

# Recurrent Quantum Neural Networks

Johannes Bausch\*

DAMTP, CQIF, University of Cambridge, UK

Recurrent neural networks are the foundation of many **sequence-to-sequence models in machine learning**, such as machine translation and speech synthesis. In contrast, applied quantum computing is in its infancy. Nevertheless there already exist quantum machine learning models such as variational quantum eigensolvers which have been used successfully e.g. in the context of energy minimization tasks.

In this work we construct a **quantum recurrent neural network (QRNN)** with demonstrable performance on **non-trivial tasks such as sequence learning and integer digit classification**. The QRNN cell is built from **parametrized quantum neurons**, which, in conjunction with amplitude amplification, create a **nonlinear activation of polynomials of its inputs and cell state**, and allow the extraction of a probability distribution over predicted classes at each step.

To study the model's performance, we provide an **implementation in pytorch**, which allows the **relatively efficient optimization of parametrized quantum circuits with thousands of parameters**. We establish a QRNN training setup by benchmarking optimization hyperparameters, and analyse suitable network topologies for simple memorisation and sequence prediction tasks from Elman's seminal paper (1990) on temporal structure learning. We then proceed to evaluate the **QRNN on MNIST classification**, both by feeding the QRNN **each image pixel-by-pixel**; and by utilising modern data augmentation as pre-processing step. Finally, we analyse to what extent the unitary nature of the network **counteracts the vanishing gradient problem** that plagues many existing quantum classifiers and classical RNNs.

---

\*jkrb2@cam.ac.uk

# 1. Introduction

Optimizing recurrent neural networks for long sequences is a challenging task: applying the same RNN cell operator iteratively often suffers from the well-studied vanishing or exploding gradients problem, which results in poor training performance [PMB12]. While long short-term memories or gated recurrent units (LSTMs and GRUs) with their linear operations acting on the cell state have been proposed as a way of circumventing this problem, they too are typically limited to capturing about 200 tokens of context, and e.g. start to ignore word order with increasing sequence lengths (more than 50 tokens away) [ZQH15; Dab08; Kha+18].

This failure of existing recurrent models to capture very long sequences surely played a role in the advent of alternative, non-recurrent models applicable to sequence-to-sequence tasks, such as transformer-type architectures with self-attention. Yet while these alternatives often outperform LSTMs, they feature a fixed-width context window that does not easily scale with the sequence length; extensions thereof are an active field of research [AIR+19; Dai+19; KKL20].

Beyond training modifications such as truncated backpropagation [AFF19] which attempt to mitigate the vanishing gradient problem for recurrent neural networks directly, there have been numerous proposals to parametrize recurrent models in a way which limits or eliminates gradient decay, by ensuring the transfer operation at each step preserves the gradient norm; examples include orthogonal [Vor+17] or unitary recurrent neural networks [ASB15; Wis+16; Jin+17]. Yet how to pick a parametrization for the RNN cell that is easy to compute, allows efficient training, and performs well on real-world tasks? This is a challenging question [HR16].

In this work, we propose a recurrent neural network model with a parametrization motivated by the emergent field of quantum computation. The interactions of any quantum system can be described by a Hermitian operator  $\mathbf{H}$ , which generates the system’s time evolution under the unitary map  $\mathbf{U}(t) = \exp(i t \mathbf{H})$  as a solution to the Schrödinger equation. The axioms of quantum mechanics thus dictate that any quantum circuit comprising a sequence of individual unitary quantum gates of the form of  $\mathbf{U}_i(t_i)$ —for a set of parameters  $t_i$ —is intrinsically unitary. This means that a parametrized quantum circuit serves a prime candidate for a unitary recurrent network.

Such parametrized quantum circuits have already found their way into other realms of quantum machine learning. A prominent example are variational quantum eigensolvers (VQE), which can serve as a variational ansatz for a quantum state, much akin to how a feed-forward network with a final softmax layer can serve as parametrization for a probability

distribution [WHB19; McC+16; Per+14; Jia+18]. VQEs have been deployed successfully e.g. in the context of minimizing energy eigenvalue problems within condensed matter physics [Cad+19], or as generative adversarial networks to load classical probability distributions into a quantum computer [ZLW19].

To date, classical recurrent models that utilize quantum circuits as sub-routines (i.e. where no quantum information is propagated) [GI19], analysing Hopfield networks on quantum states [All+], or running classical Hopfield networks by a quantum-accelerated matrix inversion method [Reb+18] have been proposed; yet neither of them have the features we seek: a concrete quantum recurrent neural network with a unitary cell that allows to side-step the problem of gradient decay, and can ideally be implemented and trained on current classical hardware—and potentially on emerging quantum devices in the short-to-mid term.

In this work we construct such a quantum recurrent neural network (QRNN), which features demonstrable performance on real-world tasks such as sequence learning and handwriting recognition. Its recurrent cell—the operation executed at each step of the input—utilizes a highly-structured parametrized quantum circuits that deviates significantly from those used in the VQE setting. Its fundamental building block is an improved type of quantum neuron based on [CGA17] to introduce a nonlinearity, and in conjunction with a type of fixed-point amplitude amplification, allows the introduction of measurements (which are projectors, and not unitary operations) such that the overall evolution on correctly-initialized runs nonetheless remains arbitrarily close to unitary.

With an implementation in pytorch, we repeat several of the learning tasks first proposed in Elman’s seminal paper “Finding Structure in Time” [Elm90], which we utilize to assess suitable model topologies such as the size of the cell state and structure of the parametrized RNN cell, and for benchmarking training hyperparameters for well-established optimizers such as Adam, RMSProp and SGD, as well as more costly methods such as L-BFGS, employed extensively within the context of VQEs.

As a next step, we evaluate the QRNN on integer digit classification, using the standard MNIST dataset. We find that feeding images pixel-by-pixel allows a discrimination of pairs of digits with up to 99.6% accuracy on the test set. Using modern data augmentation techniques the QRNN further achieves a test set performance on *all* digits of  $\approx 99.2\%$ . In order to demonstrate that the model indeed captures the temporal structure present within the MNIST images, we use the QRNN as a generative model, successfully recreating handwritten digits from an input of ‘0’ or ‘1’. As a final experiment, we assess whether the gradient quality decays for long input sequences. In a task of recognizing unique base pairs in a DNA string, we found that even for sequences of 1000 bases—where the target base pairs to be identified are over 500 steps in the past—training performance remains unaffected.



Without question the performance of our proposed model is yet to compete with state-of-the-art scores on popular, much larger datasets than e.g. MNIST. On the other hand, QRNNs are the first quantum machine learning model capable of working with non-superposed training data as high-dimensional as images of integer digits or DNA sequences—meaning binary vectors  $v_i \in [0, 1]^n$  of length  $n > 3000$ , utilized *not* in quantum superposition as a state  $\propto \sum_{i=1}^n v_i |i\rangle$ . By feeding in the vector in a “one-hot” fashion, i.e. step-by-step as a state  $|v\rangle |v_1\rangle \otimes \dots \otimes |v_n\rangle$ , all information remains accessible to the procedure. Naturally, this would not be within reach to simulate, let alone run on current or near term quantum hardware; feeding  $|v\rangle$  up front into e.g. a VQE would require  $n$  qubits.

Furthermore, as a variational quantum algorithm capable of being trained with thousands of parameters, benefits such as flattening of local minima emerge; allowing the employment of cheaper, well-established optimization algorithms such as Adam; and enhancing generalization performance [DL18; ACH18].

## 2. Recurrent Quantum Neural Networks

### 2.1. Parametrized Quantum Gates

Typical VQE quantum circuits are very dense, in the sense of alternating parametrized single-qubit gates with entangling gates such as controlled-not operations. This has the advantage of compressing a lot of parameters into a relatively compact circuit. On the other hand, while it is known that such circuits form a universal family, their high density of entangling gates and lack of connection between parameters results in models that are currently hard to train on classification tasks for inputs larger than a few bits [Ben+19].

In this work, we construct a QRNN cell that is a highly-structured parametrized quantum circuit, built in a fashion such that few parameters are re-utilized over and over, and such that each of them steers a much higher-level logical unit than the components of a VQE circuit. The cell is built mainly from a novel type of quantum neuron—an extension of [CGA17]—which rotates its target lane according to a non-linear activation function applied to polynomials of its binary inputs (eqs. (2) and (3); figs. 1 to 3 in section 2.2). These neurons are combined in section 2.3 to form a structured RNN cell, as shown in fig. 4. The cell is a combination of an input stage that, at each step, writes the current input into the cell state. This is followed by multiple work stages that compute with access to input and cell state, and a final output stage that creates a probability density over possible predictions. Applying these QRNN cells iteratively on the input sequence as shown in fig. 5 results in a recurrent model much like traditional RNNs.

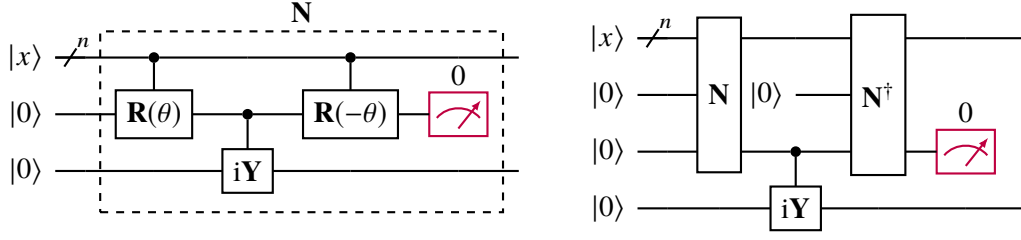


Figure 1: Quantum neuron by [CGA17]; left a first order neuron, right a second order application. Recursive iteration yields higher-order activation functions, respectively. The purple meter indicates a postselection (using fixed-point amplitude amplification) as described in [Tac+19].

During training we perform quantum amplitude amplification (see [Gue19]) on the output lanes, to ensure that we measure the correct token from the training data at each step. While measurements are generally non-unitary operations, the amplitude amplification step ensures that the measurements during training are as close to unitary as we wish.

While the resulting circuits are comparatively deep as compared to a traditional VQE models, they require only as many qubits as the input and cell states are wide (plus a few ancillas for the implementation of quantum neurons and amplitude amplification).

## 2.2. A Higher-Degree Quantum Neuron

The strength of classical neural networks emerges through the application of nonlinear activation functions to the affine transformations on the layers of the network. In contrast, due to the nature of quantum mechanics, any quantum circuit composed of unitary gates and measurements will necessarily be a linear operation.

However, this does not mean that no nonlinear behaviour occurs anywhere within quantum mechanics: a simple example is a single-qubit gate  $R(\theta) := \exp(iY\theta)$  for the Pauli matrix  $Y$ , which acts like

$$R(\theta) = \exp\left(i\theta \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}\right) = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix},$$

i.e. as a rotation within the two-dimensional space spanned by the computational basis vectors of a single qubit,  $\{|0\rangle, |1\rangle\}$ . While the rotation matrix itself is clearly a linear operator, we note that the amplitudes of the state— $\cos \theta$  and  $\sin \theta$ —depend non-linearly on

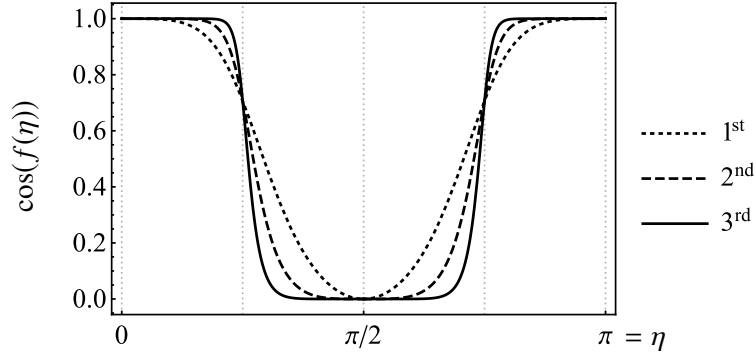


Figure 2: Quantum neuron amplitude  $\cos(f(\eta))$ , as given in eq. (2). Shown are the first to third order activations  $o = 1, 2, 3$ .

the angle  $\theta$ . If we raise the rotation to a controlled operation  $\mathbf{cR}(i, \theta_i)$  conditioned on the  $i^{\text{th}}$  qubit of a state  $|x\rangle$  for  $x \in \{0, 1\}^n$ , one can derive the map

$$\mathbf{R}(\theta_0) \mathbf{cR}(1, \theta_1) \cdots \mathbf{cR}(n, \theta_n) |x\rangle |0\rangle = |x\rangle (\cos(\eta) |0\rangle + \sin(\eta) |1\rangle)$$

$$\text{where } \eta = \theta_0 + \sum_{i=1}^n \theta_i x_i. \quad (1)$$

This corresponds to a rotation by an affine transformation of the basis vector  $|x\rangle$  with  $x = (x_1, \dots, x_n) \in \{0, 1\}^n$ , by a parameter vector  $\theta = (\theta_0, \theta_1, \dots, \theta_n)$ . This operation extends linearly to superpositions of basis and target states; and due to the form of  $\mathbf{R}(\theta)$  all newly-introduced amplitude changes are real-valued.<sup>1</sup>

This cosine transformation of the amplitudes by a controlled operation is already non-linear; yet a sin function is not particularly steep, and also lacks a sufficiently “flat” region within which the activation remains constant, as present e.g. in a rectified linear unit. Cao et al. [CGA17] proposed a method for implementing a linear map on a set of qubits which yields amplitudes that feature such steeper slopes and plateaus, much like a sigmoidal activation function. The activation features an order parameter  $\text{ord} \geq 1$ —the “order” of the neuron—that controls the steepness of the functional dependence, as shown in fig. 2; the circuit which gives rise to such an activation amplitude is shown in fig. 1.

On pure states this quantum neuron gives rise to a rotation by an angle

$$f(\theta) = \arctan \left( \tan(\theta)^{2^{\text{ord}}} \right).$$

<sup>1</sup>Complex amplitudes are not necessary for the power of quantum computing.

Starting from an affine transformation  $\eta$  for the input bitstring  $x_i$  as given in eq. (1), this rotation translates to the amplitudes

$$\cos(f(\eta)) = \frac{1}{\sqrt{1 + \tan(\eta)^{2 \times 2^{\text{ord}}}}} \quad \text{and} \quad \sin(f(\eta)) = \frac{\tan(\eta)^{2^{\text{ord}}}}{\sqrt{1 + \tan(\eta)^{2 \times 2^{\text{ord}}}}}, \quad (2)$$

emerging from normalising the transformation  $|0\rangle \mapsto \cos(\theta)^{2^{\text{ord}}} |0\rangle + \sin(\theta)^{2^{\text{ord}}} |1\rangle$ , as can be easily verified. For  $\text{ord} = 1$ , the circuit is shown on the left in fig. 1; for  $\text{ord} = 2$  on the right. Higher orders can be constructed recursively.

When executed on pure states, this quantum neuron is a so-called repeat-until-success (RUS) circuit, meaning that the ancilla that is measured (purple meter in fig. 1) indicates whether the circuit has been applied successfully. When the outcome is zero, the neuron has been applied. When the outcome is one, a (short) correction circuit reverts the state to its initial configuration. Started from a pure state (e.g.  $|x\rangle$  for  $x \in \{0, 1\}^n$ , as above) and repeating whenever a 1 is measured, one obtains an arbitrarily-high success probability.

Unfortunately this does not work for non-pure inputs. For states in superposition, such as a state  $(|x\rangle + |y\rangle)/\sqrt{2}$ , for  $x \neq y$  two bit-strings of length  $n$ , the amplitudes within the superposition will depend on the history of success [CGA17, appdx. B]. Using a technique called fixed-point oblivious amplitude amplification [Gue19; Tac+19], one can alleviate this issue, and essentially post-select on measuring outcome 0 while preserving unitarity of the operation to arbitrarily high accuracy. This comes at the cost of performing multiple rounds of these quantum circuits (and their inverses), the number of which will depend on the likelihood of measuring a zero (i.e. success) in first place. This naturally depends on the parameters of the neuron,  $\theta$ , and the input state given. In the following we will thus simply assume that the approximate postselection is possible, and carefully monitor the overhead due to the necessary amplitude amplification in our empirical studies in section 4; we found that in general the postselection overhead remained mild, and tended to converge to zero as learning progressed.

The specific activation function this quantum neuron gives rise to is depicted in fig. 2. We point out that other shapes of activation functions can readily be implemented in a similar fashion [de +19].

In this work, we generalize this quantum neuron first proposed in [CGA17] by increasing the number of control terms. More concretely,  $\eta$  as given in eq. (1) is an affine transformation of the boolean vector  $x = (x_1, \dots, x_n)$  for  $x_i \in \{0, 1\}$ . By including multi-control gates—with their own parametrized rotation, labelled by a multiindex  $\theta_I$  depending on the qubits

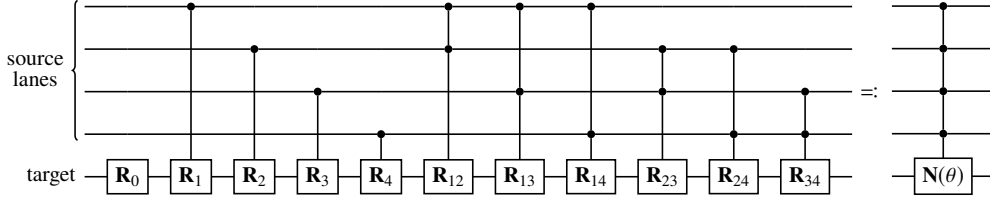


Figure 3: Degree  $d = 2$  controlled rotation for quantum neuron shown in fig. 1, on  $n = 4$  input neurons; the controlled rotations are  $\mathbf{R}_I := \mathbf{R}(\theta_I)$  for  $I \subset [n]$  with  $|I| \leq d$ . The quantum neuron thus carries a parameter vector  $\theta \in \mathbb{R}^D$  for  $D = \sum_{i=0}^d \binom{n}{i}$ .

$i \in I$  that the gate is conditioned on—we obtain the possibility to include higher degree polynomials, namely

$$\eta' = \theta_0 + \sum_{i=1}^n \theta_i x_i + \sum_{i=1}^n \sum_{j=1}^n \theta_{ij} x_i x_j + \dots = \sum_{\substack{I \subseteq [n] \\ |I| \leq d}} \theta_I \prod_{i \in I} x_i, \quad (3)$$

where  $d$  labels the degree of the neuron; for  $d = 2$  and  $n = 4$  an example of a controlled rotation that gives rise to this higher order transformation  $\eta'$  on the bit string  $x_i$  is shown in fig. 3. In this fashion, higher degree boolean logic operations can be directly encoded within a single conditional rotation: an AND operation between two bits  $x_1$  and  $x_2$  is simply  $x_1 x_2$ .

### 2.3. QRNN Cell

The quantum neuron defined in section 2.2 will be the crucial ingredient in the construction of our quantum recurrent neural network cell. Much like for classical RNNs and LSTMs, we define such a cell which will be applied iteratively to the input presented to the network. More specifically, the cell is comprised of in- and output lanes that are reset after each step, as well as an internal cell state which is passed on to the next iteration of the network. The setup and inner workings of this cell are depicted and described in fig. 4.

### 2.4. Sequence to Sequence Model

In order to be able to apply the QRNN cell constructed in section 2.3, we need to iteratively apply it to a sequence of input words  $\text{in}_1, \text{in}_2, \dots, \text{in}_L$ . This is achieved as depicted in fig. 5.

The output lanes  $\text{out}_i$  label a measured discrete distribution  $p_i$  over the class labels (which we can do by reading out the statevector weights if running a simulation on a classical



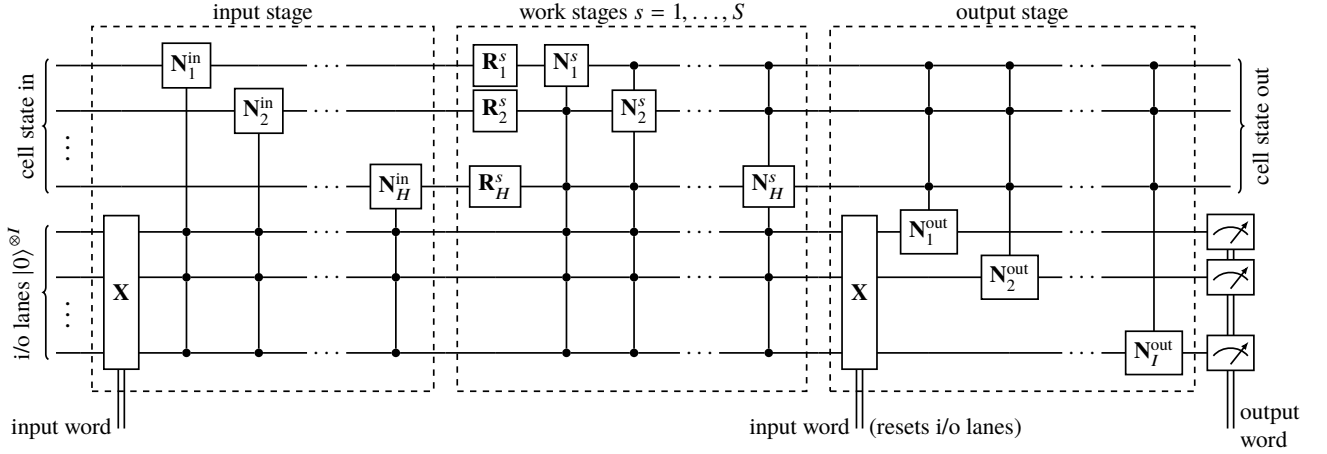


Figure 4: Quantum recurrent neural network cell. Each controlled quantum neuron  $\text{cN}_j^i$  is implemented as explained in section 2.2 and fig. 1 and comes with its own parameter vector  $\theta_j^i$ , where we draw the control lanes from the rotation inputs as depicted in fig. 3, with ancillas omitted for clarity. The  $R_j^i$  are extra rotations with a separate parameter set  $\phi_j^i$ .

computer; or by repeated measurements on quantum hardware). This distribution can then be fed into an associated loss function such as cross entropy or CTC loss.

### 3. Implementation and Training

We implemented the QRNN in pytorch [Bau20], using custom quantum gate layers and operations that allow us to extract the predicted distributions at each step. As this is in essence a simulation of a quantum computation, we take the following shortcuts: instead of truly performing fixed-point amplitude amplification for the quantum neurons and output lanes during training, we **postselect**; as aforementioned, we kept track of the postselection probabilities which allows us to compute the overhead that would be necessary for amplitude amplification. We further extract the output probability distribution instead of estimating it at every step using measurements.

In our experiments we focus on character-level RNNs. Just like in the classical case, the sequence of predicted distributions  $\{p_i\}$  is fed into a standard `nn.CrossEntropyLoss` to minimize the distance to a target sequence. With pytorch’s autograd framework we are able to perform gradient-based learning directly; on quantum hardware either gradient-free optimizers such as L-BFGS, NatGrad would have to be utilized, or numerical gradients

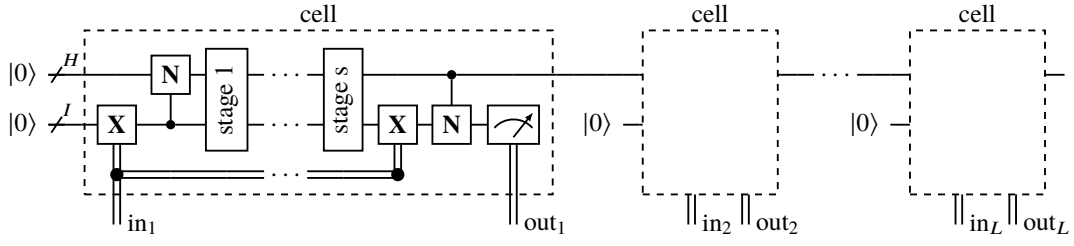


Figure 5: Quantum recurrent neural network, by applying the same QRNN cell constructed in section 2.3 iteratively to a sequence of input words  $in_1, \dots, in_L$ . All input and ancilla qubits used throughout can be reused; we thus need  $H + I + \text{ord}$  qubits, where  $H$  is the cell state workspace size,  $I$  the input token width (in bits), and  $\text{ord}$  the order of the quantum neuron activation, as explained in section 2.2.

extracted [W GK20]. All experiments were executed on 2-8 CPUs, and required between 500MB and 35GB of memory per core. We implement a mechanism train batches of data in parallel. A straightforward implementation in pytorch, on real-world hardware batching would simply mean executing the QRNN with the same parameters on multiple devices in parallel, and averaging the resulting losses.

## 4. Empirical Results

### 4.1. Sequence Memorization

The first task we implement is whether the network can learn to reproduce the two sequences 44444...4 and 12312...3. For a QRNN with 2 stages, neuron degree 3 and a workspace size of 5 (1162 parameters) this poses no problem; in fact, this is vastly over-parametrized for the task at hand, and good convergence can be achieved with a much smaller network. Nonetheless, with this setup we benchmark optimizer and learning rate hyperparameters; our findings are summarized in fig. 6.

While it tended to produce the lowest validation loss over a wide range of learning rates, we found the L-BFGS optimizer commonly used with VQE circuits to behave unpredictably for all but the simple sequence learning tasks: for a large set of initial seeds, only very few resulted in a good convergence within 500 training iterations—a problem that Adam did not exhibit, where generally many random initializations resulted in a good training run. Furthermore, L-BFGS is extremely costly, taking about an order of magnitude longer for convergence and a significantly higher memory consumption than the other, purely gradient-based methods.

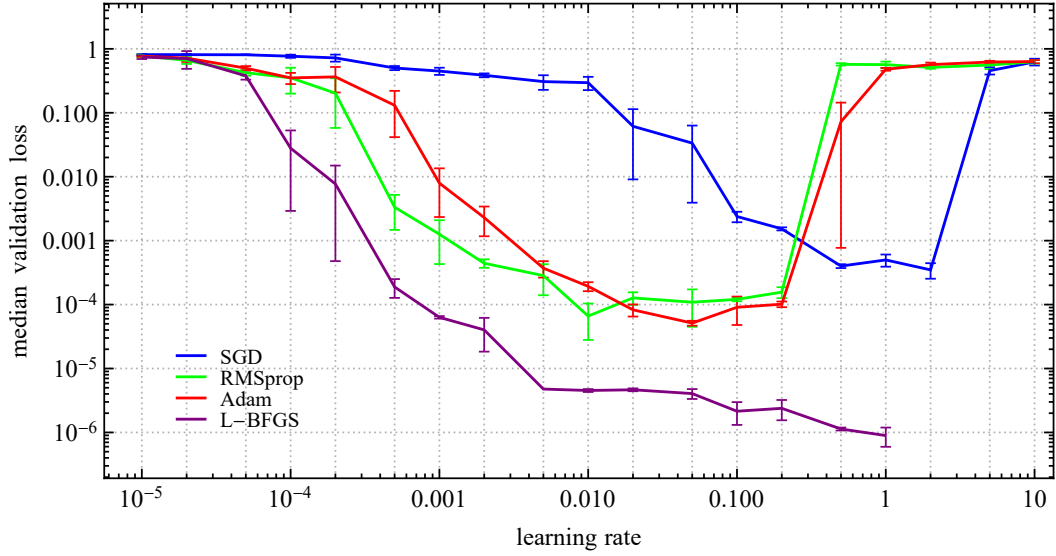


Figure 6: Training performance of SGD, RMSprop, Adam and L-BFGS optimizers over a range of learning rates, as described in section 4.1. Shown is the validation loss achieved after 500 training steps. The plotted points indicate the median of five runs; the error bars indicate the median absolute deviation, a robust measure of dispersion within a univariate dataset.

SGD has a narrow window of good learning rates, whereas RMSprop and Adam are less sensitive to this choice, with Adam generally outperforming the former in convergence time. Due to its performance and relative robustness with respect to the choice of learning rate, Adam was our default choice for all following experiments if not stated otherwise; learning rates were generally kept at or around 0.05.

In fig. 11 we plot the postselection overhead during training. Notably, the overhead quickly converges to a factor of one as the validation loss decreases.

## 4.2. Finding Structure in Time

In his 1990 paper, Elman describes two basic sequence learning tasks to evaluate structure in time [Elm90]. The first task is that of learning XOR sequences, which are binary strings  $s = s_1 s_2 s_3 \dots s_L$  such that each third digit is the XOR value of the preceding two, i.e.  $s_{3i} = s_{3i-1} \oplus s_{3i-2}$ ; one example being  $s = 000\ 011\ 110\ 011\ 101$ .

Due to the simplicity of the test, we use a QRNN with workspace size 4 and a single work stage to explore which parameter initialization converges to a validation loss threshold of  $10^{-3}$  first. As shown in fig. 4, there are two groups of parameters: those for the neurons,

and those for the single-qubit unitaries within the work stages. Each quantum neuron as depicted in fig. 3 and eq. (3) itself comprises two parameter sets: a bias gate  $\mathbf{R}_0$  with angle  $\theta_0$ , and the weights (all other parameters). Choosing to initialize each of them with a normal distribution with mean  $\mu$  and width  $\sigma$ , we have four parameter group hyperparameters: bias  $\mu$ , bias  $\sigma$ , weights  $\sigma$  (the mean is already captured in the bias gate), and unitaries  $\sigma$  (for which we chose the mean to be zero by default).

Our findings and choices for the default initialisation are collected in fig. 10 in the appendix. The most influential meta parameter is the bias  $\mu = \pi/4$ —which, as shown in fig. 2, places the initial polynomial  $\eta$  at the steepest slope of the activation function, which results in a large initial gradient.

The second task described in [Elm90] is that of learning the structure of a sentence made up of the three words “ba”, “dii” and “guuu”; an example being “ba dii ba guuu dii”. Having seen the letter ‘d’, the network thus has to remember that the next two letters to predict are “ii”, and so on. We chose this slightly more difficult task to assess how the QRNN topology influence convergence speed. We found that the neuron order  $\text{ord} = 2$  performs best, which results in an activation function with relatively steep flanks and flat plateau sections. The latter allow the incoming signal to remain constant for a significant range over its parameters, while still maintaining a small but nonzero gradient throughout the plateau area (this plateau gradient is significantly suppressed already at  $\text{ord} = 3$ ). The other parameters are discussed in appendix B.

### 4.3. MNIST Classification

To test whether we can utilize our QRNN setup to classify more complex examples, we assess its performance on handwritten integer digit classification. While this is a rather untypical task for a recurrent network, there exist baselines both for a comparison with a classical RNN, as well as with quantum classifiers.

We use the MNIST dataset, with a 55010 : 5000 : 10000 train : validate : test split, where the training subset is the standard one provided with the dataset, and the validate : test split is chosen manually at random from the provided test set [LCB10]. As a post-processing step we first crop the images to  $20 \times 20$  pixels, downscale them to size  $10 \times 10$ , and then binarize each image such that it has a bit depth of 1. Naturally, this significantly reduces the information content of the training data; yet as we are not competing with top-of-the-line classifiers with our QRNN model we found the resulting input sequence length of 100 (for the  $10 \times 10$  pixel images) to give a good compromise between computational cost, classification

Digit Set	Method	Data Augmentation	Accuracy [%]
{0, 1}	QRNN (12 qubits, Adam)	none	$99.2 \pm 0.2$
	<i>ensemble of 4</i>	none	<b><math>99.6 \pm 0.16</math></b>
{3, 6}	VQE (17 qubits) [FN18] <sup>§</sup>	ambiguous samples removed	98
	QRNN (12 qubits, Adam)	none	$89.7 \pm 0.8$
	QRNN (10 qubits, L-BFGS)	none	$97.1 \pm 0.7$
	<i>ensemble of 6</i>	none	<b><math>99.0 \pm 0.3</math></b>
full MNIST	VQE (10 qubits) [Gra+19] <sup>†</sup>	partitioned into {even, odd}	82
	uRNN [ASB15]	none	95.1
	LSTM [ASB15]	none	98.2
	QFD (> 200 qubits) [KL20] <sup>‡</sup>	PCA, slow feature analysis	98.5
	QRNN (10 qubits, Adam)	PCA, t-SNE	$94.6 \pm 0.4$
	QRNN (13 qubits, Adam)	UMAP	$96.7 \pm 0.2$
	<i>ensemble of 3</i>	UMAP	<b><math>99.23 \pm 0.05</math></b>

Table 1: Classification of MNIST using QRNNs on 12 qubits (workspace size 8, 2 stages, neuron degree 2; 1212 parameters), 10 qubits (workspace size 6, 2 stages, neuron degree 3; 1292 parameters), and 13 qubits (workspace size 7, 2 stages, neuron degree 3; 3134 parameters), as well as ensembles thereof. Input either presented pixel-by-pixel; resp. t-SNE/UMAP augmented with discretized coordinates (2 to 4 dimensions, presented bit by bit up to 8 bits of precision).

<sup>§</sup>) Images down-scaled to  $4 \times 4$  pixels and binarized. The set used for the accuracy measurement thus contains  $\approx 70\%$  samples already present during training. Naturally, this is a problem of the small sample dimensions; and to varying extent also present in the full MNIST classification models below that employ other dimensionality reduction techniques. Our  $10 \times 10$  pixel-by-pixel data contains 80 duplicates for the digit ‘1’ between training and test set (so  $\approx 7\%$  of the test class), and a single duplicate for the digit ‘7’ (so  $\approx 0.1\%$  of the test class).

<sup>†</sup>) Image data prepared in superposition, i.e. as a state  $\propto \sum_i v_i |i\rangle = (v_1, \dots, v_n)$ , where  $v_i \in [0, 1]$  is the pixel value at position  $i$  in the image; VQE with such an embedding thus serve as linear discriminator.

<sup>‡</sup>) The paper exploits a well-known quantum speedup in performing sparse linear algebra [HHL08] to implement a quantum variant of slow feature analysis as performed in [Ber05], but with a different classifier. It relies on quantum random access memory (QRAM), recently shown to allow “de-quantization” of claimed exponential quantum speedups [Tan19]. The authors do not explicitly state a qubit count, so it is lower-bounded from the number of qubits required for the QRAM alone.

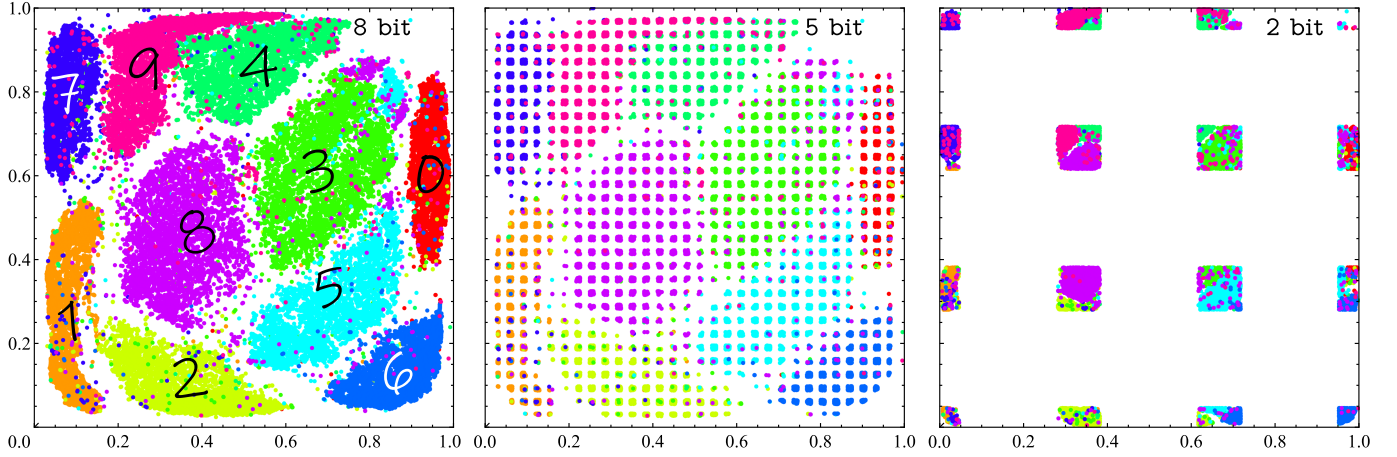


Figure 7: MNIST training dataset mapped to a 2D plane by t-SNE; where each coordinate is further mapped by an inverted Gaussian to distribute all samples approximately uniformly over the intervals  $[0, 1]$  in  $x$  and  $y$  direction, respectively. The left, middle and right pictures show discretized coordinates of 8, 5 and 2 bits, respectively. t-SNE is an unsupervised feature extractor; meaning the clusters of digits emerge without knowledge of the digits’ labels. It is clear that the discretization step introduces additional errors; the lower the resolution of the coordinate, the more the information content is diminished.

performance, and visual acuity—the latter is of particular importance for the generative task, where we wish to qualitatively assess that the network can render handwritten digits.

We choose two “scanlines” across each image: one left-to-right, top-to-bottom; the other one top-to-bottom, left-to-right. This means that two bits of data are presented at each step. The output labels are then simply binary values of the numbers 0 to 9, which have to be written to the output at the last few steps of the sequence (in little Endian order). We found that pairs of digits such as ‘0’ and ‘1’ (arguably the easiest ones) could be discriminated with  $\approx 99.2\%$  success probability when using a single network ( $\approx 99.6\%$  for an ensemble of four); but even more complicated pairs like ‘3’ and ‘6’ could be distinguished with  $\approx 99\%$  likelihood.

In addition to classifying digits by presenting images pixel-by-pixel, we also used 2D and 3D t-distributed stochastic neighbor embedding (t-SNE, [MH08]) clustering as data augmentation in a first step; and a more modern dimensionality reduction technique called uniform manifold approximation and projection (UMAP, [MHM18]), which reduces the 792-dimensional MNIST vectors to 2 to 4 dimensions. The RNN was then presented with the up to four coordinates, discretized to between two and eight bits of precision, which the network has learn to decode and classify. To aid in the resulting information loss due to the discretization of floating point values, we mapped the raw dimensionality-reduced

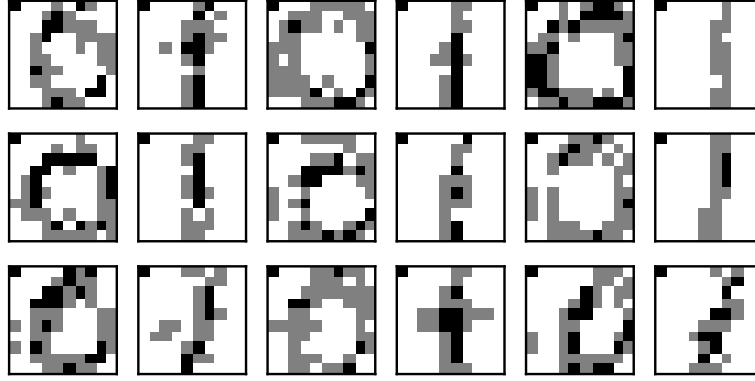


Figure 8: QRNN-generated handwritten digits ‘0’ and ‘1’, as explained in section 4.4. The QRNN topology is the same as for MNIST classification, with 1212 free parameters (section 4.3).

coordinates to the discretized ones in a nonlinear fashion (an inverted Gaussian) to make full use of the available dynamic range.

It is worth emphasizing that similar to other feature extraction methods such as principal component analysis (PCA), all of the feature maps are learned *only* on the training set, and then applied as-is to the validation and test sets. This ensures that only information from the training samples is used in the creation of a preprocessing pipeline. While t-SNE is extremely powerful at revealing clusters in data (see fig. 7), using it as a generic preprocessing step is computationally costly. We can still employ it for a dataset the size of MNIST, by first performing PCA to reduce each image to 30 dimensions (instead of 100), and then—as a second step—t-SNE. This is a common chain of reductions and was e.g. also used in [KL20], with a combination of PCA and slow feature analysis.

Perhaps unsurprisingly we found UMAP to outperform t-SNE by a large margin; this is to be expected, as UMAP is designed for dimensionality reduction, whereas t-SNE’s primary goal is visualization of high-dimensional datasets.

We summarize our findings and a comparison with existing literature in table 1.

#### 4.4. QRNNs as Generative Models

Instead of classifying digits, QRNNs can also be used as generative models. With a ‘0’ or ‘1’ presented to the QRNN at the first step, we train it to re-generate handwritten digits. As shown in fig. 8, the network indeed learns the global structure of the two digits; where it is evident that the intrinsic randomness due to quantum measurements present during inference gives rise to digits with various characteristics, such as character slant or line thickness.

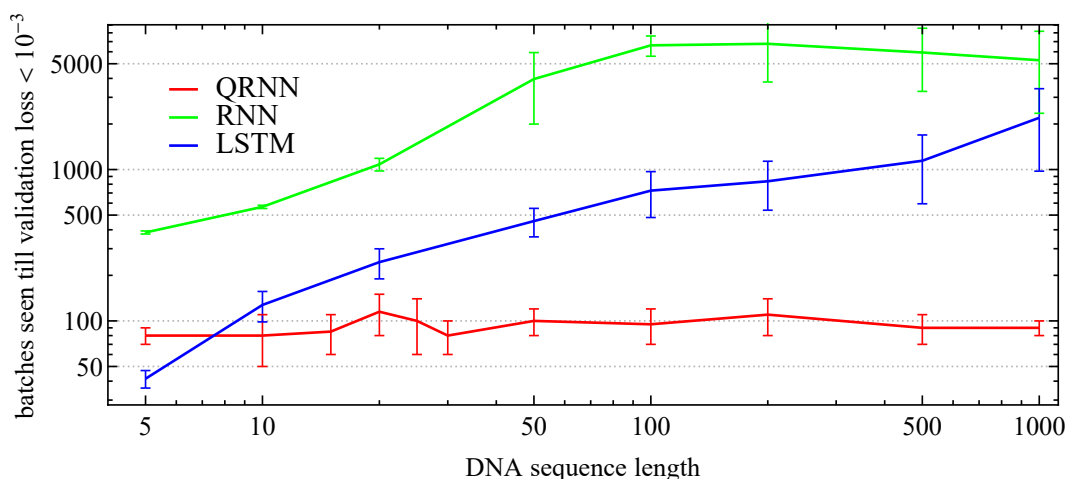


Figure 9: Median number of training steps to reach a validation loss  $10^{-3}$  (with error bars indicating median absolute deviation). Shown are QRNN (red; 1 work stage, cell state size 5, 837 parameters), RNN (blue; 1 layers, hidden layer size 22, and a final linear layer; 888 parameters) and LSTM (green; 1 layers, hidden layer size 10, and a final linear layer; 888 parameters) for DNA sequence recognition task described in section 4.5.

#### 4.5. Long Sequence Tests

To assess our claims of high-quality gradients even in a situation of long sequences, we set up a test set consisting of gene sequences made up of the bases ‘G’, ‘A’, ‘T’ and ‘C’; a single ‘U’ is then inserted at a random position within the first half of the string, and the task of the network is to identify the base following after the ‘U’. For instance, the label to identify for the sequence ‘AGAUATTCAGAAT’ is ‘A’. We repeat this classification task for multiple sequence lengths and several initial seeds, and stop after the validation loss is below a threshold of  $10^{-3}$ . The number of steps required to reach this threshold—where at every training step the network is presented with a batch of 128 random training samples—is then our metric of success; the lower the better.

We found that a QRNN with a workspace size of 5, one work stage, and activation degree 3 (resulting in 837 parameters) can be trained within an almost constant number of steps, even for string lengths of 1000 bases.

As comparison, we train an RNN and an LSTM variant on precisely the same dataset. To field an objective view from the optimizer’s perspective, we recreated topologies with a parameter count that matched the QRNN’s 837 as closely as possible: both the RNN with one layer of width 22, and the LSTM with one layer of width 10 have 888 trainable



parameters. For each classical network we optimized the learning rate as hyperparameter, but left the other stock choices provided by pytorch.<sup>2</sup>

The results of this test can be seen in fig. 9. While the QRNN features a relatively stable number of necessary training steps till convergence ( $\approx 100$ , which means it has seen  $\approx 1.28 \times 10^4$  samples), the LSTM shows a steady increase—starting out faster than the QRNN, but ending up with a more than order-of-magnitude worse performance in this test. The RNN was hardest to train for this task.<sup>3</sup>

## 5. Conclusion and Outlook

Without doubt, existing recurrent models—even simple RNNs—outclass the proposed QRNN architecture in this paper in real-world learning tasks. In part, this is because we cannot easily simulate a large number of qubits on classical hardware: the memory requirements necessarily grow exponentially in the size of the workspace, for instance, which limits the number of parameters we can introduce in our model—on a quantum computer this overhead would vanish, resulting in a linear execution time in the circuit depth.

What should nevertheless come as a surprise is that the model *does* perform relatively well on non-trivial tasks such as the ones presented here, in particular given the small number of qubits (usually between 8 and 14) that we utilised. As qubit counts in real-world devices are severely limited—and likely will be for the foreseeable future—learning algorithms with tame system requirements will certainly hold an advantage.

Moreover, while we motivate the topology of the presented QRNN cell given in fig. 4 by the action of its different stages (writing the input; work; writing the output), and while the resulting circuits are already far more structured than existing VQE setups, our architecture is still simplistic as compared to the various components of an RNN, let alone an LSTM. In all likelihood, a more specialized circuit structure will outperform the “simple” quantum recurrent network presented herein.

Beyond the exploratory aspect of our work, our main insights are twofold. On the classical side—as discussed in the introduction—we present an architecture which can run on current hardware and ML implementations such as pytorch; and which is a candidate parametrization

---

<sup>2</sup>Without doubt it will be possible to improve upon the convergence times for the task at hand with a more thorough analysis of hyperparameters, optimizers, and activation functions, or by allowing the network a larger capacity in terms of trainable parameters. We expect this to hold true for both traditional recurrent models, as well as the QRNN.

<sup>3</sup>We note that for sequences of length 100 or longer, several of the RNN runs timed out at 100k training steps, which is when we interrupted and re-started the training. For RNN’s, data above this point is thus to be taken with a grain of salt.

for unitary recurrent models that hold promise in circumventing gradient degradation for long sequence lengths. On the quantum side, we present a quantum machine learning model that allows ingestion of data of far more than a few bits of size; demonstrate that models with large parameter counts can indeed be evaluated and trained; and that classical baselines such as MNIST classification are within reach for variational quantum algorithms.

Variants of this “quantum-first” recurrent model might find application in conjunction with other quantum machine learning algorithms, such as quantum beam search [BSP19], which could be employed in the context of language modelling. With a more near-term focus in mind, modelling the evolution of quantum systems with noisy dynamics is a task that can be addressed using classical recurrent models [Flu+20]. Due to the intrinsic capability of a QRNN to keep track of a quantum state, a quantum recurrent model might promise to better capture the exponentially-growing phase space dimension of the system under study.

## Acknowledgements

J. B. acknowledges support of the Draper’s Research Fellowship at Pembroke College, and wishes to thank Stephanie Hyland and Jean Maillard for helpful discussions and suggestions.

## References

- [PMB12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “Understanding the exploding gradient problem”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. JMLR.org, Nov. 2012, pp. III–1310–III–1318. arXiv: 1211.5063.
- [ZQH15] Chenxi Zhu, Xipeng Qiu, and Xuanjing Huang. “Transition-based dependency parsing with long distance collocations”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2015.
- [Dab08] Ewa Dabrowska. “Questions with long-distance dependencies: A usage-based perspective”. In: *Cognitive Linguistics* 19.3 (Jan. 2008).
- [Kha+18] Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. “Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2018, pp. 284–294.

- [AIR+19] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. “Character-Level Language Modeling with Deeper Self-Attention”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 2019), pp. 3159–3166.
- [Dai+19] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context”. In: (Jan. 2019). arXiv: 1901.02860.
- [KKL20] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. “Reformer: The Efficient Transformer”. In: *International Conference on Learning Representations*. 2020.
- [AFF19] Christopher Aicher, Nicholas J Foti, and Emily B Fox. “Adaptively Truncating Backpropagation Through Time to Control Gradient Bias”. In: *UAI*. May 2019. arXiv: 1905.07473.
- [Vor+17] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. “On orthogonality and learning recurrent networks with long term dependencies”. In: *ICML*. Jan. 2017. arXiv: 1702.00071.
- [ASB15] Martin Arjovsky, Amar Shah, and Yoshua Bengio. “Unitary Evolution Recurrent Neural Networks”. In: (Nov. 2015). arXiv: 1511.06464.
- [Wis+16] Scott Wisdom, Thomas Powers, John R Hershey, Jonathan Le Roux, and Les Atlas. “Full-Capacity Unitary Recurrent Neural Networks”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS’16. Red Hook, NY, USA: Curran Associates Inc., Oct. 2016, pp. 4887–4895. arXiv: 1611.00035.
- [Jin+17] Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljačić, and Yoshua Bengio. “Gated Orthogonal Recurrent Units: On Learning to Forget”. In: *Neural Computation* 31.4 (June 2017), pp. 765–783. arXiv: 1706.02761.
- [HR16] Stephanie L Hyland and Gunnar Rätsch. “Learning Unitary Operators with Help From  $u(n)$ ”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI’17. AAAI Press, July 2016, pp. 2050–2058. arXiv: 1607.04903.
- [WHB19] Daochen Wang, Oscar Higgott, and Stephen Brierley. “Accelerated Variational Quantum Eigensolver”. In: *Physical Review Letters* 122.14 (Apr. 2019), p. 140504.

- [McC+16] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. “The theory of variational hybrid quantum-classical algorithms”. In: *New Journal of Physics* 18.2 (Feb. 2016), p. 23023.
- [Per+14] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O’Brien. “A variational eigenvalue solver on a photonic quantum processor”. In: *Nature Communications* 5.1 (Sept. 2014), p. 4213. arXiv: 1304.3061.
- [Jia+18] Zhang Jiang, Jarrod McClean, Ryan Babbush, and Hartmut Neven. “Majorana loop stabilizer codes for error correction of fermionic quantum simulations”. In: (Dec. 2018). arXiv: 1812.08190.
- [Cad+19] Chris Cade, Lana Mineh, Ashley Montanaro, and Stasja Stanisic. “Strategies for solving the Fermi-Hubbard model on near-term quantum computers”. In: (Dec. 2019). arXiv: 1912.06007.
- [ZLW19] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. “Quantum Generative Adversarial Networks for learning and loading random distributions”. In: *npj Quantum Information* (2019). arXiv: 1904.00043.
- [GI19] Laszlo Gyongyosi and Sandor Imre. “Training Optimization for Gate-Model Quantum Neural Networks”. In: *Scientific Reports* 9.1 (Dec. 2019), p. 12679.
- [All+] R. Allauddin, K. Gaddam, E.C. Behrman, J.E. Steck, and S.R. Skinner. “Advantages of quantum recurrent networks: an examination of stable states”. In: *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN’02 (Cat. No.02CH37290)*. IEEE, pp. 2732–2737.
- [Reb+18] Patrick Rebentrost, Thomas R. Bromley, Christian Weedbrook, and Seth Lloyd. “Quantum Hopfield neural network”. In: *Physical Review A* 98.4 (Oct. 2018), p. 042308.
- [CGA17] Yudong Cao, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. “Quantum Neuron: an elementary building block for machine learning on quantum computers”. In: (Nov. 2017). arXiv: 1711.11240.
- [Elm90] J Elman. “Finding structure in time”. In: *Cognitive Science* 14.2 (June 1990), pp. 179–211.
- [DL18] Simon S. Du and Jason D. Lee. “On the Power of Over-parametrization in Neural Networks with Quadratic Activation”. In: *ICML*. 2018.

- [ACH18] Sanjeev Arora, Nadav Cohen, and Elad Hazan. “On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization”. In: *ICML*. 2018.
- [Ben+19] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. “Parameterized quantum circuits as machine learning models”. In: (June 2019). arXiv: 1906.07682.
- [Gue19] Gian Giacomo Guerreschi. “Repeat-until-success circuits with fixed-point oblivious amplitude amplification”. In: *Physical Review A* 99.2 (Feb. 2019), p. 022306.
- [Tac+19] Francesco Tacchino, Chiara Macchiavello, Dario Gerace, and Daniele Bajoni. “An artificial neuron implemented on an actual quantum processor”. In: *npj Quantum Information* 5.1 (Dec. 2019), p. 26.
- [de +19] Fernando M. de Paula Neto, Teresa B. Ludermit, Wilson R. de Oliveira, and Adenilton J. da Silva. “Implementing Any Nonlinear Quantum Neuron”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2019), pp. 1–6.
- [Bau20] Johannes Bausch. *QRNN*. 2020. URL: <https://bitbucket.org/rumschuettel/rvqe.git> (visited on 06/25/2020).
- [WKG20] David Wierichs, Christian Gogolin, and Michael Kastoryano. “Avoiding local minima in variational quantum eigensolvers with the natural gradient optimizer”. In: (Apr. 2020). arXiv: 2004.14666.
- [FN18] Edward Farhi and Hartmut Neven. “Classification with Quantum Neural Networks on Near Term Processors”. In: (Feb. 2018). arXiv: 1802.06002.
- [Gra+19] Edward Grant, Leonard Wossnig, Mateusz Ostaszewski, and Marcello Benedetti. “An initialization strategy for addressing barren plateaus in parametrized quantum circuits”. In: (Mar. 2019). arXiv: 1903.05076.
- [KL20] Iordanis Kerenidis and Alessandro Luongo. “Quantum classification of the MNIST dataset via Slow Feature Analysis”. In: *Phys. Rev. A* (May 2020). arXiv: 1805.08837.
- [HHL08] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. “Quantum algorithm for solving linear systems of equations”. In: *Physical Review Letters* 103.15 (Nov. 2008), p. 150502. arXiv: 0811.3171.
- [Ber05] Pietro Berkes. “Pattern recognition with slow feature analysis”. In: (2005).

- [Tan19] Ewin Tang. “A quantum-inspired classical algorithm for recommendation systems”. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing - STOC 2019*. New York, New York, USA: ACM Press, 2019, pp. 217–228.
- [LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. “MNIST handwritten digit database”. In: *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist> 2 (2010).
- [MH08] Laurens van der Maaten and Geoffrey E. Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.
- [MHM18] L. McInnes, J. Healy, and J. Melville. “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”. In: *ArXiv e-prints* (Feb. 2018). arXiv: 1802.03426 [stat.ML].
- [BSP19] Johannes Bausch, Sathyawageeswar Subramanian, and Stephen Piddock. “A Quantum Search Decoder for Natural Language Processing”. In: (Sept. 2019). arXiv: 1909.05023.
- [Flu+20] E. Flurin, L. S. Martin, S. Hacohe-Gourgy, and I. Siddiqi. “Using a Recurrent Neural Network to Reconstruct Quantum Dynamics of a Superconducting Qubit from Physical Observations”. In: *Physical Review X* 10.1 (Jan. 2020), p. 011006.
- [NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press, 2010, p. 676.
- [Gro05] Lov K. Grover. “Fixed-Point Quantum Search”. In: *Physical Review Letters* 95.15 (Oct. 2005), p. 150501.

## A. QRNN Postselection Analysis

A QRNN as described in section 2.4 has two locations where we utilise amplitude amplification with a subsequent measurement to mimic the process of postselection. This introduces an overhead, since sub-circuits and their inverse operations need to be repeated multiple times, depending on the likelihood of the event that we postselect on. We emphasize that this overhead is only present when running this model on a quantum device; classically, since we have access to the full statevector, we can postselect by multiplying with a projector, and renormalizing the state. This is also how the quantum neuron is implemented as a pytorch layer.

Amplitude amplification is a generic variant of Grover search, described in detail e.g. in [NC10]. In brief, a state  $|\psi'\rangle = \alpha |0\rangle |x\rangle + \sqrt{1 - \alpha^2} |1\rangle |y\rangle$  (for normalised  $|x\rangle, |y\rangle$ ) has a likelihood  $\propto |\alpha|^2$  to be measured in state  $|0\rangle |x\rangle$ . Amplitude amplification allows the state to be manipulated such that this probability can be bumped close to 1.

More precisely, the variant of amplitude amplification we utilise is called “fixed-point oblivious amplitude amplification” [Tac+19; Gro05], which is suitable for the case where we have a unitary  $U$   $|\psi\rangle = |\psi'\rangle$  that produces the state (which is where “oblivious” comes from); and where we do not know  $\alpha$  (which is where “fixed-point” comes from). By repeatedly applying  $U$  and its inverse  $U^\dagger$  in a specific fashion, the likelihood of a subsequent measurement to observe outcome  $|0\rangle |x\rangle$  can be amplified to a probability  $\geq 1 - \epsilon$ , with  $O(\log \epsilon / |\alpha|)$  many applications of  $U$ .

### A.1. Quantum Neuron

The first location where amplitude amplification is necessary is in the application of each quantum neuron; the purple meters in fig. 1 in the main text indicate that we would like to measure  $|0\rangle$  on the respective lanes—if a  $|1\rangle$  was measured, a wrong operation results. As discussed, the authors in [CGA17] named their quantum neuron with a similar structure a repeat-until-success (RUS) circuit. Such circuits generally have the feature that the “recovery” operation is simple, which essentially means that one can just “flush and repeat” the operation until it finally succeeds. This is true for their first degree quantum neuron, as it is for our higher-degree quantum neuron—but only on the Hilbert space spanned by product states (e.g. on inputs like  $|1\rangle |0\rangle$ , but not states like Bell pairs such as  $(|00\rangle + |11\rangle)/\sqrt{2}$ ). This means that, without modification, a quantum neuron as proposed in [CGA17] cannot be lifted to a RUS circuit on the full Hilbert space of input states. The necessary modification was proposed in [Tac+19]: amplify the 0 measurement outcome. This means that (almost)

never encounters the situation where one would have to correct an invalid application of the quantum neuron; as the issue with superposition states only ever occurs when a 1 is measured, the quantum neuron—when postselected on measuring 0 every time—works as intended on the full Hilbert space of input states.

Since we can detect failure (i.e. measuring 1), we can simply choose the likelihood of measuring 0—i.e.  $1 - \epsilon$ —to be such that we do not fail too often; and in case of a failure simply repeat the entire QRNN run. Due to the logarithmic dependence on  $\epsilon$  in the time complexity of fixed-point amplitude amplification this is possible with an at most logarithmic overhead in  $\epsilon$  and the number of postselections to be done.

But what is the overhead with respect to  $\alpha$ ? I.e. when applying a quantum neuron, how many times do we have to apply the neuron and its inverse in order to be able to apply the intended nonlinear transformation in eq. (2) in the main text? A loose bound can be readily derived as follows. The “good” overall transformation which we wish to postselect on is a map

$$|0\rangle \mapsto \cos(\theta)^{2^{\text{ord}}} |0\rangle + \sin(\theta)^{2^{\text{ord}}} |1\rangle =: |x\rangle \quad \text{with} \quad \| |x\rangle \|^2 = \cos(\theta)^{2 \times 2^{\text{ord}}} + \sin(\theta)^{2 \times 2^{\text{ord}}}.$$

As we treat the order  $\text{ord}$  of the neuron as a constant (it is a hyperparameter, and it is not beneficial to think about its scaling; choices of  $\text{ord} \in \{1, 2, 3, 4\}$  seem sensible) is easy to derive  $\| |x\rangle \|^2 \geq 1/2^{\text{ord}^2-1}$ —which results in an amplitude amplification overhead of about 2, 8, 128, or 32768 for  $\text{ord} = 1, 2, 3, 4$ , respectively. For all our experiments we chose  $\text{ord} = 2$ ; this choice is based on empirical evidence, and the fact that the activation function for  $\text{ord} = 2$ —shown as the dashed line in Fig. 5 in the main text—features relatively steep slopes around  $\theta = \pi/4$  and  $3\pi/4$ ; and a relatively flat plateau around 0 and  $\pi/2$ .

## A.2. QRNN Cell Output

The second point where we amplify is during training. For each application of the QRNN cell as depicted in **[fig:qrnn]** in the main text, we write the input bit string onto the in/out lanes with a series of classically-controlled bit flip gates. After this, a series of stages process the new input together with the hidden cell state. Each of the gates therein can be *conditioned* on the input, but *do not* change the in/out lane at all (see fig. 4). This is crucial: if e.g. the input bit string was **0110**, the overall state of the QRNN after the input has been written is  $|0110\rangle |h\rangle$ , where  $|h\rangle$  represents the hidden state. The subsequent controlled lanes thus cannot create entanglement between the  $|0110\rangle$  state and  $|h\rangle$ , as  $|0110\rangle$  is not in a superposition. This allows us to reset the in/out lanes with an identical set of bit flips that



entered the bit string in first place; resulting in a state  $|0000\rangle|h'\rangle$  right at the start of the output stage. The output neurons can then utilize this clean output state to write an output word, which is measured.

It is this output word that we perform postselection on during training. For instance, if the character level QRNN is fed an input string (e.g. ascii-encoded lower-case English letters) *fisheries*, then after having fed the network *fish* the next expected letter is a *e*. Yet, at this output stage, all the QRNN does is to present us with a quantum state; measuring the output word results in a distribution over predicted letters, much like in the classical case for RNNs and LSTMs.<sup>4</sup> Depending on which outcome is measured, this means a different hidden cell state is retained: if—for our example the state at the end of the output stage in fig. 4 is

$$|\psi\rangle = p_a |a\rangle |h_a\rangle + p_b |b\rangle |h_b\rangle + \dots + p_z |z\rangle |h_z\rangle,$$

then measuring *z* collapses the QRNN cell state to  $|h_z\rangle$ ; measuring *q* collapses it to  $|h_q\rangle$ .

This is a useful feature during inference: if one measures a certain letter, we expect the internal state of the QRNN to reflect this change.<sup>5</sup> This natural source of randomness is e.g. the basis for the different handwritten digits produced in fig. 8: a measurement of a white pixel determines the likelihood of measuring consecutive white or black pixels further down the line, recreating what is either a '0' or a '1'.

Yet while this collapse of the statevector during inference is a useful feature, during training this results in poor performance, as the output distribution is not predictive enough yet to give any meaningful correlation between measured output and resulting internal state.

To circumvent this, we postselect on the next letter that we expect—e.g. in the above example of *fisheries* we would postselect on finding the letter *e*. One point to emphasize here is that it suffices to repeat the *current* QRNN cell unitary (and its inverse) for the amplitude amplification steps; one does not have to iteratively apply the entire QRNN up to that point; the latter would necessarily result in an exponential runtime overhead. This is not the case here.

We chose to analyse the resulting amplitude amplification overhead only empirically, and implemented a monitoring feature into pytorch that allowed us to, at any point in time, track

---

<sup>4</sup>As explained in the main text, depending on whether we run this QRNN on a classical computer or a quantum device, we can either extract these probabilities by calculating the marginal of the statevector—which is done in our pytorch implementation—or by sampling. The sampling overhead naturally depends on the precision to which one wishes to reproduce the distribution.

<sup>5</sup>Note how this change is due to the collapse of an entangled state by simply measuring the output lanes; we never actively modify the cell state.

the minimum postselection probability that *would* result in an overhead if running the QRNN on a quantum device.

We found three trends during our experiments.

1. The overall postselection overhead was relatively small, but tends to be larger the wider the in/out lanes.
2. For memorization tasks or learning simple sequences (e.g. Elman’s XOR test), the overhead started larger, but then converged to one.
3. For learning more complicated sequences as e.g. the pixel-by-pixel MNIST learning task, the postselection probability converged to roughly a constant  $> 1$ .

Examples for the postselection overheads during two representative training tasks are plotted in fig. 11.

## B. QRNN Network Topology

As explained in Sec. 4.2 in the main text, we used Elman’s task of learning sequences comprising the three words “ba”, “dii” and “guuu” to assess what network topologies work best in this scenario; i.e., we ask the question of how many work stages within the QRNN cell are useful, and what influence the neuron degree<sup>6</sup> and workspace size has on the learning speed.

Our findings are summarised in fig. 12. The input for this task has a width of three bits (which suffices for the six different letters used), so the useful degree in the input stage in the QRNN cell is upper-bound by three. A higher degree becomes useful only if the workspace size is increased accordingly.

Despite this, we found that a degree of two is already optimal; a degree of three is not better, and a degree of four has a longer expected convergence time again. This is likely due to the larger number of parameters necessary for higher-degree neurons, as explained in Fig. 2 in the main text.

A similar picture can be seen when looking at the number of work stages in the QRNN cell: a single stage takes considerably longer than two stages; for more stages, the learning time increases again.

In contrast to this, it appears that the more workspace we have present the better; yet even here there appears to be a plateau when going to  $\geq 6$  qubits. This is likely due to the

---

<sup>6</sup>As a reminder, and as explained in the last section, we set our neurons to have *order* 2. The *degree* of the neuron is the degree of the polynomial of the inputs as shown in eq. (3) in the main text.

simplicity of the learning task. For instance, as listed in Tab. 1 in the main text, a workspace of six was enough to classify MNIST when using data augmentation (which, with an input width of 2 bits, and an order 2 neuron requires two ancillas, resulting in 10 qubits overall). On the other hand, the pixel-by-pixel task required a higher information capacity; we found that a workspace of eight performed better in this setting.

So in general, and as in the case of classical neural networks, there must be a tradeoff between the number of parameters and the expected learning time. Too few parameters and the model does not converge. Too many parameters become costly, and potentially start to overfit the dataset. While the QRNN workspace size has a direct analogy to layer width in classical RNNs and other network architectures, and stages with the depth of the network, the quantum neuron's degree finds no good analogy in common neural network architectures.

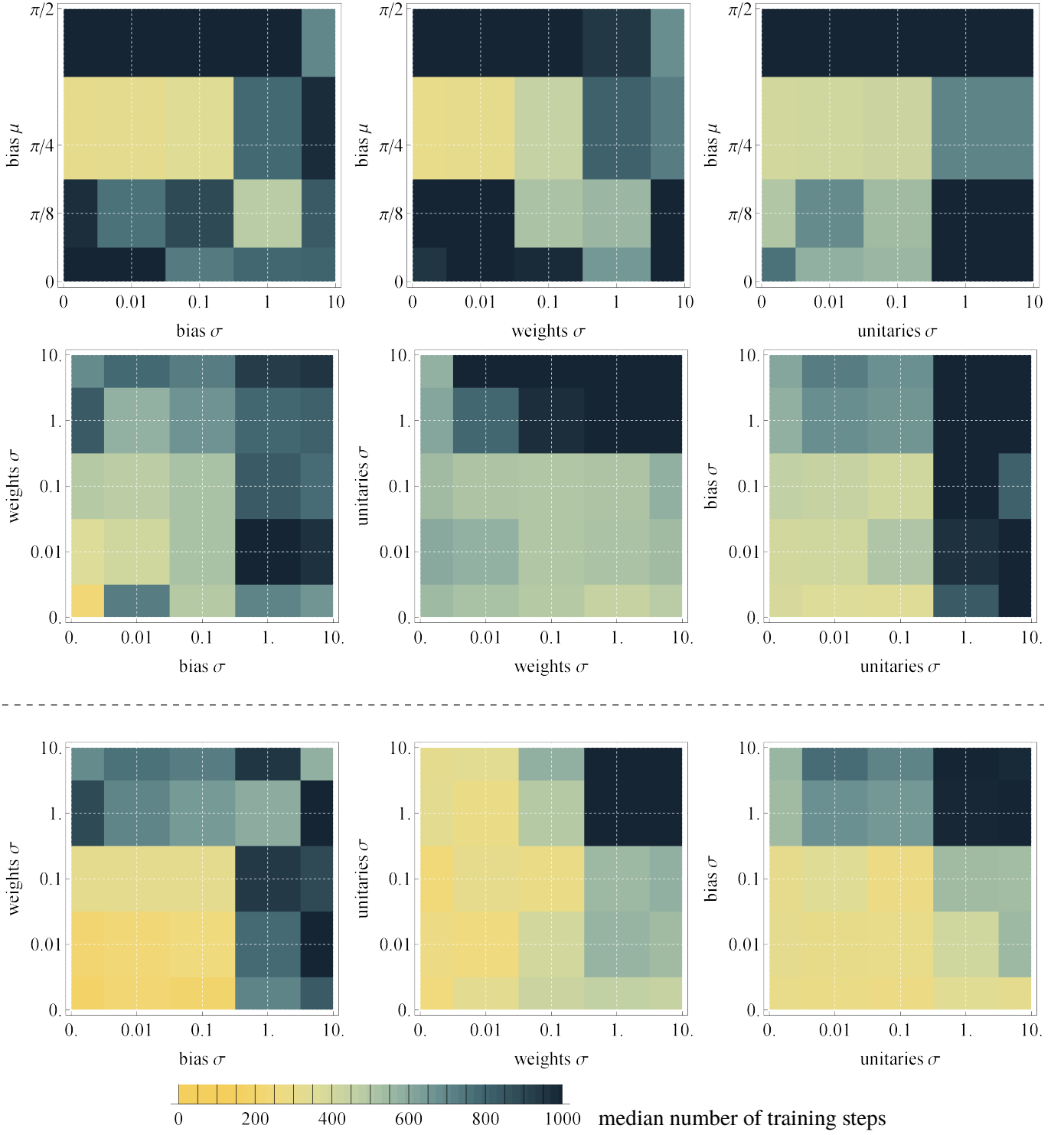


Figure 10: Pairwise comparison of median convergence time for hyperparameter initialization of QRNN cell as described in section 4.2. A cell bias  $\mu = \pi/8$  yields the fastest convergence; shown in the third row are the same parameter comparisons as in the second row, but only considering runs with  $\mu = \pi/8$ .

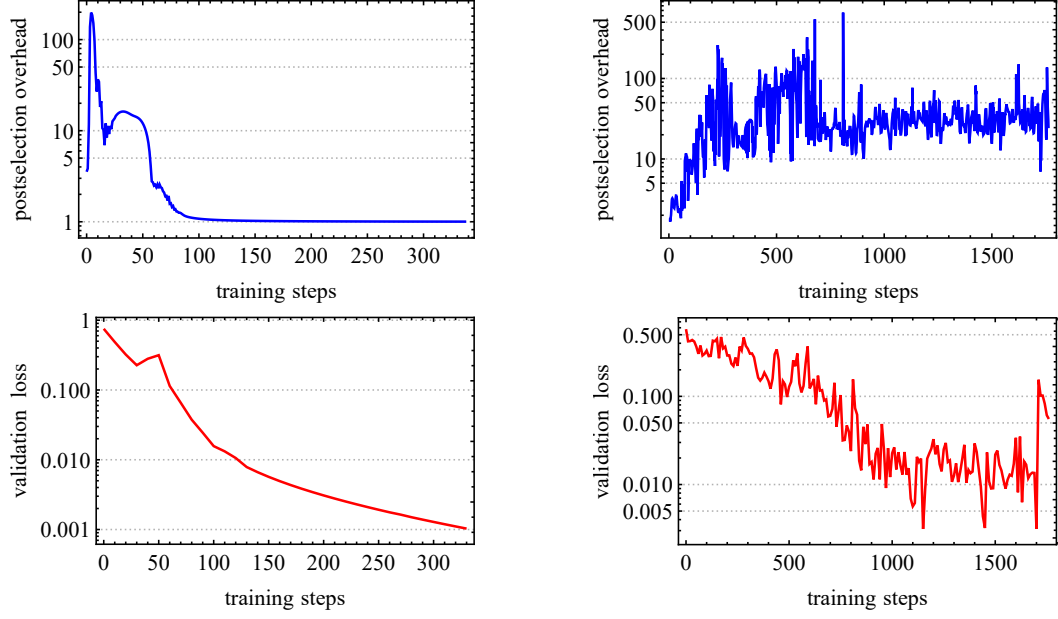


Figure 11: Typical amplitude amplification overhead during training. Left: memorization of simple sequences as described in section 4.1; the overhead approaches one as the validation loss converges to zero. Right: pixel-by-pixel MNIST classification from section 4.3; the overhead stabilises around a constant of  $\approx 40$  as the validation loss decreases.

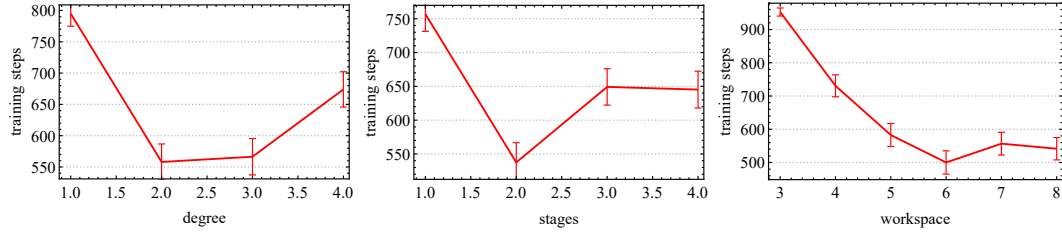


Figure 12: Average number of training steps for sentence learning task described in section 4.1, for various combinations of neuron degree, neuron stages, and workspace. For this task, we found that a combination of degree 2, workspace 6, and 2 stages performed best.