DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Quantum Computing

Quantum and Classical Generative Modeling for Quantum States Preparation

Wiktor Jurasz

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Quantum Computing

Quantum and Classical Generative Modeling for Quantum States Preparation

Quantenbasierte und klassische generative Modellierung zur Erzeugung von Quantenzuständen

Author: Wiktor Jurasz
Supervisor: Prof. Dr. Christian Mendl
Submission Date: 15.07.2021

I confirm that this master's thesis in quantum computing is my own work and I have documented all sources and material used.

Munich, 15.07.2021 Wiktor Jurasz

Abstract

Kurzfassung

Contents

1. Introduction

1.1. Problem Statement

Generative Modeling aims to learn a conditional probability $P(X|Z = z)$, where X is some observable variable and Z is a target variable. With knowledge of this conditional probability, it is possible to generate new observations $\bar{x} \in X$. In general case, one would not try to obtain the probability $P(X|Z)$ exactly, but learn an approximation. To do so a set of samples $x \in X$ is necessary to train a generator function $G : Z \to X$ which given a target variable $z \in Z$ generates new observation $x \in X$.

In the generative framework, the variable X is a multidimensional vector, in particular it can be used to describe an arbitrary quantum state. With this setup, given a finite set of quantum states $\mathcal{Q} = \{|\psi_i\rangle\}, |\psi_i\rangle \in X \forall i$ the generator function G prepares a new quantum state $|\hat{\psi}\rangle$. This new quantum state is expected to come from the same distribution as the samples in the input set $\mathcal{Q}$.

The only missing piece in the above description is the target variable Z. In the context of the function G, generating the quantum states, we can think about Z as a label of the generated state. That is, for a specific $z \in Z$ the function G always generates the same $|\hat{\psi}\rangle$.

In this work we evaluate different approaches to find the probability $P(X|Z = z)$ by learning the function G. We also address the limitations of the existing methods propose a new one that combines quantum and classical generative modeling.

1.2. Previous Work

There exist many different types of generative models. In this work we focus on one particular type, namely Generative Adversarial Networks (GANs). First version of GANs was proposed bu Goodfellow et al. [1] (to which we refer as Standard GANs - SGANs), since then many different variations of GANs were invented [2][3][4]. In context of this work, particularly interesting are Wasserstein GANs (WGANs)[5] which minimize *Earth-Mover* distance between two probability distribution (see Chapter 3) instead of *Jensen–Shannon* divergence (see Chapter 3) as in SGANs.

In recent years there has been an increasing interest in realizing Generative Adversarial Networks in Quantum Computing (QC) realm. Dallaire-Demers et al. proposed QuGANs [6] - Quantum Generative Adversarial Networks where generator and discriminator are parametrized quantum circuits. Similarly Benedetti et al. proposed fully quantum GANs for pure state approximation [7], but with different (more suitable for NISQ [8]) learning method. Hybrid methods were also explored, Zoufal et al. build qGAN [9] - with parametrized

quantum circuit as the generator and classical neural network as the discriminator.

De Palma et al. proposed quantum equivalent of Wasserstein distance of order 1 [10] which made the Quantum Wasserstein GANs (QWGANs) [11] possible. This variation of quantum GANs consist of the parametrized quantum circuit as the generator and the classical linear program as the discriminator.

1.3. Our Contribution

There has been a substantial effort in the direction of bringing GANs into the quantum realm. Nevertheless, this is still very early stage and many more routs are yet to be explored. In this work we focus on building quantum GANs that can generate new, unseen before, quantum states. Majority of models proposed so far are only able to generate the states the has been a part of the training data. Only some architectures [6] account for random noise in the input that allows to generate unseen states. However, as we discuss later, those are mostly theoretical and do not seem to work well in practice.

We propose a new quantum-classical hybrid approach that allows to generate an unlimited number of unseen quantum states. We utilize the fully quantum and fully classical generative models that work together in one framework.

We proceed as follows. In Chapter 2 we introduce briefly the quantum computing concepts necessary to understand the problems we are solving. We also establish the quantum computing notation used in the reminder of this work. In Chapter 3 we give a general introduction to classical GANs. In Chapter 4 we combine the knowledge from the previous chapters to introduce the concept of quantum GANs and talk more about the different variations of quantum GANs and their limitations. In Chapter 5 we introduce and describe in depth about our concept of hybrid quantum-classical generative framework. In Chapter **??** we and analyse numerous experiments with the proposed framework. We empirically prove the quality of the states generated by the proposed framework. Finally, in Chapter 6 we conclude our finding and talk briefly about the possible future directions.

2. Quantum Computing Notation

In this chapter we provide a very brief introduction to the key concepts of quantum computing and introduce the notation used in the rest of this paper.

Any quantum state can be represented in some complex Hilbert space, in this work we consider only finitely dimensional spaces $\mathbb{C}^N$. We denote the quantum state using bra-ket notation, with $|\cdot\rangle \in \mathbb{C}^N$ column vector and $\langle\cdot| \in (\mathbb{C}^N)^\dagger$ row vector and complex conjugate of $|\cdot\rangle$.

2.1. Parametric Circuits

3. Generative Adversarial Networks Introduction

Generative Adversarial Network (GAN)[1] is a machine learning framework designed to estimate generative models using adversarial process. At the core it consists of two models: the generative one G which is capable of learning the distribution of the provided data and discriminative model D that, given a data point, estimates whether it comes from input data or was generated by G. The models are set to compete with each other in a minmax game. D is trained to maximize the probability of correctly distinguishing between the generated and real samples, while G is trained to minimize it.

To approximate the true distribution p_r over data X we define a prior noise distribution p_z over noise input Z and the generator distribution p_g over data X. The generator represents a mapping $G(z \sim Z; \theta_g) \rightarrow x \sim X$, where θ_g is some learnable parameter (summary in Table 3.1).

The discriminator $D(x, \theta_d) \rightarrow [0; 1]$ is a function that given a sample $x \sim X$ outputs the probability whether x comes from data or was generated by G.

In classical GANs both G and D are most often modeled as multi-layer perceptrons[1] or other types of neural networks (e.g. convolutional neural networks [4]). However, there exist many different variations of cost functions used during the mixmax training procedure. In the following paragraphs we take a closer look at two of them, namely SGANs and WGANs, which are the most relevant in the context of this work. To simply the notation, we skip the θ parameters, i.e. $G(z, \theta_g) = G(z)$ and $D(x, \theta_d) = D(x)$

3.1. Standard Generative Adversarial Networks (SGANs)

The goal of D is to distinguish between the real and generated samples. For $x \sim X$ the output $D(x)$ should approach 1, while for $x \sim G(z)$ it should approach 0. In other terms it simultaneously maximizes $\mathbb{E}_{x \sim p_r(x)}[\log D(x)]$, and $\mathbb{E}_{z \sim p_z(z)}[\log 1 - D(G(z))]$.

Generator G has an opposite objective, thus it minimizes the $\mathbb{E}_{z \sim p_z(z)}[\log 1 - D(G(z))]$.

Probability Distribution	Description
p_z	True distribution over noise input Z
p_g	Approximated distribution over input data X given by G
p_r	True distribution over input data X

Table 3.1.: Description of the probability distributions used in GANs

Putting it all together, we get the objective for SGANs 3.1.

$$\begin{aligned}\min_G \max_D \mathcal{L}(\mathcal{G}, \mathcal{D}) &= \min_G \max_D \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log 1 - D(G(z))] \\ &= \min_G \max_D \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}[\log 1 - D(x)]\end{aligned} \tag{3.1}$$

It can be shown, that if D is optimal, this loss function measures the similarity between p_r and p_g according to Jensen–Shannon divergence (see Appendix A for detailed analysis).

SGANs are a very powerful tool, however, the training process is very difficult and unstable [12]. Additionally, if the data exists in low dimensional manifold (which seems to be the case for most real world data [13]), p_r and p_g are very likely disjoint and we are always capable of finding the perfect discriminator [14]. This might seem like a good characteristic, but if we examine the loss function closer, we find that it makes generator incapable of learning. For the perfect D we have $\forall x \sim X, D(x) = 1$ and $\forall z \sim Z, D(G(z)) = 0$. For those values $\mathcal{L}(\mathcal{G}, \mathcal{D}) = 0$ gradient vanishes and the gradient based optimizers cannot improve the generator anymore.

3.2. Wasserstein Generative Adversarial Networks (WGANs)

One of the very prominent improvement to SGANs and a way to mitigate the vanishing gradient problem is changing the cost function to use Wasserstein Distance.

The Wasserstein Distance, also called Earth-Mover's Distance is a distance measure between two probability distributions. The name "Earth Mover" comes from the fact, that informally the distance can be described as follows: Given two probability distributions, if we imagine them as piles of dirt, the "Earth Mover" distance says what is the minimum cost of turning one pile of dirt into the other one. Where cost is the volume that has to be moved times the distance is has to be moved.

The main advantage of the Wasserstein Distance over Jensen–Shannon Divergence is, that even in the case of disjoint distribution we get meaningful, stable distance which is well suited for gradient based learning.

Formally, the Wasserstein or Earth-Mover's Distance (EM Distance) is defined as follows:

$$W(p_r, p_g) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{3.2}$$

Where $\Pi(p_r, p_g)$ denotes the set of all possible joint distributions, $\gamma(x, y)$ says how much "dirt" has to be transported in order to transform p_g into p_r. Since we take inf, the EM Distance is then the cost of such transport done in the optimal way.

The computation in 3.2 is highly intractable, but according to Kantorovich-Rubinstein duality [15], we can rewrite it as follows.

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L < K} \mathbb{E}_{x\sim p_r}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)] \tag{3.3}$$

Where the supremum is overall all K-Lipschitz functions f. In practice it is impossible to go over all such functions. Instead we define a family of parameterized functions $\{f_w\}_{w\in W}$ and using the same notation for the generator as in SGANs the WGANs objective becomes

$$\max_{f} \min_{G} \mathcal{L}(f, G) = \max_{f} \min_{G} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_z}[f_w(G_\theta(z))] \tag{3.4}$$

.

Here the function f_w represents the "discriminator", however it does not anymore tries to distinguish between real and fake samples. Instead it is trained to to approximate K-continous Lipschitz function and compute the Wasserstein distance. The one missing part from the Equation 3.4 is ensuring the f_w Lipschitz continuity, this can be achieved by gradient clipping [5] or gradient penalty [16].

4. Quantum Generative Adversarial Networks

The field of Quantum Machine Learning (QML) is still in very early stages and there has been an ongoing effort on translating the classical Machine Learning (ML) concepts into QML realm. Because of the limitations of current quantum computers (NISQ) [17] and overall different paradigm of Quantum Computing (QC), this process is difficult and there is no clear answer to the question "how this concept should be realized on QC device?". While in classical GANs generator and discriminator are realized as deep neural networks, NISQ devices are not yet powerful enough to support such architecture. Instead parametric quantum circuits [18] are used.

In this chapter we take a closer look at two different designs of quantum GANs. We briefly explain theory behind them and show the results of our evaluation. The quantum GANs introduced in this chapter are the base of our hybrid classical-quantum generative framework.

4.1. Standard Quantum GANs (SQGANs)

In the classical GANs, both the discriminator and generator are deep, general purpose neural networks. The logical extension of this design in the quantum realm is to model those are general purpose parametric quantum circuits. This idea was formalized by Dallaire-Demers et al. [6] in the architecture we call SQGANs (Figure 4.1). The generator starts in the state $|0\rangle^{\otimes n}$ in the **Our R/G** wire, where n is the dimension of the real quantum samples. Additionally, the generator takes the label state $|\lambda\rangle$ in the **Label R/G** wire and the random state $|z\rangle$ that provides the entropy in the **Batch R/G** wire. The label state $|\lambda\rangle$ lets the generator to learn the conditional distribution $p_g(x|\lambda)$ instead of $p_g(x)$, this design was inspired by classical Conditional GANs [2]. The discriminator takes the generated state ρ_λ^G or real state ρ_λ^R and the corresponding label $|\lambda\rangle$ in the **Label D** wire, it also uses the workspace **Batch D** initialized in $|0\rangle$ state. The measurement on the single qubit wire **Out D** corresponds to the probability of the state in **Out R/G** being real or generated.

There are not any particular restrains on how G and D circuits should look like. However, to be able to learn and differentiate between arbitrary quantum states the ansatz used should be universal (i.e. be able to generate every quantum state given appropriate depth). The ansatz used in this work is described in details in Appendix B.1.

4.1.1. Training

As in the SGANs we are interested in the minmax game setting. Specifically, in the **Out D** wire, the measurement of $|1\rangle$ indicates that the sample was real and $|0\rangle$ that the state was

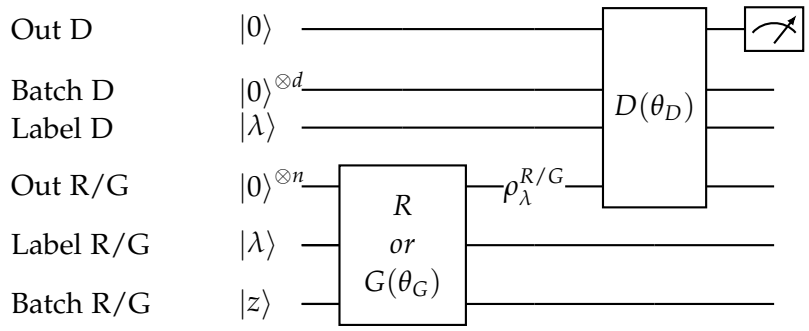


Figure 4.1.: SQGANs schema. The discriminator D and the generator G are parametric quantum circuits. The first 3 wires go directly to the discriminator. **Out D** outputs the probability of the input being generated. **Batch D** is an additional workspace of the discriminator and **Label D** contains the label state. **Out R/G** carries the generated or real state. **Label R/G** contains the label state and **Batch R/G** is the noise source for the generator, not used with the real samples.

generated by G. [1] This should be the case for each label state $|\lambda\rangle$, which gives the objective 4.1.

$$\begin{aligned} &\max_D \min_G \mathcal{L}(G, D) = \\ &\max_D \min_G \frac{1}{\Lambda} \sum_{\lambda \in \Lambda} P((D(\theta_D, |\lambda\rangle, R(\lambda)) = |1\rangle) \wedge (D(\theta_D, |\lambda\rangle, G(\theta_G, |\lambda\rangle, |z\rangle) = |0\rangle))) \end{aligned} \tag{4.1}$$

Where the discriminator objective is to maximize the probability of measuring $|1\rangle$ given real sample and measuring $|0\rangle$ given generated state. At the same time for the generator the objective is to do the opposite.

The SQGANs cost function 4.1 in contrast to the SGANs cost function 3.1 is not defined with log-likelihood. In quantum setup is more natural to work with linear functions and since the log is convex, the optimum is the same for both.

The cost function $\mathcal{L}(G, D)$ expressed in terms of measurements and assuming equal probability of sampling from R and G takes the form **??**[6].

$$\mathcal{L}(G, D) = \frac{1}{2} + \frac{1}{4\Lambda} \sum_{\lambda \in \Lambda} tr((\rho_\lambda^{DR}(\theta_D) - \rho_\lambda^{DG}(\theta_D, \theta_G, z))Z) \tag{4.2}$$

Gradient Estimation

To optimize the parameters θ_D and θ_G classical gradient descent method was used. First, the value of the cost function $\mathcal{L}(G, D)$ was estimated by sampling from the circuit 4.1. This allows to calculate the gradient w.r.t. θ_D and θ_G on classical computer and update the parameters at

[1] Using $|1\rangle$ and $|0\rangle$ in this order is just a convention we assume. Any other orthogonal pair can be used.

the step k in the following way

$$\begin{aligned} \theta_D^{k+1} &= \theta_D^k + \alpha_D^k \nabla_{\theta_D} \mathcal{L}(\theta_G^k, \theta_D^k) \\ \theta_G^{k+1} &= \theta_G^k - \alpha_G^k \nabla_{\theta_G} \mathcal{L}(\theta_G^k, \theta_D^k) \end{aligned} \tag{4.3}$$

where α_D^k and α_G^k are learning rate metaparameters that can depend on the step k.

It is also possible to estimate the gradient directly on quantum computer by creating an explicit quantum circuit for each element of vectors θ_D/θ_G and reading the grading by sampling for those circuits [6].

4.1.2. Evaluation Results

Experimental Setup

In all of the experiments the real samples are generated by evaluating the circuit from Figure 4.2. This circuit was constructed by Smith et al. [19] to study topological phase transitions. All the gates in the circuit are parameterized by a single real valued parameter $g \in [-1; 1]$. The detailed gates layout is described in Appendix B.2.

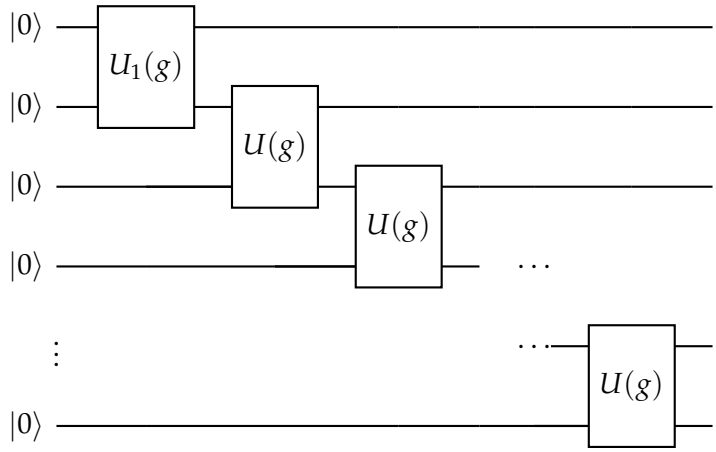


Figure 4.2.: The circuit used for generating real samples. All the gates are parametrized by a real valued parameter $g \in [-1; 1]$. For detailed description of the circuit see Appendix B.2

The generator and discriminator are both build using the generic circuit architecture from Appendix B.1. The number of layers differs and is specified for each experiment separately.

In all the experiment we use Adam optimizer [20] with parameters $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\hat{\epsilon} = 1e - 9$. The learning rate is calculated as

$$lr = \max\left(\exp\left(-\frac{(k + 200) * \ln 100}{4000}\right), 0.01\right) \tag{4.4}$$

where k is the optimization step number. The learning rate decreases from ~ 0.8 to 0.01 in the first 3800 steps and then remains at 0.01 for the rest of the training. The exact values were derived experimentally and show overall good convergence. The exact number of epochs and iterations is specified for each experiment separately.

Results For Pure State Real Input

In this setup the generator is fed the same real input state at every iteration. The real input state is generated using circuit from Figure 4.2 for random $g \in [-1; 1]$.

In Figure 4.3 we present the results of experiments for different widths of the real input circuit. For All the experiments, the generator and discriminator were built using ansatz from Appendix B.1 for each layer. The **Label** and **Batch** wires are not used for any circuit.

For each real input width the generator was able to approximate the real source, increasing the fidelity as the training progressed. The training was harder for the bigger real source circuits and it can be clearly seen that the results for wider inputs are worse. Not only the final fidelity is lower, but also it fluctuates much more. It is also much harder for the generator to fool the discriminator for wider inputs.

It is not surprising that it is more difficult to train circuits with more qubits. We also acknowledge that with different metaparameters or optimization method the results could be better. However, our best results for more than 5 qubits in the real source are not longer useful for the real source approximation.

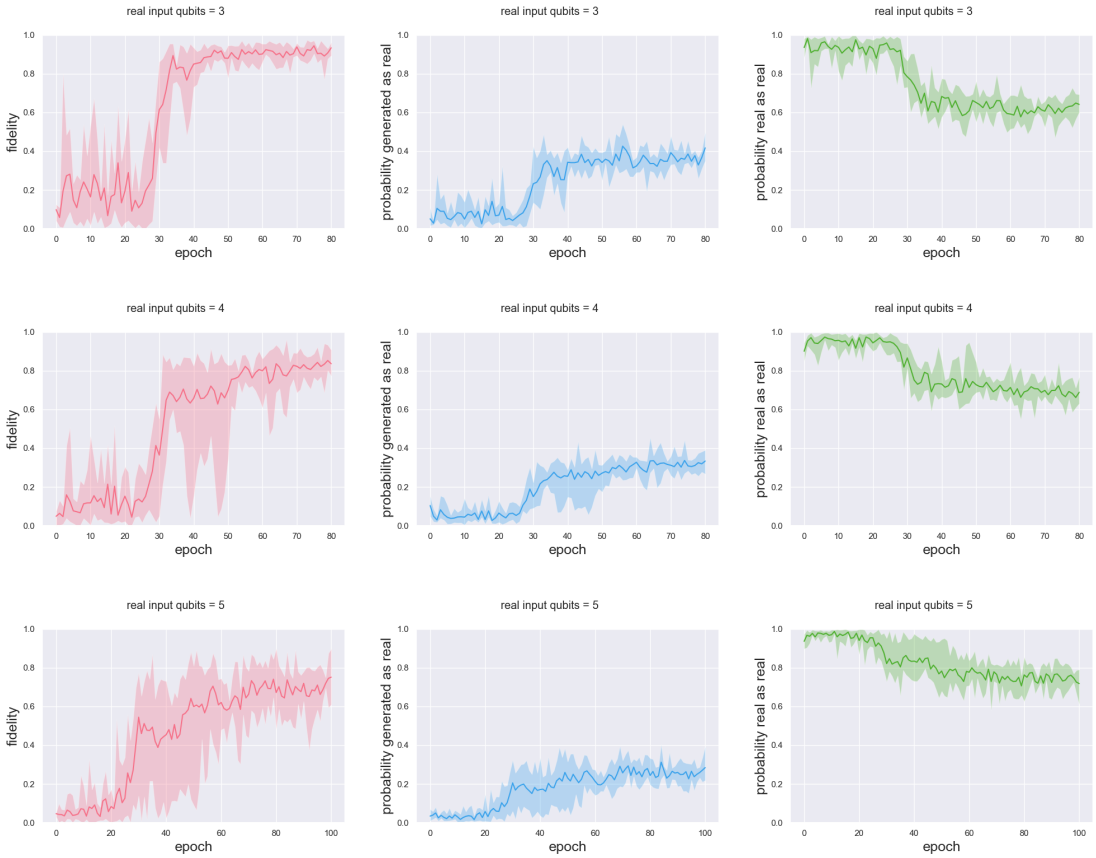
Results For Mixed State Real Input

If the real input source can provide more than one state, we can say that the generator is learning mixed state of all the input states. Dallaire-Demers et al. [6] trained a simple 2 qubits circuit with two real input states $|0\rangle$ or $|1\rangle$. In this setup **Label** wires consist of one qubit and also takes values $|0\rangle$ or $|1\rangle$, effectively learning $CNOT$ gate. This strategy do not have generalize well in to more complex cases. First, if there is more than one non-$|0\rangle$ input state, there would have to be a separate $CNOT$ gate for each state. Second, the number of qubits necessary for the **Label** wires grows logarithmically with the number of real input states. Both of those cause the circuit to grow in width and depth.

Another approach we explored was to insert $R_x(\lambda_k)$ parametrized rotation gates between each layer of the generator and discriminator. The generator is essentially learning how to rotate input state $|0\rangle$ to some desired real input state $|r_k\rangle$. Then, if for each k the rotation is pushed in different direction, the final generator might give different output state for each λ_k. However, in our experiments the generator was always learning to ignore the $R_x(\lambda_k)$ and in the end was only able to produce one, slightly distorted, state independent of λ_k.

4.1.3. Conclusions

We have experimentally evaluated SQGANs and confirmed their ability to learn pure quantum state. However, the problems with labeled states and small circuit width for which the training



Fidelity between real and generated state

Probability of classifying generated state as real

Probability of classifying real state as real

Figure 4.3.: The solid line represents the average value and the shaded area represents the range from 5 different experiments. The generator has the number of layers equal to the width of real input the discriminator has one more layer than the generator. In each epoch generator is optimized for 11 iterations and discriminator for 111.

succeeds prevented us from succeeding in mixed state learning.

The very important aspect of generative models is ability to generate new, unseen before, samples. In SQGANs this is theoretically possible by using the textbfBatch register. However, it is not straight forward to use this register in the intended way. First, the task of generating random quantum state is not trivial on its own. Second, all the problems associated with the label state λ also apply to the random state z.

Because of the above, we decided not to pursue the exploration of SQGANs further and turned into other methods.

4.2. Wasserstein Quantum GANs (WQGANs)

As in the classical case, the WQGANs relay on calculating the Wasserstein distance between real and generated state. However, the definition of the Wasserstein distance cannot be trivially translated into the quantum setup. There has been multiple approaches to defining the quantum Wasserstein distance [21, 22, 23, 24, 25, 26, 27, 28], however the first take on the Wasserstein distance in the context of quantum GANs was proposed by Chakrabarti et. al[29]. They proposed the Wasserstein semi-metric and successfully used it to train the quantum GANs. The shortcoming of this semi-metric is that, it does not preserve the triangle inequality.

4.2.1. Quantum Wasserstein Distance

In this work, we use the definition by De Palma et. al[10]. They proposed a generalization of Wasserstein distance of order 1 into the quantum realm. The dual formulation of this distance is stated as follows:

$$W_1(\rho,\sigma) = \max_{H\in\mathcal{O}_n} (Tr[(\rho-\sigma)H] : \|H\|_L \leq 1). \tag{4.5}$$

Where $\mathcal{O}_n$ is a set of all $2^n \times 2^n$ Hermitian matrices and $\|H\|_L$ is the quantum Lipschitz constant of the matrix H defined as:

$$\|H\|_L = 2 \max_{i=1\ldots n} \min_{H_{\bar{i}}\in\mathcal{O}_n} (\|H - H_{\bar{i}}\|_\infty). \tag{4.6}$$

Where $H_{\bar{i}}$ is a Hermitian matrix that does not act on the i-th qubit. The quantum Lipschitz constant and the Wasserstein distance defined in this way, recover their classical counterparts for operators diagonal in the canonical basis.

Definition 4.2.1 (Neighboring States)**.** Two quantum states ρ and σ are Neighbouring States if they coincide after discarding one qubit, i.e. $\exists i : Tr_i[\rho] = Tr_i[\sigma]$

Informally, the W_1 distance is the maximum distance that is induced by a norm that assigns the distance at most one to any couple of Neighbouring States.

The quantum Wasserstein distance has several properties that make it particularly useful in the context of training the generative models.

1. It is invariant with respect to qubit permutations and super additive with respect to tensor product, (i.e. $W_1(\rho,\sigma) \geq W_1(\rho_{1\ldots m},\sigma_{1\ldots m}) + W_1(\rho_{1+m\ldots n},\sigma_{1+m\ldots n})$).

   This property implies, that an operation that reduces distance between some marginal states also reduces the distance between the full states. For example, $W_1(|100\rangle, |111\rangle) > W_1(|110\rangle, |111\rangle)$, this is not the case however for the trace distance or fidelity, as they are always 0 between orthogonal states.

2. The quantum Wasserstein distance is bound by the trace distance, i.e. $T(\rho,\sigma) \leq W_1(\rho,\sigma) \leq nT(\rho,\sigma)$, where n is the number of qubits. This ensures that minimizing the W_1 distance also minimizes the trace distance.

3. Because the quantum Wasserstein distance recovers classical Wasserstein distance for diagonal operators, we can expect that generative models build using this metrics preserve the advantages of their classical counterparts.

4.2.2. WQGANs Architecture

In contract to SQGANs, the design of generator and discriminator for WQGANs differs significantly. Here we use the architecture proposed by Kiani et. al [11]. The discriminator takes a form of simple linear program, while the generator is similar to the one for SQGANs.

Discriminator

The set $\mathcal{O}_n$ in Equation 4.5 is too big to be useful in any practical scenarios. Instead the set of parametrized k-length Pauli String is used. Specifically,

$$\begin{aligned} H(W) &= \sum_{i_1=1}^{n-k+1} \ldots \sum_{i_k=i_{k-1}+1}^{n} \sum_{\sigma_1,\ldots,\sigma_k \in \{I,X,Y,Z\}} w_{(i_1,\ldots,i_k)}^{\sigma_1,\ldots,\sigma_k} \sigma_{i_1}^1 \otimes \ldots \otimes \sigma_{i_k}^k = \\ &= \sum_{\mathcal{I}_k \subseteq \{1,\ldots,n\}} \sum_{H \in \{I,X,Y,Z\}^{\otimes k}} w_{\mathcal{I}_k}^H H_{\mathcal{I}_k} \end{aligned} \tag{4.7}$$

Where $\mathcal{I}_k$ is a k-set of qubit indexes used to generate the k-lenght Pauli String, $H_{\mathcal{I}_k}$ is n-length Pauli String that acts non trivially on the set of at the most k qubits corresponding to $\mathcal{I}_k$ and W is the set of all weights.

We can also use the fact that the quantum Lipschitz constant of $H(W)$ is bounded by:

$$\|H(W)\|_L \leq 2 \max_{i=1,\ldots,n} \left\| \sum_{\{\mathcal{I}_k : \mathcal{I}_k \subseteq \{1,\ldots,n\} \wedge i \in \mathcal{I}_k\}} \sum_{\substack{H \in \{I,X,Y,Z\}^{\otimes k} \\ H_{(i)} \neq I}} w_{\mathcal{I}_k}^H H_{\mathcal{I}_k} \right\|_\infty . \tag{4.8}$$

Where the sum is taken only over the operators which act non-trivially on qubit i[11].

To simplify the notation, the parameter set W is enumerated as $W = \{w_1, \ldots, w_N\}$, where $N = |W|$ and H_i, $\mathcal{I}_{k_i}$ are Hamiltonian and index set associated with the weight w_i.

Now, the equation 4.5 takes the form:

$$W_1(\rho,\sigma) = \max_{w} \left(Tr[(\rho-\sigma)\sum_{i=1}^{N} w_i H_i]\right) \tag{4.9}$$

with the following constraint steaming from the quantum Lipschitz constant bound from Equation 4.8.

$$\sum_{\{i:i\in\{1,\ldots,n\}\wedge j\in\mathcal{I}_{k_i}\}} |w_i| \leq 1, \qquad j=1,\ldots,n \tag{4.10}$$

This optimization problem can be translated into the canonical form of linear programming. First, let

$$c_i = Tr[(\rho-\sigma)H_i], \tag{4.11}$$

then Equation 4.9 becomes

$$W_1(\rho,\sigma) = \max_{w} \sum_{i=1}^{N} w_i c_i. \tag{4.12}$$

Then, with

$$w_i = w_i^+ - w_i^- \tag{4.13}$$

the absolute value constraint from Equation 4.10 is equivalent to the following set of constraints:

$$\begin{aligned} w_i^+ &\geq 0 \\ w_i^- &\geq 0 \\ \sum_{\{i:i\in\{1,\ldots,n\}\wedge j\in\mathcal{I}_{k_i}\}} (w_i^+ + w_i^-) &\leq 1, \qquad j=1,\ldots,n \end{aligned} \tag{4.14}$$

Now, with two vectors defined as:

$$\begin{aligned} \boldsymbol{w'} &= [w_1^+, w_1^-, \ldots, w_N^+, w_N^-] \\ \boldsymbol{c'} &= [c_1, -c_1, \ldots, c_N, -c_N], \end{aligned} \tag{4.15}$$

matrix $A^{n\times N}$ defined as:

$$A_{j,i} = \begin{cases} 1 & \text{if} \quad j\in\mathcal{I}_{k_i} \\ 0 & \text{else} \end{cases}$$

The discriminator linear program in the canonical form looks as follows:

$$\begin{aligned} \max_{\boldsymbol{w'}} \quad & \boldsymbol{c'}^T\boldsymbol{w'} \\ \text{subject to} \quad & \boldsymbol{w'} \geq \mathbf{0} \\ & A\boldsymbol{w'} \leq \mathbf{1}. \end{aligned} \tag{4.16}$$

The weights from the original set W are recovered as:

$$w_i = w'_{2i-1} - w'_{2i}. \tag{4.17}$$

The linear program with n constraints outputs at most n non-zero weights[30], so the optimal Hamiltonian which approximates the quantum Wasserstein distance the best is given by:

$$\hat{H} = \sum_{i=1}^{\hat{N}} \hat{w}_i \hat{H}_i, \tag{4.18}$$

Where $\hat{N} \leq n$.

The Hamiltonian obtained in this way acts as the "discriminator" and it used to train the generator in the typical minmax game of GANs.

Generator

Similarly to SQGANs, the generator is built using parametrized quantum circuits. Specifically, the generator is defined as a sum of parametrized quantum circuits with corresponding probabilities:

$$G(\theta) = \sum_{i=1}^{r} p_i G_i(\theta_i) \rho_0 G_i(\theta_i)^\dagger. \tag{4.19}$$

Where $\sum_{i=1}^{r} p_i = 1$ and p_i is the probability associated with the circuit G_i and ρ_0 is the initial state of the circuit. The summation is essentially the maximal rank of the output state that the generator is able to generate. Namely, the generator constructed like that is able to learn a mix of at most r pure states.

In all of our experiments we use the same design for each circuit within the generator, i.e. $G_i = \bar{G} \quad \forall i \in 1, \ldots, r$, and only the parameters θ_i differ. We use different designs for $\bar{G}$ for different experiments, however they always fall into two categories.

1. Generic $\bar{G}$ described in Appendix B.1 (the same as for SQGANs).
2. $\bar{G}$ the same as the circuit used to generate the real input state.

We also always set the initial state $\rho_0 = |0\rangle$.

In the end the generator becomes:

$$G(\theta) = \sum_{i=1}^{r} p_i \bar{G}(\theta_i) |0\rangle \bar{G}(\theta_i)^\dagger. \tag{4.20}$$

4.2.3. Training

Similarly to WGANs the discriminator part of the WQGANs computes the best approximation of the quantum Wasserstein distance and the generator tries to minimize it. This can be again encoded as the minmax game.

Given real input state ρ_r, generated state $G(\theta)$ and using trace properties from Equation 4.9 we get:

$$W_1(\rho_r, G(\theta)) = \max_w (Tr[(\rho_r - G(\theta)) \sum_{i=1}^{N} w_i H_i]) = \max_w (\sum_{i=1}^{N} w_i (Tr[\rho_r H_i] - Tr[G(\theta) H_i])). \tag{4.21}$$

The generator tries to minimize the quantum Wasserstein distance to the real input state, so in general terms WQGANs optimization objective is stated as:

$$\max_w \min_\theta \mathcal{L}(w, \theta) = \max_w \min_\theta (\sum_{i=1}^{N} w_i (Tr[\rho_r H_i] - Tr[G(\theta) H_i])) \tag{4.22}$$

In practice however, we do not have to calculate the expectation $Tr[G(\theta)H_i]$ for every i when training the Generator, instead we can use the $\hat{H}$ defined earlier. Also, the real input state expectations $Tr[\rho_r H_i]$ can be precomputed and only accessed during the training.

The overall training procedure consists of the following steps:

1. Compute the expectations $Tr[G(\theta)H_i]$ and use them together with the precomputed expectation for the real input state to calculate $Tr[\rho_r H_i] - Tr[G(\theta)H_i]$ for each i.
2. Find $\hat{H}$ using the linear programming formulation.
3. Use $\hat{H}$ to find gradients of $Tr[G(\theta)\hat{H}]$ w.r.t parameters θ_i and p_i and update them.

Those 3 steps are repeated until some stopping condition is satisfied, in our case we run the training for given number of steps.

4.2.4. Evaluation Results

Experimental Setup

We use two different circuits to generate real input samples. First one is the same as used in the SQGANs and is described in Appendix B.2. The second one is a butterfly circuit (also used by Kiani et. al[11]) which is described in Appendix B.3. For the butterfly circuit we use random initialization for all the gates.

We try two different approaches to the generator design. We use the generic generator described in Appendix B.1 and generator design matching the circuit used to generate real input data. We specify the details of generator architecture for each experiment separately.

In all the experiment we use Adam optimizer [20] with parameters $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\hat{\epsilon} = 1e-9$ and the learning rate of 0.1. This learning rate was determined by experimentally. It is important to note that the training procedure is very sensitive to changes to this parameter. If the learning rate is too low, gradient very quickly falls in a local optimum and the fidelity between real and generated state stays close to 0.

The generator and discriminator are trained interchangeably one iteration each and the exact number of epoch is specified for each experiment separately.

Results For Pure Real State

First we look at the results for topological phase transition estimation in Figure 4.4. The real input state is generated by running the circuit from Appendix B.2 for random $g \in [-1; 1]$ and the generator is built using the generic ansatz from Appendix B.1. Again, no surprisingly the training gets more difficult as the real input size grows.

Two the most impactful training parameters are the number of generator layers and k-length of Pauli strings used in the discriminator. With $k = 3$ the fidelity is larger than 0.9 for real input size up to 6 qubits. For bigger real input sizes, expectation of larger operator is needed. The convergence is very fast, for 4 generator layers only around 100 epoch are necessary to achieve very good approximation, the improvements beyond that are rather minimal. For 5 layers the training must be longer and it's much harder to achieve high fidelity. For more results refer to Appendix C.1.

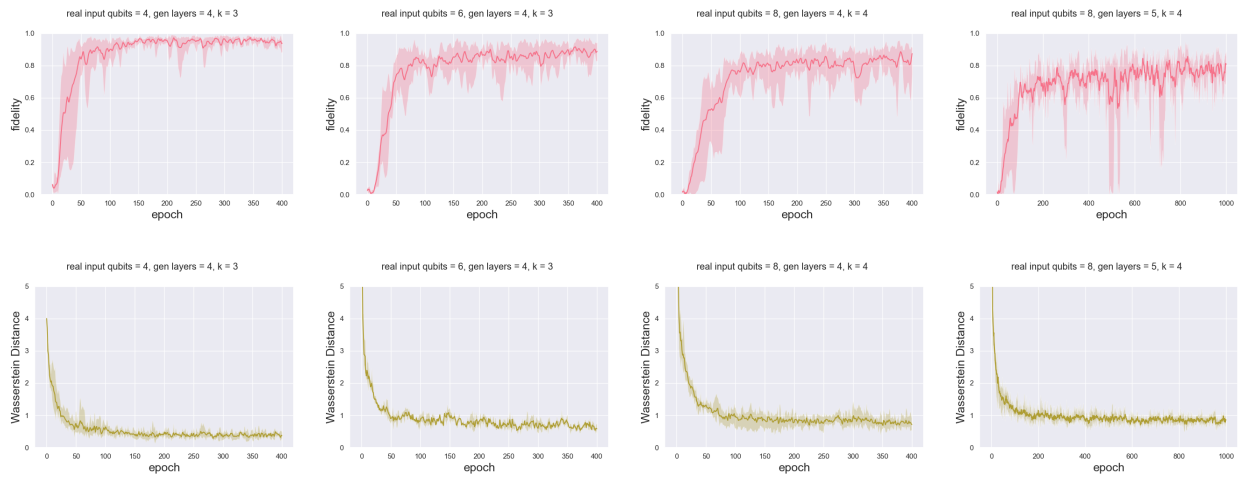


Figure 4.4.: Results of topological phase transition estimation (ansatz Appendix B.2). The solid line represents the average value and the shaded area represents the range from 5 different experiments. The upper row shows the fidelity and the bottom row shows the corresponding Wasserstein distance. In all the experiments the generator is built using ansatz from B.1.

The results for the butterfly circuit estimation are shown in the Figure 4.5. The real input state is generated by running the circuit from Appendix B.3 for randomly selected values in each gate and the generator is built using the generic ansatz from Appendix B.1. Again, no surprisingly the training gets more difficult as the real input size grows. However, in this case we can see that the results are worse than for the phase transition circuit.

It is interesting to compare those results, with the one obtained using the same circuit ansatz for both, real input and the generator. In Figure 4.6 we see the results for using the butterfly circuit for both, real input and generator. Regardless of the size, very high fidelity was achieved. This shows the importance of the generator design in the training process.

In all the examples we see that the approximated Wasserstein distance approaches 0 as

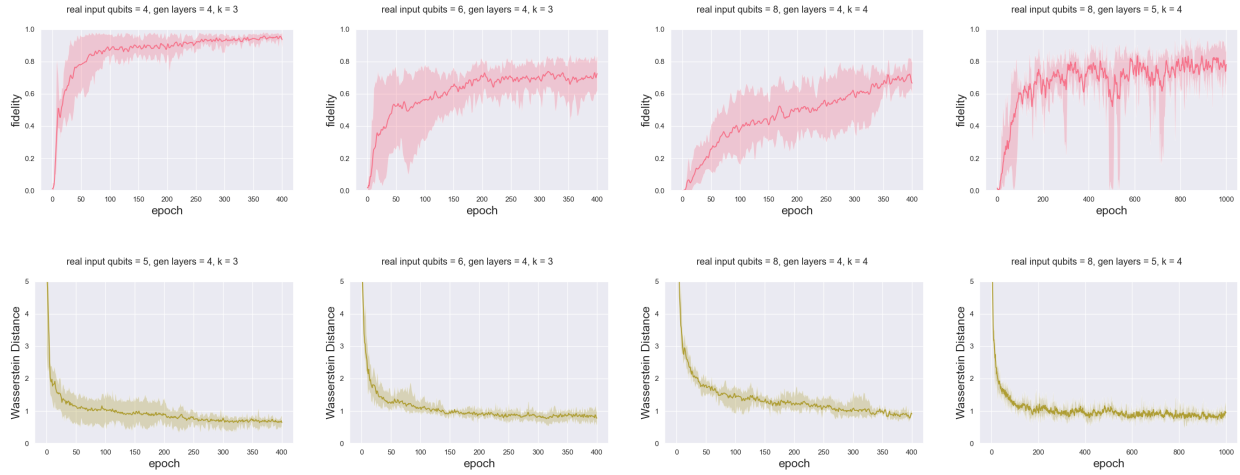


Figure 4.5.: Results for butterfly circuit (ansatz Appendix B.3). The solid line represents the average value and the shaded area represents the range from 5 different experiments. The upper row shows the fidelity and the bottom row shows the corresponding Wasserstein distance. In all the experiments the generator is built using ansatz from B.1.

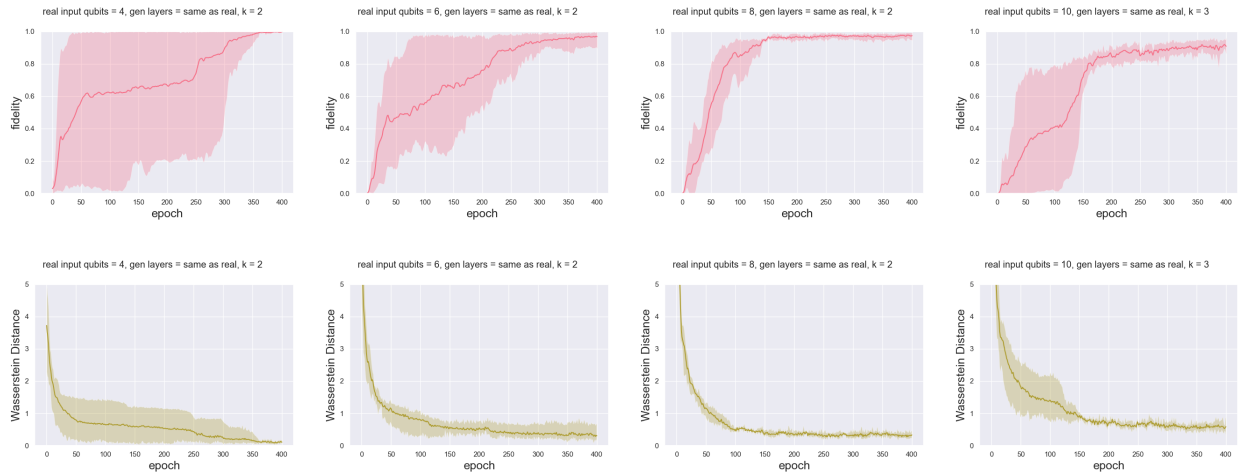


Figure 4.6.: Results for butterfly circuit (ansatz Appendix B.3). The solid line represents the average value and the shaded area represents the range from 5 different experiments. The upper row shows the fidelity and the bottom row shows the corresponding Wasserstein distance. In all the experiments the generator is built using the same butterfly circuit.

fidelity approaches 1, which is the expected behavior. The Wasserstein distance below 1 indicates high fidelity between real input and generated states.

It is also important to note here, that using fixed k might not be optimal strategy. If for given operator, the different between expectation of the generator and the real input source is close to 0, this operator does not provide any useful information in context of learning anymore. So instead of evaluating all operators for given k, we can start with some subset and remove and add new ones as the training progresses [11].

Results For Mixed Real State

The generator definition allows for mixed state approximation. In theory each circuit in the generator should be able to learn different pure state and its probability in the mix. However, we found it difficult to achieve in practice.

We did not manage to train mixed state generator using the generic ansatz. But when using the same ansatz for the real input and the generator we were able approximate an equal mixture of two topological phase transition circuits (results presented in Figure **??**). The generator with 4 circuits used 2 of them to approximate the real input and also learnt the correct probability of 0.5 for both of them. The other 2 circuits got assigned the probability close to 0 and states far from the one in the real input.

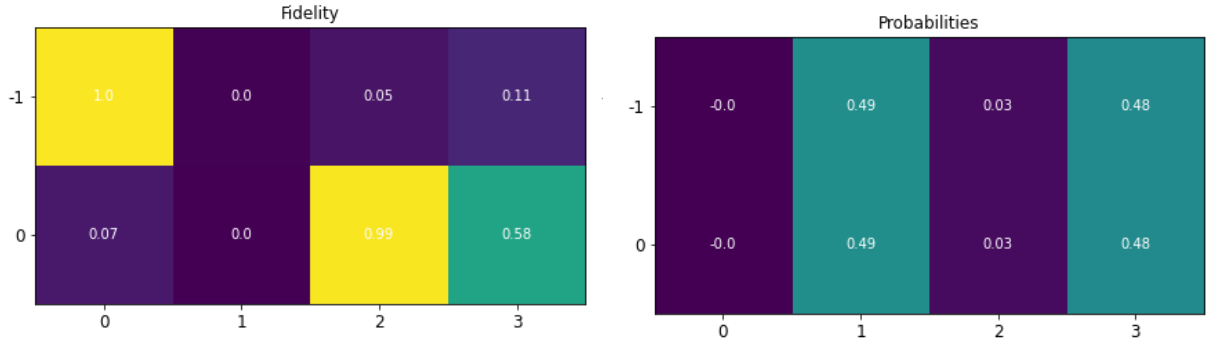


Figure 4.7.: Results for topological phase transition circuit (Appendix B.2). The real input source produces two states (for $g = 0$ and $g = -1$) each with 0.5 probability. The generator consist of 4 circuits, each with the same ansatz as the real input. (left) the circuit with index 0 learnt the state for $g = -1$ and the circuit with index 2 learnt the state for $g = 0$. The other two circuits did not any of the states well. (right) How far is the probability of given circuit from the actual probability in the real input. For the circuit 0 and 2 the number is close to 0, which means that the probability was correctly learnt to be 0.5, for the other two it's close to 0.5, which means it is correctly around 0.

The results for the mixture of 3 states in the real input were much worse. While there were signs that the generator was learning, usually one of the states was forgotten (i.e. the generator learnt only the mixture of 2 states).

4.2.5. Conclusions

Our experiments confirmed the ability to approximate quantum states using the quantum Wasserstein distance. It is easier to train such networks and they give better results than SQGANs. As the input size grows, increasing the Pauli Strings length k allows for high approximation (at the cost of increased training complexity).

The design of the generator plays a crucial role in the quality of the approximations. Better design also allows to use lower k which makes the training more efficient.

WQGANs also work for mixed state approximation, although only for small mixtures.

However, this architecture still does not allow to generate new, unseen before, states. In the next chapter we explore ways in which this can be achieved using WQGANs.

5. Unseen Quantum State Generation

In the previous chapter we evaluated two different types of quantum GANs. The common problem of those is the inability to generate new, unseen states. In this chapter we propose the hybrid classical-quantum framework that can overcome this limitation.

Our idea is based WQGANs and how the quantum Wasserstein distance is approximated during the training. The discriminator at every step approximates the distance between some fixed, real input state and the generated state which changes after each iteration. However, the discriminator never needs an access to the actual real input state, it only operates on the set of measured expectations.

Given a parametrized circuits U and set of parameters $\Theta = \{\theta_i\}$ and a set of operators $H = \{H_j\}$, we prepare a set of vectors of expectations S. Each vector $s_{\theta_i} \in S$ contains the expectations of the circuit $U(\theta_i)$, such that $s_{\theta_i}^{(j)} = \langle H_j \rangle_{U(\theta_i)}$.

Assuming that all the vectors in S come from the same distribution p_S, the proposed framework is defined in two parts as follows:

1. *Classical*: Takes as the input the set S and uses it to learn a function $f : \mathbb{R}^n \to \mathbb{R}^{|H|}$. Given an arbitrary vector $g \in \mathbb{R}^n$ (e.g. random noise), this function produces a new vector $s' = f(g)$ such that $s' \sim p_S$.

2. *Quantum*: Takes s' as the input and uses it as expectations of real input source in the WQGANs setting described in the previous chapter. The generator trained using s' produces new, unseen before quantum state. The WQGANs optimization objective from Equation 4.22 becomes:

$$\max_w \min_\theta \mathcal{L}(w, \theta) = \max_w \min_\theta \left(\sum_{i=1}^{N} w_i (s'^{(i)} - Tr[G(\theta) H_i])\right) \tag{5.1}$$

Once the function f is learnt, it can be used arbitrary many times to produce new vectors of expectations. With those vectors, it is possible to generate new quantum states that come from some circuit $U(\theta')$, without ever knowing U or θ'. In the following sections, we show how exactly the function f can be obtained for two different cases.

5.1. Labeled State Generation

If the quantum state produced by the circuit U can be labeled by some continuous variable, we can use this variable to find the function f.

Specifically, here we assume that each parameter $\theta_i^{(j)}$ is described by some function, i.e. $\theta_i = \theta(g_i) = [\theta^{(1)}(g_i), \theta^{(2)}(g_i), \ldots, \theta^{(l)}(g_i)]$, for some $g_i \in V \subseteq \mathbb{R}$, where l is the number of parameters in the circuit U. We also assume that the expectations of the state produced by the circuit $U(\theta(g_i))$, can be described by some other functions, i.e. $s_{\theta_i} = s(g_i) = [s^{(1)}(g_i), s^{(2)}(g_i), \ldots, s^{(|H|)}(g_i)]$, $s^{(j)} : V \to [-1;1]$ $\forall_{j \in 1,\ldots,|H|}$. Then, the input to the classical part of the framework is the set S, together with corresponding g_i for each $s_{\theta_i} = s(g_i) \in S$. To find $s^{(j)}$ $\forall_{j=1,\ldots,|H|}$ functions interpolation is sufficient. So, the function $f : V \to \mathbb{R}^{|H|}$ simply takes any value of $g \in V$ and returns the expectations for this value using interpolations of functions $s^{(j)}$.

5.1.1. Evaluation Results

This approach can be used when U is the topological phase transition circuit from Appendix B.2. All the parameters of this circuit can be described by three functions $\theta_v, \theta_w, \theta_r$ over $V = [-1;1]$. To prepare the input to the classical part m ($m = |S|$) values of $g \in V$ are sampled and the expectations of $U(\theta_v(g_i), \theta_w(g_i), \theta_r(g_i))$ $\forall_{i=1,\ldots,m}$ are calculated for all operators $H_i \in H$. Similarly as in the WQGANs chapter, H is chosen to be a set of all k-length Pauli Strings. This data is used to interpolate the expectation functions for those operators.

In Figure 5.1 interpolated expectations of the circuit for $k = 3$ and $m = 11$ are plotted (only subset of the expectation is plotted for readability).

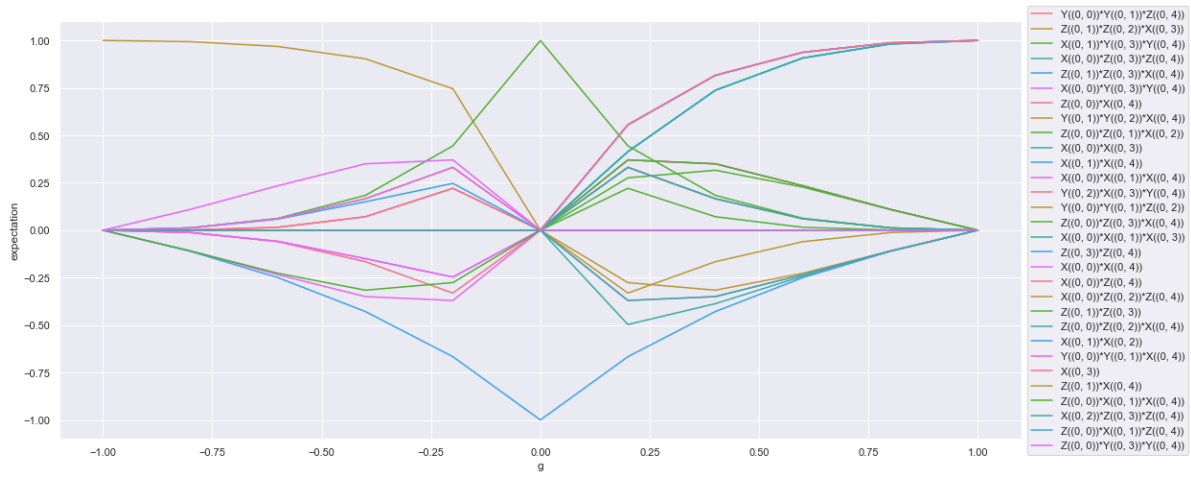


Figure 5.1.: Interpolated expectations of topological phase transition circuit (Appendix B.2) with 5 qubits width for 30 random 3-Pauli Strings operators. Interpolation using evenly spaced 11 values of the parameter $g \in [-1;1]$.

The interpolated expectations are used to learn the quantum states for the values of g that were not part of the classical input. In Figure 5.2 we see fidelity and Wasserstein distance between the real input states and the ones learnt using the interpolated expectations. Those results are comparable with the ones obtained by learning from known expectations (Figure 4.4).

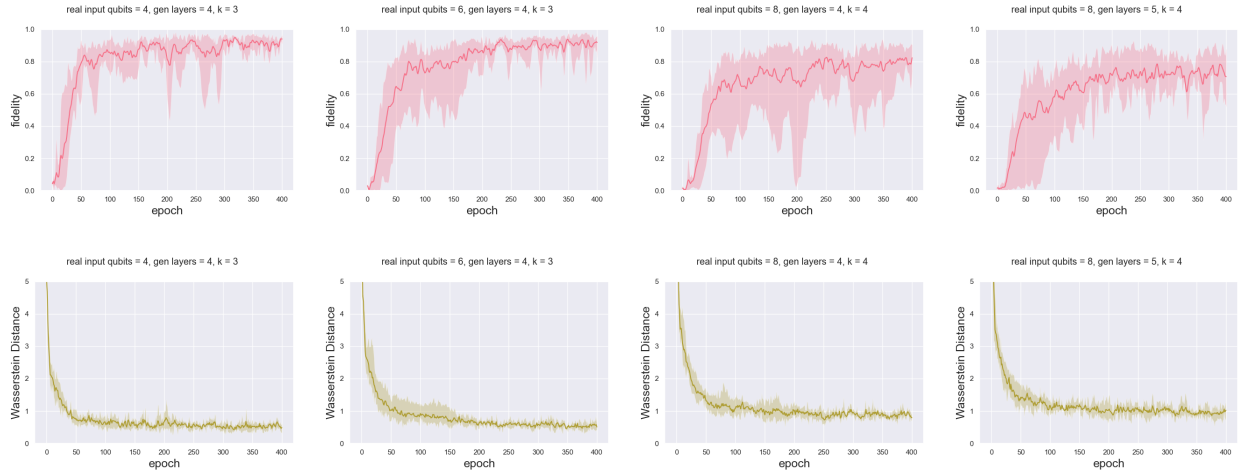


Figure 5.2.: Results for topological phase transition circuit (ansatz Appendix B.2) using interpolated expectations. The solid line represents the average value and the shaded area represents the range from 5 different experiments. The upper row shows the fidelity and the bottom row shows the corresponding Wasserstein distance. In all the experiments the generator is built using ansatz from Appendix B.1.

String Order Parameters

In their work Smith et. al[19] define two string order parameters

$$\begin{aligned} S^{\mathbb{1}} &= \langle\psi| \left(\prod_{i=3}^{N-2} X_i\right) |\psi\rangle \\ S^{ZY} &= \langle\psi| Z_2 Y_3 \left(\prod_{i=4}^{N-3} X_i\right) Y_{N-2} Z_{N-1} |\psi\rangle , \end{aligned} \tag{5.2}$$

where N is the width of the circuit and $|\psi\rangle$ is the final state obtained by the topological phase transition circuit from Appendix B.2. The measurements of $S^{\mathbb{1}}$ and S^{ZY} on states learnt using the interpolated expectations are shown in Figure 5.3. The obtained results closely follow the expected value and the phase transition point at $g = 0$ is clearly distinguishable.

5.1.2. Conclusions

Using the interpolated expectations allows to learn unknown quantum states. In all the experiments generic generator ansatz was used, meaning that also the design of U its parametrization was unknown to the quantum Generator and Discriminator.

Although the setup described here assumes one dimensional variable g, this notion can be extended to multi-variable case where $g \in V \subseteq \mathbb{R}^m$.

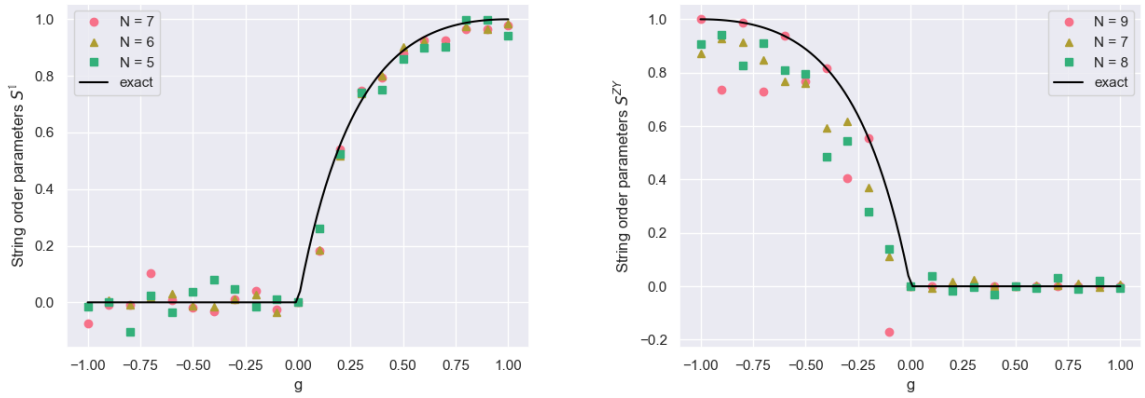


Figure 5.3.: String order parameters $S^{\mathbb{1}}$ and S^{ZY} measured on the generic generator from Appendix B.1, trained using the interpolated expectations, for different width of circuit N. The phase transition at $g = 0$ is clearly visible, the results are very close to the exact ones.

5.2. Unlabeled State Generation

In the general case the assumptions from the previous Section do not hold and more powerful tools are need to find the function f. However, following the assumption that all vectors in input set S come from the same distribution p_S, we can use the generative modeling to learn the function f. In particular, here we use classical Wasserstein Generative Adversarial Networks (WGANs) to approximate the distribution p_S and later use the classical Generator as the function f to produce the vectors s'.

5.2.1. Evaluation Results

We use this technique to generate new, previously unseen states from the butterfly circuit (Appendix B.3). First we generate the set S and use it to train a simple WGAN with gradient penalty[16], with penalty factor 10. We use simple 2-layers deep neural network for both, generator and discriminator and Adam optimizer [20] with the following parameters $\beta_1 = 0$, $\beta_2 = 0.9$, $\hat{\epsilon} = 1e-9$ and learning rate of 0.0001. The layers dimensions of generator and discriminator varies depending on the real input width and are specified separately.

6. Conclusions

A. SGANs approximate Jensen–Shannon Divergence

B. Circuits

B.1. SQGANs Ansatz

B.2. Topological Phase Transition Ansatz

B.3. Butterfly Ansatz

C. Additional Results

C.1. WQGANs Results

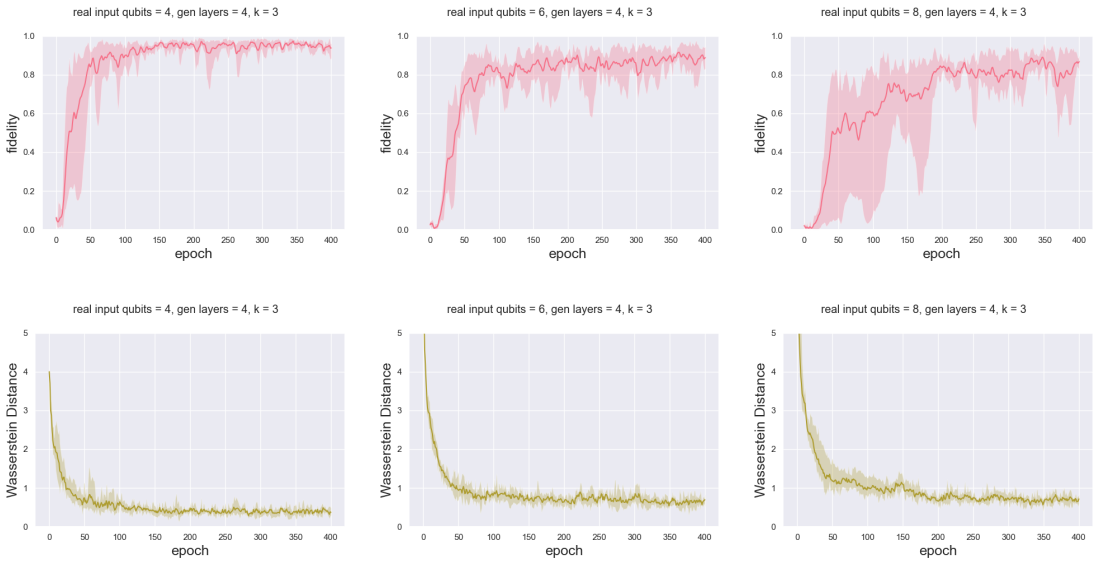


Figure C.1.: The solid line represents the average value and the shaded area represents the range from 5 different experiments.

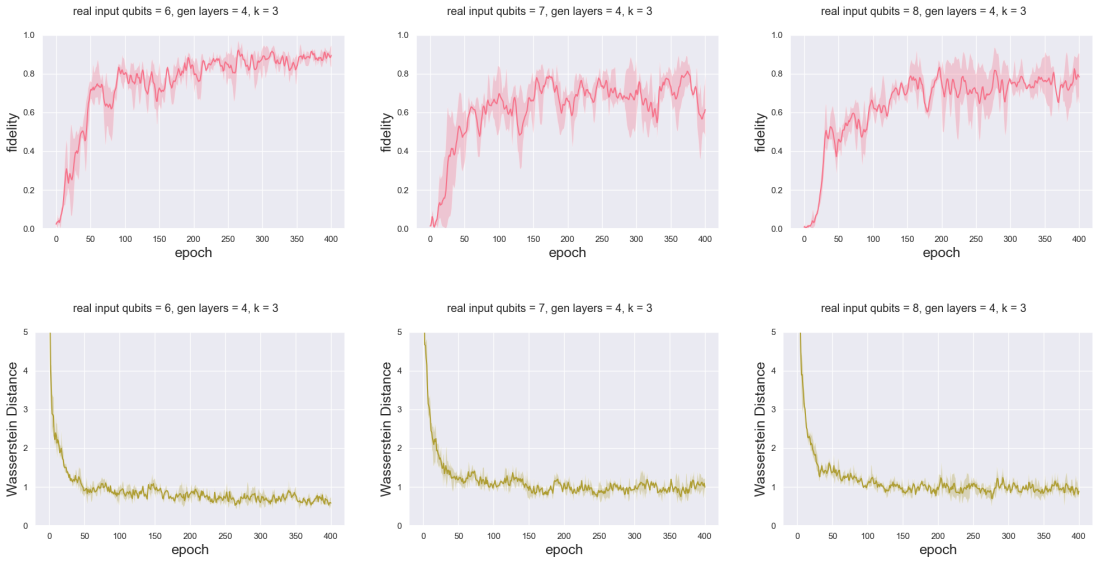


Figure C.2.: The solid line represents the average value and the shaded area represents the range from 5 different experiments.

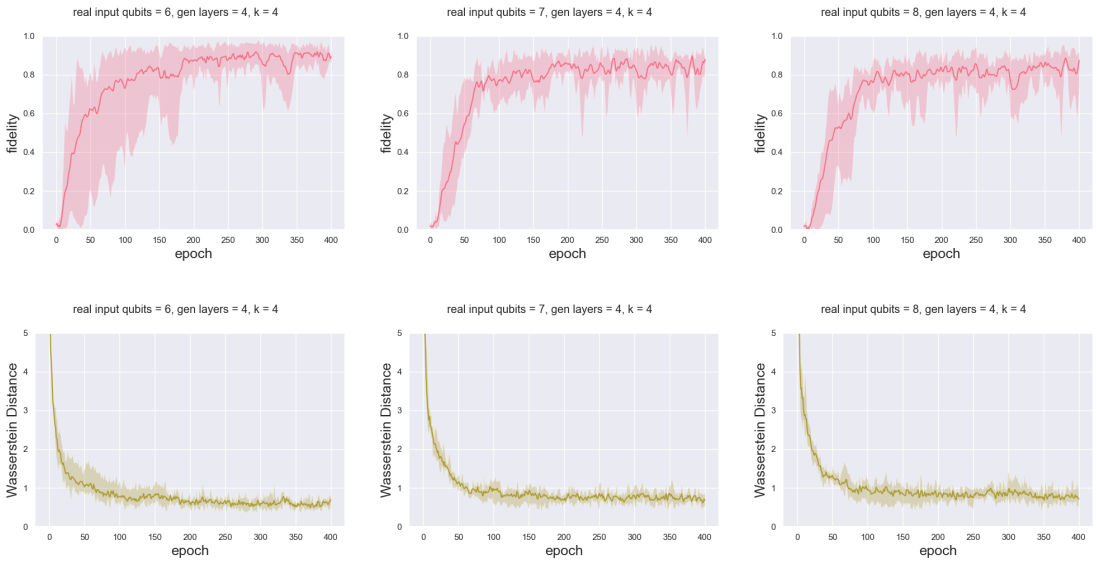


Figure C.3.: The solid line represents the average value and the shaded area represents the range from 5 different experiments.

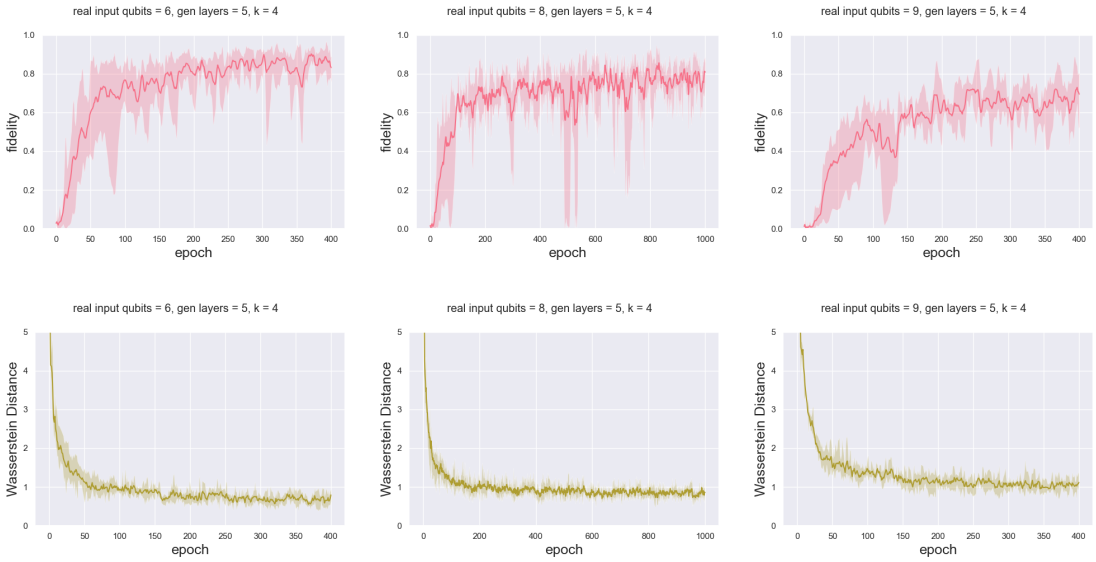


Figure C.4.: The solid line represents the average value and the shaded area represents the range from 5 different experiments.

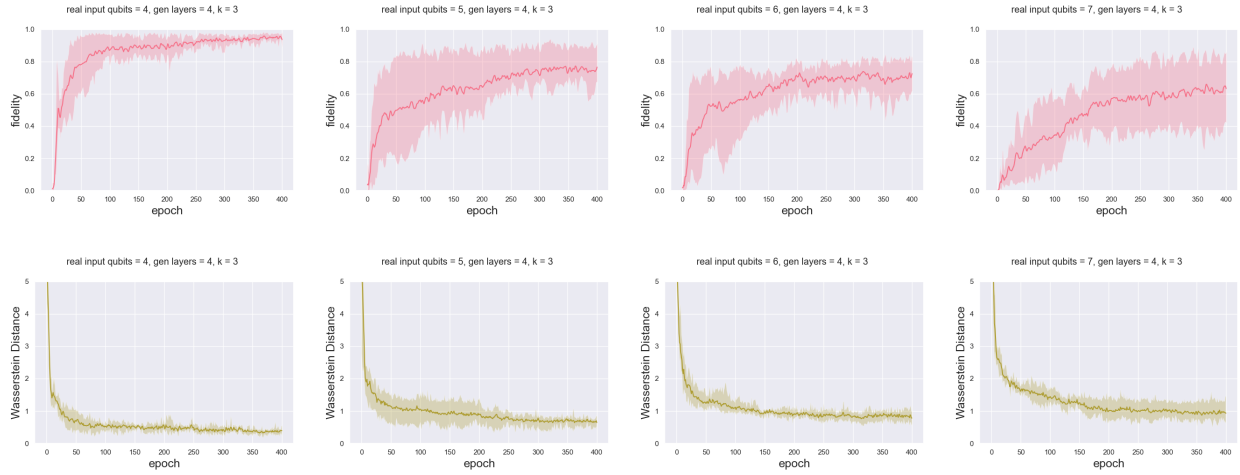


Figure C.5.: Results of butterfly circuit estimation (ansatz Appendix B.3). The solid line represents the average value and the shaded area represents the range from 5 different experiments. The upper row shows the fidelity and the bottom row shows the corresponding Wasserstein distance. In all the experiments the generator is built using ansatz from B.1.

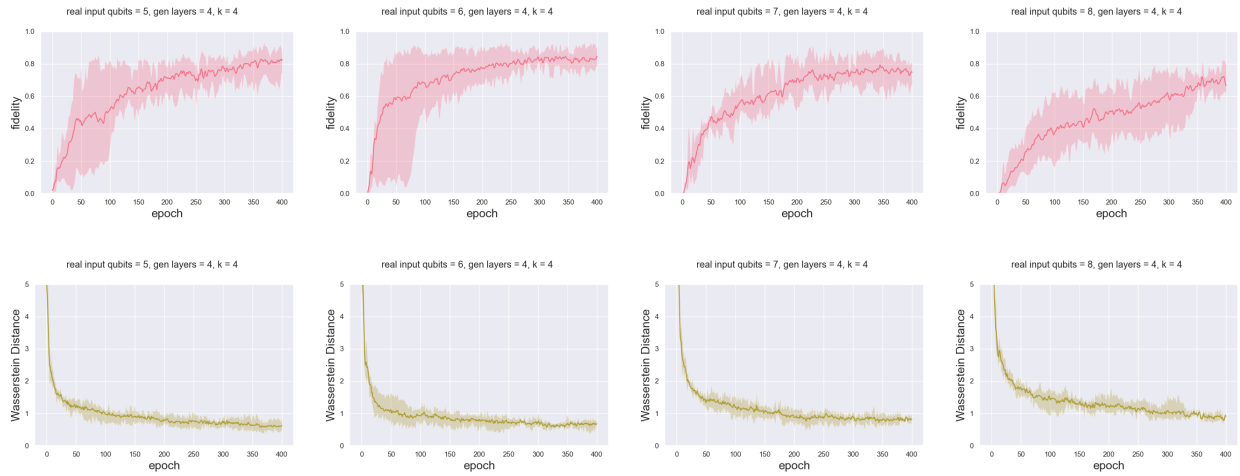


Figure C.6.: Results of butterfly circuit estimation (ansatz Appendix B.3). The solid line represents the average value and the shaded area represents the range from 5 different experiments. The upper row shows the fidelity and the bottom row shows the corresponding Wasserstein distance. In all the experiments the generator is built using ansatz from B.1.

List of Figures

List of Tables

Bibliography

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[2] M. Mirza and S. Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[3] T. Karras, S. Laine, and T. Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].

[4] A. Radford, L. Metz, and S. Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[5] M. Arjovsky, S. Chintala, and L. Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[6] P.-L. Dallaire-Demers and N. Killoran. "Quantum generative adversarial networks". In: *Physical Review A* 98.1 (July 2018). ISSN: 2469-9934. DOI: 10.1103/physreva.98.012324. URL: http://dx.doi.org/10.1103/PhysRevA.98.012324.

[7] M. Benedetti, E. Grant, L. Wossnig, and S. Severini. "Adversarial quantum circuit learning for pure state approximation". In: *New Journal of Physics* 21.4 (Apr. 2019), p. 043023. ISSN: 1367-2630. DOI: 10.1088/1367-2630/ab14b5. URL: http://dx.doi.org/10.1088/1367-2630/ab14b5.

[8] J. Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: http://dx.doi.org/10.22331/q-2018-08-06-79.

[9] C. Zoufal, A. Lucchi, and S. Woerner. "Quantum Generative Adversarial Networks for learning and loading random distributions". In: *npj Quantum Information* 5.1 (Nov. 2019). ISSN: 2056-6387. DOI: 10.1038/s41534-019-0223-2. URL: http://dx.doi.org/10.1038/s41534-019-0223-2.

[10] G. D. Palma, M. Marvian, D. Trevisan, and S. Lloyd. *The quantum Wasserstein distance of order 1*. 2020. arXiv: 2009.04469 [quant-ph].

[11] B. T. Kiani, G. D. Palma, M. Marvian, Z.-W. Liu, and S. Lloyd. *Quantum Earth Mover's Distance: A New Approach to Learning Quantum Data*. 2021. arXiv: 2101.03037 [quant-ph].

[12] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[13] H. Narayanan and S. Mitter. "Sample Complexity of Testing the Manifold Hypothesis". In: *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*. NIPS'10. Vancouver, British Columbia, Canada: Curran Associates Inc., 2010, pp. 1786–1794.

[14] M. Arjovsky and L. Bottou. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017. arXiv: 1701.04862 [stat.ML].

[15] C. Villani. "Optimal transport – Old and new". In: vol. 338. Jan. 2008, pp. xxii+973. DOI: 10.1007/978-3-540-71050-9.

[16] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[17] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, W.-K. Mok, S. Sim, L.-C. Kwek, and A. Aspuru-Guzik. *Noisy intermediate-scale quantum (NISQ) algorithms*. 2021. arXiv: 2101.08448 [quant-ph].

[18] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe. "Circuit-centric quantum classifiers". In: *Physical Review A* 101.3 (Mar. 2020). ISSN: 2469-9934. DOI: 10.1103/physreva.101.032308. URL: http://dx.doi.org/10.1103/PhysRevA.101.032308.

[19] A. Smith, B. Jobst, A. G. Green, and F. Pollmann. *Crossing a topological phase transition with a quantum computer*. 2020. arXiv: 1910.05351 [cond-mat.str-el].

[20] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[21] E. A. Carlen and J. Maas. *An Analog of the 2-Wasserstein Metric in Non-commutative Probability under which the Fermionic Fokker-Planck Equation is Gradient Flow for the Entropy*. 2012. arXiv: 1203.5377 [math.FA].

[22] Y. Chen, T. T. Georgiou, and A. Tannenbaum. *Matrix Optimal Mass Transport: A Quantum Mechanical Approach*. 2016. arXiv: 1610.03041 [math-ph].

[23] Y. Chen, T. Georgiou, and A. Tannenbaum. "Wasserstein Geometry of Quantum States and Optimal Transport of Matrix-Valued Measures". In: Jan. 2018, pp. 139–150. ISBN: 978-3-319-67067-6. DOI: 10.1007/978-3-319-67068-3_10.

[24] L. Ning, T. T. Georgiou, and A. Tannenbaum. *Matrix-valued Monge-Kantorovich Optimal Mass Transport*. 2013. arXiv: 1304.3931 [cs.SY].

[25] G. Peyré, L. Chizat, F.-X. Vialard, and J. Solomon. *Quantum Optimal Transport for Tensor Field Processing*. 2017. arXiv: 1612.08731 [cs.GR].

[26] F. Golse, E. Caglioti, and T. Paul. *Quantum optimal transport is cheaper*. 2021. arXiv: 1908.01829 [math.AP].

[27] F. Golse, C. Mouhot, and T. Paul. "On the Mean Field and Classical Limits of Quantum Mechanics". In: *Communications in Mathematical Physics* 343.1 (Jan. 2016), pp. 165–205. ISSN: 1432-0916. DOI: 10.1007/s00220-015-2485-7. URL: http://dx.doi.org/10.1007/s00220-015-2485-7.

[28] N. Yu, L. Zhou, S. Ying, and M. Ying. *Quantum Earth mover's distance, No-go Quantum Kantorovich-Rubinstein theorem, and Quantum Marginal Problem*. 2019. arXiv: 1803.02673 [quant-ph].

[29] S. Chakrabarti, Y. Huang, T. Li, S. Feizi, and X. Wu. *Quantum Wasserstein Generative Adversarial Networks*. 2019. arXiv: 1911.00111 [quant-ph].

[30] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. 1st. Athena Scientific, 1997. ISBN: 1886529191.