# Summer research report

Name: Zhiying Tan
1st supervisor: prof. Russell Steel
2nd supervisor: prof. Yi Yang

## Introduction

This report summarizes my research topic during this summer: the generative adversarial network and its visualization. After reading the GANorignal[1] paper and some related references, I have a basic understanding of what GAN is and how it works. Using two-dimensional dataset to run the GAN process also helps me to have a better understanding of what it is going on during the training. Finally, GANlab[2] is a useful tool to visualize the training process and discover some potential problems behind the GAN structure.
This report is consisting of 5 parts:
1) basis of GAN 2) how GAN works 3) The visualization of GAN 4) research area 5) summary

## 1 . Basis of GAN

GAN (generative adversarial network) is a newly developed model which is one of the most popular research topics in recent years. Many datasets can be trained much more efficiently by using GAN model and there are a wide range of relevant applications nowadays. One example is the *Single image super-resolution*: The purpose is to transfer some old photos with low resolution to photo with much higher quality. Since the photo with low resolution have much fewer pixels than photo with high resolution, we can use GAN to create some pixels for them.
Furthermore, *TensorFlow* is a library with a large amount of machine learning model which also applied to GAN model (TensorFlow contains the 'deep convolution GAN' model which is a combination of GAN model and deep convolution neural network).However, many people find that it is hard to understand such a complicated model, so we will introduce some important features of GAN first.

### 1.1 unsupervised learning and supervised learning

GAN is under the category of unsupervised deep learning models, for the simple reason that the input data for training are unlabeled (for both the input latent data and the real sample's data). The label behind the data structure can only be discovered by training (e.g. the output result of discriminator is a number between 0 and 1 for determining whether the input is real or fake, but the 'real' and 'fake' labels are unknown before training).

### 1.2 implicit density model

For Implicit density model, although the probability density function is unknow during the whole training process, we can still generate sample form that distribution. GAN is an implicit density model: During the training process, it samples a dataset (under $P_{data}$) at first and generates another dataset (under $P_{model}$). Then the GAN model will try to minimize the difference between two datasets.

### 1.3 divergence

**KL divergence** and **JS divergence** are widely used in GAN model (e.g. GAN lab) which can measure the difference between two probability distribution ($P_{real}$ and $P_{fake}$).
The KL divergence (between $P_{data}$ and $P_{model}$) can be regarded as maximum likelihood (let x denoted as the dataset sampled from $P_{data}$ and θ denoted as the parameter).

The maximum likelihood: $$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}} \left( \boldsymbol{x}^{(i)}; \boldsymbol{\theta} \right)$$

The KL divergence:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} D_{\text{KL}} \left( p_{\text{data}}(\boldsymbol{x}) \| p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) \right)$$

As the above equation shown, maximum likelihood function is trying to maximize the total density of $P_{model}$ at each point which is equivalent to minimize the difference between $P_{data}$ and $P_{model}$.
However, the JS divergence is symmetric but KL divergence is not symmetric:

The JS divergence:

$$D_{JS}(P_{model} \| P_{data}) = \frac{1}{2} D_{KL}(P_{data} \| \frac{P_{data}+P_{model}}{2}) + \frac{1}{2} D_{KL}(P_{model} \| \frac{P_{data}+P_{model}}{2})$$

The KL divergence:

$$D_{KL}(P_{data} \| P_{model}) = -\sum_i P_{data}(i) \log \frac{P_{model}(i)}{P_{data}(i)}$$
$$D_{KL}(P_{model} \| P_{data}) = -\sum_i P_{model}(i) \log \frac{P_{data}(i)}{P_{model}(i)}$$

The above two direction of KL divergence are not the same: For example, let the $P_{data}$ (data distribution) be a mixture of more than one Gaussian then its probability density diagram will have more than one peak while the $P_{model}$ is defined as a single Gaussian distribution with only one peak. By choosing $D_{KL}(P_{data} || P_{model})$ for optimization will cause $P_{model}$ to put its probability to everywhere that the data occurs, but $D_{KL}(P_{data} || P_{model})$ prefers to place no probability to the area with no data point.

## 2. how GAN works

Basically, GAN is consisting of 2 models: generator and discriminator. Generator aims to generate some fake data (generated data) to be as real as possible (real means the real dataset) so that the discriminator is unable to tell the difference between them , while the discriminator is like a detector who aims to figure out the difference between generated data and real data .
Then the training process begins, the generator and discriminator compete with each other where the generator strives to make the value of **P**(generated data is real ) to approach 1 and the discriminator wants this value to approach 0. Finally, There may exist an equilibrium for this battle which is P (sampling real data is real) = 0.5 that the discriminator is unable to tell whether the real data is real or not ( also means that the $P_{model}$ is the same as $P_{data}$ )

### 2.1 generator and discriminator

Generator and discriminator are two leading roles in GAN.

### 2.1.1 The generator

**The generator** can be denoted as G, the input is latent variable set z. G is a differentiable function. First, by sampling we can have a latent dataset z which is the input to the generator and the output is fake dataset (denoted by G(z)). Normally, 'Neural network' or 'convolution neural network' (e.g. DCGAN architecture) are used to build the inside layers for generator. G(z) will be used when calculating the loss and the parameters in each layer will be updated according to the calculated result.

### 2.1.2 The discriminator

**The discriminator** is denoted as D and the real dataset is denoted as x. For the discriminator, it is also built by neural network or convolution network. The output of discriminator function is D(x) and D(G(z)) which is a real number between 0 and 1 (probability that the input data belongs to real dataset). The discriminator is trying to get the output: D(x) = 1 and D(G(z)) =0 while the generator is trying to optimize the G function to reach D(G(z)) =1. Since these two goals are contradict with each other, there exists a competition between generator and discriminator.

## 2.2 cost function

The output of cost function is the loss value of input dataset. By computing the loss value, people can learn about the behaviour of that sub-model (sub-model means generator and discriminator).

### 2.2.1 The cost function of discriminator

The cost function of discriminator can be denoted as $J^{(D)}$ which is the standard cross-entropy function in the following form ($\theta^D$ represents the neurons of each layer in the discriminator and $\theta^G$ represents the neurons of each layer in the generator):

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(z)\right)\right).$$

The discriminator always desires to minimize its cost function with respect to $\theta^{(D)}$ and the value can be minimized when log D(x) and log (1- D(G(z) ) are close to zero. Under the minimizing process, we will do:

$$\frac{\delta}{\delta D(\boldsymbol{x})} J^{(D)} = 0.$$

And then we can get:

$$D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}.$$

Which will be large whenever the model density is too low and small whenever the model density is too large. Then the generator can optimize its model by following the direction of enlarging D*(x) value.

### 2.2.2 The cost function of generator

The cost function of generator can be denoted as $J^{(G)}$ which has 3 forms and the Heuristic, non-saturating is the most common one with a relative smaller sample variance than the other two forms.

**Minmax**

The first one is Minimax game where $J^{(G)} = -J^{(D)}$ and $V(\theta^{(D)}, \theta^{(G)}) = -J(\theta^{(D)}, \theta^{(G)})$, then we can formalize the minmax cost function as:

$$\boldsymbol{\theta}^{(G)*} = \arg\min_{\boldsymbol{\theta}^{(G)}} \max_{\boldsymbol{\theta}^{(D)}} V\left(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}\right).$$

Where we maximize V among the space of $\theta^{(D)}$ at first, and then we do the minimization among the space of $\theta^{(G)}$. However, the maximization may have rejected the generator's best option and the gradient may have probably been vanished.

**Heuristic, non-saturating**

The second cost function is Heuristic, non-saturating game where we have the form of:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D(G(\boldsymbol{z}))$$

Which is more directly than the minmax game. According to this function, the generator's cost function is only related to the behaviour of fake samples.

**maximum likelihood**

The last one is maximum likelihood game:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{z} \exp\left(\sigma^{-1}\left(D(G(\boldsymbol{z}))\right)\right)$$

Where σ is the sigmoid function and the equation is related to the KL divergence when the discriminator is optimal:

$$\boldsymbol{\theta}^{*} = \arg\min_{\boldsymbol{\theta}} D_{\mathrm{KL}}\left(p_{\mathrm{data}}(\boldsymbol{x}) \| p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta})\right).$$

## 2.3 training process

In this section, we will first talk about gradient descent and optimizer that are used in GAN, then the detailed processing of GAN.

### 2.3.1 minibatch gradient descent

Gradient descent is updating each parameter according to its partial derivative value and the learning rate accordingly at each step until reaching its local maximum or the local minimum.
Normally we will use stochastic gradient descent for the training process, but it takes a really time long (O(m), where m is the sample size) to complete. Since for each epoch, it takes only one data into consideration.
The minibatch gradient descent belongs to stochastic gradient descent, but minibatch gradient descent will proceed much faster and more accurately. It is when a minibatch size selected (usually more than 1 but less than m), then each epoch of training will be finished under that minibatch. The next epoch will continue on updating the variable according to the next minibatch set's data and there are totally (m/minibatch size) epoch.

In the GAN process, each epoch will sample a minibatch called $X^{[t]}$ ($X^{[t]}$ means real dataset at $t^{th}$ epoch) and a minibatch called $z^{[t]}$ (the latent variable set at $t^{th}$ epoch) will be imported into the generator to give $G(z^{[t]})$. Then the loss value of generator and discriminator can be calculated for gradient descent. As a result, $\theta^D$ and $\theta^G$ will be updated by gradient descent at each epoch.

### 2.3.2 Optimizer

There are a great many choices of optimizer (e.g. Adam optimizer, RMSprop) for updating $\theta^D$ and $\theta^G$. In particular, RMSprop, a new developed optimizer which is implemented by exponential weighted method. At each backpropagation step, it will take a few more previous steps into calculation (dividing the gradient by an accumulated average value). $\theta^D$ and $\theta^G$ will then have a much direct path (less oscillation) to the local minimum.
Adam optimizer is a combination of gradient descent with momentum and RMSprop optimizer where the gradient is a weighted sum of few previous steps rather than just the current step.
Adam optimizer and RMSprop are much more efficient than gradient descent which are widely used for GAN's training. (see section 3.2.4 for the comparison of ADAM and SGD)

### 2.3.3 detailed processing

At first, the total number of iteration (epoch) will be set. The GAN algorithm can include both Minibatch gradient descent and optimizer to make it more efficient, so a minibatch size n can be selected for each epoch of training and we can use the optimizer at each epoch also.
First, we need a function for generating a sample dataset $z^{[t]}$ (for example, a sample can be drawn under the uniform distribution or Gaussian distribution) of size n. Also, a dataset $X^{[t]}$ (MNIST dataset is commonly used as GAN's real dataset X) of size n is needed to be imported in every epoch.
There are totally two parameter sets ($\theta^D$ and $\theta^G$) for discriminator and generator which will be restored at every epoch. The generator and discriminators' activation function (like sigmoid function, leaky Relu function and so on) may have defined for each layer. However, the parameter sets are undefined at the beginning of training (at the $1^{st}$ epoch), so the parameter sets can be started by some random number under an appropriate range.
At each epoch, $z^{[t]}$ will be the input of generator to give $G(z^{[t]})$. Then $D(X^{[t]})$ and $D(G(z^{[t]}))$ will be used to calculate the generator loss and discriminator loss. The optimizer will update the parameter sets simultaneously according to their loss function's trend. After that, the training process will enter the next epoch.

## 3. Visualization of GAN

**Visualization** of GAN means that we can use some tools like image, iterative graph to visualize the training process and their result. Visualization can help the non-expert in machine learning area to have a basic understanding of GAN. A simple two-dimensional example of visualization can help us have a better understanding of GAN's behaviour and the GAN lab can help us to find out the current problem of GAN.

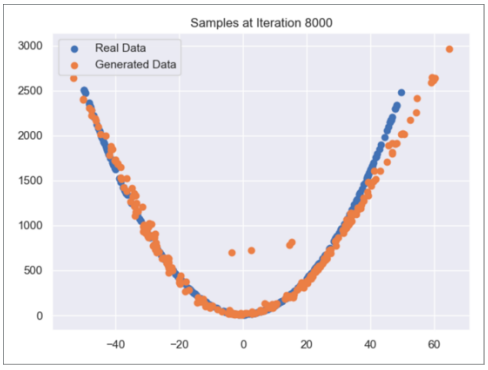
### 3.1 a simple example

This simple example is raised by a two-dimensional real dataset $(x,y)^{[t]}$. The two-dimensional real dataset is formed by sampling $X^{[t]}$ from uniform distribution and import the dataset into

function $y = 10 + x^2$. The $z^{[t]}$ dataset is also a two-dimensional array raised by uniform distribution. Then the generator wishes the $G(z^{[t]})$ to be as similar as possible to $(X, Y)^{[t]}$.

### 3.1.1 visualization of real sample and generated sample

We can have a plot statement within each epoch then the output figure shows the current minibatch's data (real sample $(x,y)^{[t]}$ and generated sample $G(z^{[t]})$). By comparing the real data set and fake data set at each epoch (or few epoch), we can know whether the generator can gradually make the fake data set resemble the real data set (which can be seen by the following image):




### 3.1.2 visualization of generator

To visualize the generator's development, one way is to make a comparison of before updating generator's weight (the neurons in each layer) and after updating generator's weight. The presented dataset is a two-dimensional output before the last hidden layer of discriminator.


We can then plot the centroid of each colour segment of the upper figure to make it easy (see the following figure). By comparing the generated data before G and generated data after G, we can see how many change does the generator's weight have? As shown below, the blue point coincide the orange point as no change happens to real data by applying the weight change. When the number of iterations increase the changing distance between real data's centroid ( the orange point) and generated data's centroid (the red point) also shows how similar that the real data set and generated data set is.


## 3.2 GAN lab

GAN lab designed for visualization where the details of training can be shown more directly to user. In GAN lab, the users can draw their own dataset which. During the whole training process, the true dataset will remain unchanged and the generator will use the random noise as latent variable. However, the generator, discriminator, G(z), gradient's direction is varying at each epoch.

There are 3 ways of visualization in GAN lab: visualization of model structure, visualization through the layered distribution and visualization in metrices space. After that, GAN lab also emphasizes the selection of appropriate hyperparameters.

### 3.2.1 basis of GAN lab

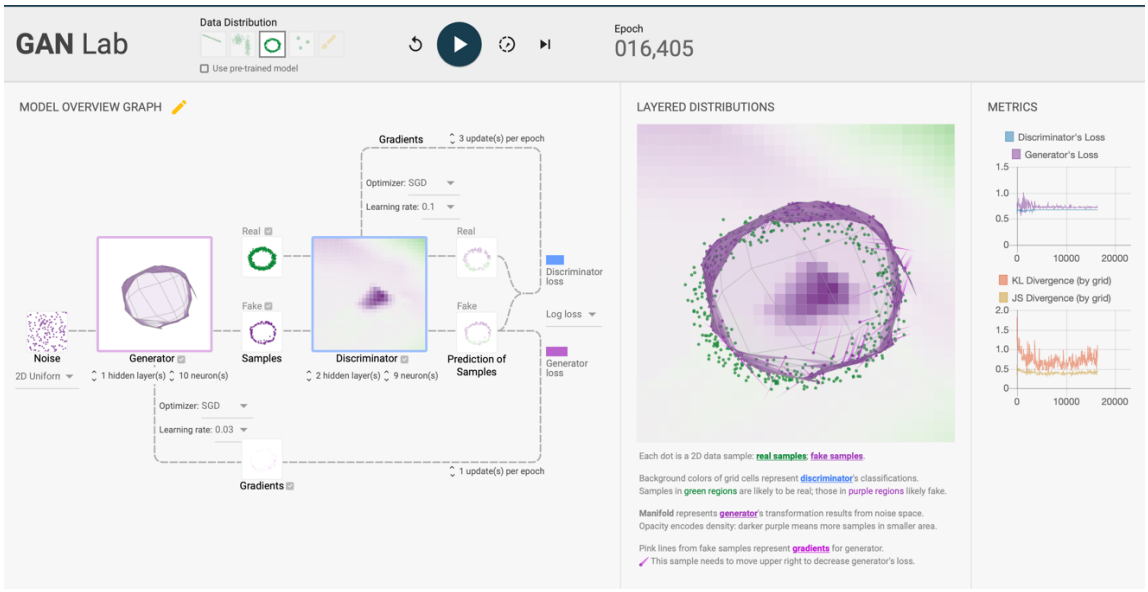



Figure.4. Here is the interface of GAN lab

According to Figure.4, we can see that there are many icons and graphs in GAN lab which have different meanings and predictions. The two main colours in this graph are purple and green, representing the fake data and the true data respectively.

**Data distribution**, is shown at the very top of the figure. Users can choose the existing data distribution or draw a new data distribution (see figure.5). As each point in the drawing space has its own position data and these position data are transformed into real data set.


Figure.5 the left picture shows a distribution drawn by user
and then a real sample set can be generated according to the drawing picture.

**Manual step execution**

This is a sub-model level training. Beside the 'start' icon, there is a step icon under which there are 3 options (generator, discriminator and both) which decide the epoch-level of training (see figure.6).

By clicking generator's button, the background colour will remain unchanged while the generated data and the gradient's direction are varying, since only the generator's parameter has updated as the discriminator's variable remains unchanged. By training the discriminator only, the background colour and the gradient's direction will change but the generated data will not move.

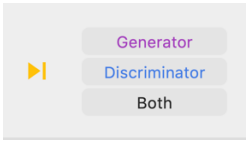



Figure.6 icon of manual step execution

**Slow motion icon**

The slow motion is under a component-level training where it will have 5 steps for each sub-model training. The five steps include: 1) evaluating G(z), 2) evaluating D(x) and D(G(z)),
3) compute loss, 4) compute gradients and 5) update the sub-model based on gradients.

**Model overview graph**

There are also many connected nodes and users can select one of the many options in the dropdown menu. On the most left-hand side, users can choose the type of input noise: 1D uniform, 1D Gaussian, 2D uniform or 2D Gaussian. For both the generator and the discriminator, the number of hidden layers and neurons can also be customized. Setting the learning rate for the Adam optimizer or the SGD is needed before starting. By backpropagating, the log loss or the least square loss will return back to the optimizers to calculate the gradient's value. Furthermore, users can also choose the number of updates per epoch in the dropdown menu for both the generator and the discriminator.

**Layered distribution**

Layered distribution view is a dynamic picture which is varying as the number of epoch is increasing. The green dots mean real data and the purple dots with purple line is the generated data point with its own gradient's direction.

The purple line always points to the direction of decreasing loss value with the length of line indicating the strength. The background's colour can be used to visualize the discriminator and the generator can transform the square grid into a manifold (see section 3.2.2).

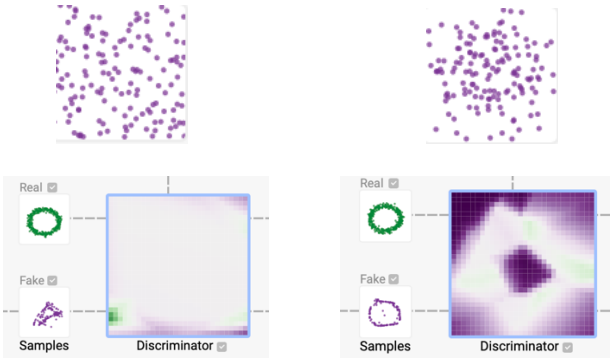
**Metrics**

The 2 diagrams in this part record values at each epoch. One of them exhibiting the generator's loss and discriminator's loss while another diagram showing KL divergence and JS divergence.

### 3.2.2 visualization of model structure

The model structures include real sample, random noise, fake samples, generator, discriminator, predictions and gradients. In this part, all of the figures will only contain one of the model structures.

**Visualization of data** is the visualization of real sample, input noise and generated data. Real sample points are coloured with green and generated sample points are coloured with purple. The real sample will remain unchanged during the whole training process while the generated data may be varied at each epoch. Another thing is the type of input noise which can also influence the training process: The gaussian distribution concentrates most of its data in the middle while uniform distribution's noise is equally distributed (see the following figure). Furthermore, A 1D input noise is only able to form a line shape sample, so it can only be adopted to one dimensional real sample. However, the 2D input noise can be transformed into two-dimensional real sample.




Normal random noise Gaussian random noise

Figure. It is obvious that the above 2 types of input noise can have different generated samples and background 2D heatmap. We can conclude that the choice of input noise can also affect the output.

**Visualization of generator**

The generator can be visualized by their output: the output generated sample or the manifold. There's a 2D manifold in the generator's box which have different shape in each epoch (as shown in figure.6). To form the manifold, we first have a 20 by 20 square grid for the random noise where each cell can contain some or none of the random noise data (as the first graph of figure.6). Since each random noise data point has its own position (denoted by (x[i],y[i]) ) and each cell can represent a range in both x and y direction,(e.g.0<x<0.05 and 0<y<0.05). Before the training start, every cell is coloured by purple and the opacity of each cell means the density of random noise (higher opacity means higher density). As the training starts, each cell will gain their new position by putting its four corner's position data into the generator (e.g. G(0,0) = (0.35,0.2); G( 0.05,0.05) =(0.12,0.45) ). After the transformation occurs, the square cell becomes a quadrangle and the opacity will also be changed. The new opacity value can be calculated by (original density value/area of the new quadrangle). The manifold will always equivalent to the generated data, so by looking at the manifold reshape procedure, people can learn how the fake sample are formed (as shown in figure.7). Furthermore, people can compare the manifold with the real sample to see whether the generator performs well during the process.

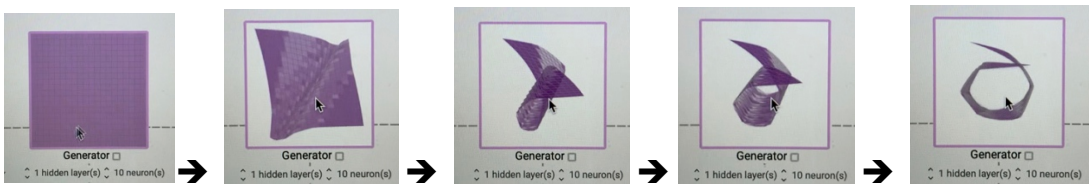



Figure.6 This series of graph shows how a square grid gradually becomes a manifold

Figure.7 The first figure (counting from left to right) shows the generated data and the third graph shows the manifold at that point. It is obvious that the manifold is equivalent to the fake sample. By comparing the manifold and the real sample is also a good way to judge whether the generator performs well.

**Visualization of discriminator**

The discriminator can be visualized by the background colour (the 2D heatmap) or the predictions of samples. The background (see figure.8) is a square grid with many small square cells and the colour of each grid cell is defined by the output of discriminator (the input is the position of four corner into the discriminator). Purple means that grid cell is judged as fake region and green means true region while grey means uncertain. Since darker colour means stronger believe, so darker green means that grid cell is more likely to be true and darker purple means more likely to be fake.

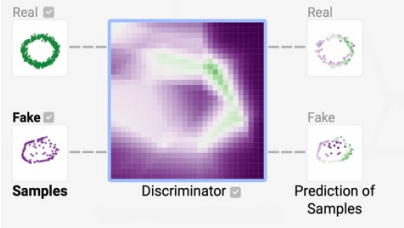



Figure.8 The background (2D heatmap) is presented in the discriminator's node. According to the figure, real samples and fake samples are two input sources of discriminator, the output are the prediction for both samples.

Predictions of sample (see figure.8) is made by the discriminator for both real sample and fake sample. By putting all of the true data points into discriminator, the colour of some data points become purple while other points are still coloured by green (green means that point is considered to be the data under $P_{data}$, purple means that point is likely to be a fake data point). Some of the fake data points are predicted as true data point by the discriminator.
A well performing discriminator should predict real data set as mostly green and the fake sample to be mostly purple, so the user can judge its performance by viewing the predictions of samples.

### 3.2.2 combinational visualization

The combinational visualization is presented in the form of layered distribution. In this part, the visualizing graph can contain more than one model structure. For the simple reason that the comparison between data structures can give a better insight of GAN.

**Real samples and fake samples**

By putting the real samples and fake samples (see figure.9) together can Inform us indirectly of whether the generator works well (whether the fake sample can coincide with the real sample).

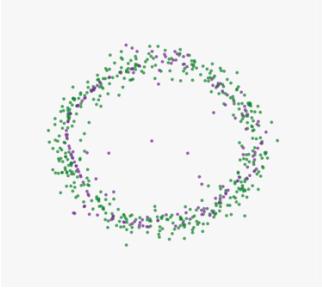


Figure.9 real samples and fake samples. In this graph, most of the fake data points match the true sample's 'ring'.

**Data samples and 2D heatmap**

In this part, the user can distinguish the heatmap's colour around the real and fake samples and the colour in other parts to see how the discriminator performs.
A perfectly preforming generator (keep clicking the generator's slow-motion button only) can transfer most of the fake samples into the green region rather than the purple region (see figure 10.1) while a perfectly preforming discriminator (keep clicking the discriminator's slow motion button only) can put the purple region in any other area except for real sample's region (see

figure 10.2). However, when the generator and discriminator compete with each other, it is almost impossible to reach perfect state of any two side (see figure 11.1).

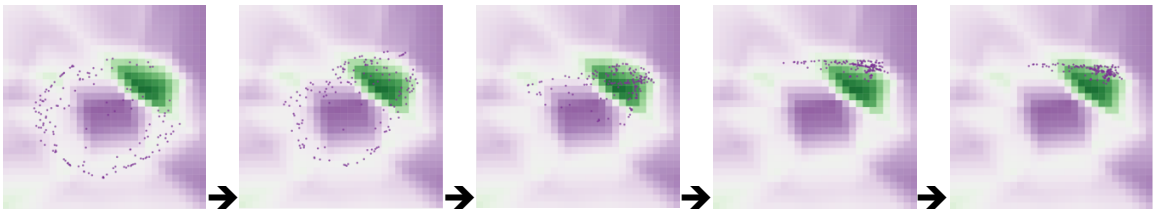


Figure 10.1 This series of figures are the combination of 2D heatmap and the fake sample. By training generator only, the 2D heatmap will remain unchanged but the fake sample is moving towards the green region even though it has been far from the real sample's region.

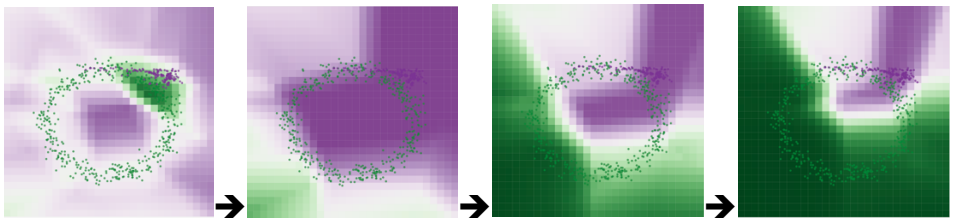


Figure 10.2 This series of figures are the combination of 2D heatmap and the data sample. The first diagram in this series is exactly the last diagram in Figure 10.1. By training the discriminator only, the fake sample and real sample remain unchanged while the green region moves towards the real sample and purple region moves towards the fake sample.

**Gradients and 2D heatmap**

The gradient does not always have to point the green region of heatmap (tend to be considered as real data by the discriminator), but the direction of decreasing the loss value (see figure 11.2)

Figure.11.1

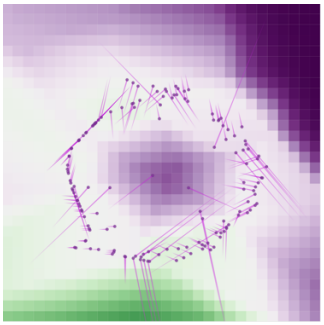


figure.11.2

Figure.11.1 show us the data samples and the 2D heatmap. In this diagram, most of the fake sample points coincide with the 'ring' (the real sample), and the colour in this area is much lighter than the other region. However, the region inside and outside the real data 'ring' are darker colour. By clicking the generator's slow-motion button will cause the purple points move towards the green region at the right bottom while training discriminator only will cause the bottom green region to become one of the purple regions.

Figure.11.2 display the 2D heatmap and gradient. In this diagram, the samples do not have consistent pointing direction with some of them pointing to the green region.

### 3.2.3 metrices spaces visualization

The charts of metric space include the loss chart and the divergence chart.

The loss value chart can show the reader whether the sub-model perform better as the number of epoch increase (see figure 12.).

The divergence (see figure 13.) is measuring how similar $P_{data}$ and $P_{model}$ is. The KL divergence and JS divergence are:

$$KL(P_{\text{real}}||P_{\text{fake}}) = -\sum_i P_{\text{real}}(i) \log \frac{P_{\text{fake}}(i)}{P_{\text{real}}(i)}$$

$$D_{JS}(P_{real}||P_{fake}) = \frac{1}{2} D_{KL}(P_{real}||\frac{P_{real}+P_{fake}}{2}) + \frac{1}{2} D_{KL}(P_{fake}||\frac{P_{real}+P_{fake}}{2})$$

The $P_{real}(i)$ is the probability density in the $i^{th}$ cell which is calculated by (number of real data points in $i^{th}$ cell/ total number of real data points). Then $P_{fake}(i)$ is defined by (number of fake data points in the $i^{th}$ cell/ total number of fake data points). When $P_{real}(i)$ and $P_{fake}(i)$ getting closer and closer to each other, the $KL(P_{real}||P_{fake})$ value will getting smaller and smaller, so the visualization of KL divergence can show us whether the fake sample are generated in a good shape.

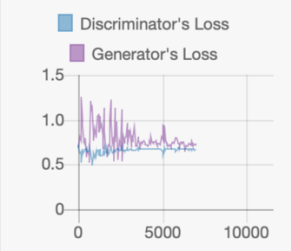



Figure 12

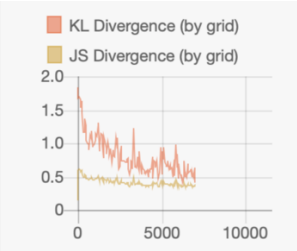



figure 13.

Figure 12. The chart shows discriminator's loss and generator's loss. In this example, both of the two lines become more and more stable.

Figure 13. This chart illustrating the KL divergence value and JS divergence value as the epoch number increase.

### 3.2.4 hyperparameter

Balancing the hyperparameter between models and choosing an appropriate hyperparameter (e.g. number of layers and neurons for each sub-model, type of input noise) can have a more efficient training process.

**Balance in sub-model level**

The following figure 14. give a more direct idea of the meaning of updates per epoch (number of training loop in each epoch):

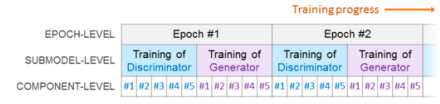


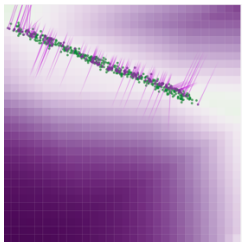

Figure 14. there are 5 updates (training) per epoch for both generator and discriminator

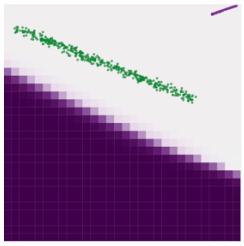
By setting a higher number of training loops for discriminator than generator (e.g. three discriminator's training and one generator's training in each epoch) can give the discriminator more chance to optimize its parameter and a larger gradient's value for generator, then the fake

sample will move faster in each step. By these kinds of setting, GAN's training can have a faster convergence rate than before.

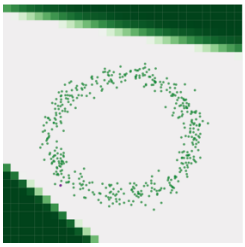
**Selection of optimizer**

Choosing SGD or ADAM will cause different training result. The following figures show the different training result by using the SGD and ADAM optimizer respectively, while the other hyperparameter are set to be the same (e.g. the input noise, number of neurons and layers):

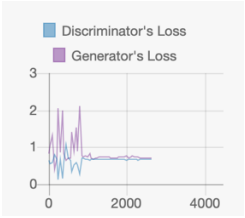

SGD


ADAM

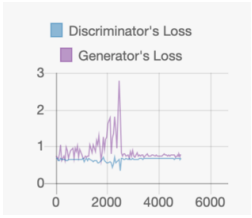

SGD


ADAM

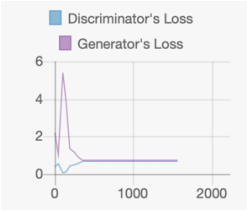
SGD



ADAM



SGD



ADAM

The first line of figures is presented by layered distribution and the second line of figures show the corresponding loss value. It shows that SGD can give better generated sample than ADAM, but the data points and 2D heatmap are moving in a larger step in ADAM. Considering the loss function, ADAM can converge faster than SGD. According to this series of figures, SGD is a better option for GAN's training but a relative slower speed of convergence. However, some people think that by choosing ADAM for discriminator and SGD for generator can take the advantage of both training faster and generating better samples, but it is not working in all cases.

# 4. Research area

There is a large amount of research area in GAN. By doing the research in GAN area can help us optimize the GAN in an efficient way.

## 4.1 Non-convergence

Briefly speaking, the convergence is a problem about: when should we stop the training? In the training process, both generator and discriminator keep updating until reaching the equilibrium or it may be unable to reach an equilibrium but keep competing. This is one of the biggest facing problems nowadays and mode collapse is under the category of non-convergence.

## 4.2 mode collapse

Mode collapse is also called Helvetica scenario which is one of the most popular research topics in recent years. It is illustrating a problem when a large number of latent variables are transferred into to the same output point by the generator. There are many examples refer to mode collapse (see figure 15. and figure 16.) especially when generating some picture, the output pictures may be too similar with each other to find a difference.

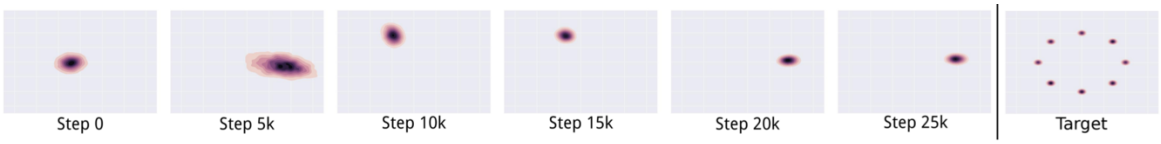



Figure 15. The last graph of this series shows that the real sample is consisting of 5 dense regions. However, as the number of epoch increase, all the latent variables z are transferred into only one region by the generator.

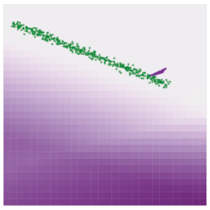


Figure 16. here is an example of partially mode collapse when training in GAN lab
Where the green points are real samples and purple points are fake samples

### 4.2.1 cause of mode collapse

Mode collapse is a common problem in the application of GAN, for it will cause the generating sample to be lack of diversity. However, the GAN model does not force the output data to be diverse, but as close to real sample as possible. When the generator finds an optimized output region where the discriminator regards as true, it will probably transfer most of the latent variable z to that region.

### 4.2.2 handling mode collapse

**Minibatch discrimination and feature mapping** are one of the most combination to deal with mode collapse problem directly. Minibatch discrimination means that we can have a function o(x) to measure the similarity among real sample or generated sample. If the generated minibatch has a much higher similarity than real minibatch, then the discriminator should be able to detect that and judge this minibatch as fake dataset. For the implementation of the detection, we can add some layers to the discriminator's network (since GAN is under the category of unsupervised learning). However, the feature matching is implemented by considering the output statistics of both real samples and generated samples into the generator's network.

**Unrolled GAN** is another method to deal with mode collapse indirectly. When using the GAN lab, it is allowed to train the discriminator only where the generator remains unchanged (see figure 10.2). The unrolled GAN is using the similar idea: the generator considers few more steps of 'training discriminator only' when updating its parameter.

Although the discriminator will be considered for few more steps, the discriminator does not actually update for few more steps. Backpropagation is the key in unrolled GAN, the generator need to backpropagate few more discriminator's steps now (calculating the gradients for all these steps) for optimization (see figure 17.)

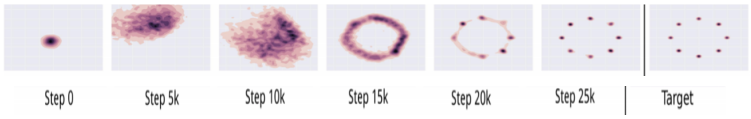



Figure 17. Here is an example of applying unrolled GAN, the generated sample at step 25k is almost the same as the real sample (target)

## 5. summary

At the beginning of this summer, I started from a blank sheet of paper in the field of deep learning. Then I began to read the GAN's original paper and its tutorial to have a basic idea of GAN model (as illustrated in the first and second part of this report). Meanwhile, I also spent time on the study of neural network, gradient descent, optimizer and some other topics related to GAN. After that, I learnt the step by step GAN's training process in python by applying some of the TensorFlow's library. The visualization can give me the insight of GAN model (as illustrated in the 3rd part), so I used python's plot to visualize the training of a very simple dataset. To learn GAN in a much more detailed way, I used GAN lab as another tool for visualization and I have read the related papers also. However, GAN lab has raised my interest by the mode collapse problem which is one of the most popular research areas in GAN (as illustrated in the 4th part).

## Reference

[1] Ian goodfellow. NIPS 2016 Tutorial: generative adversarial networks. arXiv:1701.00160v4
[2] Ian J.Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative adversarial Nets. arXiv: 1406.2661v1
[3] Minsuk Kahng, Nikhil Thorat, Duen Horng (polo) Chau, Fernanda B.Vigégas,and Martin Wattenberg. GAN lab: understanding complex deep generative models using interactive visual experimentation. arXiv: 1809.01587v1
[4] Luke Metz, Ben Poole, David Pfau and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. arXiv:1611.02163
[5] Aiden Nibali. Mode collapse in GANS
[6] Tim salimans, Ian Goodfellow, Wojeiech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. Improved techniques for Training GANs. arXiv:1606.03948
[7] Jonathan Hui. GAN-Ways to improve GAN performance.
[8] Jonathan Hui. GAN-unrolled GAN (how to reduce mode collapse).
[9] Aadil Hayat. Building a simple generative adversarial network (GAN) using TensorFlow.
[10] Geoff Hinton. Neural Networks for machine learning.
[11] Diedrik P. Kingma, Jimmy Lei Ba. Adam: a method for stochastic optimization. arXiv: 412.6980
[12] Mirantha Jayathilaka. Understanding and optimizing GANs (going back to first principles)

[13] Liangchen Luo, Yuanhao Xiong. ICLR 2019: ‘fast as Adam & good as SGD’-new optimizer has both
[14] Andrew Yan-Tak Ng. RMSProp (C2W2L07)
[15] Andrew Yan-Tak Ng. Adam Optimization Algorithm(C2W2L08)