

# PQC Function Evaluation

Weeks 1-3

David Amorim

01/07/2024 - 19/07/2024

# Table of Contents

## ① Background

## ② Approach: a QCNN

Convolutional Layers

Input Layers

Summary: QCNN Structure

## ③ Training the QCNN

## ④ Initial Tests

## ⑤ Results

Encoding the Amplitude

Encoding the Phase

Full Waveform

## ⑥ Next Steps

# Background

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

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

using  $n = \lceil \log_2 N \rceil$  qubits

- This requires operators  $\hat{U}_A$  and  $\hat{U}_\Psi$  such that

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

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

---

<sup>1</sup><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:

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

with  $\Psi'(j) \equiv \Psi(j)/2\pi$

- Currently,  $\hat{Q}_\Psi$  is implemented using gate-intensive *linear piecewise functions (LPFs)*

## 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.

# 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)**<sup>2</sup> involve controlled single-qubit operations on target qubits
- Input qubits only appear as controls throughout the QCNN

---

<sup>2</sup>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**

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

is applied, at the cost of restricting the search space

- This can be decomposed<sup>3</sup> into  $3CX$ ,  $3R_z$ , and  $2R_y$  gates
- A two-parameter real version,  $\mathcal{N}_{\mathbb{R}}(\lambda, \mu)$ , can be obtained by removing the  $R_z$

---

<sup>3</sup><https://arxiv.org/pdf/quant-ph/0308006>

# Convolutional Layers (CLs)

- Two types of convolutional layers are implemented:<sup>4</sup>
  - **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

---

<sup>4</sup>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 *CR<sub>y</sub> gates*
- Each input qubit controls precisely one *CU3* (or *CR<sub>y</sub>* 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

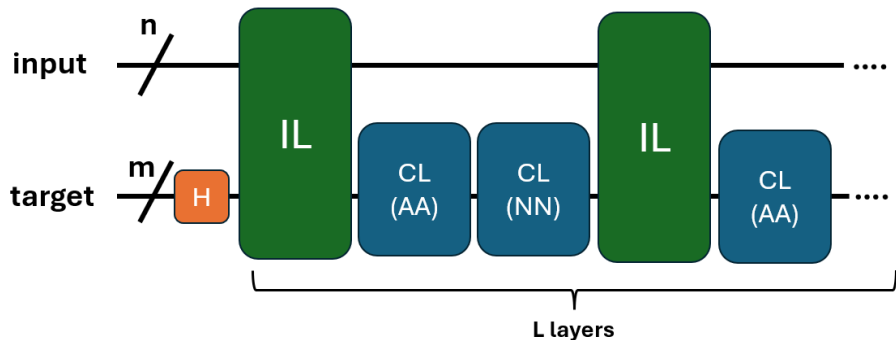


Figure 1: Schematic of QCNN structure

# 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:

$$M = 1 - | \langle \psi_{\text{target}} | \psi_{\text{out}} \rangle | \quad (6)$$

- There are two ways to train the QCNN on input data:<sup>5</sup>
  - ① Training on **individual states**
  - ② Training in **superposition**

---

<sup>5</sup>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

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

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

# Initial Tests

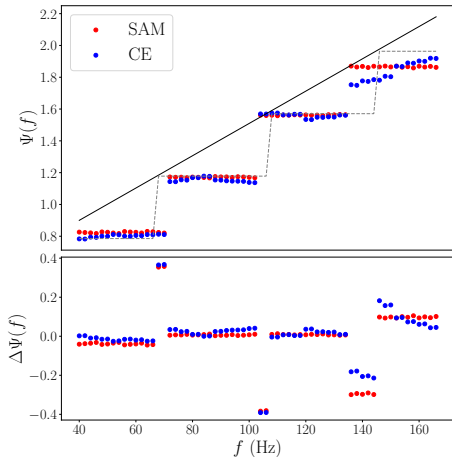
- Initial tests need to be carried out to **inform QCNN design choices** regarding:
  - a Number of layers
  - b Number of epochs
  - c Training mode (individually versus in superposition)
  - d Use of  $\mathcal{N}$  and  $CU3$  versus  $\mathcal{N}_{\mathbb{R}}$  and  $CR_y$
  - e Choice of loss function
  - 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  $\Psi$  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  $CR_y$ ) 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**):

$$\text{SAM}(x, y) = \left| 1 - \sum_j x_j y_j \right| \quad (8)$$

- 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

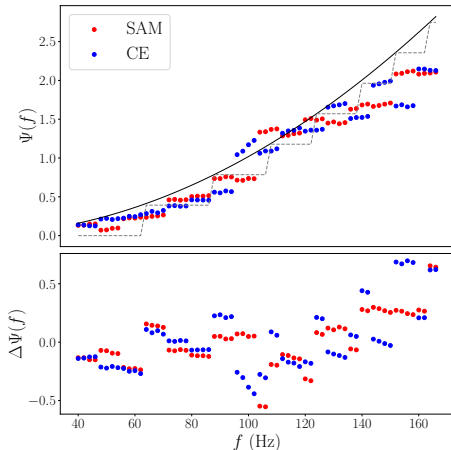


Figure 3: Comparison of loss functions for  $\Psi \sim x^2$  and  $m = 4$ . Target in black; rounded target dashed in grey.

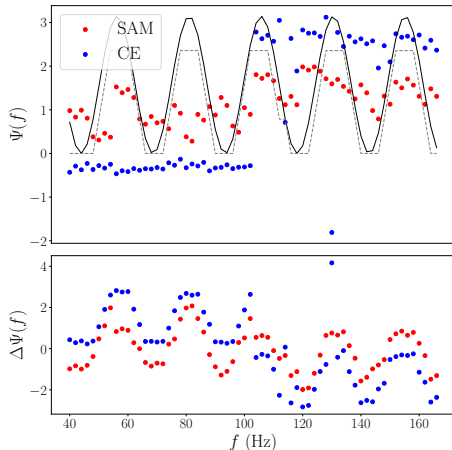


Figure 4: Comparison of loss functions for  $\Psi \sim \sin x$  and  $m = 3$ . Target in black; rounded target dashed in grey.

# Initial Tests

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

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

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  $|j\rangle |\Psi'(j)\rangle$

---

<sup>6</sup>Based on just a few tests



- WIM REDUCES STDEV BUT NOT OVERALL PERFORMANCE...MAKES OUTLIERS A BIT BETTER AND THE REST A BIT WORSE
- SAM STILL BETTER THAN WIM OVERALL!
- maybe: weighted mse / mae better ?? [no! even worse!]
- USE DIFFERENT WEIGHTS ??

- In the following, a QCNN is applied to implement both  $\hat{U}_A$  and  $\hat{U}_\Psi$  for the problem studied in Hayes 2023:

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

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

with  $40 \text{ Hz} \leq f \leq 168 \text{ Hz}$

- 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  $\hat{U}_\Psi$
- Hayes 2023 uses  $n = 6$  as well as 22 ancilla qubits

# Encoding the Amplitude

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

Method	CX	Mismatch
QGAN	100	$8.6 \times 10^{-3}$
GR <sup>7</sup>	23,796	$5.7 \times 10^{-4}$
QCNN	72	$7.1 \times 10^{-4}$

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

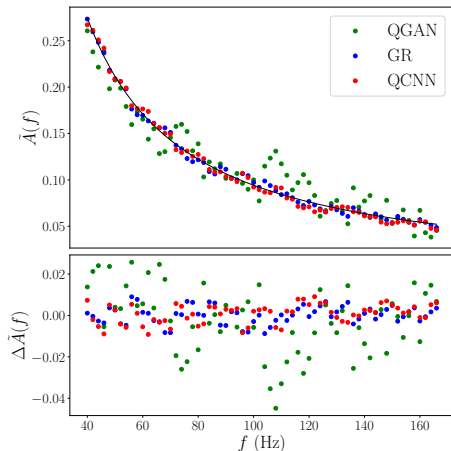


Figure 5: Reconstruction of  $\tilde{A}(f)$  from different methods. Target in black.

<sup>7</sup>Hayes 2023 reports a mismatch of  $4.1 \times 10^{-4}$

# Encoding the Phase

- The implementation of  $\hat{U}_\Psi$  is not the only factor affecting the encoding of  $\Psi(f)$
- The size,  $m$ , of the target register limits the available precision due to rounding to  $\sim 2^{-m}$
- A meaningful representation of  $\Psi(f)$  requires  $m \gtrsim 6$
- The LPF approach in Hayes 2023 uses  $m = 8$

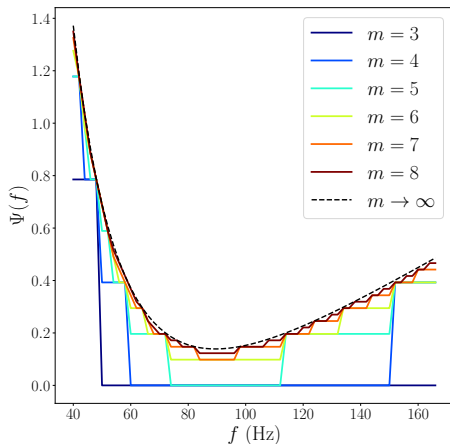
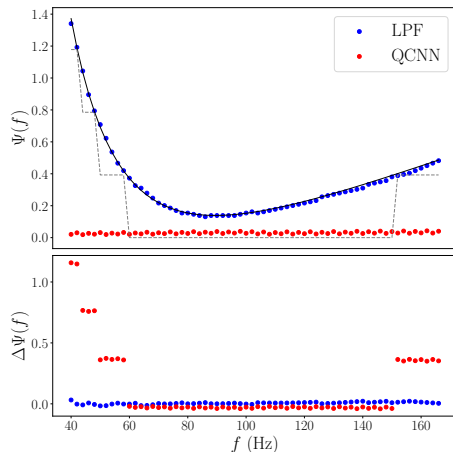


Figure 6: Attainable precision due to rounding for different target register sizes

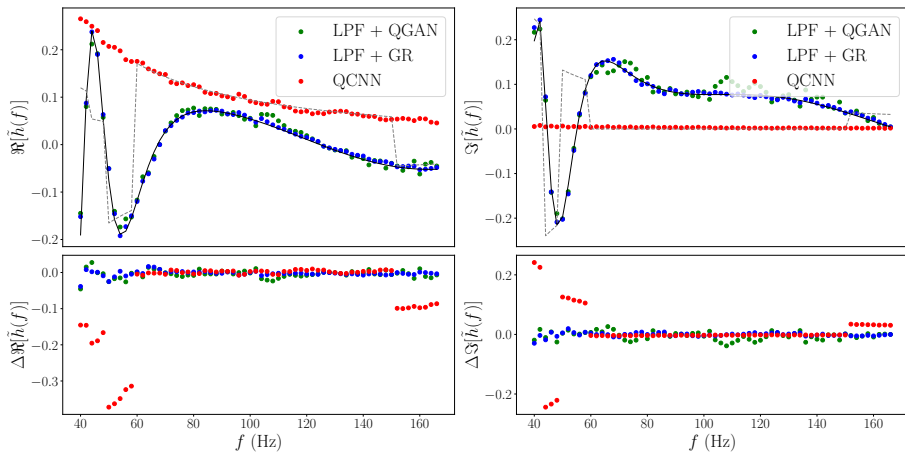
# Encoding the Phase

- ... [improve fig caption]



**Figure 7:** Encoding of  $\Psi(f)$  using LPFs versus a QCNN. Target in black; rounded target dashed in grey.

# Full Waveform



**Figure 8:** Encoding of  $h(f)$  as waveform  $\tilde{h}(f)$  using different methods. Target in black; rounded target dashed in grey.

# Next Steps

- improve wim
- more fully explore PQC parameter space ... (depth, epoch...)
- look into barren plateau mitigation (layer-by-layer training....) =  $i$  might help increase  $L/e$