Applied Quantum Algorithms
Quantum Generative Adversarial Network

Dinu Catalin-Viorel

March 2023

1 Generative Adversarial Networks

Generative (unsupervised) learning problems are defined as following:
Given an initial data set $data = (x)_{i=1}, x_i \in \Omega \subset \mathbb{R}$ sampled according to an unknown distribution P_{data}, we want to sample new instances from that distribution. In order to tackle this problem, we are going to use GANs
Generative Adversarial Networks (GANs) 2 are generative models composed of

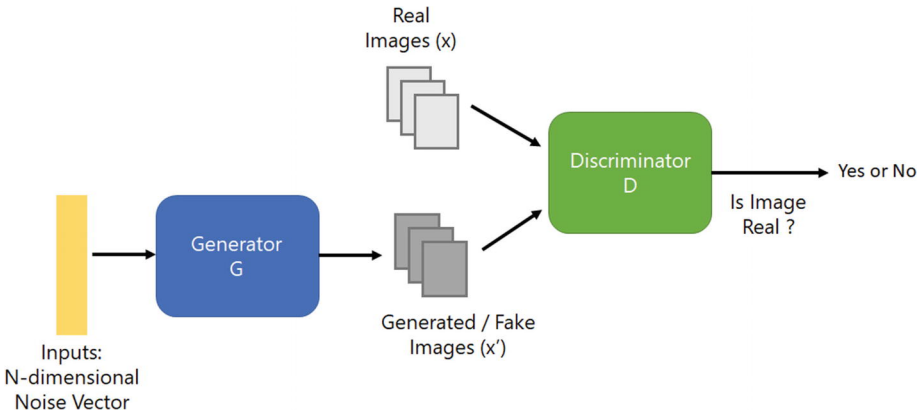


Figure 1: Generative Adversarial Network

2 components that play a min-max game.:

- Generator: -The purpose of this component is to generate new data point starting from a noise vector that follows some known distribution:

$$G : \mathbb{R}^m \to \mathbb{R}^n$$
$$z \sim P_z$$
$$G(z) \to \text{new data point}$$

- Discriminator: -The purpose of this component is to distinguish images from the dataset and the generated ones.

$$D : \mathbb{R}^n \to [0, 1]$$
$$D(x)s = \mathbb{P}[x \sim P_{data}], \text{ probaility that the input is a real data point}$$

So the algorithm resumes the idea of the competition between the two components. The discriminator wants to become better at distinguishing between real data points and generated data points, while the generator is trying to become better at fooling the discriminator by creating more realistic data points. So, we have the following min-max game:

$$\min_G \max_D V(G, D)$$

$$V(G, D) = \mathbb{E}_{x \sim P_{data}(x)}[log(D(x))] + \mathbb{E}_{z \sim P_z(z)}[log(1 - D(G(z)))]$$

We consider G_θ, D_γ the parametrised version of generator(G) and discriminator(D), respectively. So, we split the above min-max game into two loss functions:

$$\max_\gamma \ \mathcal{L}_D(\theta, \gamma)$$
$$\mathcal{L}_D(\theta, \gamma) = \mathbb{E}_{x \sim P_{data}(x)}[log(D_\gamma(x))] + \mathbb{E}_{z \sim P_z(z)}[log(1 - D_\gamma(G_\theta(z)))]$$
$$\max_\theta \ \mathcal{L}_G(\theta, \gamma)$$
$$\mathcal{L}_G(\theta, \gamma) = \mathbb{E}_{z \sim P_z(z)}[log(D_\gamma(G_\theta(z)))]$$

Sequential training procedure for Generative Adversarial Networks:

1. Lock the generator parameters
2. Create a data set composed of equal parts of generated samples and real samples
3. Train the discriminator on that data set using gradient ascent:

$$\gamma = \gamma + \alpha_D \nabla_\gamma \mathcal{L}_D(\theta, \gamma)$$
$$\alpha_D \to \text{ learning rate of the discriminator}$$

4. Lock the discriminator parameters
5. Create a data set composed of only generated samples
6. Train the discriminator on the data set by gradient ascent using the locked discriminator to compute the performance of the generator

$$\theta = \theta + \alpha_G \nabla_\theta \mathcal{L}_G(\theta, \gamma)$$
$$\alpha_D \to \text{ learning rate of the generator}$$

Either of the two components can be implemented both as a quantum circuit or a classical algorithm. For our method presented in this report, we work with a Quantum Circuit Born Machine as our generator and a Convolutional Neural Network as our discriminator (binary classifier). Moreover, we consider the problem of generating images for the rest of the report.

2 Quantum Generator

Similar to [**?**], we want to develop a resource-efficient generator. Using only one quantum circuit to generate the whole high dimension data (like images) would necessitate a high number of qubits. So, in order to reduce the number of qubits needed for the quantum circuit, we partitioned the Generator (G_θ) into T different generators, each of which will generate unique parts of the output data point.

$$\begin{aligned} G &: \mathbb{R}^m \to \mathbb{R}^n \\ G_\theta &= \{G^1_{\theta^1}, ..., G^T_{\theta^T}\} \\ G^t_{\theta^t} &: \mathbb{R}^m \to \mathbb{R}^{\frac{n}{T}} \\ \theta &= \{\theta^1, ..., \theta^T\} \end{aligned}$$

Each $G^t_{\theta^t} (t \in 1..T)$ represents a different quantum circuit that works with m different qubits from which we consider m_a to be ancilary qubits (we need $\frac{n}{T} < 2^{m-m_a}$). Those generators are composed of two parts:

1. Generating the input state $|z\rangle$. This part is the same for all generators($G^t_{\theta_t}, t \in 1..T$). Given $z \in \mathbb{R}^m, z \sim P_z$, we want to compute $|z\rangle$ as following:

$$|z\rangle = U(z)\,|0^m\rangle = (\bigotimes_{i=1}^{m} R_y(z_i))\,|0^m\rangle$$

2. Parametrised quantum circuit composed of several similar blocks (d different blocks). Each of those building blocks consists of 2 parts, a trainable unitary $U_P(\theta)$ and an entanglement layer U_E:

$$U_E = \bigotimes_{i=1}^{m-1} CZ_{i,i+1}$$

$$S \subseteq \{R_x, R_y, R_z\}$$

$$U_P(\theta) = \bigotimes_{i=1}^{m} \prod_{U_j \in S} U_j(\theta_{i,j})$$

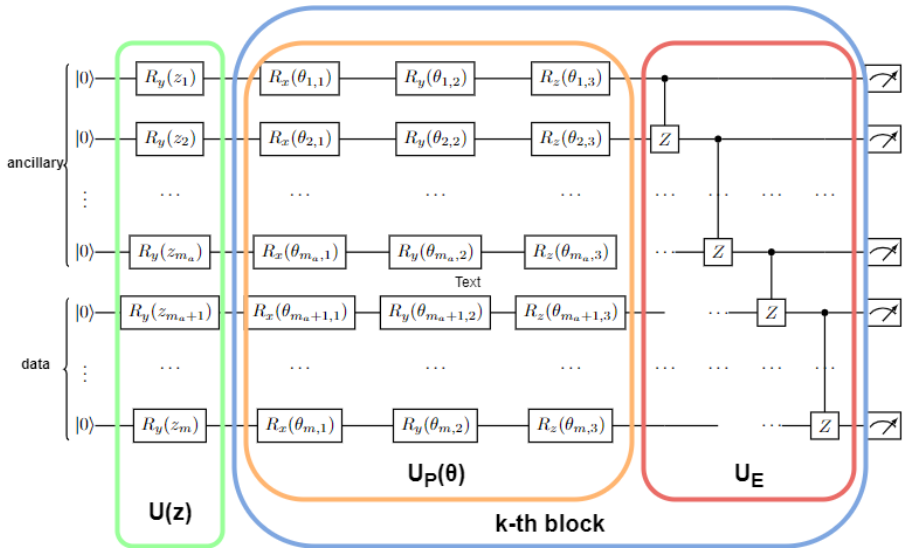


Figure 2: Quantum circuit for $G^t_{\theta_t}$

So the final state of the generator $G^t_{\theta_t}$ is:

$$\left|\psi_t(\theta^t)\right\rangle = \left(\prod_{k=1}^{d}(U_{\text{block}}(\theta^t_k))\right)|z\rangle\,, \text{ where:}$$
$$U_{\text{block}}(\theta_k) = U_P(\theta_k)U_E, \forall k$$

In order to achieve a non-linear mapping, we take the partial measurement $\Pi_{\mathcal{A}}$ on the ancillary qubits system $\mathcal{A}$. So the state of the system after the measurement is as follows:

$$\rho_{t,\theta}(z) = \frac{Tr_{\mathcal{A}}(\Pi_{\mathcal{A}} \otimes \mathbb{I}\left|\psi_t(\theta^t)\right\rangle\left\langle\psi_t(\theta^t)\right|)}{\left\langle\psi_t(\theta^t)\right|\Pi_{\mathcal{A}} \otimes \mathbb{I}\left|\psi_t(\theta^t)\right\rangle}$$

In our experiments we set $\Pi_{\mathcal{A}} = |0\rangle\langle 0|^{m_a}$ So in the end the generator $G^t_{\theta_t}$ will output:

$$G^t_{\theta_t}(z) = [P(0), ..., P(2^{m-m_a}-1)], where:$$
$$P(k) = Tr(|k\rangle\langle k|\,\rho_{t,\theta}(z))$$

We want to generate images, but $P(k) \in [0,1] and \sum_{k=0}^{2^{m-m_a}-1} P(k) = 1$. Meanwhile, any partition of the image contains pixel values between [0,1] but their sum can be way above 1, so we decided to transform the output of the generator as:

$$G^t_{\theta_t}(z) = \left[\frac{P(0)}{\max_k P(k)}, ..., \frac{P(2^{m-m_a}-1)}{\max_k P(k)}\right], where:$$
$$P(k) = Tr(|k\rangle\langle k|\,\rho_{t,\theta}(z))$$

We want to train the generator using the gradient ascent, so we used the gradient shift method to compute the gradients:

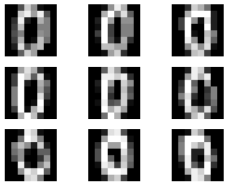
$$\frac{\partial L_G(\theta,\gamma)}{\partial \theta_i} = \frac{L_G(\theta_0,...,\theta_i+\frac{\pi}{2},...\theta_l,\gamma) - L_G(\theta_0,...,\theta_i-\frac{\pi}{2},...\theta_l,\gamma)}{2}$$

3 Experiments

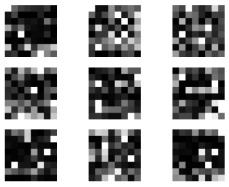
We tried two configurations in the experiment, but they failed to converge to meaningful results.
For the experiment, as the discriminator, we used a Convolutional Neural Net with a convolutional layer, a max-pull layer and a fully connected layer with only one output.

First, we tried to train on the 8x8 images of handwritten 0s. We partitioned the image into eight elements resulting in 8 different sub-generators. For this experiment, we used all three rotations R_x, R_y, R_z for each generator's parametrised part of the blocks. The number of blocks is 3. The number of qubits used was 5, with one ancillary qubit. We trained the generator in 2 different configurations:

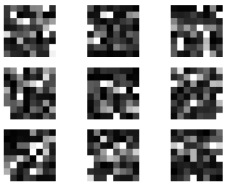
- Simulation: we simulated the quantum circuits for 1000 repetitions estimating the probability distribution based on the observed values.
- State Vector: we computed the state vector from which we computed the real probabilities using the Born rule



(a) Initial dataset of 8x8 images of 0s



(b) Generated images simulating the circuit 1000 times

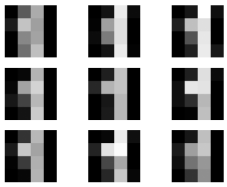


(c) Generated images using the final state vector of the circuit

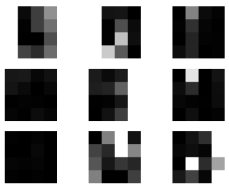
Secondly, we used a dataset of 4x4 images of handwritten 1s to reduce the training time. We partitioned the image into four different sub generators and for each generator's parametrised part of the three used blocks we used only R_y gate, considerably reducing the number of parameters of the quantum circuit. The number of qubits used was 4, with one ancillary qubit.
We trained the generator in 2 different configurations:

- Simulation: we simulated the quantum circuits for 1000 repetitions estimating the probability distribution based on the observed values.

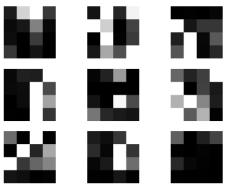
- State Vector: we computed the state vector from which we computed the real probabilities using the Born rule



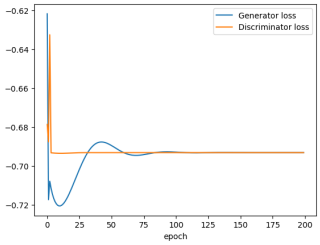
(a) Initial dataset of 4x4 images of 1s



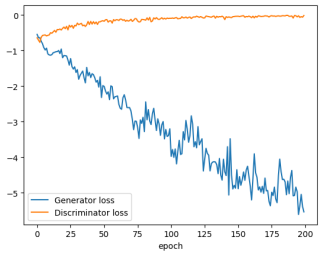
(b) Generated images simulating the circuit 1000 times



(c) Generated images using the final state vector of the circuit



(a) Generator loss and discrimination loss over time simulating the circuit 1000 times. It seems that the generator and the discriminator got into an equilibrium prematurely without the discriminator getting properly trained.



(b) Generator loss and discrimination loss over time using the final state vector of the circuit. We can observe how the discriminator becomes too good at his job and the generator failed to catch up.

4 Discussion and future work

We failed to train the models due to computational resource limitations. For 200 epochs for the 8x8 image, it took over four days of run time to train the algorithm for each configuration (Simulation or State Vector). Even though our algorithm didn't converge in 200 epochs, we want to try training for a longer period. If the algorithm converges, an interesting addition would be to study how different types of noise would affect the learning procedure.
Computing gradients for the generator is the element of the algorithm that consumes the most resources. As future work, we would like to try gradient-free methods for training the generator, like evolutionary strategies (Covariance matrix adaptation evolution strategy).