Quantum machine learning: on the design, trainability and noise-robustness of near-term algorithms

Skolik, A.

CHAPTER 4

Layerwise learning for quantum neural networks

A key aspect of successfully training the variational quantum machine learning models introduced in Section 3.3 is the classical outer loop that optimizes the circuit parameters. One of the most popular choices for parameter optimization in this context are gradient-based methods such as stochastic gradient descent, inspired by their extensive use in the optimization of classical NNs. While the gradient-based backpropagation algorithm [87] is one of the most successful methods used to train NNs today, its direct translation to quantum neural network (QNN)s has been challenging. For QNNs, parameter updates for minimizing an objective function are calculated by stochastic gradient descent, based on direct evaluation of derivatives of the objective with respect to parameters in a PQC, as described in Section 2.2.1. The PQC is executed on a quantum device, while the parameter optimization routine is handled by a classical outer loop. The outer loop's success depends on the quality of the quantum device's output, which in the case of gradient-based optimizers are the partial derivatives of the loss function with respect to the PQC.

As the authors of [41] have shown, gradients of random PQCs vanish exponentially in the number of qubits, as a function of the number of layers. Furthermore, the authors of [44] show that this effect also depends heavily on the choice of cost function, where the barren plateau effect is worst for global cost functions like the fidelity between two quantum states. Furthermore, it was shown that the partial derivatives vanish exponentially in the number of qubits under local Pauli noise, if the depth of the circuit grows linearly with the number of qubits [49]. When executed on a NISQ device, the issue is further amplified because small gradient values can not be distinguished from hardware noise, or will need exponentially many measurements to do so. These challenges motivate the study of training strategies that avoid initialization on a barren plateau, as well as avoid creeping onto a plateau during the optimization process.

Indeed, a number of different optimization strategies for PQCs have been explored, including deterministic and stochastic optimizers [182, 23, 183, 184, 185, 186]. Regardless of the specific form of parameter update, the magnitudes of partial derivatives play a crucial role in descending towards the minimum of the objective function. Excessively small values will slow down the learning progress significantly, prevent progress, or even lead to false convergence of the algorithm to a sub-optimal objective function value. Crucially to this work, small values ultimately lead to a poor signal-to-noise ratio in PQC training algorithms due to the cost of information readout on a quantum device. Even if only sub-circuits of the overall circuit become sufficiently random during the training process, gradient values in a PQC will become exponentially small in the number of qubits [41]. Moreover, in quantum-classical algorithms there is a fundamental readout complexity cost of $\mathcal{O}(1/\epsilon)$ [187] as compared to a similar cost of $\mathcal{O}(\log(1/\epsilon))$ classically. This is because classical arithmetic with floating point numbers for calculating analytic gradients may be done one digit at a time, incurring a cost $\mathcal{O}(\log(1/\epsilon))$. In contrast, quantum methods that require estimation of expectation values by measurements, such as those utilized in NISQ algorithms, converge similarly to classical Monte Carlo sampling. This means that small gradient magnitudes can result in an exponential signal-to-noise problem when training quantum circuits, as the number of measurements required to precisely estimate a value is related to the magnitude of that value. As a consequence, gradients can become too small to be useful even for modest numbers of qubits and circuit depths, and a randomly initialized PQC will start the training procedure on a saddle point in the training landscape with no interesting directions in sight.

To utilize the capabilities of PQCs, methods that overcome this challenge have to be studied. Due to the vast success of gradient-based methods in the classical regime, this work is concerned with understanding how these methods can be adapted effectively for quantum circuits. We propose layerwise learning, where individual components of a circuit are added to the training routine successively. By starting the training routine in a shallow circuit configuration, we avoid the unfavorable type of random initialization described in [41] and Section 2.2.1.2, which is inherent to randomly initialized circuits of even modest depth. In our approach, the circuit structure and number of parameters is successively grown while the circuit is trained, and randomization effects are contained to subsets of the parameters at all training steps. This does not only avoid initializing on a plateau, but also reduces the probability of creeping onto a plateau during training, e.g., when gradient values become smaller on approaching a local minimum.

We compare our approach to a simple strategy to avoid initialization on a barren plateau, namely setting all parameters to zero, and show how the use of a layerwise learning strategy increases the probability of successfully training a PQC with restricted precision induced by shot noise by up to 40% for classifying images of handwritten digits. Intuitively, this happens for reasons that are closely tied to the sampling complexity of gradients on quantum computers. By restricting the training and randomization to a smaller number of circuit components, we focus the magnitude of the gradient into a small parameter manifold. This avoids the randomization issue associated with barren plateaus, but importantly is also beneficial for a NISQ quantum cost model, which must stochastically sample from the training data as well as the components of the gradient. Simply put, with more magnitude in fewer components at each iteration, we receive meaningful training signal with fewer quantum circuit repetitions.

Another strategy to avoid barren plateaus was recently proposed by Grant et al. [53], where only a small part of the circuit is initialized randomly, and the remaining parameters are chosen such that the circuit represents the identity operation. This prevents initialization on a plateau, but only does so for the first training step, and also trains a large number of parameters during the whole training routine. Another way to avoid plateaus was introduced in [54], where multiple parameters of the circuit are enforced to take the same value. This reduces the overall number of trained parameters and restricts optimization to a specific manifold, at the cost of requiring a deeper circuit for convergence [188]. Aside from the context of barren plateaus, [189] investigates a layer-by-layer training scheme to speed up the learning process of a variational quantum eigensolver.

In the classical setting, layerwise learning strategies have been shown to produce results comparable to training a full network with respect to error rate and time to convergence for classical NNs [190, 191]. It has also been introduced as an efficient method to train deep belief networks, which are generative models that consist of restricted Boltzmann machines (RBMs) [192]. Here, multiple layers of

RBMs are stacked and trained greedily layer-by-layer, where each layer is trained individually by taking the output of its preceding layer as the training data. In classical NNs, [193] shows that this strategy can be successfully used as a form of pre-training of the full network to avoid the problem of vanishing gradients caused by random initialization. In contrast to greedy layerwise pre-training, our approach does not necessarily train each layer individually, but successively grows the circuit to increase the number of parameters and therefore its representational capacity.

4.1 Layerwise learning

In Section 2.2.1.2, we described why training random PQCs with growing system size becomes increasingly harder, as the variance of gradients vanishes exponentially in the number of qubits and layers for these types of circuits. This necessitates the search for i) non-random circuit structures that are immune to this issue, or ii) optimization techniques that can combat the problem of vanishing gradients. We will later focus our attention on i) in Chapter 6, while in this section, we introduce a training method for PQCs in the line of ii). We call this optimization technique layerwise learning (LL) for parametrized quantum circuits, a training strategy that creates an ansatz during optimization, and only trains subsets of parameters simultaneously to ensure a favorable signal-to-noise ratio. The algorithm consists of two phases.

Phase one: The first phase of the algorithm constructs the ansatz by successively adding layers. The training starts by optimizing a circuit that consists of a small number s of start layers, e.g. $s = 2$, where all parameters are initialized as zero. We call these the initial layers $l_1(\boldsymbol{\theta}_1)$:

$$l_1(\boldsymbol{\theta}_1) = \prod_{j=1}^{s} U_{1_j}(\boldsymbol{\theta}_{1_j})W \ , \tag{4.1}$$

where $\boldsymbol{\theta}_1$ is the set of angles parametrizing unitary U_{1_j}, and contains one angle for each local rotation gate per qubit, and W represents operators connecting qubits. The number of start layers is a hyperparamter, and should be chosen so that the initial circuit is shallow, but still sufficiently deep in order to advance training. How many start layers are required to fulfill this depends strongly on the learning task.

After a fixed number of epochs, another set of layers is added, and the previous layers' parameters are frozen. We define one epoch as the set of iterations it takes the algorithm to see each training sample once, and one iteration as one update over all trainable parameters. E.g., an algorithm trained on 100 samples with a batch size of 20 will need 5 iterations to complete one epoch. The number of epochs per layer, e_l, is a tunable hyperparameter. Each consecutive layer $l_i(\boldsymbol{\theta}_i)$ takes the form

$$l_i(\boldsymbol{\theta}_i) = \prod_{j=1}^{p} U_{i_j}(\boldsymbol{\theta}_{i_j})W \ , \tag{4.2}$$

with a fixed W, as the connectivity of qubits stays the same during the whole training procedure, and p denoting the number of layers added at once. All angles in $\boldsymbol{\theta}_i$ are set to zero when they are added, which provides additional degrees of freedom for the optimization routine without perturbing the current solution. The parameters added with each layer are optimized together with the existing set of parameters of previous layers in a configuration dependent on two hyperparameters p and q. The hyperparameter p determines how many layers are added in each step, and q specifies after how many layers the previous layers' parameters are frozen. E.g., with $p = 2$ and $q = 4$, we add two layers in each step, and layers more than four steps back from the last layer are frozen. This process is repeated either until additional layers do not yield an improvement in objective function value, or until a desired depth is reached. The final circuit that consists of L layers can then be represented by

$$U(\boldsymbol{\theta}) = \prod_{i=1}^{L} l_i(\boldsymbol{\theta}_i) \ . \tag{4.3}$$

Phase two: In the second phase of the algorithm, we take the pre-trained circuit acquired in phase one, and train larger contiguous partitions of layers at a time. The hyperparameter r specifies the percentage of parameters that is trained in one step, e.g., a quarter or a half of the circuit's layers. The number of epochs for which these partitions are trained is also controlled by e_l, which we keep at the same value as in phase one for the sake of simplicity, but which could in principle be treated as a separate hyperparameter. In this setting, we perform additional optimization sweeps where we alternate over training the specified subsets of parameters simultaneously, until the algorithm converges. This allows us to train larger partitions of the circuit at once, as the parameters from phase one provide a sufficiently non-random initialization. As the randomness is contained to shallower sub-circuits during the whole training routine, we also minimize the probability to creep onto a plateau during training as a consequence of stochastic or hardware noise present in the sampling procedure.

In general, the specific structure of layers $l_i(\boldsymbol{\theta}_i)$ can be arbitrary, as long as they allow successively increasing the number of layers, like in the hardware-efficient ansatz introduced in [170]. In this work, we indirectly compare the quality of gradients produced by our optimization strategy with respect to the results described in section 2.2.1.2 through overall optimization performance, so we consider circuits that consist of layers of randomly chosen gates as used in [41]. They can be represented in the following form:

$$U(\boldsymbol{\theta}) = \prod_{l=1}^{L} U_l(\boldsymbol{\theta_l})W \ , \tag{4.4}$$

where $U_l(\boldsymbol{\theta}_l) = \prod_{i=1}^{n} \exp(-i\theta_{l,i}V_i)$ with a Hermitian operator V_i, n is the number of qubits, and W is a generic fixed unitary operator. For ease of exposition, we drop the subscripts of the individual gates in the remainder of this work. We consider single qubit generators V which are the Pauli operators X, Y and Z for each qubit, parametrized by θ_l, while W are CZ gates coupling arbitrary pairs of qubits. An example layer is depicted in Figure 4.1.

The structure and parameters of a quantum circuit define which regions of an optimization landscape given by a certain objective function can be captured. As the number of parametrized non-commuting gates grows, this allows a more fine-grained representation of the optimization landscape [188]. In a setting where arbitrarily small gradients do not pose a problem, e.g. noiseless simulation of PQC training, it is often preferable to simultaneously train all parameters in a circuit to make use of the full range of degree of freedom in the parameter landscape. We will refer to this training scheme as complete-depth learning (CDL) from now on. In a noiseless setting, LL and CDL will perform similarly w.r.t. the number of calls to a quantum processing unit (QPU) until convergence and final accuracy of results, as we show in the appendix. This is due to a trade off between the number of parameters in a circuit and the number of sampling steps to convergence [188]. A circuit with more parameters will converge faster in number of training epochs, but will need more QPU calls to train the full set of parameters in each epoch. On the other hand, a circuit with fewer parameters will show a less steep learning curve, but will also need fewer calls to the QPU in each update step due to the reduced number of parameters. When we consider actual number of

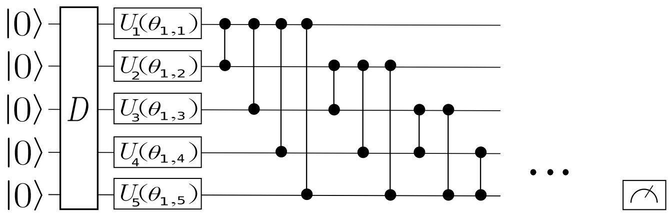


Figure 4.1: Sample circuit layout of the first layer in an LL circuit. D represents the data input which is only present once at the beginning of the circuit. The full circuit is built by successively stacking single rotation gates and two qubit gates to form all-to-all connected layers. For the classification example we show in 4.2, a measurement gate is added on the last qubit after the last layer.

calls to a quantum device until convergence as a figure of merit, training the full circuit and performing LL will perform similarly in a noise-free setting for this reason. However, this is not true when we enter a more realistic regime, where measurement of gradient values will be affected by stochastic as well as hardware noise, as we will show on the example of shot noise in Section 4.2. In such more realistic settings, the layerwise strategy offers considerable advantage in time to solution and quality of solution.

As noted in the appendix of [41], the convergence of a circuit to a 2-design does not only depend on the number of qubits, their connectivity and the circuit depth, but also on the characteristics of the cost function used. This was further investigated in [44], where cost functions are divided into those that are local and global, in the sense that a global cost function uses the combined output of all qubits (e.g., the fidelity of two states), whereas a local cost function compares values of individual qubits or subsets thereof (e.g., a majority vote). Both works show that for global cost functions, the variance of gradients decays more rapidly, and that barren plateaus will present themselves even in shallow circuits. As our training strategy relies on using larger gradient values in shallow circuit configurations, especially during the beginning of the training routine, we expect that LL will mostly yield an advantage in combination with local cost functions.

4.2 Results

4.2.1 Setup

To examine the effectiveness of LL, we use it to train a circuit with fully-connected layers as described in Section 4.1. While fully-connected layers are not realistic on NISQ hardware, we choose this configuration for our numerical investigations because it leads circuits to converge to a 2-design with the smallest number of qubits and layers [41], which allows us to reduce the computational cost of our simulations while examining some of the most challenging situations. To compare the performance of LL and CDL we perform binary classification on the MNIST data set of handwritten digits, where the circuit learns to distinguish between the numbers six and nine. We use the binary cross-entropy as the training objective function, given by

$$-\mathcal{L}(\boldsymbol{\theta}) = -\big(y \log\left(E(\boldsymbol{\theta})\right) + (1-y)\log\left(1-E(\boldsymbol{\theta})\right)\big) \ , \tag{4.5}$$

where log is the natural logarithm, $E(\boldsymbol{\theta})$ is given by a measurement in the Z-direction $M = Z_o$ on qubit o which we rescale to lie between 0 and 1 instead of -1 and 1, y is the correct label value for a given sample, and $\boldsymbol{\theta}$ are the parameters of the PQC. The loss is computed as the average binary cross entropy over the batch of samples. In this case, the partial derivative of the loss function is given by

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_i} = y\frac{1}{E(\boldsymbol{\theta})}\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i} - (1-y)\frac{1}{1-E(\boldsymbol{\theta})}\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i} \ . \tag{4.6}$$

To calculate the objective function value, we take the expectation value of the circuit of observable M,

$$E(\boldsymbol{\theta}) = \langle\psi| U^\dagger(\boldsymbol{\theta}) M U(\boldsymbol{\theta}) |\psi\rangle \,, \tag{4.7}$$

where $|\psi\rangle$ is the initial state of the circuit given by the training data set. The objective function now takes the form $\mathcal{L}(E(\boldsymbol{\theta}))$ and the partial derivative for parameter θ_i is defined using the chain rule as

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \frac{\partial \mathcal{L}}{\partial E(\theta_i)} \cdot \frac{\partial E(\theta_i)}{\partial \theta_i} \ . \tag{4.8}$$

To compute gradients of $E(\boldsymbol{\theta})$, we use the parameter-shift rule [39, 33] as described in Section 2.2.1.1. We note that in the numerical implementation, care must be taken to avoid singularities in the training processes related to $E(\theta) = \{0, 1\}$ treated similarly for both the loss and its derivative (we clip values to lie in $[10^{-15}, 1 - 10^{-15}]$). We choose the last qubit in the circuit as the readout o, as shown in Figure 4.1. An expectation value of 0 (1) denotes a classification result for class sixes (nines). As we perform binary classification, we encode the classification result into one measurement qubit for ease of implementation. This can be generalized to multi-label classification by encoding classification results into multiple qubits, by assigning the measurement of one observable to one data label. We use the Adam optimizer [38] with varying learning rates to calculate parameter updates and leave the rest of the Adam hyperparameters at their typical publication values.

To feed training data into the PQC, we use qubit encoding in combination with principal component analysis (PCA), following [24]. Due to the small circuits used in this work, we have to heavily downsample the MNIST images. For this, a PCA is run on the data set, and the number of principal components with highest variance corresponding to the number of qubits is used to encode the data into the PQC. This is done by scaling the component values to lie within $[0, 2\pi)$, and using the scaled values to parametrize a data layer consisting of local X-gates. In case of 10 qubits, this means that each image is represented by a vector $\boldsymbol{d}$ with the 10 components, and the data layer can be written as $\prod_{i=1}^{10} \exp(-id_i X_i)$.

Different circuits of the same size behave more and more similarly during training as they grow more random as a direct consequence of the results in [41]. This means that we can pick a random circuit instance that, as a function of its number of qubits and layers, lies in the 2-design regime as shown in Figure 2.3, and gather representative training statistics on this instance. As noted in Section 4.1, an LL scheme is more advantageous in a setting where training the full circuit is infeasible, therefore we pick a circuit with 8 qubits and 21 layers for our experiments, at which size the circuit is in this regime. When using only a subset of qubits in a circuit as readout, a randomly generated layer might not be able to significantly change its output. For example, if in our simple circuit in Figure 4.1, $U_5(\theta_{1,5})$ is a rotation around the Z axis followed only by CZ gates, no change in $\theta_{1,5}$ will affect the measurement outcome on the bottom qubit. When choosing generators randomly from $\{X, Y, Z\}$ in this setting, there is a chance of 1/3 to pick an unsuitable generator. To avoid this effect, we enforce at least one X gate in each set of layers that is trained. For our experiments, we take one random circuit instance and perform LL and CDL with varying hyperparameters.

4.2.2 Sampling requirements

To give insight into the sampling requirements of our algorithm, we have to determine the components that we need to sample. Our training algorithm makes use of gradients of the objective function that are sampled from the circuit on the quantum computer via the parameter shift rule as described in Section 4.2.1. The precision of our gradients now depends on the precision of the expectation values for the two parts of the r.h.s. in Equation 2.16. The estimation of an expectation value scales in the number of measurements N as $\mathcal{O}(\frac{1}{\epsilon^{\alpha}})$, with error ϵ and $\alpha > 1$ [187]. For most near-term implementations using operator averaging, $\alpha = 2$, resembling classical central limit theorem statistics of sampling. This means that the magnitude of partial derivatives $\frac{\partial E}{\partial \theta_i}$ of the objective function directly influences the number of samples needed by setting a lower bound on ϵ, and hence the signal-to-noise ratio achievable for a fixed sampling cost. If all of the magnitudes of $\frac{\partial E}{\partial \theta_i}$ are much smaller than ϵ, a gradient based algorithm will exhibit dynamics more resembling a random walk than optimization.

4.2.3 Comparison to CDL strategies

We compare LL to a simple approach to avoid initialization on a barren plateau, which is to set all circuit parameters in a circuit to zero followed by a CDL training strategy. We argue that considering the sampling requirements of training PQCs as described in Section 4.2.2, an LL strategy will be more frugal in the number of samples it needs from the QPU. Shallow circuits produce gradients with larger magnitude as can be seen in Figure 2.3, so the number of samples $1/\epsilon^2$ we need to achieve precision ϵ directly depends on the largest component in the gradient. This difference is exhibited naturally when considering the number of samples as a hyperparameter in improving time to solution for training. In this low sample regime, the training progress depends largely on the learning rate. A small batch size and low number of measurements will increase the variance of objective function values. This can be balanced by choosing a lower learning rate, at the cost of taking more optimization steps to reach the same objective function value. We argue that the CDL approach will need much smaller learning rates to compensate for smaller gradient values and the simultaneous update of all parameters in each training step, and therefore more samples from the QPU to reach similar objective function values as LL. We compare both approaches w.r.t. their probability to reach a given accuracy on the test set, and infer the number of repeated re-starts one would expect in a real-world experiment based on that.

In order to easily translate the simulated results here to experimental impact, we also compute an average runtime by assuming a sampling rate of 10kHz. This value is assumed to be realistic in the near term future, based on current superconducting qubit experiments shown in [2] which were done with a sampling rate of 5kHz, not including cloud latency effects. The cumulative number of individual measurements taken from a quantum device during training is defined as

$$r_i = r_{i-1} + 2n_p mb \ , \tag{4.9}$$

where n_p is the number of parameters (taken times two to account for the parameter shift rule shown in Section 4.2.1), m the number of measurements taken from the quantum device for each expectation value estimation, and b the batch size. This gives us a realistic estimate of the resources used by both approaches in an experimental setting on a quantum device.

4.2.4 Numerical results

For the following experiments, we use a circuit with 8 qubits, 1 initial layer and 20 added layers, which makes 21 layers in total. As can be seen in Figure 2.3, this is a depth where a fully random circuit is expected to converge to a 2-design for the all-to-all connectivity that we chose. After doing a hyperparameter search over p, q and e_l, we set the LL hyperparameters to $p = q = 2$ and $e_l = 10$, with one initial layer that is always active during training. This means that three layers are trained at once in phase one of the algorithm, and 10 and 11 layers are trained as one contiguous partition in phase two, respectively. For CDL, the same circuit is trained with all-zero initialization.

We argue that LL not only avoids initialization on a plateau, but is also less susceptible to randomization during training. In NISQ devices, this type of randomization is expected to come from two sources: (i) hardware noise, (ii) shot noise, or measurement uncertainty. The smaller the values we want to estimate and the less exact the measurements we can take from a QPU are, the more often we have to repeat them to get an accurate result. Here, we investigate the robustness of both methods to shot noise. The hyperparameters we can tune are the number of measurements m, batch size b and learning rate η. The randomization of circuits during training can be reduced by choosing smaller learning rates to reduce the effect of each individual parameter update, at the cost of more epochs to convergence. Therefore we focus our hyperparameter search on the learning rate η, after fixing the batch size to $b = 20$ and the number of measurements to $m = 10$. This combination of m and b was chosen for a fixed, small m after conducting a search over $b \in \{20, 50, 100\}$ for which both LL and CDL could perform successful runs that do not diverge during training. As we lower the batch size, we also increase the variance in objective function values similar to when the number of measurements is reduced, so these two values have to be tuned to match each other. In the remainder of this section we show results for these hyperparameters, and different learning rates for both methods. All of the results are based on 100 runs of the same hyperparameter configurations. We use 50 samples of each class to calculate the cross entropy during training, and another 50 samples per class to calculate the test error. To compute the test error, we let the model predict binary class labels for each presented sample, where a prediction ≤ 0.5 is interpreted as class 0 (sixes) and > 0.5 as class 1 (nines). The test error is then the average error over all classified samples.

Figure 4.2 shows average runtimes of LL and CDL runs that have a final average error on the test set that is less than 0.5, which corresponds to random guessing. We compute the runtime by computing the number of samples taken as shown in Section 4.2.3 and assume a sampling rate of 10kHz. Here, LL reaches a lower error on the test set on average, and also requires a lower runtime to get there. Compared to the CDL configuration with the highest success probability shown in Figure 4.3 (b) (red line), the best LL configuration (blue line) takes approximately half as much time to converge. This illustrates that LL does not only increase the probability of successful runs, but can also drastically reduce the runtime to train PQCs by only training a subset of all parameters at a given training step. Note also that the test error of CDL with $\eta = 0.05$ and $\eta = 0.01$ slowly increases at later training steps, which might look like overfitting at first. Here it is important to emphasize that these are averaged results, and what is slowly increasing is rather the percentage of circuits that have randomized or diverged at later training steps. The actual randomization in an individual run usually happens with a sudden jump in test error, after which the circuit can not return to a regular training routine anymore.

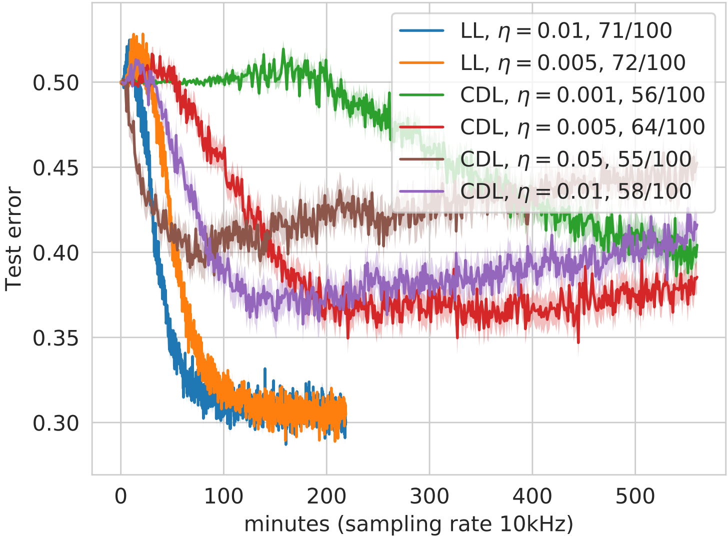
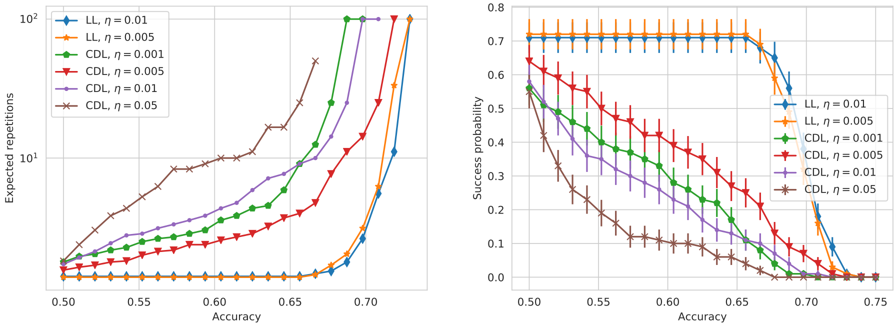


Figure 4.2: Average test error as a function of runtimes for runs that have a final average test error less than 0.5 (random guessing) over the last ten training epochs, assuming a sampling rate of 10kHz and number of samples taken as described in Section 4.2.3. Numbers in labels indicate how many out of 100 runs were included in the average, i.e. fraction of runs that did not diverge in training, exhibiting less than 50% error on the test set. Increasing test error for CDL runs with $\eta = 0.01$ and $\eta = 0.05$ is not due to overfitting, but due to a larger number of runs in the average that start creeping onto a plateau due to the increased learning rate.



(a) expected number of repetitions

(b) success probability

Figure 4.3: LL decreases expected run time and increases probability of success on random restarts. (a) Expected number of experiment repetitions needed until a given configuration reaches a certain accuracy defined as $(1 - \text{error}_{\text{test}})$, where $\text{error}_{\text{test}}$ is the average error on the test set, for LL and CDL with different learning rates. One experiment repetition constitutes in one complete training run of a circuit to a fixed number of epochs. Results are based on 100 runs for each configuration with $m = 10$, $b = 20$, and in case of LL, $e_l = 10$. LL circuits better avoid randomization during training, and therefore need less than two repetitions on average for learning rates with varying magnitudes. CDL is more susceptible to entering a plateau during training in a noisy environment, as all parameters are affected on a perturbative update. This effect becomes more pronounced as learning rates are increased. (b) Probability of reaching a certain accuracy on the test set for the same configurations shown in (a). Success probability of LL stays constant up to an accuracy of 0.65 and starts decaying from there, as fewer runs reach higher accuracies on average. All CDL configurations have a lower success probability than the LL configurations overall, which decays almost linearly as we demand a higher average accuracy. Notably, the CDL configurations with highest success probability are also the ones with the highest runtime, as shown in Figure 4.2.

Figure 4.3 (a) shows the number of expected training repetitions one has to perform to get a training run that reaches a given accuracy on the test set, where we define accuracy as $(1 - \text{error}_{\text{test}})$. One training run constitutes training the circuit to a fixed number of epochs, where the average training time for one run is shown in Figure 4.2. An accuracy of 0.5 corresponds to random guessing, while an accuracy of around 0.73 is the highest accuracy any of the performed runs reached, and corresponds to the model classifying 73% of samples correctly. We note that in a noiseless setting as shown in Chapter 8, both LL and CDL manage to reach accuracies around 0.9, and the strong reduction in number of measurements leads to a decrease in the final accuracy reached by all models. We find that LL performs well for different magnitudes of learning rates as $\eta = 0.01$ and $\eta = 0.005$, and that these configurations have a number of expected repetitions that stays almost constant as we increase the desired accuracy. On average, one needs less than two restarts to get a successful training run when using LL. For CDL, the number of repetitions increases as we require the test error to reach lower values. The best configurations were those with $\eta = 0.001$ and $\eta = 0.005$, which reach similarly low test errors as LL, but need between 3 and 7 restarts to succeed in doing so. This is due to the effect of randomization during training, which is caused by the high variance in objective function values, and the simultaneous update of all parameters in each training step. In Figure 4.3 (b), we show the probability of each configuration shown in (a) to reach a given accuracy on the test set. All CDL configurations have a probability lower than 0.3 to reach an accuracy above 0.65, while LL reaches this accuracy with a probability of over 0.7 in both cases. This translates to the almost constant number of repetitions for LL runs in Figure 4.3 (a). Due to the small number of measurements and the low batch size, some of the runs performed for both methods fail to learn at all, which is why none of the configurations have a success probability of 1 for all runs to be better than random guessing.

4.3 Conclusion and outlook

We have shown that the effects of barren plateaus in QNN training landscapes can be dampened by avoiding Haar random initialization and randomization during training through layerwise learning. While performance of LL and CDL strategies is similar when considering noiseless simulation and exact analytical gradients, LL strategies outperform CDL training on average when experimentally realistic measurement strategies are considered. Intuitively, the advantage of this approach is drawn from both preventing excess randomization and concentrating the contributions of the training gradient into fewer, known components. Doing so directly decreases the sample requirements for a favorable signal-to-noise ratio in training with stochastic quantum samples. To quantify this in a cost effective manner for simulation, we reduce the number of measurements taken for estimating each expectation value. We show that LL can reach lower objective function values with this small number of measurements, while reducing the number of overall experiment repetitions until convergence to roughly half of the repetitions needed by CDL when comparing the configurations with highest success probability. This makes LL a more suitable approach for implementation on NISQ devices, where taking a large number of measurements is still costly and where results are further diluted by decoherence effects and other machine errors. While our approach relies on manipulating the circuit structure itself to avoid initializing a circuit that forms a 2-design, it can be combined with approaches that seek to find a favorable initial set of parameters as shown in [53]. The combination of these two approaches by choosing good initial parameters for new layers is especially interesting as the circuits grow in size. This work has also only explored the most basic training scheme of adding a new layer after a fixed number of epochs, which can still be improved by picking smarter criteria like only adding a new layer after the previous circuit configuration converged, or replacing gates in layers which provide little effect on changes of the objective function value. Moreover, one could consider training strategies which group sets of coordinates rather than circuit layers. These possibilities provide interesting directions for additional research, and we leave their investigation for future works.