GAN and Game Theory

Tony Liu
College of Engineering
University of Miami
Miami, USA
yxl1837@miami.edu

***Abstract*—Generative Adversarial Network (GAN) has attracted much research interest in recent years. In this report, a simple GAN is implemented to generate handwritten digits that looks almost indistinguishable from real data. The report consists of three parts. The first part is introduction to GAN, which includes the typical architecture and the training process of GAN. The second part is the detail of the implementation. The third part is the analysis on GAN in the context of game theory.**

***Keywords*—GAN, Implementation, Game Theory**

I. INTRODUCTION TO GAN

A. GAN— A Gentle Introduction

GAN is short for generative adversarial network, which is invented in 2014. Yann LeCun, an icon of machine learning, has a comment on GAN that goes like "GANs are the most interesting idea in the last 10 years in Machine learning".

Fig 1.1 shows the progress of GAN on face generation from 2014 to 2018. The face images are all generated by GAN and it is amazing that those generated faces look like photographs of real people

Fig 1.1 Progress of GAN on face generation

B. GAN Architecture

A typical GAN has two components as in Fig 1.2: A generator and a discriminator, competing against each other, which is the reason why it is called generative adversarial network.

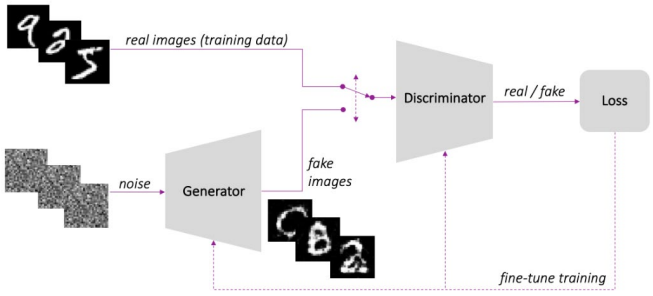

Fig 1.2 Typical GAN architecture

One the one hand, the generator is trying to generate fake data, sending it to the discriminator and wants the discriminator fail to identify it. On the other hand, the discriminator is receiving both the data from real training set and the fake data from the generator. The goal of discriminator, which is by nature a classifier is to accurately predict the data it receives is from real training set or from the generator.

In a word, the generator is trained to trick the discriminator, it wants to output data that looks as close as possible to real training data. The discriminator is trained to distinguish real data from training set and fake data from generator.

C. Training a GAN

When training a GAN, the goal is usually to obtain a generator that is eventually able to generate fake data that looks almost the same as real training data. The input of the generator is a standard Gaussian and the output of it is a sample from some distribution D on R^d, which is supposed be closer to some target distribution D_{real} during the training process. The training will use samples from D_{real} and the fake data from the generator together to train a discriminator that is trying to maximize its ability to distinguish the samples from D_{real} and D. As long as the discriminator is successful at making a distinction between these two kinds of samples, the output of it can be used as a feedback to the generator and thus improving its distribution D, making it one step closer to D_{real}.

The training is continued until the generator is able to generate perfect fake data which makes the discriminator has nothing better to do than random guessing when deciding the sample it receive comes from D or D_{real}. The discriminator can be discarded when the training is complete if generator is the only concern.

II. IMPLEMENTATION OF GAN

A. Experiment Configuration

This report is using Modified National Institute of Standards and Technology database (MNIST) of handwritten digit for training. The dataset has a training set of 60,000 examples, and a test set of 10,000 examples. The test set is not used because training a GAN does not require testing samples. The digits have been size-normalized and centered in a fixed-size image. Some samples from the dataset is shown in Fig 2.1.

Fig 2.1 Sample images from MNIST dataset

This dataset is a "hello world" dataset in machine learning field. This report is only interested in generator. Thus, the goal of this report is to train a GAN and hopefully the generator can end up with generating perfect fake data. The configuration of the generator and discriminator is shown in Table 2.1

Table 2.1 Configuration of the generator and discriminator

Item	Generator	Discriminator
Input layer size	100	784
Hidden layer size	128 (ReLu)	128 (ReLu)
Output layer size	784 (tanh)	1 (sigmoid)

The size of the output layer of the generator and the size of the input layer of discriminator are both 784, which is the "size" of the flatten input image (28×28). The size of the output layer of discriminator is 1 because the discriminator just needs to output the probability that the image it receives is real or fake.

The activation function of the output layer of generator is tanh because using tanh for output can normalize the output to be in the range of [-1, 1], which can speed up training. The activation function of the output layer of discriminator is sigmoid because it is most commonly used for outputting probability.

Some other hyper-parameters are shown in Table 2.2

Table 2.1 Other hyper-parameters for training

parameter	value
Learning rate (α)	10^{-3}
Decay rate (β)	10^{-4}
Number of epochs (e)	100
Batch size	64

The decay rate serves as decreasing the value of learning rate during the training, which is written as:

$$\alpha := \frac{\alpha}{1 + \beta \cdot e}$$

so that hopefully the model will not be trapped in local optima at the beginning of the training as learning rate is relatively big at that time.

Training of GAN involves balancing two conflicting objectives. The generator is trying to maximize:

$$J^{(G)} = \log\left[1 - D(G(z))\right]$$

which is equivalent to minimize

$$J^{(G)} = -\log\left[D(G(z))\right] \tag{1}$$

where z is the random noise from a normal distribution with zero mean and standard deviation of 1. $G(z)$ is the output of generator, which is the fake data. $D(G(z))$ is the output of discriminator given the fake data from generator.

For discriminator, standard cross-entropy cost is used so that the discriminator is trying to maximize:

$$J^{(D)} = \log\left[D(x)\right] + \log\left[1 - D(G(z))\right]$$

which is equivalent to minimize:

$$J^{(D)} = -\log[D(x)] - \log\left[1 - D(G(z))\right] \tag{2}$$

where x is a sample form real training data. The cost function of discriminator can be viewed as two parts since the discriminator is receiving real data and fake data. The term $D(x)$ is the output for real data, which the discriminator wants to maximize to 1. The term $D(G(z))$ is the output for fake data, which the discriminator wants to minimize to 0.

As for optimization strategy, this report is using gradient descent or, to be more specific, mini-batch gradient descent.

B. Training Result

The training process has been explained in detail in Part I, so this section is presenting the result without explaining the training process.

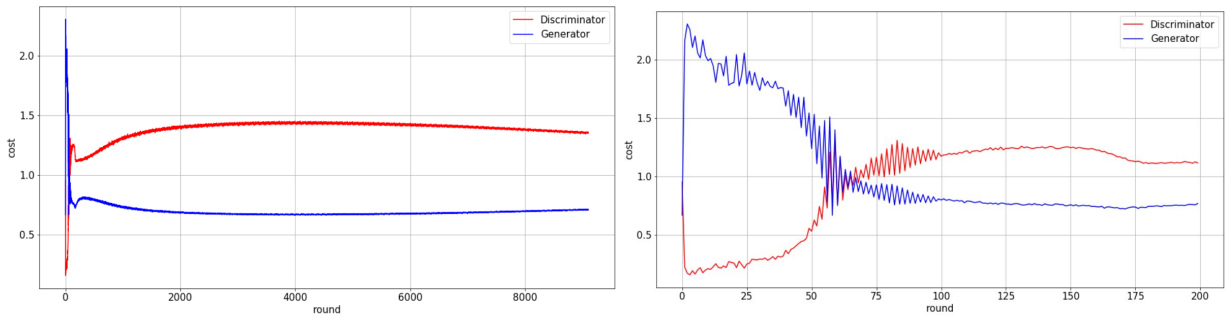

Fig 2.2 Complete learning curve (left) and the learning curve of the first 2 epochs (right)

Based on Fig 2.2, there is no big change in the complete learning curve after a few epochs. This could be the result of 1) the model has already reached the global optimal. 2) the model has yet reached the global optimal but the learning rate is too small to make a difference. Instead, the learning curve of the first 200 rounds is very interesting.

At the beginning, the fake data generated from the generator is not good enough to fool the discriminator so that the cost function of generator is very high while the cost function of discriminator is very low. However, the generator is learning to make better and better fake data, so the cost function is becoming lower and lower while the cost function of discriminator is becoming higher and higher. In the first epoch, generator and discriminator are competing against each other. If generator "wins", discriminator has to "lose" and vice versa, which is very similar with a zero-sum game. Eventually, generator is making perfect data that is indistinguishable from real data to discriminator, the discriminator has nothing better to do than random guessing, which provides useless feedback to generator. As a result, both agents cannot improve themselves through training, so the learning curve is becoming flat.

Based on the learning curve, the fake data generated from the generator is expected to be pure noise or very easy to be distinguished from real data at the beginning but before long, the fake data should be very similar with real data. The expectation matches the actual output of generator shown in Fig 2.3

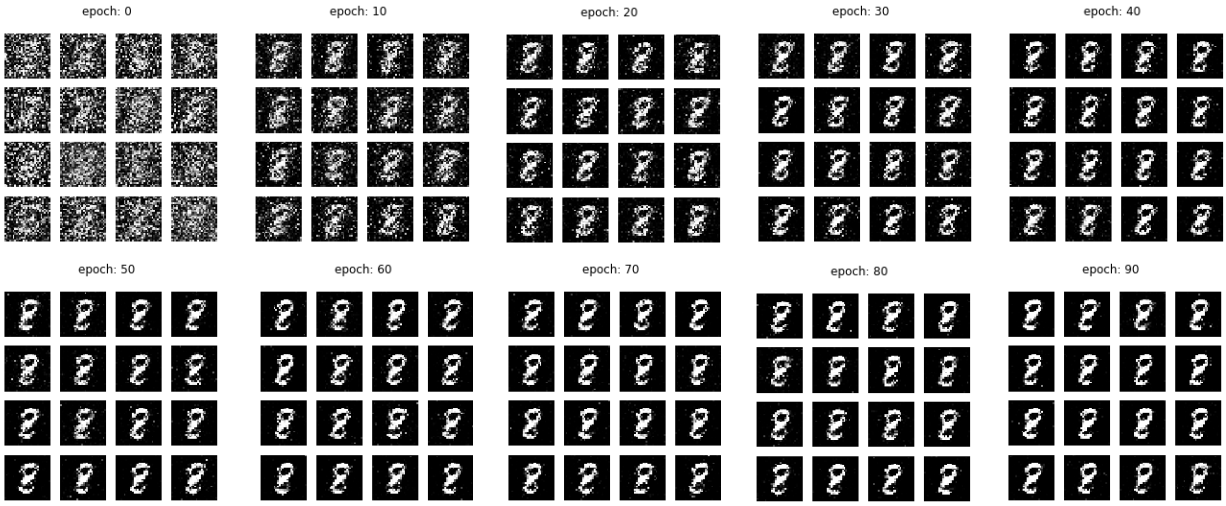

Fig 2.3 The fake data generated by generator. The output matches the expectation based on the analysis of learning curve

III. GAN IN THE CONTEXT OF GAME THEOY

A. GAN Game

One of the characteristics that makes GAN very different from traditional machine learning is that the training process is more like a game instead of an optimization problem. The cost functions of both players are defined in terms of their own parameters. The generator wants to minimize eq.1 and has to do so by controlling only the weight of each neuron in its own layers. Discriminator wants to minimize eq. 2 and has to do so by controlling only its own parameters. Because each player's cost is dependent on not only its own parameters but also the other player's parameters, and each player cannot control the other player's parameters, this scenario is thus more like a game.

If we take GAN as a kind of game, there are two players, i.e. generator and discriminator, which are represented by a multilayer perceptron in my project. These two players are playing a minimax game.

The Nash equilibrium for a GAN game corresponds to the generator recovering D_{real} exactly, that is, the generator has perfect estimation of the target distribution. In this case, the best response of discriminator is to label every data it receives as half true and half fake. Moreover, $D = D_{real}$ has been proved to be the only case where the global optimality of this game can be achieved.

B. Solving A GAN Game

However, solving the game is extremely difficult. For example, for a minimax max game as below:

$$\min_{B} \max_{A} V(D, G) = xy$$

player A is controlling x and he wants to maximize the value function $V(D,G)$. Player B is controlling y and wants to minimize the value function.

Apparently, the Nash equilibrium is the state where both x and y are equal to 0 because this is the only case where the action of the opponent does not matter, which is exactly the definition of Nash equilibrium.

However, using gradient descent will fail to reach the Nash equilibrium. In this game, the partial derivative of x and y can be easily computed as follows:

$$\Delta x = \alpha \frac{\partial(xy)}{\partial(x)}, \quad \Delta y = -\alpha \frac{\partial(xy)}{\partial(x)}$$

but if we are using gradient to update the parameters, they will never converge no matter what learning rate is used.

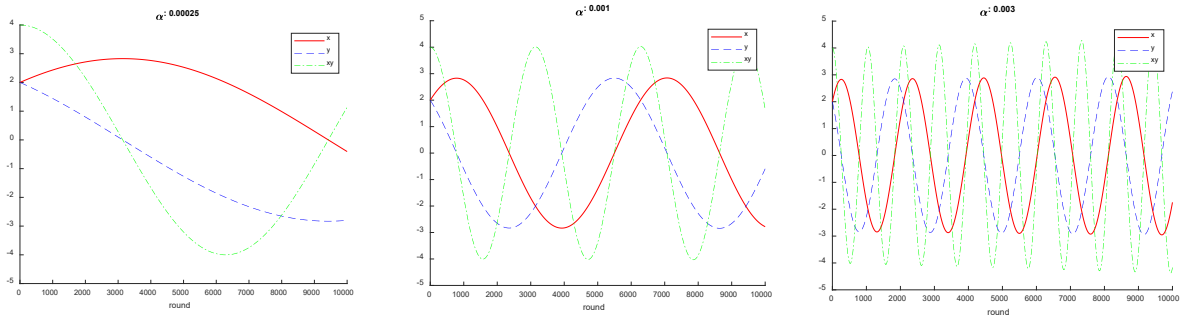

Fig 3.1 Training the example GAN game with different learning rate. x is trying to maximize value function, so it is going up at the beginning while y is trying to minimize the value function and is going down. However, when y is less than zero, x is going down. When x is less than zero too, y is going up again. x and y will repeat this process forever and never converge

The conclusion drawn from this example is that cost function may not converge using gradient descent in a minimax game, which can then be generalized to using first order optimization algorithm. In fact, training GANs requires finding Nash equilibria in high dimensional, continuous, non-convex games, which is very challenging to deal with.

C. *Existence of Equilibrium in GAN*

Another issue of GAN is the existence of equilibrium in GAN. Unfortunately, this is not yet well studied and remains an open question. To name a few researches:

In [1], the author defined a so-called resource-bounded best response. It captures not only failures of escaping local optima of gradient descent, but applies to any approximate best response computations, including methods with random restarts.

In [2], the author provides insight on how to construct a new architecture for the generator network where there exists an approximate equilibrium that is pure. (Roughly speaking, an approximate equilibrium is one in which neither of the players can gain much by deviating from their strategies.)

In [3], the author rethinks the training of GAN from the mixed Nash equilibria perspective and claim that all existing GAN objectives can be relaxed into their mixed strategy forms, whose global optima can be solved via sampling instead of optimizing.

IV. CONCLUSION

In this report, I make a brief introduction of GAN in the first part including its amazing progress in face generation, typical architecture and the training process. In the second part, a simple GAN is implemented. The actual output from generator during training matches the learning curve, which verifies the correctness of the implementation. The third part of the report focuses on the relation of GAN and game theory. GAN corresponds to a minimax game in the context of game theory, but it is difficult to find the Nash equilibrium using traditional optimization strategy such as first order optimization. In fact, the existence of equilibrium in GAN game remains an open issue. Some related work is briefly discussed in the end.

REFERENCE

[1] Oliehoek, F.A., Savani, R., Gallego-Posada, J., Van der Pol, E., De Jong, E.D. and Groß, R., 2017. GANGs: Generative adversarial network games. arXiv preprint arXiv:1712.00679.

[2] Arora, S., Ge, R., Liang, Y., Ma, T. and Zhang, Y., 2017, August. Generalization and equilibrium in generative adversarial nets (gans). In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 224-232). JMLR. org.

[3] Farnia, F. and Ozdaglar, A., 2020. GANs May Have No Nash Equilibria. arXiv preprint arXiv:2002.09124.

[4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

[5] Goodfellow, I., 2016. NIPS 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160.