Master Semester Project in Quantum and Computer Science

Binary classification and *kernel methods* in noisy QML

Advisors:
Prof. Nicolas Macris
PhD. Anastasia Remizova

Presented by:
Victor Braun

Laboratory:
Laboratory of Statisical Mechanics of Inference in Large Systems (SMILS)

Autumn Semester
2024/2025 Academical Year

Abstract

During the last years, the field of quantum computing has evolved significantly. It has been applied to different domains, seeking the grail of *quantum supremacy*, the turning point where quantum computers could achieve results that no classical computer could ever obtain.
In this project, we explore the so-called *"Quantum Machine Learning"* field, following the steps of [Havlicek et al., 2018]. We will see how some quantum algorithms can help achieve quantum supremacy in the simplest *binary classification* Machine Learning task.

All code used in the project is available in this repository.

Table of Contents

Introduction

The *supervised binary classification* task is defined as follows: we are given a set of inputs $\mathcal{S} = T \cup S \subseteq \Omega$ and a *partial labeling* function $\tilde{m} : T \to \{\pm 1\}$. Our goal is then to deduce the "hidden" complete labeling function $m : \Omega \to \{\pm 1\}$ only knowing $\tilde{m}$. Of course, we want our approximation $\hat{m}$ to be equal to m with high probability on a random input of Ω.

In the project, we will consider $\Omega \subset \mathbb{R}^n$, where n is the *input space dimension*. Furthermore, we will refer to T as the *training* set and to S as the *testing* set.

In classical ML, one of the most straightforward and efficient ways of doing it is with the *Support Vector Machine* (SVM) technique. Because this requires the data to be linearly separable, we ensure that using a *feature map* $\Phi : \Omega \to F$, where F is called the *feature space* (it is believed that any data is linearly separable in a sufficiently higher dimension). In this new space, we aim to find the best hyperplane (parametrized by some θ) that separates the data classes.

This technique is highly related to the so-called *Kernel Methods*, in fact, we can solve the SVM optimization task using the underlying *kernel function* $K(x, y) = \Phi(x) \cdot \Phi(y)$. This is where is gets interesting, a way to quantum advantage would be the possibility to implicitly compute **kernel functions that a classical computer couldn't**.

Let's explore how such functions look like, how quantum computer successfully compute them and how they could be used in our problem.

However, things are not as easy, in fact, one crucial parameter when using quantum hardware is to consider the strong *noise* and its effects. For that reason, we'll focus on simple noise models and already study how our algorithms lose precision and try to introduce possible mitigations.

1. The Quantum Model

Quantum circuits deal with *quantum states*, generally, they lie in Hilbert spaces (vector spaces) with a very-high dimensionality. We can thus imagine to use such states as features, with the state space being then the *feature space*. In other words, we can define our feature map $\Phi : x \to |\Phi(x)\rangle \in \mathbb{C}^{2^N}$, which maps an input to a quantum state on N qubits. In this new space, we find the separating hyperplane to classify the quantum state. This can be viewed as the *Quantum Machine Learning (QML)* framework.

In this framework, the quantum circuit is divided in three parts:

1. *Initalization*: The zero state ($|0^N\rangle$, where 2^N is the dimension of the feature space) is mapped to a feature vector $|\Phi(x)\rangle = \mathcal{U}_{\Phi(x)}|0^N\rangle$, where the initialization unitary $\mathcal{U}_{\Phi(x)}$ depends on the input x.

2. *Ansatz*: The variational gate $W(\theta)$ is applied to our state, this can be viewed as the hyperplane that will separate our data. We call this circuit the *Ansatz*, it is a "guessing" of the form of the optimal unitary in a specific problem. The parameters θ are then optimized during the training phase.

3. *Measurement*: Finally, our state is measured in the computational basis ($Z^{\otimes N}$ observable). Depending on the result, the datapoint is classified via an arbitrary *parity function* previously chosen $f : \{\pm 1\}^N \to \{\pm 1\}$.

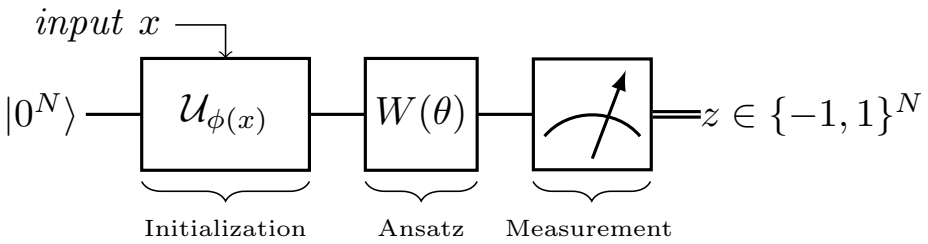

Figure 1: *Typical workflow of a QML model.*

1.1. The Feature Map Gate

In this model, the underlying dot product (or *kernel function*) is $K(x, y) = |\langle\Phi(x)|\Phi(y)\rangle|$, we believe some kernels can be very difficult to estimate on classical computer for high N, as discussed in [Bremner et al.]. We'll use one specific family of feature gates that have such kernels, more precisely:

$$\mathcal{U}_{\Phi(x)} = U_{\Phi(x)} H^{\otimes N} U_{\Phi(x)} H^{\otimes N}, \tag{1}$$

where H is the Hadamard gate and $U_{\Phi(x)}$ is a diagonal gate depending on the input vector x. The general form of the diagonal form can be written as:

$$U_{\Phi(x)} = \exp\left(i \sum_{S\subseteq[n]} \phi_S(x) Z_S\right), \quad Z_S = \bigotimes_{i\in S} Z_i \tag{2}$$

As a side note, $|S| = 1$ corresponds to single qubit rotations, $|S| = 2$ to 2-qubit interactions and so on. So, any $U_{\Phi(x)}$ with this firm could be chosen as long as it is easy to implement on the hardware (e.g some quantum processors only allow up to 2-qubit interactions)

1.2. The Variational Circuit

For our circuit, we need to "guess" its form. One common Ansatz is the family of circuits splitting the circuit into alternating local and entangling gates:

$$W(\theta) = U^{(l)}_{loc}(\theta_l) U_{ent} ... U^{(2)}_{loc}(\theta_2) U_{ent} U^{(1)}_{loc}(\theta_1), \quad U^{(i)}_{loc}(\theta_i) = \bigotimes_{j=1}^{N} \Theta^i_j(\theta_{i,j}) \tag{3}$$

Again, any $W(\theta)$ in this family could work, however, the hardware may be limiting so we may choose relatively "easy" gates. For example, in this project we set $\Theta^i_j(\varphi_z, \varphi_y) = R_Z(-\varphi_z) R_Y(-\varphi_y)$ and $U_{ent} = \prod_{\{i,j\}\in E} CZ(i,j)$. Where E are the different pairs of interacting qubits the quantum hardware allows. Because each $\theta_{i,j}$ contains two angles, it follows that $\theta \in \mathbb{R}^{2Nl}$.

1.3. The QML Algorithms

Now that we have most pieces of the puzzle, let's remember the initial goal of this circuit. We want it to behave as a binary *classifier*, in other words, we would like to use its outcomes to label any $x \in \Omega$ so that $\Pr(m(x) = \hat{m}(x))$ is very high ($\hat{m}$ being this label *prediction*).

The way how we do it is pretty natural: as mentioned earlier, we arbitrary choose a function f that has the role to map any outcome $z \in \{\pm 1\}^N$ to a label $f(z) \in \{\pm 1\}$. Now, we know that the outcome of a quantum measurement is probabilistic, that is why we can't assign a label just after one measurement. As in most quantum algorithms, we here need to *estimate* the probabilities to obtain any outcome z. We do so by running the circuit on the same input x, a number of times R large enough so that by the law of

large numbers, $\frac{r_z}{R} := \hat{p}_z(x) \approx p_z(x)$ (where r_z is the number of times z was measured and $p_z(x)$ the exact probability that z is measured on input x).
Once we have strong estimates, we simply set the label y that maximizes $\sum_{z:f(z)=y} p_z(x)$. If we say $|\psi_x\rangle$ is the state just before the measurement, we say rewrite this quantity as:

$$\sum_{z:f(z)=y} \langle\psi_x|z\rangle\langle z|\psi_x\rangle = \langle\psi_x| \underbrace{\left(\sum_{z:f(z)=y} |z\rangle\langle z|\right)}_{M_y} |\psi_x\rangle = \langle M_y\rangle \tag{4}$$

We define M_y as the measurement operator associated to the label y. In a quantum computing perspective, our repeated measurements are used to estimate the *expected value* of each M_y, under state $|\psi_x\rangle$. In a more compact form, we thus define the labeling as $\hat{m}(x) = \arg\max_y \langle M_y\rangle$. Because we often omit to explicitly write the dependence on x in this bracket notation, we'll sometimes use a different notation to denote the exact same quantity, $p_y(x) := \langle M_y\rangle$. This can be read as *the probability to measure an output z such that $f(z) = y$, on state $|\psi_x\rangle$.*

Our task now, as in any supervised ML problem, is to find the optimal parameters θ^*, that is, the parameters that will lead to the best approximation $\hat{m}$ of m. The "best" approximation is said to be the one minimizing a chosen *cost* function L that will compare the predicted and true labels of the training set. Here, we choose the *empirical risk* and define it as:

$$R_{emp}(\theta, b) = \frac{1}{|T|}\sum_{x\in T} Pr(m(x) \neq \hat{m}(x)) \approx \frac{1}{|T|}\sum_{x\in T} \sigma\left(\frac{\sqrt{R}\left(\frac{1}{2} - \left(\hat{p}_y(x) - \frac{yb}{2}\right)\right)}{\sqrt{2(1-\hat{p}_y(x))\hat{p}_y(x)}}\right), \tag{5}$$

where σ is the sigmoid function, $\hat{p}_y(x) := \sum_{z:f(z)=y} \hat{p}_z(x)$, our approximation of $p_y(x)$, and b a trainable bias that stabilizes the training (not used in the testing phase).
At each step of the training phase, our optimizer (here `COBYLA`, from the `scipy.optimize` library) computes R_{emp} to predict the next best θ. It computes this risk by running the circuit to estimate $\hat{p}_y(x)$ for all $x \in T$.

To breakdown this *training phase*, at each step, we estimate all $\hat{p}_y(x)$ thanks to our quantum computer, parametrized by a vector $\theta^{(t)}$. Then, we classically compute R_{emp} and find the next suitable $\theta^{(t+1)}$, which will modify our quantum circuit, making it ready for the next step.

2. Ideal Low-Dimensional Model

We focus on the smallest non-trivial case, setting $n = N = 2$, where already a lot of properties are to be studied.

Of course, quantum processors are by construction very *noisy*. However, we will here assume an *ideal* machine, that has absolutely no noise. *Note that we only used classical simulators in this project.*

2.1. Model Setup

2.1.1. Feature maps & Circuits

To study this concrete case of our circuit, we still need to precise what kernel functions ϕ_S we use. Here, we define them as:

$$\underline{\phi_\emptyset(\mathbf{x}) = \mathbf{0}, \quad \phi_{\{\mathbf{i}\}} = \mathbf{x_i}, \quad \phi_{\{\mathbf{1,2}\}} = (\pi - \mathbf{x_1})(\pi - \mathbf{x_2})}$$

Furthermore, to see how the *depth* l affects the performance of the model, we will do the same experiences with $\underline{\mathbf{l} \in \{\mathbf{0}, ..., \mathbf{4}\}}$.
Our circuits then have the following form:

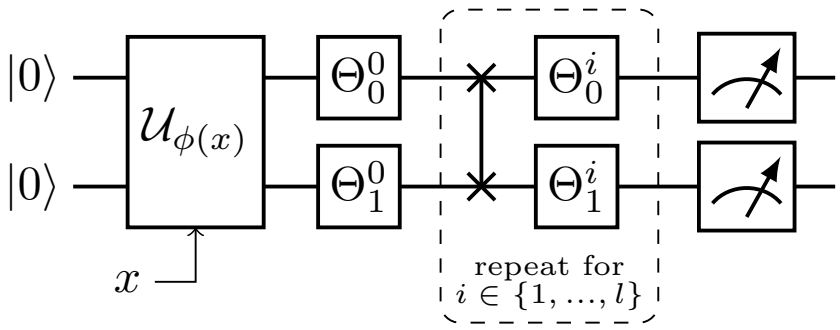

Figure 2: *Our circuit with $N = 2$ and varying depth l (the dashed box exists only for $l > 0$).*

2.1.2. Creating the data

To prove a path of *quantum advantage*, we will focus on a specific own-made case, we won't use real data. We rather create artificial data, in this way, we have more control showing when our model is efficient (a pertinent follow-up could thus be to use true data). To do so, for each dataset, we choose a random unitary V, a parity operator $\mathbf{f} = \sum_{z\in\{0,1\}^N} f(z)|z\rangle\langle z|$ and a gap Δ. We then compute $\delta = \langle\Phi(x)|V^\dagger \mathbf{f} V|\Phi(x)\rangle$, if $\delta \geq \Delta$ we assign $m(x) = 1$, else if $\delta \leq -\Delta$ we assign $m(x) = -1$, otherwise we don't assign any label ($x \notin \Omega$). For all datasets, we set $\underline{\boldsymbol{\Delta} = \mathbf{0.3},\ \mathbf{f} = \mathbf{Z}\otimes\mathbf{Z}}$ (implicitely setting $f(z) = z_1 z_2$).

Using a $\frac{2\pi}{100}$ granularity, the datasets will be extracted ($\underline{|\mathbf{T}| = \mathbf{70}, |\mathbf{S}| = \mathbf{30}}$, both *balanced*) from three Ω sets as shown in (3).

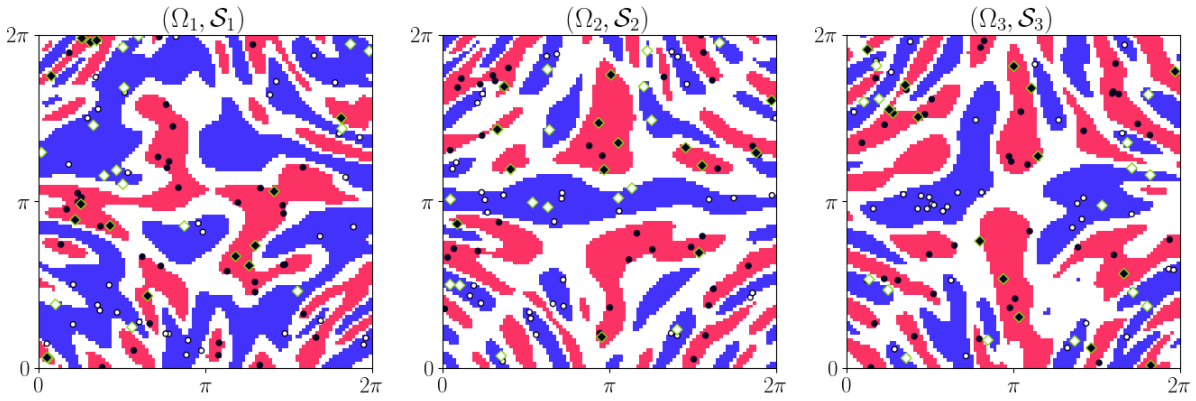

Figure 3: *Our three Ω sets and their corresponding dataset $\mathcal{S} = T \cup S$, with the binary positive (pink) and negative (blue) classes. The dots correspond to the training datasets T_i, the diamonds to the testing ones, S_i.*

2.2. Training Analysis

To study how the dataset choice impacts the performance of the algorithm, we independently run the training loop on each dataset. For each $\mathcal{S}_i$, we start with $\underline{\theta_\mathbf{0} = \mathbf{0}, \mathbf{b_0} = \mathbf{0}}$, we set $\underline{\mathbf{R} = \mathbf{2000}}$ and let the optimizer run on T_i for $\underline{\mathbf{200}}$ steps. Furthermore, we also evaluate the accuracy of the model at each $\underline{\mathbf{10}}$ iterations using S_i, to do so, we set $\underline{\mathbf{R_S} = \mathbf{10000}}$.

Finally, we shouldn't forget about the probabilistic nature of quantum computing. Due to this reason, we decide to run each experiment $\underline{\mathbf{3}}$ times, with exact same initial conditions.

2.2.1. Using different datasets

We focus on the accuracy results during the first execution of the training phase for each $\mathcal{S}_i$.

The first observation on (4) that we can make is that l highly affects the performance of our model. *However*, if we could expect that an increasing number of parameters would systematically improve the performance, it seems that this improvement is not always increasing with l. Moreover, it seems that the influence of l highly depends on $\mathcal{S}$. For instance, $l = 4$ has pretty decent scores (≈ 0.98) on $\mathcal{S}_2$ and but performs significantly worse (< 0.85) on $\mathcal{S}_1$ and $\mathcal{S}_3$. These differences are reflected during the evolution of the empirical risk, on (5).

We can explain the poor performance of deep ($l = 4$) circuits because of the higher number of parameters. On one hand, our dataset may be simply too small, on the other one, higher dimensionality may be a problem. In classical optimization, this artifact is present because a higher dimensionality results in a higher number of saddle points, which makes the the convergence very slow.
In the quantum case, a similar problem is encountered named *Barren Plateaus* (BP), as explained in [Larocca et al.], such a plateau happens because, in the quantum state space, the cost landscape is extremely flat, in other words, all non-zero gradient points are exponentially (in number of qubits, N) concentrated around very specific points. This problem applies to all optimizers, even gradient-free ones (such as ours, `COBYLA`), because finding the global minimum in a very flat cost surface is a difficult problem.

Even if our circuit uses a small number of qubits ($N = 2$), this hypothesis is not impossible, as we deal with relatively deep circuits ($l > \log N$). This may be a reason why BPs could happen, as suggested in the paper. The next section will exhibit another argument supporting this scenario.

On a side node, the approximately constant performance of $l = 0$ ($\in [0.6, 0.78]$) *suggests that the entangling* CZ *gates in* $W(\theta)$ *ansatz are crucial to the circuits' efficiency.*

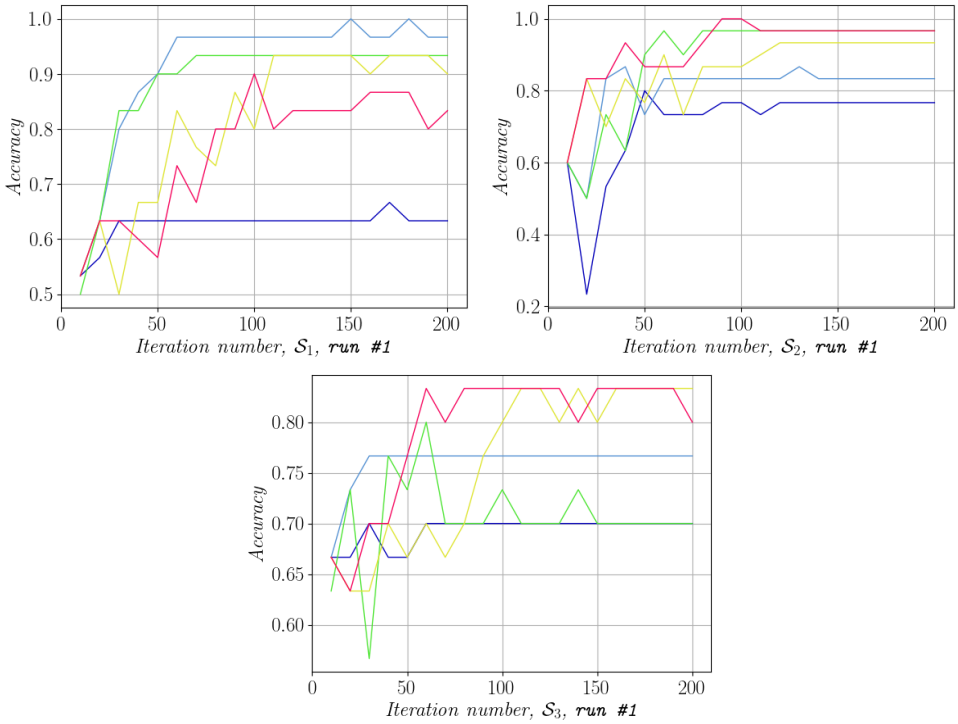

Figure 4: *Evolution of the accuracy for each $\mathcal{S}_i$. Measurements made during the training phase on the corresponding testing dataset S_i. Each curve corresponds to a specific circuit depth $l \in \{0 \equiv$ dark blue, $1 \equiv$ light blue, $2 \equiv$ green, $3 \equiv$ yellow, $4 \equiv$ pink$\}$.*

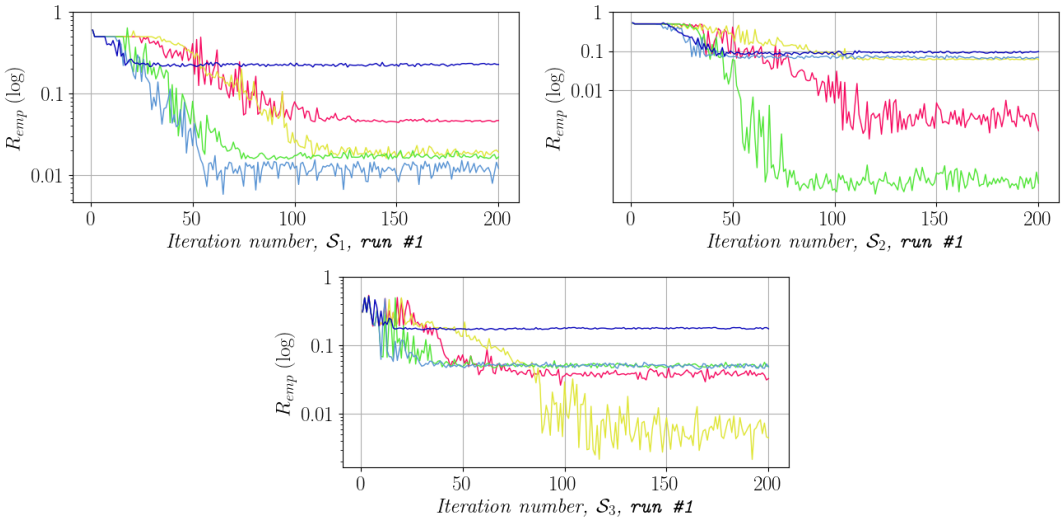

Figure 5: *Evolution of the empiral risk R_{emp} during the training phase on different datasets T_i, in log scale. Each curve corresponds to a specific circuit depth $l \in \{0 \equiv$ dark blue, $1 \equiv$ light blue, $2 \equiv$ green, $3 \equiv$ yellow, $4 \equiv$ pink$\}$.*

2.2.2. Using the same dataset

As mentioned earlier, we ran the exact same experiments three times, to explore how the quantum randomness can affect our results. In the above section, we saw that the deepest circuit ($l = 4$) performs poorly on $\mathcal{S}_1$ and $\mathcal{S}_3$. We supposed this was due to the presence of hypothetical BPs, the training plots on (6) of the other experiments on $\mathcal{S}_1$ may confirm our suppositions.

Let us focus on the pink ($l = 4$) curve. Indeed, while in the first run, the deepest circuit converges to a relatively high R_{emp} (≈ 0.05), the other runs achieve way smaller values for the same circuit ($\approx 10^{-4}$). This suggests that during the first run, the optimizer could not escape the BP for the pink circuit, while in the other ones, a non-zero gradient direction has been found. The second run is very interesting for that, as during the first ≈ 125 steps, the pink curve was stuck on the same plateau, but as soon as it found a "good" direction, R_{emp} decreased exponentially fast. *The same situation happens with the yellow curve, with it being stuck in the first two runs and finding a non-zero gradient direction in the third one.*

These differences are slightly reflected on the accuracies of (7) measured during these runs.

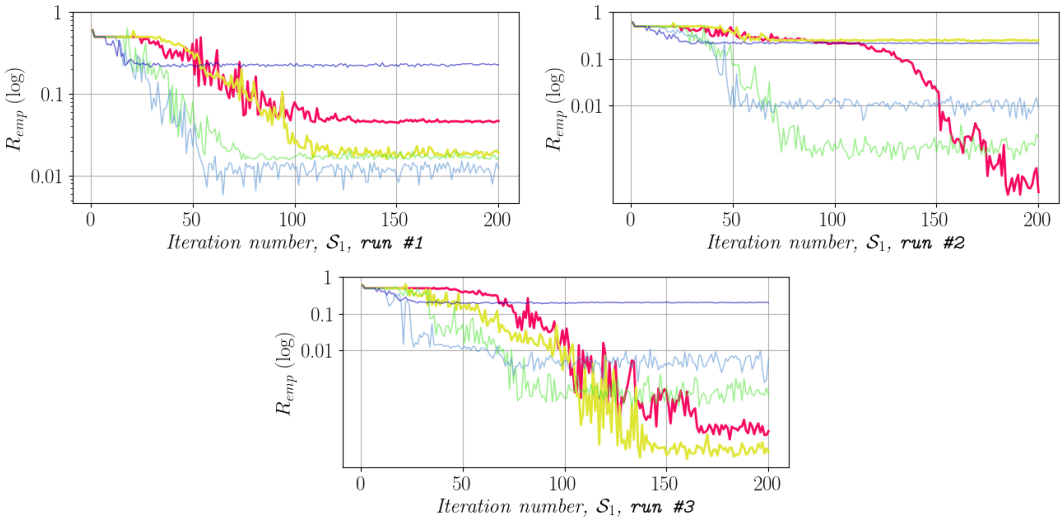

Figure 6: *Evolution of the empiral risk R_{emp} during three training phases on the same parameters and dataset, T_1, in log scale. Emphasis on the two deepest circuits, $l = 3$ (yellow), and $l = 4$ (pink).*

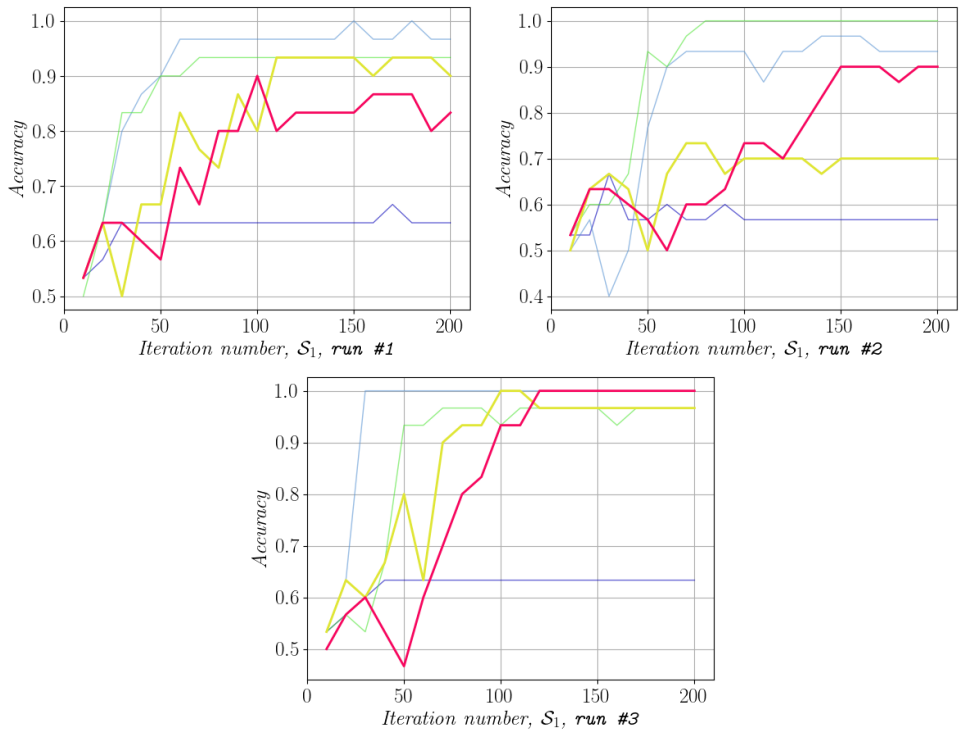

Figure 7: *Evolution of the accuracy during the three runs on $\mathcal{S}_1$. Measurements made on S_1. Focus on the two deepest circuits, $l = 3$ (yellow), and $l = 4$ (pink).*

Even though the pink accuracy on the second run isn't the best, it surpasses (along with the third one) the first run. This property of having more "successful" runs than others is found again in training executions (8) of the other datasets.

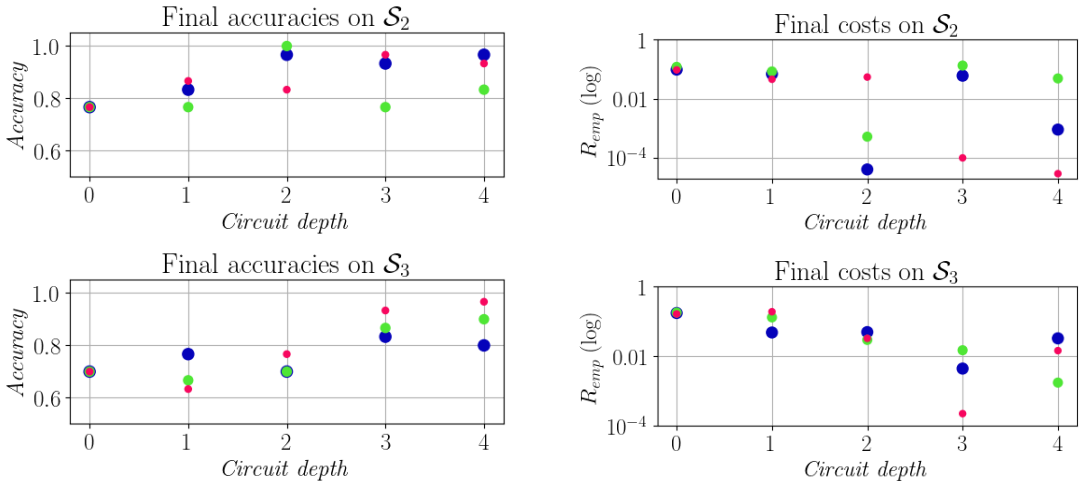

Figure 8: *Final accuracies and costs (R_{emp}) after the training phases on $\mathcal{S}_2$ (top) and $\mathcal{S}_3$ (bottom). Each dot color corresponds to an individual and identical run. On each graph, at least one run is "successful", meaning that the accuracy increases with the circuit depth and that R_{emp} decreases with it.*

2.2.3. More on Barren Plateaus

We saw many arguments suggesting that the high fluctuations may be caused by a famous phenomenon in QML: *Barren plateaus*. Even though other parameters also have their part of causality, such as the dataset choice or T's size, the very flat landscapes of the cost function are known as a common issue in this field.

To mitigate the effect of BPs, instead of always starting at the same point ($\theta_0 = 0, b_0 = 0$) and hope randomness will lead to one of the "pitfalls" of the cost surface, we can start with a random set of weights.
Our last experiment is to run the training phases on each $\mathcal{S}_i$ in the exact same way, however, this time we sample uniformly θ_0 in $\{-\frac{\pi}{2}, \frac{\pi}{2}\}^{2Nl}$ at each execution.

As we can see in (9) and (10), with this new initialization of θ_0, our results are more stable and fits more our expectations. Indeed, the random starting point allows the optimizer to spend less time on saddle points. For that reason, we will use this technique for our next experiments.

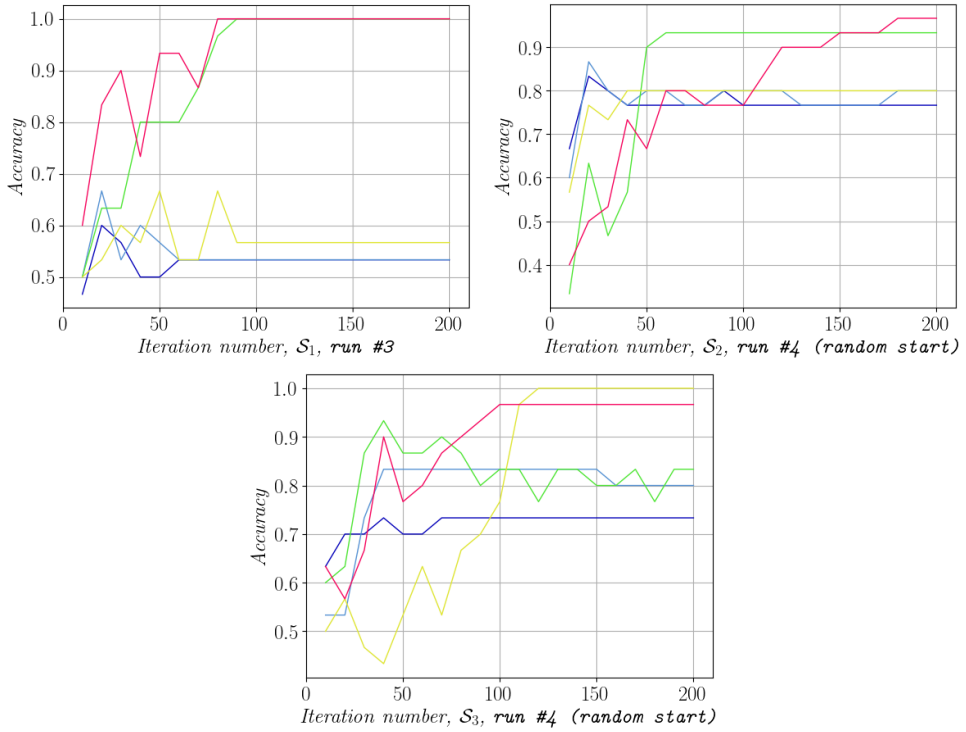

Figure 9: *Evolution of the accuracy for each $\mathcal{S}_i$, with random initial weights. Measurements made during the training phase on the corresponding testing dataset S_i. Each curve corresponds to a specific circuit depth $l \in \{0 \equiv$ dark blue, $1 \equiv$ light blue, $2 \equiv$ green, $3 \equiv$ yellow, $4 \equiv$ pink$\}$.*

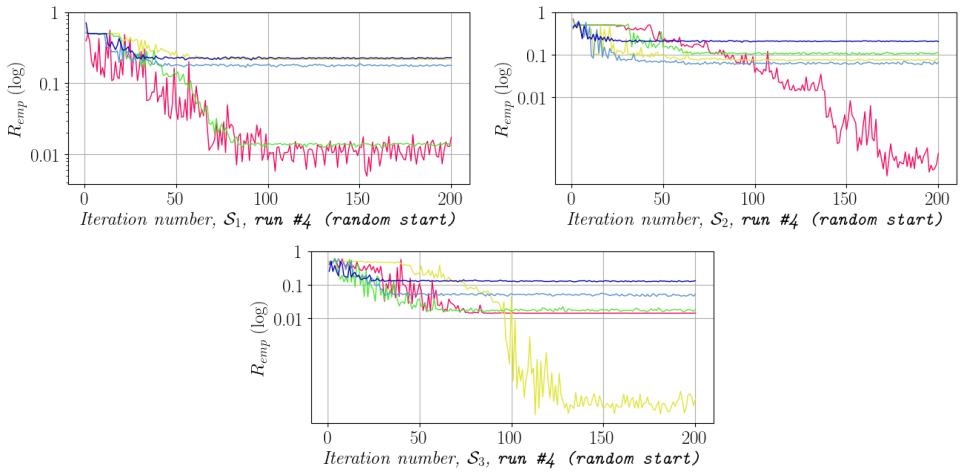

Figure 10: *Evolution of the empiral risk R_{emp} during the training phase on different datasets T_i, in log scale. Initial weights are set randomly. Each curve corresponds to a specific circuit depth $l \in \{0 \equiv$ dark blue, $1 \equiv$ light blue, $2 \equiv$ green, $3 \equiv$ yellow, $4 \equiv$ pink$\}$.*

3. Noisy Model

So far, we worked under the assumption of an ideal *non-noisy* model. However, noise is an inherent property of quantum hardware and more generally, of quantum physics, due to various physical laws applying to quantum objects.

For this reason, it is crucial to study the efficiency of our model under more realistic conditions. In this section, we will see how noise impacts results and explore different solutions to mitigate its effects.

3.1. Our Noise Model: The Depolarizing Channel

In this project, we'll stick to the simplest noise model possible, the so-called *depolarizing noise*. This model supposes a depolarizing *channel* is applied after some (or all) gates in the quantum circuit.

Until now, we only worked with *pure states*, quantum states that can be written as vectors $|\psi\rangle$. However, noisy setups require a more general mathematical tool, *density matrices* ρ. These matrices represent a probabilistic aspect of the quantum state, allowing the physical system state to be probabistic (a crucial point is that the uncertainty of the physical state and the uncertainty of the measure outcomes are two independent concepts, which is the reason why we introduce a new mathematical tool). *Every pure state* $|\psi\rangle$ *can be represented by the matrix* $\rho = |\psi\rangle\langle\psi|$.

Coming back to our depolarizing channel, instead of simply applying the gate U to our state ρ as $U : \rho \mapsto U\rho U^\dagger$, there is a probability p that the state collapses to the maximally mixed state. We define this channel $\mathcal{E}_U$ mathematically as:

$$\mathcal{E}_U : \rho \mapsto (1-p)U\rho U^\dagger + \frac{p}{2^N}\mathbb{I}, \tag{6}$$

where $\mathbb{I}$ is the identity on N qubits, furthermore, we'll now refer to p as the *noise rate*.

3.2. How Noise Affects Estimations

Now that we defined our noise model, let's come back to our quantum circuit.
In quantum hardware, the more qubits are involved during an operation, the more noisy the result will be. That is why we decide to apply a p-depolarization channel *only* at the **CZ** gates present in $W(\theta)$.

Now, to label inputs and to learn the best parameters θ, the core concept is the estimation of $\langle M_y \rangle$. However, this estimation is affected by a bias introduced by noise. The final state before the measurement for an arbitray l is:

$$\rho_{final,l} = q^l W(\theta)\rho_{\Phi(x)}W^\dagger(\theta) + (1-q^l)\frac{1}{2^N}\mathbb{I}, \tag{7}$$

where $\rho_{\Phi(x)} = \mathcal{U}_{\Phi(x)}|0^N\rangle\langle 0^N|\mathcal{U}^\dagger_{\Phi(x)}$ and $q = (1-p)$. We see that the more layers or noise we add, the closer the final state is to the maximally mixed state. This is a problem, as this induces the probabilities $p_y(x) := \langle M_y \rangle$ to be less distant, making the classes very difficult to separate. Indeed, the expectation of M_y is given by:

$$\langle M_y \rangle = \mathrm{Tr}(\rho_{final,l}M_y) = q^l \underbrace{\mathrm{Tr}(W(\theta)\rho_{\Phi(x)}W^\dagger(\theta)M_y)}_{\langle M_y \rangle_{p=0}} + \frac{(1-q^l)}{2}, \tag{8}$$

assuming $\mathrm{Tr}(M_y) = 2^{\frac{N}{2}}$ (which is true in our case) and with $\langle M_y \rangle_{p=0}$ the expected value of M_y in the noiseless case. Because the optimizer can't recognize between real data and noise, with an increasing value of p, both classes have more and more similar chances to be chosen. In other words, $\lim_{p\to 1}\langle M_y \rangle = \frac{1}{2}$, which doesn't give us any information on how to classify points. This behavior is reflected on the accuracies (11) of our model, when increasing p.

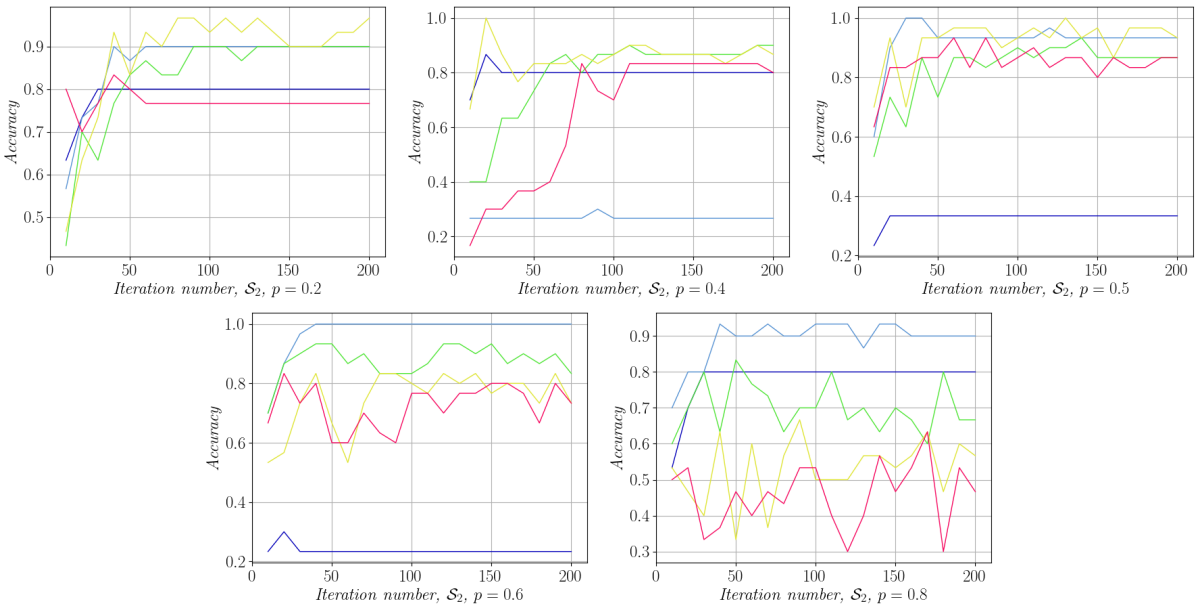

Figure 11: *Same experiment setup as in the noiseless case, with uniformly sampled θ_0. Measurements of accuracies on S_2 during the training phase on T_2, with increasing noise rate p. Each curve corresponds to a specific circuit depth $l \in \{0 \equiv$ dark blue, $1 \equiv$ light blue, $2 \equiv$ green, $3 \equiv$ yellow, $4 \equiv$ pink$\}$*

We exactly see the expected behavior, with the deeper circuits mostly performing worse with noise. Most importantly, we notice how once the threshold $p = 0.5$ passed, the accuracies reach terrible scores, with very high variances. This behavior indicates that the models learn to "randomly" label points, without any pattern, resulting in same (even for some, worse) results than the ones we would have assigning a label to x by flipping a coin.
However, it seems that even with noise, some models still achieve decent scores. This could give us a hint that there is hope and thus that there may exist some solutions to mitigate these effects, which we will explore in the next sections.

3.3. Trivial Noise Removing

Since we know the exact mathematical formulation on how our noise affects $\langle M_y \rangle$, the most straightforward solution would be to simply "reverse" the formula to get the noiseless value from the noisy one. Starting from (Eq. 8), we get:

$$\langle M_y \rangle_{p=0} = \frac{1}{q^l} \left(\langle M_y \rangle - \frac{(1 - q^l)}{2} \right) \tag{9}$$

Indeed, this formula is exact and solves perfectly any noise effect. However, this requires a lot of unrealistic assumptions:

1. Perfect Knowledge: In our case, we set our noise artificially, so we are by construction aware of the exact mathematical formulation of the model. However, in most of the cases, it is very difficult to precisely know how the model behaves. In addition, even having an approximation can induce a very complex formula, resulting in possible high inaccuracies on the "recovered" value.

2. Constant Behavior: Now, assuming we perfectly can model the noise, we further need to be certain that the model will *exactly* behave in such way, at any time. As soon as the model slightly changes, it could be because of any condition (e.g external noise, hardware temperature), our recovered technique becomes inaccurate and obsolete.

3. Extreme Precision: Finally, assuming the two first conditions are met, a last problem arises. Some mathematical operations are *very* sensitive to small value changes. For example in our case, the factor $\frac{1}{q^l}$ can become very large quickly with l. This means that it suffices to have a small inaccuracy δ in our estimation, $\widehat{\langle M_y \rangle} = \langle M_y \rangle + \delta$, for the recovered value to be wrong, $\widehat{\langle M_y \rangle}^{(recovered)}_{p=0} = \langle M_y \rangle_{p=0} + \frac{\delta}{q^l}$. Remembering that $\langle M_y \rangle \in [0, 1]$, even some changes can bias the true value, we thus need very high estimation precision. However, to estimate $\langle M_y \rangle$ with a precision ϵ, we need $O(\frac{1}{\epsilon})$ shots, which could become pretty expensive.

3.4. Zero Noise Extrapolation (ZNE)

We saw that removing noise directly from the formula is an unrealistic technique, however, there are more sophisticated ways to approximate non-noisy values. In this section, we'll focus on the so-called *zero noise extrapolation* (ZNE) [Giurgica-Tiron et al.], that exploits even noiser results to "guess" $\langle M_y \rangle_{p=0}$.

Assume that we want to measure the expected value E of any observable (M_y in our case). Furthermore, the model may contain noise, which can be characterized by a positive quantity λ, $\lambda = 0$ being the noiseless case and $\lambda = 1$ the natural noise of the model. As we saw earlier, this can directly affect measurements, making our value dependent on λ, we write it as $E(\lambda)$.

Now, we would like to estimate the noiseless expectation $E(0)$, with the constraint that we only have access to $\lambda \geq 1$. Our technique requires to collect values of $\hat{E}(\lambda) \approx E(\lambda)$ for m different values of $\lambda \geq 1$, which is possible via *noise scaling*. There are lots of efficient techniques to artificially increase the noise, such as *circuit* or *unitary* folding (which we will use), that don't even require having access to hardware. *In our case, we could manually increase p to scale λ. However, for more realism, we pretend we don't have access to noise parameters and only to the model definition.*

Once we have a set of precise enough estimations $\{\hat{E}(\lambda_i)\}_i$, we use it to extrapolate and estimate $E(0)$.

3.4.1. Noise Scaling: Unitary Folding

A simple way to increase noise, is simply to add noisy unitaries to the circuit. We make sure that both the extended and original circuits are logically equivalent by *canceling* the added gate U by overall using $U^\dagger U$, we call this method *unitary folding*.

In our case, it is the CZ gate that induces noise in the result. Let our model use l CZ gates, we can thus define the noise quantity λ as a function of k, the number of CZ gates in the circuit. We define $\lambda(k) = \frac{k}{l}$. So, to increase λ, it suffices to add pairs of CZ to the circuit (as $CZ^\dagger = CZ$). We have, using (Eq. 8):

$$E(\lambda) = q^{\lambda l} E(0) + \frac{1}{2}(1 - q^{\lambda l}) = Q^\lambda \left(E(0) - \frac{1}{2} \right) + \frac{1}{2}, \tag{10}$$

where we define $Q := q^l$. As shown in (12), our estimations approach pretty well $E(\lambda)$, an exponential-like curve.

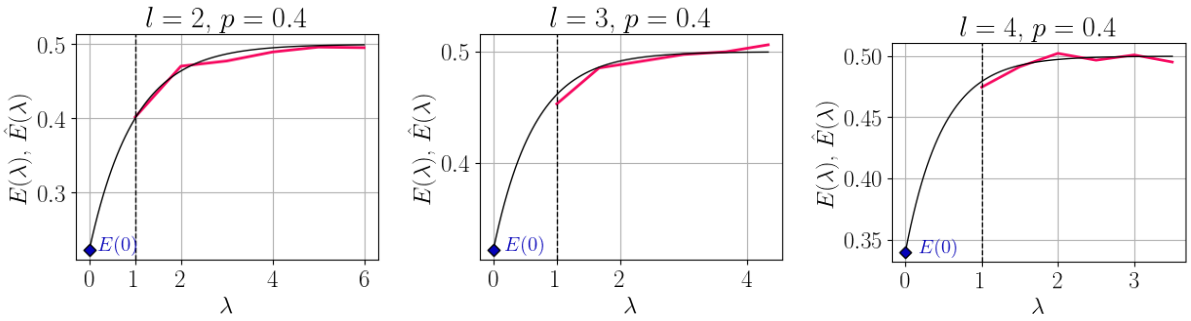

Figure 12: *Estimation of $\langle M_+ \rangle$ on input $x = (\pi, \pi)$, using the final weights after noiseless training on $\mathcal{S}_2$. The black curve is the function of (Eq. 10), the pink one is the estimation $\hat{E}(\theta)$ of $\langle M_+ \rangle$, using* 10000 *shots and a noise rate $p = 0.4$. The point we want to estimate is marked in blue. Note that as shown in (3), $m(x) = -1$.*

Let's also note that the weights we use for the plot are not relevant, as in the estimation process, the only task is to estimate $\langle M_y \rangle$, no matter the value of $W(\theta)$.

The challenge will now be to use these points $\{\hat{E}(\lambda_i)\}_i$ to extrapolate $E(0)$, using efficient approximations of $E(\lambda)$'s shape.
Now, we notice that when l increases, the accessible region ($\lambda \geq 1$) of $E(\lambda)$ becomes flatter. This behavior makes it even more challenging. It is however expected, as larger l increases noise, inducing a stronger bias to our original circuit ($\lambda = 1$), $\lim_{l \to \infty} E(1) = \frac{1}{2}$.

3.4.2. Exponential Extrapolation

As suggested in Eq. 10, noise rate $1 - p$ and layer depth l can be combined to a single quantity Q, not dependent on λ. With the final equation form, we clearly see that E can be approximated by a function of the form:

$$\tilde{E}(\lambda) = ae^{-b\lambda} + \frac{1}{2} \tag{11}$$

To find parameters a and b, we use the `scipy.optimize` built-in `curve_fit` method, that uses the *least squares* technique. Furthermore, to avoid under or overfitting, we choose to always use the three same λ coordinates, $\lambda_{i \in \{1,2,3\}} = 1 + \frac{2(i-1)}{l}$, as support points to the curve fitting. This corresponds to adding $k_i = 2(i-1)$ CZ gates.

Once we found the best fitting parameters a, b, we simply compute $\tilde{E}(0) = a + \frac{1}{2}$.

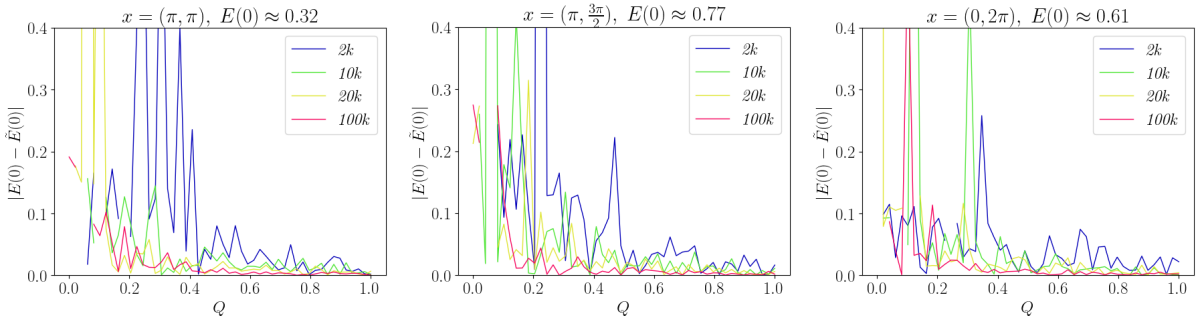

Figure 13: *Estimation of* $E(0) := \langle M_+ \rangle_{p=0}$ *on input different inputs* x*, with ZNE, using the final weights after noiseless training on* $\mathcal{S}_2$*. We plot the absolute difference between our estimate and the true value. Each curve corresponds to a different number of shots* R *to estimate* $E(\lambda)$*.*

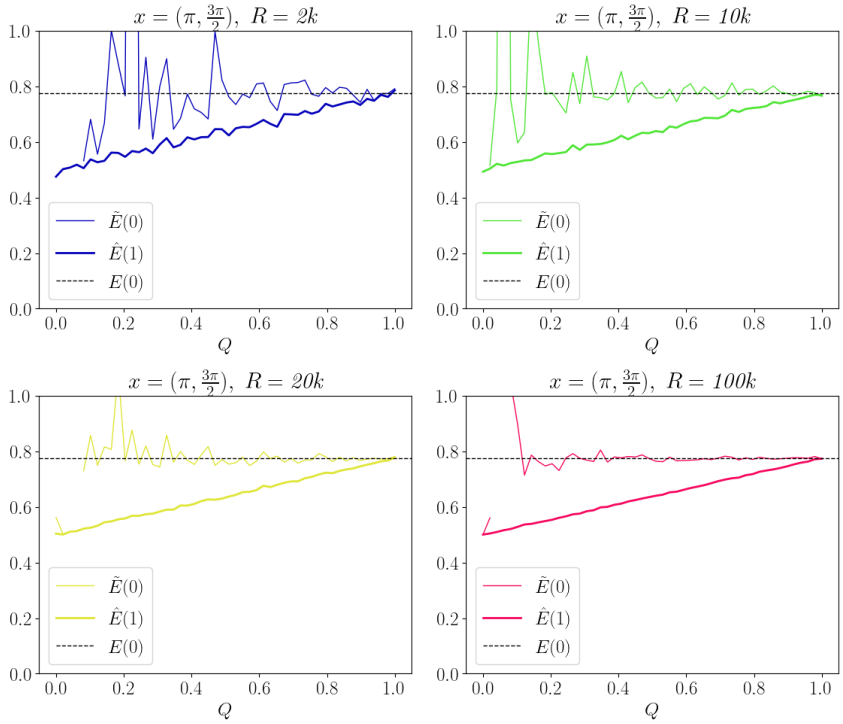

Figure 14: *Comparison of* $\hat{E}(1) := \widehat{\langle M_+ \rangle}$ *(the value used for training when there is no ZNE) with* $E(0)$ *and its ZNE-estimate* $\tilde{E}(0)$*. Each graph corresponds to a different* R*.* $E(0) \approx 0.77$*, with the same setup as in (13).*

In the two figures (13) and (14), the efficiency of our ZNE technique is proven. As expected, a higher R leads to a better estimate of $E(\lambda)$ and to a more accurate $\tilde{E}(0)$. Furthermore, a lower Q means stronger noise, which explains why the estimations become less accurate and have high variance when Q tends to zero.
This suggests that depending on R, our technique can be very efficient for Q values bigger than a threshold Q_R. E.g here we could manually define some Q_R just by looking at our plots,

$$Q_{2k} \approx 0.8,\ Q_{10k} \approx 0.5,\ Q_{20k} \approx 0.4,\ Q_{100k} \approx 0.25$$

Of course, Q is directly dependent on l, which means that for a same R, our ZNE technique could make our model robust to higher noise rates on lower l. We can thus find thresholds p_l, $l > 0$ such that our model becomes robust to depolarizing noise rates $p < p_l$. These values can be defined as:

$$p_{l,R} = 1 - (Q_R)^{1/l} \tag{12}$$

Which implies that the more layers we add, the less resistant to noise the ZNE model will be, which is expected. We fall back to one of the initial problems, to fight noise, we need to increase R, however, here we did it only by "guessing" the shape of $E(\lambda)$. Furthermore, we see that with only a simple trick such as ZNE, our model can become very resistant to relatively decent values of p. Let's see how our results are impacted by this technique in the next section.

3.4.3. Comparing Results

Now that we have proven the efficiency of ZNE on $\langle M_y \rangle_{p=0}$ estimation, we'd like to apply it to our initial noisy classification problem.

In (15), we can see what benefits and flaws our ZNE method has. At first, without ZNE, all of our circuits (except $l = 1$) has a similar score $\in [0.8, 0.9]$ which is not bad at all for $p = 4$. Then, we use a relatively inaccurate ZNE, i.e with a low $R = 2000$, clearly, it had a very negative impact on the overall performances, except perhaps slightly improving $l = 2$'s scores. Finally, we use a more precise ZNE, with $R = 10000$, where the scores are already better and similar to the no ZNE case. We notice however a very different behavior: the red and blue curves seem to be switched in performance. Using Eq. 12, we compute $p_{1,10k} = 0.5$ and $p_{4,10k} = 0.16$, leading to $p_{1,10k} > p \gg p_{4,10k}$. This can be interpreted as $l = 1$ model being very robust to our noise choice, while $l = 4$ induces a very unstable $Q \approx 0.13$ as shown in (14). This interpretation explains why the other curves still achieve good scores, as $l \leq 3$ imply $Q \geq 0.22$, which are way more stable in the same graph.

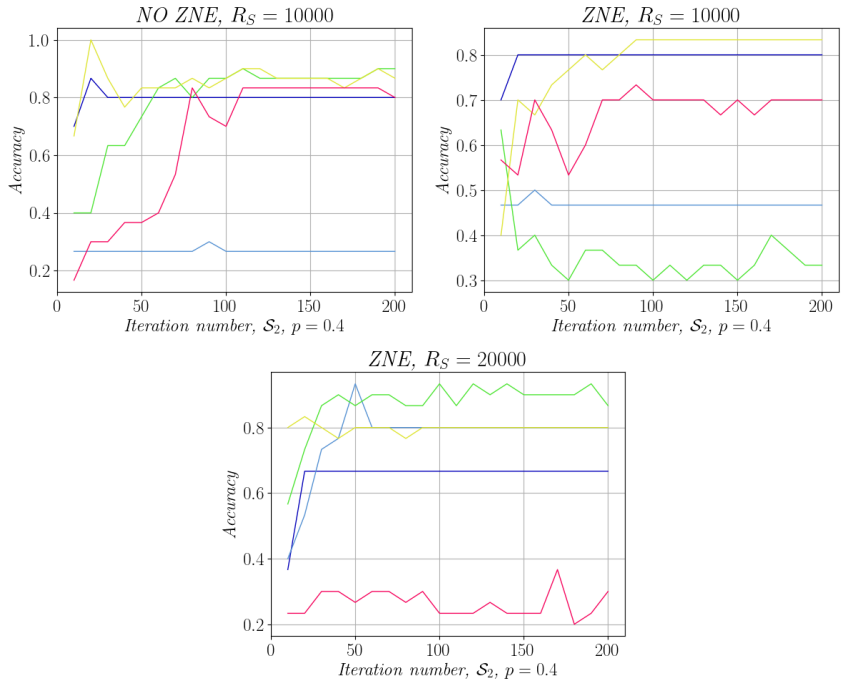

Figure 15: *Plots of the accuracies during the training loops with or without ZNE on* $\mathcal{S}_2$*. The noise rate* p *is set to* 0.4*,* $R = 2000, R_S = 10000$ *for the first row and* $R = 10000, R_S = 20000$ *for the second one. Each curve corresponds to a specific circuit depth* $l \in \{0 \equiv$ *dark blue*$, 1 \equiv$ *light blue*$, 2 \equiv$ *green*$, 3 \equiv$ *yellow*$, 4 \equiv$ *pink*$\}$*.*

As a conclusion to our technique, we can deduce that ZNE is efficient for Q values larger than some threshold Q_R. However, for values smaller than Q_R, ZNE's estimations become rapidly very inaccurate, leading to scores even worse than the no ZNE case. This suggests that this extrapolation method can be very useful, but only if it is used correctly, i.e using the appropriate number of shots R depending on how much noise p is present or how much layers l we'd like to use. That is, if we are limited in R, using ZNE can lead to poor results, making it a bad design choice for $Q < Q_R$.

3.5. Other solutions

In this project, we focused on *removing* noise to mitigate its effects. Other techniques exist, as presented in [Temme et al.] with *probabilistic error cancellation*.

Moreover, there are other directions to mitigate the effects of noise, more adapted to our global problem (not only approximating $\langle M_y \rangle_{p=0}$). Such solutions could be using a *noise-resistant* cost function, or even more sophisticated techniques, such as designing noise-aware Ansatz $W(\theta)$ for our specific task, such as in [Kandal et al.] or developping more noise-resistant ways to measure our quantum states as in [Sauvage et al.].

Conclusion & Further Directions

Throughout the project, we presented the paradigm of *Quantum Machine Learning* and showed its efficiency for simple tasks, such as *binary classifications* in low dimensional input spaces.

The strength of this idea resides in the belief that there are circuits, in our case *kernel functions*, that no classical computer could ever simulate and that furthermore, these families of circuits achieve impressive results for specific problem instances. We guided a direction for an eventual *quantum advantage*: artificially creating our dataset. A natural continuation would be to test such models on real data, that is believed to be difficult to classically classify.
Furthermore, as we saw, the model suggests that it works pretty well for small n and N, however, the goal of this idea is to use algorithms that no computer could simulate *(which is, quite ironic, as we only used classical simulators in the project).* In practice, this would mean extending the concept to higher input spaces and thus to exponentially higher feature spaces. Studying such cases with sufficiently many qubits $N \approx 100$ would be enough to strongly prove an eventual quantum advantage.

Increasing the number of dimensions would also increase the chances to have to deal with *Barren Plateaus*. We supposed that this phenomenon may be responsible for some complications that we encountered during our training loops. We used a quick solution, *random initialization*, that seemed to stabilize our results. However, this method may be inadequate for higher N, suggesting that it could be interesting to explore more sophisticated training techniques, with different cost functions, optimization techniques, or even more adapted frameworks.

From a different point of view, we also saw how noise is crucial and inherent to quantum computation: not considering it would be unrealistic. We studied a very simple model, the *depolarizing channel*, one possible continuation would be working with more complex and realistic noise models that we could find on real quantum hardware.
Moreover, many solutions exist to mitigate noise, this topic is subject to very active research, as it is one of the biggest obstacles in the domain of quantum computation real applications. In this project, we focused on simple mitigations, one of them is the so-called *zero noise extrapolation*. We explored how this technique could use the noise impact to reduce its effects on our estimations, and how efficient it is or not, depending on our problem setup. A very interesting and pertinent continuation would thus be to study other and perhaps more adapted solutions and directions to reduce noise effects.