Exploring Generative Adversarial Networks

Siheng Huang, Yicong Li, Yunhao Li

May 2023

1 Introduction

Generative Adversarial Nets (GAN) stands for Generative Adversarial Networks, which is a type of neural network that is used for unsupervised learning tasks such as image and video generation, data synthesis, and data augmentation.

The GAN architecture is composed of two parts: a generator and a discriminator. The generator produces synthetic data samples, while the discriminator evaluates whether the samples are real or fake. During training, the generator tries to produce realistic samples that can fool the discriminator, while the discriminator tries to distinguish between real and fake samples.

The adversarial nature of GANs makes them particularly effective at generating realistic data samples. By iteratively improving the generator and discriminator, GANs can produce samples that are indistinguishable from real data. This has many practical applications, such as in computer vision, where GANs can be used to generate high-quality images or to fill in missing data in incomplete images.

Since their introduction in 2014 by Ian Goodfellow and his colleagues, GANs have become one of the most popular deep learning architectures, and have led to many exciting advancements in the field of machine learning.

2 Generator

2.1 Recall - classic machine learning models

As we learned before with the classical machine learning model, after data x is input, a mod will output a value y.

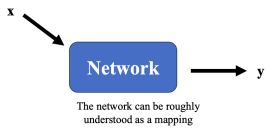


Figure 1: classical machine learning mechanism

In a neural network, x can be a set of data, a set of pictures, or even a set of voice messages, and y can be a category or a set of sequences.

2.2 Special - A random variable z will be added

Now, we add a random variable to the network.

z comes from another simple distribution. At this point, the network's input is not just a set of x, but x and z.

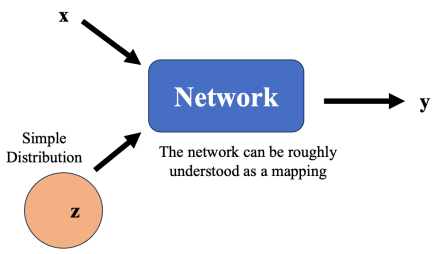


Figure 2: Special - A random variable z will be added

- The special thing about z is that it is not fixed, every time the network is used, a z will generate randomly.

- The restriction of simple distribution is that it must be simple enough, i.e., we know how its formulation, and we can generate samples from this distribution

2.3 With different z, the output y varies

If we use different value z to the network, the network will output different y as well.

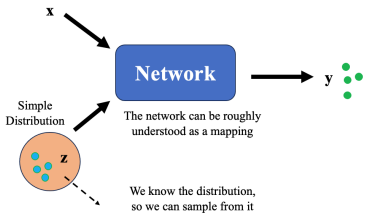


Figure 3: With different z, the output y varies

With a large number of z input to the network, the output is no longer a set of single values, but a complex distribution. (we could use a distribution to describe outputs)

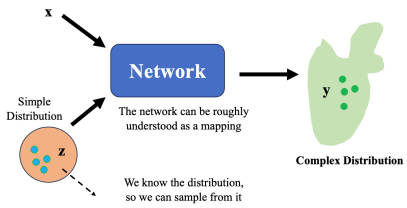


Figure 4: the output of the network is no longer a single value, but a complex distribution

To sum up, a network that can output a complex distribution called **generator**.

2.4 Why we should use a generator?

Example: Frames Prediction in an elf game [1]

The task is to predict the next frame of an elf when it reaches a corner.

[1] `https://github.com/dyelax/Adversarial_Video_Generation`

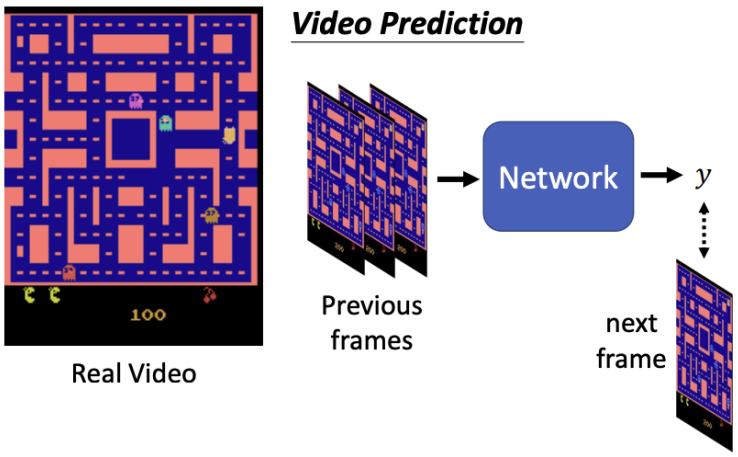


Figure 5: predict the next frame

Actually, it is not hard to reach in a classical machine learning model, you should only give the network enough previous frames, and then the network can be trained so that its output y is as close to our goal as possible,

- Give your network the previous frames
- The network should predict the elf in the corner should be to the left or to the right
- And its output will be a new frame(next moment's frame)

However, may occur some problems.

If you follow this method of training network, that is, supervisor learning training, the elf will split into two at the corner, and sometimes even disappear as they walk.

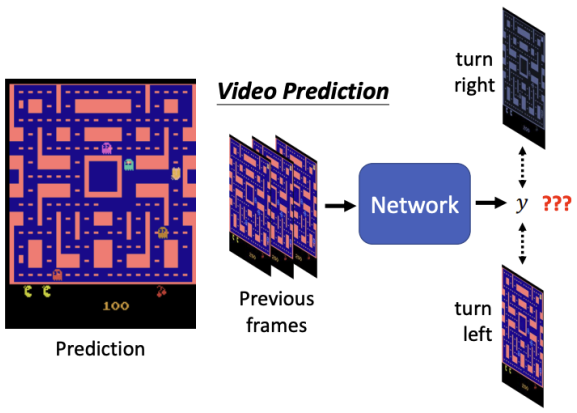


Figure 6: the elf split as he walked to the corner

Why did the elf split as it walked?

For such a network, sometimes the same input, i.e. the same corner, the elf may go to the left or to the right. These two possibilities exist simultaneously in the training set.

When you are training your network, it is given the instruction to turn left given a piece of the training set

and turn right given another piece of the training set. When both are present in a training set, your network will learn both of the information and in this way, it will balance the result → turn left and right at the same time(split into 2)

How to deal with it?

- Instead of generating a single output, make the machine output probabilistic by generating a probability distribution;
- When we add a set of z to this network, its output can be a distribution;
- That is, its output contains the possibility to turn left as well as right

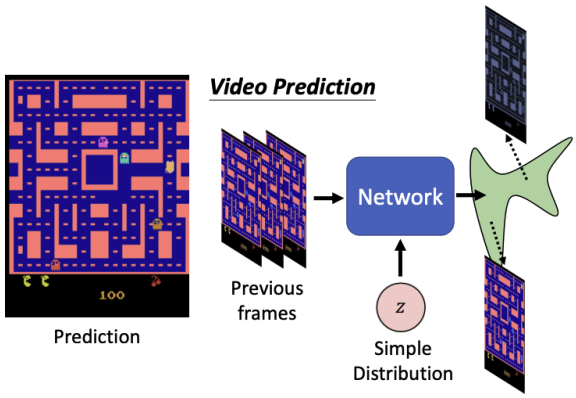


Figure 7: make the network output a set of probability

2.5 In what scenarios do we need a generator?

2.5.1 When our mission requires creativity

How to define creativity?

- The creativity of a generator refers to its ability to generate new, unique, and creative data.

- The data generated by the generator should be different from the existing data in order to provide people with new and unknown information.

- **In statistics, we regard a generator's ability to generate complex distributions as a result of its creative**

2.5.2 Some Examples

- image compositing
- style transfer
- video generation

Looking at these examples, generators can draw some pictures with just a simple request(figure 8)

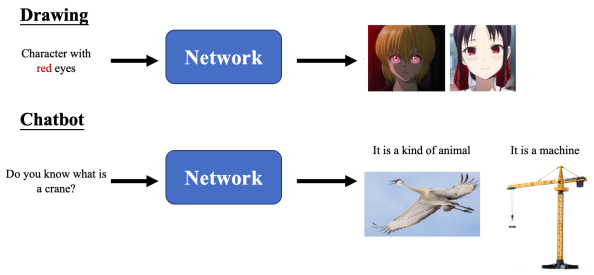


Figure 8: some example while using generator

3 Discriminator

3.1 Unconditional generation

Here a series of vectors (from a normal distribution) is input to the network, and then the network generates a series of cartoon images(figure 9)

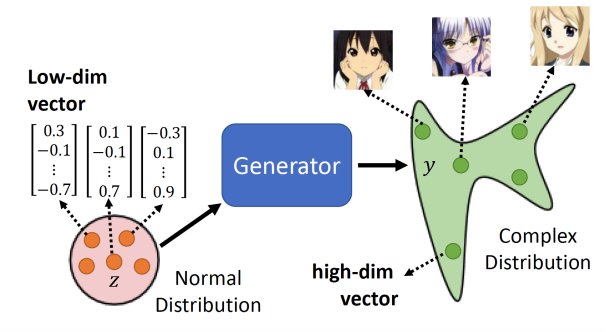


Figure 9: example: Unconditional Generation

At this point, it is mandatory that whatever z is input, the output is cartoon characters.

- About normal distribution

  In fact, the distribution on this side just needs to be simple enough (could be uniform distribution, Poisson distribution, etc), because the generator will figure out how to output cartoon characters(complex distribution).

3.2 discriminator

The special thing in GAN is that in addition to the generator, there is an additional discriminator to be trained.

3.2.1 What is the usage of a discriminator?

The purpose of the discriminator is to identify whether a picture is a cartoon character or not. The input of the discriminator is an image, and the output is a scalar value. The neural network itself can be thought of as a function.

Example: discriminate the odds of the following images being cartoon characters

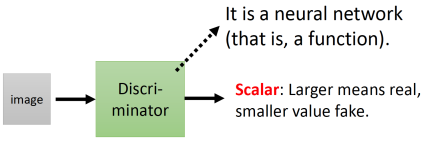


Figure 10: the usage of a discriminator

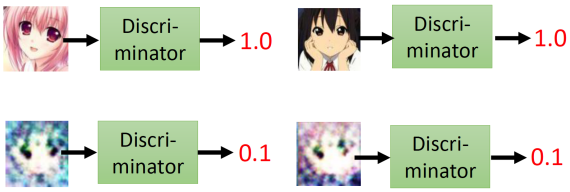


Figure 11: discriminate whether the input image is a cartoon character

3.2.2 Why we should discriminator?

Example When we want to copy an image from an original one, we want to evaluate the similarity with the original, then we need the discriminator.

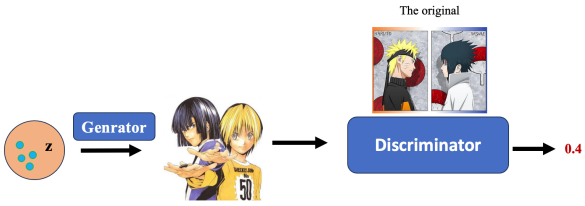


Figure 12: evaluate the similarity with the original

4 Adversary

4.1 Non-cooperative game(Nash Equilibrium)

Main idea: the sum of the interests of two sides in a game is constant.

- How to understand constant?

In game theory, a Nash equilibrium is a situation in which each player's strategy is optimal given the strategies of the other players. A constant in Nash equilibrium refers to a situation in which the players' strategies do not change over time or across multiple rounds of the game. In other words, the players' strategies remain constant and do not evolve or adapt based on the outcome of previous rounds.

For example, in a game of arm wrestling between two people, assuming the total space is fixed, if you are stronger, you will get more space and I will get less space accordingly. Conversely, if I am stronger, I will get more space and you will get less.

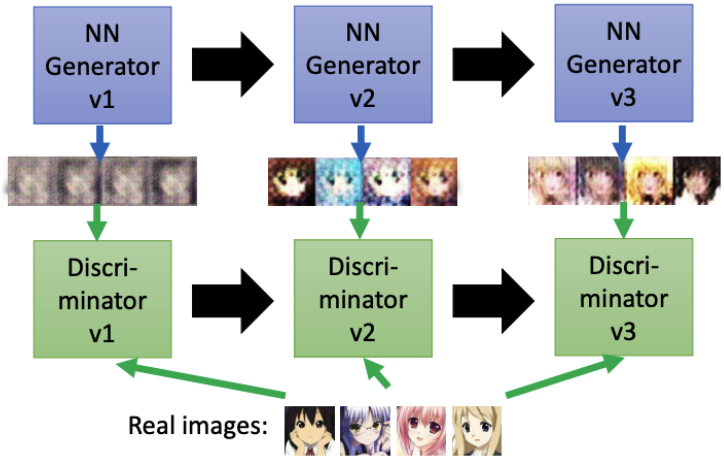
However, one thing is certain: the total space of the two of us is constant, which is the essence of a two-player game.

Figure 13: Example of Nash equilibrium: arm wrestling

4.2 CORE: How generator learns from a picture and the copy?

1. At the beginning, the parameters of the first-generation generator $G1$ are completely random, i.e. there is no way to know how to draw cartoon characters, so what $G1$ drawing is basically something inexplicable

2. At this point, the first-generation discriminator $D1$ has to do is to tell the difference between what the first-generator $D1$ drew and the real picture



3. With the information from the discriminator $D1$, the second generator$G2$ updates the parameters to adjust the target, with the aim of fooling the first-generation discriminator$D1$

> - For example, If the first-generation discriminator $D1$ determines whether it is a cartoon character by comparing whether the image has eyes, the second generator $G2$ Will generate eyes to fool the first-generation discriminator $D1$

4. However, the discriminator can also evolve, so the first-generation discriminator $D1$ will evolve into the second-generation discriminator $D2$

> - For example, the second-generation discriminator $D2$ will evaluate if a picture is a cartoon character by determining whether or not it has hair and a mouth

5. Then the third-generation generator $G3$ will try its best to update the parameters to adjust the new target, with the aim of fooling the second-generation discriminator $D2$

6. Looping through the above, the discriminator and generator will evolve together until the discriminator can't tell if the copied image is a fake cartoon character.

The purpose of the generator and discriminator is exactly opposite, one wants to discriminate well and one wants to make the other discriminate badly, so it is what we call adversarial.

4.3 Understand Nash Equilibrium in GAN

In a generative adversarial network (GAN), the generator and discriminator play a game with each other, and the goal of the game is for the generator to produce realistic samples that can fool the discriminator into thinking they are real.

In the context of a GAN, a constant Nash equilibrium would mean that the generator and discriminator have found a stable strategy that produces good results and there is no incentive for either side to change their strategy. This would result in a fixed set of generator parameters that produce the same output for a given noise input, and a fixed set of discriminator parameters that produce the same output for a given real or fake input.

While a constant Nash equilibrium may be desirable in some cases, it can also be a sign of a lack of diversity in the generated samples, as the generator is not exploring new strategies or trying to improve the quality of its output. In addition, a constant Nash equilibrium can make the GAN vulnerable to adversarial attacks, where an attacker can use small perturbations to the input to cause the GAN to produce incorrect or biased output.

Therefore, it is important to consider the possibility of evolving strategies in a GAN, where the generator and discriminator can adapt their parameters over time to produce better results and defend against attacks. This can be achieved through techniques such as training on a diverse range of input data, introducing noise into the network, or using regularization techniques to prevent overfitting. By allowing the GAN to evolve and explore new strategies, we can create more diverse and robust generative models that can better capture the complexity and diversity of real-world data.

5 Theoretical Results

5.1 Value function

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[logD(x)] + \mathbb{E}_{z\sim p_z}[log(1-D(G(z)))]$$

The value function of GAN is not like the other loss functions; it is a minmax function. We set D(x) equals 1 if x is from p_{data}, and equals 0 if x is from p_g. So if the discriminator is strong enough, both of these 2 parts will reach the maximum 0. And if it can not discriminate perfectly, this 2 part will from 0 to 1, so the value function will be negative.

We try to update D to make the function approach 0, that is the max part, while update G to make it as small as possible, which is the min part.

It is a 2-player minimax game, When the both of the 2 learners cannot make a progress, we say it reaches the equilibrium, named Nash Equilibrium.

5.2 Algorithm

The algorithm is very simple to understand. There are two for loops, an outer one that updates the generator and an inner one that updates the discriminator.

In the inner loop, we use stochastic gradient descent to update the coefficient in the value function and iterate this step for k times to learn a discriminator. In the outer loop, the function does't contain the first term in the value function because it has no relationship with θ_g.

It is worth noting that the gradient-based updates can use any standard gradient-based learning rule.

We need to consider about when to end the loop, in other word, how to judge whether the both learners have converged. In practice, this might be very difficult and become a disadvantage of GAN. During each iteration, the D cannot be trained too strong, otherwise, the D(G(z)) will be zero, and it cannot be differentiated.

Algorithm 1 GAN

for number of training iterations **do**

for k steps **do**

Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $p_g(z)$.

Sample minibatch of m examples $\{x^{(1)}, ..., x^{(m)}\}$ from data generating distribution $p_{data}(x)$.

Update the discriminator by ascending its stochastic gradient:

$$\Delta_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$$

end for

Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $p_g(z)$.

Update the generator by descending its stochastic gradient:

$$\Delta_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} log(1 - D(G(z^{(i)})))$$

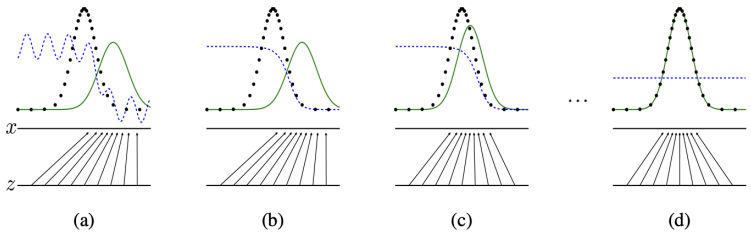
end for

5.3 Global optimum

Proposition For G fixed, the optimal discriminator D is

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

We know $p_{data} \in (0, 1)$, and so is p_g. It is like a two-sample test, which means to test weather 2 different sample are from the same distribution, we often use t-test to deal with this kind of problem. And if the G is perfect trained, p_g will equal to p_{data}, and D will be a half. Just like the graph.



The blue line is the discriminator and the green one is the generator. After some learning steps, the distribution of p_g is almost the same as p_{data} while the blue line becomes flat.

Proof. The training criterion for the discriminator D, given any generator G, is to maximize the quantity V(G, D)

we can rewrite the expectations as integrals. And then we set g(z) = x, the probability of z transform to of x, and also integral over x:

$$\begin{aligned} V(G, D) &= \int_x p_{data}(x) log(D(x)) dx + \int_z p_z(z) log(1 - D(g(z))) dz \\ &= \int_x (p_{data}(x) log(D(x)) + p_g(x) log(1 - D(x))) dx \end{aligned}$$

We simplify the integral into the following form:

$$f(y) = alog(y) + blog(1 - y)$$

which is obviously a concave function, and has a maximum when $y = a/(a+b)$. So the the optimal discriminator D is $p_{data}(x)/(p_{data}(x) + p_g(x))$.

We plug D^*_G into the value function, and note it as C(G)

$$\begin{aligned} C(G) &= max_D V(G, D) \\ &= \mathbb{E}_{x\sim p_{data}}[logD^*_G(x)] + \mathbb{E}_{z\sim p_z}[log(1 - D^*_G(G(z)))] \\ &= \mathbb{E}_{x\sim p_{data}}[logD^*_G(x)] + \mathbb{E}_{x\sim p_g}[log(1 - D^*_G(x))] \\ &= \mathbb{E}_{x\sim p_{data}}[log\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}] + \mathbb{E}_{x\sim p_g}[log\frac{p_g(x)}{p_{data}(x) + p_g(x)}] \end{aligned}$$

Theorem The global minimum of the virtual training criterion C(G) is achieved if and only if $p_g = p_{data}$. At that point, C(G) achieves the value $-log4$.

Before proving the theorem, We need to introduce the Kullback-leibler divergence.

Definition the Kullback–Leibler divergence (also called relative entropy) is a type of statistical distance: a measure of how one probability distribution P is different from a second, reference probability distribution Q.

$$KL(p \parallel q) = \mathbb{E}_{x\sim p} log\frac{p(x)}{q(x)}$$

Property KL divergence is always non-negative, $KL(p \parallel q) = 0$ if and only if $p = q$ as measures.

Proof. According to Jensen inequality: if g is a measurable real function and ϕ is convex, we have:

$$\phi(\int_{-\infty}^{+\infty} g(x)f(x)dx) \leq \int_{-\infty}^{+\infty} \phi(g(x))f(x)dx$$

We set $\phi(x) = -log(x)$ which is a strictly convex function, $g(x) = p(x)/q(x)$ and $f(x) = p(x)$. So:

$$\begin{aligned} KL(p \parallel q) &= \mathbb{E}_{x\sim p} log\frac{p(x)}{q(x)} \\ &= \int (log\frac{p(x)}{q(x)}p(x))dx = \int (-log\frac{q(x)}{p(x)}p(x))dx \\ &\geq -log(\int \frac{q(x)}{p(x)}p(x)dx) = -log(\int q(x)dx) = 0 \end{aligned}$$

Proof of theorem

$$\begin{aligned} C(G) &= \mathbb{E}_{x\sim p_{data}}[log\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}] + \mathbb{E}_{x\sim p_g}[log\frac{p_g(x)}{p_{data}(x) + p_g(x)}] \\ &= \mathbb{E}_{x\sim p_{data}}[log\frac{p_{data}(x)}{\frac{p_{data}(x)+p_g(x))}{2}}] + log\frac{1}{2} + \mathbb{E}_{x\sim p_g}[log\frac{p_g(x)}{\frac{p_{data}(x)+p_g(x)}{2}}] + log\frac{1}{2} \\ &= -log(4) + KL(p_{data} \parallel \frac{p_{data} + p_g}{2}) + KL(p_g \parallel \frac{p_{data} + p_g}{2}) \end{aligned}$$

So the minimum of C(G) is $-log4$ if and only if $p_{data} = \frac{p_{data}+p_g}{2}$ and $p_g = \frac{p_{data}+p_g}{2}$, i.e. $p_{data} = p_g$ is the only solution.

5.4 Convergence

Proposition If G and D have enough capacity, and at each step of Algorithm, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion

$$\mathbb{E}_{x\sim p_{data}}[logD^*_G(x)] + E_{x\sim p_g}[log(1 - D^*_G(x))]$$

then p_g converges to p_{data}

Proof. Consider $V(G, D) = U(p_g, D)$ as a function of p_g as done in the above criterion. Note that $U(p_g, D)$ is convex in p_g. $sup_D U(p_g, D)$ is convex in p_g with a unique global optima as proven, therefore with sufficiently small updates of p_g, p_g converges to p_x, concluding the proof.

6 Experiment

This section presents our experimental methodology, outlining data preparation, model implementation, and the training process. We deployed a Generative Adversarial Network (GAN) framework for generating handwritten digits, using the built-in MNIST dataset.

The source code for these experiments, which includes more detailed implementation and training process, is publicly available on GitHub[2].

6.1 Data Preparation

The MNIST dataset comprises images of 28x28 pixels. For GANs, data normalization to a range of (-1, 1) is more advantageous than (0, 1) due to the $Tanh$ activation function in the generator's final layer, which outputs a range of (-1, 1). We normalized the data to a mean and variance of 0.5.

We used the DataLoader class from the PyTorch library to load the dataset with a batch size of 64 and enabled shuffling to mitigate the possibility of overfitting.

6.2 Model Implementation

We implemented the Generator and Discriminator models using the PyTorch framework. The generator receives random noise of length 100 as input and outputs a generated image with the same size as the input image (1, 28, 28). The discriminator takes an image of size (1, 28, 28) as input and outputs a probability value between 0 and 1 for binary classification.

For model training, we selected the Binary Cross Entropy (BCE) loss function and the Adam optimizer, with a learning rate of 0.0002 and betas set to (0.5, 0.999) for both models.

6.3 Training Process

We trained the GAN model for 50 epochs. During each epoch, we calculated the loss for both the generator and discriminator models. The generator loss quantifies the generator's ability to deceive the discriminator, while the discriminator loss measures the performance of the discriminator in discerning between real and generated images.

6.4 Results

We saved the generated images for each epoch to visualize the improvement in image generation quality over time. After 50 epochs, the generator had improved sufficiently to produce images closely resembling handwritten digits. Concurrently, the discriminator's task of accurately classifying real and generated images became more challenging, indicating that the generator was creating increasingly realistic images.

[2] https://github.com/Sonisli/20230508_Data-Mining-Final-Project/blob/main/GAN%20Experiment.ipynb

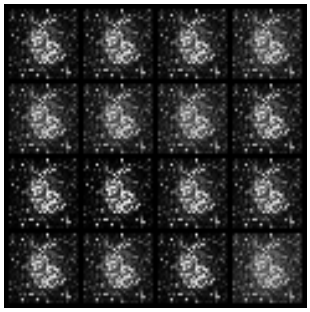


Figure 14: Images generated at epoch 1

Figure 15: Images generated at epoch 50

During the 50-epoch training period, the generator loss attained a minimum value of 1.485529899597168 at the 38th epoch, while the discriminator loss bottomed out at 0.30224746465682983 at the 3rd epoch. This data indicates a continuous improvement in the generator's performance, producing images that increasingly challenged the discriminator's ability to distinguish them from real samples.

7 Conclusion

In conclusion, Generative Adversarial Networks (GANs) have emerged as a powerful approach to generative modeling, with applications spanning across computer vision, natural language processing, and other machine learning domains. The unique adversarial training framework allows GANs to generate high-quality samples without relying on Markov chains or unrolled approximate inference networks. Additionally, GANs offer flexibility in their implementation due to the wide range of objective functions that can be employed.

Despite their groundbreaking contributions, it is important to note that GANs are no longer the state-of-the-art in some areas, as newer models like diffusion models have demonstrated improved performance in certain tasks. This highlights the rapidly evolving nature of machine learning research, where techniques and models are continuously refined and outperformed by more recent advancements. Nevertheless, GANs have laid a strong foundation for generative modeling and continue to inspire novel approaches, securing their place as a pivotal milestone in artificial intelligence research.