
Rocky Raccoon
Pennylane and the Quantum Log-Likelihood
Research by Roeland Wiersema
Front page by Pauline Wiersema
Pennylane

PennyLane and the Quantum Log-Likelihood

Roeland Wiersema

Radboud University
Nijmegen, The Netherlands

September 13, 2019

Note by the Author

When working on my thesis last this past year, I experimented a bit with quantum circuits by following Peter Wittek's online quantum computing course [1]. I got excited about the possibilities of integrating quantum circuits into my research, but graduating (as opposed to adding more stuff to my thesis) did not sound so bad either. I heard about Xanadu's Quantum Software Competition and figured that it would be a fun summer project, which is why I decided to write this paper. The work presented here is a combination of my master's thesis [2], a Physical Review A paper [3] and a couple of weeks of messing around with PennyLane. The code can be found on GitHub together with some documentation. I want to thank my sister for making the cover page artwork[1], and my supervisor professor Bert Kappen for his mentorship during this research.

[1] `http://paulinewiersema.com/`

Contents

1 Introduction **5**

2 Background **5**
2.1 Logistic Regression and the Perceptron 6
2.2 Quantum Log-Likelihood 6
2.3 Quantum Tomography 8
2.4 Multiple Classes 9

3 PennyLane Implementation **10**
3.1 Data Circuit 11
3.2 Model 1: Amplitude Encoding a Linear Predictor 12
3.3 Model 2: Feature Amplitude Encoding with Strongly Entangling Layer 13
3.4 Model 3: Strongly Entangling Layer With Neural Network Input 14

4 Results **15**
4.0.1 `XOR` 15
4.0.2 Multi-Class Example 15
4.0.3 Noisy Data 18

5 Conclusion **18**

1 Introduction

Inspired by the success of deep learning [4], there has been interest to develop quantum equivalents of neural networks that can be trained more efficiently or are more expressive than their classical counterparts [5–12]. Quantum inspired proposals utilize quantum effects in different ways: employing a superposition of perceptrons [7], using qubit weights [8, 10] or learning a unitary transformation between input and output [9]. Quantum computing work in this direction involves using an inverse quantum Fourier transform to obtain a nonlinear step function [11] or tracing out parts of a quantum circuit to create an autoencoder [12]. However, all these proposals introduce a classical cost function for learning, omitting the underlying probabilistic motivation for their model. The usage of quantum probabilistic cost functions is still relatively unexplored.

Constructing quantum probabilistic models from density matrices is a new direction of quantum machine learning research [13, 14], where one exploits quantum effects in both the model and training procedure by constructing a differentiable cost function in terms of density matrices. Density matrices are used in quantum mechanics to describe statistical ensembles of quantum states, and are represented by a positive semi-definite Hermitian matrix with trace 1. In [2, 3], we have shown that one can use density matrices to construct a model that minimizes a generalization of the classical likelihood function for learning, replacing the classical perceptron bit with a qubit. Also, we have shown that this idea can be extended to multiple classes. However, we have taken a lot of freedom with our description, not taking hardware constraints into account.

Researchers affiliated with Xanadu, a Canadian photonic quantum computing company, have developed a Python framework that enables automatic differentiation through quantum circuit, called PennyLane [15]. This framework can be used to interface with some of the proto-quantum computers available right now, and offers access to several quantum simulation backends. In this paper we will demonstrate how this framework can be used to minimize the quantum log-likelihood from [3]. In section 2 we will introduce all the relevant concepts that we require for our model. Then, in section 3 we will discuss several examples of our implementation. Finally, we will conclude in section 4, where we give an outlook on future work.

2 Background

We will assume prior knowledge of machine learning and quantum computing, For a detailed review of the former, consider [16]. For the latter, the standard introductory text is [17].

2.1 Logistic Regression and the Perceptron

Machine learning can be defined as a set of methods that can automatically detect patterns in data, and then uses these uncovered patterns to predict future data. Supervised learning is a subset of machine learning where we try to find a mapping $y = f(\mathbf{x})$ for data $\mathcal{D}\{(y, \mathbf{x})\}^N$. If the output y is from some finite set $C \in \{1, ..., p\}$ we talk about classification. Because data is often noisy, we make use of probability theory as an underlying framework for machine learning. The function $f(\mathbf{x})$ is then replaced by a conditional probability distribution $p(y|\mathbf{x}, \mathcal{D})$.

Consider a data set $\mathcal{D}\{(y, \mathbf{x})\}^N$, where $\mathbf{x} \in \mathbb{R}^n$ and $y \in \{-1, 1\}$. A suitable model for this data set would be logistic regression. This model consists of a function $\mu : \mathbb{R}^n \to (0, 1)$ that maps our input to a suitable probability and a likelihood function $\mathcal{L}$ that attributes a cost to the difference between our model output and the true label y:

$$\begin{aligned} p(y|\mathbf{x}) \equiv \mu(\mathbf{x}; \mathbf{w}) &= (1 + \exp(-\mathbf{w} \cdot \mathbf{x}))^{-1} \\ \mathcal{L} &= -\sum_{\mathbf{x}} q(\mathbf{x}) \sum_{y} q(y|\mathbf{x}) \log(p(y|\mathbf{x}))) \end{aligned} \tag{1}$$

where $q(\mathbf{x})$ and $q(y|\mathbf{x})$ are empirical data distributions obtained from $\mathcal{D}$. The function $\mathcal{L}$ is called the negative log-likelihood and μ is called the softmax function. They are derived by assuming that the binary response y is drawn from a binomial distribution. Through gradient descent we can update the parameters of this model by calculating $\nabla_{\mathbf{w}} L$ and updating $\mathbf{w}$ accordingly. Since $\mathcal{L}$ is convex we will always find the global minimum [16].

Logistic regression is closely related to a machine learning model known as a perceptron:

$$p(y|\mathbf{x}) = f(\mathbf{x} \cdot \mathbf{w}) \tag{2}$$

Where $f : \mathbb{R}^n \to (0, 1)$ is a non-linear activation function. The perceptron does not rely on some underlying probabilistic assumption; it simply maps data to the domain of the response variable. The cost function we want to minimize to update the model parameters can be introduced *ad hoc*. Clearly, we can obtain logistic regression by setting $f \to \mu$ and minimizing with respect to $\mathcal{L}$.

2.2 Quantum Log-Likelihood

To extend the classical likelihood in equation 1 to the realm of quantum mechanics we require a description of our model $p(y|\mathbf{x})$ and the conditional probability $q(y|\mathbf{x})$ in terms of density matrices. The density matrix is a trace one, positive semi-definite matrix which describes the classical uncertainty we have about a quantum state.

$$\rho = \sum_i p_i \, |\psi_i\rangle\langle\psi_i|$$

with p_i the classical probability of observing state $|\psi\rangle$ where $\sum_i p_i = 1$. If this matrix is rank one ($p_i = 1$), we have what is known as a pure state. In this case there is no classical uncertainty about what quantum state the system is in. If the density matrix has rank > 1 then we have a so called mixed state. We can link density matrices easily to classical probability distributions. Consider the density matrix

$$\rho = \sum_i p(y = i)\, |i\rangle\langle i|$$

with $|i\rangle$ a computational basis state. This simply describes a classical distribution over the labels y on the diagonal of ρ. In other words, due to the condition that the diagonal must be trace one, a density matrix defines a classical probability over the basis vectors. With this connection in mind we will attempt to formulate a log-likelihood in terms of a density matrices.

We consider the specific case where the data consists of N discrete vectors $\mathbf{x} \in \{1, -1\}^d$ with d bits and $y \in \{1, -1\}$ labels. We define the quantum log-likelihood as a cross entropy between a conditional data density matrix $\eta_{\mathbf{x}}$ and a model conditional density matrix $\rho_{\mathbf{x}}$, analogous to equation 1. To obtain $\eta_{\mathbf{x}}$, we construct a wave function based on the empirical conditional probabilities $q(y|\mathbf{x})$ for each $\mathbf{x}$:

$$|\Psi\rangle = \sqrt{q(1|\mathbf{x})}\, |1\rangle + \sqrt{q(-1|\mathbf{x})}\, |-1\rangle \tag{3}$$

where the states $|1\rangle$, $|-1\rangle$ are the eigenstates of the σ^z operator. The data density matrix is then defined as $\eta_{\mathbf{x}} \equiv |\Psi\rangle\langle\Psi|$, with components

$$\eta_{\mathbf{x}}(y, y') = \sqrt{q(y|\mathbf{x})}\sqrt{q(y'|\mathbf{x})} \tag{4}$$

Note that this is a pure density matrix. The method of encoding a classical probability distribution into the amplitudes of a wave function is also known as QSampling [18]. $q(y|\mathbf{x})$ is an empirical distribution over the label y for each $\mathbf{x}$, and is fully determined by its conditional expectation value of y given $\mathbf{x}$ written as $b(\mathbf{x})$:

$$\begin{aligned} q(y|\mathbf{x}) &= \frac{1}{2}(1 + b(\mathbf{x})y) \\ \text{with } b(\mathbf{x}) &= \frac{1}{M}\left(\sum_{\mathbf{x}'} y' \mathbb{I}(\mathbf{x}' = \mathbf{x})\right) \\ \text{and } M &= \sum_{\mathbf{x}'} \mathbb{I}(\mathbf{x}' = \mathbf{x}) \end{aligned} \tag{5}$$

Succinctly put, every time $\mathbf{x}$ appears in the data, we add its corresponding label y' to the sum. Dividing by M, the total number of times the sample $\mathbf{x}$ appears in the data we obtain the conditional expectation value $b(\mathbf{x})$. We define the

empirical probability

$$q(\mathbf{x}) = \frac{M}{N}$$

for M occurrences of $\mathbf{x}$ and N the total number of samples.

In previous work we considered a mixed state as a trainable model $\rho_{\mathbf{x}}$ [3],

$$\rho_{\mathbf{x}} = \frac{1}{Z} e^{-\beta H} \tag{6}$$

where $H = \sum_k h^k \sigma^k$. By using Euler's formula, this can be rewritten as

$$\rho_{\mathbf{x}} = \frac{1}{2} I + \frac{1}{2} \tanh h \sum_k \frac{h^k \sigma^k}{h} \tag{7}$$

with $h^k = \mathbf{w}^k \cdot \mathbf{x}$ and $\sqrt{\sum_k (h^k)^2}$. Plugging this in equation 1 we obtain

$$\mathcal{L} = -\sum_{\mathbf{x}} q(\mathbf{x}) \operatorname{Tr}\{\eta_{\mathbf{x}} \ln(\rho_{\mathbf{x}})\} \tag{8}$$

where the trace corresponds to the quantum relative entropy of $S(\rho_{\mathbf{x}} || \eta_{\mathbf{x}})$ minus the von Neumann entropy, which is jointly convex [19]. A class probability was obtained by considering the diagonal elements of the density matrix, which define a probability since the density matrix is trace one. We demonstrated that off-diagonal elements in the density matrix of equation 7 can result in a more accurate characterization of noise in certain data sets. Noise in this case means conflicting output labels: multiple instances of some sample $\mathbf{x}$ which occur in the data with different labels y.

With the model in 6, we take a lot of liberty with defining the model $\rho_{\mathbf{x}}$; we simply parametrized the basis of the density matrix. While nice for theoretical applications, a connection with the physical world is lacking. Quantum circuits provide a way to control a quantum system directly, which will allow us to implement both $\eta_{\mathbf{x}}$ and $\rho_{\mathbf{x}}$ using a physical system. How do we go about doing that? To estimate a density matrix from a quantum circuit, we require quantum tomography.

2.3 Quantum Tomography

An important ingredient for implementing the quantum log-likelihood on a quantum computer is the possibility of performing quantum state tomography on a subsystem of the quantum circuit. This process involves performing repeated measurements on a quantum system in order to reconstruct the density

matrix. Single qubit tomography is simple, the density matrix is given by

$$\rho = \frac{1}{2}\sum_{k=0}^{3} h_k \sigma_k$$
$$\mathrm{Tr}\{\sigma_l \rho\} = h_k \delta_{kl}$$

This means that we only require three expectation values to determine the density matrix (More specifically the Stokes parameters [20]).The process of constructing the density matrix through repeated measurements can be extended to multi-qubit systems. In general the density matrix has $4^n - 1$ free parameters, and is of the form

$$\rho = \frac{1}{2^n} \sum_{i_1, i_2, \ldots, i_n = 0}^{3} h_{i_1, i_2, \ldots, i_n} \sigma_{i_1} \otimes \sigma_{i_1} \otimes \ldots \otimes \sigma_{i_n} = \sum_{i}^{4^n - 1} h_i \mathcal{A}_i$$

Where $\mathcal{A}_i$ is a SU(2) product observable. It is clear that for quantum systems with a large number of qubits, the density matrix becomes harder to reconstruct, since the number of required measurements scales exponentially. However, we are interested only in measuring a density matrix of similar size as $\eta_{\mathbf{x}}$. To do this we require only measurements on a single qubit. We can build a quantum circuit that is much larger then this, and measure the output qubit to get a density matrix of the right size. This is formally referred to as "tracing out." Consider a quantum system $H = H_A \otimes H_B$. The partial trace is the linear map $\mathrm{Tr}_B : \mathcal{L}(H_A \otimes H_B) \to \mathcal{L}(H_A)$ where $\mathcal{L}(V)$ is the space of linear operators on the vector space V. This operation is then given by

$$\mathrm{Tr}_B(\rho_{AB}) \equiv \rho_A \, \mathrm{Tr}_B(\rho_B) \tag{9}$$

ρ_A is often referred to as a reduced density matrix. One of the peculiarities of quantum systems is that tracing out a pure state can lead to a mixed state. So by measuring a subsystem, we lose information about the total system. The interest for us is purely practical, preforming measurements gives us a mixed density matrix of the correct size. It also allows us to construct mixed state density matrices, since the partial trace of a pure state can be mixed. This gives us additional free parameters to learn $\eta_{\mathbf{x}}$. We assume from now on that estimating $\rho_{\mathbf{x}}$ can be done without noise, which is an unrealistic assumption for an experimental setup. Qiskit offers noise simulation, but this is out of the scope of this paper.

2.4 Multiple Classes

For binary classification $\rho_{\mathbf{x}}$ corresponds to the density matrix of a single qubit. This means that either our quantum circuit consists only of a single qubit, or that we trace out the rest of the system so that only a single qubit remains. This seems to limit the model to binary classification. However, we can construct a

quantum analogue of a multinomial regression model, a method that generalizes logistic regression to multiple classes:

$$p(y = c|\mathbf{x}; (\mathbf{w}_1, \ldots, \mathbf{w}_C)) \equiv= \mu_c^{\mu}(\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x})}{\sum_{c'}^{C} \exp(\mathbf{w}_{c'} \cdot \mathbf{x})} \tag{10}$$

For the quantum analogue we considered a similar construction in [3]:

$$\rho_{\mathbf{x}} = \frac{\exp(H)}{\mathrm{Tr}\{\exp(H)\}}$$

In the single qubit case, this can be rewritten to a nice algebraic expression, but for higher order symmetries this becomes cumbersome. In [2] we consider the case where $\rho_{\mathbf{x}}$ is a 3×3. This means that $H = \sum h_i \lambda_i$ with λ^i the generators of SU(3). These generators span the full space of 3×3 density matrices. For a quantum computer[2] however, we are limited to observables of the form $\mathrm{SU}(2) \otimes \mathrm{SU}(2) \otimes \ldots \otimes \mathrm{SU}(2)$. So the shape of the density matrix is always $2^n \times 2^n$ where n is the number of qubits. In the case of multiple classes this means that we require at least $n = \mathrm{ceil}(\log_2(N_c))$ qubits where N_c is the number of classes. Instead of single output qubit for the binary case, we thus require multiple qubits to get a model density matrix of the correct size. For three classes this implies that that for the data density matrix and model density matrix we have to set $q(y = 4|\mathbf{x}) = 0$ and $p(y = 4|\mathbf{x}) = 0$, respectively.

At this point, we have discussed all the necessary ingredients for training a model with the quantum log-likelihood. All we need now is a quantum computer. Unfortunately, quantum computing hardware is still in its infancy, but there exist a number of frameworks that allow one to simulate quantum circuits.

3 PennyLane Implementation

Besides being a song by the Beatles, PennyLane [15] is a Python framework for automatic differentiation and optimization of hybrid quantum-classical computations. This makes it the ideal tool to implement the quantum log-likelihood. PennyLane provides access multiple quantum computing frameworks such as StrawberryFields [21], Rigetti Forest [22], Qiskit [23], and ProjectQ [24]. Additionally, PennyLane provides an interface with TensorFlow [25] and PyTorch [26]. For our purposes, we chose to work with the PennyLane-TensorFlow interface. Experimenting with the Qiskit backend for noise simulation led to some buggy behaviour, so we refrained from using this.

Inspired by the methodology[3] outlined in [27], we propose three hybrid quantum machine learning models that make use of the quantum log-likelihood of

[2] Maybe one day we will have chromo computers capable of computation using quarks...

[3] The schematic representations of our examples (figures 1, 2 and 3) are an adaptation of figure 3 in [27]

equation 8. For each approach we also simulate a separate data encoding circuit for $\eta_{\mathbf{x}}$.

3.1 Data Circuit

To prepare a state on a quantum circuit we need a state preparation routine. PennyLane allows you to do this through the `QubitStateVector` function, directly returning the required wave function to the operation queue. This wave function is of the form

$$|\varphi\rangle = \frac{1}{Z} \sum_{i=0}^{2^{N_c}-1} \sqrt{q(y=i|\mathbf{x})}\, |b_i\rangle$$

On an actual quantum computer state preparation is not so trivial. We need to first generate a circuit that prepares the desired state from the zeros states. If we use the Qiskit backend, a recursive initialization algorithm, including optimizations, is called to prepare the state [28]. This algorithm requires at most $2^{n+1} - 2n$ CNOT gates to implement an arbitrary n-qubit wave function. As mentioned before, if we do not have exactly 2^n classes, we set the corresponding $q(y|\mathbf{x})$'s to zero. The density matrix for this circuit is pure by construction and has the the following block structure:

$$\eta_{\mathbf{x}} = |\varphi\rangle\langle\varphi| = \begin{pmatrix} \boxed{\eta_{\mathbf{x}}} \end{pmatrix} \quad \text{(4 classes)}$$

$$\eta'_{\mathbf{x}} = |\varphi\rangle\langle\varphi| = \begin{pmatrix} \boxed{\eta'_{\mathbf{x}}} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 0 \end{pmatrix} \quad \text{(3 classes)}$$

Although the model density matrix $\rho_{\mathbf{x}}$ will initially be filled with random values, it should have a similar structure as $\eta_{\mathbf{x}}$ after convergence.

3.2 Model 1: Amplitude Encoding a Linear Predictor

For the first model, we encode the d features of our data $\mathbf{x} \in \mathbb{R}^d$ into the amplitude of a wave function. We consider a quantum circuit consisting of n qubits, meaning we will have to first map our input to a vector $x' \in \mathbb{C}^{2^n}$. To achieve this, we take a linear map $\mathbf{W} : \mathbb{R}^d \to \mathbb{C}^{2^n}$ where $\mathbf{W}$ is a real $2^n \times d$ matrix,

$$|\psi\rangle = \frac{1}{Z} \sum_{i=0}^{2^n-1} \mathbf{W}_i \cdot \mathbf{x} \, |b_i\rangle$$

where b_i is the bitstring corresponding to integer i and Z is a normalization constant. This construction ensures that the wave function is real. To implement this in PennyLane we use the `QubitStateVector` function, which initializes the quantum circuit with a chosen wave function. All that is left now is measuring a subset of qubits, to construct the model density matrix $\rho_{\mathbf{x}}$. After successfully performing the quantum state tomography for both the model and data circuits, we can calculate the quantum log-likelihood and its corresponding gradients. A schematic representation of this model is given in figure 1.

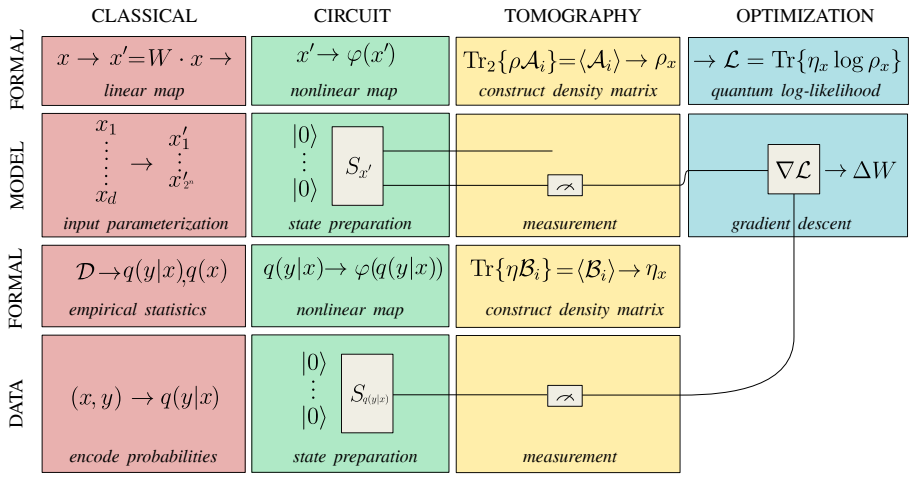

Figure 1: Schematic overview of use case 1. We map each input $\mathbf{x}$ to a vector of length m into the amplitudes of a quantum circuit. We take the number of qubits for the model to be equal to that of the data circuit. Then we construct the density matrix and calculate the gradients through the model with PennyLane and Tensorflow.

3.3 Model 2: Feature Amplitude Encoding with Strongly Entangling Layer

Instead of parametrizing the wave function directly, we can construct a variational circuit to learn the input. We encode the d features of $\mathbf{x}$ into the amplitudes, which requires $n = \text{ceil}(\log_2(d))$ qubits. We then make use of a predefined PennyLane transformation, the `StronglyEntanglingLayer`, which is described in detail in [27]. This variational circuit can cover a large section of the Hilbert space, leading to a rich model expression. This function contains a hyperparamater r, which determines the space between control and target qubit. Stacking layers of `StronglyEntanglingLayer` components with different hyperparameters creates a more complex wavefunction. A single layer consists of $1 \times n \times 3$ single qubit rotation gates, whose angles we control by linear predictors $\mathbf{w}_i \cdot \mathbf{x}$. We set the hyperparameter $r = 1$. As before, we have to measure a subsystem of the circuit ρ to estimate the reduced density matrix $\rho_{\mathbf{x}}$. A schematic representation of the model can be seen in figure **??**

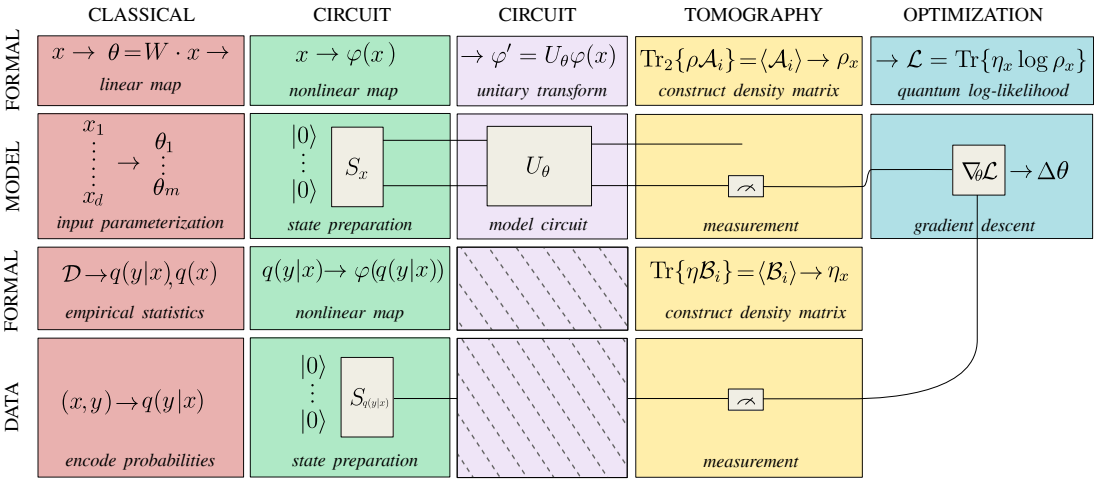

Figure 2: Schematic overview of example 2. We encode each sample $\mathbf{x}$ into the amplitudes of a quantum circuits. We take the number of qubits for the model to be equal to that of the data circuit. Then we construct the density matrix and calculate the gradients through the model with PennyLane and Tensorflow.

3.4 Model 3: Strongly Entangling Layer With Neural Network Input

Up to this point, we have always used amplitude encoding to convert our classical data to a wave function. However, this is not really necessary. We can initialize the circuit in the zero state, and make the gate parameters dependent on a complex feature embedding, such as a dense neural network. We considered a neural network with a single hidden layer of 10 neurons, which has the d-dimensional samples $\mathbf{x}$ as input and outputs $1 \times n \times 3$ parameters for a single `StronglyEntanglingLayer` circuit with hyperparameter $r = 1$. The hidden layer uses ReLu activation functions [29], whereas the output layer consists of Softmax functions scaled to $(0, 2\pi)$. We again measure a subsystem to construct our density matrix. The model is represented schematically in 3.

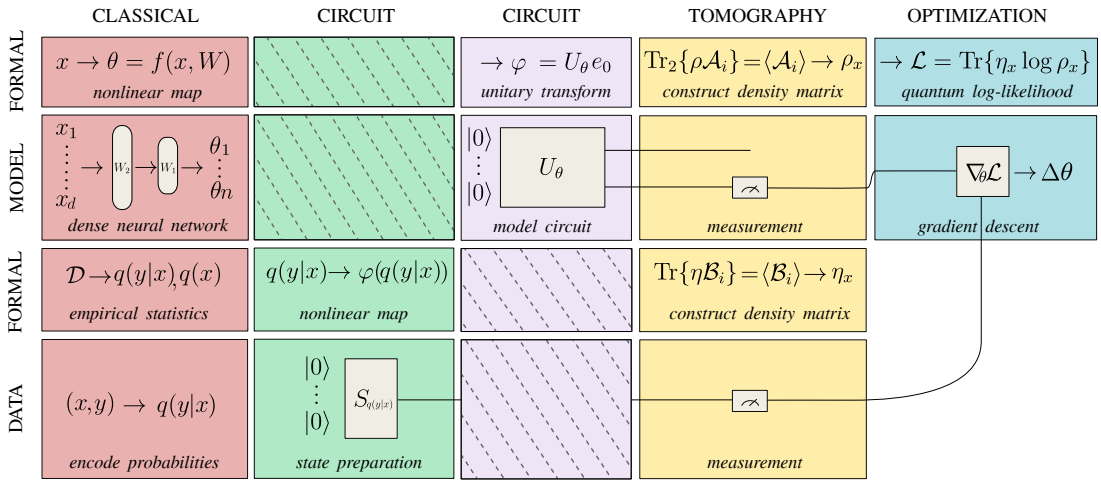

Figure 3: Schematic overview of example 3. The parameters θ of the quantum circuit are now produced by a two layer neural network. We again use the `StronglyEntangled` circuit and trace out qubits until we are left with the density matrix we require.

4 Results

We now consider a couple of toy examples, for all these models.

4.0.1 XOR

As demonstrated in [3], we can learn XOR with the amplitude encoding construction. It is clear that our PennyLane model can do the same, as can be seen in figure 4.

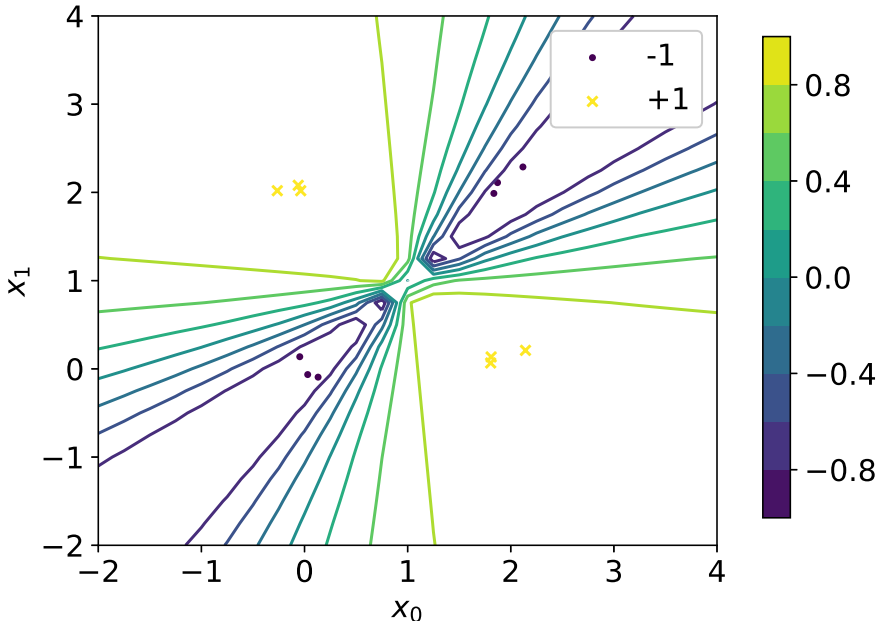

Figure 4: The XOR problem. The colored lines indicate the expected value $p(y|\mathbf{x}, \mathbf{W})$. Both classes get labeled perfectly by this model, due to the parabolic separation boundaries.

4.0.2 Multi-Class Example

In section 2.4, we showed that it is also possible to learn multi-class problems with our model. We consider the problem of learning data $\mathbf{x} = \{(1, 1), (1, -1), (-1, 1), (-1, -1)\}$ with labels $y = \{0, 1, 2, 2\}$, where we have three copies of each data sample. Also, we add a dummy dimension of value one to each sample to function as a bias for the linear predictor $\mathbf{W}$. The result can be seen in figure 5

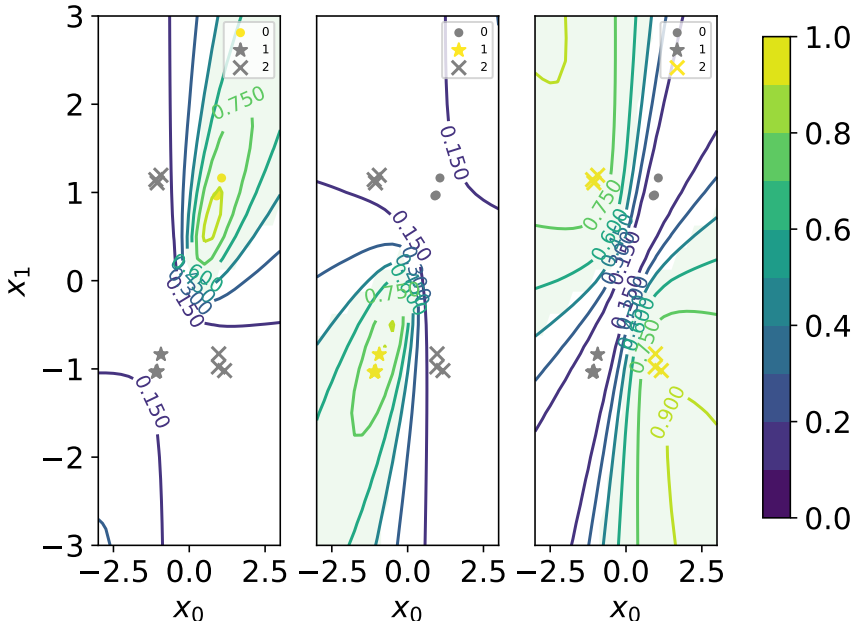

Figure 5: Simple 2D three class problem learned with Model 1. Per tile we see that. we plot the class probability of the class indicated in yellow. The light green area indicates the area where yellow class has the highest probability.

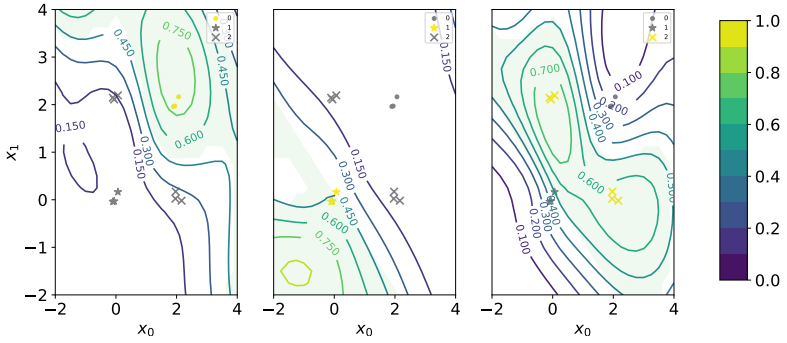

Figure 6: Simple 2D three class problem learned with Model 2.

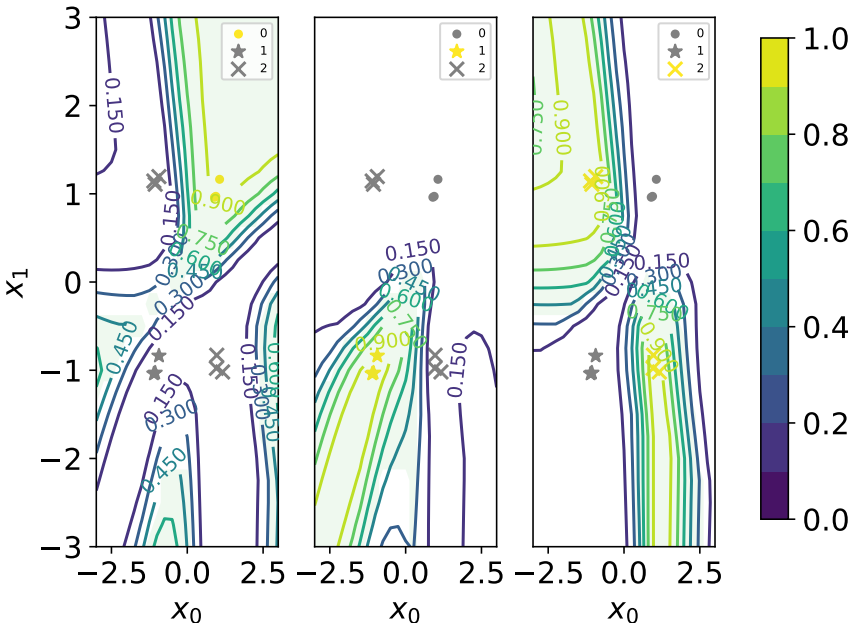

Figure 7: Simple 2D three class problem learned with Model 3.

4.0.3 Noisy Data

Problem with noisy data.

5 Conclusion

It works, but how interesting is this approach? We rely heavily on the usage of linear predictors. Only using a parameterized circuit does little, since it is just a rotation of the data vector, limiting the possible problems learnable.

Problems with only using discrete data so that eta is interesting. See thesis for more detailed analysis on using off-diagonal elements for continuous data.

Comparison with other models is missing, Is a quantum hybrid model more compact? We need multiple measurements per training cycle, and we need to have little noise. What is the complexity?

Studying larger data sets is difficult at the moment, due to concurrency issues.

References

[1] Peter Wittek. "Quantum Machine Learning". In: *University of TorontoX - UTQML101x*. EdX, 2019. URL: `https://courses.edx.org/courses/course-v1:University_of_TorontoX+UTQML101x+1T2019/course/`.

[2] Roeland Wiersema. "Implementing perceptron models with qubits". Radboud University, 2019.

[3] H.J. Kappen R.C. Wiersema. "Implementing perceptron models with qubits". In: *Phys. Rev. A* tbd (2019), tbd.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521 (2015), pp. 436–444.

[5] M. Schuld, I. Sinayskiy, and F. Petruccione. "The quest for a Quantum Neural Network". In: *Quantum Information Processing* 13 (2014), pp. 2567–2586.

[6] S. Jeswal and S. Chakraverty. "Recent Developments and Applications in Quantum Neural Network: A Review". In: *Archives of Computational Methods in Engineering* (May 2018), pp. 1–15.

[7] J. Zhou et al. "Recognition of handwritten numerals by Quantum Neural Network with fuzzy features". In: *International Journal on Document Analysis and Recognition* 2 (1999), pp. 30–36.

[8] N. Kouda et al. "Qubit neural network and its learning efficiency". In: *Neural Computing & Applications* 14 (2005), pp. 114–121.

[9] R. Zhou and Q. Ding. "Quantum M-P Neural Network". In: *International Journal of Theoretical Physics* 46 (2007).

[10] S. Fuhua. "Quantum-Inspired Neural Network with Quantum Weights and Real Weights". In: *Open Journal of Applied Sciences* 5 (2015), pp. 609–617.

[11] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. "Simulating a perceptron on a quantum computer". In: *Physics Letters A* 379 (2015), pp. 660–663.

[12] Kwok Ho Wan et al. "Quantum generalisation of feedforward neural networks". In: *npj Quantum Information* 3 (2017), p. 36.

[13] M. H. Amin et al. "Quantum Boltzmann Machine". In: *Phys. Rev. X* 8 (2018), p. 021050.

[14] H. J. Kappen. *Learning quantum models from quantum or classical data*. Preprint arXiv:1803.11278. 2018.

[15] Ville Bergholm et al. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2018. eprint: `arXiv:1811.04968`.

[16] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[17] M. A. Nielsen and I. L. Chuang. "Quantum Computation and Quantum Information". In: 10th. Cambridge University Press, 2011, pp. 98–111.

[18] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. 1st. Springer Nature Switzerland, 2018.

[19] Eric A. Carlen. "Trace Inequalities and Quantum Entropy: An introductory course". In: The American Mathematical Society, 2009, pp. 73–141.

[20] Daniel F. V. James et al. "Measurement of qubits". In: *Phys. Rev. A* 64 (2001), p. 052312.

[21] Nathan Killoran et al. "Strawberry Fields: A Software Platform for Photonic Quantum Computing". In: *Quantum* 3 (2019), p. 129.

[22] Robert S. Smith, Michael J. Curtis, and William J. Zeng. *A Practical Quantum Instruction Set Architecture*. 2016. eprint: `arXiv:1608.03355`.

[23] Gadi Aleksandrowicz et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: `10.5281/zenodo.2562110`.

[24] Damian S. Steiger, Thomas Häner, and Matthias Troyer. "ProjectQ: an open source software framework for quantum computing". In: *Quantum* 2 (2018), p. 49.

[25] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[26] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[27] Maria Schuld et al. *Circuit-centric quantum classifiers*. 2018. eprint: `arXiv:1804.00633`.

[28] V. V. Shende, S. S. Bullock, and I. L. Markov. "Synthesis of quantum-logic circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25 (2006), pp. 1000–1010.

[29] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks." In: *AISTATS*. Vol. 15. JMLR Proceedings. JMLR.org, 2011, pp. 315–323.