Reading And Special Problem Report

Name: Xu Ouyang

CWID: A20361390

Abstract

This is a report for reading and special problem. We discuss a model that generates images from natural language descriptions and implement it using several types of neural networks such as DCGANS and LSTM. DCGAN, including a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. LSTM, a recurrent neural network in order to learn the relevance among each word in sentence. After training on flickr-30k dataset, we hope our model can input description annotation and output image which is related to this annotation.

1 Introduction

There exists some promising researches of computer vision and image understanding in natural image statistics. One of research direction is exploit the inference and generative capabilities of deep neural networks. The problem of generating images from visual descriptions gained interest in the research community, but it is far from being solved. Previously study on generating images using defined distributions which were conditioned on classification labels.

In this paper, we illustrate how recurrent neural network and generative adversarial network can be used to translate descriptions into real images. For example, “Two young guys with shaggy hair loot at their hands while hanging out in the yard.” or “Two men in green shirts are standing in a yard” as in figure 1. We mainly use the flickr-30k dataset along with five text descriptions per image we collected as our evaluation setting. We also use word2vec which is a group of related models that are used to produce word embeddings in each description. We demonstrate its performance both on the training set categories and on the testing set.

2. Related work

Many researchers have recently exploited the capability of deep convolutional decoder networks to generate realistic images. Dosovitskiy et al. (2015) trained a deconvolutional network (several layers of convolution and upsampling) to generate 3D chair renderings conditioned on a set of graphics codes indicating shape, position and lighting. Yang et al. (2015) added an encoder network as well as actions to this approach. They trained a recurrent convolutional encoder-decoder that rotated 3D chair models and human faces conditioned on action sequences of rotations. Reed et al. (2015) encode transformations from analogy pairs, and use a convolutional decoder to predict visual analogies on shapes, video game characters and 3D cars.

1.Two young guys with shaggy hair look at their hands while hanging out in the yard.

2. Two young, white males are outside near many bushes.

3. Two men in green shirts are standing in a garden.

4. A man in a blue shirt standing in a garden.

5. Two friends enjoy time spent together.

Figure 1: Examples of description translate into real image.

Generative adversarial networks (Goodfellow et al., 2014) have also benefited from convolutional decoder networks, for the generator network module. Denton et al. (2015) used a Laplacian pyramid of adversarial generator and discriminators to synthesize images at multiple resolutions. This work generated compelling high-resolution images and could also condition on class labels for controllable generation. Radford et al. (2016) used a standard convolutional decoder, but developed a highly effective and stable architecture incorporating batch normalization to achieve striking image synthesis results.

In contemporary work Mansimov et al. (2016) generated images from text captions, using a variational recurrent autoencoder with attention to paint the image in multiple steps, similar to DRAW (Gregor et al., 2015). Impressively, the model can perform reasonable synthesis of completely novel (unlikely for a human to write) text such as “a stop sign is flying in blue skies”, suggesting that it does not simply memorize. While the results are encouraging, the problem is highly challenging and the generated images are not yet realistic, i.e., mistakeable for real. Our model can in many cases generate visually-plausible 64×64 images conditioned on text, and is also distinct in that our entire model is a GAN, rather only using GAN for post-processing.

Building on ideas from these many previous works, we develop a simple and effective approach for text-based image synthesis using a character-level text encoder, DCGAN and LSTM. We propose a novel architecture and learning strategy that leads to compelling visual results. We focus on the case of fine-grained image datasets, for which we use the recently collected descriptions for flickr-30k images with 5 human-generated captions per image.

3. Background

3.1 Word2vec

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

Word2vec can utilize either of two model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram. In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag-of-words assumption). In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weights nearby context words more heavily than more distant context words. According to the authors' note, CBOW is faster while skip-gram is slower but does a better job for infrequent words.

After learning word2vec and glove, a natural way to think about them is training a related model on a larger corpus, and English wikipedia is an ideal choice for this task.

3.2 Generative adversarial networks

Generative adversarial networks (GANs) consist of a generator G and a discriminator D that compete in a two-player minimax game: The discriminator tries to distinguish real training data from synthetic images, and the generator tries to fool the discriminator. Concretely, D and G play the following game on V(D,G):

$$\min_G \max_D V(D, G) = E_{x\sim pdata(x)}[\log D(x)] + E_{x\sim pz(z)}[\log(1 - D(G(z)))]$$

Goodfellow et al. (2014) prove that this minimax game has a global optimium precisely when pg = pdata, and that under mild conditions (e.g. G and D have enough capacity) pg converges to pdata. In practice, in the start of training samples from D are extremely poor and rejected by D with high confidence. It has been found to work better in practice for the generator to maximize log(D(G(z))) instead of minimizing log(1 − D(G(z))).

3.3 Long short-term memory

Let’s introduce RNN at first. A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented connected handwriting recognition or speech recognition.

Long short-term memory (LSTM) is a special form of recurrent neural networks (RNNs), which process sequence data. LSTM uses a few gate vectors at each position to control the passing of information along the sequence and thus improves the modeling of long-range dependencies.

While there are different variations of LSTMs, here we present the one adopted by Rocktaschel et al. (2016). ¨Specifically, let us use X = (x1, x2, . . . , xN) to denote an input sequence, where xk∈ R(1⩽k⩽N). At each position k, there is a set of internal vectors, including an input gate ik, a forget gate fk, an output gate ok and a memory cell ck. All these vectors are used together to generate a d-dimensional hidden state hk as follows:

ik = σ(Wixk + Vihk−1 + bi),

fk = σ(Wfxk + Vfhk−1 + bf),

ok = σ(Woxk + Vohk−1 + bo),

ck = fk ⊙ck−1 + ik ⊙tanh(Wcxk + Vchk−1 + bc),

hk = ok⊙tanh(ck),

where σ is the sigmoid function, ⊙is the elementwise multiplication of two vectors, and all W* ∈ Rd×l, V* ∈Rd×d and b* ∈Rd are weight matrices and vectors to be learned.

4. Implement details

Our approach is to train a deep convolutional generative adversarial network(DCGAN) conditioned on description features of n-1 words in each sentence, and only the generative part of DCGAN on description features of last word in each sentence using Euclidean distance as the loss function. What's more, we add a LSTM neural network above the generative model and under the discriminative model.

4.1 Network architecture

The architecture of network is as follows:

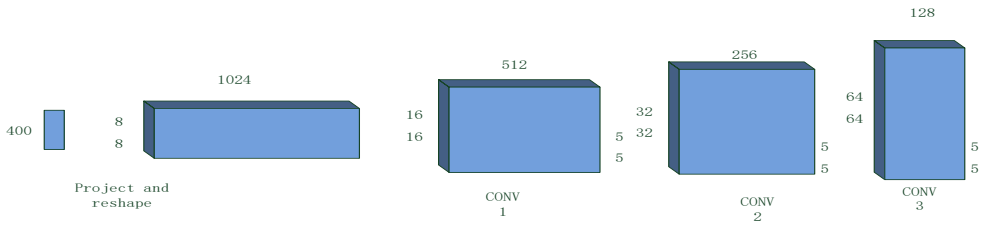


Figure 2: Generative model in DCGAN. A 400 dimensional word2vec Z is projected to a small spatial extent convolutional representation with many features maps. A series of three fractionally-strided convolutions then convert this high level representation into a 64x64 pixel image.

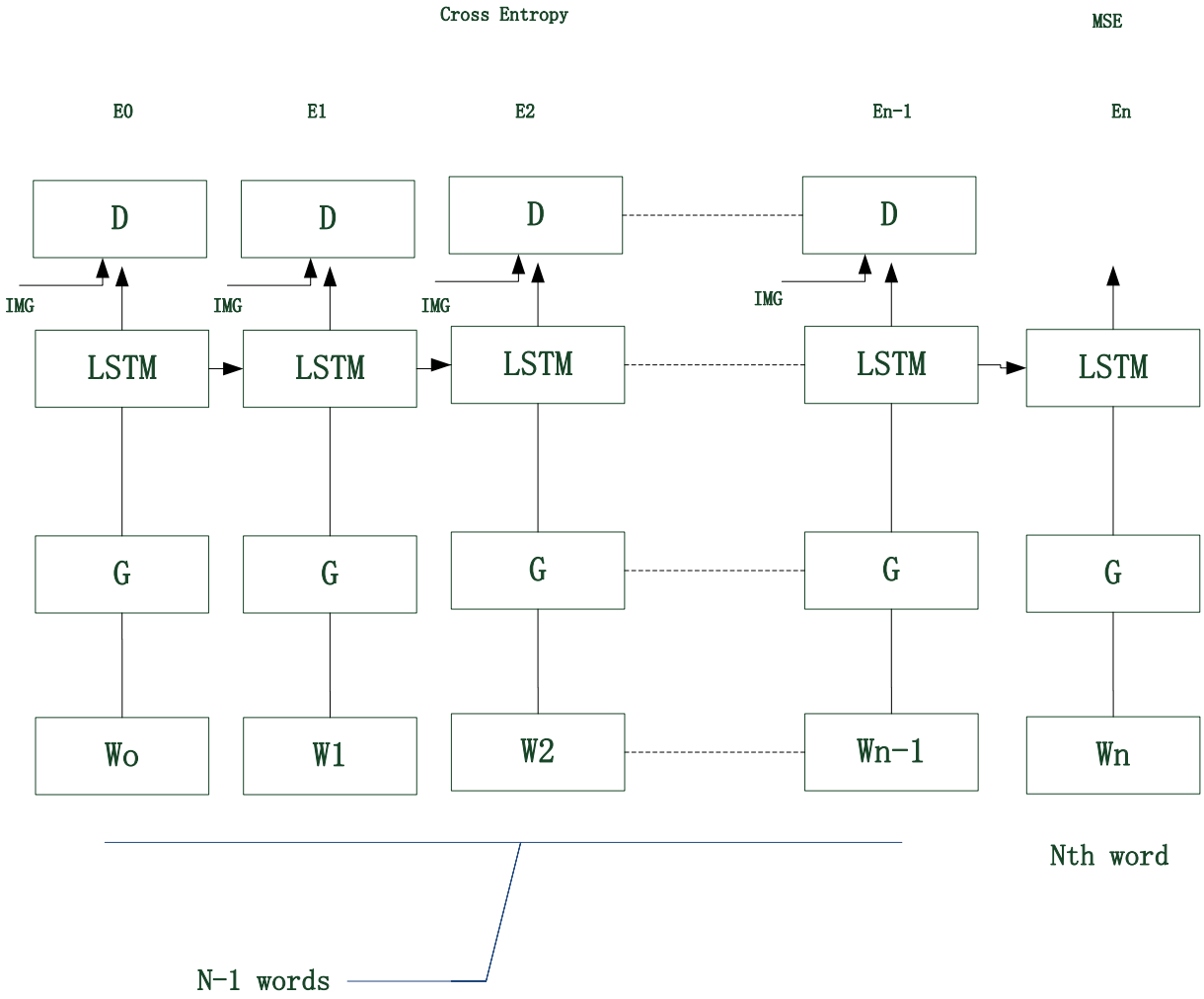


Figure 3: The whole network including DCGAN, LSTM.

There are two parts in the network, the first part we input 24 words which each of them is 400 dimensional word2vec features into the generator. The generator would output 24 number of 64x64 dimensional features via three deconvolutional layers. Meanwhile, the second part we input the last word of sentence which is 400 dimensional word2vec features into the generator. The generator output 64x64 dimensional features via three deconvolotional layers. And then we concatenated these outputs of two parts as the input of LSTM. We use the 1~24th output of LSTM and the real images as the inputs of discriminator, As a result, we hope the output of the last LSTM would be similar to the real image of this description as much as possible.

4.2 Algorithm

Input: minibatch n-1 word of 400 dimensional word2vec features, minibatch n-1 real images, minibatch nth word of 400 dimensional word2vec features, minibatch 1 real images.

For number of training iterations do

For k steps do

- Sample mninibatch of m number of n-1 word of 400 dimensional word2vec features from pg(Z)
- Sample minibatch of m number of n-1 real images from data generating distribution pdata(x)
- Update the discriminator by ascending its stochastic gradient:

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

End for

- Sample minibatch of m number of nth word of 400 dimensional word2vec features from pg(z)
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

End for

The discriminator would label the real images as one, and the generated images as zero. We use binary cross entropy as the loss function. We compute the Euclidean distance between 25th output of LSTM and the real image which is the same image of this description. And then we combine the binary cross entropy of generator with Euclidean distance as the total loss function to update the gradient of generator part. So the stochastic gradient of updating generator formula becomes:

$$\nabla_{\theta_d \frac{1}{m}} \sum_{i=1}^{m} (\log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) + ||Og - R||)$$

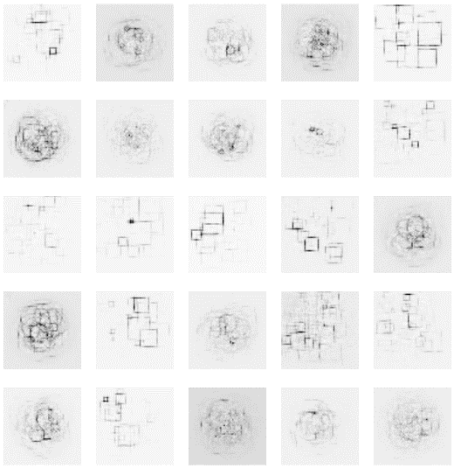
Where the Og represents the output of generator model, and the R represents the real images.

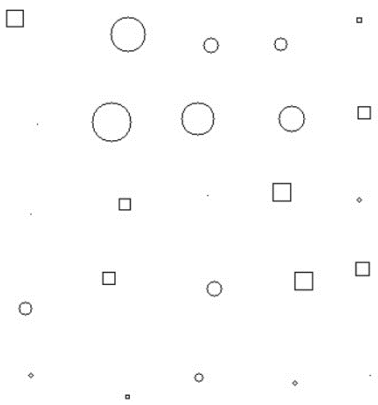
5. Experiment and discussion

In order to verify the generative neural network (GNN), DCGAN, LSTM, I did plenty of experiments.

(1). Circle and Square

First, I generated 20,000 circle and square which have different size, position. Then I use four dimensional features including the binary feature, radius of circle and side length of square, the x, y coordinate of center of circle and square as the input of GNN, the result is as the follow:

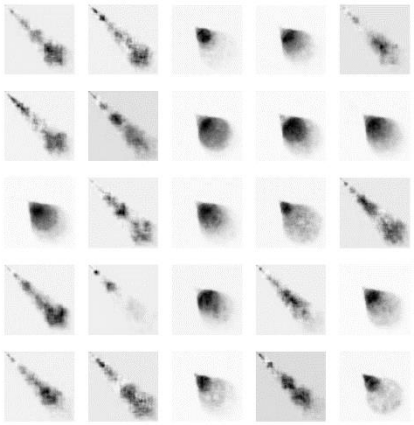


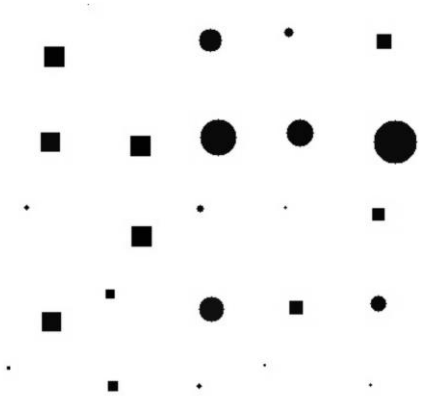


Epoch = 1000th generated image Epoch = 1000th real image

As we can see, the generated image is so awful. We found the reason is that there is no color in the circle and square which leads to no big difference among the background and the object.

So next, we filled up the object with black color, the new results are as the follows:

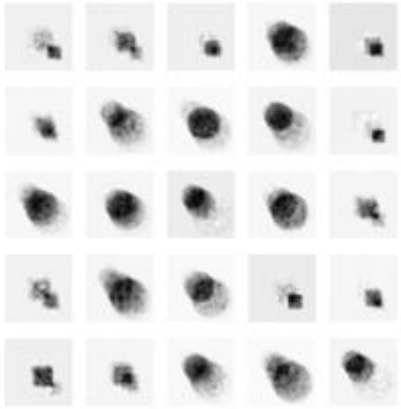




Epoch = 1100th generated image Epoch = 1100th real image

As we can see, the result of generated imaged is better than before. Basically, we can see that each generated image correspond to the real image, however, there exist an orbit in each generated image. The reason is that we generated the original images along from upper left corner to bottom right corner direction. What's more, the x,y coordinate of center of circle and square are not good features as the input.

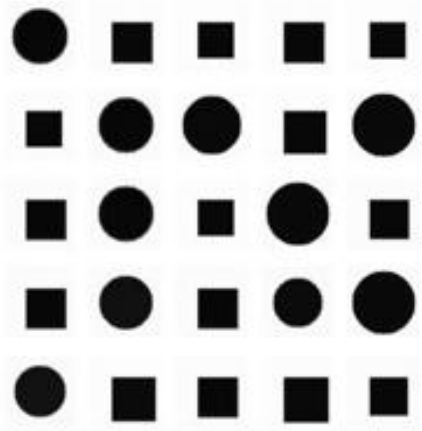
So next, I decided to generate images to arbitrary positions. And there are 6000 training samples, 1000 validation samples and 1000 test samples. I tried to run about 2000 epochs to see whether the training loss can be decreased. And the result is as the follows:

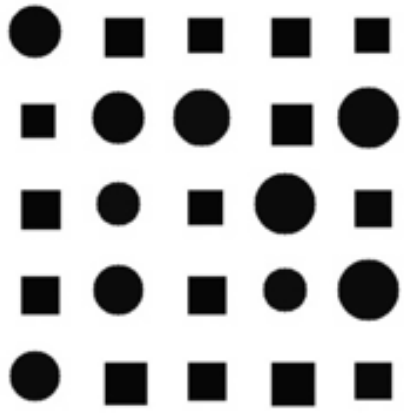


Epoch = 6000th generated image Epoch = 6000th real image

As we can see, the results are kind of blurry and there are more than one object in each image which is forbidden. We think that maybe the x, y coordinates of center of objects are not suitable features as the inputs.

So next, I decided to reduce my input features that I only used two features as the input: binary parameter, the scale size of circle/square compared with four features before: binary parameter, the scale size of circle/square, the center coordinates of x and y. And there are 6000 training samples, 1000 validation samples and 1000 test samples. I tried to run about 2000 epochs to see whether the training loss can be decreased. And the result is as the follows:



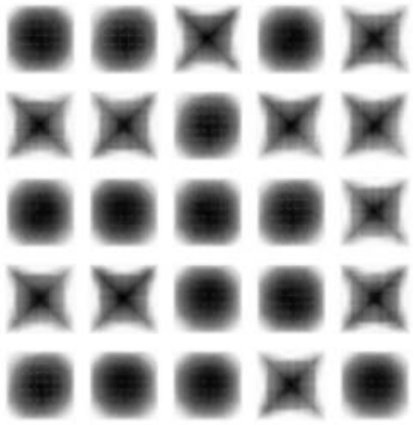


Epoch = 2000th generated image Epoch = 2000th real image

As we can see, the new generated images perform much better than before. We have gotten a pretty good result. But we was thinking about why this happened. Typically, we know that the more features we have, the better performance of network we can get. So this result looks like kind of weird.

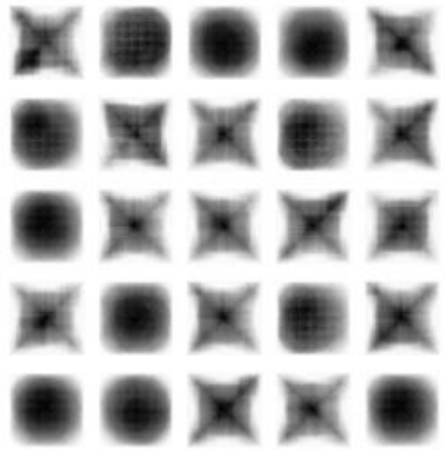
Next, as what I and professor talked about, I tried to analyze the reason of performance of the network. I did two group experiments. First, I generated the circle and square images whose

situations are not fixed, as you can see in the figure 1, and recorded two features(including binary feature and scale size) and four features(including binary feature, scale size, coordinate x and y of center of circle and square) separately. And then I generated image according to two group features, and the result after 300 epochs is as the follows:



Epoch = 300th generated image Epoch = 300th real image

Two features group



Epoch = 300th generated image Epoch = 300th real image

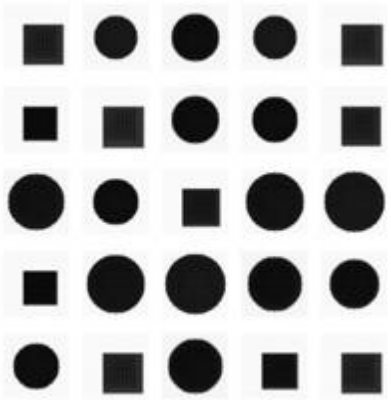
Four features group

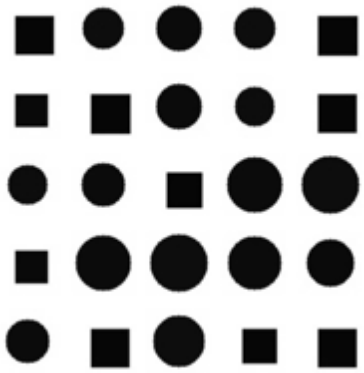
As you can see, the result is not good whatever features I used. Also, I recorded the training and validation error of each group as the follows:

Two features loss curve

Four features loss curve

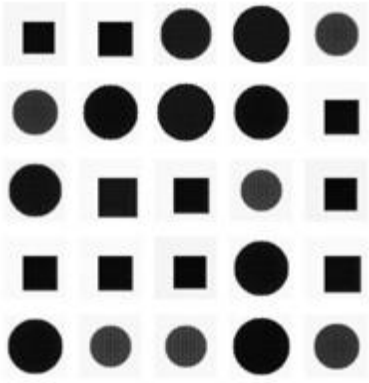
As we can see, both of the training and validation error are kind of high. I supposed that this neural network performed not well when the situation of image changed a lot. So second, I generated the circle images whose situation are fixed and the square images whose situation are changing in quarter of the whole image, as you can see in the figure 6, and recorded two features (including binary feature and scale size) and four features (including binary feature, scale size, coordinate x and y of center of circle and square) separately. And then I generated image according to two group features, and the result after 300 epochs is as the follows:

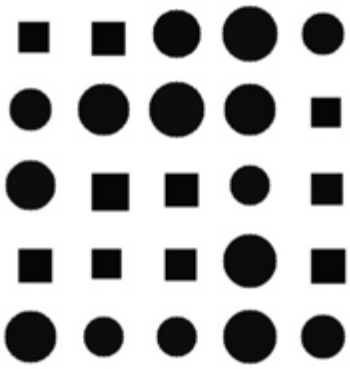




Epoch = 300th generated image Epoch = 300th real image

Two features group

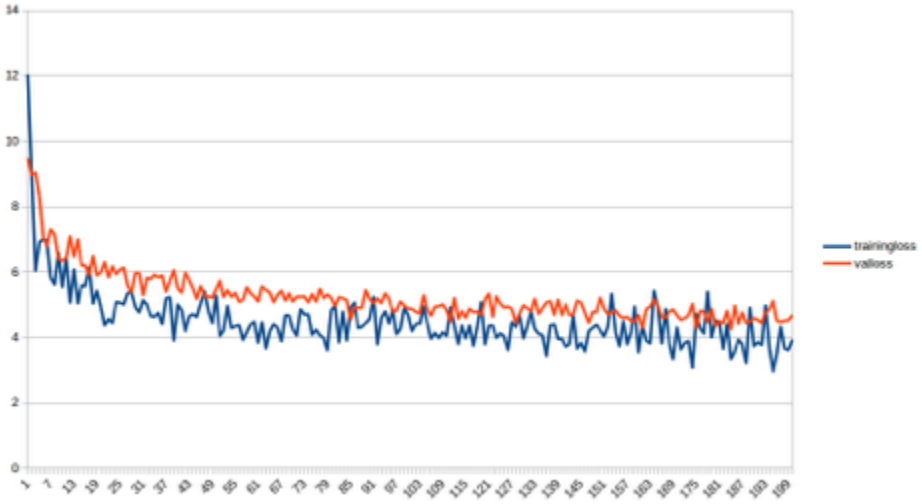


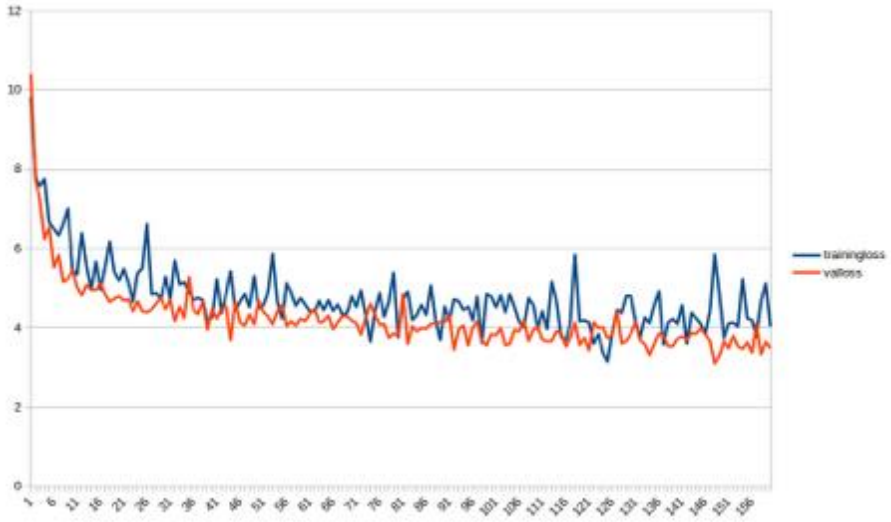


Epoch = 300th generated image Epoch = 300th real image

Four features group

As we can see the generated images look like the real image pretty much. Also, I recorded the training and validation loss of two groups as the follows:

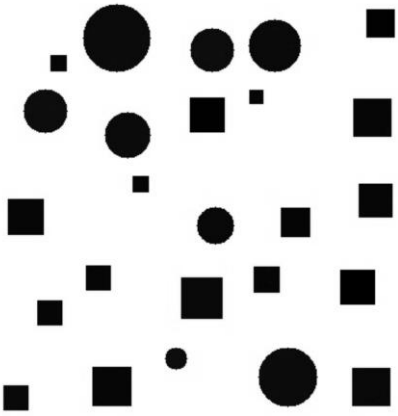




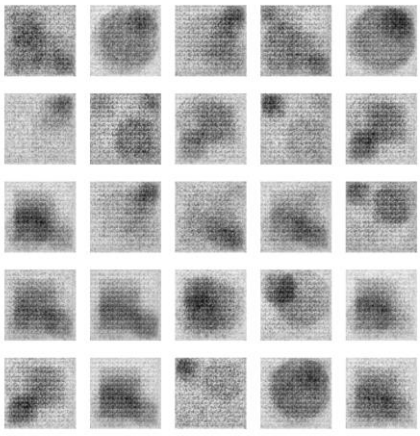
Four features

As you can see, the training and validation loss decreased continually. We got a good performance at the end. In conclusion, the original dataset is pretty important to neural network. This deconvolution network can't learn the different situation of images.

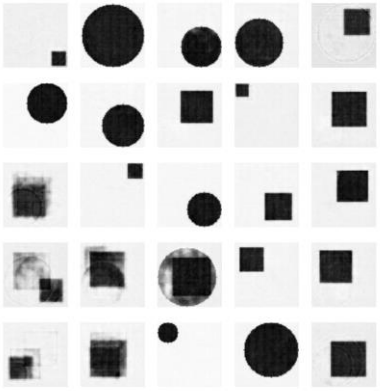
In order to find the reason of bad performance in simple generative CNN, I built up an auto-encoder generative CNN. First I set up the number of medium of outputs in auto encoder to 4(which are 4 features actually), and the output of network is as follows:



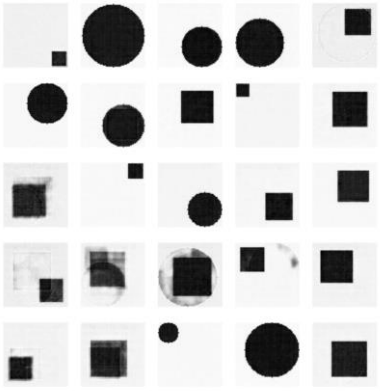
original image



epoch = 1st



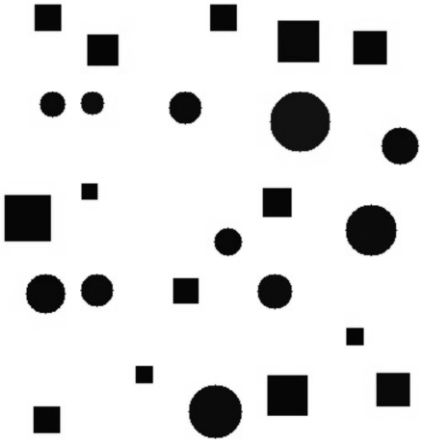
epoch = 500th



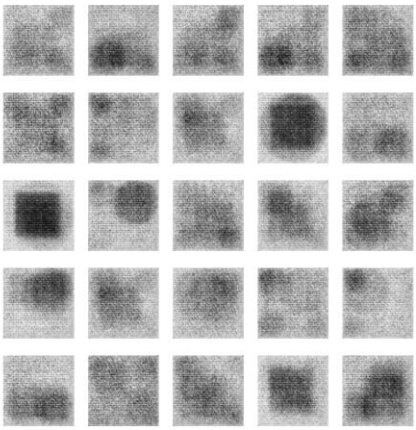
epoch = 1000th

As we can see, the auto encoder gets much better results than before, however there are still some overlapped images. I guess the reason is that the number of features are not enough, so I increase the number of features next.

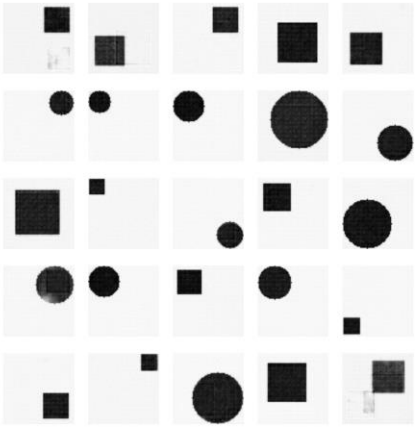
Next, I set up the number of medium of outputs in auto encoder to 5, and the output of network is as follows:

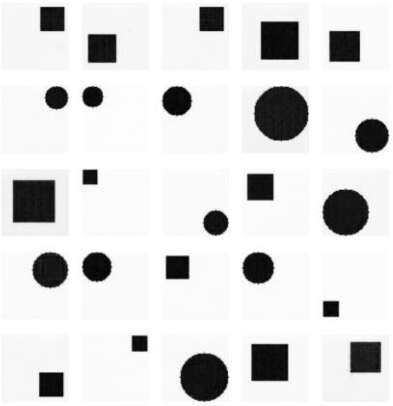


original image



epoch = 1st

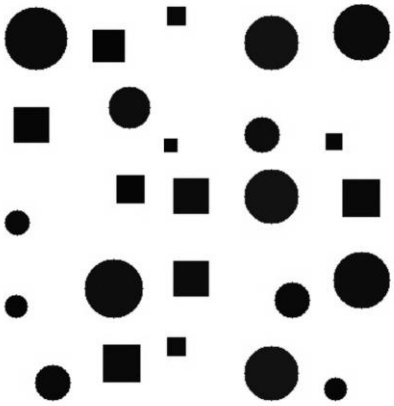


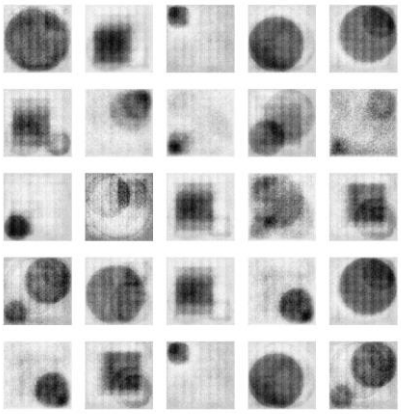


epoch = 500th epoch = 1000th

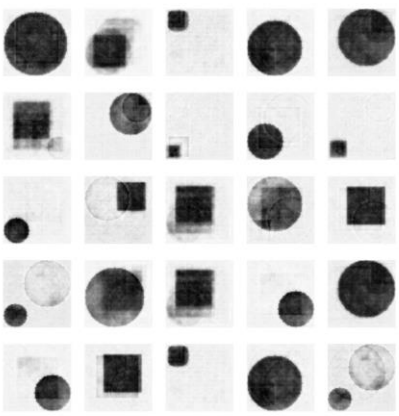
As we can see, the results are better than before which verify our suppose. I think the reason is that the way I generated four features in previous experiment: the coordinate x and y of circle is the center of circle, but x, y in square is the coordinate of top left corner vertex which means they don't present same meaning.

Next, I set up a simple generative CNN using the 4 features made by auto encoder, and the results are as the follows:

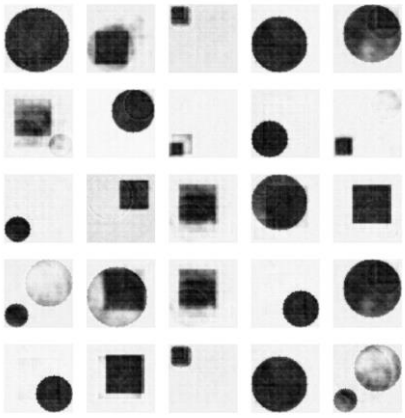




original image epoch = 10th

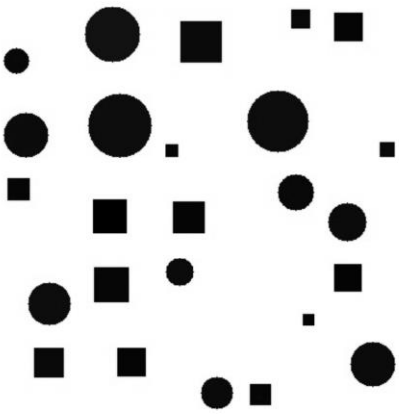


epoch =200th

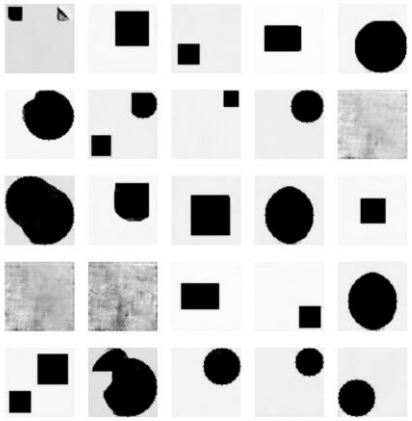


epoch = 300th

Next, we implement DCGAN on the circle and square data. First, I extracted 5 features from the center of auto-encoder network.

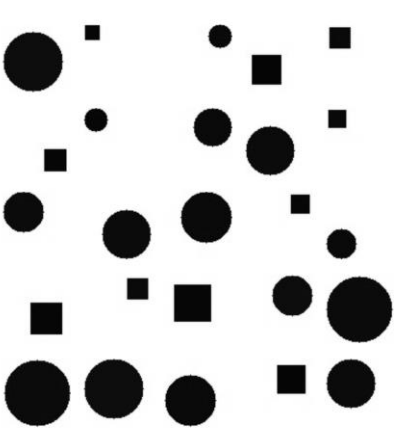


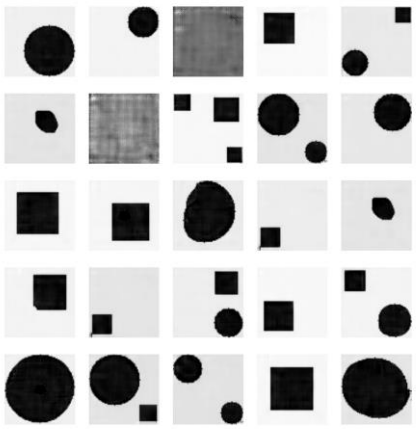
Real image



epoch = 1000th

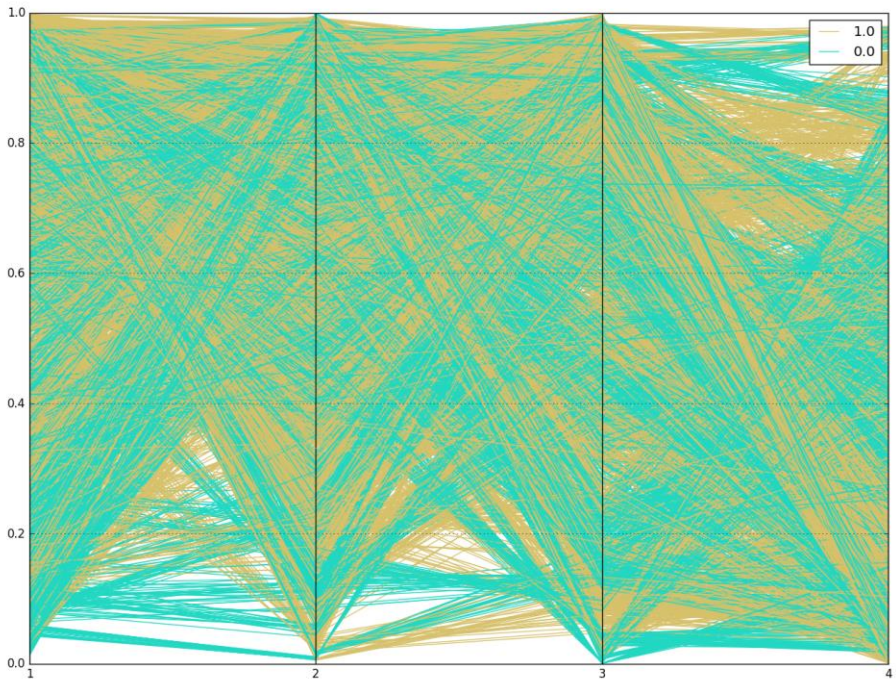
Next, in order to improve the performance of DCGAN, I tried to use several methods. I extracted 100 features from the center of auto-encoder and then used them as the input of the DCGAN, the results are as the follows:





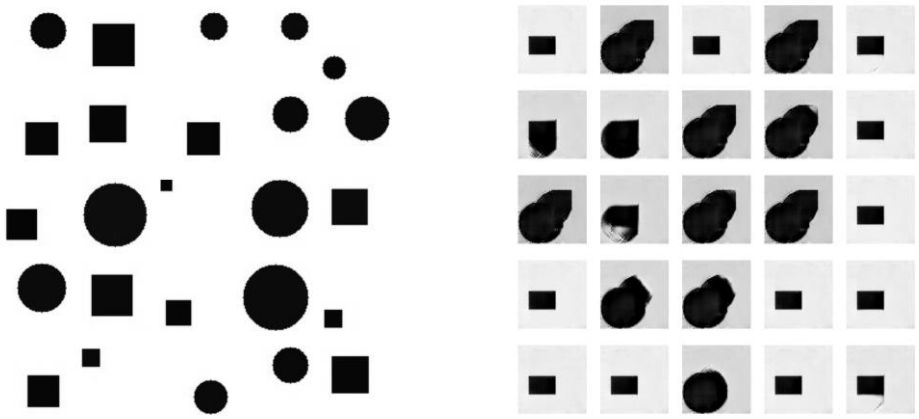
Real image epoch = 650th

As we can see, the results are not good enough but better than before which I used 4 features as the inputs. I started to think about why it can't work well. And due to the principle of DCGAN that the generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. I supposed the inputs should have more regular distribution. So I plot the output:



4 features parallel plot

It looks like no regular distribution, so I use 100 uniform distribution features (from -1 to 1) as the inputs of DCGAN, the results are as the follows:



Real image epoch = 600th

The results were still not good enough, so I tried enlarge the complexity of neural network (continual).

(2) LSTM Networks for Sentiment Analysis

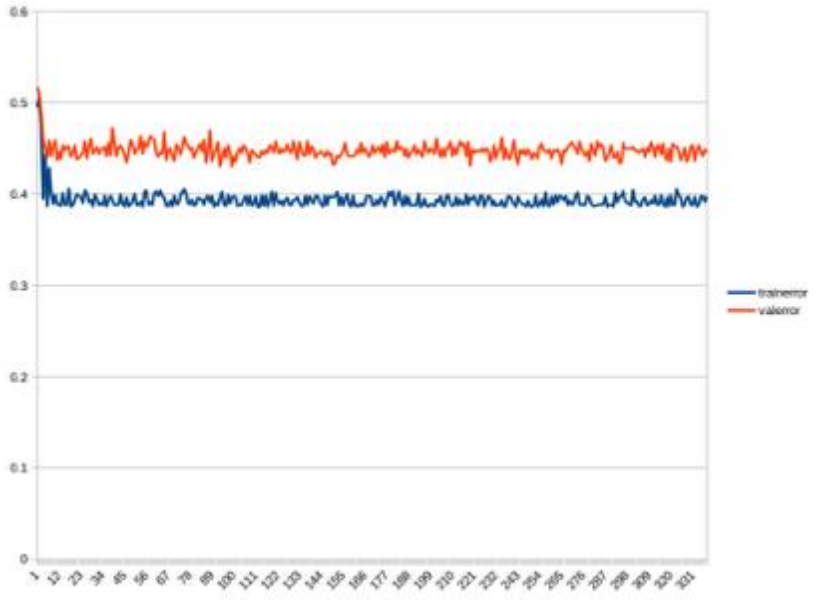
I tried to compare the LSTM with RNN and GRU algorithm. I run a sentiment analysis program based on these three networks. In this program, given a movie review, the model attempts to predict whether the emotion of this movie review is positive or negative. So this is a binary classification task. All of these three algorithm are artificial neural network where connections between units form a directed cycle.

I try to run a sentiment analysis program from a website: http://deeplearning.net/tutorial/lstm.html.

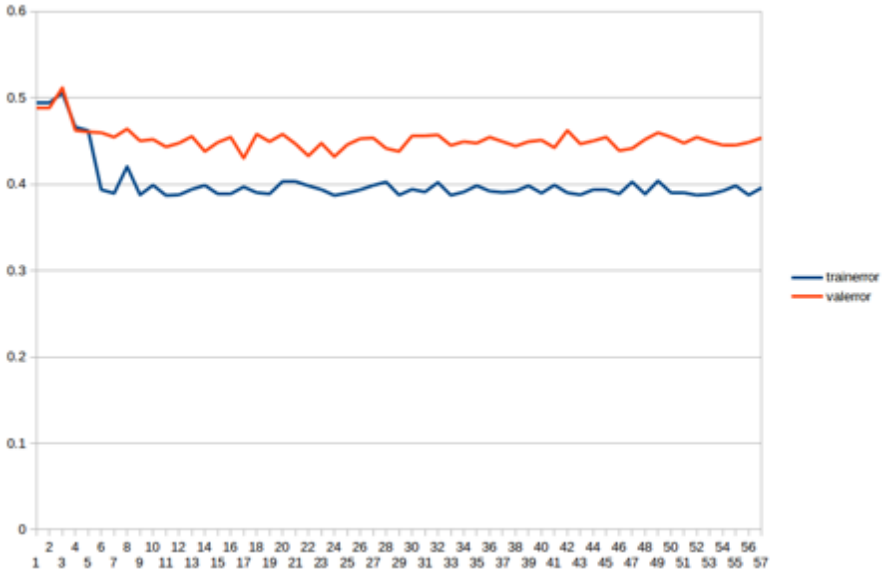
In this program, given a movie review, the model attempts to predict whether the emotion of this movie review is positive or negative. So this is a binary classification task. I choose imdb dataset which has 25000 movie reviews for training and 25000 movie reviews for testing. Here is one movie review as follows:

Due to limited time, I used 10373 movie reviews which each review's length is less than 200 as training data, 1153 movie reviews as validation data and 5000 movie reviews as testing data. I recorded the training error and validation error of three algorithms, the results are as the follows:

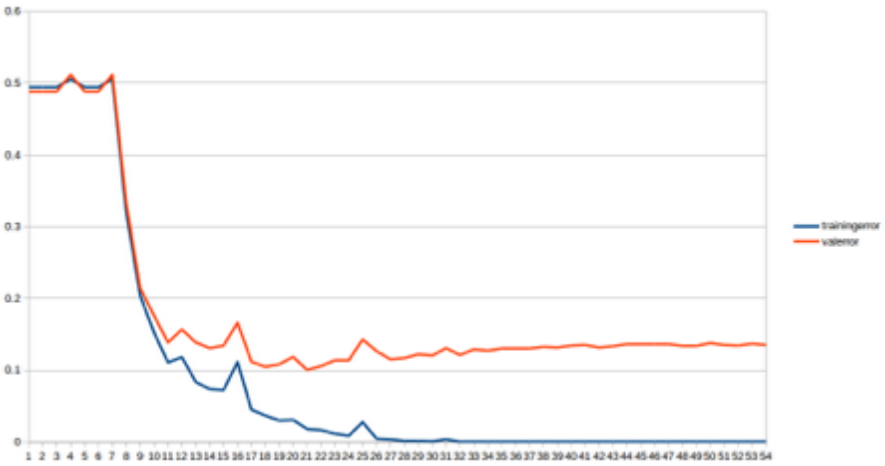
model options {'encoder': 'lstm', 'optimizer': <function adadelta at 0x106116f50>, 'validFreq': 370, 'lrate': 0.0001, 'batch_size': 16, 'decay_c': 0.0, 'patience': 10, 'reload_model': None, 'n_words': 10000, 'max_epochs': 1000, 'dispFreq': 10, 'dataset': 'imdb', 'valid_batch_size': 64, 'use_dropout': True, 'dim_proj': 128, 'maxlen': 200, 'saveto': 'lstm_model.npz', 'noise_std': 0.0, 'test_size': 5000, 'saveFreq': 1110}
10373 train examples, 1153 valid examples, 5000 test examples



RNN



GRU



LSTM

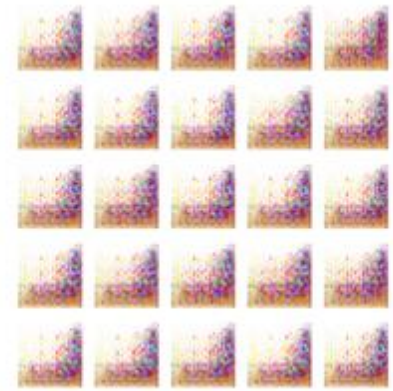
As we can see, the LSTM only use 30 epochs to converge, and the RNN and GRU are around 0.4 after 30 epochs. Maybe I need to spend time to optimize the RNN and GRU later.

To find out whether the LSTM is better than RNN and GRU, I optimized the RNN and GRU as much as possible. However, there is no obvious improvement of RNN and GRU. I tried to change the learning rate, increase the batch size, change the max length of comments, enlarge the number of unit in hidden layer and so on. To be honest, it's a hard process to fine tune the network, maybe that's why people want to improve the RNN structure to LSTM.

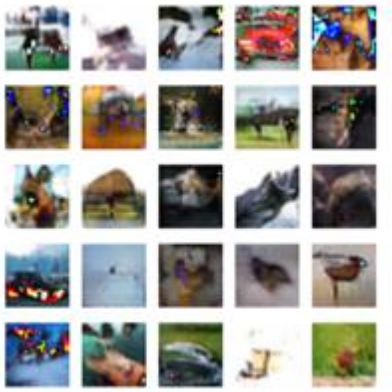
(3) MNIST and CIFAR10

I implemented DCGAN algorithm. This is a pretty interesting model. This model is made of a generative model which is similar to my previous deconvolution neural network and a discriminator model which tries to discriminate whether an input image comes from the generative model or from the real image. In this network, the generative model tries its best to generate more real image who wants to cheat the discriminator as much as possible. And the discriminator model tries its best to discriminate the fake images. It pretty look like a game. I use cifar-10 image dataset, which includes 60000 32x32 size colorful images, and they are labeled by 10 class as the follows:

The input of the network are just 100 random uniform distribution feature vectors from -1 to 1 and 32x32 real images, and the output is 32x32 generated images. And the result is as the follows:



epoch = 1th



epoch = 50th

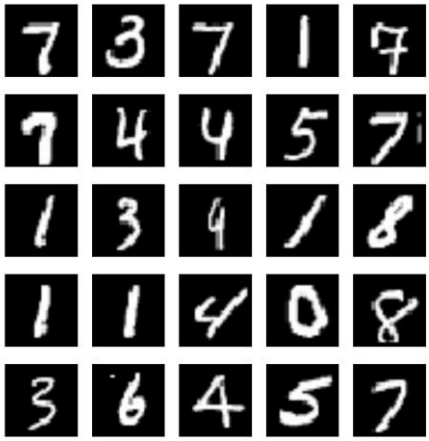
epoch = 100th

epoch = 200th

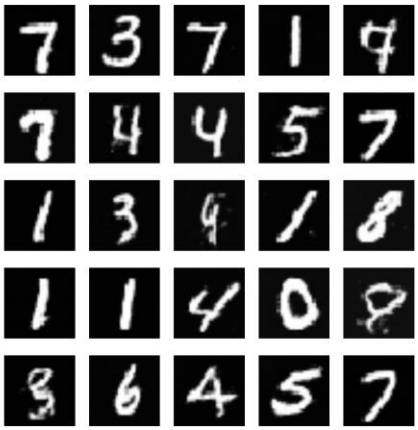
As you can see, after 100 epochs, we can get a kind of clear meaningful images. I wondered why did this neural network perform such well even if the inputs are just random values. And I discussed with Mr.Zhang, and he guessed that the process of network training might be distribution transformation, I mean the network learned how to transfer the (-1,1) uniform distribution to image distribution. Maybe I can verified this supposition and I'm still doing some research on it.

Next, I did experiments on DCGAN using mnist dataset again.

First, I extracted 100 features from the center of auto encoder. The result of auto encoder is as follow:

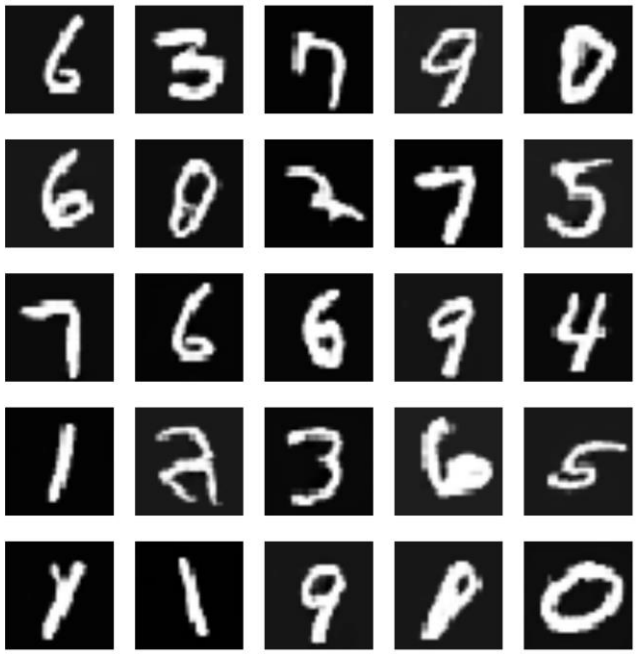


Real image



Auto-encoder: epoch 500th

Second, I used these 100 features as the input of the DCGAN, the result is as follow:



Epoch 200th

Third, I used 100 random uniform distribution features as the input of DCGAN, the result is as the follow:

Epoch 200th

5. Conclusion and future work

In conclusion, we proved that the GNN (generative adversarial neural network), DCGAN, LSTM works well.

In the next step, we will verify whether they can still get good performance on our data flickr30k. And then we will combine these three neural network together to see the output of network.

6. Reference

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B.,Warde-Farley, D., Ozair, S., Courville, A., and Bengio,Y. Generative adversarial nets. In NIPS, 2014.

[2] Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016.

[3] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee. Generative Adversarial Text to Image Synthesis, 2016

[4] Dosovitskiy, A., Tobias Springenberg, J., and Brox, T. Learning to generate chairs with convolutional neural networks. In CVPR, 2015.

[5] Mansimov, E., Parisotto, E., Ba, J. L., and Salakhutdinov,R. Generating images from captions with attention. ICLR, 2016.

[6] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. Improved Techniques for Training GANs, 2016

[7] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015