Applications of Quantum Techniques in Time Series Analysis

Nathaniel Parke, Nishad Singh, Matt Anderson and Pranay Patil

Abstract. This project explores some applications of quantum techniques in time series analysis. For certain tasks, like computing Sparse Principal Components or the Cross Correlation between two signals, quantum computing offers runtime improvements from classical methods. For other tasks, like training a Neural Net, it's not clear that quantum approaches are improvements over their classical counterparts.

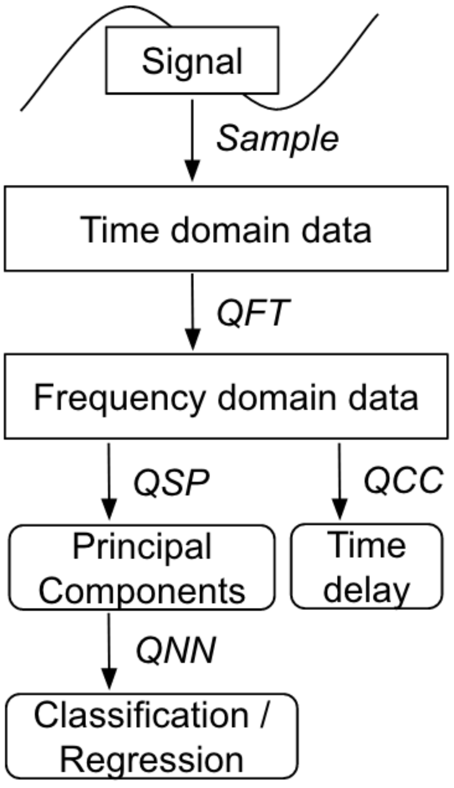
1 Introduction

1.1 Relevancy of quantum techniques to time series analysis

A time series is a sequence of numerical data points in successive order. Time series data appears in countless domains, from financial markets to traffic engineering. Analyzing time-series data quickly can be incredibly valuable.

Our objective is to explore some common techniques used in time series analysis and identify areas in which quantum algorithms can provide meaningful speedups.

1.2 Project overview



In this paper, we explore Quantum Cross Correlation (QCC), Quantum Sparse PCA (QSP), and Quantum Neural Networks (QNN). All of these can be used in different ways in time series analysis. Cross correlation can help us identify the time delay between two similar signals, retrieving principal components allows for transforming data to more computationally manageable lower-dimensional representations while removing much of the noise, and neural networks can be used for general classification and regression.

In the figure, we demonstrate a few possible usage patterns of these techniques. First, a real-world analog signal is sampled. Time series data, however, is often periodic. Therefore, frequency-domain representations can be more concise and make accessible more meaningful features than time-domain representations of this data. In order to move our sampled signal data from the time domain into the frequency domain, we use the Quantum Fourier Transform (QFT).

Once the data is in the frequency domain, We can compute signals' cross correlation to find their time delay using QCC. We can also use QSP to find their principal components, transform the data into a lower-dimensional representation whilst removing most of the noise, and then use QNN to train models on the data to get classifications or regressions.

Before performing any of the proposed algorithms on our time series data, we will need to move between classical and quantum representations of the data. We have described our choice for the quantum representation in the following section.

2 Quantum Representations

2.1 Motivation

Quantum algorithms operate on data stored in the coefficients of superpositions over composite qubit states. In order for us to use quantum algorithms on classical data, we need an efficient way to convert our classical data into corresponding quantum superpositions.

2.2 Representation

Assume that we have a classical vector of data $\vec{v} = [\alpha_0, ..., \alpha_{N-1}]^T$ with N entries. These could be normalized data points that we collect by sampling a signal at evenly spaced intervals. Prior to performing any quantum algorithms, we will translate the classical data into a quantum superposition over $n = \lceil \log_2 N \rceil$ qubits

$$|v\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle . \quad (1)$$

The states comprising the superposition above correspond to the indices of the data vector while the coefficient of each state i in the superposition is the value of the data vector at index i.

2.3 Building the Superposition

The difficult part of this process is determining how to physically create the superposition that we have described above. Since the states in the superposition correspond to a sequential ordering of indices, we want to create a system that looks like the following:

$$|0\rangle + |1\rangle + |2\rangle + \cdots + |N-1\rangle \quad (2)$$

where the numbers can be represented in binary. In other words, we want a superposition over all possible strings of length $n = \lceil \log_2 N \rceil$. To arrive at such a superposition, we can apply the Hadamard transform $H^{\otimes n}$ to $\frac{1}{\sqrt{2}}(|0^n\rangle + |1^n\rangle)$, a state we have previously seen how to create.

While we now have the correct superposition, the amplitudes of the individual states are not all correct. We need to modify these amplitudes to match the normalized values of the original data vector. Unfortunately, although it is certainly physically possible to create the desired superposition with the correct amplitudes, it was difficult to find any literature detailing a process to do so. Such a process would likely use the intuition behind Grover's algorithm in using rotations within a state space to change the amplitudes of the states in a determined fashion.

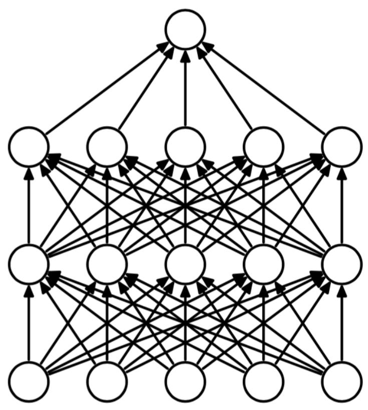
2.4 Time Complexity

An important question that remains unanswered due to the lack of an algorithm that will encode the classical data as detailed above is how fast preparing the data will actually take.

The first part of preparing the actual superposition, albeit with incorrect amplitudes, will be fairly quick. It is much harder to predict the time it will take to modify the amplitudes to reach the correct values. If the time complexity of preparing the data is much greater than the speedups obtained by implementing the algorithms discussed throughout this paper, then we would be better off performing time series analysis using classical methods. This is an area that will require further research.

3 Quantum Neural Networks

3.1 Classical Neural Networks

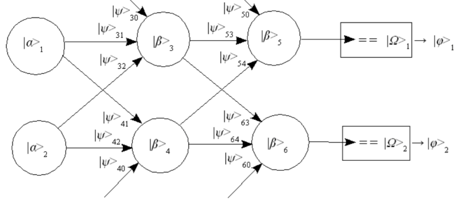


It is desirable to find a quantum approach to training artificial neural networks. The use of neural networks in regression and classification systems has greatly increased in recent years. The general approach to using neural networks involves creating a set of neurons, which are fed a linear combination of inputs. These inputs are then summed and passed through an activation function that introduces non-linearity to the output. The neurons are arranged into layers which receive inputs from the previous layer's outputs.

When seeking to solve a regression or classification task, the implementer chooses a loss function by which the current state of the neural network can be evaluated. The neural network then goes through the training stage by which gradients of the loss function with respect to the weights of the network are calculated. Typical training methods calculate the gradient for a single point and iteratively step in the direction of greatest descent in an attempt to find the global minimum of the loss function. This process is called stochastic gradient descent.

3.2 Quantum Training

The most desirable task to speed up is neural net training. Ricks and Ventura explain a method which takes advantage of Grover's algorithm to stochastically search for the optimal weights [1]. Their idea is to construct a neural network and initially randomize the weight vectors across all nodes. After initialization, randomly choose a node within the network, and run Grover's search to find the optimal set of weights for that particular node, holding constant the weights in the rest of the network.



This Grover's search requires an oracle that can mark which weight superpositions are optimal for the current state of the network. This oracle can be constructed by setting a threshold value on the loss function. Tuning the thresholding value is a separate task in and of itself. Ricks and Ventura describe a method for this in a classification task. They choose a low threshold percentage, p, that represents the percentage of misclassified samples. As the weights are updated they iteratively increase p, to step towards a lower loss value.

3.3 Optimality

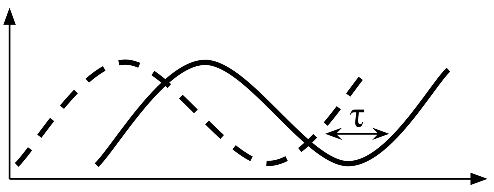
This proposal for a quantum neural network has significant problems. The first of which is the fact that this is searching for the optimal weight vector in an exponentially large space with respect to three parameters, the number of qubits required to represent a single weight, the number of weights going to each node, and the number of nodes in the neural network.

This exponential search problem is not an issue with classical neural networks because we are solving an optimization problem, rather than searching for an exact solution. This loosens the constraints on the solution, allowing for imprecision. In most cases however through the use of convex loss functions, we can place some bounds on the convergence of an optimal set of weights. That is not the case with this proposed technique. Ricks and Ventura even explain that due to the randomness of this algorithm there are likely to be little advantages to using Quantum neural networks designed in this fashion. Even though we get a quadratic speedup over the exact solution search, with Grover's, the exponential nature of the problem is prohibitive.

4 Quantum Cross Correlation

4.1 Motivation

In time series analysis, it is often valuable to find the time-delay between two signals (say, to predict the value of a trailing stock).



One can do this by finding the time shift under which the signals' cross-correlation is maximized. Cross-correlation is a measure of the similarity of two series as a function of the displacement of one relative to the other. The cross-correlation is generally high when there is no displacement and low otherwise, making it easy to determine the time-delay.

4.2 Process

Suppose we sample the two signals f and g at T evenly spaced intervals, giving us vectors $\vec{f}$ and $\vec{g}$. Computing their cross correlation vector in the time domain would require solving for all $t \in \{0, 1, ..., T-1\}$

$$(f \vec{*} g)[t] = \sum_{m=0}^{T-1} (f[m])^* g[m+t] \tag{3}$$

However, if we have the frequency-domain representation of $\vec{f}$ and $\vec{g}$, $\mathcal{F}\{\vec{f}\}$ and $\mathcal{F}\{\vec{g}\}$, by the Cross Correlation Theorem, we can compute our cross correlation with

$$f \vec{*} g = \mathcal{F}^{-1}\{(\mathcal{F}\{f\})^* \mathcal{F}\{g\}\} \tag{4}$$

To find the time delay, we compute

$$\tau = \arg \max_{t \in \{0,1,...,T-1\}} (f \vec{*} g)[t] \tag{5}$$

4.3 QCC Algorithm

1. Begin with vectors of samples of f and g at T evenly spaced intervals, $\vec{f} = [\alpha_{f,0}, ..., \alpha_{f,T}]$ and $\vec{g} = [\alpha_{g,0}, ..., \alpha_{g,T}]$.
2. Move $\vec{f_{norm}}, \vec{g_{norm}}$ into quantum superpositions: $|f\rangle = \frac{1}{|\vec{f}|} \sum_{j=0}^{N-1} \alpha_{f,j} |j\rangle$ and $|g\rangle = \frac{1}{|\vec{g}|} \sum_{j=0}^{N-1} \alpha_{g,i} |j\rangle$. This will require $n = \lceil log_2 N \rceil$ qubits for both.
3. Run the Quantum Fourier Transform on $|f\rangle$ and $|g\rangle$ to obtain $\{|\mathcal{F}f\rangle\}$ and $\{|\mathcal{F}g\rangle\}$
4. Compute their elementwise multiplication
5. Run the Inverse Quantum Fourier Transform on the result to obtain $|f * g\rangle$
6. Run the quantum argmax algorithm (described in the next section) with m instances of $|f * g\rangle$. m is variable; higher values give higher success probabilities. See the Argmax Algorithm Analysis section for details.

4.4 Quantum Argmax Algorithm

We have identical superpositions with $n = \lceil log_2 N \rceil$ qubits: $\sum_{i=1}^{N-1} B_j |j\rangle$. We want to find the $|j\rangle$ such that $|B_j|^2 \geq max_i |B_i|^2$.

1. We begin with m identical quantum superpositions $\sum_{i=1}^{N-1} B_j |j\rangle$.
2. Let *count* be a dictionary with keys $0, ..., N-1$ and values initialized to 0.
3. Let *most* be an integer that stores j if the most frequently measured term is $|j\rangle$.
4. We measure the first superposition, giving $|k\rangle$ for some $0 \leq k \leq N-1$. We increase $count[k]$ by 1 and set $most$ to k.
5. For each of the remaining $m-1$ superpositions:
   (a) Measure the superposition, giving $|k\rangle$ for some $0 \leq k \leq N-1$.
   (b) Increase $count[k]$ by 1
   (c) If $count[k] > count[most]$, set $most = k$
6. Return *most*

4.5 Argmax Algorithm Analysis

The algorithm initializes some values (taking constant time) and then runs m iterations of measurement and value updates (also constant time). It therefore takes $O(m)$ time.

It is very relevant, however, to look into the probability that this algorithm actually succeeds. Recall that the input to the argmax algorithm is m identical superpositions $\sum_{i=1}^{N-1} B_j |j\rangle$. Let j be the true argmax over the superposition:$|B_j|^2 \geq max_i |B_i|^2$.

Pr(algorithm returns j) $=$ Pr(j was seen more frequently than any other option) $\leq Pr(count[j] > \frac{m}{2})$.

Let $p = |B_j|^2 \geq max_i |B_i|^2$: the probability that in any one measurement, we retrieve the term with a maximal probability amplitude in the original superposition.

$count[j]$ is sum of the result of m trials, where the result = 1 iff the measurement was j. In other words, $count[j]$ is a random variable drawn from $Binomial(m, p)$. We can therefore re-express $Pr(count[j] > \frac{m}{2})$ as

$$Pr(Binomial(m, p) > \frac{m}{2}) =$$

$$1 - Pr(Binomial(m, p) \leq \frac{m}{2}) =$$

$$1 - \sum_{i=1}^{\frac{m}{2}} \binom{m}{i} p^i (1-p)^{m-i} \leq$$

$$1 - \sum_{i=1}^{\frac{m}{2}} \binom{m}{\frac{m}{2}} p^{m/2} (1-p)^{\frac{m}{2}} =$$

$$1 - \frac{m}{2} \binom{m}{\frac{m}{2}} ((1-p)p)^{m/2}$$

Following Stirling's approximation, we know $\binom{n}{k} \leq (\frac{en}{k})^k$, and therefore $\binom{m}{\frac{m}{2}} \leq (2e)^{\frac{m}{2}}$. Therefore,

$$Pr(count[j] > \frac{m}{2}) \leq 1 - \frac{m}{2}(2e(1-p)p)^{\frac{m}{2}}$$

If we wanted this algorithm to be correct with probability $1 - \epsilon$, we would need to set m such that

$$1 - \epsilon \leq 1 - \frac{m}{2}(2e(1-p)p)^{m/2} \equiv$$

$$\epsilon \geq \frac{m}{2}(2e(1-p)p)^{m/2}$$

4.6 QCC Algorithm Time complexity

Though we haven't formally discussed the Inverse QFT in class, we know that because there is an efficient quantum circuit implementing the quantum Fourier transform, the circuit can be run in reverse to perform the inverse quantum Fourier transform. With an input of n qubits (a superposition over them can encode $T = 2^n$ bits of classical information), the version of the QFT we saw in class required $O(n^2)$ gates. Our measure of complexity for the QFT and Inverse QFT, therefore, is $O(log^2 T)$. The FFT, for comparison, takes $O(T logT)$ time.

The quantum argmax algorithm (with $m = 1$) over T elements takes $O(\sqrt{T})$ time (the logic is symmetric to that presented in the solutions to the quantum argmin question on Problem Set 7). Finding the argmax of set of T elements in a classical setting would take $O(T)$ time.

Therefore, the quantum algorithm takes $O(log^2 T + \sqrt{T}) = O(\sqrt{T})$ time, and the classical algorithm takes $O(T logT)$ time.

If we parameterize m, we get $O(mT logT)$ time. We can then expect to see the correct answer for the time delay with probability $\geq 1 - \epsilon$ when we set m such that $\epsilon \geq \frac{m}{2}(2e(1-p)p)^{m/2}$.

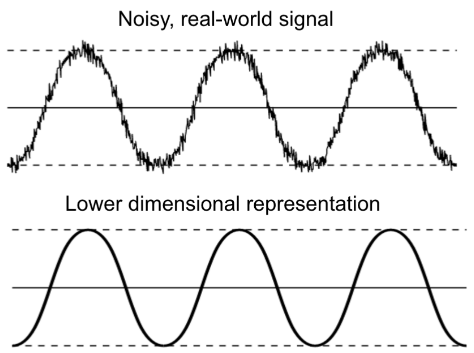
5 Quantum Sparse PCA

5.1 Motivation

In most categories of data analysis, it is often the case that the input data requires a large amount of space to be fully represented and contains a lot of noise that ultimately makes it difficult to efficiently extract meaningful patterns from that data. This holds true for time series data as well. To moderate the impact of these issues, we turn to Principle Component Analysis (PCA) [2]. PCA uses linear combinations of the input variables to find principle components that correspond to directions of maximal variance within the data. We can then use the principle components that represent the directions of most variance to find an estimate of the original data that is both low rank and less noisy.

However, the results from ordinary PCA can be quite costly in many applications of time series analysis. For example, consider financial applications. The principle components would be a weighted combination of all the original assets. Unfortunately, a large number of assets implies more transactions which, in turn, implies higher fees.

Thus, we turn to sparse PCA, which differs from ordinary PCA in that the principle components are formed by linear combinations of only a few of the input variables (or in the case of the example above, assets). This makes the outputs of the algorithm more useful in resource sensitive applications.



5.2 Semidefinite programming

Research in 2007 indicated that sparse PCA can be efficiently solved when framed as a semidefinite programming (SDP) problem [2]. A SDP problem is an optimization problem with a linear objective function that maximizes or minimizes over the intersection of an affine space and the cone of positive semidefinite matrices. A basic formulation of SDP is as follows:

$$\begin{aligned} \text{min/max} \quad & C \cdot X \\ \text{s.t.} \quad & A_i \cdot X, \quad i = 0, 1, \dots, m \\ & X \geq 0 \end{aligned}$$

where C and all of the A_i are given, all the variables represent $n \times n$ symmetric matrices, and $\cdot$ can be any function that results in a linear output [3]. The most widely used such function is the inner product

$$\langle A, B \rangle = \operatorname{tr}(A^{\mathrm{T}} B) = \sum_{i=1, j=1}^{n} A_{ij} B_{ij}. \tag{6}$$

The inputs are C and all A_i and outputs are X and the optimal objective value.

5.3 Quantum semidefinite programming

Quantum semidefinite programming reduces SDP to the following feasibility problem: Given α, an estimate for the optimal objective value, does there exist a valid X for which the objective function evaluates to less than α. Runtime analysis will be done in the following section, but it is important to note that the algorithm runs in time faster than it would take to write down all elements of X ($O(nm)$). Thus, the quantum algorithm outputs the optimal α and samples from the optimal X. Note: A more detailed, and much more complicated outline of the algorithm is included in literature [4].

1. initialize α to some guess
2. construct a Gibbs state as a linear combination of the input matrices
3. amplify the amplitudes of the Gibbs state
4. "sparsify" the Gibbs state through random sampling
5. calculate the objective value given the Hamiltonian that represents the Gibbs state
6. update α and continue the binary search as appropriate with the estimate of the objective value

A Gibbs state is a thermal quantum state at equilibrium. Arora, et al., show that if we prepare a Gibbs state of Hamiltonians given by linear combinations of the input matrices, we can satisfy the SDP [5] [4]. Additionally, a huge runtime increase of the quantum algorithm comes from the fact that we can amplify the amplitudes of the Gibbs state as outlined in [6]. The Gibbs state can be constructed in $O(\log(n))$ time [7].

5.4 Time complexity

The best classical algorithms for SDP run in $O(m(m^2+n^\omega+mns)\log(1/\delta))$, where ω represents the exponent of matrix multiplication, s represents the sparsity of the matrices, and δ represents the accuracy of the solution [4]. A quantum approach to solving SDP problems sees a speed-up to $O(n^{\frac{1}{2}} m^{\frac{1}{2}} s^2 \mathrm{poly}(\log(n), \log(m), R, r, 1/\delta)$, with n and s the dimension and row sparsity of the input matrices, respectively, m the number of constraints, δ the accuracy of the solution, and R, r a upper bounds on the size of the optimal primal and dual solutions [4]. More simply, the quantum speedup improved the classical runtime from something that grows like $O(m^3 + m^2 ns + mn^\omega)$ to something that grows like $O(n^{\frac{1}{2}} m^{\frac{1}{2}} s^2)$, a huge improvement giving quadratic speedup in m, the number of constraints, and in n, the dimension of the input matrices.

References

[1] B. Ricks, D. Ventura, *Training a Quantum Neural Network*, in *Advances in Neural Information Processing Systems* (2003), p. None

[2] A. d'Aspremont, L. El Ghaoui, M.I. Jordan, G.R. Lanckriet, SIAM review **49**, 434 (2007)

[3] R.M. Freund, *Introduction to Semidefinite Programming* (MIT, 2009), pp. 1–11

[4] F.G. Brandao, K. Svore, arXiv preprint arXiv:1609.05537 (2016)

[5] S. Arora, S. Kale, *A combinatorial, primal-dual approach to semidefinite programs*, in *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing* (ACM, 2007), pp. 227–236

[6] G. Brassard, P. Hoyer, M. Mosca, A. Tapp, Contemporary Mathematics **305**, 53 (2002)

[7] M.H. Yung, A. Aspuru-Guzik, Proceedings of the National Academy of Sciences **109**, 754 (2012)