Tunable Quantum Neural Networks in the QPAC-Learning Framework

Viet Pham Ngoc
Imperial College London
London, United Kingdom
viet.pham-ngoc17@imperial.ac.uk

David Tuckey
Imperial College London
London, United Kingdom
david.tuckey17@imperial.ac.uk

Herbert Wiklicky
Imperial College London
London, United Kingdom
h.wiklicky@imperial.ac.uk

Abstract

In this paper, we investigate the performances of tunable quantum neural networks in the Quantum Probably Approximately Correct (QPAC) learning framework. Tunable neural networks are quantum circuits made of multi-controlled $\mathbf{X}$ gates. By tuning the set of controls these circuits are able to approximate any Boolean functions. This architecture is particularly suited to be used in the QPAC-learning framework as it can handle the superposition produced by the oracle. In order to tune the network so that it can approximate a target concept, we have devised and implemented an algorithm based on amplitude amplification. The numerical results show that this approach can efficiently learn concepts from a simple class.

1 Introduction

Machine learning is believed to be an application of quantum computing that will yield promising results. It is the reason why, in the past years, significant efforts have been put into developing machine learning techniques suited to quantum computers. One interesting approach to this task consists in adapting existing classical techniques [1, 2, 3, 4] to take advantage of quantum properties and gain some speed-up.

Introduced by [5], probably approximately correct (PAC) learning provides a mathematical framework to analyse classical machine learning techniques. This framework revolves around the existence of an oracle that provides samples drawn from the instance space. These examples are then used as examples for an algorithm to learn a target concept. The efficiency of this learning algorithm can then be characterised thanks to its sampling complexity. Given the momentum gained by quantum techniques for machine learning, it seems natural for a quantum equivalence of PAC-learning to be introduced. This is exactly what was done in [6] with QPAC-learning. In this framework, instead of providing samples from the instance space, the oracle generates a superposition of all the examples. Similarly to PAC-learning, QPAC-learning can be used to evaluate the learning efficiency of quantum algorithm.

In this paper we are using this framework to study a learning algorithm based on quantum amplitude amplification [7]. This fundamental quantum procedure allows us both to compare the error rate to a threshold as well as increase the probability of measuring the errors produced by the learner. These measured errors are then used to tune the learner in order to decrease the error rate. In our case, this learner is a tunable quantum neural network [8] which is essentially a quantum circuit made of multi-controlled $\mathbf{X}$ gates. We prove that this setup is able to learn efficiently by implementing it and testing it against a simple class of concepts.

2 Related Works

Studied in [8], the tunable quantum neural networks (TNN) are an interesting kind of quantum circuits in the sense that they naturally express Boolean functions hence are able to approximate any target function. At the core of this concept is the fact that a Boolean function can be expanded into a polynomial form called the algebraic normal form (ANF). This polynomial form can then easily be transposed into a quantum circuit.

Let $u = u_0 \dots u_{n-1} \in \mathbb{B}^n$, we denote $\mathbf{1}_u = \{i \in [0, n-1] \mid u_i = 1\}$ and for $x \in \mathbb{B}^n$, $x^u = \prod_{i \in \mathbf{1}_u} x_i$. Let $f \in \mathbb{B}^{\mathbb{B}^n}$ then its algebraic normal form is:

$$f(x) = \bigoplus_{u \in \mathbb{B}^n} \alpha_u x^u \tag{1}$$

Where $\alpha_u \in \{0, 1\}$ denotes the absence or the presence of the monomial x^u in the expansion. As an example, let us consider the function $g \in \mathbb{B}^{\mathbb{B}^2}$ defined by $g(x_0, x_1) = 1 \oplus x_1 \oplus x_0.x_1$. With the previously introduced notations, its ANF can be written: $g(x) = x^{00} \oplus x^{01} \oplus x^{11}$ with $\alpha_{00} = \alpha_{01} = \alpha_{11} = 1$ and $\alpha_{10} = 0$. Using this form, any Boolean function can be expressed with a quantum circuit made of multi-controlled $\mathbf{X}$ gates.

Let $f \in \mathbb{B}^{\mathbb{B}^n}$ with ANF as in Eq (1) and consider a quantum circuit of $n+1$ qubits with the first n qubits being denoted $q_0, \dots, q_{n-1}$ and the last one being the ancillary qubit, denoted a. Now for $u \in \mathbb{B}^n$ such that $\alpha_u = 1$ in the ANF of f, place an $\mathbf{X}$ gate controlled by the qubits $\{q_i \mid i \in \mathbf{1}_u\}$ on the ancillary qubit. Let $\mathbf{QC}$ be the resulting circuit, then it is such that for $x \in \mathbb{B}^n$ and $b \in \mathbb{B}$:

$$\mathbf{QC} \left|x\right\rangle \left|b\right\rangle = \left|x\right\rangle \left|b \oplus f(x)\right\rangle$$

Let us consider the function $g \in \mathbb{B}^{\mathbb{B}^2}$ introduced previously, then the circuit pictured in Figure 1 is expressing g.

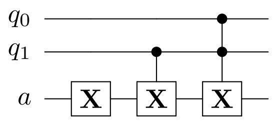


Figure 1: Circuit expressing $g(x) = 1 \oplus x_1 \oplus x_0.x_1$.

A tunable neural network is then a quantum circuit of this type for which the set of controlled $\mathbf{X}$ gates can be tuned so that the expressed function is an approximation of a target function. Now let $f \in \mathbb{B}^{\mathbb{B}^n}$ and consider the network $\mathbf{TNN}(f)$ expressing f, then for a superposition of the form $\left|\phi\right\rangle = \sum_{x \in \mathbb{B}^n} d_x \left|x\right\rangle \left|0\right\rangle$ we have:

$$\mathbf{TNN}(f) \left|\phi\right\rangle = \sum_{x \in \mathbb{B}^n} d_x \left|x\right\rangle \left|f(x)\right\rangle$$

Which is reminiscent of the output from the oracle encountered in the QPAC-learning framework.

QPAC-Learning has been introduced in [6] and is a quantum version of the work presented in [5]. In this framework, we call $\mathcal{C} \subseteq \mathbb{B}^{\mathbb{B}^n}$ a class of concept and the aim is to learn $c \in \mathcal{C}$. A probability distribution D is placed over $\mathbb{B}^n$ and for a hypothesis $h \in \mathbb{B}^{\mathbb{B}^n}$ the error is defined by

$$err_D(h, c) = P_{x \sim D}(h(x) \neq c(x)) \tag{2}$$

For $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$, the goal is then to produce $h \in \mathbb{B}^{\mathbb{B}^n}$ such that:

$$P\left(err_D(h, c) < \epsilon\right) > 1 - \delta \tag{3}$$

A class of concept $\mathcal{C}$ is said to be PAC-learnable with an algorithm A if for $c \in \mathcal{C}$, for all distribution D and for $0 < \epsilon < \frac{1}{2}$, $0 < \delta < \frac{1}{2}$, A will output a hypothesis $h \in \mathbb{B}^{\mathbb{B}^n}$ verifying Equation (3).

During the learning process, we are given access to an oracle $\mathbf{EX}(c, D)$ such that each call to this oracle will produce the superposition $\left|\Psi(c, D)\right\rangle$:

$$\left|\Psi(c, D)\right\rangle = \sum_{x \in \mathbb{B}^n} \sqrt{D(x)} \left|x\right\rangle \left|c(x)\right\rangle$$

For the rest of the paper we will assume that $\mathbf{EX}(c, D)$ is a quantum gate such that:

$$\mathbf{EX}(c, D) \left|0\right\rangle = \left|\Psi(c, D)\right\rangle$$

With these notations, for a concept c and a hypothesis h we have:

$$err_D(h,c) = \sum_{h(x)\neq c(x)} D(x)$$

A learning algorithm is then said to be an efficient PAC-learner when the number of calls to $\mathbf{EX}$ that are necessary to attain Eq (3) is polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

The learning algorithm presented in this paper is based on quantum amplitude amplification. Introduced by [7] the quantum amplitude amplification algorithm is a generalisation of Grover's algorithm [9]. Let G be the set of states we want to measure and $\mathcal{A}$ be a quantum circuit such that $\mathcal{A}\left|0\right\rangle = \left|\Phi\right\rangle$ with:

$$\left|\Phi\right\rangle = \sum_{x\notin G} d_x \left|x\right\rangle + \sum_{x\in G} d_x \left|x\right\rangle$$

This state can be rewritten as

$$\left|\Phi\right\rangle = \cos(\theta)\left|\Phi_B\right\rangle + \sin(\theta)\left|\Phi_G\right\rangle \tag{4}$$

With $\theta \in \left[0, \frac{\pi}{2}\right]$, $\cos(\theta) = \sqrt{\sum_{x\notin G}|d_x|^2}$, $\left|\Phi_B\right\rangle = \frac{1}{\cos(\theta)}\sum_{x\notin G} d_x\left|x\right\rangle$, $\sin(\theta) = \sqrt{\sum_{x\in G}|d_x|^2}$ and $\left|\Phi_G\right\rangle = \frac{1}{\sin(\theta)}\sum_{x\in G} d_x\left|x\right\rangle$. We also have $\left\langle\Phi_B|\Phi_G\right\rangle = 0$

Now let $\mathcal{X}_G$ be such that:

$$\mathcal{X}_G\left|x\right\rangle = \begin{cases} -\left|x\right\rangle & \text{if } x\in G \\ \left|x\right\rangle & \text{otherwise}\end{cases}$$

Then we have $\mathcal{X}_G\left|\Phi_G\right\rangle = -\left|\Phi_G\right\rangle$ and $\mathcal{X}_G\left|\Phi_B\right\rangle = \left|\Phi_B\right\rangle$. We also define $\mathcal{X}_0 = \mathbf{I} - 2\left|0\right\rangle\left\langle 0\right|$ then by denoting $\mathbf{Q} = -\mathcal{A}\mathcal{X}_0\mathcal{A}^{-1}\mathcal{X}_G$, $\mathbf{Q}$ is the diffusion operator and acts on $\left|\Phi\right\rangle$ in the following way:

$$\mathbf{Q}^m\left|\Phi\right\rangle = \cos\left((2m+1)\theta\right)\left|\Phi_b\right\rangle + \sin\left((2m+1)\theta\right)\left|\Phi_g\right\rangle \text{ for } m\in\mathbb{N}$$

Suppose we know θ, then by choosing m such that $(2m+1)\theta \approx \frac{\pi}{2}$, for example $m = \left\lfloor \frac{1}{2}\left(\frac{\pi}{2\theta} - 1\right)\right\rfloor$, we are ensured that when measuring, a state in G will almost certainly be measured.

This amplitude amplification procedure is at the heart of numerous quantum amplitude estimation algorithms [7, 10, 11]. In this setup, a state of a form similar to that of Eq (4) is given but $\theta \in \left[0, \frac{\pi}{2}\right]$ is unknown and these algorithms seek to approximate $\sin(\theta)$. Where [7] makes use of quantum Fourier transforms an integrant part of the algorithm, [10, 11] do without while still maintaining quantum speed-up. If $a = \sin(\theta) \in [0,1]$ is the amplitude to be evaluated and $\epsilon, \delta > 0$, then these algorithms output an estimation $\tilde{a}$ such that $P(|\tilde{a} - a| < a\epsilon) > 1 - \delta$.

3 TNN in the QPAC-Learning Framework

As said previously, tunable networks are particularly well suited to be employed in the QPAC-learning framework. Let $\mathcal{C} \subseteq \mathbb{B}^{\mathbb{B}^n}$ be a class of concept and $c \in \mathcal{C}$. Let D be a probability distribution over $\mathbb{B}^n$ and $\mathbf{EX}(c,D)$ the oracle such that:

$$\mathbf{EX}(c,D)\left|0\right\rangle = \left|\Psi(c,D)\right\rangle = \sum_{x\in\mathbb{B}^n}\sqrt{D(x)}\left|x\right\rangle\left|c(x)\right\rangle$$

Now let $\mathbf{TNN}(h)$ be a tunable neural network expressing $h \in \mathbb{B}^{\mathbb{B}^n}$ in its current state. Then we have:

$$\begin{aligned}\left|\Phi\right\rangle &= \mathbf{TNN}(h)\left|\Psi(c,D)\right\rangle \\ &= \sum_{x\in\mathbb{B}^n}\sqrt{D(x)}\left|x\right\rangle\left|c(x)\oplus h(x)\right\rangle \\ &= \sum_{h(x)=c(x)}\sqrt{D(x)}\left|x\right\rangle\left|0\right\rangle + \sum_{h(x)\neq c(x)}\sqrt{D(x)}\left|x\right\rangle\left|1\right\rangle\end{aligned}$$

The corresponding circuit is given in Figure 2.
The state $\left|\Phi\right\rangle$ can then be rewritten as:

$$\left|\Phi\right\rangle = \cos(\theta_e)\left|\phi_g\right\rangle\left|0\right\rangle + \sin(\theta_e)\left|\phi_e\right\rangle\left|1\right\rangle$$

With $\left\langle\phi_g, 0|\phi_e, 1\right\rangle = 0$. This state is similar to the one encountered in the amplitude amplification procedure. Because $\sin(\theta_e) = \sqrt{\sum_{h(x)\neq c(x)} D(x)}$, we also have

$$\sin^2(\theta_e) = \sum_{h(x)\neq c(x)} D(x) = err_D(h,c)$$

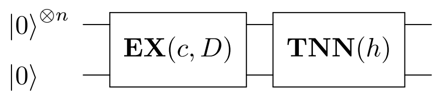


Figure 2: Circuit resulting in the state $|\Phi\rangle$.

In order to use tunable neural networks in the QPAC-learning framework we thus need a mean to estimate $\sin^2(\theta_e)$ and compare it to $0 < \epsilon < 1/2$. If the error is smaller than ϵ, stop. Otherwise, tune the network so that it expresses another hypothesis h' and repeat. The algorithm proposed in this paper does just that by using amplitude amplification.

4 Tuning Algorithm

At its core, the algorithm is similar to quantum amplitude estimation, the difference being that it compares the unknown amplitude $err_D(h, C)$ to ϵ instead of estimating the amplitude to a relative error of ϵ. Let θ_{err} and θ_ϵ such that $\sin^2(\theta_{err}) = err_D(h, C)$ and $\sin^2(\theta_\epsilon) = \epsilon$. Because D is a distribution we can take $\theta_{err} \in \left[0, \frac{\pi}{2}\right]$ and because $0 < \epsilon < \frac{1}{2}$ we can assume $\theta_\epsilon \in \left]0, \frac{\pi}{4}\right[$. There are then two possible cases: either $err_D(h, C) \geq \epsilon$ or $err_D(h, C) < \epsilon$. Because sin is increasing on $\left[0, \frac{\pi}{2}\right]$, these translate into $\theta_{err} \geq \theta_\epsilon$ or $\theta_{err} < \theta_\epsilon$ respectively. Now let m_{max} be the smallest integer such that $(2m_{\text{max}} + 1)\theta_\epsilon \geq \frac{\pi}{4}$, we introduce this following simple lemma:

Lemma 4.1. *Let $\theta \in \left[0, \frac{\pi}{2}\right]$. If for all $m \leq m_{max}$, we have $\sin^2\left((2m + 1)\theta\right) < \frac{1}{2}$ then $\theta < \theta_\epsilon$*

Proof. We show this by contraposition.
Let $\theta \geq \theta_\epsilon$.
Suppose $\theta \in \left[\frac{\pi}{4}, \frac{\pi}{2}\right]$, then taking $m = 0 \leq m_{\text{max}}$, we have $\sin^2(\theta) \geq \frac{1}{2}$.
Now suppose that $\theta \in \left[0, \frac{\pi}{4}\right[$. By definition we have $m_{\text{max}} = \left\lceil \frac{1}{2}\left(\frac{\pi}{4\theta_\epsilon} - 1\right)\right\rceil$ and we denote $m = \left\lceil \frac{1}{2}\left(\frac{\pi}{4\theta} - 1\right)\right\rceil$, then $m \leq m_{\text{max}}$. Let $0 \leq \alpha < 1$ such that $m = \frac{1}{2}\left(\frac{\pi}{4\theta} - 1\right) + \alpha$. Then we have $(2m + 1)\theta = \frac{\pi}{4} + 2\alpha\theta$ and $\frac{\pi}{4} \leq (2m + 1)\theta \leq \frac{3\pi}{4}$. This leads to $\sin^2((2m + 1)\theta) \geq \frac{1}{2}$.
Hence Lemma 4.1. □

This result is illustrated in Figures 3 and 4.

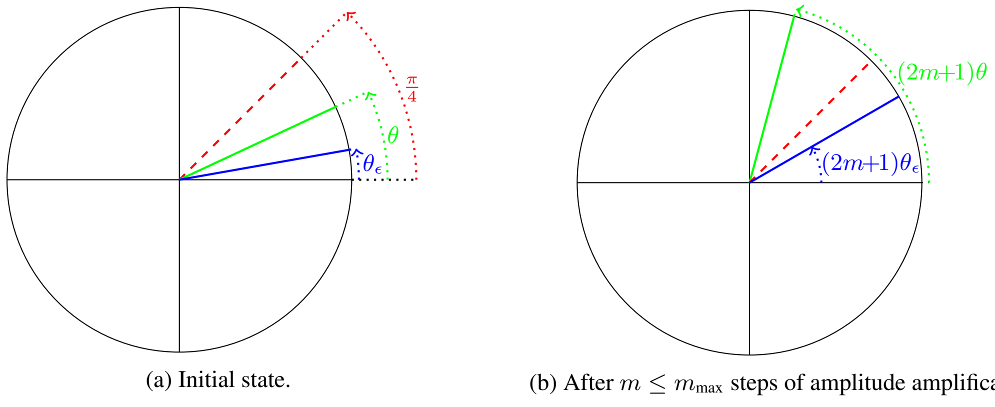
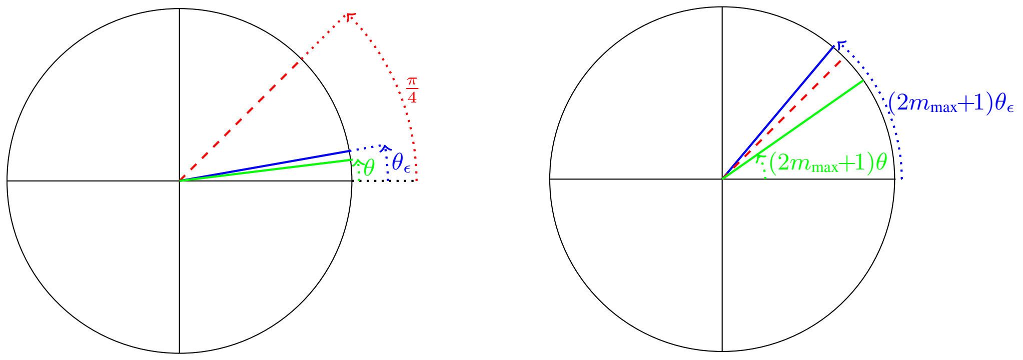


Figure 3: Case where $\theta \geq \theta_\epsilon$.

The tuning algorithm works as follow. After each update of the tunable network, the error is compared to ϵ thanks to Lemma 4.1: after $m_0 < m_{\text{max}}$ steps of amplification, the resulting state is measured N times and the number S of measurements for which the ancillary qubit is in state $|1\rangle$ is counted.
If $S > \frac{N}{2}$, then the error is greater than ϵ and the network is updated using the measurements on the first n qubits. If on the other hand $S \leq \frac{N}{2}$, the process is repeated with $m_0 + 1 \leq m_{\text{max}}$ steps of amplification.
The algorithm stops when the network reaches a state such that even after m_{max} steps of amplification we have $S \leq \frac{N}{2}$. It is thus necessary to choose N such that when the algorithm stops the error is most probably lower than ϵ.


$\frac{\pi}{4}$
θ
θ_ϵ
$(2m_{\text{max}}+1)\theta_\epsilon$
$(2m_{\text{max}}+1)\theta$

(a) Initial state. (b) After m_{max} steps of amplitude amplification.

Figure 4: Case where $\theta < \theta_\epsilon$.

To avoid the angle $(2m_{\text{max}} + 1)\theta_\epsilon$ from overshooting $\frac{\pi}{4}$ by too much, we have chosen to limit ϵ to $\left]0, \frac{1}{10}\right[$. This limitation can be achieved without loss of generality by scaling the problem by a factor of $\frac{1}{5}$. This can be done by introducing a second ancillary qubit and applying a controlled rotation gate $\mathbf{R}$ such that $\mathbf{R}\left|10\right\rangle = \frac{2}{\sqrt{5}}\left|10\right\rangle + \frac{1}{\sqrt{5}}\left|11\right\rangle$. The states of interest are then the ones for which the two ancillary qubits are in state $\left|11\right\rangle$. This means that for $\epsilon \in \left]0, \frac{1}{2}\right[$, we are effectively working with $\epsilon' = \frac{\epsilon}{5} \in \left]0, \frac{1}{10}\right[$ as required. With this procedure, the effective error is itself limited to $\left[0, \frac{1}{5}\right]$. This also ensures that we have $\theta_{err} \in \left[0, \arcsin\left(\frac{1}{\sqrt{5}}\right)\right]$ meaning that $\theta_{err} \neq \frac{\pi}{4}$ which is a stationary point of the amplitude amplification process so the error will always be amplified.

To account for this additional procedure, the diffusion operator has to be redefined. We now denote $\mathcal{A}(h) = \left(\mathbf{I}^{\otimes n} \otimes \mathbf{R}\right)\left((\mathbf{TNN}(h)\mathbf{EX}(c, D)) \otimes \mathbf{I}\right)$, $\mathcal{X}_G = \mathbf{CZ}(a_0, a_1)$, the controlled $\mathbf{Z}$ gate for which the control is the first ancillary qubit and the target is the second ancillary qubit and $\mathcal{X}_0 = \mathbf{I} - 2\left|0\right\rangle\left\langle 0\right|$. These allow us to define the diffusion operator $\mathbf{Q}$ in a similar way as previously with $\mathbf{Q}(h) = -\mathcal{A}(h)\mathcal{X}_0\mathcal{A}^{-1}(h)\mathcal{X}_G$. The corresponding circuit is represented in Figure 5 and the tuning algorithm is given in Algorithm 1.

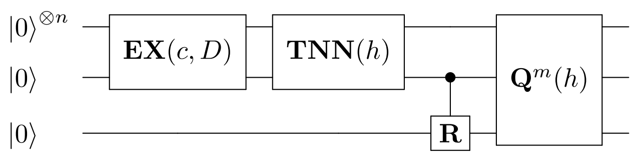


Figure 5: Circuit used to tune the network.

In Algorithm 1 the instruction "Update $\mathbf{TNN}$ according to *errors*" is given without further specification as the way it is done might depend on the class of concepts that is being learnt. The update strategy used to learn the class of concepts introduced in Section 6 is given in Algorithm 2

5 Proof and Analysis of the Algorithm

The algorithm stops when the network is in such a state that even after m_{max} rounds of amplitude amplification, if S is the number of measurements for which the ancillary qubits are in state $\left|11\right\rangle$, then $S \leq \frac{N}{2}$ where N is the number of measurements. This stage is only reached if for all the m in the schedule we also have $S \leq \frac{N}{2}$ after m rounds of amplification. Let $\left|\Phi\right\rangle = \mathcal{A}\left|0\right\rangle$, then it can be written:

$$\left|\Phi\right\rangle = \cos(\theta_e)\left|\phi_\perp\right\rangle + \sin(\theta_e)\left|\phi_e\right\rangle\left|11\right\rangle$$

Where $\left|\phi_\perp\right\rangle$ is orthogonal to $\left|\phi_e\right\rangle\left|11\right\rangle$ and $\left|\phi_e\right\rangle$ contains the inputs for which the network returns the wrong output. After m_{max} rounds of amplification we have:

$$\mathbf{Q}^{m_{\text{max}}}\left|\Phi\right\rangle = \cos((2m_{\text{max}} + 1)\theta_e)\left|\phi_\perp\right\rangle + \sin((2m_{\text{max}} + 1)\theta_e)\left|\phi_e\right\rangle\left|11\right\rangle$$

Algorithm 1: Tuning Algorithm

```
Data: $0 < \epsilon < \frac{1}{2}$, $0 < \delta < \frac{1}{2}$, $\mathbf{EX}(c, D)$
Result: $\mathbf{TNN}$ expressing $h^*$ such that $P(err_D(h^*, c) < \epsilon) > 1 - \delta$
$N \leftarrow 2\left(\lfloor \frac{1}{\pi\delta^2} \rfloor //2\right) + 2$;
$m_{\text{max}} \leftarrow$ smallest integer such that $(2m_{\text{max}} + 1)\arcsin\left(\sqrt{\frac{\epsilon}{5}}\right) \geq \frac{\pi}{4}$;
$\mathbf{TNN} \leftarrow \mathbf{I}$;
$m \leftarrow -1$;
$errors \leftarrow []$;
while $m < m_{max}$ or $length(errors) > \frac{N}{2}$ do
    $m \leftarrow m + 1$;
    $errors \leftarrow []$;
    $\mathcal{A} \leftarrow \left(\mathbf{I}^{\otimes n} \otimes \mathbf{R}\right)\left((\mathbf{TNN}.\mathbf{EX}(c, D)) \otimes \mathbf{I}\right)$;
    $|\Phi\rangle \leftarrow \mathcal{A}\,|0\rangle$;
    $\mathbf{Q} \leftarrow -\mathcal{A}(h)\mathcal{X}_0\mathcal{A}^{-1}(h)\mathcal{X}_G$;
    for $1 \leq n \leq N$ do
        Measure $\mathbf{Q}^m\,|\Phi\rangle$;
        if 11 is measured on the ancillary qubits then
            append the string of the first $n$ qubits to $errors$
        end
    end
    if $length(errors) > \frac{N}{2}$ then
        Update $\mathbf{TNN}$ according to $errors$;
        $m \leftarrow -1$;
    end
end
```

If p is the probability of measuring 11 on the ancillary qubits then $p = \sin^2((2m_{\text{max}} + 1)\theta_e)$ and we want to compare it to $\frac{1}{2}$ in order to apply Lemma 4.1. This boils down to estimating $P\left(p < \frac{1}{2} \mid S \leq \frac{N}{2}\right)$. We have:

$$P(p < 1/2 \mid S \leq N/2) = \frac{P(S \leq N/2 \mid p < 1/2)P(p < 1/2)}{P(S \leq N/2)}$$

By placing a uniform marginal distribution on p we get:

$$P(p < 1/2 \mid S \leq N/2) = \frac{\sum_{k=0}^{N/2} \binom{N}{k} \int_0^{\frac{1}{2}} \theta^k (1-\theta)^{N-k} d\theta}{\sum_{k=0}^{N/2} \binom{N}{k} \int_0^{1} \theta^k (1-\theta)^{N-k} d\theta} \tag{5}$$

From now on we assume that N is even. For $a \in [0, 1]$ it can be shown by integrating by parts that:

$$\int_0^a \theta^k (1-\theta)^{N-k} d\theta = \frac{k!(N-k)!}{(N+1)!} - (1-a)^{N-k+1} \sum_{i=0}^{k} \frac{k!(N-k)!}{(k-i)!(N-k+i+1)!} a^{k-i}(1-a)^i$$

This leads to:

$$\int_0^1 \theta^k (1-\theta)^{N-k} d\theta = \frac{k!(N-k)!}{(N+1)!}$$

And

$$\sum_{k=0}^{N/2} \binom{N}{k} \int_0^1 \theta^k (1-\theta)^{N-k} d\theta = \frac{N+2}{2(N+1)} \tag{6}$$

Similarly we have:

$$\int_0^{\frac{1}{2}} \theta^k (1-\theta)^{N-k} d\theta = \frac{k!(N-k)!}{(N+1)!} - \left(\frac{1}{2}\right)^{N+1} \sum_{i=0}^{k} \frac{k!(N-k)!}{(k-i)!(N-k+i+1)!}$$

And

$$\sum_{k=0}^{N/2} \binom{N}{k} \int_0^{\frac{1}{2}} \theta^k (1-\theta)^{N-k} d\theta = \frac{N+2}{2(N+1)} - \left(\frac{1}{2}\right)^{N+1} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \frac{N!}{(k-i)!(N-k+i+1)!} \tag{7}$$

Putting (5), (6) and (7) together yields:

$$\begin{aligned}
P(p < 1/2 \mid S \leq N/2) &= 1 - \left(\frac{1}{2}\right)^N \frac{N+1}{N+2} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \frac{N!}{(k-i)!(N-k+i+1)!} \\
&= 1 - \left(\frac{1}{2}\right)^N \frac{1}{N+2} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \frac{(N+1)!}{(k-i)!(N-k+i+1)!} \\
&= 1 - \left(\frac{1}{2}\right)^N \frac{1}{N+2} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \binom{N+1}{k-i} \\
&= 1 - \left(\frac{1}{2}\right)^N \frac{1}{N+2} \sum_{k=0}^{N/2} \sum_{i=0}^{k} \binom{N+1}{i} \\
&= 1 - \left(\frac{1}{2}\right)^N \frac{1}{N+2} \sum_{k=0}^{N/2} \binom{N+1}{k} \left(\frac{N}{2} + 1 - k\right) \qquad (8)
\end{aligned}$$

Now

$$\sum_{k=0}^{N/2} \binom{N+1}{k} \left(\frac{N}{2} + 1 - k\right) = \frac{N+2}{2} \sum_{k=0}^{N/2} \binom{N+1}{k} - \sum_{k=0}^{N/2} \binom{N+1}{k} k \qquad (9)$$

For $k > 0$ we have:

$$\binom{N+1}{k} k = \binom{N}{k-1} (N+1)$$

And N being even, we also have:

$$\sum_{k=0}^{N/2} \binom{N+1}{k} = 2^N$$

Plugging these back into (9) we get:

$$\sum_{k=0}^{N/2} \binom{N+1}{k} \left(\frac{N}{2} + 1 - k\right) = 2^{N-1}(N+2) - (N+1) \sum_{k=0}^{\frac{N}{2}-1} \binom{N}{k}$$

Coming back to (8), we thus have:

$$P(p < 1/2 \mid S \leq N/2) = \frac{1}{2} + \left(\frac{1}{2}\right)^N \frac{N+1}{N+2} \sum_{k=0}^{\frac{N}{2}-1} \binom{N}{k} \qquad (10)$$

Because N is even:

$$\sum_{k=0}^{\frac{N}{2}-1} \binom{N}{k} = \frac{1}{2} \left(2^N - \binom{N}{N/2}\right)$$

Together with (10), this leads to:

$$P(p < 1/2 \mid S \leq N/2) = \frac{1}{2} + \left(\frac{1}{2}\right)^{N+1} \frac{N+1}{N+2} \left(2^N - \binom{N}{N/2}\right)$$

But [12]:

$$\binom{N}{N/2} \leq \sqrt{2} \frac{2^N}{\sqrt{\pi N}}$$

So:

$$P(p < 1/2 \mid S \leq N/2) \geq \frac{1}{2} + \frac{N+1}{2(N+2)} \left(1 - \frac{\sqrt{2}}{\sqrt{\pi N}}\right)$$

Because $\frac{N+1}{N+2} \sim 1$ we look for a lower bound of the form $1 - \frac{\alpha}{\sqrt{N}}$. For example, we have:

$$P(p < 1/2 \mid S \leq N/2) > 1 - \frac{1}{\sqrt{\pi N}}$$

So for $0 < \delta < \frac{1}{2}$, in order to have $P(p < 1/2 \mid S \leq N/2) > 1 - \delta$, it suffices to take for N an even integer that is greater than $\frac{1}{\pi\delta^2}$, hence:

$$N = 2\left(\left\lfloor \frac{1}{\pi\delta^2} \right\rfloor //2\right) + 2$$

We have shown that when the algorithm stops, we have:

$$P\left(\sin^2\left((2m_{\text{max}} + 1)\theta_e\right) < \frac{1}{2}\right) > 1 - \delta$$

Together with Lemma 4.1, it comes that:

$$P(\theta_e < \theta_\epsilon) > 1 - \delta$$

Hence:

$$P(err < \epsilon) > 1 - \delta$$

So each sampling phase requires $\frac{1}{\pi\delta^2}$ calls to the oracle $\mathbf{EX}(c, d)$. In order to perform one update of the network, the algorithm requires at most $m_{\text{max}} \approx \frac{1}{2}\left(\frac{\pi}{4\arcsin\left(\sqrt{\epsilon/5}\right)} - 1\right)$ of these sampling phases. Now for $\epsilon \in \left]0, \frac{1}{2}\right[$, we have $\arcsin\left(\sqrt{\epsilon/5}\right) \approx \sqrt{\epsilon/5}$ so for one update of the network, the total number of call to $\mathbf{EX}(c, d)$ is $O\left(\frac{1}{\delta^2}\frac{1}{\sqrt{\epsilon}}\right)$. The number of updates needed to reach the learning target is dependant on the class of concept.

It is possible to reduce the number of calls to $\mathbf{EX}(c, d)$ during one update phase by incrementing m not by 1 but with powers of a given number. In this case the total number for an update is $O\left(\frac{1}{\delta^2}\log\left(\frac{1}{\epsilon}\right)\right)$ but this comes at the cost of possibly having a lower probability of success as we will see in Section 6.

6 Learning a Particular Class

We are interested in learning the class of concepts $\mathcal{C}_m$ defined by:

$$\mathcal{C}_m = \{\alpha \oplus x^u \mid \alpha \in \mathbb{B},\ u \in \mathbb{B}^n\}$$

In order to learn concepts from this class, we are applying the update strategy shown in Algorithm 2:

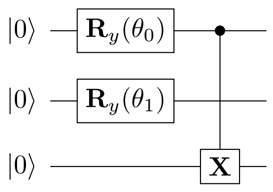
Algorithm 2: Update strategy

```
Data: $errors$ the list of measured erroneous inputs
Result: $gates$ the list of gates to be updated
$gates \leftarrow [errors[0]]$;
for err in errors do
    $empty \leftarrow True$;
    for $0 \leq i < len(gates)$ do
        $x \leftarrow gates[i]$;
        $inter \leftarrow \mathbf{1}_x \cap \mathbf{1}_{err}$;
        if $inter \neq \emptyset$ then
            $gates[i] \leftarrow inter$;
            $empty \leftarrow False$
        end
    end
    if empty = True then
        append $err$ to $gates$;
    end
end
```

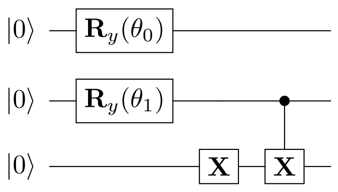
This approach has been implemented for $n = 4, 8$ and 10 using the Qiskit library[1]. For a given n, a probability distribution over $\mathbb{B}^n$ has been created randomly by applying to each qubit a $\mathbf{R}_y$ rotation gate with an angle randomly chosen in $[0, \pi]$. The controlled $\mathbf{X}$ gates corresponding to the target concept are then added to the circuit. These two blocks taken together are constituting the oracle $\mathbf{EX}(c, D)$. The construction of such oracles for $n = 2$ is illustrated in

[1]The code can be found in the following repository: https://gitlab.doc.ic.ac.uk/vjp17/tnn-in-qpac.

Figure 6. The **TNN** has then been trained to learn the concepts of the class $\mathcal{C}_m$ according to the training algorithm and the update strategy introduced in this paper. For $n = 4$ the experiments have been run for all the concepts of the class. For $n = 8$ and 10 the experiments have been performed for 32 randomly selected concepts from this class. Once the training has been completed, the error is evaluated using the Qiskit statevector simulator by running the circuit composed of the oracle and the tuned network and getting the amplitude of all the inputs for which the network's output is wrong.



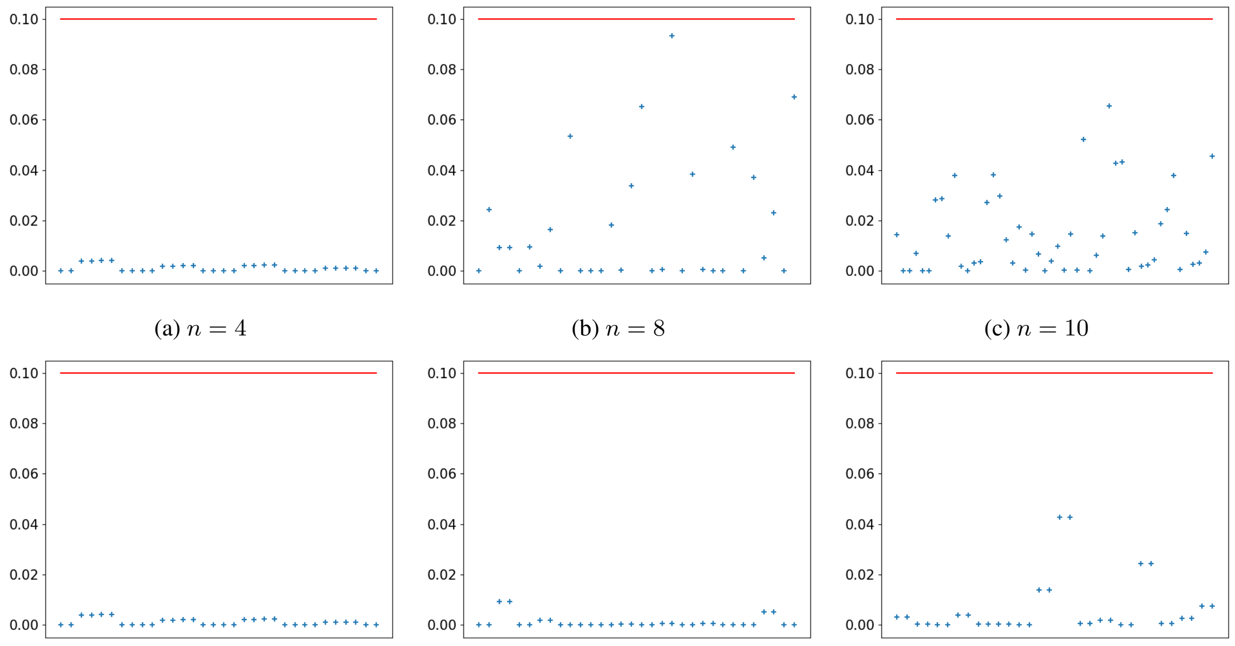
(a) Oracle for $c(x) = x^{10}$.



(b) Oracle for $c(x) = 1 \oplus x^{01}$.

Figure 6: Implementation of the oracle for $n = 2$ and different target concepts.

For each function, the training has been performed five times for different values of ϵ and δ. In order to reduce the running time, we have chosen to increment m with powers of 2 for these five runs. The results of some of these experiments are gathered in Figure 7.

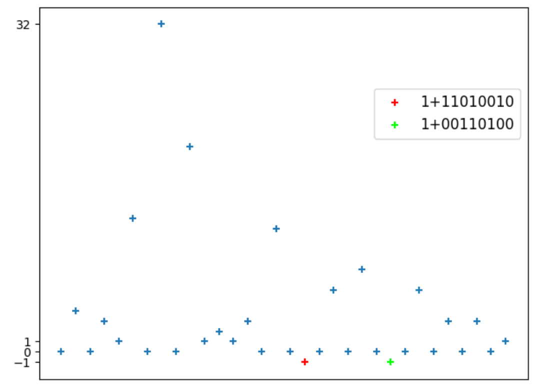


(a) $n = 4$ (b) $n = 8$ (c) $n = 10$

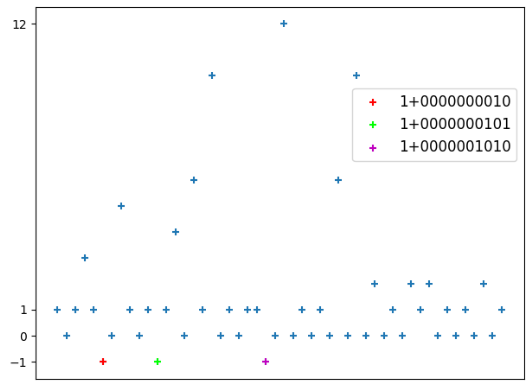
(d) $n = 4$ (e) $n = 8$ (f) $n = 10$

Figure 7: Error rate after one of the five training runs. 7a, 7b, and 7c have been obtained with $\epsilon = 0.1$ and $\delta = 0.1$. 7d, 7e and 7f have been obtained with $\epsilon = 0.1$ and $\delta = 0.001$. Each datapoint represent a different function spread on the x-axis and the y-axis represents the error rate. The red line is $\epsilon = 0.1$.

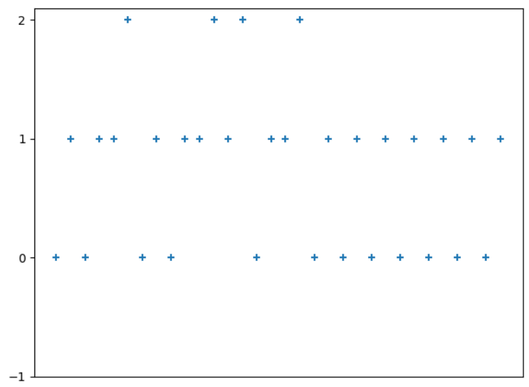
As can be seen in Figure 7, for the same function and the same ϵ, when δ decreases, the error rate gets further from ϵ. This can be explained by the fact that decreasing δ will increase the number of samples, hence increasing the chance of catching errors during the sampling phases. This enriches the input set of the update algorithm, allowing it to produce more accurate gates to be updated which has two consequences. The first is that a lower number of updates are needed to reach the training target, as can be seen in Figures 8. In order to minimise the computation time, we have introduced a maximum number of updates equal to 100. If the number of updates during a training reaches this limit, the training is stopped, the error is set to 0 and the number of updates set to -1. As illustrated in Figure 8a this happened for two functions labelled ”1+110110010” and ”1+00110100” corresponding to $1 \oplus x^{110110010}$ and $1 \oplus x^{00110100}$. Even by ignoring these 2 functions, the maximum number of updates is still equal to 32, whereas by decreasing δ to 0.001, the maximum number of updates is only 2 as shown in Figure 8c. The second consequence is a lower error rate which goes in the same direction as a larger probability of success.



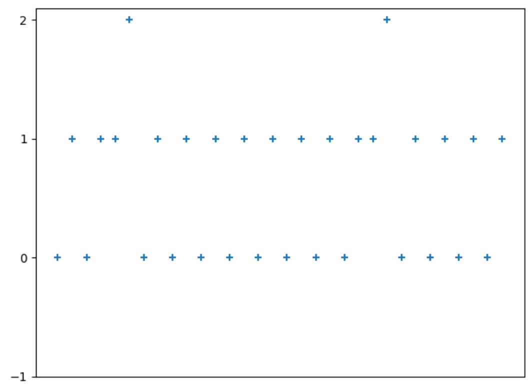
(a) $n = 8, \epsilon = 0.1, \delta = 0.1$



(b) $n = 10, \epsilon = 0.1, \delta = 0.1$



(c) $n = 8, \epsilon = 0.1, \delta = 0.001$



(d) $n = 10, \epsilon = 0.1, \delta = 0.001$

Figure 8: Figures 8a, 8b, 8c and 8d represent the number of updates needed to obtain the error rates in Figures 7b, 7c, 7e and 7f respectively. The y-axis represents the number of updates with -1 marking the functions that needed more than 100 updates. In 8a, the number of updates goes up to 32 and 12 in 8b. The marked functions are those that reached 100 updates during the training phase. In Figures 8c and 7f the maximum number of updates is 2 showing that reducing δ also reduces the number of updates needed.

When running the five training runs with the parameters $n = 4$, $\epsilon = 0.001$ and $\delta = 0.1$ we encountered the situation depicted in Figure 9a where two functions ended up with an error rate greater than ϵ. In order to estimate the probability for these events, we ran the training for the function labelled ”0+1101” 100 times and computed the resulting error rate. First we kept incrementing m with powers of 2, then we incremented it by 1 to study the influence of the incrementation schedule on the probability for the error to be less than ϵ.

Looking at Figures 9b and 9c it seems that the way m is incremented indeed does have an impact on the success probability. From Figure 9b, in the case where m is incremented with powers of 2, an estimation gives $P(error < \epsilon) = 0.65 < 1 - \delta = 0.9$. On the other hand, from Figure 9c, in the case where m is incremented by 1, an estimation would give $P(error < \epsilon) = 0.97 > 1 - \delta = 0.9$.

From these experiments, it seems that for δ small enough, the sample complexity of our algorithm applied to this class is $O\left(\frac{1}{\delta^2}\frac{1}{\sqrt{\epsilon}}\right)$ but could potentially be reduced to $O\left(\frac{1}{\delta^2}\log\left(\frac{1}{\epsilon}\right)\right)$.

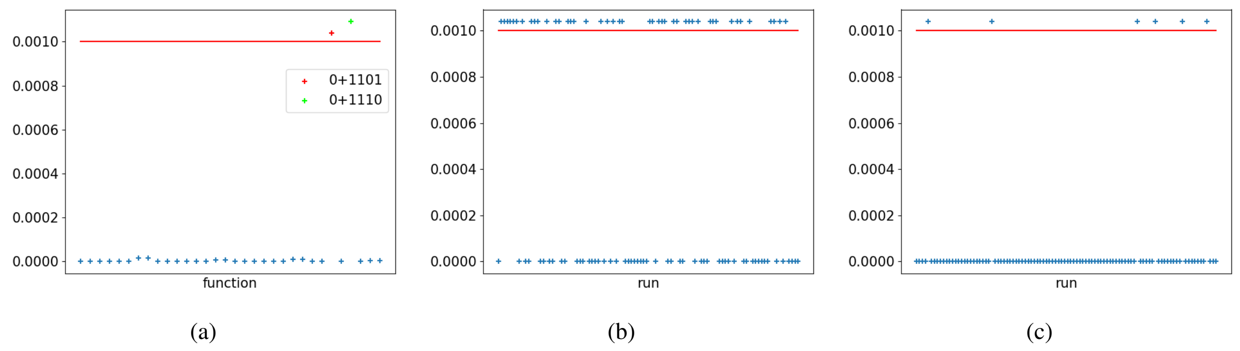


Figure 9: Error rate post-training for $n = 4$, $\epsilon = 0.001$ and $\delta = 0.1$ in 9a. We then focus on the function labelled "0+1101". In 9b, the network is trained 100 times with m being incremented with powers of 2 while in 9c, m is incremented by 1. The y-axis of the three figures carries the error rate with the red line representing $\epsilon = 0.001$. By increasing m exponentially instead of linearly the QPAC target is missed.

7 Conclusion

In this paper we have studied tunable quantum neural networks in the context of QPAC-learning. To do so, we have devised and proved a learning algorithm that uses quantum amplitude amplification. Amplitude amplification is used to both compare the error rate to the threshold ϵ and to better measure the errors. These measurements are then used to update the network. We have implemented this approach and tested it against a simple class of concepts and found that this algorithm is indeed an efficient learner as its sample complexity is $O(\frac{1}{\delta^2}\frac{1}{\sqrt{\epsilon}})$. Experimental results show that most of the error rates are quite far away from the threshold ϵ, which could be explained by an excessive number of samples. As future work we will look for a tighter lower bound to the probability of success that might reduce the dependence in $\frac{1}{\delta}$ in the algorithm's complexity. Similarly, we will study the possibility of reducing the complexity's dependence on $\frac{1}{\epsilon}$ from $\frac{1}{\sqrt{\epsilon}}$ to $\log\left(\frac{1}{\epsilon}\right)$. Finally we will study the generalisation of this approach to more complex classes of concepts.

References

[1] P. Rebentrost, T. R. Bromley, C. Weedbrook, and S. Lloyd. Quantum hopfield neural network. *Physical Review A*, 98(4), Oct 2018.

[2] F. Tacchino, C. Macchiavello, D. Gerace, and D. Bajoni. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1):1–8, Dec 2019.

[3] M. Schuld, I. Sinayskiy, and F. Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586, Nov 2014.

[4] S. Lloyd, M. Mohseni, and P. Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arViv e-print*, Jul 1, 2013.

[5] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, Nov 5, 1984.

[6] N.H. Bshouty and J.C. Jackson. Learning dnf over the uniform distribution using a quantum example oracle. *SIAM Journal on Computing*, 28(3):1136–1153, Jan 1998.

[7] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp. *Quantum Amplitude Amplification and Estimation*, volume 305 of *Quantum computation and information*, pages 53–74. Amer. Math. Soc., Providence, RI, 2002. 81P68 (81-02); 1947332.

[8] V. Pham Ngoc and H. Wiklicky. Tunable quantum neural networks for boolean functions, 2020.

[9] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[10] K. Nakaji. Faster amplitude estimation, 2020.

[11] S. Aaronson and P. Rall. Quantum approximate counting, simplified. *Symposium on Simplicity in Algorithms*, page 24–32, Jan 2020.

[12] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Ltd, 2005.