TOWARDS QUANTUM NEURAL NETWORKS: A SURVEY

KALINDA VATHUPOLA

Contents

1. Introduction

Within the context of attempts at bringing neural networks into a quantum framework, this survey intends to cover their motivation, recent work, and open questions. It will focus on the problem of classification. Thus, we will cover the quantization of perceptrons, and its implications for quantizing neural networks. We will also discuss current attempts at establishing a quantum framework for classical neural networks. Those working in quantum computing will benefit from its discussion of quantizing classical algorithms. Similarly, we hope this

Date: May 5, 2020.

survey will motivate those intending to study machine learning to keep abreast of developments in quantum computing.

No prior knowledge of deep learning is assumed, but basic familiarity with quantum computing (to the extent of CMSC457) is expected.

2. Deep Learning Background

2.1. **Overview.** Machine learning refers to the general research area of applying learning algorithms to find connections between data [13]. For the purposes of this survey, we will focus on **supervised learning** whereby each data point is an input with labeled output. We will further focus on the subfield of deep learning, which is characterized by the use of biologically-inspired artificial neural networks. In both a historical and conceptual sense, the perceptron is an essential building block of neural networks.

2.2. **Perceptron.** The perceptron is designed to solve the following problem.

Linear Classification Problem. Suppose we are given a collection of points

$$X = \{x_i \in \mathbb{R}^d : i \in [n]\}$$

and corresponding classifications

$$Y = \{y_i \in \{0, 1\} : y_i \text{ is the label for } x_i\}.$$

Suppose the data $\mathcal{D} = (X, Y)$ is linearly separable. Find a hyperplane separating points with label 0 from those with label 1.

Algorithm 1 from [7] details the routine whereby the model is trained. In this case, the model is a hyperplane with learnable parameters being the weight coefficients determining the linear function. Also from [7], algorithm 2 tests this learned model on a test point.

Although algorithm 2 is seemingly trivial, these two routines follow the general learning paradigm for neural networks: train the network on data to learn the parameters, then test the network on unseen test data.

Within the context of deep learning, the perceptron is depicted as in figure 1. For now, ignore the "neuron" label, which will be discussed in the subsection on neural networks.

2.3. **Neural Networks.** Depicted in figure 2, the neuron generalizes the perceptron. Fundamentally, a neuron consists of a linear component (the weighted input) that is passed through a function f [7]. In the case of the perceptron, f is the hard limit. In general, however, the function f is some sigmoidal non-linearity.

Algorithm 1: Perceptron Train

Input: Data $\mathcal{D}$ of d-dimensional points to classify, desired iterations K
Output: Weight vector $\mathbf{w} \in \mathbb{R}^{d+1}$ determining the hyperplane
$\mathbf{w} \leftarrow \mathbf{0}^d$;
$w_{d+1} \leftarrow 0$;
for *iter = 1, 2, . . . ,* K **do**
  forall $(\mathbf{x}, y) \in (X, Y)$ **do**
    $a \leftarrow \mathbf{w} \cdot \mathbf{x} + w_{d+1}$;
    if $ya \leq 0$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$;
      $w_{d+1} \leftarrow w_{d+1} + y$;
    end
  end
end
return $(\mathbf{w}, w_{d+1})$;

Algorithm 2: Perceptron Test

Input: The $(d+1)$-dimensional weight vector $\mathbf{w}$ determining the separating hyperplane, the point $\mathbf{x} \in \mathbb{R}^d$ to classify
Output: The label of the point
$(\mathbf{w}', w_{d+1}) \leftarrow \mathbf{w}$;
return $sgn\big(\mathbf{w}' \cdot \mathbf{x} + w_{d+1}\big)$;

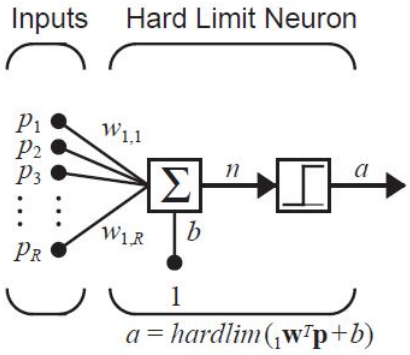

FIGURE 1. The perceptron architecture. Figure from [12].

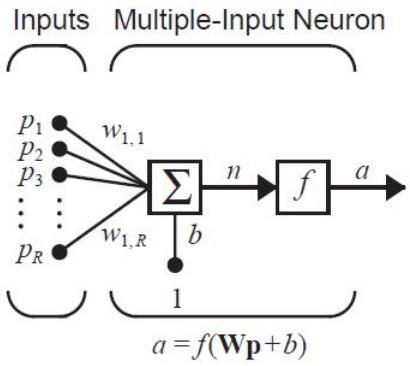

FIGURE 2. The neuron architecture. Figure from [12].

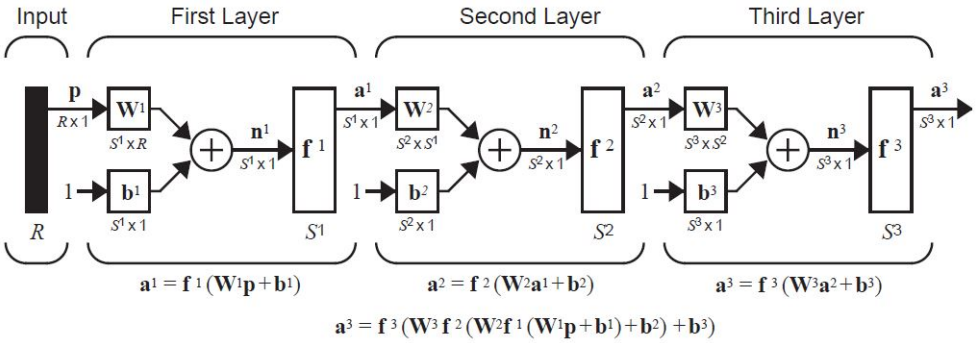

FIGURE 3. A three-layer, fully-connected neural network. Figure from [12].

A neural network is therefore a collection of neurons that are connected in layers. One such three-layer neural network is depicted in figure 3. A natural question is: what do we gain from this added complexity? The answer is that complex neural networks are generally able to learn input-output functions not learnable by simpler models. The same architecture can also be applied to a broad class of problems. The same cannot be said for analytic optimization methods. Indeed, an important result holds that real-valued continuous functions on the unit hypercube can be approximated to arbitrary degree with a two-layer neural network (see the Universal Approximation Theorem).

In the case of classifying data that is not necessarily linearly separable, neural networks are able to solve this problem while perceptrons cannot. In classification, a neural network produces some output $f(\mathbf{x})$ for each input $\mathbf{x}$. Then, $f(\mathbf{x})$ is input to a loss function $\mathcal{L}\big(f(\mathbf{x}), y\big)$.

With the loss, we can now update the weights of each *neuron* in the network by stochastic gradient descent, whose general outline is given in algorithm 3.

Finally, we can *further* regard a perceptron as a single-layer neural network with hinge-loss and one neuron under the hard-limit nonlinearity.

Thus, it is plausible that if there exists such a progression from linear classification to the perceptron, to the neuron, and to the neural network, there might be one in the quantum setting. Thus, it will be of great value to discuss quantum approaches to the early stages of linear classification and the perceptron.

Towards this end, we will now go over the pre-requisite classical notions for the optimal classical perceptron training algorithm, then the algorithm itself, some quantum notions, and finally the quantization of this algorithm.

Algorithm 3: Single Neuron Stochastic Gradient Descent

Input: The weight vector $\mathbf{w}$ for some neuron, the loss function $\mathcal{L}$, the input $\mathbf{x}$, small learning rate η

Result: Updates the linear weights of the neuron

$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}\big(f(\mathbf{x}), y\big)$;

3. Game Theory Background

In this section, we will cover material needed to understand the quantization of the optimal classical linear classification algorithm, discussed in the *current developments* section. Game theory is the mathematical study of games. Surprisingly, it is closely related to computer science by way of linear programming.

3.1. Zero-sum Games.

Although its relation to games isn't immediate, the following theorem due to von Neumann lies at the core of the field [16]:

Minimax Theorem. Let $X, Y \in \mathbb{R}^n$ be compact convex sets. Suppose $f : X \times Y \to \mathbb{R}$ is concave in X and convex in Y. Then,

$$\max_{x \in X} \min_{y \in Y} f(x, y) = \min_{y \in Y} \max_{x \in X} f(x, y).$$

Consider a zero-sum game between two players; that is, a game where if player 1 has a positive payoff, then player 2 has a negative payoff (and vice-versa). Suppose there are T turns. At each turn, each player can choose from a set of n moves. Suppose player 1 chooses the ith strategy

with probability p_i. Similarly, define q_i for player 2. Thus, $p, q \in \Delta$ where Δ denotes the unit-simplex $\{\mathbf{x} \in \mathbb{R}^n : \mathbf{x}_i \in [0,1] \wedge \sum_i \mathbf{x}_i = 1\}$. Let A denote the $n \times n$ payoff matrix where the ijth entry denotes the payoff for player 1 and player 2 making decision i and j respectively.

Since this game is zero-sum, player 1 chooses a strategy to maximize their payoff, given player 2 is minimizing the payoff of player 1. Thus, the payoff for player 1 is

$$\max_{p \in \Delta} \min_{q \in \Delta} p^T A q.$$

Similarly, player 2 chooses a strategy that minimizes the payoff of player 1. Thus, the payoff for player 2 is

$$\min_{q \in \Delta} \max_{p \in \Delta} p^T A q.$$

But by the minimax theorem, we receive the result [17]

$$\max_{p \in \Delta} \min_{q \in \Delta} p^T A q = \min_{q \in \Delta} \max_{p \in \Delta} p^T A q.$$

Thus, their payoffs are the same if they play optimally!

3.2. **Multiplicative Weights Algorithm.** Consider a scenario whereby there are n decisions, each carrying a cost $C_i^{(j)} \in [-1, 1]$ for the ith decision made on the jth turn. For T turns, the goal is to minimize the cost incurred. Intuitively, the multiplicative weights (MW) algorithm initially gives equal weight to each decision, and responds to feedback after subsequent turns [1]. Here, responding to feedback translates to penalizing incorrect decisions and rewarding correct ones. Treating the weights as probabilities, the algorithm minimizes the expected cost incurred over all turns.

This algorithm varies according to the update rule used for the weights. The particular variant we'll need is the Hedge algorithm due to Freund and Schapire [10]. It is shown in algorithm 4 for the previously discussed setup with n decisions, and T turns.

The hedge algorithm can be used to obtain an approximate solution to the minimax zero-sum game discussed earlier [1]. Suppose that we have an oracle that returns the optimal response distribution q given a distribution p. Then, we have the following result [1]:

Theorem 1. Given a payoff matrix A and optimal payoff λ^*, we want to find λ^* within some error $\epsilon \in \mathbb{R}^{>0}$; that is,

$$\lambda^* - \epsilon \leq \min_{q \in \Delta} Aq$$

$$\min_{p \in \Delta} p^T A \leq \lambda^* + \epsilon.$$

Algorithm 4: Hedge algorithm

Input: Learning rate η, number of turns T
initialize weight vector $\mathbf{w}$ with $\mathbf{w}_i = 1$ for all $i \in [n]$;
for $t = 1, 2, \ldots, T$ **do**
  initialize probability vector $\mathbf{p}$ with
  $\mathbf{p} = \{\mathbf{w}_1/|\mathbf{w}|, \mathbf{w}_2/|\mathbf{w}|, \ldots, \mathbf{w}_n/|\mathbf{w}|\}$;
  observe costs $C^{(t)}$;
  update weights by $\mathbf{w}_i \leftarrow \mathbf{w}_i \cdot e^{-\eta C_i^{(t)}}$;
end

Then, the Hedge algorithm finds the optimal payoff in $\mathcal{O}\big(\log(n)/\epsilon^2\big)$ oracle calls.

4. Classical Sublinear Perceptron

The classical sublinear perceptron algorithm due to [5] solves an approximate version of linear classification. Before this problem is stated formally, we need the following definition [7]:

Definition 1. As in the linear classification problem, suppose we are given linearly separable data $\mathcal{D}$. For the unit ball $\mathbb{B} \subseteq \mathbb{R}^n$, observe that $\mathbf{w} \cdot \mathbf{x}$ is the equation of some hyperplane where $\mathbf{w} \in \mathbb{B}$ and $\mathbf{x} \in \mathbb{R}^n$. Define the margin of the dataset to be

$$\gamma = \max_{\mathbf{x} \in \mathbb{B}} \min_{(\mathbf{x}, y) \in \mathcal{D}} y(\mathbf{w} \cdot \mathbf{x}).$$

Intuitively, the margin of a dataset is the largest possible distance that can be attained between a given dataset and any hyperplane [7]. Here, the distance between a hyperplane and a dataset is the smallest possible distance between the hyperplane and the closest data point to it.

We can now define the approximate linear classification problem [5]:

Approximate Linear Classification Problem. Suppose we are given n datapoints in $\mathbb{R}^d$ by way of an $n \times d$ input matrix A. Further suppose these datapoints are normalized to 1. For some error $\epsilon \in \mathbb{R}^{>0}$, find the ϵ-approximate solution $\mathbf{x}^*$; that is, find $\mathbf{x}^*$ such that $\forall j \in [n]$,

$$\max_{\mathbf{x} \in \mathbb{B}} \min_{i \in [n]} A_i \mathbf{x} - \epsilon \leq A_j \mathbf{x}^*.$$

[5] makes the salient observation that

$$\gamma = \max_{\mathbf{x} \in \mathbb{B}} \min_{i \in [n]} A_i \mathbf{x} = \max_{\mathbf{x} \in \mathbb{B}} \min_{p \in \Delta} p^T A \mathbf{x}$$

for unit-simplex Δ. Therefore, the von Neumann Minimax Theorem can be applied:

$$\max_{\mathbf{x}\in\mathbb{B}} \min_{p\in\Delta} p^T A\mathbf{x} = \min_{p\in\Delta} \max_{\mathbf{x}\in\mathbb{B}} p^T A\mathbf{x}$$

This looks very similar to a zero-sum game! We would, however, need to rectify the fact that we don't take $\mathbf{x}$ over a unit simplex; that is, $\mathbf{x}$ does not correspond to a probability distribution. With this observation, [5] builds upon a game theoretic algorithm for solving zero-sum games (due to [11]) to achieve sublinearity.

To be precise, they apply online gradient descent to solve the primal (original) problem and apply a variant of the hedge MW algorithm to its dual. The expected payoff from this "zero-sum game" is then the solution to the approximate linear classification problem. Their resultant algorithm is shown in algorithm 5.

Algorithm 5: Sublinear Perceptron algorithm (Classical)

Input: Error $\epsilon > 0$, dataset $A \in \mathbb{R}^{n\times d}$
initialize the number of turns $T \leftarrow 200^2\epsilon^{-2}\log n$, vector $y^{(1)} \leftarrow 0_d$, weights $w^{(1)} \leftarrow \mathbf{1}_n$;
for $t = 1, 2, \ldots, T$ **do**
  $x^{(t)} \leftarrow \frac{y^{(t)}}{\max\{1, \|y^{(t)}\|\}}$;
  choose $i^{(t)} \in [n]$ by calling a random variable returning $i^{(t)} \leftarrow i$ with probability $\frac{w_i^{(t)}}{\|w^{(t)}\|_1}$;
  choose $j^{(t)} \in [d]$ by calling a random variable returning $j^{(t)} \leftarrow j$ with probability $\frac{(x_j^{(t)})^2}{\|x^{(t)}\|^2}$;
  update $y^{(t+1)} \leftarrow y^{(t)} + \frac{1}{\sqrt{2T}} A_{i^{(t)}}$;
  for $i \in [n]$ **do**
    $(\tilde{v}^{(t)})_i \leftarrow A_{i^{(t)}j^{(t)}} \frac{\|x^{(t)}\|^2}{(x^{(t)})_{j^{(t)}}}$;
    $(v^{(t)})_i \leftarrow \max\left\{\frac{1}{\eta}, \min\{-\frac{1}{\eta}, (\tilde{v}^{(t)})_i\}\right\}$;
    update $(w^{(t)})_i \leftarrow (w^{(t)})_i\big(1 - \eta(v^{(t)})_i + \eta^2(v^{(t)})_i^2\big)$;
  end
end
Output: Hyperplane coefficients $\mathbf{x} = \frac{1}{T}\sum_t x^{(t)}$

As for the analysis of algorithm 5, we state the following without proof:

Theorem 2. Algorithm 5 runs in $\mathcal{O}\big(\epsilon^{-2}(n+d)\log n\big) = \tilde{\mathcal{O}}\big(\epsilon^{-2}(n+d)\big)$ oracle evaluations.

Theorem 3. Algorithm 5 returns an ϵ-approximate solution to the linear classification problem with probability $\frac{1}{2}$.

For the proofs to these results, consult the original text at [5].

5. Quantum Computing Background

In this section, we will cover material needed to fully understand the quantization of the optimal classical linear classification algorithm, discussed in the *current developments* section. An emphasis will be placed on circuit realization where possible.

5.1. **Dürr-Høyer Algorithm.** Let $T[1, 2, \ldots, N]$ be some table. Let there exist some index i whereby $T[i] = x$. Recall that Grover's algorithm, as modified by [3], finds i given x with quantum query complexity $\mathcal{O}(\sqrt{N/t})$, where t is the number of table values equal to x. The oracle in this case is the function $f : \{1, 2, \ldots, N\} \to \{0, 1\}$ defined by

$$f(i) = \begin{cases} 1 & \text{if } T[i] = x \\ 0 & \text{else} \end{cases}.$$

In general, denote this algorithm **Grover**$(f, 1)$ for oracle function f with target 1.

The Dürr-Høyer algorithm uses Grover search as a subroutine to solve the following problem.

Minimum Searching Problem. Let $T[1, \ldots, N]$ be a table of values from an ordered set. Find the index i such that $T[i]$ is minimum.

Detailed in figure 4 and algorithm 6, this routine uses classical processing in two areas. First, an oracle for a function $f_y : \{1, \ldots, N\} \to \{0, 1\}$ defined by

$$f_y(i) = \begin{cases} 1 & \text{if } T[i] < T[y] \\ 0 & \text{else} \end{cases}.$$

is formed by adding some classical post-processing to the oracle for f. Second, classical processing is used in the **Comparator** block of figure 4 to compare the classical bits of $|y\rangle$ and $|s\rangle$. If $T[y]$ is less than $T[s]$, we copy the measured result of Grover's algorithm into the storage bit $|s\rangle$.

Theorem 1. *In $\mathcal{O}(\sqrt{N})$ quantum queries, algorithm 6 finds the minimum index with probability at least $\frac{1}{2}$.*

Corollary 2. *After running algorithm 6 for c iterations, the probability of success is at least $1 - \left(\frac{1}{2}\right)^c$.*

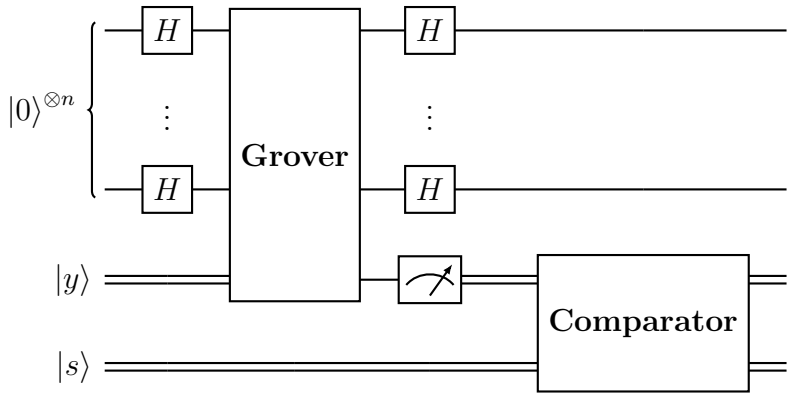

FIGURE 4. An implementation of a single unit from the **repeat** loop in algorithm 6. $|s\rangle$ is a classical storage bit that is compared with the measurement of $|y\rangle$.

For the short proofs to these results, consult the original document at [8].

Algorithm 6: Dürr-Høyer algorithm

Input: Table T of size N with guaranteed minimum
choose $y \in [N]$ from uniform distribution;
repeat
  prepare $\sum_j \frac{1}{\sqrt{N}} |j\rangle |y\rangle$;
  prepare Oracle for f_y by classical post-processing;
  $y' = \textbf{Grover}(f_y, 1)$;
  if $T[y'] < T[y]$ **then**
    let $y = y'$;
  end
until $\lceil 22.5\sqrt{N} + 1.4\ln(N)^2 \rceil$ *oracle evaulations*;
Output: Index y of minimum table value

5.2. **Amplitude Amplification Algorithm.** Amplitude amplification can be regarded as an extension of Grover's algorithm where it is no longer necessary to *a priori* know the success probability of finding a "good element" [4]. Described in algorithm 7, the algorithm resolves the following problem:

Quantum Search Problem. *Given an oracle O_f for a boolean function $f : [N] \to \{0, 1\}$, create a quantum circuit that can find $x \in \mathbb{Z}$ such that $f(x) = 1$ with probability near 1.*

Within the context of the algorithm, note that we need not use n Hadamard gates to form a superposition state; the algorithm works for any $\mathcal{A}$ where $\mathcal{A}|0\rangle$ can be decomposed into basis elements for the "good" and "bad" subspaces. We use the normal superposition state out of simplicity. Here, define

$$\mathbf{Q} = -\mathcal{A}S_0\mathcal{A}^{-1}S_f,$$

where S_0 and S_f are constructed from the oracle as in Grover's algorithm. Recall that S_0 flips the sign of $|0\rangle$ while S_f flips all good states (i.e. states x such that $f(x) = 1$).

As for analysis, we state the following result without proof:

Theorem 1. *Apply amplitude amplification to a function f with $t \in \mathbb{N}$ good elements amongst N elements total. Then, algorithm 7 finds a good element with high probability in $\mathcal{O}(\sqrt{N/t})$ oracle evaluations.*

For the full proof, please refer to the original paper at [4].

5.3. **Amplitude Estimation Algorithm.** In Grover's algorithm and in the non-fixed-point Amplitude Amplification algorithm, recall that to determine the number of iterations, we need the initial probability a of measuring some state x from $\mathcal{A}|0\rangle$ such that $f(x) = 1$ (in the case of Grover's algorithm, $\mathcal{A} = H^{\otimes n}$). This is precisely the problem amplitude estimation resolves [4]:

Amplitude Estimation Problem. Given a unitary $\mathcal{A}$, and a function $f : [N] \to \{0, 1\}$, suppose $|\phi\rangle = \mathcal{A}|0\rangle$ can be decomposed in terms of its projection onto the kernel of f (denoted $|\phi_0\rangle$) and its orthogonal complement (denoted $|\phi_1\rangle$). Since elements of the latter map to 1 under f (the "good elements"), estimate the probability of finding a "good element:"

$$a = \langle\phi_1|\phi_1\rangle\,.$$

Where $\mathbf{Q}$ is defined as in amplitude amplification as

$$\mathbf{Q} = -\mathcal{A}S_0\mathcal{A}^{-1}S_f,$$

the routine solving the problem is shown in algorithm 8. As in phase estimation, we also have

$$\Lambda_N(U)\,|ij\rangle = |i\rangle\left(U^i\,|j\rangle\,\right).$$

The number of oracle evaluations of **Est_Amp** is in $\mathcal{O}(M)$. For the

Algorithm 7: Amplitude Amplification (**QSearch**)

Input: Oracle O_f, constant $c \in (1, 2)$, $\mathcal{A} = H^{\otimes n}$
initialize $h \leftarrow 0$;
prepare $\mathbf{Q}$ as in Grover's algorithm;
repeat
  $h \leftarrow h + 1$;
  initialize $M \leftarrow \lceil c^h \rceil$;
  prepare superposition state $|\psi\rangle = \mathcal{A}\,|0\rangle^{\otimes n}$;
  measure $|\psi\rangle$ and store result x;
  if $f(x) = 1$ **then**
    return;
  else
    prepare register with state $|\psi\rangle = \mathcal{A}\,|0\rangle^{\otimes n}$;
    choose some j from a uniform distribution on $[1, M]$;
    apply Q to $|\psi\rangle$ for j iterations for $|\psi'\rangle = Q^j\,|\psi\rangle$;
    measure $|\psi'\rangle$ and store the result x;
    if $f(x) = 1$ **then**
      return;
    end
  end
until *good* $x \in \mathbb{Z}$ *found*;

Algorithm 8: Amplitude Estimation (**Est_Amp**)

Input: Unitary $\mathcal{A}$, boolean function $f : [N] \to \{0, 1\}$, $M \in \mathbb{N}$
initialize register one to $|0\rangle^M$ and register two to $\mathcal{A}\,|0\rangle^M$;
apply QFT_M to the first register;
apply $\Lambda_M(\mathbf{Q})$ across both registers;
apply $\mathrm{QFT}_M^\dagger$ to the first register;
measure the first register and store as $|y\rangle$;
Output: $a \approx \sin^2(\pi \frac{y}{M})$

specific bounds on the result, consult [4] since they will not be covered here for the sake of brevity. One very important application of amplitude estimation is to the follow problem [4]:

Counting Problem. Given a boolean function $f : [N] \to \{0, 1\}$, count the number of elements in the domain mapping to 1.

This problem can be resolved with algorithm 9. In terms of com-

Algorithm 9: Count

Input: Boolean function $f : [N] \to \{0, 1\}$, error margin $0 < \epsilon \leq 1$

initialize $l \leftarrow 0$;

repeat

  update $l \leftarrow l + 1$;

  $t' \leftarrow$ **Est_Amp**$\big(\mathrm{QFT}_N, f, 2^l\big)$;

until $t' \neq 0$ *and* $2^l < 2\sqrt{N}$;

initialize $M \leftarrow \lceil \frac{20\pi^2}{\epsilon} 2^l \rceil$;

$t' \leftarrow$ **Est_Amp**$\big(\mathrm{QFT}_N, f, M\big)$;

Output: $\tilde{t} \in \mathbb{N}$ such that $|\tilde{t} - t'| \leq \frac{2}{3}$

plexity, we state the following without proof:

Theorem 1. *Algorithm 9 outputs an estimate $\tilde{t}$ to the Counting Problem with probability at least $\frac{2}{3}$ such that*

$$|\tilde{t} - t| \leq \epsilon t.$$

The expected number of oracle evaluations is in $\mathcal{O}(\frac{1}{\epsilon}\sqrt{N/t})$. If there are no "good elements" (i.e. $t = 0$), we get an exact result in $\mathcal{O}(\sqrt{N})$.

Thus, we need not check every element's value in order to determine how many are equal to 1!

6. Current Developments

6.1. **Quantum Sublinear Perceptron.** Finally, we state the algorithm for the quantum sublinear perceptron, which quantizes the classical algorithm solving the approximate linear classification problem [15]. Specifically, we seek to find the optimal payoff from the "zero-sum game"

$$\max_{w \in \mathbb{B}} \min_{p \in \Delta} p^T X w = \min_{p \in \Delta} \max_{w \in \mathbb{B}} p^T X w.$$

But first, we establish some notation and define two subroutines. Take the following conventions:

$$\mathrm{clip}(v, 1/\eta) = \min\left\{1/\eta, \max\{-1/\eta, v\}\right\}$$

$$w_t(i) \equiv \text{The } i\text{th component of the } t\text{th iterate of } w$$

Where a is some vector, the first subroutine, denoted **Prep_State**(O_a), returns the following quantum state given an oracle $O_a |i\rangle |0\rangle = |i\rangle |a_i\rangle$:

$$\frac{1}{\|a\|_2} \sum_{i \in [n]} a_i |i\rangle$$

In essence, the oracle O_a generates the ith component of the vector a. Thus, **Prep_State**(O_a) returns a "quantum" version of this vector. The subroutine is shown in algorithm 10.

Algorithm 10: Prep_State

Input: Vector "coordinate" oracle O_a
$a_{\max} \leftarrow$ **Dürr-Høyer**(O_a);
prepare superposition state $|\psi\rangle = H^{\otimes n} |0\rangle^{\otimes n}$;
prepare register $|\psi\rangle |0\rangle^{\otimes(1+2+4)n}$;
perform operations on register to produce

$$O_a^\dagger(O_a \times I_{4n})(O_a \times I_{6n}) |\psi\rangle$$

$$= \frac{1}{\sqrt{n}} \sum_{i\in[n]} |i\rangle |0\rangle \left(\frac{a_i}{a_{\max}} |0\rangle + \sqrt{1 - \frac{|a_i|^2}{a_{\max}^2}} |1\rangle \right) \tag{1}$$

discard middle $|0\rangle$ to produce

$$|\psi'\rangle = \frac{\|a\|_2}{a_{\max}\sqrt{n}} \left(\frac{1}{\|a\|_2} \sum_{i\in[n]} a_i |i\rangle \right) |1\rangle + \left|a^\perp\right\rangle |0\rangle , \tag{2}$$

where $\left|a^\perp\right\rangle$ is a garbage state;
prepare oracle O_f for a boolean function returning 1 if and only if the second qubit is in computational basis $|1\rangle$;
choose some constant $c \in (1, 2)$ and unitary $\mathcal{A}$;
$|\phi\rangle \leftarrow$ **QSearch**$(O_f, c, \mathcal{A})$;
return $|\phi\rangle$;

The second subroutine, denoted **Prep_Oracle**, updates a persistent vector when given the previous weight vector iterates $w_1, w_2, \ldots, w_t$ and respective iterates $j_1, j_2, \ldots, j_t$ probabilistically chosen from $[d]$ [15]. In essence, it functions as an update oracle O_t whereby

$$O_t |i\rangle |0\rangle = |i\rangle |u_{t+1}(i)\rangle .$$

This subroutine is detailed in algorithm 11. Finally, the quantum sublinear perceptron algorithm is detailed in algorithm 12. We state without proof the following result [15]:

Theorem 1. *With probability at least* $2/3$*, algorithm 12 returns a solution to the approximate linear classification problem in quantum complexity* $\tilde{O}\left(\frac{\sqrt{n}}{\epsilon^4} + \frac{\sqrt{d}}{\epsilon^8}\right)$*. This, in fact, is the optimal complexity as well.*

Algorithm 11: Prep_Oracle

Input: Vector iterates $w_1, w_2, \dots w_t$, distribution iterates $j_1, \dots, j_t$, and access oracle O_X

prepare oracle $O_{s,j} |0\rangle = |j_s\rangle$;

prepare oracle $O_{s,w} |j_s\rangle = \left| \frac{\|w_s\|^2}{w_s(j_s)} \right\rangle$;

prepare oracle $O_{\text{clip}} |abc\rangle = \left|c\big(1 - \eta \text{clip}(ab, 1/\eta) + \eta^2 \text{clip}(ab, 1/\eta)^2\big)\right\rangle$;

for $s = 1, \dots, t$ **do**

$O_{s,j}^\dagger O_X^\dagger O_{s,w}^\dagger O_{\text{clip}} O_{s,w} O_X O_{s,j} |i\rangle |000\rangle |u_s(i)\rangle$

end

7. Open Problems & Future Work

With respect to the quantum sublinear perceptron algorithm, the clearest area for improvement is the algorithm's dependence on the margin of approximation error ϵ. While it polynomially improves upon the classical sublinear perceptron algorithm and achieves optimality with a corresponding lower bound, the quantum algorithm increases the dependence on ϵ by a factor of 4. Thus, it is impossible to better the complexity polynomially, but the error dependence can be improved.

As for implications of the quantum perceptron algorithm, it opens up many applications. For example, [14] has applied its results to achieve novel complexity for portfolio optimization. As for its implications for neural networks, it could perhaps have implications for training due to its polynomial improvement in online gradient descent.

As for my uneducated view on creating a quantum framework for neural networks, I believe the most interesting avenue is in the tensor network view of traditional NNs. There has been some work in this area (see [6], [9], [2]).

Algorithm 12: Sublinear Perceptron algorithm (Quantum)

Input: Error $\epsilon > 0$, access oracle O_X

initialize the number of turns $T \leftarrow 27^2\epsilon^{-2}\log n$, persistent $y_1 \leftarrow \mathbf{0}_d$, and persistent $u_1 \leftarrow \mathbf{1}_n$;

prepare superposition state $|p_1\rangle = \frac{1}{\sqrt{n}}\sum_{i\in[n]}|i\rangle$;

for $t = 1, 2, \ldots, T$ **do**

measure $|p_t\rangle$ in computational basis. store result in i_t;

initialize $y_{t+1} = y_t + \frac{1}{\sqrt{2T}}T_{i_t}$;

apply **Est_Amp** for $2\lceil\log T\rceil$ times with precision η^2. take the median of these estimates. denote it $\widetilde{\|y_t\|^2}$;

prepare vector "coordinate" oracle O_{y_t} returning the coordinates of y_t;

measure **Prep_State**(O_{y_t}) in the computational basis. denote the result j_t;

for $i \in [n]$ **do**

$\tilde{v_t}(i) \leftarrow X_i(j_t)\widetilde{\|y_t\|^2}/\big(y_t(j_t)\max\{1, \widetilde{\|y_t\|^2}\}\big)$;

$v_t(i) \leftarrow \mathrm{clip}(\tilde{v_t}(i), 1/\eta)$;

$u_{t+1}(i) \leftarrow u_t(i)\big(1 - \eta v_t(i)\eta^2 v_t(i)^2\big)$;

end

$O_t \leftarrow$ **Prep_Oracle**$\Big((w_1, w_2, \ldots w_t), (j_1, j_2, \ldots, j_t)\Big)$;

update $|p_{t+1}\rangle \leftarrow$ **Prep_State**(O_t), where

$$|p_{t+1}\rangle = \frac{1}{\|u_{t+1}\|_2}\sum_{i\in[n]} u_{t+1}(i)\,|i\rangle \tag{3}$$

end

return Hyperplane coefficients $\mathbf{w} = \frac{1}{T}\sum_t^T \frac{y_t}{\max\{1, \widetilde{\|y_t\|}^2\}}$

References

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. "The Multiplicative Weights Update Method: a Meta-Algorithm and Applications". In: *Theory of Computing* 8.6 (2012), pp. 121–164. DOI: 10.4086/toc.2012.v008a006. URL: http://www.theoryofcomputing.org/articles/v008a006.

[2] Jacob Biamonte. *Lectures on Quantum Tensor Networks*. 2019. arXiv: 1912.10049 [quant-ph].

[3] Michel Boyer et al. "Tight Bounds on Quantum Searching". In: *Fortschritte der Physik* 46.4-5 (1998), 493–505. ISSN: 1521-3978. DOI: 10.1002/(sici)1521-3978(199806)46:4/5<493::aid-prop493>3.0.co;2-p. URL: http://dx.doi.org/10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P.

[4] Gilles Brassard et al. *Quantum Amplitude Amplification and Estimation*. 2000. arXiv: quant-ph/0005055 [quant-ph].

[5] Kenneth L. Clarkson, Elad Hazan, and David P. Woodruff. *Sublinear Optimization for Machine Learning*. 2010. arXiv: 1010.4408 [cs.LG].

[6] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. "Quantum convolutional neural networks". In: *Nature Physics* 15.12 (2019), 1273–1278. ISSN: 1745-2481. DOI: 10.1038/s41567-019-0648-8. URL: http://dx.doi.org/10.1038/s41567-019-0648-8.

[7] Hal Daumé. *A Course in Machine Learning*. 2017.

[8] Christoph Durr and Peter Hoyer. *A Quantum Algorithm for Finding the Minimum*. 1996. arXiv: quant-ph/9607014 [quant-ph].

[9] G. Evenbly and G. Vidal. "Tensor Network States and Geometry". In: *Journal of Statistical Physics* 145.4 (2011), 891–918. ISSN: 1572-9613. DOI: 10.1007/s10955-011-0237-4. URL: http://dx.doi.org/10.1007/s10955-011-0237-4.

[10] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: http://www.sciencedirect.com/science/article/pii/S002200009791504X.

[11] Lov K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv: quant-ph/9605043 [quant-ph].

[12] Martin Hagan et al. *Neural Network Design*. 2nd ed.

[13] Gareth James et al. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN: 1461471370.

[14] Iordanis Kerenidis, Anupam Prakash, and Dániel Szilágyi. *Quantum Algorithms for Portfolio Optimization*. 2019. arXiv: 1908.08040 [math.OC].

[15] Tongyang Li, Shouvanik Chakrabarti, and Xiaodi Wu. *Sublinear quantum algorithms for training linear and kernel-based classifiers*. 2019. arXiv: 1904.02276 [quant-ph].

[16] John von Neumann. "Zur Theorie der Gesellschaftsspiele". In: *Mathematische Annalen* (100 1928), 295–320. DOI: 10.1007/BF01448847.

[17] Aaron Roth. *Lecture notes in Algorithmic Game Theory*. 2017. URL: https://www.cis.upenn.edu/~aaroth/courses/slides/agt17/lect06.pdf.