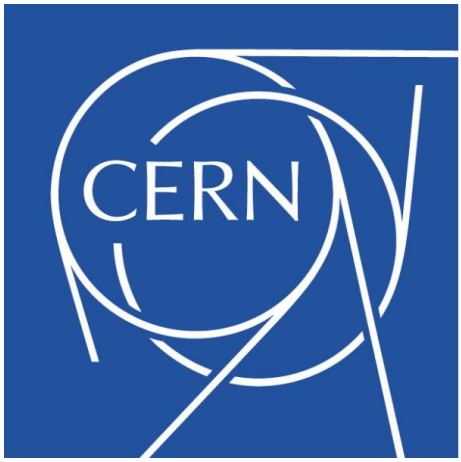


Towards advanced data encoding techniques for quantum re-uploading models

Supervisor:
Prof. Stefano Carrazza

Student:
Niccolò Laurora

Co-Supervisor:
PhD Alessandro Candido
Matteo Robbiati

Department of Theoretical Physics

Introduction

A new interdisciplinary research topic that goes by the name of Quantum Machine Learning (QML) has recently begun to explore the interplay of ideas from quantum computing and machine learning.
For example, QML investigates whether quantum computers can speed up the time it takes to train or evaluate a machine learning model. On the other hand, the QML community leverages techniques from machine learning to devise new quantum error-correcting codes, estimate the properties of quantum systems, or develop new quantum algorithms.
The state of the art of Quantum Machine Learning has as main highlight the development of algorithms that have been proven to have a quantum advantage in computational complexity over classical algorithms.
However, these algorithms require a fault-tolerant quantum computer, which is able to continuously correct errors that arise during computation, ensuring the stability and reliability of the quantum processes over extended periods. As we are still many years away from fault tolerant quantum computation, the QML community has developed a great interest toward possible applications of current and near-term quantum devices (NISQ devices), which are not capable of continuous quantum error correction. In particular, the QML research community has developed a new class of quantum procedures called variational quantum algorithms (VQA) to take advantage of current and near-term quantum hardware (figure 1).

A VQA is a hybrid quantum-classical algorithm that employs a Parametrized Quantum Circuit (PQC) $U(\theta)$, where θ represents a set of tunable parameters. These parameters are adjusted to enhance the algorithm's performance for completing a specific task. By measuring the PQC, valuable information is extracted and used to evaluate a loss function. This loss function is then minimized, and the optimization procedure suggests improved candidates for the parameters θ, starting from random or pre-trained initial values. The crucial point to emphasize of a VQA is that the PQC can be executed on current quantum devices, while the computational demanding task of optimizing the loss function is performed by a classical computer, which is far more reliable then NISQ devices. Many different VQAs have been designed to tackle standard machine learning tasks, such as classification, regression or optimization.
The main objective of my CERN' project was to develop a novel image classification PQC, which we named *block re-uploading*, and to investigate its properties (figure 2).
Specifically, we examined three key properties of the *block re-uploading* PQC (which are fundamental to all PQCs): *generalization*[1], *expressivity*[2] and *trainability*[3].

[1] Generalization refers to a model's ability to perform well on unseen data, demonstrating its effectiveness beyond the training set.

[2] Expressivity measures a PQC's ability to adapt and perform across various types of datasets.

[3] Trainability assesses how easy or difficult it is to train a PQC.

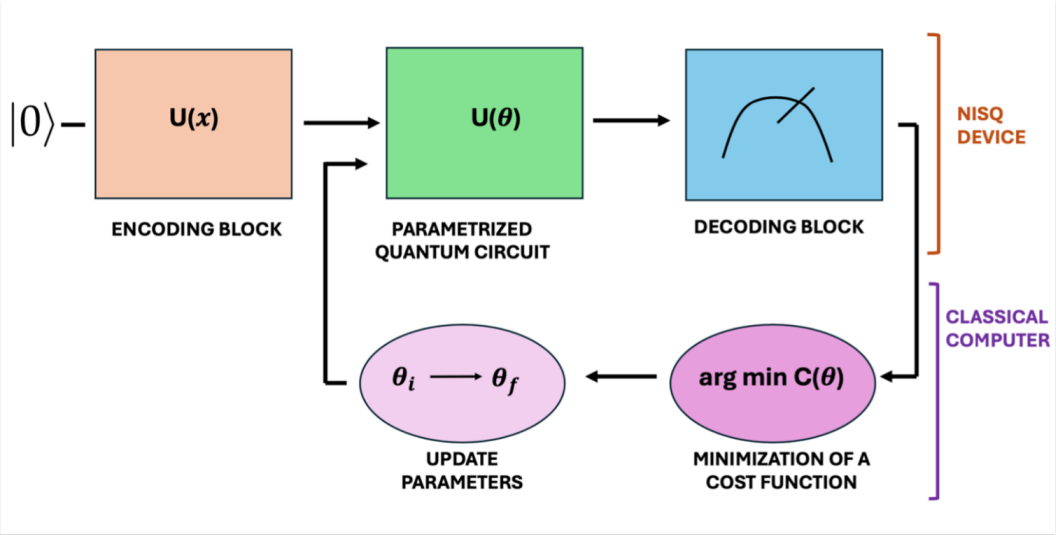


Figure 1: Schematic representation of a VQA. A VQA has different components: the first one is the encoding block, which is responsible for encoding the data of a certain problem in the circuit; the second one is the PQC, which has parameters that will be tuned to improve the performance of the algorithm; the third one is the decoding block, which consists of measuring an observable to extract information from the circuit that is used to evaluate a cost function. The PQC is executed on a NISQ device, while the minimization procedure of the cost function (which leads to the update of the parameters) is performed by a classical computer

Block re-uploading architecture

This *block re-uploading* is a new quantum machine learning algorithm inspired by classical convolutional neural networks and its main goal is to classify images. Images are composed of pixels, and each pixel is typically represented as a vector of three values corresponding to the intensity of the primary colors: red, green, and blue (RGB). Each of these values typically ranges from 0 to 255 in an 8-bit color depth image, which is common in digital images.
It is essential to observe that not all the information contained in an image is essential for classification purposes. Indeed, many preprocessing techniques, such as PCA, have been used in the literature to reduce the amount of information fed into a (quantum) neural network.
to preserve information redundancy in the images and maintain the full dimensionality of the dataset.

The fundamental idea behind the *block re-uploading* algorithm is based on the observation that neighboring pixels in an image are highly correlated. Consequently, we chose to divide each image into blocks and upload each block onto a separate qubit (figure 2).

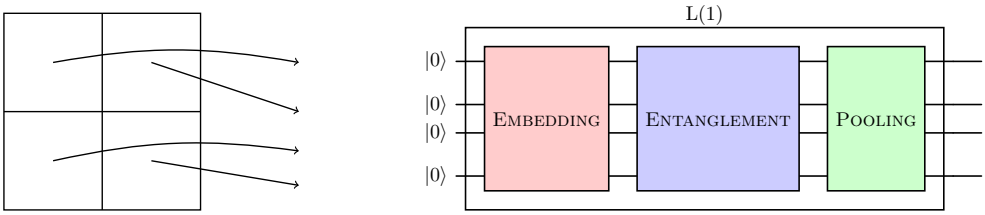


Figure 2: An image with even width and height can be divided into exactly four equal blocks. Each block will then be encoded onto a separate qubit. As we will explain in the next paragraph the *block re-uploading* architecture has three components per layer: embedding circuit, entanglement circuit, pooling circuit.

Our main goal was to investigate the *Depth vs Width trade-off*. The *Depth vs Width trade-off* can be more explicitly referred to as the *Sequential vs Parallel uploading trade-off*. If we do not split the image into blocks, the entire image will be encoded into a single qubit, resulting in sequential distribution of information across the quantum circuit. In contrast, if we divide the image into blocks and encode each block onto a separate qubit, the information will be distributed both sequentially and in parallel. As the number of blocks increases, the distribution of information becomes progressively more parallel, reaching the limit where each block consists of a single pixel. This trade-off raises the natural questions: *which distribution is better? Sequential or parallel?*. As is common when evaluating trade-offs, the optimal solution often lies between the two extremes. Therefore, we predict that the best approach to distributing information across the quantum circuit will involve a balanced mix of sequential and parallel encoding.

The *block re-uploading* algorithm is a layered algorithm and each layer has three components: an embedding circuit, an entanglement structure and a pooling circuit (figure 4).

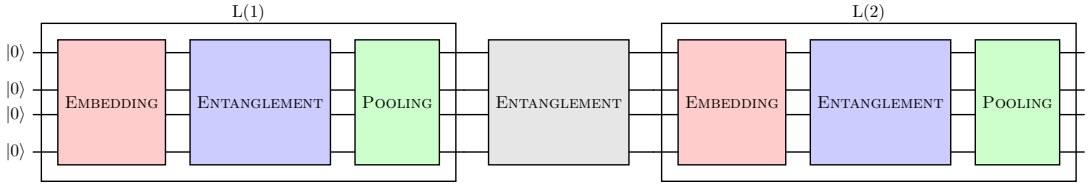


Figure 4: A four qubits and two layers *block re-uploading* architecture. Each layer has three components: an embedding circuit, an entanglement structure and a pooling circuit. Layers are separated by another entanglement structure.

Embedding The embedding component is responsible to encode each block of an image onto a different qubit of the quantum circuit (figure 5).
For instance, an 8×8 image can be divided in 4 identical 4×4 blocks. Therefore, each

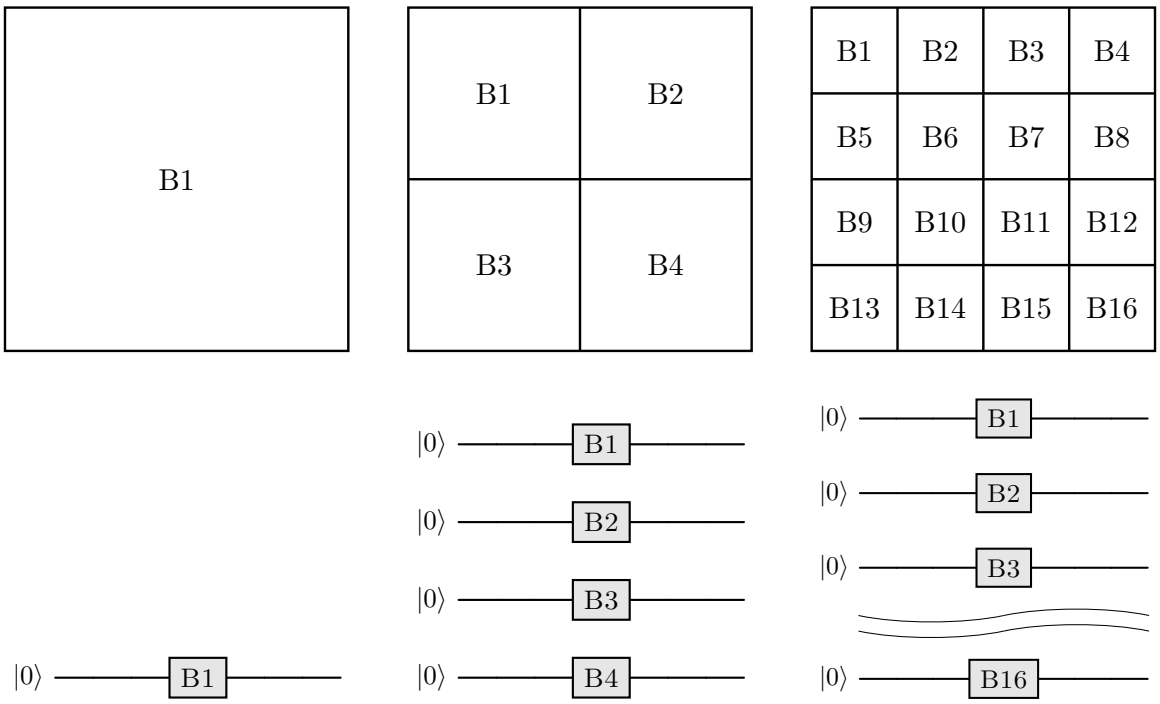


Figure 3: This figure shows three possible splitting of a 4×4 image and the corresponding circuit. If the image is not divided into blocks, it will be uploaded to a single qubit, representing a fully-sequential approach. Dividing the image into 4 blocks results in a 4-qubit architecture, positioning it roughly at the midpoint of the sequential-parallel spectrum. If the image is divided into 16 blocks, each block will contain only 1 pixel. This scenario exemplifies the fully-parallel uploading limit, as each pixel is assigned to a separate qubit.

4×4 block is a 16 dimensional vector which requires $\lceil \frac{d}{3} \rceil = \lceil \frac{16}{3} \rceil = 6$ unitary matrices to be encoded onto a qubit. In particular, each block needs 16 rotation gates, one per pixel. Therefore, the unitary matrices necessary to encode a block $\boldsymbol{x} = (x_1, x_2, ..., x_{16})$ will be:

$$U_1^1(\boldsymbol{\phi_1}) = U_1(\boldsymbol{x}, \boldsymbol{\theta_1}) = R_Z(\phi_{1,1}) R_Y(\phi_{1,2}) R_Z(\phi_{1,3}) \tag{1}$$
$$U_2^1(\boldsymbol{\phi_2}) = U_2(\boldsymbol{x}, \boldsymbol{\theta_2}) = R_Z(\phi_{2,1}) R_Y(\phi_{2,2}) R_Z(\phi_{2,3}) \tag{2}$$
$$\vdots \tag{3}$$
$$U_6^1(\boldsymbol{\phi_6}) = U_1(\boldsymbol{x}, \boldsymbol{\theta_6}) = R_Z(\phi_{6,1}) R_Y(\phi_{6,2}) R_Z(\phi_{6,3}) \tag{4}$$

where $\boldsymbol{\theta_i} = (w_{i,1}, w_{i,2}, w_{i,3}, b_{i,1})$. The angles will be defined as a linear combination of pixels and weights, for example $\boldsymbol{\phi_1}$:

$$\phi_{1,1} = x_1 \cdot w_{1,1} + b_{1,1} \tag{5}$$
$$\phi_{1,2} = x_2 \cdot w_{1,2} + b_{1,2} \tag{6}$$
$$\phi_{1,3} = x_3 \cdot w_{1,3} + b_{1,3} \tag{7}$$
$$\tag{8}$$

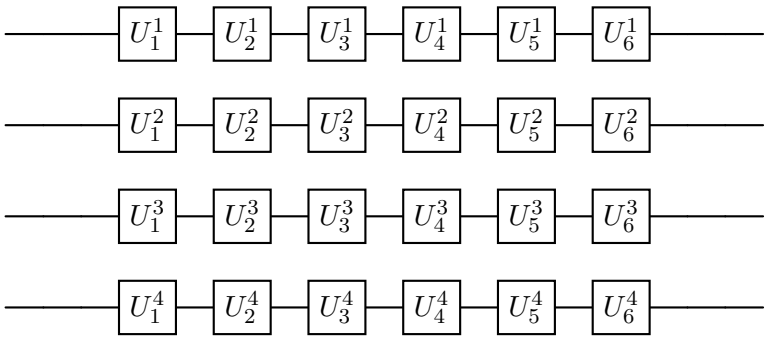


Figure 5: Embedding circuit for an 8×8 image divided in 4 identical 4×4 blocks.

Entanglement structure Since each block is correlated with its neighboring blocks, we decided to use an entanglement structure in which entangling gates create connections between adjacent blocks (figure 6). We chose CZ, as the only entangling gate. Therefore, the entanglement structure aims to ensure that each qubit shares information only with qubits that contain related information.

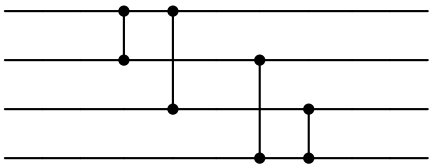


Figure 6: Entanglement structure for an image divided in 4 blocks. The first qubit will communicate with the second one and the third one, as the first block is adjacent to the second and the third one. Then, the second block will be connected to the fourth one and the third to fourth one.

Pooling In classical machine learning the pooling layers are used to make the network less sensitive to small translations and distortions in the input data.
Therefore, we decided to mimic the pooling component of classical convolutional neural

networks, by adding an X rotation gate per qubit, whose angle is defined as the linear combination of the max (or average) value of a block and weights (figure 7). Therefore, if we consider again an 8×8 image divided in 4 identical 4×4 blocks:

$$\boldsymbol{x_1} = (x_{1,1}, x_{1,2}, ..., x_{1,16}) \quad \rightarrow \quad max(\boldsymbol{x_1}) \tag{9}$$
$$\boldsymbol{x_2} = (x_{2,1}, x_{2,2}, ..., x_{2,16}) \quad \rightarrow \quad max(\boldsymbol{x_2}) \tag{10}$$
$$\boldsymbol{x_3} = (x_{3,1}, x_{3,2}, ..., x_{3,16}) \quad \rightarrow \quad max(\boldsymbol{x_3}) \tag{11}$$
$$\boldsymbol{x_4} = (x_{4,1}, x_{4,2}, ..., x_{4,16}) \quad \rightarrow \quad max(\boldsymbol{x_4}) \tag{12}$$
$$\tag{13}$$

the angles of the four X rotation gates will be:

$$\phi_1 = max(\boldsymbol{x_1}) \cdot w_1 + b_1 \tag{14}$$
$$\phi_2 = max(\boldsymbol{x_2}) \cdot w_2 + b_2 \tag{15}$$
$$\phi_3 = max(\boldsymbol{x_3}) \cdot w_3 + b_3 \tag{16}$$
$$\phi_4 = max(\boldsymbol{x_4}) \cdot w_4 + b_4 \tag{17}$$
$$\tag{18}$$

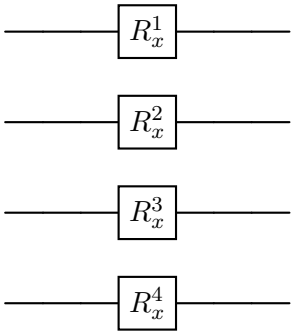


Figure 7: Pooling circuit for a 4 qubit architecture.

Numerical results

We conducted several numerical test on the *block re-uploading* architecture:

- Classification.
  We conducted both binary and multi-class classification.
- Dataset.
  We used both the MNIST digit and MNIST fashion datasets, which are both grayscale 28×28 pixels images.

- Image size.
  We used different down-scaling of the MNIST dataset: 8×8, 12×12, 14×14, 16×16, 18×18.

- Decoding observable.
  Every quantum machine learning algorithm has a *decoding* component at the end of it, which consists in measuring an observable to extract information from the PQC. The observable that we chose are: *global Pauli Z*, which is the tensor product of n (number of qubits of the circuit) Pauli Z, *local Pauli Z*, which is only one Pauli Z. Regarding the local Pauli Z measurement, in the block re-uploading architecture with multiple qubits, we had the option to measure any qubit. However, we consistently chose to measure only the first qubit.

- Architectures.
  We studied the architecture shown in figure 4.

The following sections will discuss the *generalization* capabilities and the *trainability* of the *block re-uploading* architecture.
I will limit my discussion only to 8×8 images.

Local 8x8

By looking at figure 8, we can distinguish three different behaviours:

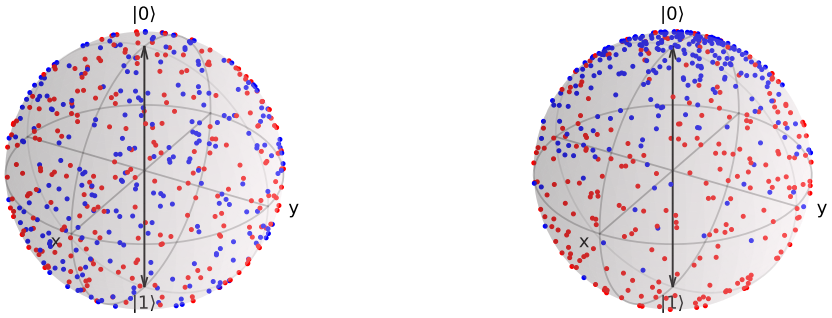
1. **Deep-Narrow**.
   As the number of layers increases in narrow architectures (1, 2, or 3 qubits), their capacity to generalize diminishes, leading to overfitting. This occurs because the increase in layers corresponds to a rise in the number of trainable parameters.
   As a result, the architecture becomes overparameterized, allowing it to capture even minor fluctuations in the training dataset, which reduces its ability to generalize effectively to new, unseen data.

2. **Shallow-Wide**.
   As the number of qubits (width) increases both training and validation accuracy of single-layer architectures (shallow) drop drastically. As the width of the architecture increases, a greater degree of entanglement is required to effectively distribute information across all qubits. However, shallow architectures lack sufficient entangling gates to achieve this, resulting in an inability to fully capture and "understand" the complete picture.
   However, by comparing training and validation accuracy, although both decrease as the width increases, there is no evidence of overfitting.
   We can conclude that, as the architecture widens, entanglement becomes increasingly crucial for the architecture to effectively "understand" the image.

3. **Proportionate**.
   In proportionate architectures (those that are neither shallow-wide nor deep-narrow)

overfitting tends to disappear. As the number of layers and qubits increases, the *block re-uploading* introduces a richer set of entanglement structures, enabling more effective information sharing across all qubits. This reduction in overfitting as entanglement grows raises some natural questions: *Could entanglement be a factor that resists overfitting?*, *Might entanglement serve as a source of regularization?*.

Figure 8: This heatmap shows the training and validation accuracy for architectures with 1-15 qubits and 1-6 layers for the 8×8 down-scaled MNIST digits and fashion dataset using a local Pauli Z.



(a) Before training.

(b) After training.

Figure 9: This figure illustrates the distribution of quantum states for an architecture with 1 qubit and 6 layers, both before and after training, in the task of classifying the digits 0 and 1 from the MNIST dataset. Before training, the predictions are uniformly distributed across the Bloch sphere. However, after training, the architecture clearly distinguishes between the zeros and ones, demonstrating its improved classification capability.

Trainability

We investigated the *trainability* (figures 10 and 11) of the *block re-uploading* architecture this methodology:

- We fixed the size of the images: 8×8;
- We fixed the number of layers: 1, 10, 15, 20, 40, 60, 80, 100;
- We fixed the number of qubits: 1, 2, 3, ..., 16;
- We sampled N (50) models, which means that we sampled N different times the parameters $\boldsymbol{\theta}$ of the model (uniform and gaussian initialization);
- We computed the absolute gradient of the loss function $|\nabla_{\boldsymbol{\theta}} J|$[1];
- We computed the variance value of the absolute gradient $Var(|\nabla_{\boldsymbol{\theta}} J|)$;

Our approach differs from the original paper on barren plateaus, as [**?**] used architectures with only parameters in their parametric gates, without data embedding. In contrast, the *block re-uploading* architecture incorporates both data and parameters in every parametric gate.

[1] Alternatively, we could have computed only the loss function. As previously discussed, barren plateaus can be defined either through the loss function or its gradient. This approach would have been computationally more efficient.

We studied the trainability of the *block re-uploading* architecture with two different decodings: local Pauli Z and global Pauli Z.
Let's compare these two decodings technique when initializing the trainable parameters uniformly between [-π, π] (figure 10):

1. $Var(|\nabla_{\boldsymbol{\theta}} J|)$ *vs layers*: Global vs Local.
   By comparing the two decoding scenarios, we can notice a common behaviour: initially, there is an exponential decrease in variance for the first few layers (which is more pronounced in the local case), after which the variance levels off and reaches a plateau. The primary difference between the global and local cases is that in the global case, the plateau is reached more quickly.
   It is particularly noteworthy that as the number of qubits increases, the plateaus in both the global and local cases become progressively lower. This behaviour indicates that the variance is inversely proportionate to the number of qubits.
   Finally, we can examine the architecture's *layers budget* before reaching barren plateaus[1].
   In the local case, every qubit architecture maintains a non-zero *layers budget* (L1). For architectures with fewer qubits, this budget is effectively unlimited as the plateau is above 10^{-4} (L2). However, as the number of qubits increases, the layers budget decreases, with the 16-qubit architecture having a layers budget ranging from 0 to 8 (L3)[2]. In contrast, the global case shows a different behavior: architectures with 14, 15, and 16 qubits have no *layers budget*, indicating that they are almost untrainable. Moreover, the local and global cases reveals distinct behaviors for single-layer architectures: in the local case, the variance for different qubit architectures is concentrated within the range [10^{-3}, 10^{-1}]; in contrast, for the global case, as the number of qubits increases, the variance progressively decreases, ranging from [10^{0}, 10^{5}].

2. $Var(|\nabla_{\boldsymbol{\theta}} J|)$ *vs qubits*: Global vs Local.
   Since both plots are in log scale, by looking at $Var(|\nabla_{\boldsymbol{\theta}} J|)$ *vs qubits* we can notice that as the number of layers increases $Var(|\nabla_{\boldsymbol{\theta}} J|)$ is exponentially suppressed[1]. In both cases (global and local), we observe that the variance is inversely correlated with the number of qubits: as the number of qubits increases, the variance decreases.

After examining *uniform initialization* (see figure 10), we evaluated *gaussian initialization* (shown in figure 11) for local decoding. However, we observed no significant differences between the two initialization methods.

[1] By *layers budget*, we refer to the number of layers an architecture can accommodate before encountering a barren plateau. For this analysis, we define the threshold for a barren plateau as 10^{-4}

[2] I have indicated with (L1), (L2), (L3) to express that those three observations are valid only for the local case.

[1] A straight-line function with a negative slope on a logarithmic scale corresponds to a negative exponential function on a linear scale.

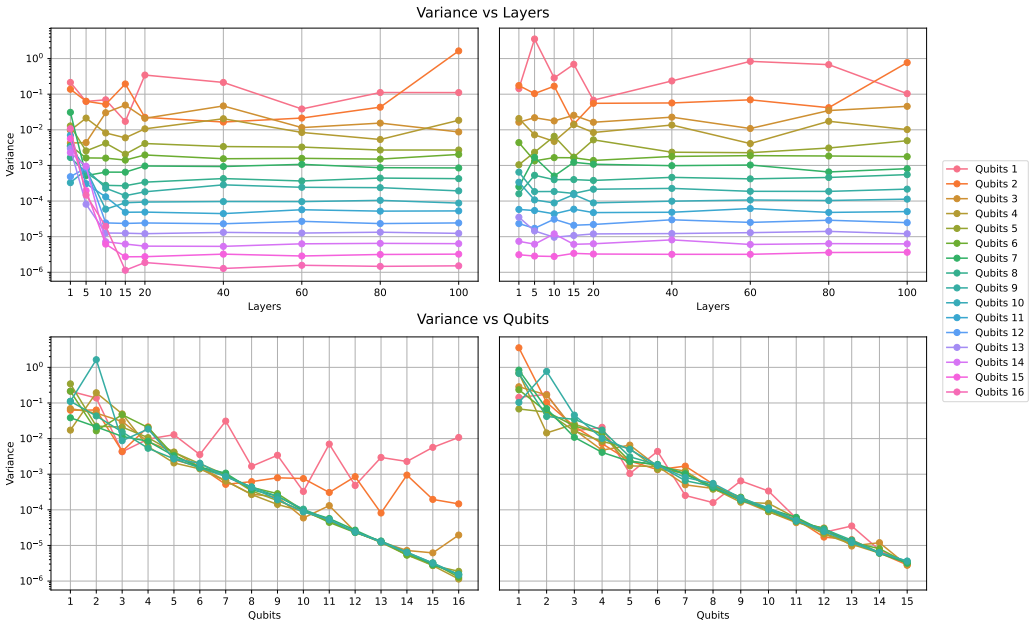


Figure 10: The first row compares the $Var(|\nabla_{\boldsymbol{\theta}} J|)$ *vs layers* plots for the local (left) and global (right) case with a uniform initialization of the parameters ranging [-π, π]. The first row compares the $Var(|\nabla_{\boldsymbol{\theta}} J|)$ *vs qubits* plots for the local (left) and global (right) case with a uniform initialization of the parameters ranging [-π, π]. By local and global case we intend that we either used local Pauli Z or global Pauli Z, as the decoding method.

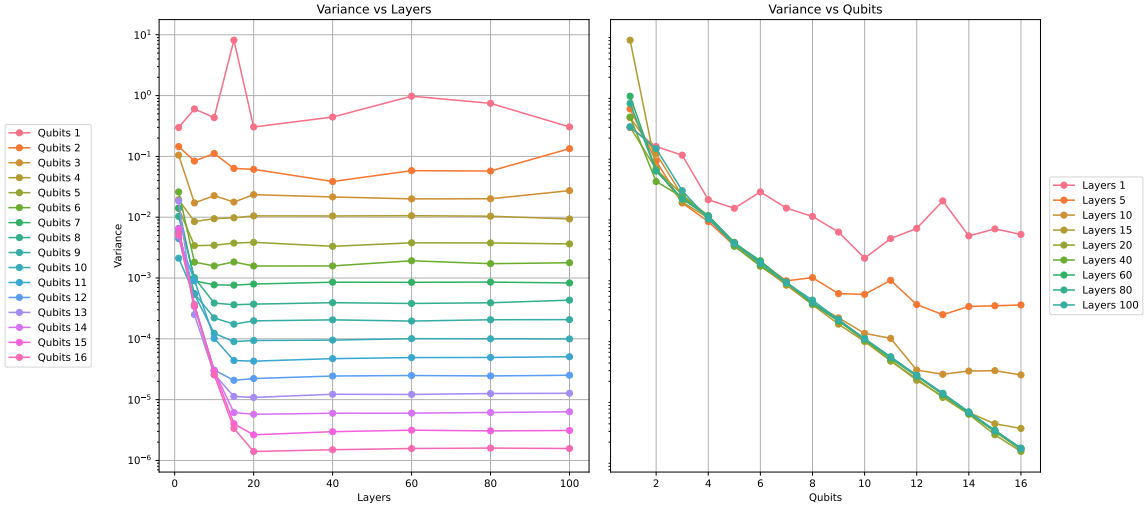


Figure 11: The figure shows the $Var(|\nabla_{\boldsymbol{\theta}} J|)$ *vs layers* and $Var(|\nabla_{\boldsymbol{\theta}} J|)$ *vs qubits* plots with a local decoding and a gaussian initialization of the parameters ($\mu = 1$, $\sigma = 1$).