The Quantum Path Kernel: a Generalized Quantum Neural Tangent Kernel for Deep Quantum Machine Learning

Massimiliano Incudini[1], Michele Grossi[2], Antonio Mandarino[3], Sofia Vallecorsa[2], Alessandra Di Pierro[1], and David Windridge[4]

[1]Department of Computer Science, University of Verona, Verona 37134, Italy
[2]European Organization for Nuclear Research (CERN), Geneva 1211, Switzerland
[3]International Centre for Theory of Quantum Technologies (ICTQT), University of Gdansk, 80-309 Gdańsk, Poland
[4]Department of Computer Science, Middlesex University, The Burroughs, London, NW4 4BT, UK

Abstract

Building a quantum analog of classical deep neural networks represents a fundamental challenge in quantum computing. A key issue is how to address the inherent non-linearity of classical deep learning, a problem in the quantum domain due to the fact that the composition of an arbitrary number of quantum gates, consisting of a series of sequential unitary transformations, is intrinsically linear. This problem has been variously approached in the literature, principally via the introduction of measurements between layers of unitary transformations. In this paper, we introduce the Quantum Path Kernel, a formulation of quantum machine learning capable of replicating those aspects of deep machine learning typically associated with superior generalization performance in the classical domain, specifically, *hierarchical feature learning*. Our approach generalizes the notion of Quantum Neural Tangent Kernel, which has been used to study the dynamics of classical and quantum machine learning models. The Quantum Path Kernel exploits the parameter trajectory, i.e. the curve delineated by model parameters as they evolve during training, enabling the representation of differential layer-wise convergence behaviors, or the formation of hierarchical parametric dependencies, in terms of their manifestation in the gradient space of the predictor function. We evaluate our approach with respect to variants of the classification of Gaussian XOR mixtures - an artificial but emblematic problem that intrinsically requires multilevel learning in order to achieve optimal class separation.

1 Introduction

Bridging classical deep neural networks and quantum computing represents a key research challenge in the field of *quantum machine learning* [1, 2]. The potential for improvement offered by quantum computing in the machine learning domain may be characterized in terms of its impact on algorithmic efficiency, generalization error, or else its capacity for treating quantum data [3].

A notable recent result in the field has been the introduction of the concept of *variational quantum algorithms* and the related neural network analog referred to as the *quantum neural network* (QNN) [4]. This, in essence, consists of a feature map encoding data into a quantum Hilbert space upon which certain parameterized unitary rotations are applied prior to final measurement in order to obtain a classification or regression output. The system as a whole is then optimized by classical methods. Such models provably lead to a computational advantage over classical models on certain artificial tasks [5], and in respect to the analysis of specific physical systems [6]. It has been quantitatively shown that QNNs can be trained faster than their classical analogues [4]. However, QNNs remain problematic in various respects. One limitation arises from the so-called barren plateau problem [7], in which the variance of the gradient vanishes exponentially with the system size as the parameterized transformation becomes increasingly expressive [8]. A number of approaches, including layer-wise training of quantum neural networks [9], have been proposed to mitigate the issue.

A second problematic aspect of QNNs, and the one that constitutes our principal focus here, is the linearity of the dynamics of quantum systems. Concatenations of linear unitary transformations remain unitary and thus 'stacked' quantum transformations, in effect, collapse to a single linear transformation, appearing to rule out de facto the hierarchical feature learning of classical deep neural networks, which relies on non-linearities to separate feature layers. This property makes the QNN essentially a kernel machine [10]. In terms of the predictor function, however, the QNN is composed of multiplications of rotation operators parameterized by both the feature and model weights. The nonlinearity of projections of rotation operators can be exploited to replicate a very constrained form of non-linearity for feature learning [11]. Another strategy is to introduce nonlinearity via the measurement operation, i.e. a *dissipative QNN* [12]. Both approaches involve the projection the quantum state into a subspace of the original Hilbert space.

Much of the recent study of the dynamics of deep neural networks in the classical realm has focused on the Neural Tangent Kernel (NTK) [13] which represents the network in terms of the corresponding training gradients in the model parameter space. The NTK hence approximates the behavior of predictors via a linear model. It is often therefore applied to study neural networks in their asymptotic, infinite-width, limit. In this regime, the network exhibits *lazy training* [14], i.e. parameter gradients remain at their initial values during the entirety of training. The NTK thus accurately characterizes the dynamics of such infinite-width neural networks, but is otherwise only an approximation

[15]. The difference in test error between the predictor and its linearized version depends on the problem structure [16], with hierarchical feature learning capability being crucial to obtaining superior performance [17]. However, the kernel nature of the NTK means that it shares with quantum computing a ready interpretation within a Hilbert space, and is thus of considerable interest within quantum machine learning. The first explicit application of NTK to quantum neural networks, the *quantum neural tangent kernel* (QNTK) was given in [18].

In this paper, we propose a method for overcoming the de facto lack of hierarchical feature learning capability in QNNs. We propose the application of Path Kernels [19] to QNNs, which we call the *Quantum Path Kernel* (QPK). Such an approach generalizes the QNTK so that the resulting kernel is representative of the ensemble of NTKs calculated over the full parameter path trajectory, i.e. the function describing the evolution of model parameters over time, including implicitly any parametric evolutions corresponding to hierarchical feature learning. We show experimentally an increased expressivity of the resulting model relative to linearized equivalents, evaluating our method on the Gaussian XOR mixture classification problem. For this problem, finite-width neural networks have both theoretically and empirically shown to be close-to-optimal performance whereas linear NTK models fail [20], suggesting that it cannot be effectively resolved without implicating multilevel learning behavior. Furthermore, we discuss possible improvements for the proposed approach, which can be obtained by considering only the contribution of the parameter gradient path that gives rise to the most decorrelated feature representation. These specifically corresponds to the contributions associated with the maximally nonlinear point of the parameter path, corresponding to the largest (positive or negative) eigenvalues of the Hessian of the predictor function [21]. We further enhance the decorrelation between feature representations via a stochastic, noisy, or non-gradient-descent-based training algorithm in which the averaging operation between decorrelated representations allows us to interpret the model as an ensemble technique.

The paper is structured as follows. In Section 2 we briefly review the necessary conceptual background. In Section 3 we present the Quantum Path Kernel and discuss the hierarchical feature learning of the induced model. In Section 4 we demonstrate how this leads to superior performance in solving the Gaussian XOR mixture classification problem. In Section 5 we draw our conclusions and present directions for further work.

1.1 Contributions

- We propose the *Quantum Path Kernel* as a mechanism for building hybrid classical/quantum machine learning models which are able to emulate the hierarchical feature learning structure of deep neural networks without violating the underlying linearity of the quantum dynamics.

- We provide numerical evidence of the superior performance of the Quantum Path Kernel compared to the QNTK on the Gaussian XOR Mixture

problem, which is Bayes optimally soluble only through implicating layer-wise nonlinear separability.

- We consider the importance of the extraction of non-correlated feature representations corresponding to maximally varying portions of the parameter gradient path.

1.2 Related works

The introduction of the NTK by [13] has marked a significant step in the theory of machine learning, sheding new light on discussions regarding the relative performance of linear and nonlinear models. For example, [16] suggests that tasks in which kernel methods (including NTK) perform worse than neural networks are those in which the kernel suffers from the curse of dimensionality whereas neural networks, in learning some useful lower dimensional representation, do not. One example of such a problem is the Gaussian XOR Mixture classification task [20]. Furthermore, linearized models have been shown to perform slightly worse than wide (i.e. large, but non-infinite) neural networks on CIFAR-10 benchmark [22], with the gap between the approaches increasing for finite width networks [23].

In relation to quantum computation, researchers have spent substantial effort on the limitations imposed by the linear dynamics of quantum systems. Authors in [24] review early approaches to the formulation of nonlinear quantum machine learning models: some have focused on developing a *quantum perceptron* equivalent or *quantum neuron*, i.e. a candidate building block for the quantum analogue of neural networks; [25] uses phase estimation to implement the functioning of a step function; [26, 27] propose to exploit the RUS (repeat until success) policy to mimic the behaviour of tangent and sigmoid activation functions, while [28] uses RUS to construct a Born machine; [29] emulates the nonlinearity of perceptrons using measurements. In relation to QNNs, [30] propose dissipative QNNs in which the nonlinearity is obtained via intertwining measurements between unitary gates; [31, 32] propose the use of a larger Hilbert space to implement the nonlinear transformation, while [33] exploits the exponential form of unitary gate to achieve periodic activation functions. Finally, non-linear models of quantum mechanics have been conjectured by [34], although these violate some computational complexity assumptions [35].

2 Background

This section briefly introduces the key concepts and notations in relation to Deep Learning and Quantum Machine Learning through which we develop our results. We denote by $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subseteq \mathcal{X} \times \mathcal{Y}$ a labelled dataset of pairs that are i.i.d. sampled from an unknown probability distribution. We indicate the data vector space with $\mathcal{X} = \mathbb{R}^d$, and the target space with either $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} \subseteq \mathbb{Z}, |\mathcal{Y}| < \infty$ for regression or classification tasks, respectively. We indicate uniform sampling

from a uniform discrete distribution with $\sim \{v_i\}_{i=1}^n$ and sampling from a normal distribution of mean μ and variance σ^2 with $\sim \mathcal{N}(\mu, \sigma)$.

2.1 A primer on quantum machine learning models

Here we fix the notation for our quantum machine learning models. The state of a quantum system of m-qubits is described by a density matrix $\rho \in \mathcal{H} \equiv \mathbb{C}^{2^m \times 2^m}$. The initial state of a quantum computation is denoted by $\rho_0 = |0\rangle\langle 0|$, and the (possibly parametric) unitary transformations by U, V, W. Any parametric unitary can be written as

$$U(\boldsymbol{\theta}) = \exp\left\{-i \sum_{k=1}^{m} f_j(\boldsymbol{\theta}) \sigma_{\alpha_1,\ldots,\alpha_k}^{(q_1,\ldots,q_k)}\right\}, \tag{1}$$

where $\alpha_i \in \{\mathrm{X}, \mathrm{Y}, \mathrm{Z}, \mathbf{1}\}$ for $i = 1, \ldots, k$, and $\sigma_{\alpha_1,\ldots,\alpha_k}$ is a tensor product of one or more corresponding Pauli matrices applied to qubits $q_1, ..., q_k$. The same transformation may be interpreted as a rotation and be equivalently denoted by $R_{\alpha_1,\ldots,\alpha_k}^{(i_1,\ldots,i_k)}(\boldsymbol{\theta})$, where $\boldsymbol{\theta} \in \mathbb{R}^P$ are rotational angles. A *quantum neural network* is a function of the form[1]:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathrm{Tr}[\rho_{\mathbf{x},\boldsymbol{\theta}} O] = \mathrm{Tr}[V^\dagger(\boldsymbol{\theta}) U_\phi^\dagger(\mathbf{x}) \rho_0 U_\phi(\mathbf{x}) V(\boldsymbol{\theta}) O], \tag{2}$$

where O indicates any measurement operator. Both the matrices U and V are decomposed in single and two-qubits parametric rotations interspersed with non-parametric gates (e.g. CNOT).

2.2 Notions of nonlinearity in classical and quantum learning models

With respect to both kernel machines and layerwise deep learning, the concepts of *linear model*, *nonlinear model*, and *feature learning* that we utilize here are as formalized in [37]. A *linear model* is thus a function of the form:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sum_{j=1}^{p} \theta_j \phi_j(\mathbf{x}), \tag{3}$$

where $\{\phi_j : \mathcal{X} \to \mathbb{R}\}_{j=0}^p$ are the *feature functions*, whose values corresponds with the model features. We might consider an additional feature $\phi_0 \equiv 1$ that incorporates the bias. The formula in Equation 3 is linear with respect to the space of the parameters[2] $\mathcal{H} \equiv \mathbb{R}^p$; in fact, we can interpret the function as an

[1] The most general form of QNN proposed is the *data re-uploading* QNN, which allows the interspersing of data encoding and trainable transformations. Such a form, however, does not add any computational power to the standard QNN approach [36].

[2] This formalism allows us to exploit even infinite dimensional Hilbert spaces, such as the one implemented by the Gaussian feature map or RBF, mapping $\mathbf{x}$ to a multivariate Gaussian of mean $\mathbf{x}$ and fixed covariance, existing in the space of square-integrable multivariate functions.

inner product in that space, i.e.

$$f(\mathbf{x};\boldsymbol{\theta}) = \langle \boldsymbol{\theta}, \boldsymbol{\phi}(\mathbf{x}) \rangle_{\mathbb{R}^p}, \tag{4}$$

with $\boldsymbol{\theta} = (\theta_1, ..., \theta_p)$ and $\boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), ..., \phi_p(\mathbf{x}))$. The optimal parameters of such a model can be found analytically by solving the linear regression problem over the Mean Squared Error loss, which is a convex, quadratic function of the parameters. The *representer theorem* guarantees that the optimal solution is a span of the m data points of the training set, which is independent from the dimensionality n of the space $\mathcal{H}$. Obviously, a model which is linear in the parameters may well behave nonlinearly with respect to the original feature space $\mathcal{X}$, due to the feature functions.

A *nonlinear model* is a function of the form:

$$f(\mathbf{x};\boldsymbol{\theta}) = \sum_{j=1}^{p} \theta_j \phi_j(\mathbf{x}) + \frac{\epsilon}{2} \sum_{j,k=1}^{p} \theta_j \theta_k \psi_{j,k}(\mathbf{x}) + \frac{\epsilon}{3!} \sum_{j,k,\ell=1}^{p} \theta_j \theta_k \theta_\ell \psi_{j,k,\ell}(\mathbf{x}) + \cdots \tag{5}$$

The higher-order terms of the expansion are characterized by their own set of features, e.g. $\{\psi_{j,k} : \mathcal{X} \to \mathbb{R}\}_{j,k=1}^{p}$ for the second order term. The elements of such sets are unique up to a permutation of their variables, thus the terms $1/2!, 1/3!, ...$ compensate the multiple counting of such elements in Equation 5. The term $\epsilon \ll 1$ adjusts the contribution of the nonlinear terms. If the model is truncated to the second term it is denoted as *quadratic model*. In such a case, the loss function is quartic, thus we cannot find analytically the optimal parameters as in the linear regression. The dynamic of such a model is described by

$$f(\mathbf{x}, \boldsymbol{\theta} + d\boldsymbol{\theta}) \tag{6}$$

$$= f(\mathbf{x},\boldsymbol{\theta}) + \sum_{j=1}^{p} d\theta_j \left[\phi_j(\mathbf{x}) + \epsilon \sum_{k=1}^{p} \theta_j \psi_{j,k}(\mathbf{x}) \right] + \frac{\epsilon}{2} \sum_{j,k=1}^{p} d\theta_j d\theta_k \psi_{j,k}(\mathbf{x}) \tag{7}$$

$$= f(\mathbf{x},\boldsymbol{\theta}) + \sum_{j=1}^{p} d\theta_j \phi_j^E(\mathbf{x};\boldsymbol{\theta}) + \frac{\epsilon}{2} \sum_{j,k=1}^{p} d\theta_j d\theta_k \psi_{j,k}(\mathbf{x}) \tag{8}$$

where ϕ^E are *effective feature functions*, i.e. features that depend on, and evolve with, the model parameters, which are learnt during the optimization phase. This behaviour can be generalized to consider terms of even higher orders: the presence of order n terms make the feature functions of order $n-1$, effectively, which may further influence the lower order terms. Models having effective feature functions have *feature learning* capabilities. A *deep learning model* is both capable of feature learning and composed of several nonlinear modules arranged in a hierarchical fashion [38]; such that differing layers can follow differing (albeit hierarchically conditioned) gradient paths.

Turning to QNNs, the quantum model

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathrm{Tr}[\rho_{\mathbf{x},\boldsymbol{\theta}} O], \tag{9}$$

where $\rho_{\mathbf{x},\boldsymbol{\theta}} = V^\dagger(\boldsymbol{\theta})U_\phi^\dagger(\mathbf{x})\,|0\rangle\langle 0|\,U_\phi(\mathbf{x})V(\boldsymbol{\theta})$, and O Hermitian observable, is a linear model in the space of density matrices of the quantum system $\mathcal{H}$: the trace operation $\mathrm{Tr}\big[A^\dagger B\big]$ is an inner product for the space of matrices $\mathbb{C}^{k\times k}$. Such a property implies that the construction of a layer-wise architecture for v, i.e. $v(\boldsymbol{\theta}) = \prod_i V_i(\boldsymbol{\theta})$ effectively collapses to a single operation: this may add more degrees of freedom to the linear transformation[3] but cannot make the model nonlinear in $\mathcal{H}$.

However, in terms of the predictor function $f(\mathbf{x};\boldsymbol{\theta})$, the quantum model does not necessarily fit the form set out Equation 3 since the parameters of the QNN model, namely the angle of rotation operation (in the form of imaginary exponential function), are subject to the trace operation. Thus, for example, consider a single-qubit quantum model acting on a single input $\mathbf{x} \in \mathbb{R}^1$, depending on a single parameter $\boldsymbol{\theta} \in \mathbb{R}^1$, with feature map $U_\phi(\mathbf{x}) = \exp(-ix\sigma_x)$, variational form $V(\boldsymbol{\theta}) = \exp(-i\theta\sigma_x)$ and measurement operator i $O = \sigma_z$, in which case $f(\mathbf{x};\boldsymbol{\theta})$ has the form:

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathrm{Tr}\left[\begin{pmatrix} \cos^2(\theta+x) & -i\sin(\theta+x)\cos(\theta+x) \\ \frac{1}{2}i\sin(2(\theta+x)) & \sin^2(\theta+x) \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\right] = \cos(2(\theta+x)) \quad (10)$$

which is nonlinear in its weights. Clearly, if we were to consider a model other than a QNN then the predictor function would change, for example as in [29], however it does not alter our argument here.

To recap, a QNN is a linear model in the Hilbert space of the density matrices due to the linearity of the evolution of closed quantum systems. However, its predictor is nonlinear in the parameter $\boldsymbol{\theta}$ since its structure results in a composition of trigonometric functions. This potentially allows a limited degree of representational learning capability if aggregated layer-wise (limited in the sense of applying only to a highly constrained set of activation functions). However, due to the Lie algebraic equivalence of any given sequence of quantum transformations to some single unitary operation in the absence of the trace operation, we are still not able to characterise truly deep models in the quantum domain.

2.3 Characterization of model dynamics through the Neural Tangent Kernel

The output $f(\mathbf{x};\boldsymbol{\theta})$ of a machine learning model trained via (possibly stochastic) gradient descent can be approximated as a first-order Taylor expansion $f(\mathbf{x};\boldsymbol{\theta}) \approx f(\mathbf{x};\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$. Such an approximation allows the representation of machine learners as linear (kernel) models via the Neural Tangent Kernel (NTK, [13]):

$$k_{\mathrm{ntk}}(\mathbf{x},\mathbf{x}';\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} f(\mathbf{x}';\boldsymbol{\theta}) \quad (11)$$

Such a tool has been used in [14] to characterize the dynamics of infinite-width neural networks, in which the NTK is independent of the random initialization and constant in time. On a coarse level of detail, we can assert that

[3] depending on the generators involved and up to a maximum of $4^n - 1$ (where n number of qubits)

model training in *lazy-training regime*, i.e. when the evolution of $\boldsymbol{\theta}(t)$ during the training of the model $f(\mathbf{x}, \boldsymbol{\theta})$ closely follows the tangent path, can be decently approximated by the NTK. A more detailed analysis in [15] has revealed that the NTK is constant if and only if the model is linear (in its parameters). Such a result allows us to quantify the nonlinearity of a model through its Hessian norm of the predictor function: if $\|H_f\| \ll \|\nabla_w f\|$ then the model is nearly linear. This has been used in [11] to analyze the behaviour of the QNNs in the lazy training regime.

3 The Quantum Path Kernel Framework

No extant quantum method is thus able to fully capture the deviations from gradient path linearity manifested by empirically optimal learners in the classical domain. Hence, in order to encompass the concepts of hierarchicality and feature learning in (implicitly kernel-based) quantum machine learning models, we here introduce for the first time in the quantum realm a key idea of Domingo's [19], namely *Path Kernelization*.

Within this paradigm, for any machine learning model $f_{\bar{\boldsymbol{\theta}}}(\mathbf{x})$ whose parameters $\bar{\boldsymbol{\theta}}$ are learned from a set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ by gradient descent via a differentiable loss function, it is possible to express the resulting (i.e. trained) classifier as:

$$f_{\bar{\boldsymbol{\theta}}}(\mathbf{x}) \approx \sum_{i=1}^{n} \alpha_i(\mathbf{x})\, k_{\text{path}}(\mathbf{x}, \mathbf{x}_i; \bar{\gamma}) + \alpha_0(\mathbf{x}) \tag{12}$$

where

$$k_{\text{path}}(\mathbf{x}, \mathbf{x}'; \gamma) = \int_{\gamma} \nabla_\theta f(\mathbf{x}; \boldsymbol{\theta}) \cdot \nabla_\theta f(\mathbf{x}'; \boldsymbol{\theta})\, d\theta \tag{13}$$

is the *Path Kernel*, i.e. the line integral of k_{ntk} over the multidimensional curve representing the evolution of the parameters $\boldsymbol{\theta} = \gamma(t), t \in [0, T]$ during training, with $\bar{\boldsymbol{\theta}} = \bar{\gamma}(T)$. In general, chain rule dependencies arising from the specifics of the architecture of the network will imply hierarchical dependencies among the parameters during learning. The result holds even for stochastic gradient descent optimization, in which case Equation 13 is a stochastic integral. However, it is not immediately clear that this path integration obeys Mercer's conditions; while it is generally true that a convex sum over Mercer kernels is itself a Mercer kernel, the path over which we are integration is here *dependent on the training objects*. We therefore dedicate Appendix A to proving that the Path Kernel is effectively Mercer, and set out the pseudocode for its construction.

It is thus central to our argument to examine the parameter path γ and its morphological evolution. For linear models, assuming a vanilla gradient descent training over a convex loss function $\mathcal{L}$, the parameter path is described by a linear vector $\{(1-t)\boldsymbol{\theta}_0 + t\boldsymbol{\theta}_f \mid t \in \mathbb{R}\}$ where $\boldsymbol{\theta}_0 \in \mathbb{R}^p$ are the parameters at their initialization, and $\boldsymbol{\theta}_f \in \mathbb{R}^p$ are the parameters at their convergence on the (ideally global) minima of $\mathcal{L}$. In such a case, it is immediately possible to check that the derivative of the linear model $\nabla_\theta f$ is independent of θ, and thus that

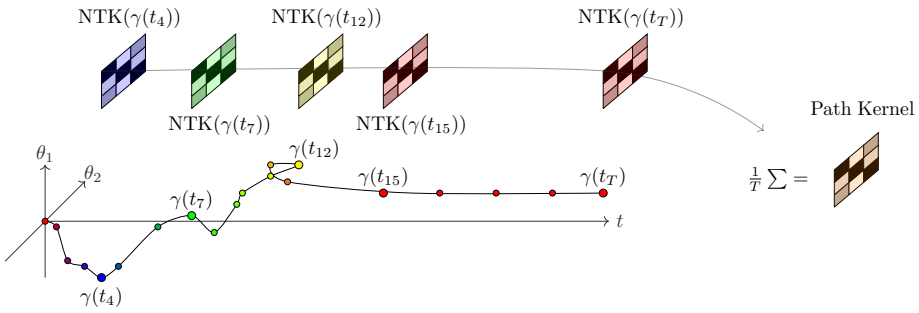


Figure 1: Computation of the Path Kernel. *Bottom left*: A typical parameter trajectory γ is depicted, representing parametric evolution during the training phase. *Top left*: as $\boldsymbol{\theta}$ evolves, it gives rise to differing NTK matrices, corresponding to distinct representations of the data. Such a sequence of matrices thus give rise to a hierarchical stack of representations in the feature learning regime. *Middle*: as the training approaches convergence, subsequent matrices become similar to each other, and thus their corresponding representations are correlated. *Right*: the Path Kernel constitutes the average over these representations.

the NTK is constant. For nonlinear models, the loss function $\mathcal{L}$ may become non-convex and γ is not constrained to be a linear trajectory. In this latter case, both the $\nabla_\theta f$ and the NTK will vary in time.

In this work, we will not focus on the possible role of Path Kernels in approximating nonlinear models. Instead, we shall exploit the intrinsically hierarchical structure of the Path Kernel to implement a hybrid deep machine learning model within a quantum neural network setting. We depict the construction of this object in Figure 1. The parameter trajectory for a nonlinear model is described by a complex, non-straight curve. Each point of the parameter path $\boldsymbol{\theta}_t = \gamma(t)$ may be used to define a new kernel representation for the training data, namely $k_{\text{ntk}}(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}_t)$. We can then define a sequence of kernels stacked in a hierarchical way (whose structure, in passing, resembles the layers of a deep neural network, though this observation is peripheral to the argument being made here). Thus, each new "layer" is a source of representation learning: the new representation (i.e. kernel matrix) is the result of an optimization process that further adapts the previous representation to the given data discrimination problem (which resembles, though is again not equivalent to, classifier boosting).

It thus becomes possible, via explicit substitution for the corresponding Quantum NTK previously defined, to construct a Quantum Path Kernel (QPK)

as follows:

$$k_{\text{qpk}}(\mathbf{x}, \mathbf{x}'; \gamma)$$

$$= \frac{1}{\|\gamma\|} \int_{\gamma} \nabla_{\boldsymbol{\theta}} \langle 0|V^{\dagger}(\mathbf{x})U^{\dagger}(\boldsymbol{\theta})OU(\boldsymbol{\theta})V(\mathbf{x})|0\rangle^{T} \cdot \nabla_{\boldsymbol{\theta}} \langle 0|V^{\dagger}(\mathbf{x}')U^{\dagger}(\boldsymbol{\theta})OU(\boldsymbol{\theta})V(\mathbf{x}')|0\rangle \, d\boldsymbol{\theta} \tag{14}$$

$$= \frac{1}{\|\gamma\|} \int_{0}^{T} \nabla_{\boldsymbol{\theta}} \langle 0|V^{\dagger}(\mathbf{x})U^{\dagger}(\gamma(t))OU(\gamma(t))V(\mathbf{x})|0\rangle^{T} \cdot \nabla_{\boldsymbol{\theta}} \langle 0|V^{\dagger}(\mathbf{x}')U^{\dagger}(\gamma(t))OU(\gamma(t))V(\mathbf{x}')|0\rangle \cdot \gamma'(t) \, dt \tag{15}$$

$$\approx \frac{1}{T} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \langle 0|V^{\dagger}(\mathbf{x})U^{\dagger}(\gamma_t)OU(\gamma_t)V(\mathbf{x})|0\rangle^{T} \cdot \nabla_{\boldsymbol{\theta}} \langle 0|V^{\dagger}(\mathbf{x}')U^{\dagger}(\gamma_t)OU(\gamma_t)V(\mathbf{x}')|0\rangle \tag{16}$$

where Equation 14 defines the QNTK as its classical analog and is equivalent to Equation 15 except for the integration with respect to time. Equation 16 is the discretized version of the preceeding equations, corresponding to actual implementation in a gradient descent-trained model.

The resulting *Quantum Path Kernel* (QPK) is consequently both a quantized version of Domingo's Path Kernel as well as a generalization of the Quantum NTK, one that is implicitly capable of embodying the complex parametric interactions (such as transient parametric co-evolutions) that occur during learning in order to arrive at the final trained model, including those implicated in hierarchal feature learning.

3.1 The Quantum Path Kernel as a generalization of Quantum Neural Tangent Kernel

In interpreting the Quantum Path Kernel as a generalization of QNTK for models exhibiting nonlinear behavior, it may be seen that the QNTK is constant only when independent of $\boldsymbol{\theta}$, in which case:

$$k_{\text{qpk}}(\mathbf{x}, \mathbf{x}'; \gamma) = \frac{1}{\|\gamma\|} \int_{\gamma} k_{\text{qntk}}(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) \, d\boldsymbol{\theta} = k_{\text{qntk}}(\mathbf{x}, \mathbf{x}'; \mathbf{0}) \int_{\gamma} \frac{d\boldsymbol{\theta}}{\|\gamma\|} = k_{\text{qntk}}(\mathbf{x}, \mathbf{x}'; \mathbf{0}). \tag{17}$$

That is, the Quantum Path Kernel becomes identical to the Quantum Neural Tangent Kernel. However, as set out in section 2.2, the particular structure of QNNs will, of itself, give rise to a nonlinear predictor. Thus, in principle, the QNTK would not be expected to be constant in output terms in the finite width regime [11]. However, a close-to-constant behavior can be expected for quantum machine learning models whose training is lazy (i.e. lazy training induced via overparameterization of the QNN, such that the large number of parameters result in a simplified loss landscape [39, 40], leading to rapid convergence to a global minima).

3.2 Decorrelation in feature representation

The Quantum Path Kernel clearly exhibits dependency on the training initialization: different initial parameter values, optimization algorithms or learning rates may lead to differing QPK matrices. In particular, the utilization of 'vanilla' gradient-descent optimization algorithms, with a fixed number of training epochs, may introduced subtle biases in the QPK. For example, if training were to converge rapidly, any contribution between the instance of convergence and the end of the training will be effectively identical and oversampled: this contribution will hence outweight the others, biasing the 'stack' of aggregated kernel matrices toward its final layer, as per 1.

To avoid this, more sophisticated optimization algorithms can be considered. For example, the ADAM optimizer adaptively increases the learning rate in locally convex portions of the loss landscape, leading to fewer similar contributions within the path kernel. Furthermore, it is possible to perturb parameter paths via stochastic, noisy or non-gradient-descent-based optimization techniques in order to decorrelate subsequent contributions to the QPK. Having different, highly decorrelated contributions would allow us to interpret the QPK as an ensemble technique analogous to bootstrap aggregation (bagging) often used for tuning the bias/variance trade off in classical machine learning. (Multiple Kernel Learning [41] might also be used to optimally weight individual contributions over the kernel at the expense of interpretability in path terms) .

Appendix A.3 discuss implementation details for the QPK and its tested variants. We therefore now turn to an examination of the test regime.

4 Experimental evaluation of the Quantum Path Kernel in classifying Gaussian XOR Mixtures

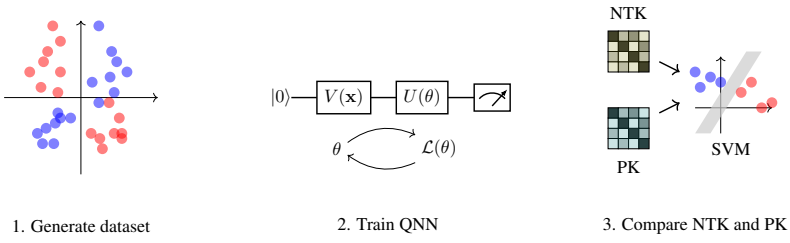


Figure 2: Gaussian XOR Mixture classification experiment workflow.

Machine-learning non-linearities such as those underpinning feature learning in empirical DNNs can thus be feasibly implemented in a quantum setting via the QPK. It remains to demonstrate that this can yield superior generalization performance on plausible quantum devices. Our evaluation, therefore, considers the reference case of the Gaussian XOR Mixture classification problem [42, 43,

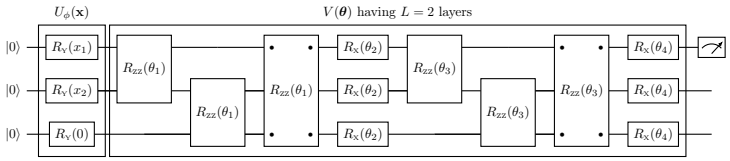


Figure 3: Quantum circuit schematic of the classification model used for $d = 3$ qubits and $L = 2$ layers.

44].

In particular, the Gaussian XOR Mixture classification problem is an important benchmark for highlighting layer-wise learning capabilities of a model (or the lack of them), in that it intrinsically requires a two-layer solution in order to achieve Bayes optimal class separation. Theoretical evidence has shown that kernel methods, in particular those with random features, struggle to accurately classify XOR data vector mixtures [20]. In Appendix B we further analyze the problem, reproducing the results of [20], and proposing an interpretation of the success of feature learning models in tackling the Gaussian XOR Mixture problem.

Our experimental workflow is pictured in Figure 2. Firstly, we generate the dataset for the above described problem. Secondly, we train several QNNs to best fit the generated data. Thirdly, we use the training information to create the QNTK and QPK matrices; the latter are used to train a kernel machine (specifically the Support Vector Machine) to obtain final classifications. Then, our analysis begins with convergence study of the QNNs with an increasing number of layers, to highlight the effect of architectural parametrization in QNNs. Finally, we compare the performances of the QNTK and QPK approaches in terms of testing and training accuracy. The simulation details are shown in Appendix C.

4.1 Experimental Setup

The ground truth Gaussian XOR Mixture dataset is specified by d the dimensionality of the features, $d' \leq d$ the number of non-zero features representing the multidimensional Gaussian XOR Mixture, $\bar{\epsilon}$ the variance of the Gaussian noise, and n the number of data points; it is composed as follows:

$$\mathcal{D}_{d,d',\bar{\epsilon},n} = \left\{ \begin{array}{r} ([x_1 + \epsilon_1, ..., x_{d'} + \epsilon_{d'}, 0, ..., 0]^T, y_i) \\ \in \mathbb{R}^d \times \{\pm 1\} \end{array} \right\}_{j=1}^{n} \tag{18}$$

where $x_i \sim \{\pm 1\}$, $\epsilon_i \sim \mathcal{N}(0, \bar{\epsilon})$ for $i = 1, ..., d'$, and $y_i = \prod_{i=1}^{d'} x_i$. Such a dataset is optimally classified via the oracle function

$$f_{\text{oracle}}(\mathbf{x}) = \prod_{i=1}^{d'} x_i. \tag{19}$$

We generate multiple datasets $\mathcal{D}_{d,d',\epsilon,n}$ having feature dimensionality ranging in $d = 2, 3, ..., 10$, noise ranging in $\epsilon = 0.1, 0.2, ..., 1.0$, number of non-zero features fixed to $d' = 2$, and number of elements fixed at $n = 32$. Then, each dataset has been randomly partitioned into a training set $\mathcal{D}_{\text{train}}$ and a testing set $\mathcal{D}_{\text{test}}$.

Each dataset is processed by a distinct quantum neural network, each sharing the same structure described by:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \text{Tr}[\rho_{\mathbf{x},\boldsymbol{\theta}} O] = \text{Tr}[V^\dagger(\boldsymbol{\theta}) U_\phi^\dagger(\mathbf{x}) \rho_0 U_\phi(\mathbf{x}) V(\boldsymbol{\theta}) O] \tag{20}$$

with data encoding:

$$U_\phi(\mathbf{x}) = \prod_{j=1}^{d} \exp\Big\{-i\, x_j \sigma_y^{(i)}\Big\} \tag{21}$$

such that the trainable ansatz is described:

$$V(\boldsymbol{\theta}) = \prod_{j=1}^{L} \exp\Big\{-i\, \theta_{2i+1} \sigma_x^{(j)}\Big\} \exp\Big\{-i\, \theta_{2i} \sigma_z^{(j)} \otimes \sigma_z^{(j+1 \bmod d)}\Big\} \tag{22}$$

with the L hyperparameter representing the number of layers of the model. Finally, the observable is $O = \sigma_z^{(0)}$.

This data encoding is been chosen for its simplicity: the encoding of one feature for each qubit results in a constant-depth circuit. The choice of the trainable ansatz, though, is particularly important: the underlying functional transformation has the potential to be affected by barren plateau issues if it is too expressive [8], for example when the parametric transformation is able to approximate any arbitrary unitary matrix. The expressibility of a quantum transformation can be examined using Lie-algebraic tools as shown in [45]. Among the class of unitaries that are non-maximally expressive, we have selected a specific form that has empirically demonstrated favorable trainability as detailed in [40, Fig. 7a]. The choice of the observable is also guided by the necessity of avoiding the barren plateau issue. According to [46], global observables are likely to exhibit vanishing gradients; we thus apply the simplest possible classifier observable acting on a single qubit. The circuit is pictured in Figure 3. In our experiment, the observed qubit is the uppermost; although any other qubit choice would result in a similar predictor due to the symmetric structure of the circuit.

Each dataset is processed with the above described QNN employing a number of layers ranging from $L = 1$ to 20. According to [47], the QNNs should be initialized at $\boldsymbol{\theta} = \mathbf{0}$ to avoid further trainability issues. However, we do not need to consider such initialization strategy for the variational unitary since the previous expedients were sufficient to allow successful training. Thus, the parameters θ_j are sampled from a standard normal distribution. Each QNN is trained using the stochastic gradient-descent algorithm ADAM for 1000 epochs using an initial (adaptive) learning rate $\eta = 0.1$. The loss function is either BCE or MSE and, for the sake of simplification, the batch size is equal to the total cardinality of the training set.

In the experimental setup described above, we study, both epoch-wise and depth-wise, the effect induced by different initialization parameters on the convergence of the loss function during training.

4.2 Results

We evaluate the depthwise convergence characteristics of the respective $f(\mathbf{x}; \boldsymbol{\theta})$ models in terms of the corresponding accuracies of the Quantum Path Kernel and Quantum NTK under SVM final classification. Of particular interest is evaluating the closeness of models to the *lazy training* regime, indicative of the model being near to linear. Lazy training, in classical machine learning, typically occurs for very wide neural networks with the loss decreasing to zero exponentially rapidly, while network parameters stay close to their initialization values throughout training. In the current context, this would correspond to the Quantum Path Kernel collapsing to the Quantum Neural Tangent Kernel, and we would anticipate convergent classification performances for the two approaches.

We therefore evaluate training loss for each of the QNN models over the respective training epochs with an increasing number of QNN layers $L = 1, ..., 20$. This will be used to determine proximity to the lazy training regime (i.e. identifying if the QNN converges exponentially fast to zero loss). We additionally plot the norm difference between the parameters during training compared to their initialization values. These will be used to determine the extend to which parameters vary from their initialization, indicative the training richness of models in the *non*-lazy training regime.

We are also interested in determining the robustness of the classifiers to stochastic noise influences during training and their corresponding resilience to overfitting (or the extent to which *benign overparameterization* [39] effects exists), measured in terms of generalization performance. Therefore, the above evaluations are repeated for datasets additively noise-perturbed in an increasing signal-to-noise ratio.

Finally, we are interested in comparing the generalization performances of our approach to that of the QNTK. For this, we evaluate test accuracy score for the QPK and QNTK, against the oracle. Superior performance of the QPK, in solving the Gaussian XOR Mixture problem, will be taken to be indicative of superior ability to replicate the layerwise feature-learning capability of classical multilayer networks.

4.2.1 Depthwise convergence characteristics

Figure 4 indicates the respective convergence behavior of the evaluated quantum machine learning models with respect to the increasing number of layers. Column 1 has illustrative samples from the training distribution with row-wise decrements in the signal-to-noise ratio, column 2 gives the corresponding loss curves during training, and column 3 indicates the corresponding change in the

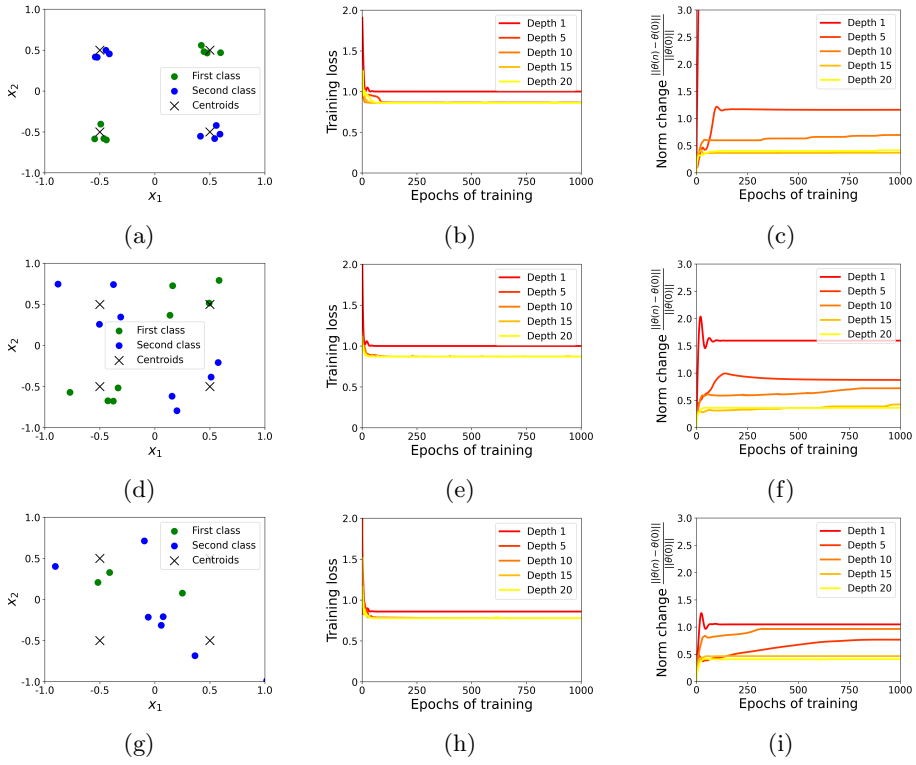
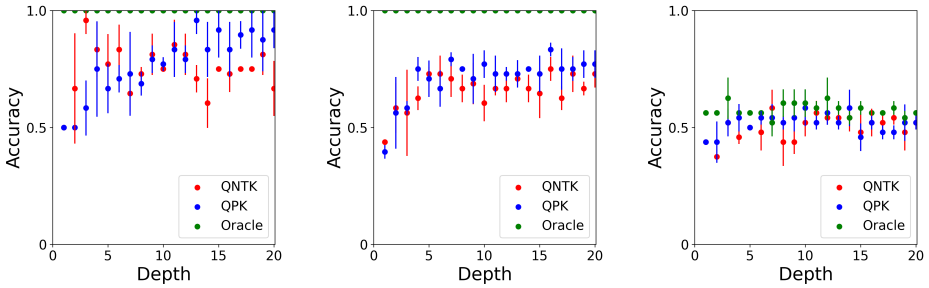


Figure 4: Behavior of the quantum machine learning models $f(\mathbf{x}; \boldsymbol{\theta})$ over the training phase. (4a) illustrates the training dataset for the parameter selection $d = 4, \epsilon = 0.1$; (4b) shows the evolving loss for each of the 20 evaluated depthwise models ($L = 1, ..., 20$) during training; (4c) quantifies the deviation of the parameter vector from its initialization. (4d-4e-4f) show the corresponding information when $d = 4, \epsilon = 0.4$; (4g-4h-4i) for $d = 4, \epsilon = 1.0$.



(a) (b) (c)

Figure 5: Respective test accuracy scores for the Quantum Path Kernel model, the Quantum NTK and the oracle. Error bars represents the standard deviation over three (otherwise identical) experiments having parametric specifications $d = 4, \epsilon = 0.1$, (5a); $d = 4, \epsilon = 0.4$ (5b); $d = 4, \epsilon = 1.0$ (5c).

magnitude of the parameter vector offset from initialization:

$$\frac{\|\boldsymbol{\theta}(n) - \boldsymbol{\theta}(0)\|}{\|\boldsymbol{\theta}(0)\|} \tag{23}$$

where $\boldsymbol{\theta}(0)$ is the value of the parameters at their initialization, and $\boldsymbol{\theta}(n)$ is their value at the n-th epoch.

It is evident that none of the models reach the interpolation threshold [48] - i.e. the point at which the training data is fitted perfectly with zero training error. To fit the training dataset we would need at least 32 parameters (2 non-zero coordinates per point per 16 points). However, we are not able to reach the interpolation threshold even in the deepest configuration with a total of 40 parameters. This behaviour is expected by the choice of a parametrically-constrained U in effect acting as a form of regularisation. As in the classical DNN case, an increasing number of parameters results in a decrease in the loss (Figure 4b-4e-4h), and in an increase in the proximity between the parameter vectors and their initialization (Figure 4c-4f-4i).

We can conclude that none of the QNN models exhibit evidence of lazy training. In particular, while models having a higher number of parameters do indeed converge more rapidly, parameters are nonetheless varying substantially from their initialization. This behaviour is even more noticeable in the smaller models, with a norm difference oscillating substantially prior to the convergence. Such non-trivial training is suggestive of the QPK differing largely from the QNTK in its training characteristics.

4.2.2 Test and train accuracy of the Quantum Path Kernel verses the Quantum NTK

Figure 5 indicates the corresponding test accuracies, measuring how well the respective models generalize to unseen data. While the QPK and Quantum NTK

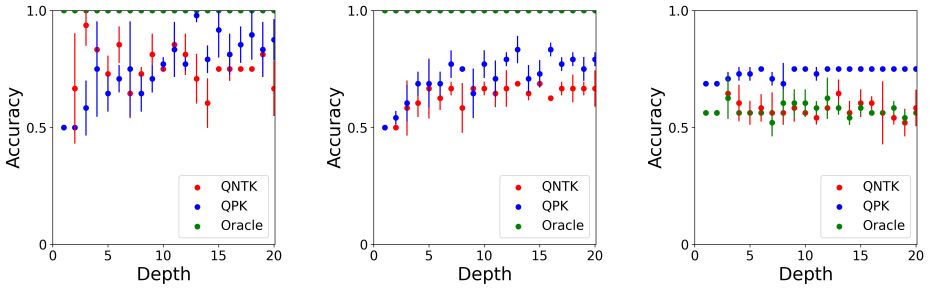


Figure 6: Respective training accuracies of the Quantum Path Kernel model, the Quantum NTK and the oracle. Error bars represents the standard deviation over three (otherwise identical) experiments having specifications $d = 4, \epsilon = 0.1$, (6a); $d = 4, \epsilon = 0.4$ (6b); $d = 4, \epsilon = 1.0$ (6c).

models both perform similarly at low signal-to-noise ratios, it is particularly striking to observe the outperformance of the QPK over the Quantum NTK with increasing hierarchical depth at the highest signal-to-noise setting..

Figure 6 indicates the training accuracy with depth at the point of convergence. It may be observed that the QPK exhibits lower loss than the Quantum NTK across the full signal-to-noise range, with the effect becoming more marked at higher noise levels (ultimately over-fitting relative to the noise-free oracle in panel c), consistent with the expectation that QPK has a lower bias than the Quantum NTK.

In sum, results confirm the anticipated improvement in performance for the QPK over the QNTK in the Gaussian XOR mixture setting.

5 Conclusion and Further Work

We have introduced the Quantum Path Kernel as a mechanism for incorporating key complex classical multi-layer network learning behaviors, in particular hierarchical feature learning, within quantum neural networks via an appropriately expressive kernelization of the training process. We evaluate our approach on the Gaussian XOR mixture classification problem, a straightforward benchmark of multilayer learning capacity that requires a minimum two-layer solution in order to approach Bayes optimally. Experimental results indicate superior generalization performance relative to the Quantum NTK, an advantage which is especially pronounced in high-depth, low signal-to-noise settings.

We have shown theoretically that the Quantum Path Kernel converges to the Quantum NTK only in the lazy training regime, i.e. when the training loss decreases to zero exponentially fast whilst model parameters stay close to their initializations across training. Such behaviour is classically seen in infinite-wide neural networks, whose behaviour is then close to that of a linear model.

Our experiments, by contrast, indicate that QNNs do not operate in the linear regime.

We have discussed, though do not evaluate in the current paper, the potential for using stochastic, noisy or non-gradient descent based optimization techniques to artificially perturb parameter paths within the QPK in order to implicate more decorrelated feature representations. We, furthermore, propose in future to extend the QPK approach via weighting of individual kernel representations in a more heuristic way, for example via Multiple Kernel Learning. We have also referred in passing to the interpretation of the QPK as an ensemble method due to the averaging operation over its kernel matrices. This will be explored more fully in future investigations.

References

[1] Peter Wittek. *Quantum machine learning: what quantum computing means to data mining*. Academic Press, 2014.

[2] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015.

[3] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R McClean. Power of data in quantum machine learning. *Nature communications*, 12(1):1–9, 2021.

[4] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, 2021.

[5] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. A rigorous and robust quantum speed-up in supervised machine learning. *Nature Physics*, 17(9):1013–1017, 2021.

[6] Hsin-Yuan Huang, Michael Broughton, Jordan Cotler, Sitan Chen, Jerry Li, Masoud Mohseni, Hartmut Neven, Ryan Babbush, Richard Kueng, John Preskill, et al. Quantum advantage in learning from experiments. *Science*, 376(6598):1182–1186, 2022.

[7] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), 11 2018.

[8] Zoë Holmes, Kunal Sharma, Marco Cerezo, and Patrick J Coles. Connecting ansatz expressibility to gradient magnitudes and barren plateaus. *PRX Quantum*, 3(1):010313, 2022.

[9] Andrea Skolik, Jarrod R. McClean, Masoud Mohseni, Patrick van der Smagt, and Martin Leib. Layerwise learning for quantum neural networks. *Quantum Machine Intelligence*, 3(1), 1 2021.

[10] Maria Schuld. Supervised quantum machine learning models are kernel methods, 2021.

[11] Junyu Liu, Francesco Tacchino, Jennifer R Glick, Liang Jiang, and Antonio Mezzacapo. Representation learning via quantum neural tangent kernels. *PRX Quantum*, 3(3):030323, 2022.

[12] Kunal Sharma, Marco Cerezo, Lukasz Cincio, and Patrick J Coles. Trainability of dissipative perceptron-based quantum neural networks. *Physical Review Letters*, 128(18):180505, 2022.

[13] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[14] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in Neural Information Processing Systems*, 32, 2019.

[15] Chaoyue Liu, Libin Zhu, and Misha Belkin. On the linearity of large nonlinear models: when and why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 33:15954–15964, 2020.

[16] Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. When do neural networks outperform kernel methods? *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124009, 2021.

[17] Minshuo Chen, Yu Bai, Jason D Lee, Tuo Zhao, Huan Wang, Caiming Xiong, and Richard Socher. Towards understanding hierarchical learning: Benefits of neural representations. *Advances in Neural Information Processing Systems*, 33:22134–22145, 2020.

[18] Norihito Shirai, Kenji Kubo, Kosuke Mitarai, and Keisuke Fujii. Quantum tangent kernel. *arXiv preprint arXiv:2111.02951*, 2021.

[19] Pedro Domingos. Every model learned by gradient descent is approximately a kernel machine. *arXiv preprint arXiv:2012.00152*, 2020.

[20] Maria Refinetti, Sebastian Goldt, Florent Krzakala, and Lenka Zdeborová. Classifying high-dimensional gaussian mixtures: Where kernel methods fail and neural networks succeed. In *International Conference on Machine Learning*, pages 8936–8947. PMLR, 2021.

[21] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pages 2232–2241. PMLR, 2019.

[22] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *Advances in Neural Information Processing Systems*, 32, 2019.

[23] Yu Bai, Ben Krause, Huan Wang, Caiming Xiong, and Richard Socher. Taylorized training: Towards better approximation of neural network training at finite width. *arXiv preprint arXiv:2002.04010*, 2020.

[24] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586, 2014.

[25] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Simulating a perceptron on a quantum computer. *Physics Letters A*, 379(7):660–663, 3 2015.

[26] Yudong Cao, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. Quantum neuron: an elementary building block for machine learning on quantum computers. *arXiv preprint arXiv:1711.11240*, 2017.

[27] Wei Hu. Towards a real quantum neuron. *Natural Science*, 10(3):99–109, 2018.

[28] Kaitlin Gili, Mykolas Sveistrys, and Chris Ballance. Introducing non-linearity into quantum generative models. *arXiv preprint arXiv:2205.14506*, 2022.

[29] Francesco Tacchino, Chiara Macchiavello, Dario Gerace, and Daniele Bajoni. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1):1–8, 2019.

[30] Kunal Sharma, Marco Cerezo, Lukasz Cincio, and Patrick J Coles. Trainability of dissipative perceptron-based quantum neural networks. *Physical Review Letters*, 128(18):180505, 2022.

[31] Naixu Guo, Kosuke Mitarai, and Keisuke Fujii. Nonlinear transformation of complex amplitudes via quantum singular value transformation. *arXiv preprint arXiv:2107.10764*, 2021.

[32] Zoë Holmes, Nolan Coble, Andrew T Sornborger, and Yiğit Subaşı. On nonlinear transformations in quantum computation. *arXiv preprint arXiv:2112.12307*, 2021.

[33] Ammar Daskin. A simple quantum neural net with a periodic activation function. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2887–2891. IEEE, 2018.

[34] Steven Weinberg. Precision tests of quantum mechanics. *Phys. Rev. Lett.*, 62:485–488, 1 1989.

[35] Daniel S. Abrams and Seth Lloyd. Nonlinear quantum mechanics implies polynomial-time solution for *NP*-complete and # *P* problems. *Phys. Rev. Lett.*, 81:3992–3995, 11 1998.

[36] Sofiene Jerbi, Lukas J Fiderer, Hendrik Poulsen Nautrup, Jonas M Kübler, Hans J Briegel, and Vedran Dunjko. Quantum machine learning beyond kernel methods. *arXiv preprint arXiv:2110.13162*, 2021.

[37] Daniel A. Roberts, Sho Yaida, and Boris Hanin. *The Principles of Deep Learning Theory*. Cambridge University Press, 2022.

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[39] Evan Peters and Maria Schuld. Generalization despite overfitting in quantum machine learning models. *arXiv preprint arXiv:2209.05523*, 2022.

[40] Martin Larocca, Nathan Ju, Diego García-Martín, Patrick J. Coles, and M. Cerezo. Theory of overparametrization in quantum neural networks, 2021.

[41] Mehmet Gönen and Ethem Alpaydın. Multiple kernel learning algorithms. *The Journal of Machine Learning Research*, 12:2211–2268, 2011.

[42] Zeyu Deng, Abla Kammoun, and Christos Thrampoulidis. A model of double descent for high-dimensional binary linear classification. *arXiv preprint arXiv:1911.05822*, 2019.

[43] Xiaoyi Mai and Zhenyu Liao. High dimensional classification via regularized and unregularized empirical risk minimization: Precise error and optimal loss. *arXiv preprint arXiv:1905.13742*, 2019.

[44] Marc Lelarge and Léo Miolane. Asymptotic bayes risk for gaussian mixture in a semi-supervised setting. In *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 639–643. IEEE, 2019.

[45] Martin Larocca, Piotr Czarnik, Kunal Sharma, Gopikrishnan Muraleedharan, Patrick J Coles, and M Cerezo. Diagnosing barren plateaus with tools from quantum optimal control. *Quantum*, 6:824, 2022.

[46] Marco Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature communications*, 12(1):1–12, 2021.

[47] Edward Grant, Leonard Wossnig, Mateusz Ostaszewski, and Marcello Benedetti. An initialization strategy for addressing barren plateaus in parametrized quantum circuits. *Quantum*, 3:214, 2019.

[48] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[49] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3681–3688, 2019.

[50] Yilan Chen, Wei Huang, Lam Nguyen, and Tsui-Wei Weng. On the equivalence between neural network and support vector machine. *Advances in Neural Information Processing Systems*, 34:23478–23490, 2021.

[51] Noureddine El Karoui. The spectrum of kernel random matrices. *The Annals of Statistics*, 38(1):1–50, 2010.

[52] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

[53] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

A Theoretical and implementational details of the Path Kernel in the classical machine learning domain

The Path Kernel was introduced in [19] as a means of replicating arbitrary gradient-descent based machine learning models in the form of kernel machines, under some weak assumptions. The Path Kernel is consequently of inherent interest in the theory of classical machine learning in that it grants a further layer of interpretability to models, including those, such as the neural networks, that often lacks this [49]. In contrast, kernel machines permit a clear interpretation of prediction functions in terms of linear combinations of data in the training set as a consequence of the Representer Theorem. In particular, [19, Theorem 1] indicates that the model $f(\mathbf{x}; \mathbf{w}) : \mathbb{R}^D \times \mathbb{R}^P \to \mathbb{R}$ (with D the dimensionality of the data and P the number of model parameters) can be rewritten:

$$f(\mathbf{x}; \bar{\mathbf{w}}) = \sum_{i=1}^{m} w_i(\mathbf{x})\, K_{\text{path}}(\mathbf{x}, \mathbf{x}_i; \bar{\gamma}) + w_0(\mathbf{x}). \tag{24}$$

where

$$K_{\text{path}} : \mathbb{R}^D \times \mathbb{R}^D \times ([0, T] \to \mathbb{R}^P) \to \mathbb{R}^D \tag{25}$$

$$K_{\text{path}}(\mathbf{x}, \mathbf{x}_i, \gamma) = \int_0^T K_{\text{tang}}(\mathbf{x}, \mathbf{x}_i, \gamma(t)) \cdot \gamma'(t) dt \tag{26}$$

is the Path Kernel, a parametric kernel function (this parameterization has been rendered explicit in current formulation). In this case, $\bar{\gamma} : [0, T] \to \mathbb{R}^P$ is the parameter path as detailed in Section 3 with a terminal parameter value $\bar{\gamma}(T) = \bar{\mathbf{w}}$. The Neural Tangent Kernel can also be expressed as a parametric kernel,

$$K_{\text{tang}} : \mathbb{R}^D \times \mathbb{R}^D \times \mathbb{R}^P \to \mathbb{R} \tag{27}$$

$$K_{\text{tang}}(\mathbf{x}, \mathbf{x}_i; \mathbf{w}) = \nabla_{\mathbf{w}} f(\mathbf{x}, \mathbf{w}) \cdot \nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}). \tag{28}$$

Equation 24 holds under the proviso that f is differentiable in $\mathbf{w}$, and trained via Gradient Descent (GD) for the given training dataset $\{(\mathbf{x}_i, y_i^*)\}_{i=1}^m \subseteq \mathbb{R}^D \times \mathbb{R}$ using the convex differentiable loss function $L(w) = \sum_{i=1}^M \ell(f(\mathbf{x}_i), y_i^*)$.

Equation 24 differs from a linear model due to the explicit dependency of the data $\mathbf{x}$ in the weights w_i, and it remains a matter of discussion as whether the path kernel in fact represents a more generalized model class than that of kernel machines (although it is clearly equivalent for infinitely small learning rates [50]). This debate need not concern us for the present purposes, where the intent is to obtain a class of models capable of representing the network gradient trajectory in a manner expressible on current quantum computers.

As the Path Kernel is not widely deployed in practical machine learning, we detail here some of its properties. In A.1 we prove the Path Kernel is a Mercer Kernel. In A.2 we briefly comment on the proof of [19, Theorem 1]. In A.3 we demonstrate a numerical implementation of the Path Kernel.

A.1 Path Kernel is a Mercer Kernel

Given any $\bar{\gamma}$, the function $\bar{K}_{\text{path}}(\mathbf{x}, \mathbf{x}') = K_{\text{path}}(\mathbf{x}, \mathbf{x}'; \bar{\gamma})$ is a positive definite or Mercer kernel on $\mathbb{R}^D$. A Mercer kernel satisfies

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{29}$$

for all sequence of elements $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^D$ and constants $c_1, ..., c_n \in \mathbb{R}$.

It is straightforward to demonstrate that such a condition is valid of the Path Kernel. Firstly, $\bar{K}_{\text{tang}}(\mathbf{x}, \mathbf{x}_i) = K_{\text{tang}}(\mathbf{x}, \mathbf{x}_i; \mathbf{w})$ is a positive definite function for any $\mathbf{w}$ in consequence of the positive definiteness of the Gram matrix of inner products in its parameter space $\mathbb{R}^P$. Secondly, since both the positive combination and the infinitesimal limit of combinations of positive definite kernels still satisfy the Mercer condition, then the preceding is immediately valid for the Path Kernel in both its discrete and continuous formulations.

A.2 Comment on Theorem 1 in Domingo's work

In this section we comment on [19, Theorem 1] in order to highlight some of its limitations. The dynamics of any predictor under training via gradient descent

may be described by a first-order non-homogeneous differential equation:

$$\frac{df(\mathbf{x};\mathbf{w})}{dt} = -\sum_{j=1}^{P} \frac{\partial f}{\partial w_j} \cdot \frac{\partial L}{\partial w_j}. \tag{30}$$

where $f(\mathbf{x};\mathbf{w}) : \mathbb{R}^D \times \mathbb{R}^P$ and L is the convex differentiable loss function. We can describe these predictor dynamics over training in terms of the Tangent Kernel:

$$\begin{aligned}
\frac{df(\mathbf{x};\mathbf{w}(t))}{dt} &= \sum_{j=1}^{d} \frac{\partial f(\mathbf{x};\mathbf{w})}{\partial w_j} \cdot \frac{dw_j}{dt} && (31)\\
&= \sum_{j=1}^{d} \frac{\partial f(\mathbf{x};\mathbf{w})}{\partial w_j} \cdot \left(-\frac{\partial L(w(t))}{\partial w_j} \right) && (32)\\
&= \sum_{j=1}^{d} \frac{\partial f(\mathbf{x};\mathbf{w})}{\partial w_j} \cdot \left(-\sum_{i=1}^{m} \frac{\partial \ell(y_i^*, f(x_i;w))}{\partial w_j} \right) && (33)\\
&= \sum_{j=1}^{d} \frac{\partial f(\mathbf{x};\mathbf{w})}{\partial w_j} \cdot \left(-\sum_{i=1}^{m} \frac{\partial \ell(y_i^*, y_i)}{\partial y_i} \frac{\partial f(x_i;w)}{\partial w_j} \right) && (34)\\
&= -\sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \sum_{j=1}^{d} \frac{\partial f(\mathbf{x};\mathbf{w})}{\partial w_j} \frac{\partial f(\mathbf{x}_i;\mathbf{w})}{\partial w_j} && (35)\\
&= -\sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \nabla_w f(\mathbf{x};\mathbf{w}) \cdot \nabla_w f(\mathbf{x}_i;\mathbf{w}) && (36)\\
&= -\sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} K_{\text{tang}}(\mathbf{x}, \mathbf{x}_i; \mathbf{w}) && (37)
\end{aligned}$$

In the limit $\epsilon \to 0$ we obtain:

$$f(\mathbf{x}) \;=\; f(\mathbf{x};\gamma(T)) \;=\; f(\mathbf{x};\gamma(0)) - \int_0^T \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \; K_{\text{tang}}(\mathbf{x}, \mathbf{x}_i; \gamma(t)) \; dt. \tag{38}$$

Such a function cannot be straightforwardly represented as a linear model. However, by multiplying and dividing by the Path Kernel itself we obtain the following equation, at the cost of introducing a dependency of $\mathbf{x}$ in the model parameters:

$$\begin{aligned}
f(\mathbf{x};\gamma(T)) &= f(\mathbf{x};\gamma(0)) + \sum_{i=1}^{m} \left(-\frac{\int_0^T \frac{\partial \ell}{\partial y_i} K_{\text{tang}}(\mathbf{x}, \mathbf{x}_i; \gamma(t)) dt}{K_{\text{path}}(\mathbf{x}, \mathbf{x}_i; \gamma)} \right) K_{\text{path}}(\mathbf{x}, \mathbf{x}_i; \gamma)\\
&= f(\mathbf{x};\gamma(0)) + \sum_{i=1}^{m} \alpha_i(\mathbf{x}) \, K_{\text{path}}(\mathbf{x}, \mathbf{x}_i; \gamma). && (39)
\end{aligned}$$

procedure CreateNeuralTangentKernel
Input: predictor function $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$, data set $\{\mathbf{x}_i \in \mathbb{R}^d\}_{i=0}^{n-1}$, parameter value $w \in \mathbb{R}^p$.
Output: real symmetric matrix $n \times n$ representing the neural tangent kernel of f over the given dataset.
▷ Start procedure
$M \leftarrow$ zero filled $n \times p$ matrix
for $i \in 0, ..., n-1$ **do**
$M[i] \leftarrow \nabla f(\mathbf{x}_i, w)$ ▷ The array has size p.
return MM^T ▷ Matrix size: $(n \times p)(p \times n) = (n \times n)$

Figure 7: Pseudo-code for the Neural Tangent Kernel formulation.

procedure CreatePathKernel
Input: predictor function $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$, data set $\{\mathbf{x}_i \in \mathbb{R}^d\}_{i=0}^{n-1}$, parameter path $\gamma \in \mathbb{R}^{p \times t}$ obtained during the gradient descent-based training phase.
Output: real symmetric matrix $n \times n$ representing the Path kernel of f over the given dataset.
▷ Start procedure
$M \leftarrow$ zero filled t elements array
for $j \in 0, ..., t-1$ **do**
$w \leftarrow \gamma[j]$
$M \leftarrow M +$ CreateNeuralTangentKernel$(f, \{\mathbf{x}_i\}_{i=0}^{n-1}, w)$
return $\frac{1}{t}M$

Figure 8: Pseudo-code for the Path Kernel formulation.

Various works have suggested that imposing stronger assumptions on training can remove the dependency of $\mathbf{x}$ in the model parameters. For example, the authors in [50] achieve this by imposing a requirement that the loss derivative is of constant sign during training.

A.3 Numerical calculation of the Path Kernel

We can calculate the value of the Path Kernel by approximating the integral with a direct sum

$$K_{\text{path}}(\mathbf{x}, \mathbf{x}_i, \gamma) = \int_0^T K_{\text{tang}}(\mathbf{x}, \mathbf{x}_i, \gamma(t)) \cdot \gamma'(t)\, dt \approx \sum_{t=0}^{T-1} K_{\text{tang}}(\mathbf{x}, \mathbf{x}_i, \gamma[t]) \quad (40)$$

The implementation details are reported in the following pseudo-code listings. In Figure 7 we indicate how to calculate the Neural Tangent Kernel of the predictor f once the parameter value w is fixed. In particular, the gradient can be calculated with the finite difference method or, if the predictor is implemented with a Quantum Neural Network, with the parameter-shift rule.

The procedure for calculating the Path Kernel is shown in Figure 8 and uses the Neural Tangent Kernel to calculate the individual contribution of each training epoch and thereafter calculates the average kernel matrix pointwise.

In Section 3.2 we discussed the potential significance of decorrelated features; we here propose a numerical implementation of the *Effective* Path Kernel. In contrast to the original Path Kernel, the Effective Path Kernel seeks to avoid to biasing due to multiple similar kernel contributions. This is especially important if the training has converged signficiantly earlier that the last training epoch: any contribution after convergence has the same Neural Tangent Kernel and will

procedure CreateEffectivePathKernel

Input: predictor function $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$, data set $\{\mathbf{x}_i \in \mathbb{R}^d\}_{i=0}^{n-1}$, number of training epochs t, parameter path $\gamma \in \mathbb{R}^{p \times t}$ obtained during the gradient descent-based training phase, correlation threshold $C \in [0, 1]$.

Output: real symmetric matrix $n \times n$ representing the Effective Path kernel of f over the given dataset.

$\ell \leftarrow$ CreateEffectivePathKernelRec$(f, \{\mathbf{x}_i\}_{i=0}^{n-1}, \gamma, C, 0, t-1)$

$n \leftarrow$ number of elements in the list ℓ

return $\frac{1}{n}\sum_{i=0}^{n-1} \ell_i$

procedure CreateEffectivePathKernelRec

Input: predictor function $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$, data set $\{\mathbf{x}_i \in \mathbb{R}^d\}_{i=0}^{n-1}$, parameter path $\gamma \in \mathbb{R}^{p \times t}$ obtained during the gradient descent-based training phase, correlation threshold $C \in [0, 1]$, start instant $t_s \in [0, ..., t)$, end instant $t_e \in [0, ..., t), t_s < t_e$.

Output: array of t elements, each one coarsely quantifying the curvature of the parameter path at each training epoch.

▷ Start procedure

if $t_s \geq t_e$ **then**

return [] ▷ Empty list

$M_s \leftarrow$ CreateNeuralTangentKernel$(f, \{\mathbf{x}_i\}_{i=0}^{n-1}, \gamma[t_s])$

$M_e \leftarrow$ CreateNeuralTangentKernel$(f, \{\mathbf{x}_i\}_{i=0}^{n-1}, \gamma[t_e])$

$c \leftarrow$ correlation between M_s and M_e ▷ interpret the matrix as vectors to calculate the correlation, or change evalutation metric, e.g. Frobenius norm

if $|c| > C$ **then**

return $[M_s, M_e]$ ▷ Highly correlated representation

else if $t_s + 1 < t_e$ **then**

$t_m \leftarrow \text{int}(t_s/2 + t_e/2)$

$L \leftarrow$ CreateEffectivePathKernelRec$(f, \{\mathbf{x}_i\}_{i=0}^{n-1}, \gamma, C, t_s, t_m)$

$R \leftarrow$ CreateEffectivePathKernelRec$(f, \{\mathbf{x}_i\}_{i=0}^{n-1}, \gamma, C, t_m + 1, t_e)$

return $[M_s, M_e] \cup L \cup R$ ▷ Concatenate lists

Figure 9: Pseudo-code for the Effective Path Kernel formulation.

increase its relative weight as the number of epochs after convergence increases. Its formulation is given in Figure 9. Both the Path Kernel and Effective Path Kernel can be straightforwardly implemented in parallel over multiple CPUs (or multiple QPUs) for the evaluation of f.

B Numerical evidence for the inability of random feature kernel techniques in solving the Gaussian XOR Mixture classification

In [20] the authors demonstrate that a two-layer-depth neural network with only a small number of neurons can easily outperform kernel methods on the Gaussian Mixture classification problem, under the assumption that the number of training data points $n \to \infty$ is linearly proportional to the dimensionality of the data $d \to \infty$.

We modify Refinetti's experiment for the current purposes to show the same result in a more straightforward way.

We define the two-layer neural network as the function:

$$f_{\text{nn}}(\mathbf{x}; W_1, W_2, W_3, b_1, b_2, b_3) = W_3 \cdot \text{relu}(W_2 \cdot \text{relu}(W_1 \cdot \mathbf{x} + b_1) + b_2) + b_3 \quad (41)$$

parameterized by $W_1 \in \mathbb{R}^{h \times d}, W_2 \in \mathbb{R}^{h \times h}, W_3 \in \mathbb{R}^{1 \times h}, b_1, b2 \in \mathbb{R}^{h \times 1}, b_3 \in \mathbb{R}$, where h is the number of hidden neurons per layer (the number of hidden neurons is here fixed to $h = \lceil \sqrt{d} \rceil$). In our setting, we randomly initialize the weights W_1, W_2, W_3 by sampling the matrix element i.i.d. from a Gaussian of zero mean and unitary variance. The model is then trained using the gradient-descent-based algorithm ADAM for a maximum 1000 epochs with learning rate

0.001 (the model is implemented in Python3 library scikit-learn, with the default configuration).

We define a random feature kernel machine as:

$$k_{\text{rf}}(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle, \qquad \phi(\mathbf{x}) = \text{relu}(W \cdot \mathbf{x}) \tag{42}$$

with the activation weights parameterized by $W \in \mathbb{R}^{f \times d}, w_{i,j} \sim \mathcal{N}(0, 1)$, where f has been chosen such that the number of parameters of the random feature kernel is greater that or equal to the number of parameters in the neural network, thus:

$$f = \frac{(dh + hh + h) + (h + h + 1)}{d}. \tag{43}$$

For $h = \lceil \sqrt{d} \rceil$ we can tightly upper bound f with $f < \lceil \sqrt{d} \rceil + 5$. This kernel function is then fed to a SVM for classification (as implemented in scikit-learn).

We randomly generate the dataset $\mathcal{D}_{d,d',\bar{\epsilon},n}$ as detailed in Section 4.1. The experiment described below consists in comparing the performance of the neural network classifier with variations of the random feature kernel on the dataset $\mathcal{D}_{d,3,\epsilon,16d}$ for data point dimensionality $d = 4, 8, 12, 16, 20$ and noise $\epsilon = 0, 0.1, 0.2, ..., 1.9, 2.0$. We keep the number of non-zero features $d' = 3$, meaning we are effectively classifying 3D Gaussian XOR mixtures, with the number of training vectors of the dataset fixed to be $16d$. The dataset is then randomly split 75% in the training dataset and 25% in the test set. For each dataset, we compare the performances of the oracle with the performances of the best of 10 randomly initialized neural networks and the best of 10 random feature kernels. For each dataset specification, we repeat this procedure 10 times.

In Figure 10 we set out the results of the above described experiments. It may be observed that Neural Networks outperform the kernel approach in each case, with the differential in accuracy increasing with the number of zero-valued features. Refinetti et al. [20] suggest that this difference in performance is accounted for by the fact that random feature kernels in high dimension behave as linear transformations [51].

We have here suggested a complementary interpretation of the results of such experiments. We have shown that the difference of performance between the two models is not uniquely determined by the failure of kernel methods *per se*. In fact, it is determined also by the feature learning capabilities of neural networks; inspecting the evolution of the W_1 parameters during the training of a neural network reveals that elements in W_1 related to the zero-features do indeed go to zero (Figure 11). This results in having all of the hidden neurons (whose number is proportional to $\sqrt{d}$ and thus increasing with the number of features) working adaptively to classify the three discriminatively informative components or features, thereby improving overall performance in contrast to the random feature kernel approach, for which adding feature (and parameters) drastically decreases performance (which is to say the path model outperforms the random feature kernel in this problem by being able to discharge junk features candidates, thus performing feature learning).

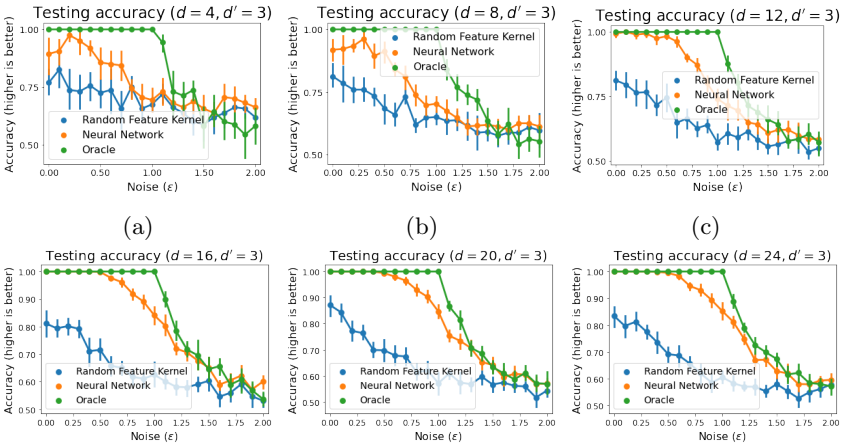


Figure 10: Comparison of the performance of Random Feature Kernel and (2 layer) Neural Networks over the 3D Gaussian XOR Mixture problem with an increasing number of features set to zero. 10a, 10b, 10c, 10d, 10e, 10f have respectively 4, 8, 12, 16, 20, 24 feature per point, the first three being the only non-zero ones.

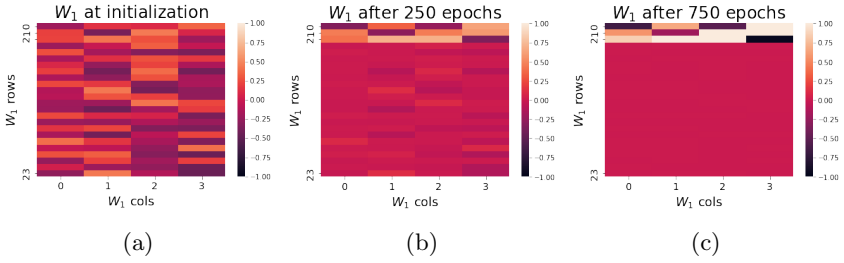


Figure 11: Values of the W_1 matrix for individual neural networks of the form of Equation 41 during training on the Gaussian XOR Mixture datasets $\mathcal{D}_{24,3,0.8,384}$: 11a, 11b, 11c represent the coefficients at initialization, after 250 training epochs and after 750 training epochs of training with ADAM at a learning rate 0.001.

C Data, Code, and Simulation details

Both the code to reproduce the indicated experiments and also the relevant data are freely available at `https://github.com/incud/QuantumPathKernel`. The code is released open-source.

The indicated experiments have been simulated on two devices:

- one Dell Latitude 5510 having: Intel Core i7-10610U CPU with 4 physical cores, 16GB RAM, without CUDA-enabled GPUs;
- one cluster node having: Intel Xeon Silver 4216 CPU with 64 physical cores, 180GB RAM, with 4 x CUDA-enabled GPUs NVidia Tesla V100S 32GB VRAM.

The software runs on Ubuntu 20.04 LTS and uses Python v3.9.11, PiP packet-manager v22.0.4 along with the other libraries listed in `requirements.txt` file in the root of the attached repository. Installation and simulation instructions are documented in the `README.md` file in the root of the repository. Our code is based upon freely available, open-source frameworks only.

The framework used to define and simulate the quantum circuit is PennyLane [52]. The simulations have been accelerated using the JAX library [53]. (JAX might require installation from source code if used on operating systems different from Ubuntu). Alternatively, the source code can be set such that PennyLane does not require this library. (However, in this case, the circuit simulation might be substantially slower and would not benefit the full potential of multicore CPUs and GPUs). These experiments have not been run on quantum hardware.

The input and output of each experiment are contained in different sub-folders within the root directory. They contain the specifications needed to generate the training and testing datasets, the datasets themselves, the trace of the parameters during the training for any model, and the Quantum NTK and Quantum Path Kernel Gram matrices for each model (which may be used to create a pre-trained model), and also the resulting plots. The `README.md` explains in detail the commands needed to reproduce our results.

The simulations for all experiments have taken approximately 600 hours across both machines used.