Hamming weight preserving QCNN for Perceval

Letao Wang, Pan LIU, Yongkang ZOU

Abstract

Followed by the Hamming weight perserving QCNN work of RBS gates [1][2], we can use the same idea to implement it with the Perceval photonic framework. Technically the code for this job isn't finished completely because some questions represent in part 6, but the intuition and the overall idea is similar with the previous work [1] (see references in part 8).

1. Quick Summary of Missions

1. Names of your brave team members - **has been shown**
2. The arcane details of your quantum-enhanced model - **part 2 to part 4**
3. The theoretical scrolls (or intuitions) supporting your expected improvements - **part 2.2, and [1]**
4. Prophecies of how your solution might scale with greater quantum resources - **figure 2 in part 5 [1]**

Reminder before reading:
The idea of this work is highly relevent to the work "Subspace Preserving Quantum Convolutional Neural Network Architectures" [1], we use a lot of ideas of it (Letao Wang is co-author of the paper).
If you meet something you don't understand, you can first take a look at the schematic diagram of our QNN in part 5, it may be helpful.

2. Feature map - from classic to quantum

2.1. Encoding introduction

2.1.1. Fock encoding

It means the default encoding method of Fock state, except we consider a single mode can only contain a single photon, and the value of postselect is always equals to n.

For example, $m = 4, n = 2$, the state dimension should be $\binom{m}{n} = 6$,

basis: $|1100\rangle$ $|1010\rangle$ $|1001\rangle$ $|0110\rangle$ $|0101\rangle$ $|0011\rangle$

2.1.2. HW Preserving Tensor encoding

We use the HW Preserving Tensor encoding definition in [1]. Consider a classical tensor of dimension k such that $x = (x_{1,\ldots,1}, \ldots, x_{d_1,\ldots,d_k}) \in \mathbb{R}^{d_1 \times \cdots \times d_k}$. An amplitude encoding tensor encoding data loader is a parametrized n-qubit quantum circuit (with $n = \sum_{i \in [k]} d_i$ that prepares the quantum states:

$$|x\rangle = \frac{1}{||x||} \sum_{i_1 \in [d_1]} \cdots \sum_{i_k \in [d_k]} x_{i_1,\ldots,i_k} \left|e_{i_1}^{d_1}\right\rangle \otimes \cdots \otimes \left|e_{i_k}^{d_k}\right\rangle$$

where $\left|e_{i_l}^{d_l}\right\rangle = |0 \ldots 010 \ldots 0\rangle$ is a state corresponding to a bit-string with d_l bits and only the bit i_l is equal to 1. Therefore, for any $j \in [k]$, $\left\{\left|e_i^{d_l}\right\rangle \mid i \in [d_l]\right\}$ is a fixed family of d orthonormal quantum states, and $\|\cdot\|$ denotes the 2 -norm of $\mathbb{R}^d$.

For example we can map a 2×2 matrix image x to a state $|x\rangle$ using this encoding:

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix} \longrightarrow |x\rangle = \frac{1}{||x||}(x_{1,1}|1010\rangle + x_{1,2}|1001\rangle + x_{2,1}|0110\rangle + x_{2,2}|0101\rangle)$$

$|x\rangle = \frac{1}{||x||} \sum_{i,j=1}^{2} x_{i,j}|v_i\rangle \otimes |v_j\rangle$ where $|v_i\rangle, |v_j\rangle$ are N-dimensional vectors represent rows and columns repectively, $|v_1\rangle = |10\rangle, |v_2\rangle = |01\rangle$. (See more details in the end of page 3 of [1])

2.2. Important intuitions

In [2] we know how to design a quantum data loader with RBS gates, and the Reconfigurable Beam Splitter (RBS) gate is a 2-qubit gate that corresponds to a θ-planar rotation between the states $|01\rangle$ and $|10\rangle$:

$$RBS(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \text{ bases are } |11\rangle, |10\rangle, |01\rangle, |00\rangle$$

This looks similar with Perceval BS gates, we assume they have some similar intuitions.

First important intuition is, local gates limit expressivity. We can calculate the Dynamical Lie Algebra dimension of local gates (the RBS gate that connects neighbor modes), it turns out that local gates have poor expressivity when $n > 1$, we shouldn't use only local gates. Although Perceval gates differ fromRBS gates, this intuition should remain valid, and we can also verify it by calculation. We only need to know the 4 × 4 matrix form of the Perceval BS gates to proceed. (See “B. Existence of Quantum Data Loader” in page 3 of [2] for details.)

Another intuition is, HW Preserving Tensor encoding is good for convolutional layers.

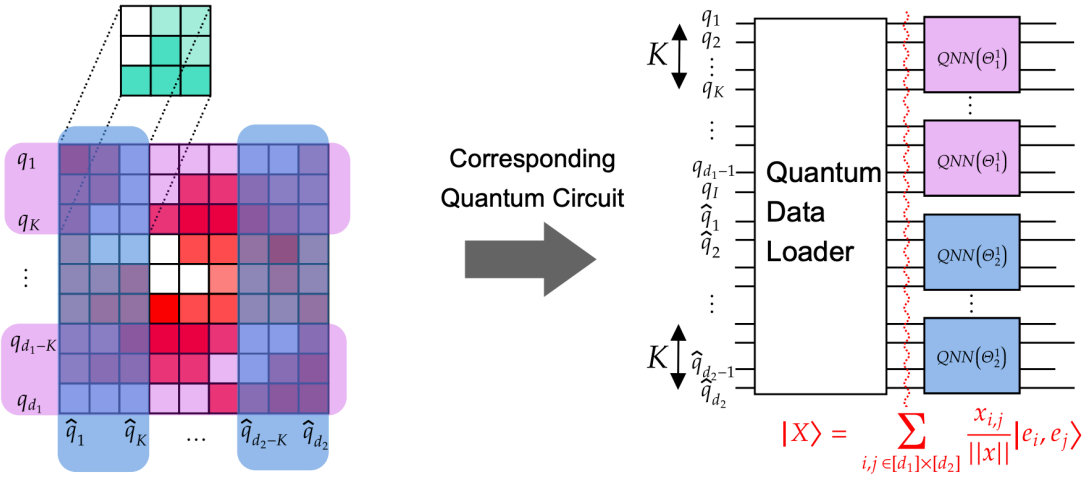


Figure 3: A 2 dimensional convolutional layer using HW preserving quantum circuits and tensor encoding.

From this figure in [1], suppose we have a good quantum data loader. The pink $\mathrm{QNN}\left(\Theta_1^1\right)$ means a row convolutional kernel, and the blue $\mathrm{QNN}\left(\Theta_2^1\right)$ means a column convolutional kernel, we apply them together to get a good kernel. This can simulate the Convolutional layers of CNN very well.

(See "Hamming-Weight Preserving Convolutional Layer" in page 4 of [1] for details)

2.3. Feature map implementation

2.3.1. method to build the quantum data loader

In [2] we know that finding a perfect data loader for $n > 1$ is hard, so we propose that we consider it as an optimisation problem, so we use gates with trainable parameters to encode our data, now our question is which gate layout provides good trainability.
(See page 3 and Appendix A.2 in [2] for more details)

2.3.2. Find a good gate layout

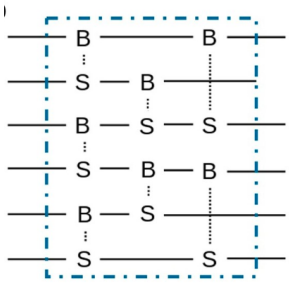


Figure 1: The gate layout of this circuit is [(0,1),(2,3),(4,5),(1,2),(3,4),(0,2),(3,5)], this has full DLA rank when $n = 1$ and $n = 2$, means we can repeat this layout to reach the maximum QFIM rank.

We believe using only local gates is not sufficient , but how do we evaluate a good gate layout? QFIM (Quantum Fisher Information Matrix) rank.

Let us consider an initial state $|e_s\rangle$ and $U(\Theta)$ the unitary that represents a quantum circuit with $\Theta = \{\theta_1, \ldots, \theta_D\}$ the set of variational parameters. The Quantum Fisher Information Matrix (QFIM) is a $D \times D$ matrix defined as:
$[QFIM_s(\Theta)]_{i,j} = 4\,\mathrm{Re}[\langle \partial_{\theta_i}\psi_s(\Theta) \mid \partial_{\theta_j}\psi_s(\Theta)\rangle - \langle \partial_{\theta_i}\psi_s(\Theta) \mid \psi_s(\Theta)\rangle\langle \psi_s(\Theta) \mid \partial_{\theta_j}\psi_s(\Theta)\rangle]$
with $|\psi_s(\Theta)\rangle = U(\Theta)|e_s\rangle$, intuitively, rank of QFIM means the degree of freedom of $|\psi_s(\Theta)\rangle$.

We can calculate the QFIM rank of a specific $U(\Theta)$ corresponds to its gate layout, **we consider a gate layout is good when it has the maximum QFIM rank, i.e.,** $\binom{m}{n} - 1$**.**

Furthermore, for a set ofgates, if the DLA rank is maximum, i.e., $\binom{m}{n}\left(\binom{m}{n} - 1\right)/2$, we know we have possibility to reach the maximum QFIM rank. Otherwise it's impossible, just like local gates scenario.

2.3.3. Circuit implementation

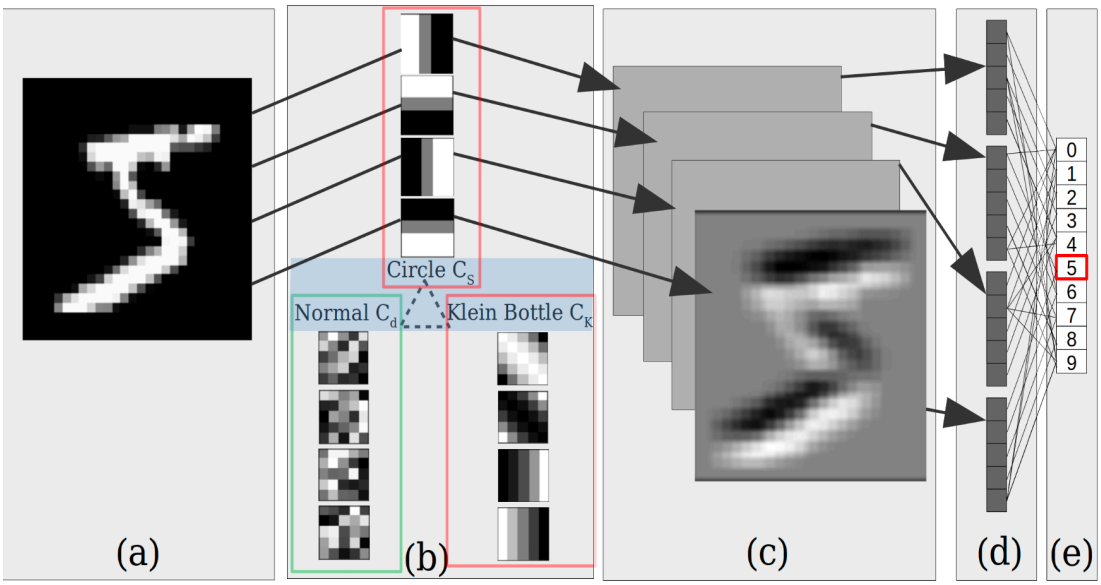
From a MNIST image, calculate the $|x\rangle$ in HW Preserving Tensor encoding, then convert this vector to Fock encoding by adding zero elements to extra basis position to get $|x'\rangle$.
For the circuit, initially since we are in the Fock encoding, we can find a good gate layout and build the circuit. Then optimize the gates parameters by gradient descent with the target $|x'\rangle$ and get the final parameters values. Because we only need to do this once for each image, it is acceptable to use Torch CUDA matrix calculations **(it's shown in "torch_calculation_example.ipynb")**. Unfortunately, at the moment , we do not know the 4× 4 matrix form of the Perceval BS gates, so we are not sure the torch matrix calculation result is absolutely correct. But we still do the calculation with Perceval components, just super slow.

3. Circuit structure

3.1. Convolutional layer

Just like the 1.2 intuition I presented, for each convolutional kernel $\mathrm{QNN}\left(\Theta_1^1\right)$ we can define the same gate layout, such as triangle or full connect layout.

3.2. Topological convolutional layer



This Topological CNN idea is from [3]. It presents two new topological convolutional kernels: Klein Filters (KF) and Klein One Layer (KOL). KF layers are convolutional layers that their weights are instantiated based on corresponding topological features and conclusions, and these weights are fixed during training. KOL layers are convolutional layers where all weights between channels whose distances are greater than some fixed threshold are forced to be zero, it means convolutional kernels are are no longer strictly square.
(See "Topological Convolutional Layers", page 9 to 12 in [3] for more details)

Since there are many results that prove this intuition is useful for generalization, we can use it in our QCNN. For KF just build extra kernels with some specific parameters and make them fixed during training. For KOL I don't have very analog solution yet, but we can convert the 2-dimensional tensor MNIST images into 3-dimensional tensors, by using stride replication (check details in [1]), then use a not good gate layout in the extra dimensional circuit to analog zero weights. We believe quantum topological CNN is a promising direction that has not yet been explored, but unfortunately we have not found the time to implement it. We plan to focus on this after all M2 exams.

3.3. Dense layer

Since we already have the good gate layout, we just use it and do training.

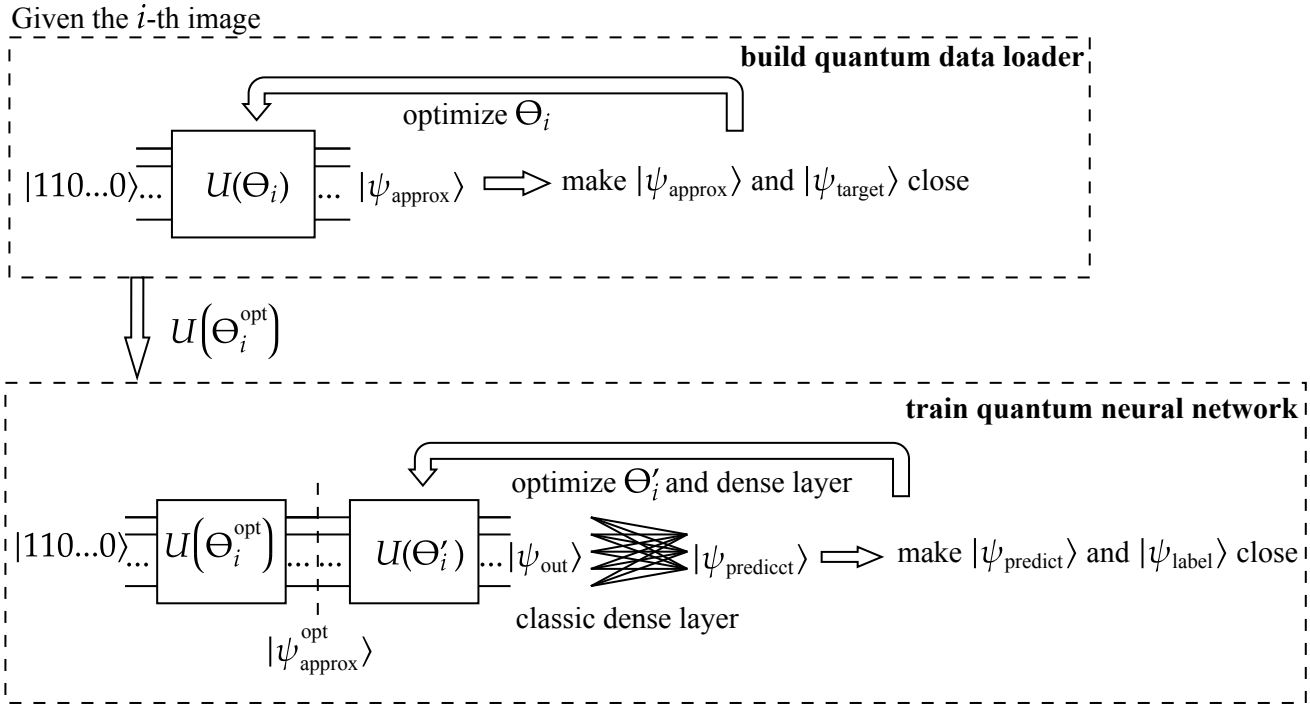
4. Measurement - from quantum to classic

We can have two measurement methods:

1. simply measure some specific modes, for 10 labels MNIST we only pick 5 modes when $n = 2$, so we obtain a $\binom{5}{2} = 10$ dimensional state as the output. That means we only need to measure a small number modes instead of all modes, work [1] used this method and it works.
2. measure all modes, then consider the distribution as input to a classic dense layout. Many quantum neural network works used this method, we do so as well in this work.

5. The picture and code implementation

The schematic diagram for our quantum neural network is as follow:



And the dataflow:

1. build quantum data loader step:

MNIST image matrix $M_{(28\times28)} \xrightarrow{\text{averge pooling}} \text{matrix}_{\left(\frac{m}{2}\times\frac{m}{2}\right)} \xrightarrow{\text{vectorization}} |\phi\rangle_{\left(\frac{m^2}{4}\right)}$

$\xrightarrow[\text{to Fock encoding}]{\text{map from HW Preserving Tensor encoding}} |\psi_{\text{target}}\rangle_{\binom{m}{2}}$

and $|110...0\rangle \xrightarrow{U(\Theta_i)} |\psi_{\text{approx}}\rangle_{\binom{m}{2}}$

2. train quantum neural network:

$|110...0\rangle \xrightarrow{U\left(\Theta_i^{\text{opt}}\right)} |\psi_{\text{approx}}^{\text{opt}}\rangle_{\binom{m}{2}} \xrightarrow{U(\Theta_i')} |\psi_{\text{out}}\rangle_{\binom{m}{2}} \xrightarrow{\text{dense layer}} |\psi_{\text{predicct}}\rangle_{(10)}$

You can check the circuit strucutre of U in the annex of part 9.

The process in **"QCNN_Perceval.ipynb"** is:

1. given the value of m, n, find a good gate layout, i.e., full QFIM rank.
2. build quantum data loader: given some MNIST images, optimize the Perceval gates with parameters $\{\Theta_i\}$ which corresponds to these MNIST images, finally we have $\left\{U\left(\Theta_i^{\text{opt}}\right)\right\}$ as our quantum data loaders.

3. <u>train quantum neural network:</u> implement the Perceval circuit $U\left(\Theta_i^{\text{opt}}\right)$ and $U(\Theta_i')$, where Θ_i is obtained by the previous step and fixed during training, Θ_i' is initialize randomly, can be optimized during training. $U\left(\Theta_i^{\text{opt}}\right)$ means the data loader and $U(\Theta_i')$ means the quantum dense neural network. They use the same U because they have same gate layout, finally we do the training.

Now our code is to slow to train, but the figure in [1] has shown the intuition is good, we can have good result (see figure below), and we just need to use the torch CUDA matrix calculation to save our time. **Attention the figure below is totally calculated by torch RBS gate matrix, we don't use any Perceval framework components here.**

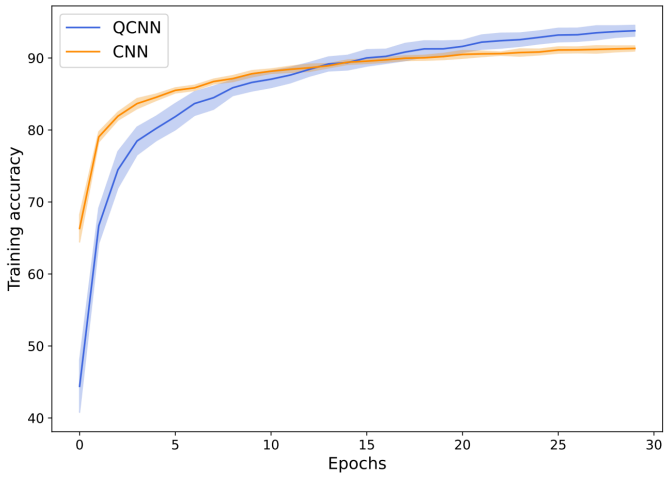


Figure 2: Average training accuracy and standard deviation comparison between classical CNN architecture and a HW preserving architecture for classification of 10 label MNIST datasets with 2000 input images. The average values and standard deviation are derived from 10 different trainings. The quantum architecture (QCNN) has 755 parameters and the classical architecture (CNN) has 990 parameters.

6. Question

6.1. Matrix form of BS gates

Like $RBS(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, bases are $|11\rangle, |10\rangle, |01\rangle, |00\rangle$,

I want to know the matrix form of BS.H in Perceval, I tried $\text{BS.H}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & \sin(\theta) & -\cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ but it turns out it's not correct.

6.2. gradient descent

Can we calculate gradient descents and do optimizing automatically in Perceval? In the code I define backward() manually, but it's slow. I hope we have a good method like the torch package.

7. Conclusion

Obviously this work is not complete, and we need to understand the Perceval framework better, as well as more free time for research to complete it. Hopefully we will have the opportunity to continue working on this related work in the future. Really sorry if this has caused you any confusion.

8. References

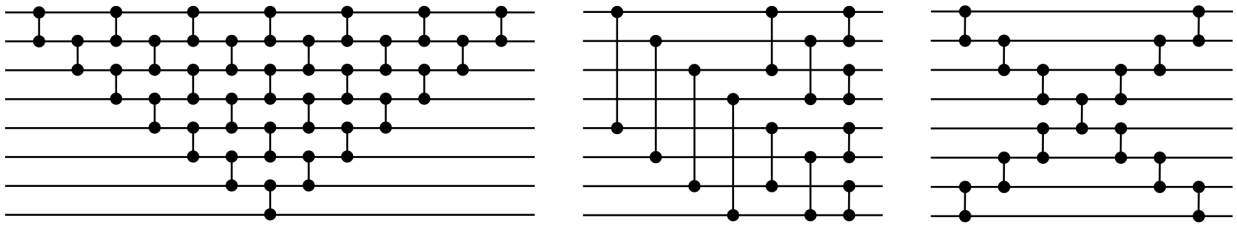
[1] Monbroussou, L., Landman, J., Wang, L., Grilo, A. B., & Kashefi, E. (2024). Subspace Preserving Quantum Convolutional Neural Network Architectures. arXiv:2409.18918. Retrieved from https://arxiv.org/abs/2409.18918

[2] Monbroussou, L., Mamon, E. Z., Landman, J., Grilo, A. B., Kukla, R., & Kashefi, E. (2024). Trainability and Expressivity of Hamming-Weight Preserving Quantum Circuits for Machine Learning. arXiv:2309.15547. Retrieved from https://arxiv.org/abs/2309.15547

[3] Love, E. R. et al. Topological Convolutional Layers for Deep Learning.
Retrieved from https://www.jmlr.org/papers/volume24/21-0073/21-0073.pdf

9. Annex

Possible U:



(See "https://github.com/ptitbroussou/HW_QCNN/tree/main" for more possibilities)

And the ideal situation I envision for matrix form of BS.H gates in **"torch_calculation_example.ipynb"**:

e.g. when $n = 2, m = 4$, $BSH(\theta) = \begin{pmatrix} \cos(\theta/2) & \sin(\theta/2) \\ \sin(\theta/2) & -\cos(\theta/2) \end{pmatrix}$, we apply a BS.H gate on mode 0 and 1:

$|1010\rangle$ $|1001\rangle$ $|0110\rangle$ $|0101\rangle$

$$\begin{array}{c} |1010\rangle \\ |1001\rangle \\ |0110\rangle \\ |0101\rangle \end{array} \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & \cos(\theta) & 0 & \sin(\theta) \\ \sin(\theta) & 0 & -\cos(\theta) & 0 \\ 0 & \sin(\theta) & 0 & -\cos(\theta) \end{pmatrix} = \widetilde{U}_{0,1}(\theta) \text{ with HW Preserving Tensor encoding}$$

$$\begin{array}{c} \\ |1100\rangle \\ |1010\rangle \\ |1001\rangle \\ |0110\rangle \\ |0101\rangle \\ |0011\rangle \end{array} \begin{array}{c} \begin{array}{cccccc} |1100\rangle & |1010\rangle & |1001\rangle & |0110\rangle & |0101\rangle & |0011\rangle \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \cos(\theta) & 0 & \sin(\theta) & 0 & 0 \\ 0 & 0 & \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & \sin(\theta) & 0 & -\cos(\theta) & 0 & 0 \\ 0 & 0 & \sin(\theta) & 0 & -\cos(\theta) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} = U_{0,1}(\theta) \text{ with Fock encoding}$$