Final Paper for Hardware for Deep Learning: Quantum Algorithms for Deep Convolutional Neural Networks

by Alejandro Rosales

May 2024

1 Introduction

Computer scientists utilize principles of quantum mechanics, mathematics, and computer science in quantum computing. By borrowing concepts from each field scientists can rigorously define both a broad and narrow theoretical model of a quantum computer and later apply it to the real world.

These theoretical models, such as the result of a quantum system manipulating subatomic particles, the theoretical circuits quantum computers implement to perform larger operations, and how to optimize the resource complexity for quantum systems, are just a few fundamental concepts in quantum computing theory.

In this paper presented by Kerenidi et al. [1], we will look at how Quantum Convolution Neural Network QCNN can be used for deep networks, potentially paving the way for groundbreaking advancements in image recognition. Quantum computing can improve the time complexity of performing larger operations, such as convolution matrix multiplication, therefore, allowing for an increased number or size of convolution kernel, as well as exploring larger or more complex input structures.

2 Quantum Preliminaries

2.1 Qubit

The states of the quantum system or qubit can be denoted as a vector like: $\alpha|0\rangle + \beta|1\rangle$ This vector has a unit length of 1, so $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha|^2$ and $|\beta|^2$ are the probabilities of the system being in the representative states. This means that the probabilities that when the qubit is measured will give a state 0 or 1 in this two-level quantum system. We will go more in-depth into probabilities here soon.

First, however, we will look at the formal definition of the inner dot product of some given qubit θ. For $|\theta\rangle$ the unit length is equivalent to its inner product. Where, for ket $|\theta\rangle$ and bra $\langle\theta| = (a^*b^*)$, a, b are both complex numbers and both have two real numbers. The inner dot product is $|\theta\rangle|\theta^*\rangle = |\theta\rangle|\theta^\dagger\rangle$, formulated as:

$$\langle\theta|\theta\rangle = (a^*b^*)$$

which can can be written in terms of a, b as:

$$\begin{pmatrix} a \\ b \end{pmatrix} = |a|^2 + |b|^2 = 1$$

For some quantum state for the qubit ψ can be defined as:

$$|\psi\rangle = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} = \langle v|\psi\rangle|v\rangle + \langle h|\psi\rangle|h\rangle$$

Where squaring our projections, $\langle v|\psi\rangle$ and $\langle h|\psi\rangle$, onto axis an axis gives us our respective probabilities for $|v\rangle$ and $|h\rangle$ respectfully.

An example of a qubit is the spin of an electron. The two levels of this qubit are spin up or spin down. What differs from a classical system is that quantum mechanics allows for the qubit to be in a coherent superposition of both states simultaneously. Measuring a qubit in a basis gives a projective measurement of a qubit of state ϕ in its computational basis can be expressed as a linear combination of state vectors, such as:

$$|\psi\rangle = a|0\rangle + b|1\rangle$$

When the qubit is measured in a basis, collapses the qubit to either the quantum state $|0\rangle$ or $|1\rangle$ given by the respective norm-square of the probability amplitudes a and b, or a^2 and b^2.

2.2 Superposition

In classical computing, states 0 and 1 would be the only states that exist for the bit. However, in quantum mechanics, a qubit can be both in the state of $|0\rangle$ and $|1\rangle$. This is what gives quantum computers more processing power, as a single qubit can be in more states and therefore represent more information than a single classical bit. This means that the qubit can have an 80 percent of being in state $|0\rangle$ and 20 percent of being in state $|1\rangle$, or 75 percent of being in state $|0\rangle$ and 25 percent of being in state $|1\rangle$, unlike a classical bit where there is either a 100 percent of the classical bit being in state 0 and 0 percent of being a 1 or a 0 percent of the classical bit being a 0 and 100 percent of being a 1. To allow for the qubit to be in superposition, we need to leverage Hilbert Space. Again, Hilbert space is represented using complex vector space. We use complex vector space because it is the easiest way for the math to work. Let's represent this qubit in superposition as a vector using Dirac notation. For the qubit have, for example, a 50 percent of being in state $|0\rangle$ and 50 percent of being in state $|1\rangle$, the vector should look like:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

The coefficient for 0 and 1 are both $\frac{1}{\sqrt{2}}$ and therefore equal parts 0 and 1.

3 Current Tools in Quantum Computing

3.1 Quantum Parallelism

Suppose we want to evaluate a function $f(x)$, where the function f expresses some computation or algorithm. A use case for quantum parallelism is to evaluate $f(x)$ with many different values for the output of the computation or algorithm on the input x simultaneously. In essence, we can evaluate many different values of x on f in parallel by exploiting quantum effects. This quantum effect exploit feature is fundamental in many quantum algorithms and the quantum convolutional algorithm we will look at later. This is why we will look at how quantum parallelism works.

Consider the one-bit domain and range function:

$$f(x) : \{0,1\} \rightarrow \{0,1\}$$

To compute this function f on a quantum computer, we will use a two-qubit quantum computer with the starting state $|x, y\rangle$. The transformation of the domain or 'data' register to the range or 'target' register of this initial two-qubit state is described by the following unitary function:

$$U_f : |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$$

where the $\oplus$ represents addition modulo 2 and $x = q_0, y = q_1$. When $\oplus$ acts on y, and its value is 0 then the value of the second qubit in the 'target' register is the value $f(x)$, given whatever function f represents. The function's effect on x is arbitrary for now. The final collapsed state $|\psi\rangle$ is an element of the set of final states or 'target' register $|x, y \oplus f(x)\rangle$, which again is given by the unitary transformation U_f on the start state $|x, y\rangle$. Given the input $q_0 = x = |0\rangle$, we will apply the Hadmard gate H to x, such that now:

$$x = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, y = |0\rangle$$

where the resulting state is:

$$\frac{|0f(0)\rangle + |1f(1)\rangle}{\sqrt{2}}$$

which the resulting new state is not a part of the starting computational basis $\{0, 1\}$. Next, the unitary function or black box computation/algorithm U_f can be applied to the current 'data' register. The resulting mapping of the unitary function U_f is:

$$U_f\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}, |0\rangle\right) = \frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}$$

which is equivalent to:

$$\frac{1}{\sqrt{2}}(|0, f(0)\rangle + |1, f(1)\rangle) = .5|0, f(0)\rangle + .5|1, f(1)\rangle$$

This means that the final resulting state for a two-qubit quantum computer has a 50 percent chance of being $|0, f(0)\rangle$ and 50 percent of being $|1, f(1)\rangle$. Given in the same form as the range for U_f given above:

$$|x, y'\rangle,\ y' = y \oplus f(x)$$

All of this means that the information given by the mapping for $f(0)$ and $f(1)$ was simultaneously evaluated by applying superposition and the unitary function on the starting 'data' register. Therefore, $f(x)$ has been computed for two values of x in parallel. The resulting 'target' register gives the resulting set of all possible states computed in parallel is given by quantum exploitation and aptly named 'quantum parallelism'. Thus, a single $f(x)$ circuit can be used to evaluate the result for n values of x simultaneously.

4 Quantum Convolutional Neural Network

The operation of a classical convolutional neural network CNN that we will only look at in this paper is optimizing the computational of the convolutional layer. To do so, will look at each procedural step and see how quantum computing techniques can be applied to improve the operation. the authors [1] describe the procedure for the quantum convolutional layer in the following four sequential steps: inner product estimation, non linearity, quantum sampling, and then Quantum Random Access Memory QRAM update and pooling.

4.1 Quantum Convolutional Layer

First, let us define our parameters. The input for the quantum convolution layer is a 3D tensor input:

$$X^{\ell} \in R^{H^{\ell} \times W^{\ell} \times D^{\ell}}$$

and the weights layer, filter layer, or kernel layer is a 4D tensor:

$$K^{\ell} \in R^{H \times W \times D^{\ell} \times D^{\ell+1}}$$

where the input and kernel layer are both stored in QRAM. Given precision parameters ϵ, $\Delta > 0$, there exists a quantum algorithm that computes a quantum state that is Δ close to $|f(\bar{X}^{\ell+1})\rangle$ where $X^{\ell+1} = X^{\ell} * K^{\ell}$, and $f : R \mapsto [0, C]$ is a non-linear activation function. Note, that this function could be a sigmoidal, hyperbolic tangent, ReLU, softmax, or any other activation function and is rather dependent mainly on the task the activation function needs to complete.

To continue, a classical approximation for the computer quantum state exists where:

$$\left\| f\left(\bar{X}^{\ell+1}\right) - f\left(X^{\ell+1}\right) \right\|_{\infty} \leq \epsilon$$

The time complexity of the quantum algorithm that computes the output quantum state $|f(\bar{X}^{\ell+1})\rangle$ is $\widetilde{O}(M/\epsilon)$. In this equation, M denotes the maximum norm of the product state between one of the kernels and one of the tensor input regions in X^{ℓ}, where the size is HWD^{ℓ}. It is important to note, that $\widetilde{O}$ is made to hide factors poly-logarithmic in Δ with respect to the size of X^{ℓ} and K^{ℓ}. Given that a convolution product is equivalent to a matrix-matrix multiplication and the convolutional product between X^{ℓ} and K^{ℓ} is:

$$X^{\ell+1}_{i^{\ell+1},j^{\ell+1},d^{\ell+1}} = \sum_{i=0}^{H}\sum_{j=0}^{W}\sum_{d=0}^{D^{\ell}} K^{\ell}_{i,j,d,d^{\ell+1}} X^{\ell}_{i^{\ell+1}+i,j^{\ell+1}+j,d}$$

it is possible to reformulate this convolution product equation as a matrix product.

To express this reformulation mathematically, we must do the following. First, the original 3D tensor input input for the quantum convolution layer $X^{\ell} \in R^{H^{\ell}\times W^{\ell}\times D^{\ell}}$ and the 4D tensor kernel layer $K^{\ell} \in R^{H\times W\times D^{\ell}\times D^{\ell+1}}$ needs to be reshaped to matrices. First, X^{ℓ} needs to be expanded into a matrix $A^{\ell} \in R^{\left(H^{\ell+1}W^{\ell+1}\right)\times\left(HWD^{\ell}\right)}$, such that each row of A^{ℓ} is a vectorized version of a subregion of X^{ℓ}. Next, the original kernel tensor K^{ℓ} is reshaped into a matrix $F^{\ell} \in R^{\left(HWD^{\ell}\right)\times D^{\ell+1}}$, such that each column of F^{ℓ} is a vectrozied version of one of the $D^{\ell+1}$ kernels.

This matrix expression of the tensor input and kernel is needed for the convolution product $X^{\ell} * K^{\ell} = X^{\ell+1}$ to be written as a matrix multiplication $X^{\ell} * K^{\ell} = X^{\ell+1}$, such that each column of the output matrix $Y^{\ell+1} \in R^{\left(H^{\ell+1}W^{\ell+1}\right)\times D^{\ell+1}}$ is a first vectorized form of one of the $D^{\ell+1}$ channels of $X^{\ell+1}$.

To continue, quantum states proportional to the rows of input A^{ℓ} and F^{ℓ} are used, denoted $|A^{\ell}_p\rangle$ and $|F^{\ell}_q\rangle$ respectively. These quantum states are defined as:

$$|A^{\ell}_p\rangle = \frac{1}{\left\|A^{\ell}_p\right\|}\sum_{r=0}^{HWD^{\ell}-1} A^{\ell}_{pr}|r\rangle, |F^{\ell}_q\rangle = \frac{1}{\left\|F^{\ell}_q\right\|}\sum_{s=0}^{D^{\ell+1}-1} F^{\ell}_{sq}|s\rangle$$

and will continue to be used throughout this section.

4.1.1 Inner Product Estimation

First, we load input row vector A^{ℓ}_p and kernel vector F^{ℓ}_q into quantum states by querying QRAM in the following manner:

$$\begin{cases} |p\rangle|0\rangle \mapsto |p\rangle\left|A^{\ell}_p\right\rangle \\ |q\rangle|0\rangle \mapsto |q\rangle\left|F^{\ell}_q\right\rangle \end{cases}$$

This is done so that the following mapping can be performed with the two vectors:

$$\frac{1}{K}\sum_{p,q}|p\rangle|q\rangle \mapsto \frac{1}{K}\sum_{p,q}|p\rangle|q\rangle|\bar{P}_{pq}\rangle\,|g_{pq}\rangle$$

Here $\bar{P}_{pq}$ is the inner product estimation of A^{ℓ}_p and F^{ℓ}_q, $K = \sqrt{H^{\ell+1}W^{\ell+1}D^{\ell+1}}$ is the normalization factor, and $|g_{pq}\rangle$ is some garbage state. The "true" value of the inner product is calculated as:

$$P_{pq} = \frac{1 + \left\langle A^{\ell}_p \mid F^{\ell}_q\right\rangle}{2}$$

such that the inner product estimation differs by a chosen constant ϵ.

4.1.2 Non Linearity

Our obtained approximated convolution output $\bar{P}_{pq}$ or $\bar{Y}^{\ell+1}$ differs from the "true" convolution output $Y^{\ell+1}_{p,q} = \left(A^{\ell}_p, F^{\ell}_q\right)$ by ϵ. Now, we apply a non-linear function f as a boolean circuit giving the quantum state:

$$\frac{1}{K}\sum_{p,q}|p\rangle|q\rangle|f(\bar{Y}^{\ell+1}_{p,q})\rangle\,|g_{pq}\rangle$$

4.1.3 Quantum Sampling

To procure the state below, states are conditionally rotated and the probabilistic amplitudes are amplified, such that we arrive at the state:

$$\frac{1}{K}\sum_{p,q}\alpha'_{pq}|p\rangle|q\rangle|f(\bar{Y}^{\ell+1}_{p,q})\rangle\,|g_{pq}\rangle$$

Quantum tomography is performed with precision η so that all values and positions $(p, q, f(\bar{Y}^{\ell+1}_{pq}))$ are obtained with a high probability. Values above η are known exactly, while values that are less than or equal to η are set to 0.

4.1.4 QRAM Update and Pooling

Next, QRAM needs to be updated with the value for the next layer, which is $A^{\ell+1}$, while sampling. Pooling needs to be implemented in this step as well, either through a specific update or by using a QRAM data structure.

At the end of layer ℓ, the pooling operation of size P is performed on the convolution layer output $f(X^{\ell+1})$, yielding the tensor after pooling $\tilde{X}^{\ell+1}$. Here, for some point at position $(i^{\ell+1}, j^{\ell+1}, d^{\ell+1})$ in $f(X^{\ell+1})$, the pooling region it corresponds to is at postion $(\tilde{i}^{\ell+1}, \tilde{j}^{\ell+1}, \tilde{d}^{\ell+1})$ in $\tilde{X}^{\ell+1}$, such that:

$$\begin{cases} \tilde{d}^{\ell+1} = d^{\ell+1} \\ \tilde{j}^{\ell+1} = \left\lfloor \frac{j^{\ell+1}}{P} \right\rfloor \\ \tilde{i}^{\ell+1} = \left\lfloor \frac{i^{\ell+1}}{P} \right\rfloor \end{cases}$$

5 Conclusion

The QCNN developed by the authors [1] is fully realized, indicating that almost all classical architectures can be adapted into a quantum framework. This adaptation includes integrating any non-negative and upper-bounded nonlinearity, employing pooling mechanisms, adjusting the number of layers, and accommodating diverse kernel sizes.

The reduced execution time showcases enhanced performance compared to the classical algorithm, thanks to swift linear algebra computations during convolution and the selective sampling of vital values from the resulting quantum state. This improvement is particularly significant in scenarios with a high number of input tensor channels, enabling the utilization of deep CNN architectures, akin to the recent strides made in the DeepMind AlphaGo algorithm.

Despite the innovative methods employed by the authors [1] to simplify complexity, introducing nonlinearity and reusing layer outputs for subsequent layers necessitates register encoding and state tomography. Consequently, this hampers the potential for achieving an exponential acceleration in the context of input parameter count.

6 References

[1] Kerenidis, Iordanis, Jonas Landman, and Anupam Prakash. "Quantum Algorithms for Deep Convolutional Neural Networks." CNRS, IRIF, Universite Paris Diderot, Paris, France, November 5, 2019.