INTER-FACULTY MASTER PROGRAM on
COMPLEX SYSTEMS and NETWORKS
SCHOOL of MATHEMATICS
SCHOOL of BIOLOGY
SCHOOL of GEOLOGY
SCHOOL of ECONOMICS
ARISTOTLE UNIVERSITY of THESSALONIKI
http://cosynet.auth.gr/

MASTER THESIS
COMPARATIVE STUDY OF CLASSICAL AND QUANTUM ENGINEERING LEARNING ALGORITHMS

Supervisor: Dr Kostantinos Chatzisavvas,
Senior Research Fellow

George Balas

2019

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ στα
ΠΟΛΥΠΛΟΚΑ ΣΥΣΤΗΜΑΤΑ και ΔΙΚΤΥΑ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΤΜΗΜΑ ΒΙΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΓΕΩΛΟΓΙΑΣ
ΤΜΗΜΑ ΟΙΚΟΝΟΜΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
http://cosynet.auth.gr/

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Συγκριτική Μελέτη Κλασικών και Κβαντικών Αλγορίθμων Μηχανικής Μάθησης

Επιβλέπων: Δρ. Χατζησάββας Κωνσταντίνος,
Ανώτερος Ερευνητής
Μπαλάς Γεώργιος

2019

Abstract

Machine learning is considered as the top trend in 21^{st} century. Everyone interacts with it, everyone talks about it. In the end of the day, nobody knows exactly what it is and what exactly does. And above that, the last few years a new meaning introduced to us, quantum computing. Alongside machine learning, quantum computing brought quantum machine learning that will breakthrough at science and eventually our lives. This thesis will present some basic principles of the two types of machine learning, will analyze some basic algorithms of both of them, reproduce them in Python and compare them. The whole thesis alongside the code and the results are uploaded public on GitHub at `https://github.com/gkbalas/machine_learning_thesis`.

Keywords

Quantum Machine Learning, Machine Learning, Supervised Learning, Classification, Support Vector Machine, Naive Bayes, Decision Tree, Neural Network, Quantum Neural Network, Quantum Gradient Descent Optimizer, Variational Classifier, Data-reuploading Classifier, IBM Q Experience

Σύνοψη

Η μηχανική μάθηση θεωρείται ως η κορυφαία τάση στον $21^{ου}$ αιώνα. Όλοι αλληλεπιδρούν με αυτήν, όλοι μιλούν γι άυτήν. Στο τέλος της ημέρας, κανείς δεν γνωρίζει ακριβώς τι είναι και τι ακριβώς κάνει. Και πάνω από όλα αυτά, τα τελευταία χρόνια μία νεά έννοια μας παρουσιάζεται, η κβαντική υπολογιστική. Παράλληλα με την μηχανική μάθηση, ο κβαντικός υπολογισμός έφερε την κβαντική μηχανική μάθηση που θα επιφέρει σημαντική ανακάλυψη στην επιστήμη και τελικά στη ζωή μας. Αυτή η εργασία θα παρουσιάσει μερικές βασικές αρχές των δύο τύπων μηχανικής μάθησης, θα αναλύσει κάποιους βασικούς αλγόριθμους και των δύο, θα τις αναπαράγει στην Python και θα τις συγκρίνει. Η εργασία μαζί με τον κώδικα και τα αποτελέσματα δημοσιεύονται δημόσια στο GitHub στη σελίδα `https://github.com/gkbalas/machine_learning_thesis`.

Contents

Chapter 1

Introduction

1.1 Machine Learning

Machine learning is the field of study that gives computers the capability to learn without being explicitly programmed as Arthur Samuel [14], pioneer in the field of artificial intelligence and computer gaming, baptised this chapter of artificial intelligence and statistics, in 1959.

There are two types of machine learning, supervised and unsupervised. Supervised learning builds a mathematical model from a set of data that contains both the inputs and the desired outputs. Classification and regression are the two kinds of supervised learning. When the algorithm is trying to labelize the data to some existing distinct labels is called classification. For continuous outputs in a range of values, regression is used.

In unsupervised learning the data are unlabeled and the algorithm is trying to group them and create clusters of data. So the process is called clustering and it is used to discover patterns in data group into categories. Dimensional reduction is the process of reducing the number of the "features", or inputs, in a set of data.

Machine learning even if it is an almost 60-year old theory it got an impressive recognition, development and usage in the last decade. The reason for this is the exceptional growth in technology and the population of data caused by internet and personal data collection from big companies (like Google, Facebook) or constitutions for experimental reasons.

Except from these two principle ways of machine learning several others

have been proposed through the years for different applications and research theories. These are:

- Semi-supervised Learning
  It uses both labeled and unlabeled data for more learning accuracy.
- Reinforcement Learning
  The cumulative reward is the special characteristic of reinforcement learning and it is used in a great spectrum of applications.
- Self learning
  The system is driven by the interaction between cognition and emotion. It learns without external rewards or external teacher advices.
- Feature learning
  It is often used as a pre-processing step before performing classification or predictions. It replaces manual feature engineering, and allows a machine to both learn the features and use them to perform a specific task.
- Sparse dictionary
  Sparse dictionary learning is a feature learning method where a training example is represented as a linear combination of basis functions, and is assumed to be a sparse matrix.
- Anomaly detection
  Also known as outlier detection, is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data.
- Association Rules
  It is a rule-based machine learning method for discovering relationships between variables in large databases. It is intended to identify strong rules discovered in databases using some measure of "interestingness".

1.2 Quantum Machine Learning

After the production of the first quantum computer for broader use from IBM, the IBM Q Experience, a lot of researches in the field of data analysis and machine learning turn their head to the field of quantum machine learning. The reasons for this are that quantum computing is at last realizable when entanglement between space and earth was achieved in 2017, the classical machine learning algorithms seems to reach their theoretical saturation

and that quantum mechanics can upgrade data analysis and processing to another level.

Before this critical point, several attempts to contribute to the field in theoretical level where accomplished, even from 2000, Ezhov and Ventura [5] tried to create a theory for a quantum neural network. After them, a lot of tries for quantum perceptrons and circuits were trying to adapt the classical machine learning algorithms in quantum mechanics which led on losing the quantum competitive advantages, just using linear algebra and unitary operators or detaching from real problems such as data encoding to feed an algorithm based on qubits.

The ways to implement quantum mechanics in machine learning creates hybrid or pure quantum algorithms of ml. These mechanics and quantum algorithms that have been proposed are:

- Linear algebra simulation with quantum amplitudes
  The goal of algorithms based on amplitude encoding is to formulate quantum algorithms whose resources grow polynomially in the number of qubits n, which amounts to a logarithmic growth in the number of amplitudes and thereby the dimension of the input.

- Quantum machine learning algorithms based on Grover search
  Improving classical machine learning with quantum information processing uses amplitude amplification methods based on Grover's search algorithm, which has been shown to solve unstructured search problems with a quadratic speedup compared to classical algorithms.

- Quantum-enhanced reinforcement learning
  In quantum-enhanced reinforcement learning, a quantum agent interacts with a classical environment and occasionally receives rewards for its actions, which allows the agent to adapt its behavior.

- Quantum annealing
  Quantum annealing is an optimization technique used to determine the local minima and maxima of a function over a given set of candidate functions.

- Quantum sampling techniques
  Sampling from high-dimensional probability distributions is at the core of a wide spectrum of computational techniques. A computationally hard problem, which is key for some relevant machine learning tasks,

is the estimation of averages over probabilistic models defined in terms of a Boltzmann distribution.

- Quantum neural networks
  Quantum neural networks apply the principals quantum information and quantum computation to classical neurocomputing. They are often defined as an expansion on Deutsch's model of a quantum computational network.

- Hidden Quantum Markov Models
  Hidden Quantum Markov Models are a quantum-enhanced version of classical Hidden Markov Models, which are typically used to model sequential data. Unlike the approach taken by other quantum-enhanced machine learning algorithms, HQMMs can be viewed as models inspired by quantum mechanics that can be run on classical computers as well.

1.3 Models

In this thesis, we are going to focus on the problem of classification for both classical and quantum algorithms, compare them in terms of accuracy metrics and execution time. Iris and wine dataset are chosen for being classified. They are both trivial datasets and both of them have been tested and optimized, so the results can be evaluated by third except from metrics. Also for QML will be better for presentation reasons.

The algorithms which will classify these datasets are Decision Tree, Gaussian Naive Bayes, SVM and Neural Network for Classical Machine Learning and Variational Classifier and Data-reuploading Classifier for Quantum Machine Learning.

All the operations will be handled by an Intel i5-7200u processor with 16GB RAM and Ubuntu 18.04 OS running. Python 3 version 3.6.9 with the libraries numpy, scikit-learn, keras, tensorflow, pandas, matplotlib, graphviz, pennylane, openpyxl is the language of the running scripts. For the quantum classifiers and the connection to IBM Q Experience, Pennylane [2] is used, along with the aforementioned libraries in Python 3.

Chapter 2

Classical Machine Learning

In the perpetuity of time many algorithms have been proposed for solving classification problems. The selection between them depends on the original problem and the structure of data. In the next sections some principle algorithm will be analysed and tested in action.

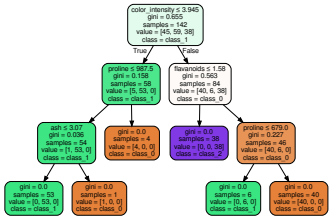
2.1 Decision Tree

Decision Tree Classifier is a flowchart-like graph, in which every node is a test about the features of the sample. It is also called as tree-like model, because three characteristics exist:

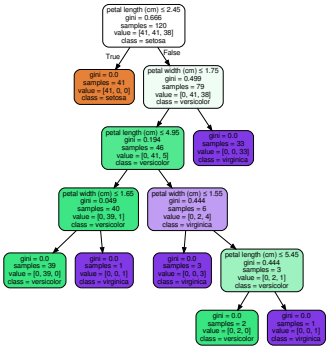
- root (first division or test outcome)
- branch (outcome of the test)
- leaf (decision-class label)

There are either classification or regression trees whether the outcome is a class or a number. Several algorithms have been proposed as decision tree classifier creating. The more notable and widely-known are:

- ID3 (Iterative Dichotomizer 3)
- C4.5 (successor of ID3)
- C5.0 (greatly improved successor of C4.5, commercial)
- CART (Classification And Regression Tree)
- CHAID (CHi-squared Automatic Interaction Detector)



(a) Wine



(b) Iris

Figure 2.1: Datasets Trees

- MARS (extends decision trees to handle numerical data better)
- Conditional Inference Trees (Statistics-based approach that uses non-parametric tests as splitting criteria, correcter for multiple testing to avoid overfitting)

All of them come with their own advantages and disadvantages, but also share some common as they inherit them from the basic concept. The great difference from other algorithm concepts is that it is understandable and easy to interpret to human mind as it follows the same concepts as human logic. This is pictured in the white-box model it has. Except of that they need no data preparation to run the models and are able to handle both numerical and categorical data.
The drawbacks that decision trees come with have to do with the lack of robustness of the models and overfitting to the sample. Also they are NP-complete problem and greedy algorithm.

2.2 Support Vector Machine

SVM is a supervised learning model, which is used either for classification or regression. This method tries to find the optimal hyperplane to minimize the error of the predicted $f(x)$ instead of the actual y. Optically, it draws the boundary, between the two classes of our sample, that has the greater margin of both of them. In its simplest form it is the dot product $\mathbf{w} \cdot \mathbf{x_i} + \mathbf{b} = \mathbf{0}$ for a linear classifier. In the case of non-linear classifiers, SVM uses the kernel trick, where every dot product is replaced by a non-linear kernel function. Some common kernels are:

- Polynomial (homogeneous): $K(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j})^d$
- Polynomial (in-homogeneous): $K(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$
- Gaussian radial basis function: $K(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma\|\mathbf{x_i} - \mathbf{x_j}\|^2)$, for $\gamma > 0$
- Hyperbolic tangent: $K(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa\mathbf{x_i} \cdot \mathbf{x_j} + c)$

The algorithm of SVM was invented by Vladimir Vapnik and Alexey Chervonenkis in 1963. The kernel trick to maximum-margin hyperplanes for non-linear classifiers is introduced in 1992 by Bernhard Boser, Isaabelle Guyon and Vladimir Vapnik [3].

Nowadays, SVM is widely used in many applications such as medical decision support, face detection and pattern recognition, speaker identification, even text categorization and many more.The reason for this largely usage is the minimum limitations it has:

- Two-class algorithm (for multiple classes the approach is multiple binary classifications)
- Requires full labeling of input data
- Kernel choose
- Parameters of a solved model are difficult to interpret

2.3 Naives Bayes

Naives Bayes Classifier is based on Bayes' Theorem Probabilities without being a single algorithm but a family of algorithms. Except the Bayes' Theorem, they share a common principle, the feature independence of each other, given the class variable. Naive Bayes is a conditional probability $P(C_k|\mathbf{x})$, which under the Bayes' Theorem is decomposed as

$$P(C_k|\mathbf{x}) = \frac{p(C_k)p(x|X_k)}{p(x)}.$$

Under the main assumption of the features being independent between them, we can safely assume that

$$P(C_k|\mathbf{x}) = P(C_k|\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}) = \frac{p(C_k)\Pi_{i=1}^{n} p(x_i|C_k)}{p(x_1)\ldots p(x_n)}.$$

Because the denominator is the same for k classes, the choice of the best class is made by the function

$$C_k = \arg \max_{k \in \{1,\dots,k\}} p(C_k) \Pi_{i=1}^{n} p(x|C_k).$$

This classifier works for discrete data, for continuous data we can use one of

- Gaussian Naive Bayes:

$$p(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(x_i-\mu_k)^2}{2\sigma_k^2}}$$

  where μ_k and σ_k^2 the mean and the variation of x in class k.

- Multinomial Naive Bayes:

$$p(x|C_k) = \frac{(\Sigma_{i=1} x_i)!}{\Pi_{i=1}(x_i!)} \Pi_{i=1} p_{ki}{}^{x_i}$$

  which becomes linear classifier

$$\log p(C_k|x) \propto \log(p(C_k)\Pi_{i=1}^{n} p_{ki}{}^{x_i}) = \log p(C_k) + \Sigma_{i=1}^{n} x_i \log p_{k_i}$$

- Bernoulli Naive Bayes:

$$p(x|C_k) = \Pi_{i=1}^{n} p_{k_i}^{x_i} (1 - p_{k_i})^{(1-x_i)}$$

  where x_i a boolean expressing the occurrence or absence of ith term and p_{k_i} the probability of class C_k generating the term x_i.

NBC is popular even though in real-life problems there is always some dependence between the features, because of two reasons. Firstly, the decoupling of the class conditional feature distributions which is translated as each distribution can be independently estimated as an one-dimensional. Second, the little training time as a result of the above feature solving the curse of dimensionality. It is mainly used for document classification and spam filtering.

2.4 Neural Network

Artificial Neural Network is the field of machine learning that had the greatest evolve in practical use the last two decades, because two of the major difficulties where solved. Internet expansion helped solving the data problem and high-end machines for the great complexity.

The first contribution to the field was made by Warren McCullogh and Walter Pits [9] in 1943 introducing us the first artificial neuron, called threshold logic. Which was later used by Rosenblatt [13] to make the first Perceptron. This perceptron has several inputs with their weights, by multiplying them, summing them with a bias and feed them to a linear step function which works as a threshold for the neuron. $Sum = \sum_{i=1}^{n} I_i W_i$, $y = f(Sum)$. Where y is the neuron be active $y = 1$ or inactive $y = 0$.

Iraakhenko and Lapa published the first functional networks in 1965. The problem of weight updating was solved by Werbos [17] (1975) with his backpropagtion algorithm.

These publications formed the artificial neural networks as we know them. Some of these network structures and models are:

- Multi-Layer Neural Networks (MLNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Deep Learning Neural Network

These are proposed even for different types of use of machine learning either supervised, unsupervised or hybrid. The reason of the great success of ANNs is the universal usage they offer. They are used for image and speech recognition, translation, social network filtering, playing board and video games, medical diagnosis and many more.

2.5 Metrics

Accuracy
Accuracy is the first metric to use for an easy and fast evaluation of the algorithm.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ Predictions}$$

the great and only drawback in this metric is about the proportion of samples in each class. Equal separated samples can solve this problem.

Precision and Recall
Precision and Recall have the same logic behind their mathematic formula about true or false positive and negative predictions.

$$Prediction = \frac{TruePositives}{TruePositives + FalsePositives}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

They try show the proportion of the true positives samples to two different sets, the predictions the algorithm made and the actual positives samples.

F_1 score
In order to identify uneven class distribution and even evaluate Precision and Recall metrics, there is their harmonic mean called F_1 score.

$$F_1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}} \Rightarrow$$

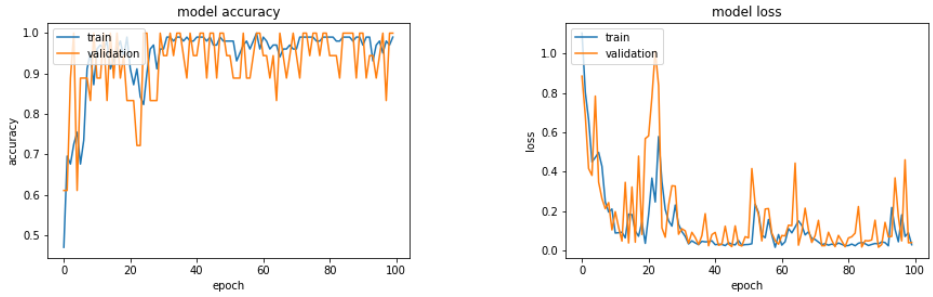
$$\Rightarrow F_1 = 2 * \frac{Presicion * Recall}{Presicion + Recall}$$

The translation of F_1 score is that it reveals how precise the classifier is (how many instances it classifies correctly) and how robust it is (it does not miss a significant number of instances).

2.6 Results

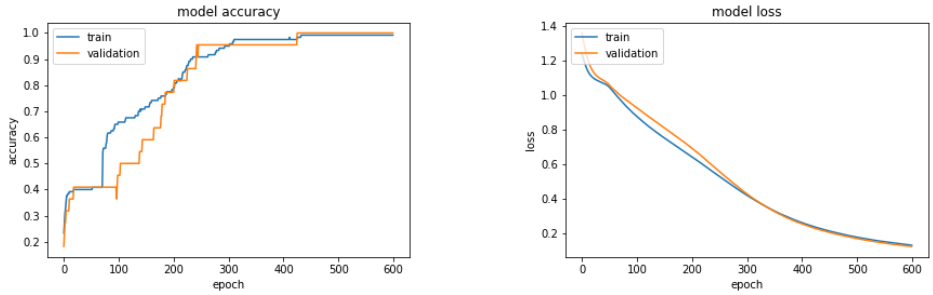
Iris Dataset
All four algorithms have been fed with iris dataset trying to classify data to 3 different classes equal distributed with 50 observation each one from the total and wine dataset with 3 different classes and unequal number of observation for them. The results were really good despite lack of optimization of the algorithms on the datasets.

All algorithms were able to overfit the data, which is a drawback, but they were able to predict all unseen data and reach 100%. To minimize any possible error or the optimizers do not adheres in local minima the models run



(a) Accuracy (b) Loss

Figure 2.2: Neural Network for iris



(a) Accuracy (b) Loss

Figure 2.3: Neural Network for wine

many times and selected overall the best results.The aforementioned metrics evaluated the models with the best results. Even when a model could not reach 100%, their values were high enough to classify the models as really accurate. The execution time for the neural network was between 1-2 seconds and for the rest of the methods were much less than a second.

Wine Dataset

The wine dataset with 13 attributes and 178 instances is classified to three types of wine. The classes are unequal and the data need scaling. We tried to check the algorithms without preprocessing so on this dataset the results were not that accurate. All the methods had more than 95% accuracy on training and on test validation. Which means that besides the overfiting where able to classify unseen data. The other metrics followed the results of accuracy and evaluate the algorithms as really good and accurate. The run-times were 6 times the iris' ones and that is why this dataset is a lot bigger and complex than iris.

Chapter 3

Quantum Machine Learning

3.1 Quantum Gradient Descent Optimizer

First of all, we are going to present the differences between the classical gradient descent and quantum analogue one. In classical neural networks, the natural gradient descent was first introduced by Amari (1998) [1]. The standard gradient descent is

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

where F is the Fisher information matrix. The Fisher information matrix acts as a metric tensor, transforming the steepest descent in the Euclidean parameter space to the steepest descent in the distribution space.

In a similar vein, it has been shown that the standard Euclidean geometry is sub-optimal for optimization of quantum variational algorithms (Harrow and Napp, 2019). The space of quantum states instead possesses a unique invariant metric tensor known as the Fubini-Study metric tensor g_{ij}, which can be used to construct a quantum analog to natural gradient descent:

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

where g^{+} refers to the pseudo-inverse.

A block-diagonal approximation to the Fubini-Study metric tensor of a variational quantum circuit can be evaluated on quantum hardware.

Considering a variational quantum circuit

$$U(\theta) |\psi_0\rangle = V_L(\theta_L) W_L V_{L-1}(\theta_{L-1}) W_{L-1} \dots V_l(\theta_l) W_l \dots V_0(\theta_0) W_0 |\psi_0\rangle$$

where

- $|\psi_0\rangle$ is the initial state,
- W_l are layers of non-parametrized quantum gates,
- $V_l(\theta_l)$ are layers of parametrized quantum gates with n_l parameters $\theta_l = \theta_0^{(l)}, \ldots, \theta_n^{(l)}$.

Further, assuming all parametrized gates can be written in the form $X(\theta_i^{(l)}) = e^{i\theta_i^{(l)} K_i^{(l)}}$, where $K_i^{(l)}$ is the generator of the parametrized operation.

For each parametric layer l in the variational quantum circuit the $n_l \times n_l$ block-diagonal submatrix of the Fubini-Study tensor $g_{ij}^{(l)}$ is calculated by:

$$g_{ij}^{(l)} = \langle\psi_{l-1}|K_i K_J|\psi_{l-1}\rangle - \langle\psi_{l-1}|K_i|\psi_{l-1}\rangle \langle\psi_{l-1}|K_j|\psi_{l-1}\rangle$$

where

$$|\psi_{l-1}\rangle = V_{l-1}(\theta_{L-1})W_{l-1} \ldots V_0(\theta_0)W_0 |\psi_0\rangle$$

(that is, $|\psi_{l-1}\rangle$ is the quantum state prior to the application of parameterized layer l), and we have $K_i \equiv K_i^{(l)}$ for brevity.

In our code we reproduced a variational quantum circuit trying to compare two gradient descent for it, vanilla gradient descent and quantum natural gradient descent. The results where obviously much better for quantum natural gradient descent.

3.2 Variational Classifier

Variational quantum classifiers are quantum circuits that can be trained from labeled data to classify new data samples. Farhi and Neven [6] as well as Schuld et al. [16] have proposed the architecture for this task.

This architecture reminds a quantum analog of neural network, because of weights, bias, error computing and backpropagation. Inputs are coded according to the scheme in Möttönen, et al. [10], or—as presented for positive vectors only—in Schuld and Petruccione [15]. Also controlled Y-axis rotations decomposed into more basic circuits following Nielsen and Chuang [11].

State preparation is not as simple as when we represent a bitstring with a basis state. Every input x has to be translated into a set of angles which

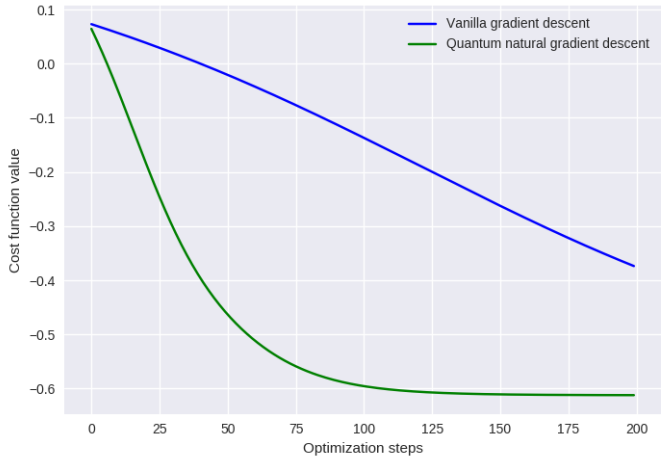


Figure 3.1: Gradient descent optimizers

can get fed into a small routine for state preparation. To simplify things a bit, data will be worked from the positive subspace, so that signs can be ignored (which would require another cascade of rotations around the z axis).

In order to test this circuit, iris data is chosen and preprocessed as mentioned above keeping the first two classes and features.

The features feed the model as angles for the qubits' rotations. The two classes are encoded as 1 (blue) and -1 (red). Qubits are linked together with CNOT gates passing the information of their angle rotation to the next one, ending to the last one which state will be measured and compute the error. Weights are predefined rotations between qubits that are updated every time after backpropagation.

The system needed only 67 iterations to reach 100% accuracy in validation and stay stabilized at this point. Due to lack of data only 15 inputs held as validation. The model was overfitted but also it was able to predict unseen data accurately. The whole process of training kept 197.78 seconds. The architecture of the circuit needed only 2 qubits.

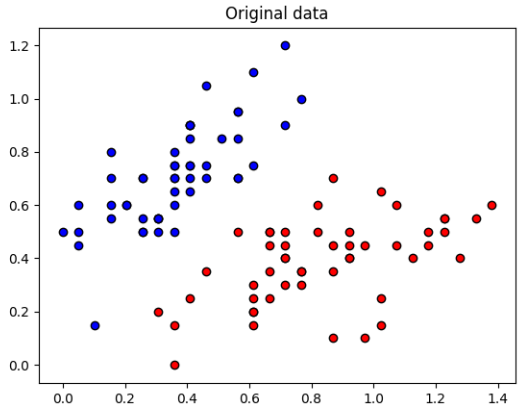
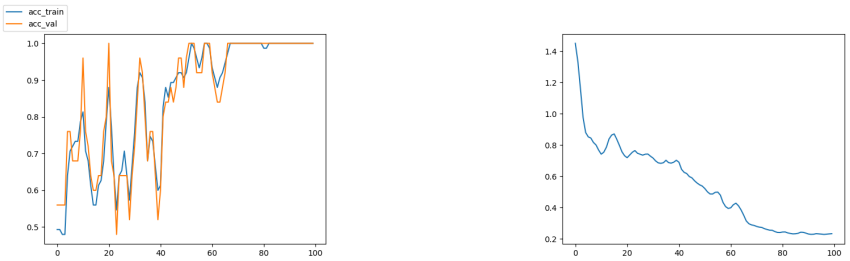


Figure 3.2: Iris data preprocessed



(a) Accuracy (b) Cost

Figure 3.3: Variational Metrics

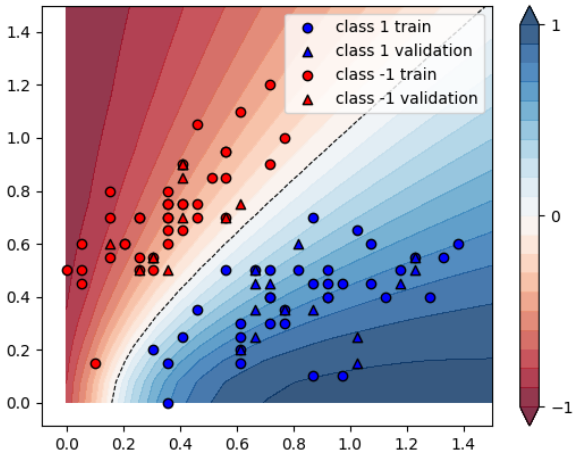


Figure 3.4: Variational Predictions

3.3 Data-reuploading Classifier

A single-qubit quantum circuit which can implement arbitrary unitary operations can be used as a universal classifier much like a single hidden-layered Neural Network. Pérez-Salinas et al. [12] discuss this with their idea of 'data reuploading'. It is possible to load a single qubit with arbitrary dimensional data and then use it as a universal classifier.

A simple classification problem is considered training a single-qubit variational quantum circuit to achieve this goal. The data is generated as a set of random points in a plane (x1,x2) and labeled as 1 (blue) or 0 (red) depending on whether they lie inside or outside a circle. The goal is to train a quantum circuit to predict the label (red or blue) given an input point's coordinate.

A single-qubit quantum state is characterized by a two-dimensional state vector and can be visualized as a point in the so-called Bloch sphere. Instead of just being a 0 (up) or 1 (down), it can exist in a superposition with say 30% chance of being in the $|0\rangle$ and 70% chance of being in the $|1\rangle$ state. This is represented by a state vector $|\psi\rangle = 0.3\,|0\rangle + 0.7\,|1\rangle$ - the probability "amplitude" of the quantum state. In general we can take a vector (α, β) to represent the probabilities of a qubit being in a particular state and visualize it on the Bloch sphere as an arrow.

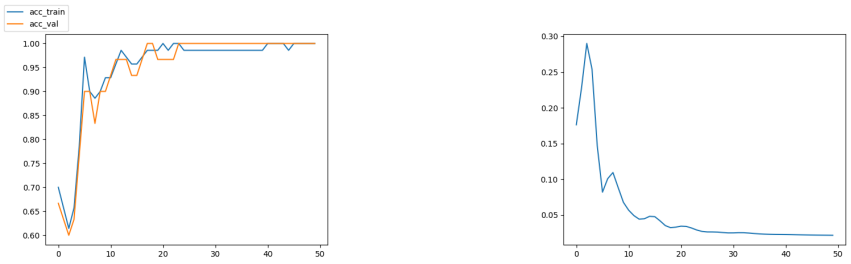
In order to load data onto a single qubit, we use a unitary operation $U(x_1, x_2, x_3)$ which is just a parameterized matrix multiplication representing the rotation of the state vector in the Bloch sphere. Pérez-Salinas et al. discuss how to load a higher dimensional data point ($[x_1, x_2, x_3, x_4, x_5, x_6]$) by breaking it down in sets of three parameters ($U(x_1, x_2, x_3), U(x_4, x_5, x_6)$).

Once we load the data onto the quantum circuit, we want to have some trainable nonlinear model similar to a neural network as well as a way of learning the weights of the model from data. This is again done with unitaries, $U(\theta_1, \theta_2, \theta_3)$, such that we load the data first and then apply the weights to form a single layer $L(\overrightarrow{\theta}, \overrightarrow{x}) = U(\overrightarrow{\theta})U(\overrightarrow{x})$. In principle, this is just application of two matrix multiplications on an input vector initialized to some value. In order to increase the number of trainable parameters (similar to increasing neurons in a single layer of a neural network), we can reapply this layer again and again with new sets of weights, $L(\overrightarrow{\theta_1}, \overrightarrow{x})L(\overrightarrow{\theta_2}, \overrightarrow{x}) \ldots L(\overrightarrow{\theta}, \overrightarrow{x})$ for L layers.

So far, we have only performed linear operations (matrix multiplications) and we know that we need to have some nonlinear squashing similar to activation functions in neural networks to really make a universal classifier (Cybenko 1989 [4]). Here is where things gets a bit quantum. After the application of the layers, we will end up at some point on the Bloch sphere due to the sequence of unitaries implementing rotations of the input. These are still just linear transformations of the input state. Now, the output of the model should be a class label which can be encoded as fixed vectors (Blue = 4[1, 0], Red = [0, 1]) on the Bloch sphere. We want to end up at either of them after transforming our input state through alternate applications of data layer and weights.

We can use the idea of the "collapse" of our quantum state into one or other class. This happens when we measure the quantum state which leads to its projection as either the state 0 or 1. We can compute the fidelity (or closeness) of the output state to the class label making the output state jump to either $|0\rangle$ or $|1\rangle$. By repeating this process several times, we can compute the probability or overlap of our output to both labels and assign a class based on the label our output has a higher overlap. This is much like having a set of output neurons and selecting the one which has the highest value as the label.

We can encode the output label as a particular quantum state that we want to end up in. We construct an observable corresponding to the output label



(a) Accuracy (b) Cost

Figure 3.5: Reuploading Classifier Metrics

using the Hermitian operator. The expectation value of the observable gives the overlap or fidelity. We can then define the cost function as the sum of the fidelities for all the data points after passing through the circuit and optimize the parameters ($\overrightarrow{\theta}$) to minimize the cost.

$$Cost = \sum_{datapoints} (1 - fidelity(\psi_{output}(\overrightarrow{x}, \overrightarrow{\theta}), \psi_{label}))$$

Now, we are going to use Adam optimizer to maximize the sum of the fidelities over all data points (or batches of datapoints) and find the optimal weights for classification. Gradient-based optimizers such as Adam (Kingma et. al., [7]) can be used if we have a good model of the circuit and how noise might affect it. Or, some gradient-free method can be used such as L-BFGS (Liu, Dong C., and Nocedal, J., [8]) to evaluate the gradient and find the optimal weights where the quantum circuit can be treated as a black-box and the gradients are computed numerically using a fixed number of function evaluations and iterations.

Also for this model iris dataset has been selected from which the first two classes and features where chosen and the preprocessing of variational classifier. The model was a real fast-learner as it needed only 18 epochs to predict all the unseen data with the right label. Furthermore, it was not able to stabilize the accuracy to 100% which means it was not able to overfit data. Training for 50 epochs had a run time of 97.44 seconds.

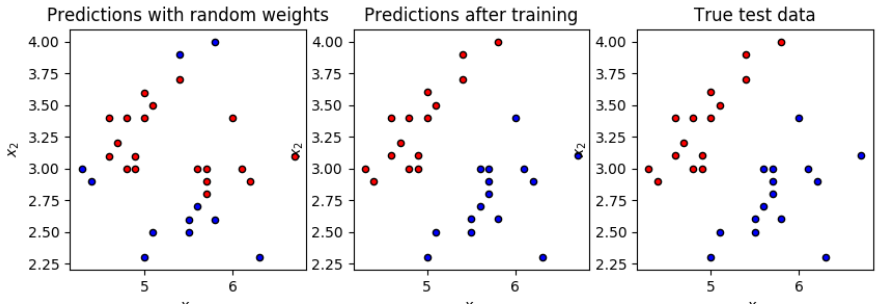


Figure 3.6: Reuploading Predictions

3.4 IBM Q Experience

In May 2016, IBM launched the IBM Q Experience, with a five qubit quantum processor and matching simulator connected in a star shaped pattern, which users could only interact with through the quantum composer, with a limited set of two-qubit interactions, and a user guide that assumed background in linear algebra.

Nowadays, it hosts 8 processors on the cloud, open-source.

- Armonk: 1 qubit
- Yorktown: 5 qubit
- Ourense: 5 qubit
- Vigo: 5 qubit
- London: 5 qubit
- Burlington: 5 qubit
- Essex: 5 qubit
- Melbourne: 14 qubit

IBM provides the architecture of each one of them, the gate error rates and the calibration info. Anyone is able to create an account and test quantum computing. The UX is really simple as it is a drag'n'drop framework. Also it provides online Qiskit notepad to write your own quantum code on the cloud to run it later.

We used Pennylane's plugin to connect to IBM's processors. We confronted

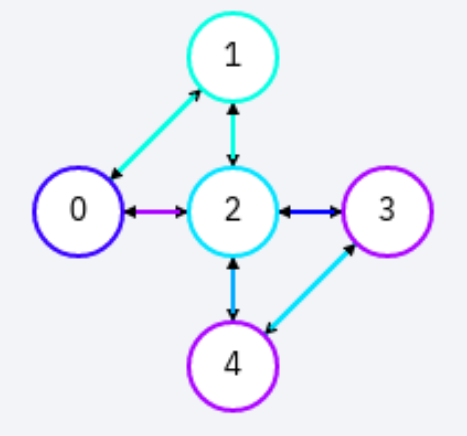


Figure 3.7: Architecture 1

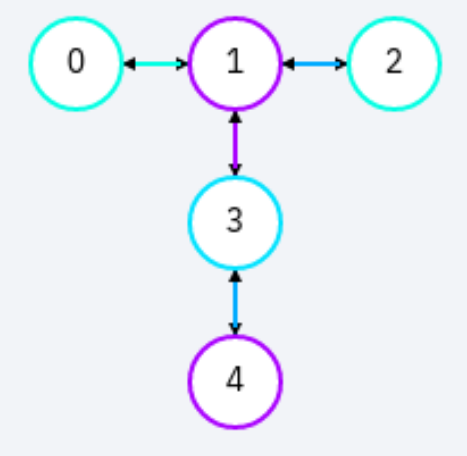


Figure 3.8: Architecture 2

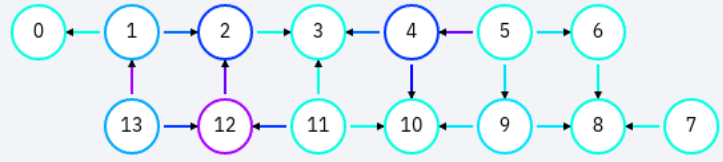


Figure 3.9: Architecture 3

with two great deals. First of all, the so-called improvement of speed and computational complexity reduction of quantum computers had fallen apart on our try to minimize the errors of the measurements. In order to have a robust result we had to run every operation for 1000 shots. This led to really long-waiting, complex models and code that did not offer any speedup at all. A second hurdle was the pennylane's plugin that was returning less results than was expected, ending up raising errors on the code, where it shouldn't. Debugging was also almost unable to be done with every iteration of our models running for almost 2 hours at least.

Chapter 4

Conclusion

Since this thesis started, plenty of papers were published, many new models were proposed and a lot of them were more realistic and able to test on a simulator or even a real quantum computer.

The Quantum Machine Learning is on its baby steps and needs to give more time to be fully operative. Quantum Computers are really fast and change the way of processing data, but still there is a long road to go through and structural problems to solve.

Firstly, the error correction and the probabilistic nature of quantum world demands thousands of shots of each problem implementation to optimize the result and to be sure that this is the right one, slowing down the whole process and losing its competitive advantage.

Secondly, quantum mechanics have been proposed in data analysis and data science to solve the problem of big data processing cause of the exponential parallel process ability and data loading. In real terms, this theory comes to many physical boundaries. The architecture of the quantum circuit and the qubits' in-between connections are throttling and decreases the realization of many theoretical models. The great quantum entanglement is still not only a nature mystery, but also a great difficult and costly task to perform it and maintain it.

QML gives a hope for new discoveries and breakthroughs, but still it does not answer real life problems and has the uncertainty of its results for the next years. Also it was a real disappointment that IBM Q Experience and Pennylane were not able to answer to generic tasks working together leaving a gap for an easier software development.

On the other hand, Classical Machine Learning has been so much studied the last decade that has produced really good, optimized and fast models and applications, passing the problem of classification and prediction to data preprocessing, lack of data and to the point problem question.

Future personal goals have been set to continue study the two cutting-edge fields, work and offer on them. Solve the connection problem of Pennylane and IBM Q Experience. Run real data models in more quantum circuits and computers like Google Cirq. Also, start collaborate with other colleagues and contribute to the collective knowledge that helped me with this thesis.

Bibliography

[1] Shun-ichi Amari. "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2 (Feb. 1998), pp. 251–276. DOI: 10.1162/089976698300017746. URL: https://doi.org/10.1162/089976698300017746.

[2] Ville Bergholm et al. "PennyLane: Automatic differentiation of hybrid quantum-classical computations". In: (2018).

[3] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*. ACM Press, 1992. DOI: 10.1145/130385.130401. URL: https://doi.org/10.1145/130385.130401.

[4] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/bf02551274. URL: https://doi.org/10.1007/bf02551274.

[5] A.A. Ezhov and Dan Ventura. "Quantum Neural Networks". In: Jan. 2000, pp. 213–235. DOI: 10.1007/978-3-7908-1856-7_11.

[6] Edward Farhi and Hartmut Neven. "Classification with Quantum Neural Networks on Near Term Processors". In: (Feb. 2018).

[7] Diederik Kingma et al. "Semi-Supervised Learning with Deep Generative Models". In: *Advances in Neural Information Processing Systems* 4 (June 2014).

[8] Dong C. Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical Programming* 45.1-3 (Aug. 1989), pp. 503–528. DOI: 10.1007/bf01589116. URL: https://doi.org/10.1007/bf01589116.

[9] Warren S. McCulloch and Walter Pitts. "Neurocomputing: Foundations of Research". In: ed. by James A. Anderson and Edward Rosenfeld. Cambridge, MA, USA: MIT Press, 1988. Chap. A Logical Calculus of the Ideas Immanent in Nervous Activity, pp. 15–27. ISBN: 0-262-01097-6. URL: http://dl.acm.org/citation.cfm?id=65669.104377.

[10] Mikko Möttönen et al. "Transformation of Quantum States Using Uniformly Controlled Rotations". In: *Quantum Info. Comput.* 5.6 (Sept. 2005), pp. 467–473. ISSN: 1533-7146. URL: http://dl.acm.org/citation.cfm?id=2011670.2011675.

[11] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2009. DOI: 10.1017/cbo9780511976667. URL: https://doi.org/10.1017/cbo9780511976667.

[12] Adri'an P'erez-Salinas et al. "Data re-uploading for a universal quantum classifier". In: 2019.

[13] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* (1958), pp. 65–386.

[14] A. L. Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of Research and Development* 44.1.2 (Jan. 2000), pp. 206–226. ISSN: 0018-8646. DOI: 10.1147/rd.441.0206.

[15] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-96424-9. URL: https://doi.org/10.1007/978-3-319-96424-9.

[16] Maria Schuld et al. "Circuit-centric quantum classifiers". In: 2018.

[17] P. J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (Oct. 1990), pp. 1550–1560. ISSN: 1558-2256. DOI: 10.1109/5.58337.