Recurrent Quantum Neural Networks
Supplementary Information

June 11, 2020

1 QRNN Postselection Analysis

A QRNN as described in Sec. 2 in the main text 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 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 wavevector, we can postselect by multiplying with a projector, and renormalizing the state. This is 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 $\mathbf{U}|\psi\rangle = |\psi'\rangle$ that produces the state (which is where "oblivious" comes from); and where we do not know α (which is where "fixed-point" comes from). By repeatedly applying $\mathbf{U}$ and its inverse $\mathbf{U}^\dagger$ 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 $\mathrm{O}(\log\epsilon/|\alpha|)$ many applications of $\mathbf{U}$.

1.1 Quantum Neuron

The first location where this 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 RUS (repeat-until-success) circuit. Such circuits generally have the feature that the "recover" operation is simple, and 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 $(|00\rangle + |11\rangle)/\sqrt{2}$).

The circuit cannot really be "lifted" to a RUS circuit on the full Hilbert space of input states. Instead, we amplify the 0 measurement outcome, essentially postselecting on measuring 0 every time. 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 operation. Due to the logarithmic dependence on ϵ in the time complexity of fixed-point amplitude amplification this is possible with an at most logarithmic overhead in ϵ and the number of postselections to be done.

But what is the overhead with respect to α? 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

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

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

1.2 QRNN Cell Output

The second point where we amplify is during training. For each application of the QRNN cell as depicted in Fig. 4 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. 3). This is crucial: if e.g. the input bit string was `0110`, the overall state of the QRNN after the input is written is $|0110\rangle |h\rangle$, where $|h\rangle$ is 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 originally wrote the bit string; 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.[1] 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. 3 is

$$|\psi\rangle = p_\mathtt{a}\,|\mathtt{a}\rangle\,|h_\mathtt{a}\rangle + p_\mathtt{b}\,|\mathtt{b}\rangle\,|h_\mathtt{b}\rangle + \ldots + p_\mathtt{z}\,|\mathtt{z}\rangle\,|h_\mathtt{z}\rangle\,,$$

then measuring `z` collapses the QRNN cell state to $|h_\mathtt{z}\rangle$; measuring `q` collapses it to $|h_\mathtt{q}\rangle$.

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.[2] 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 runtine 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 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. 1.

2 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[3] and workspace size has on the learning speed.

Our findings are summarised in fig. 2. 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.

---

[1] 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 wavevector—which is done

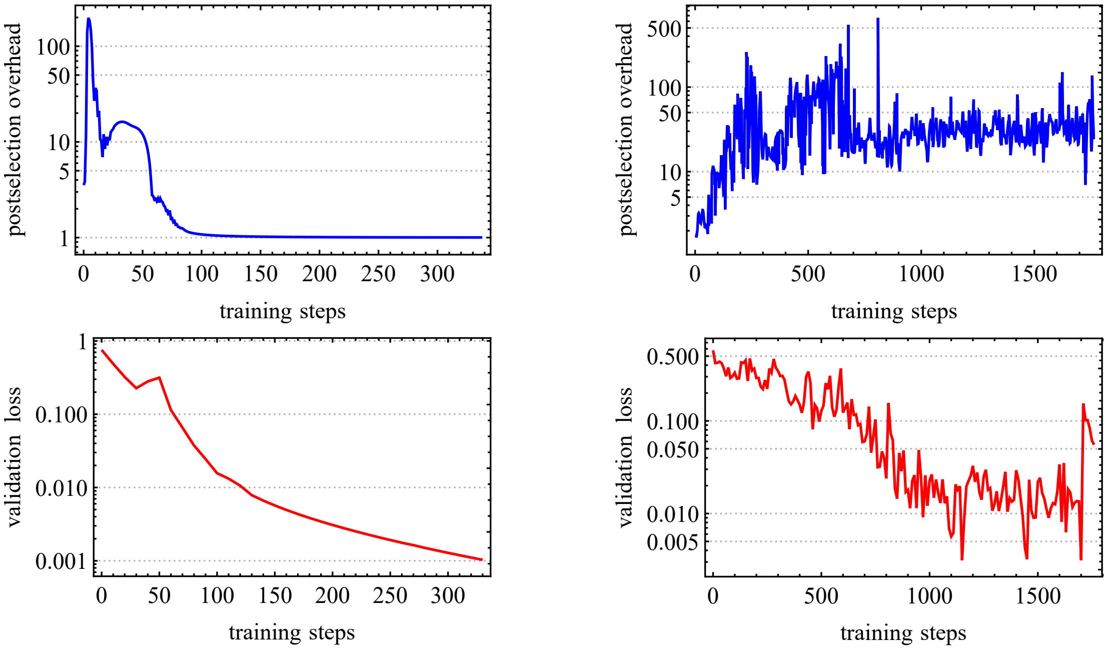

Figure 1: Typical amplitude amplification overhead during training. Left: memorization of simple sequences as described in Sec. 4.1; the overhead approaches one as the validation loss converges to zero. Right: pixel-by-pixel MNIST classification from Sec. 4.3; the overhead stabilises around a constant of ≈ 40 as the validation loss decreases.

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 ≥ 6 qubits. This is likely due to the 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.

---

in our pytorch implementation—or by sampling. The sampling overhead naturally depends on the precision to which one wishes to reproduce the distribution.

[2] 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.

[3] 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.

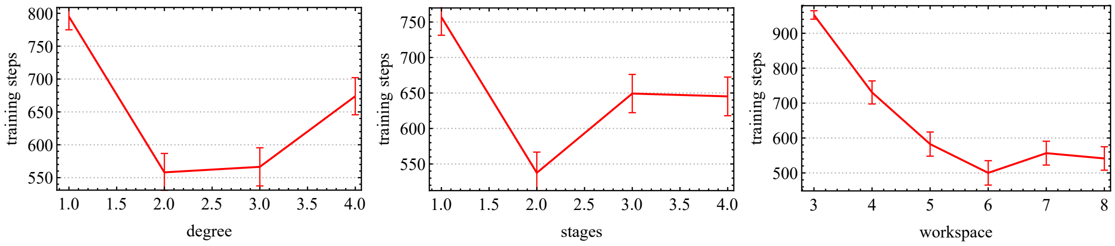

Figure 2: Average number of training steps for sentence learning task described in Sec. 4.2, for various combinations of neuron degree, neuron stages, and workspace. For this task, we found that a combination of degree 2, workspace 6, and 2 stages performed best.

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.

References

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press, 2010, p. 676.

[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.

[Gro05] Lov K. Grover. "Fixed-Point Quantum Search". In: *Physical Review Letters* 95.15 (Oct. 2005), p. 150501.

[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.