Report for Individual Project

Yuhan Liu

1. Experiments

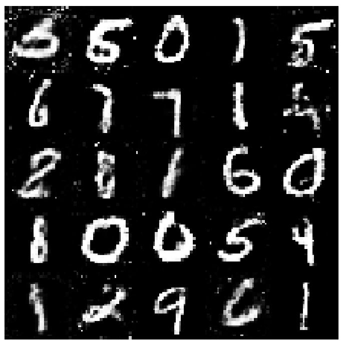
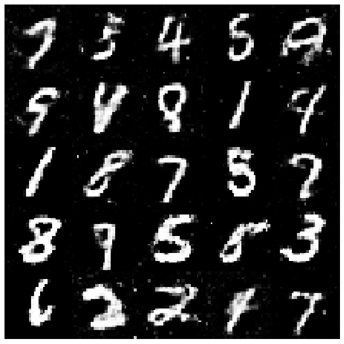
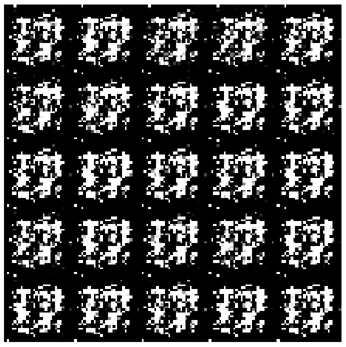
1) Fully Connected Dense Neural Network

Structure: (1) 5 dense layers for both generator and discriminator, for the generator, construct it with 128 units, 256 units,512 units and 1024 units, the output layer has 784 units, which is exactly the number of the pixels. And for the discriminator, just reverse the number of units for these 5 layers.

(2) Activation function is leakyReLU, and for generator, the activation function of the output layer is tanh, and sigmoid for the discriminator. Loss function is binary cross entropy, and the optimizer is adam.

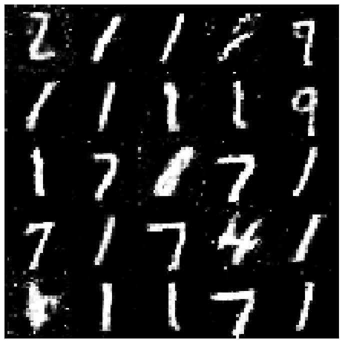
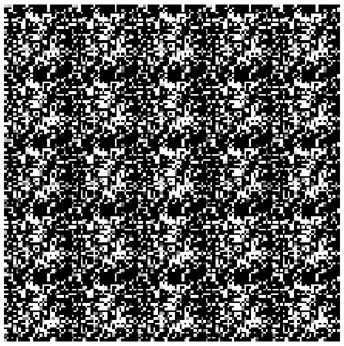
(3) Batch size is 128, number of epochs of training is 30 most of the time.

First, I tried to train it for different epochs, 50 ,100 and 300, The results showed as below:

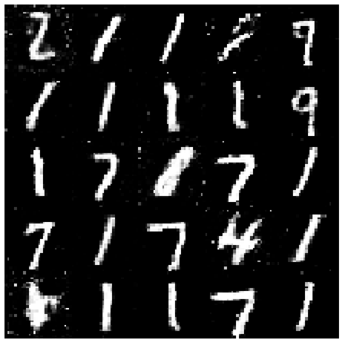
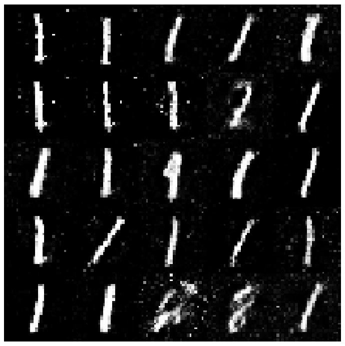
There is a really interesting phenomenon, at the beginning, the results seem to improve as the training epoch increasing, but at a certain point, the loss of both generator and discriminator decrease suddenly, after that, the value of loss begins to fluctuate over a wide range. It seems that, the generator finds a way to create a bunch of garbage to fool the discriminator. From the result, we can see that for a GAN model, sometimes, training for too many epochs may cause some unexpected result.

Second, I tried to decrease the number of the neuron for each layer, replace the multiply increased neuron number with a fixed number 128 for both generator and discriminator, and train both the decreased model and original model for 30 epochs to see the effect of the neuron number:

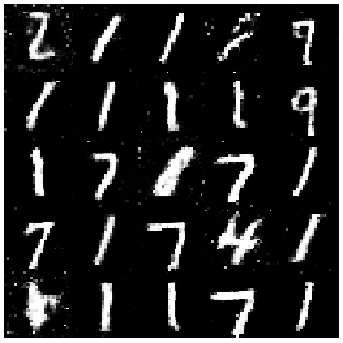
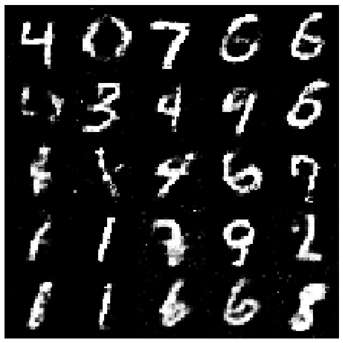
First picture is the original model, and the second one is constructed with 128 units of each layer, we can see that, with the same training epoch, increase of the neuron for each layer help to make much better result.

Third, I tried to add additional drop out layer after each leakyReLU layer in the generator with 40% of drop out probability, also train them for 30 epochs. Actually, more training epochs inside a range will bring better result, but the computation resource is limited, so I only trained all of them for 30 epochs. It is enough to make the comparison.

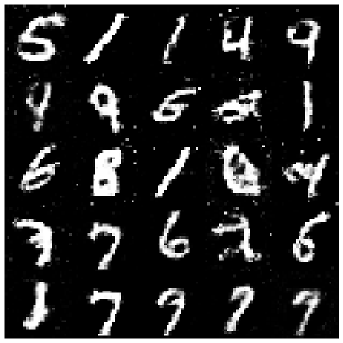
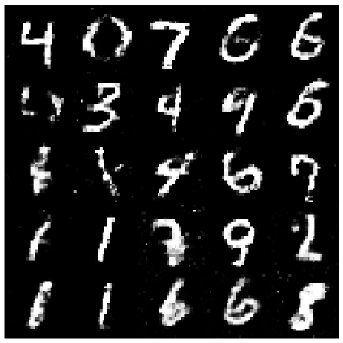
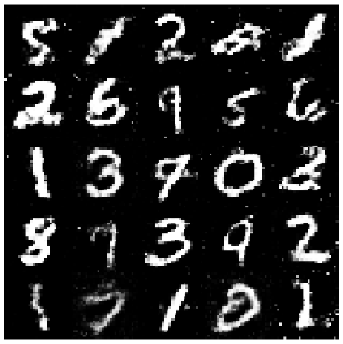
The left one is the result with original model, and the right one with additional drop out layer in the generator, from now on, it is hard to see whether the drop out layer helped or not.

Fourth, I tried to add the additional drop out layer to the discriminator and see what will happened:

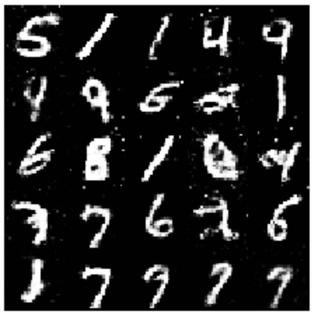
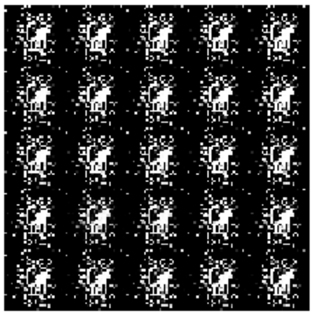
This result is also very interesting, since from the tips link, it said that adding drop out layer to the generator will improve the performance, but from the result, we can see that when adding the drop out layer to the discriminator, the generator actually learned more details from the dataset when the training epoch is same.

After that, I tried to adjust the percent of drop out to 30%, 40% and 50%:

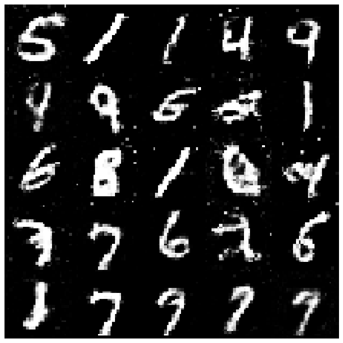
From the above result, discriminator with different percent of drop out does not have too much difference, 30% and 50% perform a little better than 40%, but almost same with 30 training epochs.

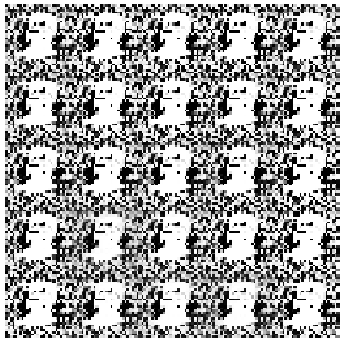
Then, I try to keep the 30% drop out of the discriminator, and try a different optimizer for the generator, since the tips mentioned that SGD for discriminator and adam for generator will be the best choose.

From the result I got above, it seems that using adam for both generator and discriminator performs better than using SGD for discriminator and adam for generator.

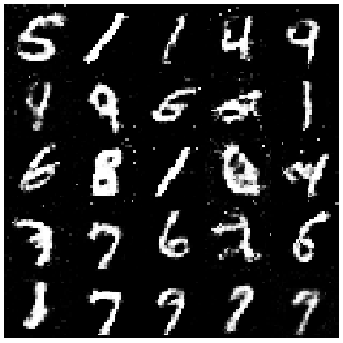
And then, I study the importance of normalize the data into range (-1,1), which is same as the range of the output of tanh function, that means preprocessing the data as the same range of the generator's output.

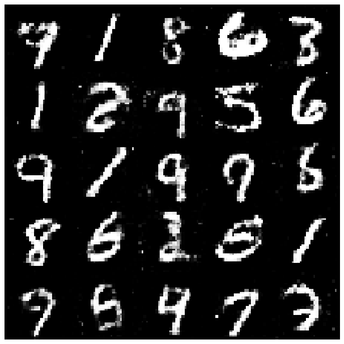




The left one is trained with preprocessed data, and the right one is trained with original data, we can see that normalize the data into the same range of the generator's output really helped a lot.

Then, I tried to add additional batch normalization layer into generator, between the dense layer and the activation function.





From the result, it shows that after normalized the data before training, add additional batch normalization layer does not helped a lot, the performances are almost same under 30 training epochs.

2) Introduce Convolutional layer into the GAN model

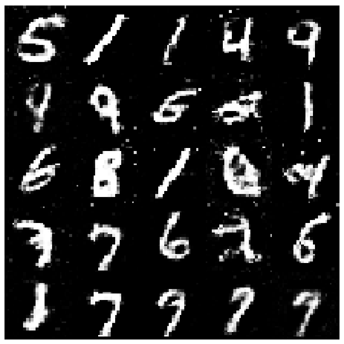
After play around with fully connected dense neural network, I tried to introduce 2D convolutional layer into the model.

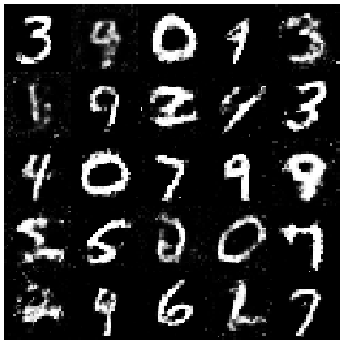
Structure: (1) Maintain the discriminator with the best performance one when using fully connected dense neural network, which has five dense layers with the number of units 1024,512,256,128,1, and with 30% of the drop out for each layer. Change dense layer inside the generator from dense layer to 2D convolutional layer, keep the first layer as dense layer with 784 units. And using a Reshape layer shape the output to (28,28,1), which can be the input of the Conv layer, following that by three Conv layers, with the same kernel size (3,3), set the padding to same, which will maintain the dimension of the image, and the filter sizes are 8,16,32.

(2) Activation function is leakyReLU, and for generator, the activation function of the output layer is tanh, and sigmoid for the discriminator. Loss function is binary cross entropy, and the optimizer is adam.

(3) Batch size is 128, number of epochs of training is 30 most of the time.

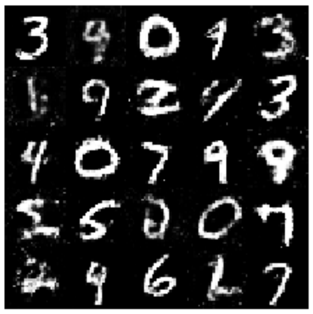
First, compared the new one with Conv layer only in the generator with the FC dense model.

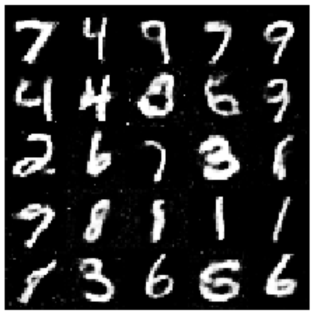




The left one is FC model, and the right one using Conv in the generator. The result does not have much differences. But using the Conv layer significantly increased the training time, even trained it with a 4 GPU instance using Microsoft Azure, it still costed almost an hour to train it only for 30 epochs.

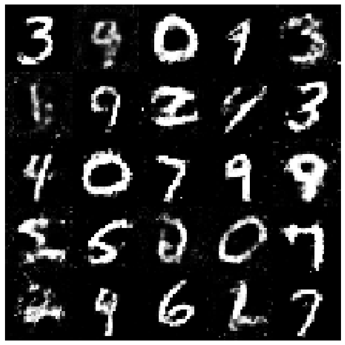
Then I tried to train it for 100 epochs, which really a time-consuming task.

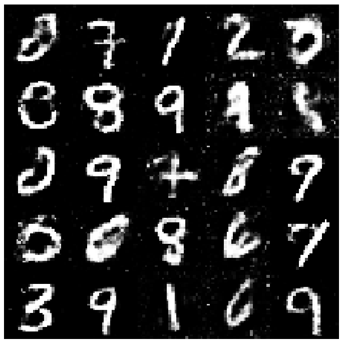




The result improved a lot after enough training epoch, and I believe that if we have enough computational resources to train it, the Conv model will perform better than the simple FC neural network.

Then, I tried to change the number of filters at each Conv layer, from 8,16,32 to 16,32,64.





From the result, we can see that, when increasing the filter number of the Conv layer, the performance will be a little better, but almost the same. But during the process, the training time increased a lot.

2. Special Skills

In the first section, I already mentioned several skills which intend to improve the performance of the GAN model according to the tips link. But some of them actually didn't work as expect. Now, let make a conclusion about all the special skills used during the training process.

1) Normalize the inputs

That definitely helped a lot, when normalize the images between -1 and 1, and using tanh as the last layer of the generator output, the result improved a lot. And it is easy to explain, since through researchers' studies, tanh performs better than sigmoid function, so if we can form the data in the same range as the generator output, that will make the loss function act more effective.

2) Sample from a gaussian distribution

This method also helps a lot, I tried uniform distribution and gaussian distribution as the input noise, and gaussian distribution highly improved the model's performance.

3)Batch Normalization

After normalized the image before training, batch normalization doesn't help a lot from my experience.

4)DCGAN

I only tried implement convolutional neural network in the generator, it performs better but meanwhile, the training time significantly increased. As for the simple picture generation, for example, MNIST dataset, when the computational resources are not enough, I think fully connected dense net is a better choice. Of course, if the computational resources are sufficient, DCGAN will be a good choice. That's somehow a trade-off.

4) Optimizer

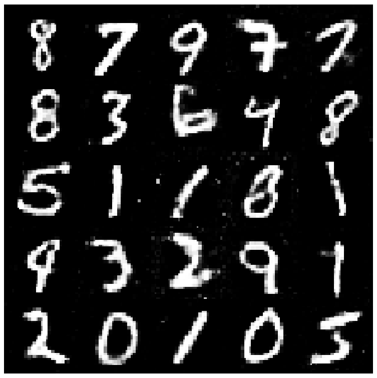
From the tips link page, it said that use SGD for discriminator and ADAM for generator will be a better choice. But from my experience, using ADAM for both of them actually performed better.

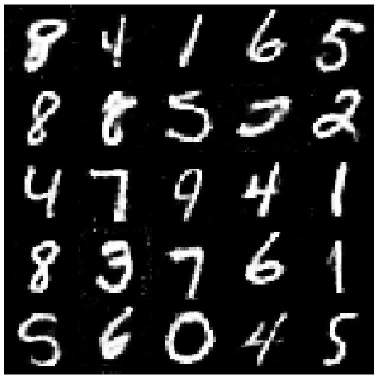
5) Use Dropouts in generator

I also got a different result from the tip link page for this advice, from my experience, using dropouts in discriminator will help a lot, and both 30% and 50% perform good.

3. Best Result

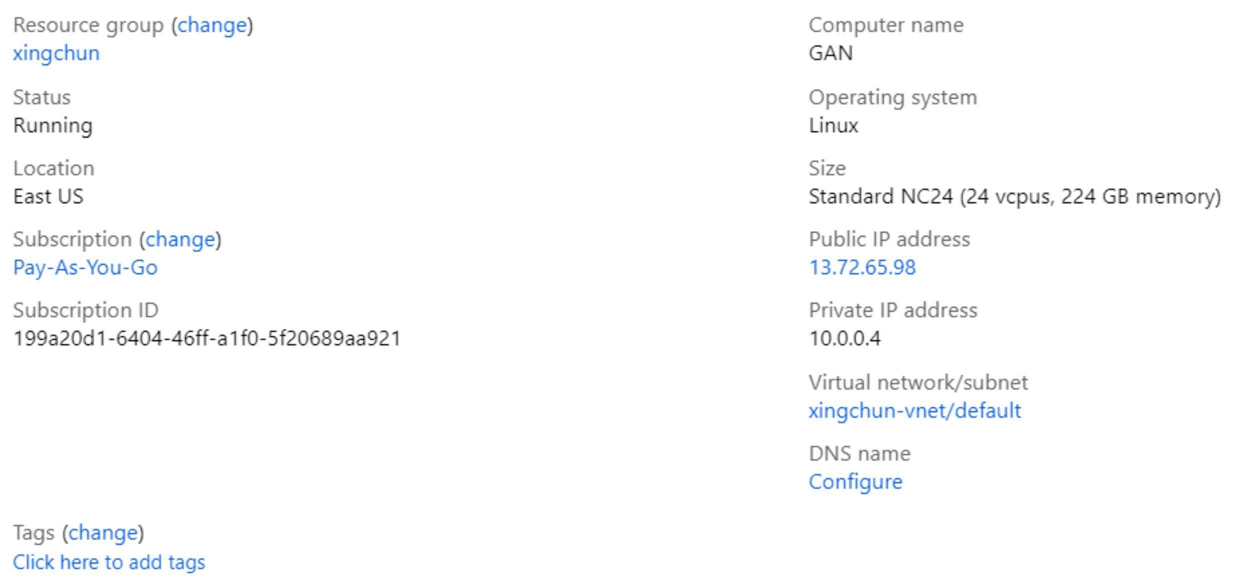
I got several acceptable results from different model. The results are showed below:





They can definitely be better if I train some of the models for more epochs, but because of the insufficient GPU resources, that's the best I can get.

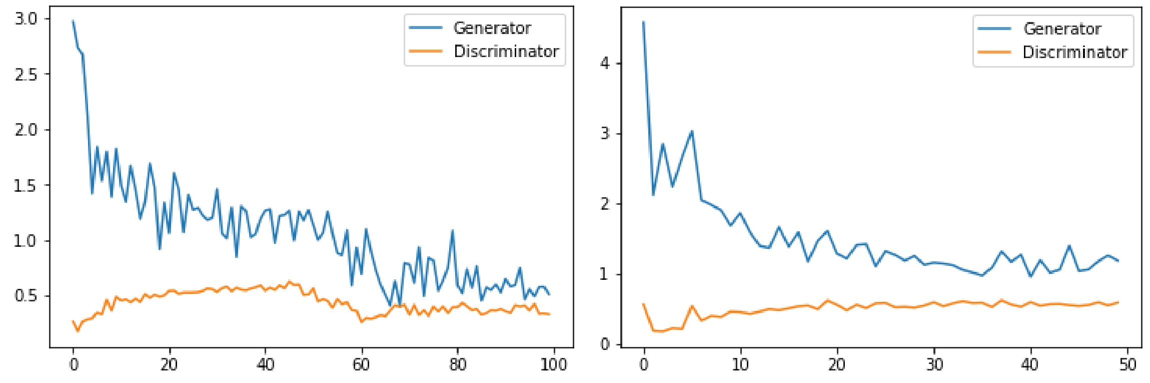
Here is the instance I used for training: (Microsoft Azure GPU Instance)



Resource group (change)	Computer name
xingchun	GAN
Status	Operating system
Running	Linux
Location	Size
East US	Standard NC24 (24 vcpus, 224 GB memory)
Subscription (change)	Public IP address
Pay-As-You-Go	13.72.65.98
Subscription ID	Private IP address
199a20d1-6404-46ff-a1f0-5f20689aa921	10.0.0.4
	Virtual network/subnet
	xingchun-vnet/default
	DNS name
	Configure
Tags (change)	
Click here to add tags	

It's a standard NC24 GPU instance, with 24vcpu and 4 GPUs, but even with that configuration, when adding the convolutional layer, the training time for each epoch can be several minutes.

For the generator and discriminator loss, here is the best result (I mean the closest result as you showed in class, Lol):



For one hundred and fifty epochs.

4. What I learned from the project

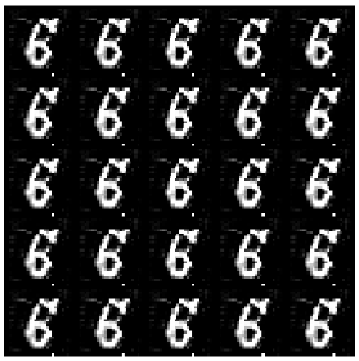
But I found that when I use the Conv version generator, the loss will not look like that, both generator and discriminator loss are stable, and almost fixed at some value. And sometimes, it is hard to explain the loss. And it seems that there is no uniform standards for the loss of a good GAN model, I tried a lot of different model, some have the same loss tendency as the above one, but in fact the generator was keeping create some garbage.

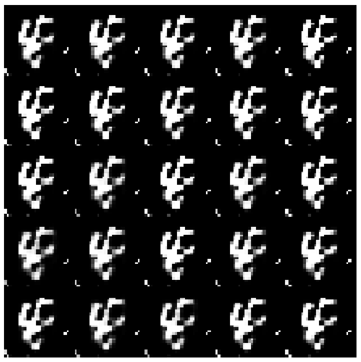
The theoretical conclusions are quite different from the actual application of the GAN model. In most cases, neither the generator nor the discriminator will learn according to the ideal situation, but interestingly, many times they can learn the key feature.

For example, when the discriminator's loss tends to zero, it is a symbol of GAN model failure, but under some special GAN models, such as Cyclegan, even when the discriminator's loss is equal to 0, the entire GAN model can still rely on Cycle loss to continue learning.

Therefore, parameters adjustment is more complicated. In many cases, I don't know if my model is not good enough or the number of training epochs is insufficient. Under limited training resources, it takes a lot of time and computing resources to find the answer. Most of the time, the tendency of the loss can't be clearly explained.

5. Some interesting Garbage





That's my "smart" generator learned, to create such garbage and successfully fool my "stupid" discriminator.