Weekly Report(July 18th, 2019)

Jianghao Lin
Shanghai Jiao Tong University
chiangel.ljh@gmail.com

Abstract

I read the paper *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks* written by Alec Radford in 2016. Also I implement DCGAN on MNIST.

1 Summary of paper DCGANs

1.1 Architecture guidelines for stable DCGANs

1. Do not use any pooling layers. Use strided convolutional layers in discriminator network and fractional-strided convolutional layers in generator network instead.
2. Use batch normalization in both generator and discriminator network. Note that directly applying batch normalization to all layers may result in sample oscillation and instability. In order to avoid this situation, do not apply batch normalization to the output layer of generator and the input layer of discriminator.
3. Do not use any fully connected layers. We can use global pooling instead which contributes to model stability but slow down the convergence.
4. Use ReLU activation in generator network for all layers except for the output layer which uses tanh activation to project values to $[-1, 1]$.
5. Use Leaky ReLU in all layers of discriminator network.

1.2 Training details for DCGANs

1. No pre-processing. Only use tanh activation to scale the range of value into $[-1, 1]$
2. SGD
3. Batch size of 128
4. All weights are initialized from a zero-center normal distribution with standard deviation 0.02
5. The slope of Leaky ReLU is 0.2
6. Adam optimizer
7. Learning rate of 0.0002
8. Momentum term β_1 is 0.5 instead of 0.9

1.3 Validate the model capacity

In this part, the author mainly measure the capacity of discriminator network as feature extractors compared with other unsupervised learning methods.

1.3.1 CIFAR dataset

We use all convolutional layers of discriminator which comes from a pre-trained DCGAN model on Imagenet-1k. We maxpool each layer representation to produce a 4×4 spatial grid. We then flatten and concatenate these grids to form a 28672 dimensional vector. By putting the output of the feature vector into a L2-SVM linear model, we can get a classification scores for supervised learning tasks.

As a result, DCGAN+L2-SVM performs better than k-means baseline method but is not as good as the Exemplar CNN(another method to apply unsupervised learning tasks with CNN).

1.3.2 SVHN dataset

After the similar approach above, we can transfer the discriminator' convolutional layers into feature vector and put it with L2-SVM. This time, DCGAN+L2-SVM outperform the previous works when labeled data is scarce.

1.4 Visualization

In this part, the author wants to mainly validate that the DCGAN model is actually learning features and representation from the image instead of simply memorizing and fitting the input image. The demonstration is various and imaginative.

1.4.1 Walking in the latent space

Interpolation applied to latent vector z results in smooth transition on the generated images. E.g. images transfer from having windows to not having windows. This indicates the feature representation in latent vector z.

1.4.2 Visualizing discriminator features

Using the *guided backpropagation*, we can visualize the last convolutional layer of discriminator network. And we can see many base outline of specific objects like a mirror or a window. This indicates the features learned in discriminator network.

1.4.3 Visualizing generator features

The author do the following two experiments, which strongly demonstrate that there is feature hierarchy architecture in generator network.

1. **Eliminate objects by modifying the generator** - Use logistic regression to predict whether a feature activation is on a window or not based on 150 manually labeled generated images. Drop all activations that are greater than zero(indicating a window). Then, applying the same vector z, generator network will produce image without window accordingly! But the generated image is also blurrier.
2. **Vector arithmetic** - Applying vector arithmetic on input z will result in feature object combination or elimination.

2 Implementation of DCGAN on MNIST

I apply DCGAN on dataset MNIST using PyTorch. By following the architecture guideline and training tutorial in the paper, I think I finally get the generative model converged after 50 epochs training (is that too much?).

The loss of generator and discriminator network are shown in figure 1.

I set a fixed noise input vector to estimate the output of generator during the training. We can clearly see in figure 2 that the fake image generated from G transfer from the noisy figure to images that indicate numbers.

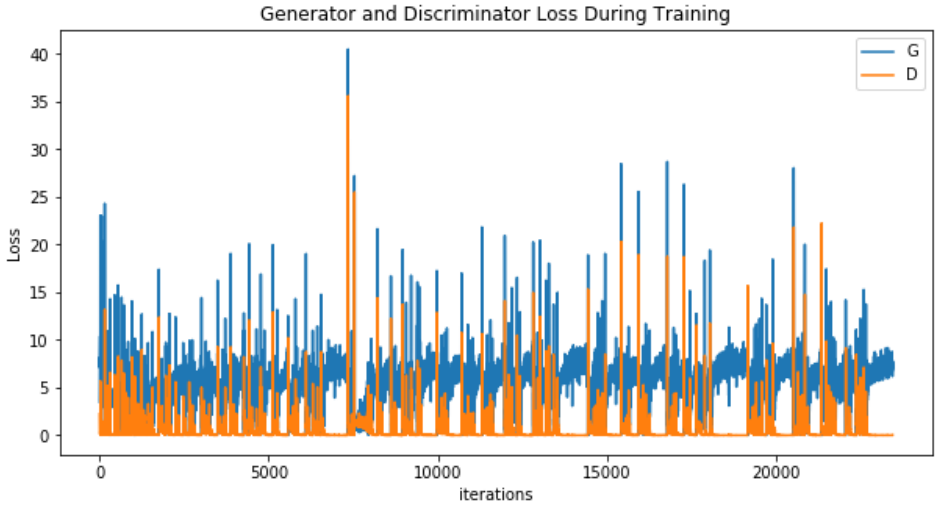
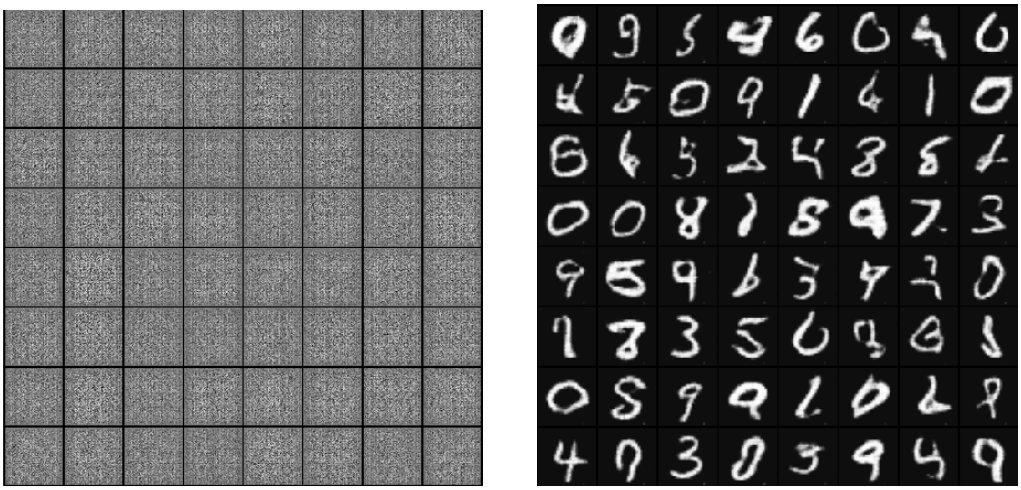


Figure 1: Loss of Generator and Discriminator Network



(a) Initial output of Generate Network

(b) Final output of Generate Network

Figure 2: Output of Generator Network