QRNN Updates

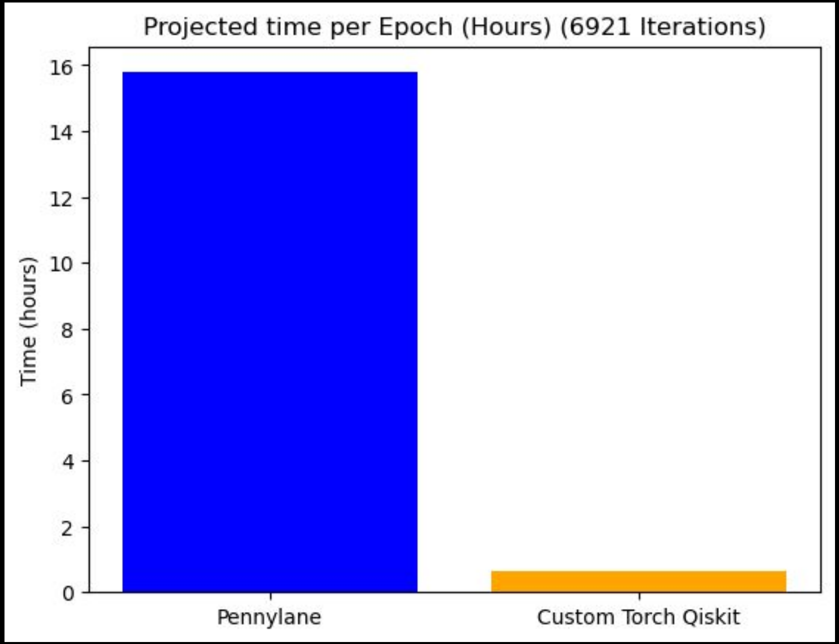
Erik C and Michael Stewart

Initial Implementation

- Used Pennylane to create an initial implementation of the hybrid QRNN model.
- Pennylane offers many tools out of the box but does not have the flexibility required for a quantum circuit of this type.
  - In particular, support for QCs that have mid-circuit measurements and non-unitary operations is not well supported, and very slow
  - Because of this, weight updates for a single data sequence took roughly ~8 seconds on a CPU, making an entire epoch ~16 hours
- To alleviate this, a custom torch NN model using IBM Qiskit was created as a comparison point

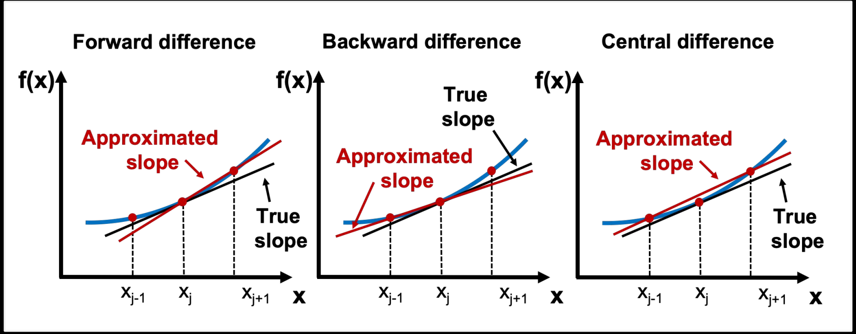
Pennylane vs IBM Custom model

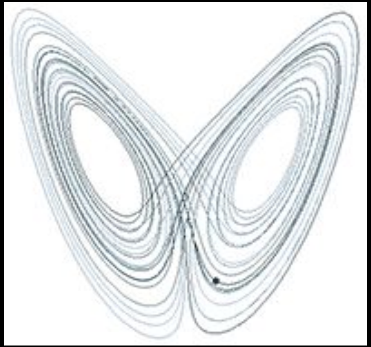
- With the first attempt PyTorch IBM Custom Model, there is already an order-of-magnitude improvement of train time using CPU.
- Weight updates with the custom model only take ~.33 seconds, making it possible to perform an epoch of updates in < 1 hour
- Small 6 qubit circuits were compared for training speed.



Current Pitfalls

- The original plan was to test the SPSA training algorithm vs Parameter-shift on the custom QRNN model, but it has now been understood that parameter-shift cannot be used with non-unitary QCs, which our circuit is one of,
    - **Solution:** We will compare **SPSA** vs **Finite-differences** instead.
- We originally planned to test the chaotic Kuramoto-Sivashinksy system, with 128 DOFs (degrees of freedom), but are currently running experiments on the simpler Lorenz system as we try to get things working.





SPSA

- Asymptotic runtime of O(N), where N is the number of optimization steps.
- $$\hat{\theta}_{k+1} = \hat{\theta}_k - a_k \hat{g}_k(\hat{\theta}_k),$$

$$\hat{g}_{ki}(\hat{\theta}_k) = \frac{y(\hat{\theta}_k + c_k\Delta_k) - y(\hat{\theta}_k - c_k\Delta_k)}{2c_k\Delta_{ki}},$$

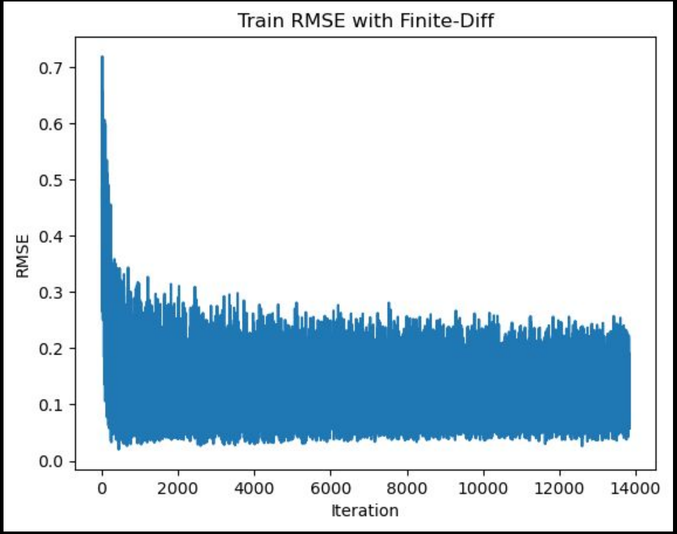
Finite Differences

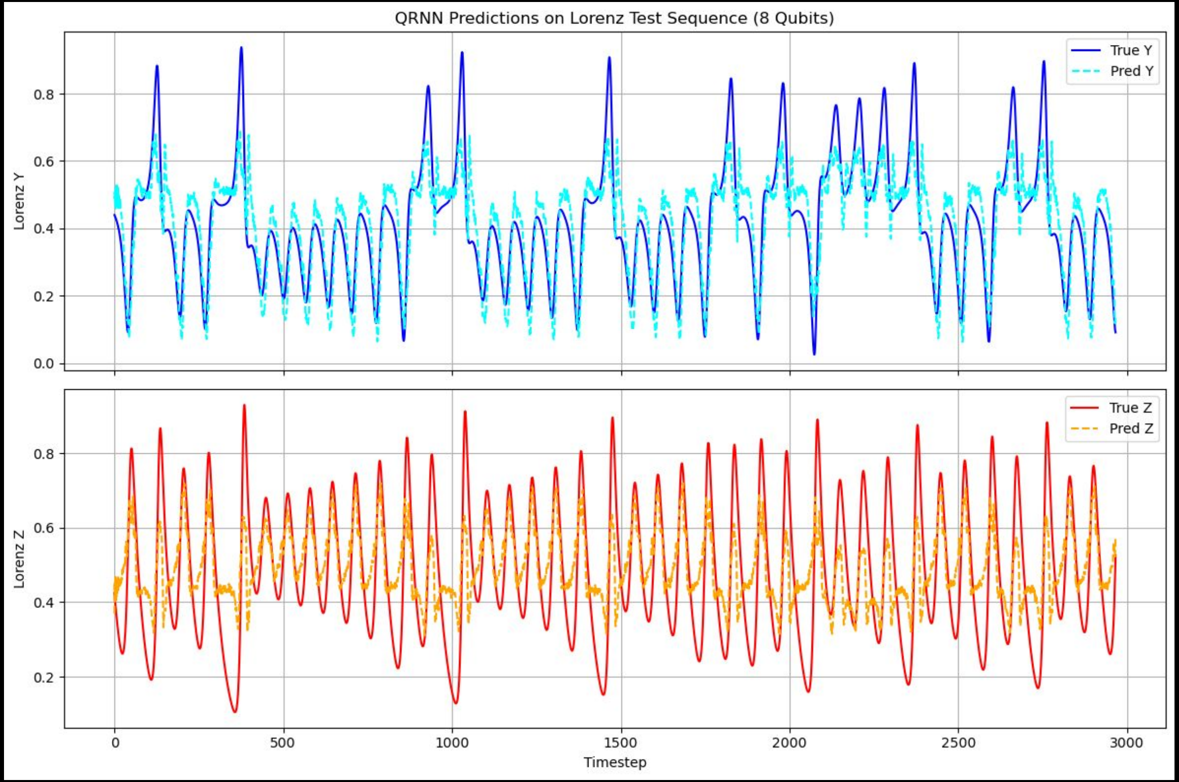
- Asymptotic runtime of O(PN), where P is the number of parameters in the VQC, and N is the number of training steps.
- Formulation is similar, except only one parameter is perturbed at a time, and the cost function must be evaluated for each parameter.

Source: Optimization using SPSA | PennyLane Demos

Initial Results

Test RMSE: 0.116425





Next steps

1. Determine which hyperparameters lead to the best performance with each training method.
2. Return back to the more difficult Kuramoto-Sivashinsky PDE
3. Enable GPU training for more parallelism and faster execution time
4. Configure quantum circuit to a "template" so we can transpile just once and then append values, reducing overhead massively