A QCNN for Quantum State Preparation

Carnegie Vacation Scholarship

David Amorim

Weeks 1-3
(01/07/2024 - 19/07/2024)

Table of Contents

Background

- Hayes 2023[1] presents a scheme to prepare a complex vector $\boldsymbol{h} = \{\tilde{A}_j e^{i\Psi(j)} | 0 \leq j < N\}$ as the quantum state

$$|h\rangle = \frac{1}{|\tilde{A}|} \sum_{j=0}^{2^n-1} \tilde{A}(j) e^{i\Psi(j)} |j\rangle \, , \tag{1}$$

  using $n = \lceil \log_2 N \rceil$ qubits
- This requires operators $\hat{U}_A$ and $\hat{U}_\Psi$ such that

$$\hat{U}_A |0\rangle^{\otimes n} = \frac{1}{|\tilde{A}|} \sum_{j=0}^{2^n-1} \tilde{A}(j) |j\rangle \, , \tag{2}$$

$$\hat{U}_\Psi |j\rangle = e^{i\Psi(j)} |j\rangle \tag{3}$$

[1] `https://arxiv.org/pdf/2306.11073`

Background

- $\hat{U}_\Psi$ is constructed via an operator $\hat{Q}_\Psi$ that performs function evaluation in an ancilla register:

$$\hat{Q}_\Psi |j\rangle |0\rangle_a^{\otimes m} = |j\rangle |\Psi'(j)\rangle_a \,, \tag{4}$$

  with $\Psi'(j) \equiv \Psi(j)/2\pi$
- Due to rounding the size, m, of the ancilla register limits the precision with which Ψ' can be encoded to $\sim 2^{-m}$
- Currently, $\hat{Q}_\Psi$ is implemented using gate-intensive *linear piecewise functions (LPFs)*

Background

Aim

Implement $\hat{Q}_\Psi$ in a gate-efficient way using a parametrised quantum circuit (PQC)

Remark

The n-qubit register containing the $|j\rangle$ and the m-qubit register containing the $|\Psi'(j)\rangle$ will be referred to as the input register and target register, respectively.

Table of Contents

Approach: a QCNN

- A *quantum convolutional neural network* (QCNN) is used to tackle the problem
- A QCNN is a parametrised quantum circuit involving multiple layers
- Two types of network layers are implemented:
  - Convolutional layers (CL) involve multi-qubit entanglement gates
  - Input layers (IL)[2] involve controlled single-qubit operations on target qubits
- Input qubits only appear as controls throughout the QCNN

[2] Replacing the conventional QCNN *pooling layers*

Convolutional Layers (CLs)

- Each CL involves the cascaded application of a two-qubit operator on the target register
- A general two-qubit operator involves 15 parameters
- To reduce the parameter space the canonical three-parameter operator

$$\mathcal{N}(\alpha, \beta, \gamma) = \exp\left(i\left[\alpha X \otimes X + \beta Y \otimes Y + \gamma Z \otimes Z\right]\right) \tag{5}$$

  is applied, at the cost of restricting the search space
- This can be decomposed[3] into 3 CX, 3 $\mathrm{R_z}$, and 2 $\mathrm{R_y}$ gates
- A two-parameter real version, $\mathcal{N}_{\mathbb{R}}(\lambda, \mu)$, can be obtained by removing the $\mathrm{R_z}$

[3] https://arxiv.org/pdf/quant-ph/0308006

Convolutional Layers (CLs)

- Two types of convolutional layers are implemented:[4]
  - Neighbour-to-neighbour/linear CLs: the $\mathcal{N}$ (or $\mathcal{N}_{\mathbb{R}}$) gate is applied to neighbouring target qubits
  - All-to-all/quadratic CLs: the $\mathcal{N}$ (or $\mathcal{N}_{\mathbb{R}}$) gate is applied to all combinations of target qubits
- The $\mathcal{N}$-gate cost of neighbour-to-neighbour (NN) layers is $\mathcal{O}(m)$ while that of all-to-all (AA) layers is $\mathcal{O}(m^2)$
- The QCNN uses alternating linear and quadratic CLs

[4] Loosely based on Sim 2019 (`https://arxiv.org/pdf/1905.10876`)

Input Layers (ILs)

- ILs, replacing pooling layers, feed information about the input register into the target register
- An IL involves a sequence of controlled generic single-qubit rotations (CU3 gates) on the target qubits, with input qubits as controls
- For an IL producing states with real amplitudes, the CU3 gates are replaced with $\mathrm{CR_y}$ gates
- Each input qubit controls precisely one CU3 (or $\mathrm{CR_y}$ operation), resulting in an $\mathcal{O}(n)$ gate cost (no CX gates!)
- ILs are inserted after every second convolutional layer, alternating between control states 0 and 1

Summary: QCNN Structure

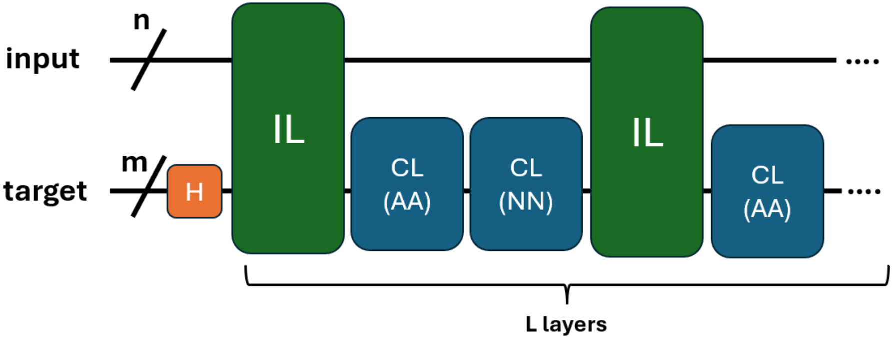

Figure 1: Schematic of QCNN structure

Table of Contents

Training the QCNN

- For training, the QCNN is wrapped as a *SamplerQNN* object and connected to PyTorch's Adam optimiser via *TorchConnector*
- The optimiser determines improved parameter values for each training run (epoch) based on the loss between output and target state
- Beyond loss, mismatch is an important metric:

$$\mathrm{M} = 1 - |\langle \psi_{\text{target}} | \psi_{\text{out}} \rangle| \tag{6}$$

- There are two ways to train the QCNN on input data:[5]
  1. Training on individual states
  2. Training in superposition

[5] One can also train the QCNN to produce a target distribution independent of the input register, which is equivalent to constructing $\hat{U}_A$

Training the QCNN

1. Training on Individual States

- One of the 2^n input states, $|j\rangle$, is randomly chosen each epoch
- The network is taught to transform $|j\rangle|0\rangle \mapsto |j\rangle|\Psi'(j)\rangle$ for each of the states individually

2. Training in Superposition

- The same input state is chosen each epoch
- The network is taught to transform

$$\left(\frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}|j\rangle\right)|0\rangle \mapsto \frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}|j\rangle|\Psi'(j)\rangle \tag{7}$$

- By linearity, this teaches the network to transform $|j\rangle|0\rangle \mapsto |j\rangle|\Psi'(j)\rangle$ for each $|j\rangle$

Table of Contents

Initial Tests

- Initial tests need to be carried out to inform QCNN design choices regarding:
  a. Number of layers
  b. Number of epochs
  c. Training mode (individually versus in superposition)
  d. Use of $\mathcal{N}$ and CU3 versus $\mathcal{N}_{\mathbb{R}}$ and $\mathrm{CR_y}$
  e. Choice of loss function
  f. Network structure
- This constitutes a large parameter space that is difficult to explore systematically
- Overly simple benchmark problems (e.g. $n = m = 2$, $\Psi(x) = x$) do not extrapolate well to more general cases
- Thus, tests are carried out for simplified versions of Ψ in the context of Hayes 2023 ($n = 6$, $m \geq 3$)

Initial Tests

- Heuristically: train in superposition with real circuits ($\mathcal{N}_{\mathbb{R}}$ and $\mathrm{CR_y}$) of depth $L = 6$ using 600 epochs and focus on optimising the loss function
- Best results achieved with *cross entropy* (CE) and *sign-adjusted mismatch* (SAM):

$$\mathrm{SAM}(x, y) = 1 - \sum_j |x_j||y_j| \tag{8}$$

- SAM is tailored to reduce mismatch and enforce positive amplitudes

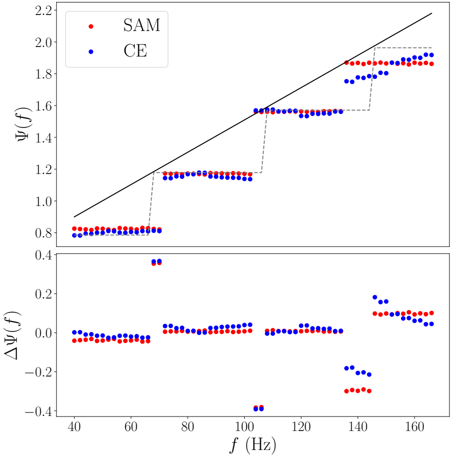

Figure 2: Comparison of loss functions for $\Psi \sim x$ and $m = 4$. Target in black; rounded target dashed in grey.

Initial Tests

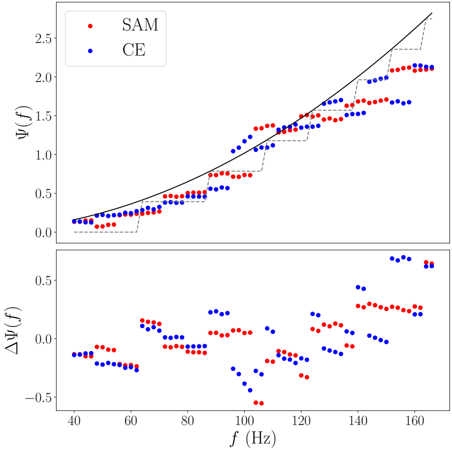

Figure 3: Comparison of loss functions for $\Psi \sim x^2$ and $m = 4$. Target in black; rounded target dashed in grey.

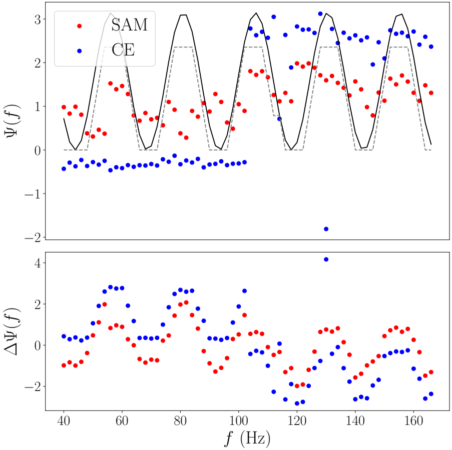

Figure 4: Comparison of loss functions for $\Psi \sim \sin x$ and $m = 3$. Target in black; rounded target dashed in grey.

Initial Tests

- SAM significantly outperforms CE when taking into account state amplitudes
- QCNN performance not much improved by increasing L or number of epochs[6] (no brute force solution)
- Instead implement a weighted loss function, taking into account the features of Ψ: define *weighted mismatch* (WIM) as

$$\text{WIM}(x, y) = 1 - \sum_j w_j |x_j||y_j|, \tag{9}$$

  where the weights $w_j \in \mathbb{R}_+$ may be recomputed between epochs
- The w_j are based on the mismatch between $\hat{Q}_\Psi |j\rangle |0\rangle$ and $|j\rangle |\Psi'(j)\rangle$
- *Despite considerable efforts, WIM could not (yet) improve upon SAM*

[6] Based on just a few tests

Table of Contents

Results

- The following shows how the methods developed so far compare to the ones used in Hayes 2023
- QCNNs are applied to implement both $\hat{U}_A$ and $\hat{U}_\Psi$ for the problem studied in Hayes 2023:

$$\tilde{A}(f) = f^{-7/6}, \tag{10}$$

$$\Psi(f) = c_0 + c_1 f + c_2 f^{-1/3} + c_3 f^{-2/3} + c_4 f^{-1} + c_5 f^{-5/3}, \tag{11}$$

  with $40\,\text{Hz} \leq f \leq 168\,\text{Hz}$ and $n = 6$
- In the paper, $\hat{U}_A$ is implemented via a quantum generative adversarial network (QGAN) as well as the Grover-Rudolph (GR) algorithm while LPFs are used for $\hat{U}_\Psi$

Encoding the Amplitude

- The QCNN outperforms the QGAN and reaches GR w.r.t. mismatch
- The QCNN ($L = 3$, $n = 6$) was trained in 600 epochs with SAM
- GR also requires 22 ancilla qubits

Method	CX	Mismatch
QGAN	100	8.6×10^{-3}
GR	23,796	4.1×10^{-4}
QCNN	72	4.0×10^{-4}

Table 1: Comparison of $\hat{U}_A$ implementations

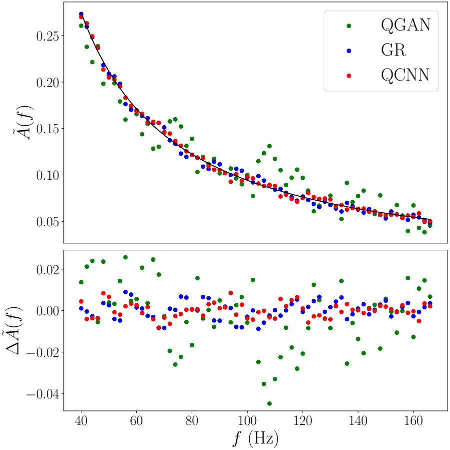

Figure 5: Reconstruction of $\tilde{A}(f)$ from different methods. Target in black.

Encoding the Phase

- The LPFs vastly outperform the QCNN
- The QCNN ($L = 3$, $n = 6$, $m = 4$) was trained in 600 epochs using SAM and involves 76 CX gates
- The LPF approach ($m = 6$) requires 22 ancilla qubits and 9,464 CX gates

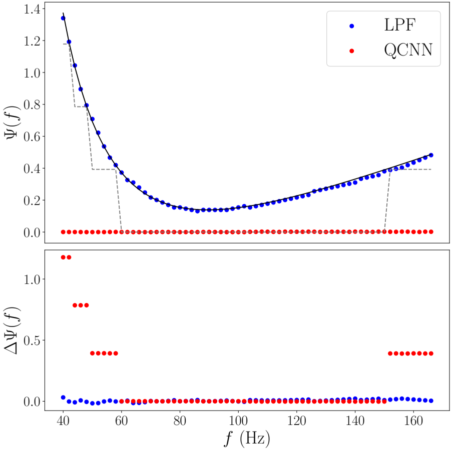

Figure 6: Encoding of $\Psi(f)$ using LPFs versus a QCNN. Target in black; rounded target dashed in grey.

Full Waveform

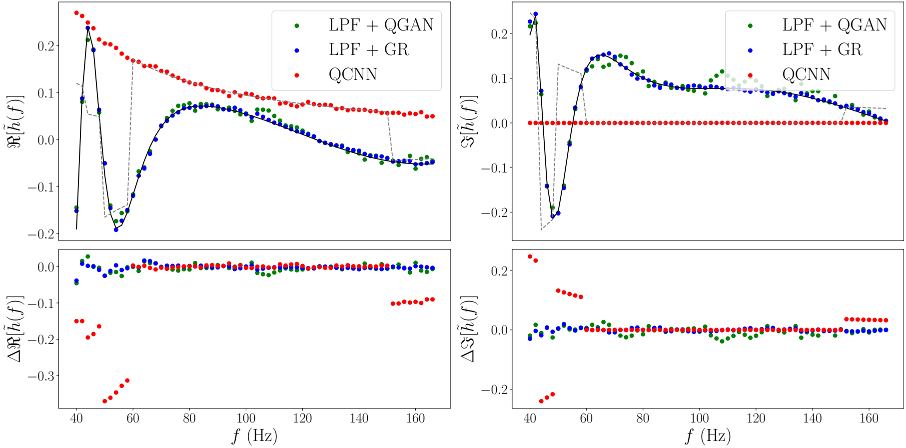

Figure 7: Encoding of $\boldsymbol{h}(f)$ as waveform $\tilde{h}(f)$ using different methods. Target in black; rounded target dashed in grey.

Table of Contents

Next Steps

Priorities:

- Continue work on WIM (develop a distance measure taking into account Ψ, e.g. $w_j \sim L_\Psi(x, y) = \sum_j 2^{j-m}|x_j - y_j|$)
- Change IL structure (all-to-all...)
- Keep parameters fixed for each layer

Others:

- Explore the QCNN parameter space further (e.g. circuit depth, network structure...)
- Look into barren plateau mitigation (e.g. layer-by-layer training, local loss functions...)
- Look into weight re-mapping