Loading Classical Distributions into Quantum States via Quantum Generative Adversarial Network

Eden Schirman*
School of Physics and Astronomy, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel-Aviv University, Tel-Aviv 6997801, Israel and
School of Electrical Engineering, Iby and Aladar Fleischman Faculty of Engineering, Tel Aviv University, Tel Aviv 69978, Israel
(Dated: August 24, 2021)

In the last decade, quantum computing underwent intense study and gained significant media coverage due to the improvement of quantum hardware. However, it is not yet clear what would be the usage of it for real world applications. Nonetheless, it is common assumption that the loading of classical probability distributions into quantum states in an efficient way will be a key enabler in utilizing the quantum advantage. In this work, the loading of classical distributions into quantum states using quantum generative adversarial network (QGAN) was researched. Specifically, an improvement of previous work has been achieved by improving the classical neural network in the model. Improvement of factor $10 - 100$ in the averaged relative entropy was achieved compared to previous work by Zoufal *et al.* (2019) [1].

I. INTRODUCTION

Quantum computing is a new way of computing, i.e. processing information by utilizing the unique features of quantum mechanics (QM) [2]. Specifically, by exploiting the superposition and entanglement features, quantum computing enables the calculation of several computational tasks in parallel. This is called "quantum parallelism" and it offers a significant computational speed-up compared to classical computing. However, due to the probabilistic nature of QM, naive measurement of the output of a quantum computation yields the outcome of only one computational task. Therefore, in order to take advantage of the quantum parallelism, one needs to calculate some global features of the entire tasks space.

In 1995, Shor have devised a quantum algorithm for factoring prime numbers in polynomial time [3]. Prime factorization is the basis of modern encryption (*e.g.* RSA), hence this algorithm ignited the interest of both private companies and governments. In recent years, the race for building the first quantum computer is accelerating with multinational corporations like Google, Amazon and IBM investing billions of dollars. Nonetheless, the realization of a complete quantum computer is still years ahead, so researchers look for ways to utilize the current partial quantum computers for solving real world problem. One of these avenues is quantum machine learning (QML).

QML is an hybrid quantum-classical approach which utilizes both quantum computing and machine learning [4]. For example, a possible way to build a quantum circuit with a desired functionality (the quantum equivalent of a classical circuit) is to initiate a generic quantum circuit that is governed by some classical parameters, and then to optimize these parameters with a ML algorithm according to the desired output.

In this work, I followed several papers published by members of IBM Quantum Research *et al.* [1, 5, 6] on how to price stock option using QML. Specifically, I implemented and improved the loading of classical distributions into quantum states using a quantum generative adversarial network (QGAN). This is a key step for pricing options on a quantum computer in particular, but an important capability for utilizing the quantum speed-up in general. The rest of the paper is organized as follows. First, in section II, the motivation will be discussed by describing the big picture of what and how to calculate option pricing using a quantum computer, and why the QGAN is needed for loading distributions. Then, a thorough theoretical analysis of loading classical distributions into quantum states using QGAN will be discussed in section III. The methods of the practical implementation are presented in section IV followed by the results section V. The paper will be concluded with an overall discussion VI.

II. THE BIG PICTURE

A. Classical Option Pricing

A basic stock option is a contract between a buyer and a seller that gives the buyer the right to buy or sell the stock at a future date at a specific, predetermined price (called the strike price). For example, suppose a stock worth 10$ on $t = 0$ and Alice buys

* schirman.eden@gmail.com

from Bob the option to buy the stock at a strike price of 12\$ between $t = 1$ to $t = 10$. If, in $t = 4$ the stock's price rose to 14\$, Alice might exercise the option and buy the stock from Bob in 12\$ and to sell it in the market at 14\$. Alice's payoff will be $14\$ - 12\$ = 2\$$, and her net profit will be 2\$ less the amount it cost her to buy the option. The complexity of option pricing naturally arises, due to the statistical nature of the stock market. It should be priced in such a way that it would be worth it for the buyer to buy, but on the other hand for the seller to sell.

Pricing an option that could be exercised on a specific time in the future is manageable to calculate numerically in a straight forward way, since it requires the estimation of the stock's price distribution at a specific date. However, as in the example above, pricing an option that could be exercised at some time-frame $t_1 - t_2$, is much harder since the space of the possible prices is much bigger. A common method to tackle the problem is by using Monte Carlo (MC) simulations [7]. According to the stochastic model one chooses to work with (usually the Black-Sholes-Merton model [8]), stochastic differential equations are used to simulate N random paths of possible evolutions of the stock's price from t_1 to t_2. For each path, the average value of the stock is calculated, followed by a calculation of the path's payoff. Then, an average of all simulated paths is taken in order to calculate an estimator of the mean payoff of the stochastic process (this is called an Asian option [5]).

Let us denote the stock's price random process by $\mathbf{S}$, and the random variables representing the stock's price at specific time points as $\{S_t\}$. A realization of the random process (variable) will be denoted by $\mathbf{S}^l$ (S_t^l) where $l \in \{1, 2, ..., N\}$. The strike price will be denoted as K. Then, the payoff of each path, f^l, is calculated as:

$$f^l = max(\overline{\mathbf{S}^l} - K, 0), \tag{1}$$

where $\overline{\mathbf{S}^l}$ is the average of the stock's price realization l. The average of f across all paths is the unbiased estimator of the mean payoff:

$$\hat{\mu}_f = \frac{1}{N} \sum_{l=1}^{N} f^l. \tag{2}$$

The mean payoff estimation could be used as the price of the option to buy the stock in the determined time-frame $t_1 - t_2$ at the strike price K.

The law of large numbers states that the estimator $\hat{\mu}_f$ converges to the expectation value μ_f with an error of order $\mathcal{O}(\frac{1}{\sqrt{N}})$. However, with the usage of a quantum computer, a similar estimation of the mean could be performed with an error of order $\mathcal{O}(\frac{1}{N})$. Meaning with $\mathcal{O}(\sqrt{N})$ operations on a quantum computer, one can achieve the same precision as with $\mathcal{O}(N)$ operations on a classical computer. This quadratic speed-up is realized with the quantum amplitude estimation algorithm (QAE). In the next subsection the QAE will be explained, and then the usage of the algorithm for pricing an option will be presented in the following subsection.

B. Quantum Amplitude Estimation

The quantum circuit model is the model of quantum computing that uses quantum gates to manipulate quantum states in order to process information. The quantum states are some abstract states in Hilbert space and are composed from qubits. A quantum gate is the implementation of some unitary operator. Suppose there is a function $f(x) : \{0,1\}^n \to \{0,1\}$. The function is applied to a quantum state via a unitary operator U_f which satisfies:

$$U_f \left|x\right\rangle_n \left|0\right\rangle_1 = \left|x\right\rangle_n \left|f(x)\right\rangle_1, \tag{3}$$

where $x \in \{0, 1, .., 2^n - 1\}$, and the subscript n (1) indicates that the corresponding state is composed from n (1) qubits. Furthermore, suppose there is a unitary operator $\mathcal{A}$ such that:

$$\mathcal{A} \left|0\right\rangle_n \left|0\right\rangle_1 = \sqrt{1-a} \left|\psi_0\right\rangle_n \left|0\right\rangle_1 + \sqrt{a} \left|\psi_1\right\rangle_n \left|1\right\rangle_1. \tag{4}$$

The quantum amplitude estimation (QAE) algorithm estimates the value of a [9]. It does so by applying some other operator $\mathcal{Q} = \mathcal{Q}(\mathcal{A})$, M times. The estimator $\hat{a}$ converges to the real value of a up to an error of order $\mathcal{O}(\frac{1}{M})$. This way, by applying $\mathcal{O}(M)$ operations, a precision of $\mathcal{O}(\frac{1}{M})$ is achieved.

C. Quantum Option Pricing

An overall analysis of how to utilize the QAE algorithm for pricing option with a quadratic speed-up will be made. The analysis will not describe the small details, but mainly will explain the main steps necessary for executing the algorithm. The setup assumes that the time-stock's price grid is discritized into a grid of $2^n T$ cells. The time line is discritized into T points, and the stock price is discritized and transformed by an affine transformation such that:

$$S_j = S_{min} + j\Delta S, \tag{5}$$

where S_j is the real stock value, $\Delta S = \frac{S_{max} - S_{min}}{2^n - 1}$, and $j \in \{0, 1, ..., 2^n - 1\}$ is represented by an n qubit quantum state $\left|j\right\rangle_n$. At each one of the T time points, the stock's price is represented by an n qubits

state. Therefore, for representing an entire path, an nT qubits state is needed. A possible path will be denoted by $\omega \in \Omega$.

For taking advantage of the QAE speedup, the following steps needs to be executed[5, 6]:

1. Load the distribution of a grid with T time samples and $2^n - 1$ values into nT qubits:

$$\sum_{\omega=0}^{2^{nT-1}} \sqrt{p(\omega)} \left|\omega\right\rangle_{nT}, \tag{6}$$

where $p(\omega)$ is the probability of the path ω.

2. Apply a payoff operator analogously to $\mathcal{A}$ in Eq.(4):

$$\sum_{\omega=0}^{2^{nT-1}} \sqrt{p(\omega)} \left|\omega\right\rangle_{nT} \rightarrow \sum_{\omega=0}^{2^{nT-1}} \sqrt{p(\omega)(1-f(\omega))} \left|\omega\right\rangle_{nT} \left|0\right\rangle + \sum_{\omega=0}^{2^{nT-1}} \sqrt{p(\omega)f(\omega)} \left|\omega\right\rangle_{nT} \left|1\right\rangle. \tag{7}$$

In order to see the connection to Eq.(4) let us develop the following state function (which is part of the second term in Eq.(7):

$$\begin{aligned} &\sum_{\omega} \sqrt{p(\omega)f(\omega)} \left|\omega\right\rangle_{nT} = \\ &= \sqrt{\sum_{\omega} p(\omega)f(\omega)} \sum_{\omega} \frac{\sqrt{p(\omega)f(\omega)}}{\sqrt{\sum_{\omega} p(\omega)f(\omega)}} \left|\omega\right\rangle_{nT} \\ &= \sqrt{\hat{\mu}_f} \left|\psi_1\right\rangle_{nT}. \end{aligned} \tag{8}$$

So the final state after applying the payoff operator is:

$$\sqrt{1-\hat{\mu}_f} \left|\psi_0\right\rangle_{nT} \left|0\right\rangle_1 + \sqrt{\hat{\mu}_f} \left|\psi_1\right\rangle_{nT} \left|1\right\rangle_1. \tag{9}$$

3. Apply the QAE algorithm in order to calculate $\hat{\mu_f}$ an estimator for μ_f.

After understanding the main steps for achieving the quantum advantage, a rough estimation of when the quantum advantage starts to be applicable will be made. MC methods require the simulation of N paths in order to converge to the expectation value with an error of $\mathcal{O}(\frac{1}{\sqrt{N}})$,where each path takes T_{MC} to calculate. On the other hand, the QAE algorithm enables to converge to an error of $\mathcal{O}(\frac{1}{M})$ where one needs to apply the quantum operator $\mathcal{Q}$ M times. The time it takes to apply $\mathcal{Q}$ is T_{QAE}. Therefore, performing a rough estimation as a 'back on the envelope' calculation, for each M that satisfies $MT_{MC} > \sqrt{M}T_{QAE}$, that is

$$M > (\frac{T_{QAE}}{T_{MC}})^2, \tag{10}$$

there would be a quantum advantage that scales quadraticaly with M. Chakrabarti *et al.* [6] assessed the resources needed to take advantage of this quantum speed-up, and showed that today's quantum computers do not contain the required resources yet. The road maps of several companies developing quantum computer might reveal that this situation might change in the near future (seeIBM's Quantum Decade report,Google's declaration of the new Quantum AI campus).

The analysis that was just made regarding the quantum advantage assumes that there is an efficient way for loading classical distributions into a quantum state. This assumption might be made according to the theoretical work of Grover and Rudolph [10]. However, Chakrabarti *et al.* have showed that "the Grover-Rudolph methods are not applicable in practice" [6]. For this reason, finding efficient practical ways for loading distributions into quantum states is an important task. The usage of QGAN is one of the solutions for this task. Furthermore, as will be discussed below, for practical applications the QGAN is a very convenient solution.

III. THEORETICAL ANALYSIS

The goal of the QGAN is to load a desired classical probability function to a quantum state. First impressions of the task could be received by viewing the top row in Fig.1. The structure of the QGAN is very similar to the regular GAN model, and is depicted in Fig.2. A discriminator is fed alternately with real data samples and generated data samples, and its goal is to discriminate between the two. The goal of the generator is to learn the pattern of the real data, and to optimize its parameters so the generated data will be as close as possible to the real training data. However, in contrary to the classical GAN, the generator here is a quantum circuit that is governed by classical parameters. So the output of the generator is a quantum state.

A quantum state becomes a classical state when it is measured. Suppose the following quantum state:

$$\left|\psi\right\rangle = \sqrt{0.5}\left|0\right\rangle + \sqrt{0.25}\left|1\right\rangle + \sqrt{0.15}\left|2\right\rangle + \sqrt{0.1}\left|3\right\rangle. \tag{11}$$

After a measurement of $\left|\psi\right\rangle$, only one of the possible outcomes will be yielded $\{0, 1, 2, 3\}$. The corresponding probability for each measurement outcome is $[0.5, 0.25, 0.15, 0.1]$. Thus, in order to compare

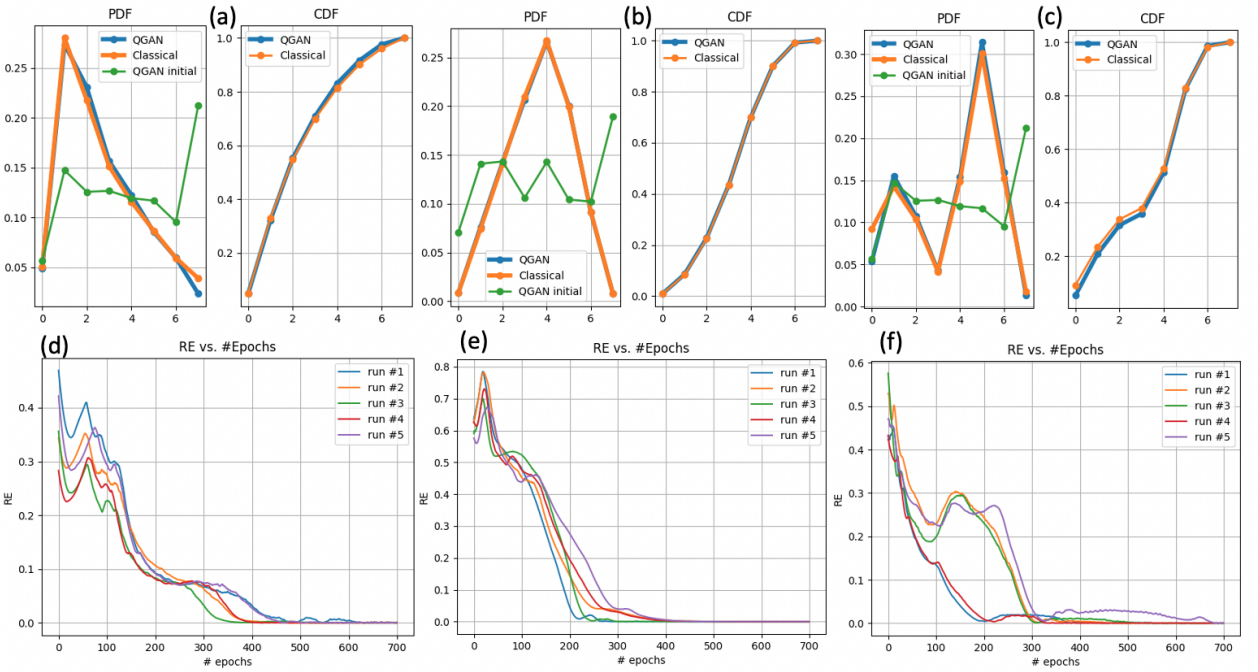

FIG. 1. The top row presents the real desired PDF (left) and CDF (right) and the generated PDF and CDF correspondingly for each distribution function. The initial generated probability is depicted in the left as well. The distribution functions presented are *Log-Normal* (a), *Triangular* (b) and *Bi-Modal* (c). The distribution's parameters are described in section IV. The bottom row represents the corresponding convergence graph for each distribution. Five runs of each training session are depicted. All the graphs depict the configuration of $K = 2$, three hidden layers with $[256, 512, 256]$ nodes.

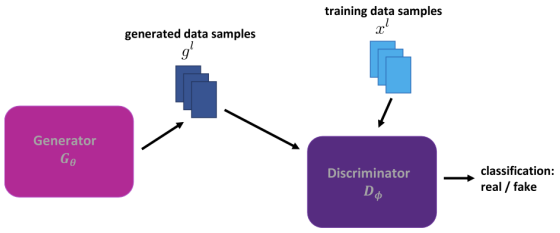

FIG. 2. The network architecture. The generator is a quantum circuit and several measurements of it yields the generated data samples g^l. The real (training) data samples, x^l, are drawn from the desired distribution. The discriminator is a classical neural network with *Sigmoid* activation function on the output layer, such that the output is a soft classification (a probability for for being real or generated). The figure is taken from [1].

between the quantum generator and the real data, measurement samples of the quantum generator, and samples drawn from the real probability distribution are fed into the discriminator.

The discriminator is a few layers neural network. Its input is one integer sample in the range $[0, 2^n - 1]$ (where n is the number of qubits). Its output layer contains one node which is activated with a *Sigmoid* function. The output of the discriminator is a number in the range $[0, 1]$, which represents the probability that the input sample belongs to the real data.

The generator is a quantum circuit that is composed with 1-qubit gates and two-qubits entangling gates. The structure of the generator is depicted in Fig.3. The 1-qubit gates are $R_y(\theta)$ which are rotations in θ radians around the $y-$axis of the Bloch sphere. The entangling gates are Controlled$-Z$ gates which gives a global phase of π if the two qubits are 1 ($Cz\,|10\rangle = |10\rangle\,; Cz\,|11\rangle = -\,|11\rangle$). The amount of gates is an hyper-parameter, but they constitute a specific structure. First, $\#n$ $R_y(\theta)$ gates are applied to all the qubits, this is the 0^{th} layer. Then, each layer is constituted from $\#n$ entangling gates such that each qubit is entangled to its nearest neighbors, and another $\#n$ rotations gates $R_y(\theta)$. The classical parameters governing the quantum generator are the rotations angles. They will be described by the parameter vector θ. In addition, at the beginning of the circuit an equal superposition is achieved by applying $\#n$ Hadamard gates (H). The amount of quantum gates is: #1-qubit gates$= n(K+1)$ and #2-qubits gates $= nK$.

In each training epoch, both the classical training probability and the quantum state $|g_\theta\rangle$ are sam-

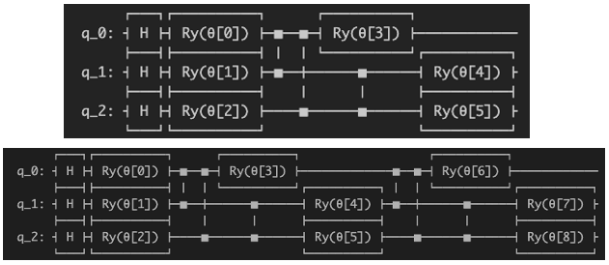

FIG. 3. The two configurations of the generator circuit implemented. The top figure depicts the configuration of $K = 1$, and the bottom circuit for $K = 2$. H stands for Hadamard gate and $R_y(\theta[j])$ are rotations around the $y-$axis by $\theta[j]$ radians, where $j \in 0, 1, ..., n(K+1)-1$. The filled dots connected by a dashed line are Controlled Z gates (Cz).

pled N times. These samples are then divided into batches of m samples such that every epoch includes $\lfloor \frac{N}{m} \rfloor$ batches. Each one of the m samples is an integer in the range $[0, 2^n - 1]$ (where n is the number of qubits). For example, for $n = 3$ qubits, and $N = 200, m = 10$, there would be 20 batches in each epoch. So, a possible generated vector could be $g = [0, 0, 7, 5, 3, 6, 2, 5, 1, 3]$, and a possible real data vector could be $x = [1, 0, 2, 5, 4, 1, 2, 3, 1, 2]$. A specific value from the two vectors will be described as g^l and x^l respectively, where $l \in \{1, 2, ..., m\}$.

For training the entire model the following steps are executed:

1. For each batch, calculate the discriminator's loss:

$$L_D(\boldsymbol{\phi}, \boldsymbol{\theta}) = \frac{1}{m} \sum_{l=1}^{m} logD_\phi(x^l) + log(1 - D_\phi(g^l)), \quad (12)$$

which could also be regarded as:

$$L_D(\boldsymbol{\phi}, \boldsymbol{\theta}) = \sum_{j=0}^{2^n-1} f_{\mathcal{R}}^j logD_\phi(j) + f_G^j log(1 - D_\phi(j)), \quad (13)$$

where $f_{\mathcal{R}}^j$ is the frequency that the state j appeared in this real batch, and f_G^j is the frequency that the state j appeared in this generated batch. At the limit of very big m, $f_{\mathcal{R}}^j \to p_{\mathcal{R}}^j$, and $f_G^j \to p_G^j$ where $p_{\mathcal{R}}^j$ and p_G^j are the real and generated probabilities respectively. The generated probability is the absolute square amplitudes of the generator's output state: $G_\theta |\psi_{in}\rangle = \sum_{j=0}^{2^n-1} \sqrt{p_\theta^j} |j\rangle = |g_\theta\rangle$ where $(p_\theta^j \equiv p_G^j)$.

2. . Update the discriminator's parameters ϕ according to $\nabla_\phi L_D(\phi, \theta)$ as in the classical case [11].

The essence of this stage is to find the parameters ϕ that minimize L_D. They minimize it by 'choosing' for each number $j \in \{0, 1, ..., 2^n - 1\}$ the optimal probability this number belongs to the real data, according to f_G^j and $f_{\mathcal{R}}^j$.

3. For each batch, calculate the generator's loss:

$$L_G(\boldsymbol{\phi}, \boldsymbol{\theta}) = -\frac{1}{m} \sum_{l=1}^{m} logD_\phi(g_l). \quad (14)$$

In the limit of big m, it could be regarded as:

$$L_G(\boldsymbol{\phi}, \boldsymbol{\theta}) = -\sum_{j=0}^{2^n-1} p_\theta^j logD_\phi(j). \quad (15)$$

4. Update the generator's parameters θ according to $\nabla_\theta L_G(\theta, \theta)$. Mitarai *et al.* [12] have showed that the gradient could be calculated as:

$$\frac{\partial p_\theta^j}{\theta^k} = \frac{p_{\theta^k + \frac{\pi}{2}}^j - p_{\theta^k - \frac{\pi}{2}}^j}{2}, \quad (16)$$

where $k = \{0, ..., (K+1)n - 1\}$. This is known as the "parameter shift rule" [13].

The essence of this stage is to update the parameters θ such that p_θ^j will minimize $L_G(\boldsymbol{\phi}, \boldsymbol{\theta})$ for a fixed $\boldsymbol{\phi}$.

Steps 1-4 are repeated for all the batches in each epoch.

Finally, for evaluating the results, the natural statistical measure for comparing the resemblance between two distributions is the relative entropy (RE) (also known as the Kullback-Leibler divergence). The RE between a generated distribution p_G and a real distribution $p_{\mathcal{R}}$ is:

$$RE(p_G || p_{\mathcal{R}}) = \sum_{j=0}^{2^n-1} p_G(j) log_2(\frac{p_G(j)}{p_{\mathcal{R}}(j)}). \quad (17)$$

IV. METHODS

The project is divided into two steps. In the first step I have tried to restore the results from the original paper [1]. In the second step, I have tried to improve the results by adapting the discriminator while keeping the quantum generator unchanged. The reason for this is to show the pure classical advantage one could achieve by optimizing the discriminator network to fit the QGAN model. In addition, the original paper presented several configuration of the quantum generator. For the same reason mentioned above, I chose only one quantum configuration to

Distribution	K	$\mu^M_{RE_{min}}$	$\mu^M_{RE[-1]}$	μ^P_{RE}	$[\frac{\mu^P_{RE}}{\mu^M_{RE_{min}}}$ - $\frac{\mu^P_{RE}}{\mu^M_{RE[-1]}}]$
Log-Normal	1	0.0333	0.0726	0.454	[1.36 - 0.63]
	2	0.0809	0.1311	0.0739	[0.91 - 0.56]
Triangular	1	0.2454	0.2855	0.0624	[0.25 - 0.22]
	2	0.0808	0.1498	0.0091	[0.11 - 0.06]
Bi-modal	1	0.0071	0.0571	0.3524	[49.85 - 6.17]
	2	0.2281	0.3722	0.0192	[0.08 - 0.05]

TABLE I. A summary of the restored results. K is the depth of the generator's circuit. $\mu^M_{RE_{min}}$ is the average of 5 runs of the minimum RE (as a function of epochs) established in this configuration. $\mu^M_{RE[-1]}$ is the average of 5 runs of the last RE (as a function of epochs) established in this configuration. μ^P_{RE} is the paper averaged RE acquired in this configuration. The last column is the range of improvement factor - by how much the paper's average RE is greater than the averaged restored REs.

work with. The *Qiskit Quantum Machine Learning* package was the main package in use. It contains an interface between IBM's quantum computing package *Qiskit*, and the classical machine learning package *Pytorch*. The original paper had a supplementary git tutorial which did not include this interface (see Apendix.A), but only a simple implementation of a basic neural network in *Numpy*.

For restoring the original paper's results I assembled a discriminator with two hidden layers. The first layer contained 50 nodes and the second layer contained 20 nodes. The input and output layers contain 1 node each as described above. Both hidden layers were fully connected layers that were activated with *Relu* function with slope 0.2. The discriminator was optimized with the AMSGrad version of the ADAM optimizer. The initial learning rate was 10^{-4} and the momentum terms were the default terms $\beta_1 = 0.9, \beta_2 = 0.999$. The initial parameters were chosen randomly form a uniform distribution in the range $[-1, 1]$ (according to the default *nn.Linear* module). It is important to notice that some of these parameters were not explicitly stated by the original paper's authors so there was no way to replicate their results perfectly. For evaluating the restored results, the statistics of 5 runs were gathered, were each run's goal is to upload the same desired distribution into a 3-qubit state. In each run, $N = 5000$ samples were generated from the desired classical probability (with *Numpy.Random* package), and then rounded and truncated to the range $[0, 7]$. The batch size was 1000, and due to the truncation there were 4 batches in each epoch. The number of epochs applied were 700.

There were three distribution functions examined. *Log-Normal* with $\mu = 1, \sigma = 1$, *triangular* with left edge at 0, right edge at 7 and pick at 4, and a *Bi-Modal* normal distribution with $\mu_1 = 1, \sigma_1 = 1, c_1 = 0.4$ and $\mu_1 = 5, \sigma_2 = 0.8, c_2 = 0.6$. The generator's initial parameters were chosen randomly from a normal distribution and were multiplied by 10^{-2}. The number of layers in the generator circuit K, was scanned for $K = 1, 2$.

For the improvement stage of the model several avenues were examined. First, different amount of hidden layer's nodes were examined. Secondly, an application of dropouts were examined. And third, the addition of another hidden layer was tested. Different configurations of those three features were experimented. Due to the low number of qubits under examination, the usage of real quantum computer was not necessary. Instead, all the experiments were executed on a quantum simulator as part of the *Qiskit* package. The results were compared with the corresponding results in the paper that were executed with a quantum simulator as well.

V. RESULTS

The results for the restoring part are summarized in Table.I. The goal of this part was to get a rough estimation if the original paper's results could be restored. In the original paper, they have trained their models 10 times with 2000 epochs. Due to limited resources, and due to the fact that this part of the project was in a 'proof-of-concept' attitude, only 700 epochs were applied and for 5 times. An early-stop feature was not configured, hence I have compared both the averaged minimum RE ($\mu^M_{RE_{min}}$)and the averaged last RE ($\mu^M_{RE[-1]}$) to the original papers

Distribution	K	*#Layers*	μ_{RE}^{M}	σ_{RE}^{M}	μ_{RE}^{P}	σ_{RE}^{P}	$[\frac{\mu_{RE}^{P}}{\mu_{RE}^{M}}]$
Log-Normal	1	2	0.0252	0.0175	0.0454	0.0856	2
		3	0.0048	0.0022			10
	2	2	0.0042	0.0048	0.0739	0.051	18
		3	0.0007	0.0006			102
Triangular	1	2	0.0053	0.0016	0.0624	0.0535	12
		3	0.0025	0.0009			25
	2	2	0.0049	0.0053	0.0091	0.0042	2
		3	0.0001	0.0000			63
Bi-Modal	1	2	0.0023	0.0009	0.3524	0.0146	153
		3	0.0016	0.0007			222
	2	2	0.0147	0.0107	0.0192	0.0252	1
		3	0.0009	0.0004			22

TABLE II. A summary of the results of the improvement stage. The first three columns present the configuration examined with K the generator's depth and #Layers the amount of hidden layers. The forth and fifth columns depict the averaged RE and its standard deviation among the five runs (the RE taken was the RE of the last epoch). Columns six and seven show the original paper's average RE and its standard deviation respectively. Column eight summarizes the improvement factor for each configuration. See further elaboration in section V.

averaged RE μ_{RE}^{P}. The subscripts M, P represent My results and the Paper's results respectively. The last column represent the range of improvement factor - how much the paper's RE is bigger than the restored results.

One can see that the improvement factor is approximately in the range of 1 with the only difference for the *Bi-Modal* distribution that is one order of magnitude higher and lower for $K = 1, 2$ respectively. When these results are viewed with a 'proof-of-concept' attitude, one may deduce that the concept was proven, and can move on to the next step. (Another thing to state here, is that in the original paper there was no explicit equation for the RE measure, and in the corresponding tutorial, the logarithm base in the RE formula was taken as e rather than 2. Thus, if this is the actual case, the improvement factor should be greater by factor of $\frac{1}{ln(2)} \approx 1.44$.)

The results for the improvement part are summarized in Table.II. The last column represents the improvement factor and one can easily verify that for each distribution and each generator's depth K an improvement of at least 10 times was achieved with approximately one third of the epochs used in the paper.

Before I will get into details, one may appreciate the results qualitatively by viewing Fig.1. The figure depicts the distribution and the convergence graphs for the three distributions with the configuration of $K = 2$ and 3 hidden layers. One can immediately appreciate the resemblance of the final QGAN distribution to the classical one in all three cases. In addition, and equally important, is the relative small variance of the convergence graph among the 5 runs. This point would be further discussed in section VI.

The chosen amounts of nodes for the hidden layers are $[512, 256]$ and $[256, 512, 256]$ for the 2 and 3 hidden layers configurations respectively. This number was chosen by trial and error method. An addition of dropouts layer was examined as well, but failed to improve the results in any configuration.

The first three columns in Table.II presents the configuration examined. The forth and fifth columns depict the averaged RE and its standard deviation among the five runs. The RE taken was the RE of the last epoch, and as can be viewed from Fig.1(d-f), the minimum RE was approximately the last RE value. Columns six and seven show the original paper's results. Column eight summarizes the improve-

ment factor for each configuration. For all the non-discriminator configurations (distribution $\&K$), the 3-hidden layer configuration yielded the best results in a substantial manner.

VI. DISCUSSION

In this work, I have improved Zoufal et al. work [1] for loading classical distributions into a quantum state efficiently. The improvement was achieved without adding any quantum resources (e.g. quantum gates) and by only adapting the classical discriminator. The added complexity of the discriminator is negligible in terms of computational resources compared to the classical computational resources utilized in a typical quantum computing setup.

The improvement of the results are manifested twice fold. First, in the obvious statistical measure of the average RE as discussed in section V. Moreover, the variance of the RE across the different runs improved in two order of magnitudes for the case of three layers for all configurations. Thus, this model is much more robust then the one presented in the original paper. One may view the robustness of the model as having the same roll as resilience to over-fitting. In the task at hand (loading distributions), each model is trained from scratch for a specific distribution. Hence, there is no actual over-fitting issue in this task, since the real input to the discriminator is almost identical each time, and it should be this way. Nonetheless, one would desire the same model to fit different distributions with different training each time. In this paper I have showed precisely that. For the case of 3 qubits, the presented discriminator architecture is suitable for a variety of distributions, and in the several cases presented in the paper, the RE statistical measure, converged smoothly in the range of 700 epochs (see Fig.1).

Once the quantum computing ecosystem will achieve substantial amount of practical applications, it would be clearer what sort of use cases would be needed for the task of loading classical distributions into quantum states. Further improvements that could be achieved in the meantime are expansion of the topology to the amount of qubits that are currently running on a quantum computer (up to tens of qubits). It is interesting to explore if the suggested discriminator architecture is suitable for these cases as well. In addition, further improvements of the quantum parts of the QGAN could be made.

Appendix A: Code Availability

The code for this project is available in Github (*edenico76/QML_project*). Please follow the ReadMe file in the folder. A tutorial made by Zoufal *et al.* [1] is available in Github as well (link).

[1] C. Zoufal, A. Lucchi, and S. Woerner, Quantum generative adversarial networks for learning and loading random distributions, npj Quantum Information **5**, 10.1038/s41534-019-0223-2 (2019).

[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, 2009).

[3] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM Journal on Computing **26**, 1484 (1997).

[4] M. Schuld, I. Sinayskiy, and F. Petruccione, An introduction to quantum machine learning, Contemporary Physics **56**, 172 (2014).

[5] N. Stamatopoulos, D. J. Egger, Y. Sun, C. Zoufal, R. Iten, N. Shen, and S. Woerner, Option pricing using quantum computers, Quantum **4**, 291 (2020).

[6] S. Chakrabarti, R. Krishnakumar, G. Mazzola, N. Stamatopoulos, S. Woerner, and W. J. Zeng, A threshold for quantum advantage in derivative pricing, Quantum **5**, 463 (2021).

[7] P. P. Boyle, Options: A Monte Carlo approach, Journal of Financial Economics **4**, 323 (1977).

[8] F. Black and M. Scholes, The pricing of options and corporate liabilities, Journal of Political Economy **81**, 637 (1973).

[9] G. Brassard, P. Høyer, M. Mosca, and A. Tapp, Quantum amplitude amplification and estimation (2002).

[10] L. Grover and T. Rudolph, Creating superpositions that correspond to efficiently integrable probability distributions, (2002).

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial nets, Advances in neural information processing systems **27** (2014).

[12] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, Quantum circuit learning, Physical Review A **98**, 10.1103/physreva.98.032309 (2018).

[13] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, Evaluating analytic gradients on quantum hardware, Physical Review A **99**, 10.1103/physreva.99.032331 (2019).