma(x_i) = \tanh(x_i)$	Hidden layers, binary classification output
ReLU	$\sigma(x_i) = \begin{cases} x_i, & x \geq 0 \\ 0 & x < 0 \end{cases}$	Hidden layers
Leaky ReLU	$\sigma(x_i) = \begin{cases} x_i, & x \geq 0 \\ 0.01x_i, & x < 0 \end{cases}$	Hidden layers
Softmax	$\sigma(x_i) = e_i^x / \sum_{j=1}^{k} e^{x_j}$	Multi-class classification output

Convolutional neural networks

Convolutional neural networks (CNNs) are a special type of neural network that are particularly suited for certain tasks like image processing. A greatly simplified CNN is shown in fig. 1.4.

The basic component of the CNN is the convolutional layer. In it, a kernel or filter is applied to the input data, which often is as simple as a 3×3 matrix. It is applied to only parts of the input, and thus only extracts local properties. Typically, pooling layers complement the convolutions by reducing the dimensionality through some simple non-parametrised operation, such as taking the maximum value in a 2×2 matrix. CNNs generally finish with one or more dense layers, which then operate on a significantly reduced number of features. The reduction of dimensionality is important because it reduces the number of parameters in the model and the risk of overfitting. Furthermore, the convolutional approach forces the model to learn local features. This is very beneficial for tasks such as image classification, where local features, such as edges, are more important than global ones, such as the position of the subject.

Generative adversarial networks

Generative adversarial networks (GANs) refer to special kind of design where two different networks are trained by competing against each other. With an unlabelled dataset, the goal is to generate data that is indistinguishable from the real data. The first model, the generator, attempts to generate samples from the underlying distribution. On the other hand, the discriminator is tasked with distinguishing between data produced by the

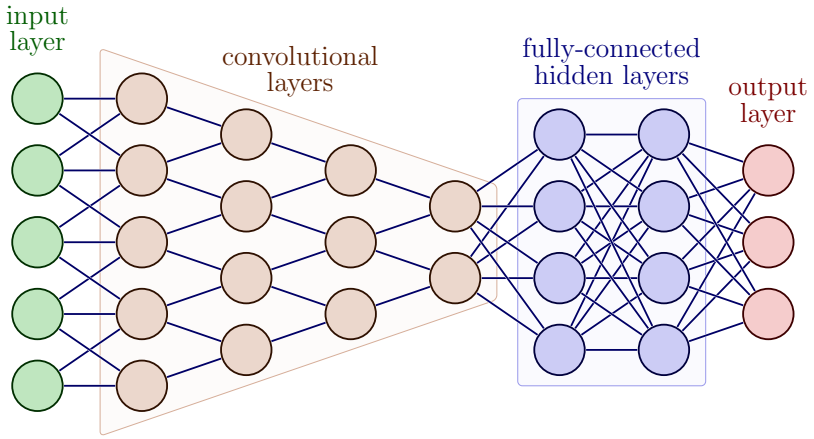

Figure 1.4: The basic structure of a convolutional neural network. Data is input before conventional layers are applied, in which perceptrons are only connected to small regions of the previous layer. After sufficient dimensionality reduction, regular dense layers can be used. From [4].

generator and the real samples from the data set. Accordingly, the discrimination is a supervised problem. Both models are trained simultaneously by first sampling from the generator and the data set, using supervised methods to update the discriminator, and then using the discriminators predictions to update the generator.

GANs are mainly used for unsupervised learning, and they have seen great success in generating random images. They have also seen success in more abstract tasks, such as translating text prompts to images or predicting what will happen next in a video.

References

[1] Ville Bergholm et al. *PennyLane: Automatic Differentiation of Hybrid Quantum-Classical Computations*. 29th July 2022. DOI: 10.48550/arXiv.1811.04968.

[2] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 29th Jan. 2017. DOI: 10.48550/arXiv.1412.6980.

[3] Preetum Nakkiran et al. 'Deep Double Descent: Where Bigger Models and More Data Hurt*'. In: *Journal of Statistical Mechanics: Theory and Experiment* 2021.12 (1st Dec. 2021), p. 124003. ISSN: 1742-5468. DOI: 10.1088/1742-5468/ac3a74.

[4] Izaak Neutelings. *Neural networks*. Licensed under a Creative Commons Attribution-ShareAlike 4.0 International License with © Copyright 2021 – TikZ.net. Sept. 2021. URL: https://tikz.net/neural_networks/ (visited on 24/10/2022).

[5] Adam Paszke et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html (visited on 13/11/2022).

[6] Matthew Treinish et al. *Qiskit/Qiskit: Qiskit 0.39.2*. Version 0.39.2. Zenodo, 4th Nov. 2022. DOI: 10.5281/ZENODO.2573505.