Weekly Report(July 12th, 2019)

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

Abstract

This is a collection of my paper notes for future reference.

Contents

1 Transposed Convolution **3**

2 GAN: Generative Adversarial Networks **4**

2.1 Basic idea of GAN 4

2.2 How to measure the divergence ? 4

2.2.1 KL divergence 4

2.2.2 JS divergence 4

2.3 Definitions in GANs 5

2.4 Solve $\arg\max_D V(G, D)$ 5

2.5 Why $\max_D V(G, D)$ evaluates the difference of distributions? 5

2.6 Final target - the generator 6

2.7 Tips for implementing a GAN 6

3 DCGAN: Deep Convolutional Generative Adversarial Network **7**

3.1 Architecture guidelines for stable DCGANs 7

3.2 Training details for DCGANs 7

3.3 Validate the model capacity 7

3.3.1 CIFAR dataset 7

3.3.2 SVHN dataset 7

3.4 Visualization 7

3.4.1 Walking in the latent space 8

3.4.2 Visualizing discriminator features 8

3.4.3 Visualizing generator features 8

4 CGAN: Conditional Generative Adversarial Network **9**

4.1 Motivations 9

4.2 How to combine the inputs ? 9

5 **9**

1 Transposed Convolution

Transposed Convolution, also known as Deconvolution or Fractionally-strided Convolution, is often used to up-sample images from low resolution to high resolution. Compared with some traditional interpolation method for up-sampling, transposed convolution has learnable parameters.

2 GAN: Generative Adversarial Networks

2.1 Basic idea of GAN

Generative adversarial nets are based on a game theoretic scenario where the generator network directly produces images x from a random noise z and another discriminator network tries to distinguish the generated image from the real ones.

Ideally, the distribution of the generated data will be the same as the real data image and the accuracy of the discriminator network will be 0.5 because it can not tell an image is actually fake or not.

2.2 How to measure the divergence ?

2.2.1 KL divergence

KL divergence is also called relative entropy.

For discrete probability distributions P and Q defined on the same probability space, the KL divergence between P and Q is defined to be:

$$KL(P||Q) = \sum_x P(x)log(\frac{P(x)}{Q(x)}) \tag{1}$$

For continuous random variables, it is defined to be:

$$KL(P||Q) = \int_{-\infty}^{\infty} p(x)log(\frac{p(x)}{q(x)})dx \tag{2}$$

We can have the inequation because the log function is a convex function:

$$\begin{aligned} KL(P||Q) &= \int_{-\infty}^{\infty} p(x)log(\frac{p(x)}{q(x)})dx \\ &= -E(log\frac{P(x)}{Q(x)}) \\ &\geq -log(E(\frac{P(x)}{Q(x)})) \\ &= -log(P(x) \times \frac{P(x)}{Q(x)}) = 0 \end{aligned} \tag{3}$$

Therefore, KL divergence is in range of $[0, 1]$. KL divergence is equal to 0 if and only if two distribution is exactly the same. And the smaller the KL divergence is, the more similar two distributions are.

2.2.2 JS divergence

It is obvious that KL divergence is asymmetric, which means that $KL(P||Q) \neq KL(Q||P)$. So we introduce the JS divergence:

$$JS(P||Q) = \frac{1}{2}KL(P||M) + \frac{1}{2}KL(Q||M) \tag{4}$$

where $M = \frac{1}{2}(P + Q)$. The range of JS divergence is $[0, log2]$. JS divergence is obviously symmetric and the smaller the JS divergence is, the more similar two distributions are.

2.3 Definitions in GANs

- **Generator G** - G is a function that takes a vector z and outputs x. So given a prior distribution $P_z(z)$, then a probability distribution $P_G(x)$ is defined by function G.
- **Discriminator D** - D is a function (actually a binary classification) takes x as inputs and outputs a scalar in range of $[0, 1]$ to tell the input x is faked or not.
- **V(G, D)** - The function V(G, D) is defined to be:

$$V(G, D) = E_{x \sim p_{data}}[log(D(x))] + E_{z \sim p_z}[log(1 - D(G(z)))] \tag{5}$$

  Defining V(G, D) like this allows the following missions:

  1. Given a G, $\max_D V(G, D)$ evaluates the difference between p_G and p_{data}.
  2. $\min_G \max_D V(G, D)$ picks a generator that minimize the difference.

2.4 Solve $\arg\max_D V(G, D)$

Given a G, thus distribution p_G is determined. We have

$$\begin{aligned} V &= E_{x \sim p_{data}}[log(D(x))] + E_{z \sim p_z}[log(1 - D(G(z)))] \\ &= E_{x \sim p_{data}}[log(D(x))] + E_{x \sim p_G}[log(1 - D(x))] \\ &= \int_x p_{data}(x) log(D(x)) dx + \int_x p_G(x) log(1 - D(x)) dx \\ &= \int_x [p_{data}(x) log(D(x)) + p_G(x) log(1 - D(x))] dx \end{aligned} \tag{6}$$

Because p_{data} and p_G are both fixed, for each input x, we can easily compute the result that maximizes $V(G, D)$:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \tag{7}$$

2.5 Why $\max_D V(G, D)$ evaluates the difference of distributions?

$$\begin{aligned} &\max_D V(G, D) \\ &= V(G, D^*) \\ &= E_{x \sim p_{data}}[log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}] + E_{x \sim p_G}[log \frac{p_G(x)}{p_{data}(x) + p_G(x)}] \\ &= \int_x p_{data}(x) log \frac{p_{data}(x) \times \frac{1}{2}}{(p_{data}(x) + p_G(x)) \times \frac{1}{2}} dx \\ &\quad + \int_x p_G(x) log \frac{p_G(x)}{\frac{1}{2}(p_{data}(x) + p_G(x)) \times \frac{1}{2}} dx \\ &= -2log2 + \int_x p_{data}(x) log \frac{p_{data}(x)}{\frac{1}{2}(p_{data}(x) + p_G(x))} dx \\ &\quad + \int_x p_G(x) log \frac{p_G(x)}{\frac{1}{2}(p_{data}(x) + p_G(x))} dx \\ &= -2log2 + KL(p_{data} || \frac{p_{data} + p_G}{2}) + KL(p_G || \frac{p_{data} + p_G}{2}) \\ &= -2log2 + JS(p_{data} || p_G) \end{aligned} \tag{8}$$

Therefore, we can see that $V(G, D^*)$ is the sum of a JS divergence and a constant $-2log2$. So it can evaluate the difference of distributions. And the smaller the $V(G, D^*)$ is, the more similar p_{data} and p_G are.

2.6 Final target - the generator

Our final aim is to achieve a generator G that have the same distribution p_G as p_{data}, which means the smallest divergence. So our optimal objective is

$$G^* = \arg\min_G \max_D V(G, D) \tag{9}$$

2.7 Tips for implementing a GAN

1. Due to the difficulty of computing the real expectation of distributions, we use the arithmetic mean of samples $\{x(1), ..., x^{(m)}\}$ and $\{z^{(1)}, ..., z^{(m)}\}$ to approximate it:

$$\frac{1}{m}\sum_{i=1}^{m}[log(D(x^{(i)})) + log(1 - D(G(z^{(i)})))] \tag{10}$$

2. Update the generator by descending the following stochastic gradient can be inefficient:

$$\nabla_{\theta_G}\frac{1}{m}\sum_{i=1}^{m} log(1 - D(G(z^{(i)}))) \tag{11}$$

   Because the equation indicates that the more real the generated image z is, the smaller the gradient is. That is, the gradient will be almost 0 at the begin of training and increase significantly when the generator is able to generate a good enough image! This will make the training inefficient and uneasy to converge. So we usually use the gradient below as an alternative. More real the image, less the gradient.

$$\nabla_{\theta_G}\frac{1}{m}\sum_{i=1}^{m} log(D(G(z^{(i)}))) \tag{12}$$

3 DCGAN: Deep Convolutional Generative Adversarial Network

3.1 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.

3.2 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

3.3 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.

3.3.1 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).

3.3.2 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.

3.4 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.

3.4.1 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.

3.4.2 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.

3.4.3 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.

4 CGAN: Conditional Generative Adversarial Network

4.1 Motivations

In the previous GAN models, we can not control the kind of images we will generate from a random noise z. That is, from a random noise z, we may generate a cat image, a dog image and so on. It is kind of out of control.

But in CGAN, by adding additional information like class labels or textual descriptions, we can somehow control the type of the generated images.

The general architecture of CGAN are shown in figure 1.

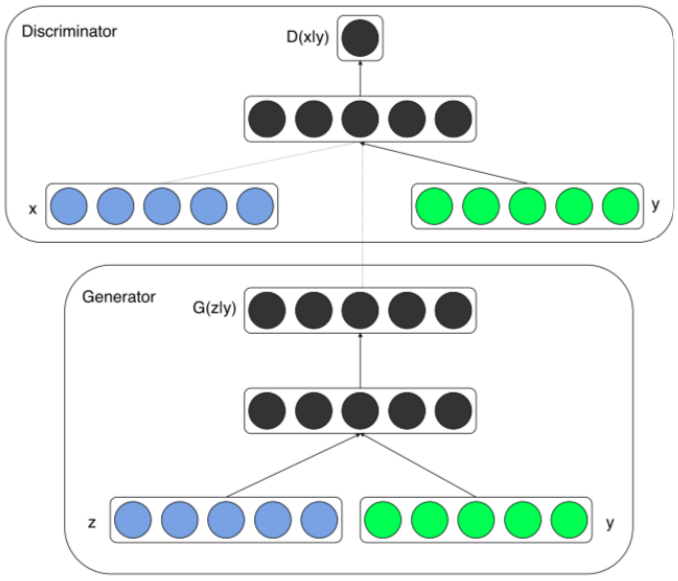

Figure 1: Architecture of CGAN

4.2 How to combine the inputs ?

It is not hard to get the core idea of CGAN. But what really confused me a while is how do we actually combine those inputs z and y together and feed it to the network?

As we are trying to model the joint conditional, the simplest way to do it is to just concatenate both variables. Hence, in $G(z, y)$, we are concatenating z and y before we feed it into the networks. The same procedure is applied to $D(X, y)$. Also we can apply other data process to y before we concatenate it with z like a fully connected layer if needed.

5