Generative Adversarial Networks

Simran Sinha*
Indian Institute of Technology Bombay

In this report, we present an in-depth study of the implementation of Generative Adversarial Networks (GANs) with a focus on both classical and quantum versions of the model. We begin by providing a thorough overview of the theory behind GANs, including their architecture and training process. We then present the implementation details of classical GANs, including the choice of model architecture, hyperparameter tuning, and evaluation metrics. We also provide several examples of real-world applications of classical GANs. Next, we extend our discussion to quantum GANs and their unique properties that can be leveraged to achieve improved performance compared to classical GANs. We provide a detailed description of the quantum computing libraries and frameworks that can be used to implement quantum GANs. We also present the implementation details of quantum GANs and provide examples of their potential applications in various domains. Additionally, we discuss the challenges and open research questions in the field of quantum GANs. Overall, this report provides a comprehensive guide to the implementation of GANs and an in-depth understanding of their potential impact on the field of machine learning and artificial intelligence.

I. INTRODUCTION

Deep learning is a method of creating complex models to understand and interpret data, such as images, audio, and text. So far, the most successful applications of deep learning have been in creating models that can classify or identify different objects or sounds in the data. These models use techniques like backpropagation and dropout to learn and make predictions. However, there is a different type of deep learning model called a *generative model* which aims to understand and recreate the underlying probability distributions of the data. These models have not been as successful as discriminative models due to the difficulties in approximating complex mathematical calculations and using certain types of units in the model.

A. Classical GANs

The adversarial nets framework is a method where two neural networks, a generative and a discriminative model, are trained to work together in order to create realistic data samples.

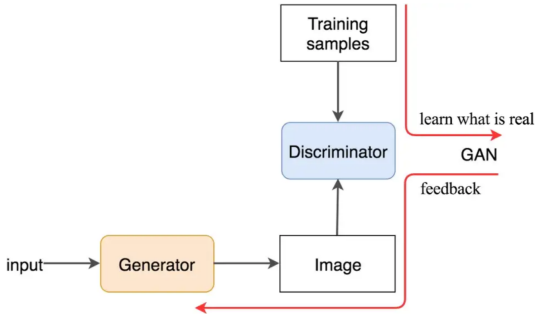


fig: GAN framework

The generative model is responsible for creating new data samples and the discriminative model is responsible for identifying if a sample is real or fake. The two models are trained together in an adversarial way, where the generative model is trying to create samples that can fool the discriminative model and the discriminative model is trying to identify the fake samples created by the generative model. As they compete, they both improve, with the generative model creating more realistic samples and the discriminative model becoming better at identifying the fake samples. Eventually, the goal is to reach a point where the samples created by the generative model are indistinguishable from real data samples.

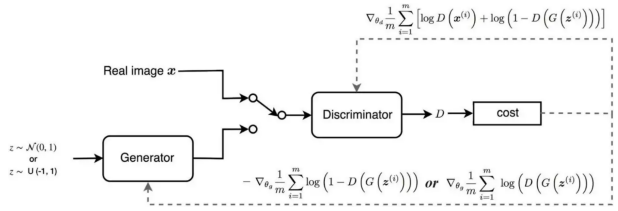


fig: Summary of GAN design

Formally, GANs are a structured probabilistic model containing latent variables z and observed variables x. The two players in the game are represented by two functions, each of which is differentiable both with respect to its inputs and with respect to its parameters. The generator is defined by a function G that takes z as input and uses θ as parameters, i.e., $G : G(z, \theta) \rightarrow x'$. The discriminator is a function D that takes either the generated data x' or the real data x as input and the binary classification result (real or fake) as output, i.e., $D : D(x, \gamma) \rightarrow (0, 1)$ where γ being trainable parameters for D. Both G and D are typically implemented by deep neural networks.

The training process of GANs involves both finding the parameters of a discriminator γ to maximize the classification accuracy, and finding the parameters of a generator θ to maximally confuse the

* 210260051@iitb.ac.in

discriminator. Both players have cost functions that are defined in terms of both players' parameters, cost function for discriminator is $\max_\gamma \mathcal{L}_D = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x))] + \mathbb{E}_{x\sim p_z(z)}[\log(1-D(G(z)))]$ and for generator is $\min_\theta \mathcal{L}_G = \mathbb{E}_{x\sim p_z(z)}[\log(1-D(G(z)))]$. The original cost function has a vanishing gradient issue and an alternative cost function is proposed $\min_\theta \mathcal{L}_G = \mathbb{E}_{x\sim p_z(z)} -[\log(D(G(z)))]$ and the parameters of two models are updated iteratively using gradient descent methods. The discriminator tries to minimize its cost function, $\mathcal{L}_{(D)}(\gamma,\theta)$, by adjusting its own parameters, γ. Similarly, the generator tries to minimize its cost function, $\mathcal{L}_{(G)}(\gamma,\theta)$, by adjusting its own parameters, θ. However, both players' cost functions are dependent on each other's parameters, which they do not control. This situation is described as a game rather than an optimization problem as the solution is not just a minimum in a single player's cost function, but a balance between both players' cost functions. The solution to the game is known as a Nash equilibrium, which is a combination of the discriminator's parameters and the generator's parameters that result in a local minimum of $\mathcal{L}_{(D)}$ with respect to γ and a local minimum of $\mathcal{L}_{(G)}$ with respect to θ [1]. If both models have sufficient capacity, then the Nash equilibrium of this game corresponds to the G(z) being drawn from the same distribution as the training data, and $D(x) = 1/2$ for all x.

B. Quantum GANs

Quantum Generative Adversarial Networks (QGANs) are a new class of machine learning models that harness the power of quantum computing to improve the performance of traditional Generative Adversarial Networks (GANs). The quantum generator has the ability to model and fit the data distribution in a more efficient and effective way compared to classical generators. The reason for this is that the quantum generator uses quantum circuits to generate data, which can have more powerful expressive power compared to classical methods. This, in turn, can lead to better results in terms of capturing the underlying distribution of the data.

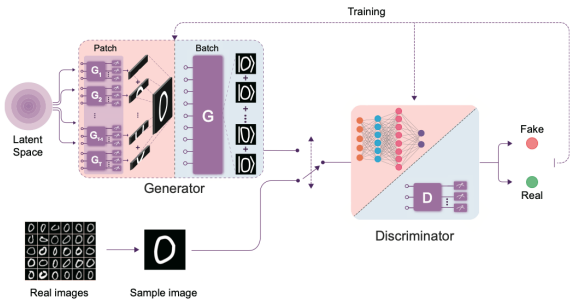


fig: QGANs scheme (quantum generator G and a discriminator D, can be classical or quantum)

The generator takes a latent vector z, sampled from a specific distribution, and generates data $G(z) \sim P_g(G(z))$, $P_g(G(z)) \approx P_{data}(x)$ that aims to look like real data x drawn from a data distribution $Pdata(x)$. Meanwhile, the discriminator tries to tell the difference between the real data $x \sim P_{data}(x)$ and generated data $G(z)$. The amount of quantum resources used (N-qubits with O(poly(N)) circuit depth) depends on the feature dimension M of the data. Two strategies have been devised to use these resources effectively: the patch strategy for cases when resources are limited ($N < \lceil \log M \rceil$) and the batch strategy for cases when resources are sufficient ($N > \lceil \log M \rceil$). The design of quantum patch GAN aims to use insufficient quantum resources to generative high-dimensional features, while the quantum batch GAN can be used for parallel training given sufficient resources. The quantum GAN can adapt and use available resources optimally.

1. *Quantum patch GAN*

The patch strategy is applied to manipulate large M with small N, i.e., when the dimensionality of the features in the training data (M) is large and the number of qubits available on the quantum device (N) is small, the patch strategy is used to manipulate the generation of these high-dimensional features. The goal is to make the most of the limited quantum resources to generate high-dimensional features. The three core components of quantum patch GAN are the quantum generator, classical discriminator, and optimization rule.

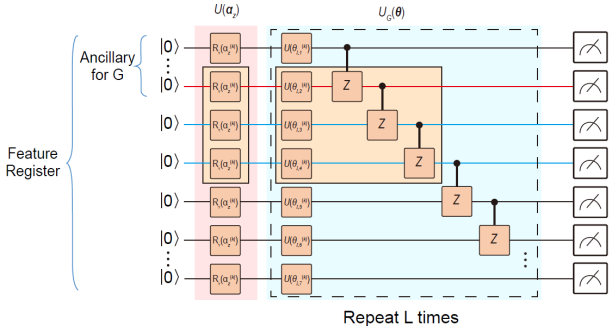


fig: Architecture for quantum patch GAN

The quantum generator consists of T sub-generators $\{G_t\}_{t=1}^T$, and each sub-generator is responsible for generating a specific part of the feature vector. At each iteration, the implementation of one sub-generator G_t is described. Suppose that the available quantum device has N qubits, and these qubits are divided into two parts: N_G qubits are used to generate a feature vector of length 2^{N_G}, while the remaining N_A qubits are used to perform a nonlinear mapping, which is crucial in deep learning. The input state, $\left|z^{(k)}\right\rangle$ is then prepared with a mathematical form $\left|z^{(k)}\right\rangle = (\bigotimes_{i=1}^N R_Y(\alpha_z^{(k)}))\,|0\rangle^{\otimes N}$, where R_Y refers to the rotation single qubit gate along the y-axis and $\alpha_z^{(k)}$ is sampled from the uniform distribution. At each iteration, the same latent state $\left|z^{(k)}\right\rangle$ is input into all sub-generators G_t namely, a trainable unitary $U_{\theta_t^{(k)}}$. The gen-

erated quantum state of G_t is $\left|\Psi_t^{(k)}(z)\right\rangle = U(\theta_t^{(k)})\left|z^{(k)}\right\rangle$. The discriminator is implemented with a classical deep neural network, i.e., the fully-connected neural network. The input of the discriminator can either be a generated image $\tilde{x}$ or a real image x sampled from $\mathcal{D}$. The output of the discriminator $D(x)$ or $D(\tilde{x})$ is in the range between 0 (label "False") and 1 (label "True").
A loss function $\mathcal{L}(\theta,\gamma)$ is employed to iteratively optimize the quantum generator G and the classical discriminator D during K-iterations. The mathematical form of the loss function is $\mathcal{L}(\theta,\gamma) = \frac{1}{M'}\sum_{i=1}^{M'}[\log\big(D_\gamma(x^{(i)})\big) + \log\big(1-D_\gamma(G_\theta(z^{(i)}))\big)]$, where $x^{(i)}\in\mathcal{D}$, $z^{(i)}\sim P(z)$, M' is the size of mini-batch, and θ and γ are trainable parameters for G and D, respectively. The objectives of the generator and the discriminator are to minimize and maximize the loss, i.e., $\max_\gamma \min_\theta \mathcal{L}(\theta,\gamma)$. The updating rule for G is $\theta^{(k+1)} = \theta^{(k)} - \eta_G * \partial_\theta \mathcal{L}(\theta^{(k)},\gamma^{(k)})/\partial\theta^{(k)}$ and for D is $\gamma^{(k+1)} = \gamma^{(k)} + \eta_D * \partial_\gamma \mathcal{L}(\theta^{(k)},\gamma^{(k)})/\partial\gamma^{(k)}$, where η_G, η_D refers to the learning rate of G and D.

2. *Quantum batch GAN*

The main difference between QPGAN and QBGAN lies in how they optimally use the available quantum resources when $N > \lceil \log M \rceil$. The availabe N qubits are divided into two separate parts: the feature register R_F with N_F qubits and the index register R_I with N_I qubits. The feature register is used to store the information about the features of the examples, while the index register is used to store the information about a batch of generated or real examples. The training examples are encoded using the amplitude encoding method and the resulting state takes the form $\frac{1}{\sqrt{N_e}}\sum_i |i\rangle_I |x_i\rangle_F$, where N_e is the batch size. This encoding method allows to manipulate multiple examples at once and effectively acquire the gradient information, which is the most computationally expensive part of training a GAN.
Quantum batch GAN is composed of a quantum generator, quantum discriminator, and an optimization rule. Both G and D are constructed using PQCs.

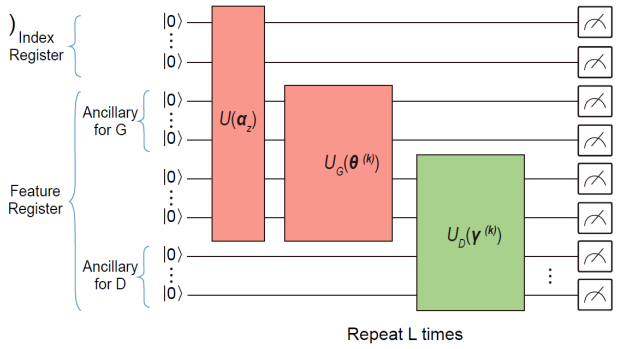


fig: Architecture for quantum batch GAN

In the training procedure, a pertained oracle to generate the latent space $\left|z^{(k)}\right\rangle$, i.e., $\left|z^{(k)}\right\rangle = 2^{-N_I}\sum_i |i\rangle_I \left|z_i^{(k)}\right\rangle_F$. N_F qubits are decomposed into two parts, where the first N_G qubits are used to generate feature vectors and the remaining N_A are used to introduce nonlinearity. Then the quantum generator $U_G(\theta^{(k)}) \in \mathbb{C}^{2^{N_F}\times 2^{N_F}}$ to the generated state, i.e.,

$$\left|\Psi_t^{(k)}(z)\right\rangle = (\mathbb{I}_{2^N}\otimes(U_D(\gamma^{(k)})U_G(\theta^{(k)})))\left|z^{(k)}\right\rangle.$$

Finally, a POVM Π is employed to obtain the output of the discriminator $D(G(z))$, i.e., $D(G(z)) = Tr(\Pi\left|\Psi_t^{(k)}(z)\right\rangle\left\langle\Psi_t^{(k)}(z)\right|)$ with $\Pi = \mathbb{I}_{2^{N-1}}\otimes|0\rangle\langle 0|$. Similarly, $D(x) = \left\langle\Psi_t^{(k)}(x)\right|\Pi\left|\Psi_t^{(k)}(x)\right\rangle$ with $\left|\Psi_t^{(k)}(x)\right\rangle = (\mathbb{I}_{2^N}\otimes U_D(\gamma^{(k)}))\left|x_{(k)}\right\rangle$.

II. METHODS

DCGAN Architecture [1]: Key insights of deep, convolution GAN (DCGAN) architecture:

- Use batch normalization [2] layers in most layers of both the discriminator and the generator, with the two minibatches for the discriminator normalized separately. The last layer of the generator and first layer of the discriminator are not batch normalized, so that the model can learn the correct mean and scale of the data distribution.
- The overall network structure is mostly borrowed from the all-convolutional net [3]. This architecture contains neither pooling nor "unpooling" layers. When the generator needs to increase the spatial dimension of the representation it uses transposed convolution with a stride greater than 1.
- The use of the Adam optimizer rather than SGD with momentum.

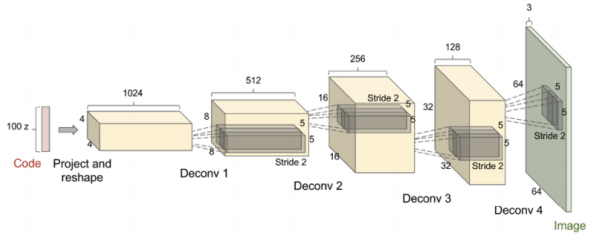


fig: Generator network used by DCGAN

DCGANs are able to generate high quality images when trained on restricted domains of images, such as images of bedrooms. DCGANs also clearly demonstrated that GANs learn to use their latent code in meaningful ways, with simple arithmetic operations in latent space having clear interpretation as arithmetic operations on semantic attributes of the input.

Conditional GAN(CGAN) [4]: In CGAN, labels act as an extension to the latent space z to generate and discriminate images better. In GAN, there is no control over

modes of the data to be generated. The conditional GAN changes that by adding the label y as an additional parameter to the generator and hopes that the corresponding images are generated. We also add the labels to the discriminator input to distinguish real images better.

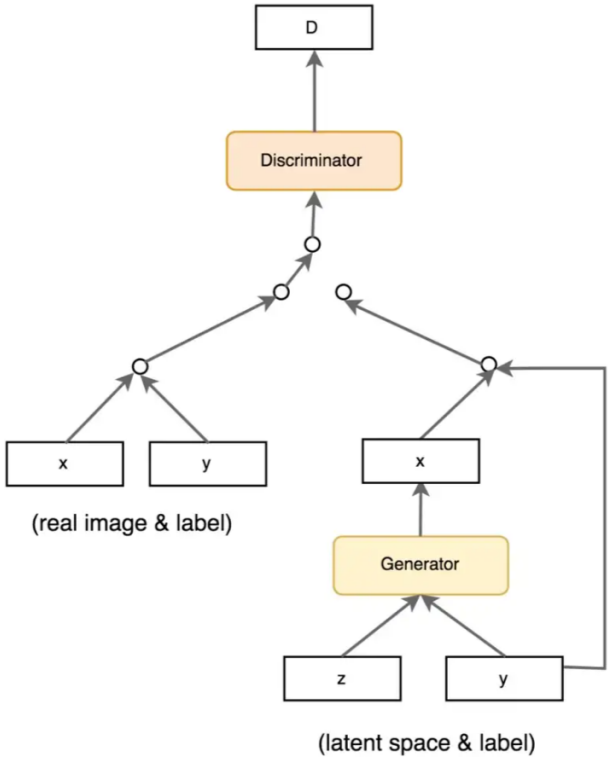


fig: data flow for CGAN

The cost function for CGAN: $\min_G \max_D \mathcal{L}(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x|y))] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z|y)))]$. $D(x|y)$ and $G(z|y)$ demonstrates we are discriminating and generating an image given a label y. In CGAN, we can expand the mechanism to include other labels that the training dataset may provide.

VQGAN Architecture: Variational Quantum Generative Adversarial Networks is a combination of classical Variational Autoencoder (VAE) and QGAN. It utilizes a quantum encoder to map samples to a quantum latent space and approximates the underlying data distribution by learning the parameters of both generator and encoder. The generator produces samples in a quantum representation, while the discriminator performs classification in the classical representation. Methods:

1. Quantum Encoding: Map data samples to a quantum representation, typically using a quantum neural network
2. Training the Encoder: Train the parameters of the encoder to maximize a variational lower bound on the log-likelihood of the data
3. Training the Generator: Train the parameters of the generator to maximize the variational lower bound while fooling the discriminator
4. Training the Discriminator: Train the parameters of the discriminator to maximize its classification accuracy between generated and real samples
5. Evaluation Metrics: Evaluate the performance of the VQGAN using common metrics such as accuracy, reconstruction loss, or Fréchet Inception Distance.

III. IMPLEMENTATION

A. Quantum Patch GAN

The quantum patch GAN [5] under the setting $N < \lceil \log M \rceil$ is composed of a quantum generator, a classical discrimiantor and a classical optimizer.

1. *Quantum Generator*

Quantum sub-generator G_t, analogous to classical generator, receives the input latent state $|z\rangle$ and outputs the generated result $G_t(z)$ [5].

Input Latent state: The latent state is prepared by applying a set of rotation single qubit gates $U_S(\alpha_z(i))$ to the input state $|0\rangle^{\otimes N}$ with $\alpha_z \in \mathbb{R}^N$ and $U_S(\alpha_z(i)) \in \{R_X, R_Y, R_Z\}$, i.e., $|z\rangle = (\bigotimes_{t=1}^{N} U_S(\alpha_z(i)))\,|0\rangle^{\otimes N}$. In the training procedure, the same input state $|z\rangle$ is used as input for all T sub-generators. This is done to ensure that the sub-generators are all working towards the same goal, which is to find the Nash equilibrium of the game being played between the generator and discriminator networks. By using the same input state for all sub-generators, it is guaranteed that the quantum patch GAN will converge to the Nash equilibrium.

Computation model $U_{G_t}(\theta)$: aim is to amp the input state $|z\rangle$ to a specific quantum state that well approximates the target data. Two main components of the model are MPQC(Multilayer parameterized quantum circuit) and the nonlinear transformation. The motivation to use MPQC is for two reasons: first it can be modified to accommodate the limitations of quantum hardware, such as the limited number of quantum gates and circuit depth, and second it has a powerful expressiveness compared to classical circuits, which is beneficial for estimating the real data distribution. The nonlinear transformation is introduced to bridge the gap between the intrinsic properties of quantum computation and the requirements of generative models. Generative models aim to learn a nonlinear mapping from a distribution $P(z)$ to the target data distribution $P_{data}(x)$. However, the trainable unitary in quantum computation, such as MPQC, can only linearly transform the input to the output. Therefore, a nonlinear transformation strategy is necessary for the

quantum generator.
The purpose of introducing an ancillary subsystem in the quantum generator is to enable it to achieve a nonlinear map, which is crucial for a generative model to learn the target data distribution. This is done by adding an additional subsystem to the generator and then tracing it out. This method is similar to ones used in quantum discriminative models. The generator G_t with N qubits is divided into two parts, an ancillary subsystem $\mathcal{A}$ with $N_\mathcal{A}$ qubits and a data subsystem with $N - N_\mathcal{A}$ qubits. The input state $|z\rangle$ is defined in a specific way to allow for the ancillary subsystem to add nonlinearity to the generator's output:

$$|z\rangle = \left(\bigotimes_{i=1, i\in S}^{N_S} R_Y(\alpha_z(i)) \bigotimes_{k=1, k\in[N]\setminus S}^{N-N_S} I_k \right) |0\rangle^{\otimes N}$$

where S is the index set with $S \subset [N]$ and $|S| = N_S$, $R_Y(\alpha_z(i))$ applies to the i-th qubit, identity gate I_k applies to the k-th qubit, and $\alpha_z(i)$ refers to the $i-th$ entry of the vector $\alpha \in \mathbb{R}^{N_S}$ with α being sampled from a predefined distribution. MPQC is denoted as the unitary $U_{G_t}(\theta) \in \mathbb{C}^{2^N \times 2^N}$. The generated state $|\Psi_t(z)\rangle$ for G_t after interacting $U_t(\theta)$ with $|z\rangle$ is $|\Psi_t(z)\rangle = U_{G_t}(\theta)\,|z\rangle$. Taking the partial measurement $\Pi_\mathcal{A}$ on the ancillary subsystem $\mathcal{A}$ of $|\Psi(z)\rangle$, i.e., the post-measurement quantum state $\rho_t(z)$ is

$$\rho_t(z) = \frac{Tr_\mathcal{A}(\Pi_\mathcal{A}\,|\Psi_t(z)\rangle\,\langle\Psi_t(z)|)}{Tr(\Pi_\mathcal{A} \otimes \mathbb{I}_{2^{N-N_\mathcal{A}}}\,|\Psi_t(z)\rangle\,\langle\Psi_t(z)|)}.$$

An immediate observation is that state $\rho_t(z)$ is a nonlinear map for $|z\rangle$, since both the numerator and the denominator are the functions of the variable $|z\rangle$.

Output: Obtained by measuring $\rho_t(z)$ using a complete set of computation bases $\{|j\rangle\}_{j=0}^{2^{(N-N_\mathcal{A})}-1}$. For image generation, the measured result $P(j)$ of the computation basis $|j\rangle$ represents the j-th pixel value for the t-th sub-generator, i.e., $P(J = j) = \mathrm{tr}(|j\rangle\,\langle j|\,\rho_t(z))$. Consequently, $G_t(z) = [P(J = 0), ..., P(J = 2^{(N-N_\mathcal{A})} - 1)]$ and the output for the generator $G(z) = [G_1(z), ..., G_T(z)]$.

2. Discriminator

The discriminator in the quantum patch GAN is a classical neural network, specifically a fully connected neural network (FCNN). The input to the discriminator can either be the real training data x or the generated data $G(z)$. The output of the discriminator is a scalar value between 0 and 1, i.e., $D(x)$, $D(G(z)) \in [0, 1]$ which represents the confidence that the input data is either real (True and labeled as 1) or generated (False and labeled as 0). The ReLU function is used to build the FCNN and the number of hidden layers and neurons can be customized for different tasks.

3. Loss function and optimization rule

Loss function to train quantum patch GAN

$$\min_\theta \max_\gamma \mathcal{L}(D_\gamma(G_\theta(z)), D_\gamma(x))$$

$$:= \mathbb{E}_{x\sim P_{data}(x)}[\log D_\gamma(x)] + \mathbb{E}_{x\sim P(z)}[\log 1 - D_\gamma(G_\theta(z))]$$

The optimization of the loss function in the training process of quantum patch GAN involves adjusting the parameters of both the quantum generator θ and the classical discriminator γ. This optimization process is similar to that of classical GANs. The optimization of the quantum generator is achieved using a zeroth-order method and an automatic differentiation package of PyTorch, while the classical discriminator is optimized using backpropagation. The number of parameters for the quantum generator and the classical discriminator are denoted as $N_G = |\theta|$ and $N_D = |\gamma|$ respectively. The optimization of the quantum generator involves computing the gradients of its i-th parameter $\theta(i)$ with $i \in [N_G]$ by shifting each parameter by $\pi/2$ and evaluating the original expectation twice, keeping the parameters γ fixed. The classical discriminator, on the other hand, is optimized by updating its parameters γ based on the obtained loss, while keeping the parameters of the quantum generator θ fixed.

$$\frac{\partial\mathcal{L}(\theta,\gamma)}{\partial\theta(i)} =$$

$$\frac{\mathcal{L}(\theta(1),..,\theta(i)+\frac{\pi}{2},..,\theta(N_G),\gamma) - \mathcal{L}(\theta(1),..,\theta(i)-\frac{\pi}{2},..,\theta(N_G),\gamma)}{2}$$

Update rule for θ at k-th iteration is

$$\theta^{(k)} = \theta^{(k-1)} - \eta_G \frac{\partial\mathcal{L}(\theta^{(k-1)}, \gamma^{(k-1)})}{\partial\theta^{k-1}}$$

where η_G is the learning rate. Analogous to the classical GAN, we iteratively update parameters θ and γ in total K iterations.

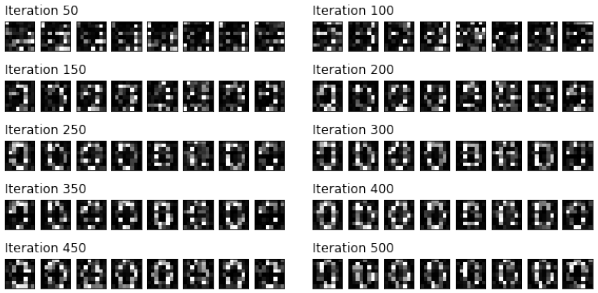


fig: Series of different distributions learned over time as the QPGAN is trained on MNIST handwritten data set

B. Quantum Batch GAN

The quantum patch GAN [5] under the setting $N > \lceil \log M \rceil$ is composed of a quantum generator and discrimiantor. The N-qubits in a quantum system are divided into two parts: the index register $\mathcal{R}_I$ with N_I qubits and the feature register $\mathcal{R}_F$ with N_F qubits. The feature register is further divided into three parts: N_D qubits for generating fake examples, N_{A_G} qubits for conducting nonlinear operations for the generator G, and N_{A_D} qubits for conducting nonlinear operations for the discriminator D. This division allows for easy calculation of mini-batch gradient information through simple measurements.

Input state: To capture the mini-batch gradient information, two oracles U_z and U_x are employed to encode different latent vectors and classical training examples into quantum states, respectively. The mini-batch size is $|B_k| = 2^{N_I}$. For U_z, $U_z : |0\rangle_I^{\otimes N_I} |0\rangle_F^{\otimes N_F} \to 2^{-N_I} \sum_i |i\rangle_I \left|z^{(i)}\right\rangle_F$. With a slight abuse of notation, $z^{(i)}$ refers to the i-th latent vector and $\left|z^{(i)}\right\rangle = \left|\bar{z}^{(i)}\right\rangle |0\rangle^{\otimes N_{A_D}}$, where $\left|\bar{z}^{(i)}\right\rangle \in \mathbb{C}^{2^{N_I+N_{A_G}}}$. Similarly for U_x, $U_x : |0\rangle_I |0\rangle_F \to 2^{-N_I} \sum_i |i\rangle_I \left|x^{(i)}\right\rangle_F$. For a dataset of 2^{N_I} inputs with M features, the complexity of encoding the full dataset into a quantum state using amplitude encoding is $O(2^{N_I} M/(N_I \log(M)))$. This means that the time needed to prepare the quantum state is similar to that of classical machine learning. However, the number of qubits needed for quantum machine learning is $N_I \log(M)$, while classical machine learning requires at least $O(2^{N_I} M)$ bits.

Computation model $U_G(\theta)$: The quantum generator G is built by MPQC $U_G(\theta)$ associated with the nonlinear mappings. $U_G(\theta) \in \mathbb{C}^{2^{N_D+N_{A_G}} \times 2^{N_D+N_{A_G}}}$ only operates with the feature register $\mathcal{R}_F$. In particular to generate fake data, first $\mathbb{I}_{2^{N_I}} \otimes U_G(\theta) \otimes \mathbb{I}_{2^{N_{A_D}}}$ is applied to the input state, i.e., $|\Psi(z)\rangle = 2^{-N_I} \sum_i |i\rangle_I U_G \otimes \mathbb{I}_{2^{N_{A_D}}}(\theta) \left|z^{(i)}\right\rangle$. Then a partial measurement $\Pi_{A_G} = (|0\rangle\langle 0|)^{\otimes N_{A_G}}$ is taken to introduce the nonlinearity. The generated state $|G(z)\rangle$ corresponding to $|B_k|$ fake examples is

$$|G(z)\rangle := 2^{-N_I} \sum_i |i\rangle_I \left|G(z^{(i)})\right\rangle_F$$

$$= \frac{\mathbb{I}_{2^{N_I}} \otimes \Pi_{A_G} \otimes \mathbb{I}_{2^{N_D+2^{N_{A_D}}}} |\Psi(z)\rangle}{\sqrt{Tr(\mathbb{I}_{2^{N_I}} \otimes \Pi_{A_G} \otimes \mathbb{I}_{2^{N_D+2^{N_{A_D}}}} |\Psi(z)\rangle\langle\Psi(z)|)}}$$

In the training procedure, the quantum discriminator is directly applied to operate with the generated state $|G(z)\rangle$.

Computation model $U_D(\gamma)$: Quantum discriminator D, implemented by MPQC $U_D(\gamma)$ associated with the nonlinear operations, aims to output a scalar that represents the averaged classification accuracy. Given a state $|x\rangle$ that represents $|B_k|$ real examples, first $\mathbb{I}_{2^{N_I+N_{A_G}}} \otimes U_D(\gamma)$ is applied to the state $|x\rangle$, i.e., $|\Phi(x)\rangle = 2^{-N_I} \sum_i |i\rangle_I \mathbb{I}_{2^{N_{A_G}}} \otimes U_D(\gamma) \left|x^{(i)}\right\rangle_F$. Then a partial measurement $\Pi_{A_D} = (|0\rangle\langle 0|)^{\otimes N_{A_D}}$ is taken to introduce the nonlinearity. The generated state $|D(x)\rangle$ corresponding to the classification result for $|B_k|$ examples is

$$|D(x)\rangle := 2^{-N_I} \sum_i |i\rangle_I \left|D(x^{(i)})\right\rangle_F$$

$$= \frac{\mathbb{I}_{2^{N-N_{A_D}}} \otimes \Pi_{A_D} |\Psi(x)\rangle}{\sqrt{Tr(\mathbb{I}_{2^{N-N_{A_D}}} \otimes \Pi_{A_D} |\Psi(z)\rangle\langle\Psi(z)|)}}$$

It is noteworthy that, by introducing N_I additional qubits to encode 2^{N_I} inputs as a superposition state, quantum batch GAN could obtain the batch gradient descent of all the 2NI inputs in one training process. This shows that quantum batch GAN has the potential to efficiently process big data.

IV. RESULTS

Hyper-parameter settings of quantum patch GAN in hand-written digits image generation: $N_S = N = 5$ to generate latent states. The number of sub-generators and layers for each U_{G_t} are set as T=4 and L=5, respectively. To compress the depth of the quantum circuits, all trainable single qubits gates are set as R_Y. Equivalently the total number of trainable parameters for quantum generator G is in total $T \times L \times N = 100$. The number of measurements to readout the quantum state is set as 3000. The discriminator is implemented by FCNN with two hidden layers, and the number of hidden neurons for the first and second hidden layer is 64 and 16, respectively. In the training procedure, the learning rates are set as $\eta_G = 0.05$ and $\eta_D = 0.001$.

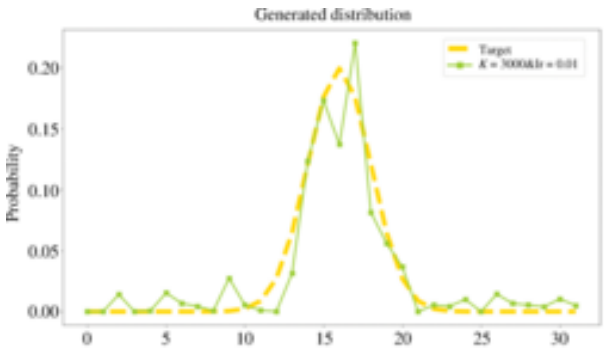


fig: Performance of approximated discrete gaussian ("target" refers to the target Gaussian distribution to be approximated; learning rate=0.01)

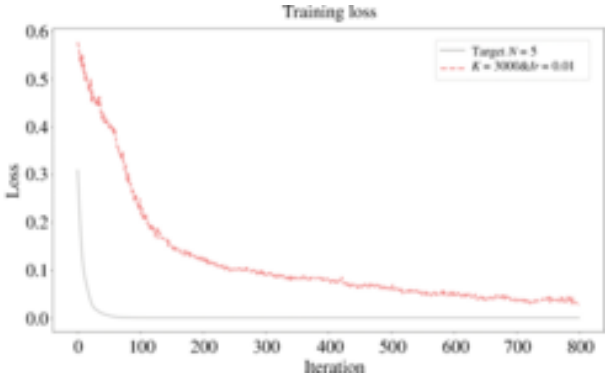


fig: Training loss for handwritten digits image generation

In GANs, the objective function for the generator and the discriminator usually measures how well they are doing relative to the opponent. For example, we measure how well the generator is fooling the discriminator. It is not a good metric in measuring the image quality or its diversity. We look into the Inception Score and Fréchet Inception Distance on how to compare results from different GAN models.

Inception Score(IS): IS is a commonly used metric for evaluating the quality and diversity of the generated images. It uses two criteria in measuring the performance of GAN: the quality of the generated images, and their diversity. Entropy can be viewed as randomness. If the value of a random variable x is highly predictable, it has low entropy. On the contrary, if it is highly unpredictable, the entropy is high. In GAN, we want the conditional probability $P(y|x)$ to be highly predictable (low entropy), i.e., given an image, we should know the object type easily. So we use an Inception network to classify the generated images and predict $P(y|x)$ — where y is the label and x is the generated data. This reflects the quality of the images.
Next we need to measure the diversity of images. $P(y)$ is the marginal probability computed as: $\int_z p(y|x = G(z))dz$. If the generated images are diverse, the data distribution for y should be uniform (high entropy). To combine these two criteria, we compute their KL-divergence and use the equation $IS(G) = exp(\mathbb{E}_{x\sim p_g} D_{KL}(p(y|x)||p(y)))$ to compute IS. One shortcoming for IS is that it can misrepresent the performance if it only generates one image per class. $p(y)$ will still be uniform even though the diversity is low.

Fréchet Inception Distance(FID): FID is a commonly used metric for evaluating the similarity between the generated images and the real images from the training set. In FID, we use the Inception network to extract features from an intermediate layer. Then we model the data distribution for these features using a multivariate Gaussian distribution with mean μ and covariance Σ. The FID between the real images x and generated images g is computed as:

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\Sigma_x + \Sigma_g - 2(\Sigma_x\Sigma_g)^{1/2}),$$

where Tr sums up all the diagonal elements. A lower FID score identifies a better model. FID is sensitive to mode collapse.

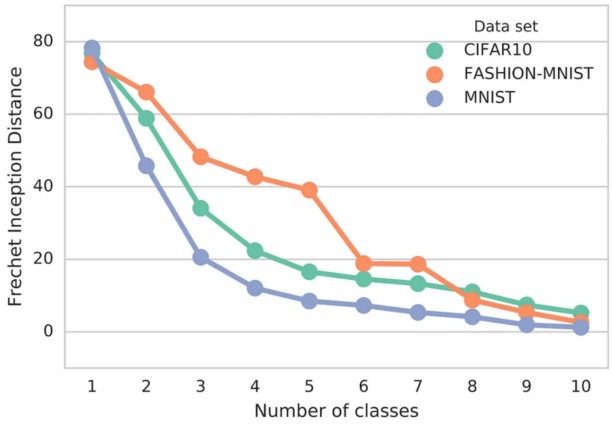


fig: Shows FID is extremely sensitive to mode dropping

As shown above, the distance increases with simulated missing modes. FID is more robust to noise than IS. If the model only generates one image per class, the distance will be high. So FID is a better measurement for image diversity. FID has some rather high bias but low variance. By computing the FID between a training dataset and a testing dataset, we should expect the FID to be zero since both are real images. However, running the test with different batches of training sample shows none zero FID.

	MNIST	FASHION	CIFAR	CELEBA
MM GAN	9.8 ± 0.9	29.6 ± 1.6	72.7 ± 3.6	65.6 ± 4.2
NS GAN	6.8 ± 0.5	26.5 ± 1.6	58.5 ± 1.9	55.0 ± 3.3
LSGAN	7.8 ± 0.6*	30.7 ± 2.2	87.1 ± 47.5	53.9 ± 2.8*
WGAN	6.7 ± 0.4	21.5 ± 1.6	55.2 ± 2.3	41.3 ± 2.0
WGAN GP	20.3 ± 5.0	24.5 ± 2.1	55.8 ± 0.9	30.0 ± 1.0
DRAGAN	7.6 ± 0.4	27.7 ± 1.2	69.8 ± 2.0	42.3 ± 3.0
BEGAN	13.1 ± 1.0	22.9 ± 0.9	71.4 ± 1.6	38.9 ± 0.9
VAE	23.8 ± 0.6	58.7 ± 1.2	155.7 ± 11.6	85.7 ± 3.8

fig: FID score on some of the datasets

Also, both FID and IS are based on the feature extraction (the presence or the absence of features).

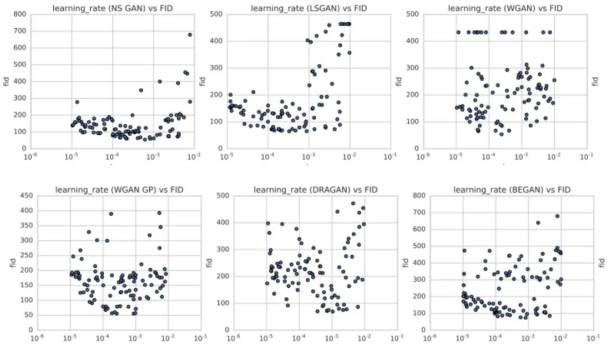


fig: performance (y-axis) between various learning rates (x-axis) under different cost functions

Precision, Recall and F1Score: If the generated images look similar to the real images on average, the precision is high. High recall implies the generator can generate any sample found in the training dataset. A F1 score is the harmonic average of precision and recall.

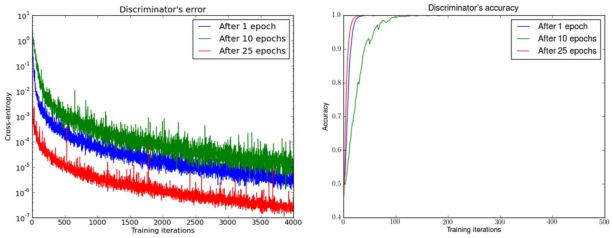


fig: Discriminator's error(left) and accuracy(right) for DCGAN Architecture

V. DISCUSSION

Quantum patch GAN is a variation of the classical Generative Adversarial Network (GAN) architecture that utilizes quantum computing to improve the performance of the generative model. The key feature of a quantum patch GAN is the exploitation of a latent variable input state, denoted as $|z\rangle$, which is analogous to the use of a latent variable in classical GANs.

In classical GANs, the generator network takes a random noise input, referred to as the latent variable, and maps it to a sample in the target data distribution. Similarly, in quantum patch GANs, the generator network takes a quantum state, represented by the vector $|z\rangle$, and maps it to a quantum state that approximates a sample in the target data distribution.

One of the major challenges in training GANs is the difficulty in finding the Nash equilibrium of the non-convex game between the generator and discriminator networks. The use of quantum computing in the quantum patch GAN architecture allows for the efficient optimization of the generator network to converge to the Nash equilibrium. In the training procedure, the same $|z\rangle$ is employed to input into all T sub-generators. Such an operation guarantees that quantum patch GAN is capable of converging to Nash equilibrium, as classical GAN claimed.

The quantum patch GAN architecture also utilizes a technique called virtual batch normalization (VBN) to avoid the problem of output dependency on other inputs in the same minibatch. VBN normalizes each example based on the statistics collected on a reference batch of examples that are chosen once and fixed at the start of training. This reference batch is normalized using only its own statistics, which helps to improve the performance of the generative model.

One of the main advantages of quantum patch GANs is their ability to generate highly complex and detailed images with a high degree of realism. This is a result of the increased capacity of the generator network, which is able to capture more subtle features of the target data distribution. Additionally, quantum patch GANs are able to overcome the problem of mode collapse, which is a common issue in classical GANs where the generator network produces limited variations of the same image.

In the QBGAN, the generator and discriminator networks are quantum circuits. These quantum circuits are implemented using quantum gates, which are the building blocks of quantum computation.

One of the key features of QBGANs is the use of quantum batch normalization (QBN). In classical GANs, batch normalization is used to improve the stability and convergence of the training process. However, in QBGANs, the use of QBN allows for a more efficient and effective training process. This is because QBN uses quantum entanglement to normalize the data, which results in a more robust and accurate representation of the data.

Another advantage of QBGANs is the use of quantum parallelism. In classical GANs, the generator and discriminator networks process one input at a time. In contrast, QBGANs are able to process multiple inputs in parallel, which can result in a significant speedup in the training process.

Despite these advantages, there are also limitations to QBGANs. One limitation is that QBGANs require a significant amount of quantum resources, such as qubits and quantum gates, to train. Additionally, QBGANs are still in the early stages of development, and further research is needed to fully understand their potential and limitations.

VI. CHALLENGES

GANs have seen a lot of success in recent years as a powerful tool for generating new data. However, there are still limitations to GANs that researchers are working to overcome.

Non-convergence [1]: occurs when the model parameters oscillate, destabilize and never converge. This is often due to the inherent instability of the GAN training process, which can make it difficult for the generator and discriminator to find an equilibrium. This can result in the generator producing poor quality samples, or the discriminator never reaching a high level of accuracy. To improve non-convergence, researchers have proposed various techniques such as using a different loss function, adding a gradient penalty, or using a different optimization algorithm.

Mode Collapse [1]: Mode collapse is one of the hardest problems to solve in GAN. A complete collapse is not common but a partial collapse happens often. The objective of the GAN generator is to create images that can fool the discriminator D the most. But let's consider one extreme case where G is trained extensively without updates to D. The generated images will converge to find

the optimal image x^* that fool D the most, the most realistic image from the discriminator perspective. In this extreme, x^* will be independent of z.

$$x^* = \arg\max_x D(x)$$

The mode collapses to a single point. The gradient associated with z approaches zero ($\frac{\delta J}{\delta z} \approx$)0. To improve mode collapse, researchers have proposed techniques such as using a different architecture for the generator or discriminator, or introducing a regularization term to encourage the generator to explore different modes of the data distribution.

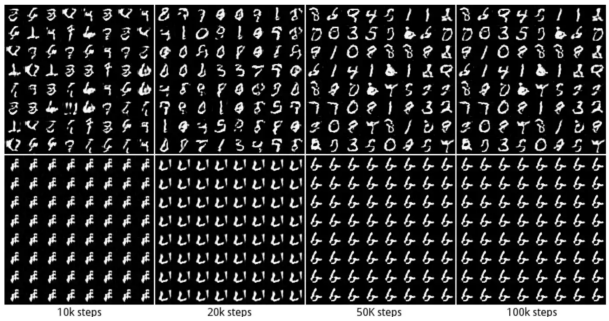


fig: Mode Collapse in MNIST data-set

For example, in MNIST, there are 10 major modes from digit "0" to digit "9". The samples below are generated by two different GANs. The top row produces all 10 modes while the second row creates a single mode only (the digit "6").

Diminished Gradient [6]: This occurs when the generator becomes too strong, causing the discriminator to always predict that the generated samples are real. As a result, the gradients used to update the generator's parameters become small, making it difficult for the generator to learn. This can lead to a lack of diversity in the generated samples and can also cause the model to converge to a suboptimal solution. There are a few ways to improve the issue of diminished gradient in GANs. One method is to use a different loss function, such as the Wasserstein distance, which has been shown to stabilize the training process and reduce the vanishing gradient problem. Another approach is to use techniques such as gradient penalty, which helps to enforce a Lipschitz constraint on the discriminator and thus prevent the generator from becoming too strong. Another method is to use a different generator architecture, such as the use of self-attention mechanism, which has been shown to be more stable during training and to generate more diverse samples.

VII. IMPROVEMENTS

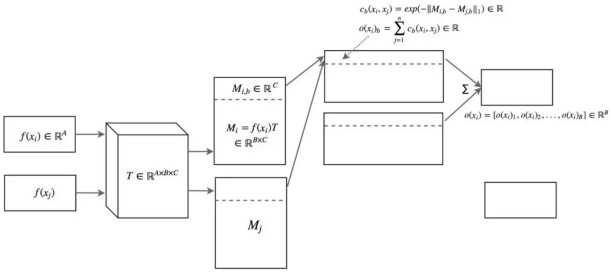
Training GANs requires finding a Nash equilibrium of a non-convex game with continuous, high-dimensional parameters. GANs are typically trained using gradient descent techniques that are designed to find a low value of a cost function, rather than to find the Nash equilibrium of a game. This can lead to the algorithm getting stuck in suboptimal solutions, or failing to converge altogether. Next, we introduce several techniques intended to encourage convergence of the GANs game [7].

Feature Matching: Feature matching changes the cost function for the generator to minimizing the statistical difference between the features of the real images and the generated images. Often, measuring the L2-distance between the means of their feature vectors. Therefore, feature matching expands the goal from beating the opponent to matching features in real images. Here is the new objective function:

$$||\mathbb{E}_{x\sim p_{data}} f(x) - \mathbb{E}_{z\sim p_z(z)} f(G(z))||_2^2$$

where $f(x)$ is the feature vector extracted in an immediate layer by the discriminator. The means of the real image features are computed per minibatch which fluctuate on every batch. It is good news in mitigating the mode collapse. It introduces randomness that makes the discriminator harder to overfit itself. Feature matching is effective when the GAN model is unstable during training.

Minibatch discrimination: When mode collapses, all images created looks similar. To mitigate the problem, we feed real images and generated images into the discriminator separately in different batches and compute the similarity of the image x with images in the same batch. We append the similarity $o(x)$ in one of the dense layers in the discriminator to classify whether this image is real or generated. If the mode starts to collapse, the similarity of generated images increases. The discriminator can use this score to detect generated images and penalize the generator if mode is collapsing. The similarity $o(x_i)$ between the image x_i and other images in the same batch is computed by a transformation matrix T.



where x_i is the input image and x_j is the rest of the images in the same batch. We use a transformation matrix T to transform the features x_i to $M_i (M_i = f(x_i T))$ which is a BC matrix. The similarity between the image i and j, $c_b(x_i, x_j) = exp(-||M_{i,b} - M_{j,b}||_1) \in \mathbb{R}$. The similarity between image x_i and the rest of images in the batch is $o(x_i) = [o(x_i)_1, o(x_i)_2, ..., o(x_i)_B] \in \mathbb{R}^B$ where $o(x_i)_b = \sum_{j=1}^{n} c_b(x_i, x_j) \in \mathbb{R}$. Minibatch discrimination allows us to generate visually appealing samples very

quickly, and in this regard it is superior to feature matching.

One-sided label smoothing: Deep networks may suffer from overconfidence. For example, it uses very few features to classify an object. To mitigate the problem, deep learning uses regulation and dropout to avoid overconfidence. In GAN, if the discriminator depends on a small set of features to detect real images, the generator may just produce these features only to exploit the discriminator. The optimization may turn too greedy and produces no long term benefit. To avoid the problem, the discriminator is penalized when the prediction for any real images go beyond 0.9 (D(real image)¿0.9). This is done by setting target label value to be 0.9 instead of 1.0.

Historical averaging: The model parameters for the last t models is tracked. Alternatively, we update a running average of the model parameters if we need to keep a long sequence of models. Each player's cost is modified to include a term $||\theta - \frac{1}{t}\sum_{i=1}^{t}\theta[i]||^2$, where θ_i is the value of the parameters at past time i. For GANs with non-convex object function, historical averaging may stop models circle around the equilibrium point and act as a damping force to converge the model.

Virtual batch normalization(VBN): The use of batch normalization is a widely used technique in training neural networks to improve their optimization. It was found to be particularly effective in the training of GANs, specifically DCGANs. However, using batch normalization causes the output of a neural network for a given input example x to be highly dependent on other inputs x' in the same minibatch. To address this issue, an alternative method called VBN was introduced, in which each example x is normalized based on the statistics collected from a reference batch of examples that is chosen once and fixed at the start of training, and on x itself. This reference batch is only normalized using its own statistics. However, VBN is computationally expensive as it requires running forward propagation on two minibatches of data. As a result, it is typically only used in the generator network.

Spectral Normalization: It is a weight normalization that stabilizes the training of the discriminator. It controls the Lipschitz constant of the discriminator to mitigate the exploding gradient problem and the mode collapse problem. The concept is based heavily on maths but conceptually, it restricts the weight changes in each iteration and not over depending on a small set of features in distinguishing images by the discriminator.

Quantum GANs: By using quantum computing, QGANs are able to overcome the limitations of classical GANs, such as mode collapse, and achieve higher levels of accuracy and efficiency. One of the key advantages of QGANs is that they can leverage the properties of quantum computing, such as superposition and entanglement, to generate high-quality samples from the target distribution. This is achieved by using quantum algorithms, such as the Quantum Approximate Optimization Algorithm (QAOA), to optimize the parameters of the generator and discriminator networks. Additionally, QGANs can leverage the power of quantum computing to perform complex computations, such as quantum machine learning, which can improve the performance of GANs.

VIII. APPLICATIONS

The application of Generative Adversarial Networks (GANs) has been widely explored in various fields such as computer vision, natural language processing, and speech synthesis.

Create Anime characters [8]: GAN can autogenerate and colorize Anime characters. The generator and the discriminator composes of many layers of convolutional layers, batch normalization and ReLU with skip connections.

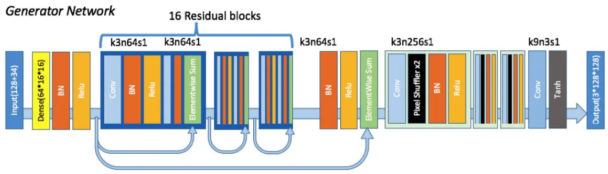


fig: Generator Architecture

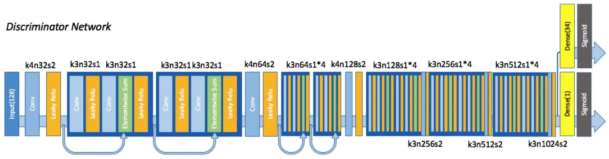


fig: Discriminator Architecture

Pose Guided Person Image Generation [9]: With an additional input of the pose, an image can be transformed into different poses. The design composes of a 2-stage image generator and a discriminator. The generator reconstruct an image using the meta-data (pose) and the original image. The discriminator uses the original image as part of the label input to a CGAN design.

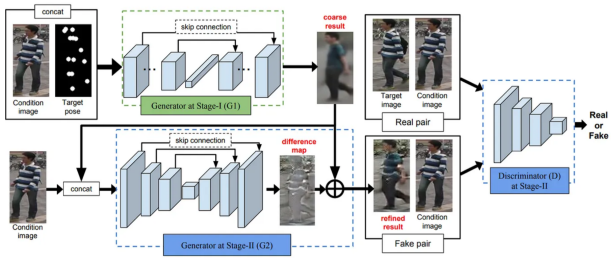


fig: Pose Guided Person Image Generation

PixelDTGAN [10]: Suggesting merchandise based on celebrity pictures has been popular for fashion blogger

and e-commerce. PixelDTGAN creates clothing images and styles from an image.

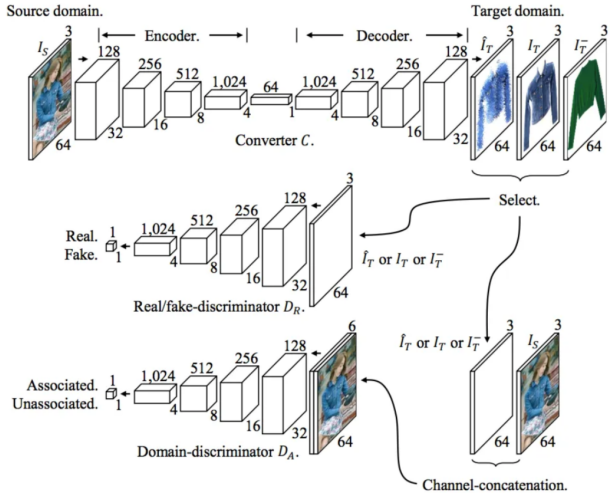


fig: Architecture for pixel-level domain transfer

Super Resolution [11]: Creating super-resolution images from the lower resolution. This is one area where GAN shows very impressive result with immediate commercial possibility. It composes of many layers of convolutional layer, batch normalization, advanced ReLU and skip connections.

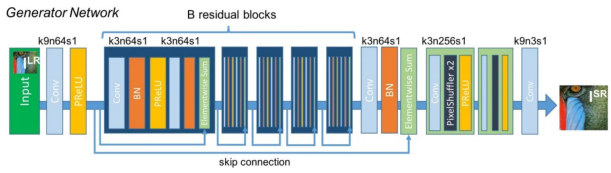


fig: Generator network for SRGAN

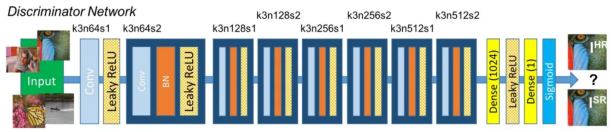


fig: Discriminator network for SRGAN

Progressive GANs [12]: It applies the strategy of divide-and-conquer to make training much feasible. Layers of convolution layers are trained once at a time to build images of 2×2 resolution.

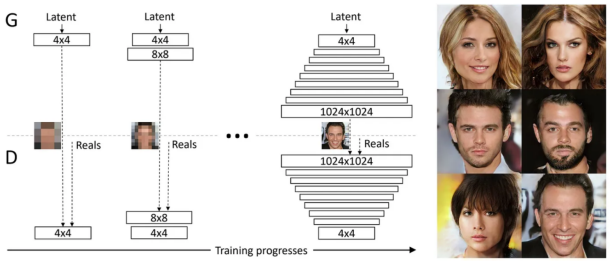


fig: Progressive growing of GANs

Training starts with both the generator(G) and discriminator(D) having a low spatial resolution of 4×4 pixels. As the training advances, incrementally layers are added to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $N \times N$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably.

Text to image: Input is a sentence based on which multiple images fitting the description is generated.

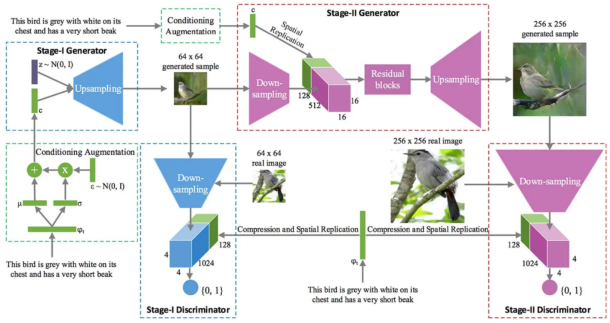


fig: Architecture of StackGAN

The Stage-I generator draws a low resolution image by sketching rough shape and basic colors of the object from the given text and painting the background from a random noise vector. The Stage-II generator generates a high resolution image with photo-realistic details by conditioning on both the Stage-I result and the text again [13]. Another popular implementation:

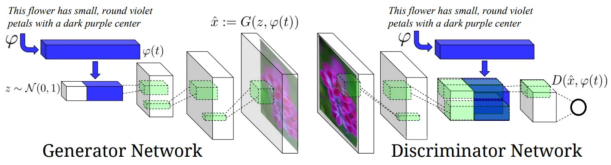


fig: Text-conditional GAN architecture

Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing [14].

DTN:Creating emoji from pictures.

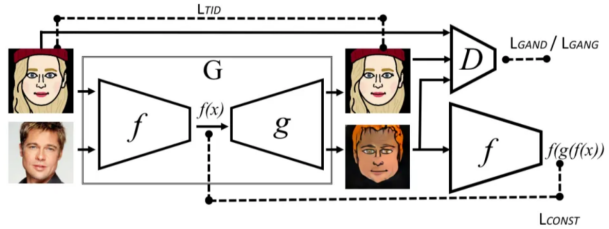


fig: Domain Transfer Network
Losses are drawn with dashed lines, input/output with solid lines. After training, the forward model G is used for the sample transfer [15]

Object Detection [16]: The generator is a deep residual network which takes the features with fine-grained details from lower-level layer as input and passes them to

3×3 convolutional filters followed by 1×1 convolutional filters to increase the feature dimension to be aligned with that of "Conv5". Then B residual blocks each of which consists of convolutional layers followed by batch normalization and ReLU activation are employed to learn the residual representation, which is used to enhance the pooled features from "Conv5" for small objects to super-resolved representation through element-wise sum operation. The discriminator takes the features of large object and the super-resolved representation of small object as inputs and splits into two branches. The adversarial branch consists of three fully connected layers followed by sigmoid activation, which is used to estimate the probability that the current input representation belongs to that of real large object. The perception branch consists of two fully connected layers followed by two output sibling layers, which are used for classification and bounding box regression respectively to justify the detection accuracy benefiting from the generated super-resolved representation.

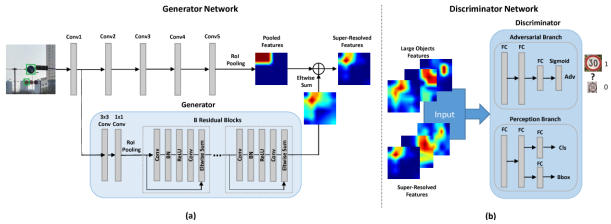


fig: Perceptual GAN

Music generation [17]: GAN can be applied to non-image domain, like composing music.

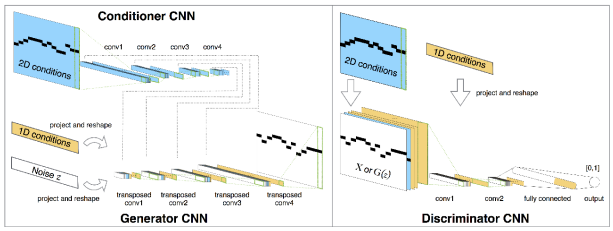


fig: System diagram of the proposed MIdiNet model for symbolic-domain music generation

Anomaly Detection [18]: The preprocessing step includes extraction and flattening of the retinal area, patch extraction and intensity normalization. Generative adversarial training is performed on healthy data and testing is performed on both, unseen healthy cases and anomalous data.

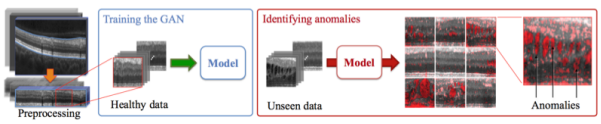


fig: Anomaly detection framework

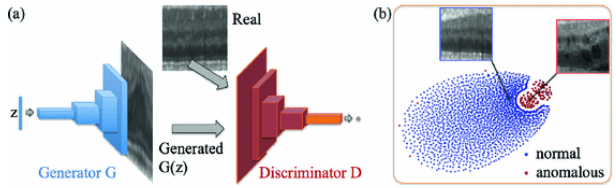


fig: (a)DC-GAN (b)t-SNE embedding of normal (blue) and anomalous (red) images on the feature representation of the last convolutional layer (orange in (a)) of the discriminator

QGANs have been used for tasks such as quantum state generation where they have been used to generate new quantum states that can be used as training data for quantum neural networks, quantum state classification, quantum state distillation and have been used to generate new quantum circuits that can be used for quantum computing tasks such as quantum error correction and quantum cryptography.

IX. CONCLUSIONS

Generative Adversarial Networks (GANs) have proven to be a powerful tool in the field of deep learning and computer vision. The basic concept of a GAN involves training a generator to generate samples that are indistinguishable from real data and a discriminator to distinguish between real and generated samples. The generator and discriminator then compete with each other to improve their respective models. The classical GANs use deep neural networks for both the generator and discriminator and have been successful in various tasks such as image generation and translation.

However, with the advent of quantum computing, researchers have started exploring the possibility of extending GANs to quantum domain, leading to the development of Quantum GANs (QGANs). The key idea behind QGANs is to replace the classical neural networks with quantum circuits, which can offer exponential speedups in certain tasks compared to classical algorithms. In the context of GANs, QGANs can achieve faster convergence, improved stability, and enhanced sample quality compared to classical GANs.

In this report, we discussed the basic idea of classical GANs and how they can be extended to quantum domain by replacing the classical neural networks with quantum circuits. We also discussed some of the recent advancements in QGANs, such as the quantum patch GAN, which leverages quantum circuits to generate high-dimensional feature vectors and quantum batch GAN. Our discussion showed that quantum computing holds a great promise for the future of GANs and can bring about a significant improvement over the classical GANs in terms of performance and efficiency. In future work, it would be interesting to investigate the use of other quantum-inspired GAN architectures, such as Quantum-inspired GANs and Quantum-enhanced GANs.

In conclusion, GANs are a powerful tool for generating synthetic data and have a wide range of applications in computer vision and deep learning. While classical GANs have been successful in various tasks, the recent advancements in quantum computing have led to the development of QGANs, which have the potential to significantly enhance the performance and efficiency of GANs. With the rapid development of quantum computing, we expect that QGANs will play an increasingly important role in various fields in the near future.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my mentors, Aditya Sriram and Siddhant Midha, for their guidance and support throughout the project and writing process. Their invaluable insights and expertise were instrumental in helping me understand the complexities of Quantum Machine Learning especifically Quantum GANs and in shaping this report.

[1] I. Goodfellow, arXiv preprint arXiv:1701.00160 (2016).
[2] S. Ioffe and C. Szegedy, in *International conference on machine learning* (pmlr, 2015) pp. 448–456.
[3] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, arXiv preprint arXiv:1412.6806 (2014).
[4] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017) pp. 1125–1134.
[5] H.-L. Huang, Y. Du, M. Gong, Y. Zhao, Y. Wu, C. Wang, S. Li, F. Liang, J. Lin, Y. Xu, *et al.*, Physical Review Applied **16**, 024051 (2021).
[6] M. Arjovsky and L. Bottou, arXiv preprint arXiv:1701.04862 (2017).
[7] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, Advances in neural information processing systems **29** (2016).
[8] Y. Jin, J. Zhang, M. Li, Y. Tian, H. Zhu, and Z. Fang, arXiv preprint arXiv:1708.05509 (2017).
[9] L. Ma, X. Jia, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool, Advances in neural information processing systems **30** (2017).
[10] D. Yoo, N. Kim, S. Park, A. S. Paek, and I. S. Kweon, in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14* (Springer, 2016) pp. 517–532.
[11] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017) pp. 4681–4690.
[12] T. Karras, T. Aila, S. Laine, and J. Lehtinen, arXiv preprint arXiv:1710.10196 (2017).
[13] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, in *Proceedings of the IEEE international conference on computer vision* (2017) pp. 5907–5915.
[14] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, in *International conference on machine learning* (PMLR, 2016) pp. 1060–1069.
[15] Y. Taigman, A. Polyak, and L. Wolf, arXiv preprint arXiv:1611.02200 (2016).
[16] J. Li, X. Liang, Y. Wei, T. Xu, J. Feng, and S. Yan, in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017) pp. 1222–1230.
[17] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, arXiv preprint arXiv:1703.10847 (2017).
[18] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, in *Information Processing in Medical Imaging: 25th International Conference, IPMI 2017, Boone, NC, USA, June 25-30, 2017, Proceedings* (Springer, 2017) pp. 146–157.