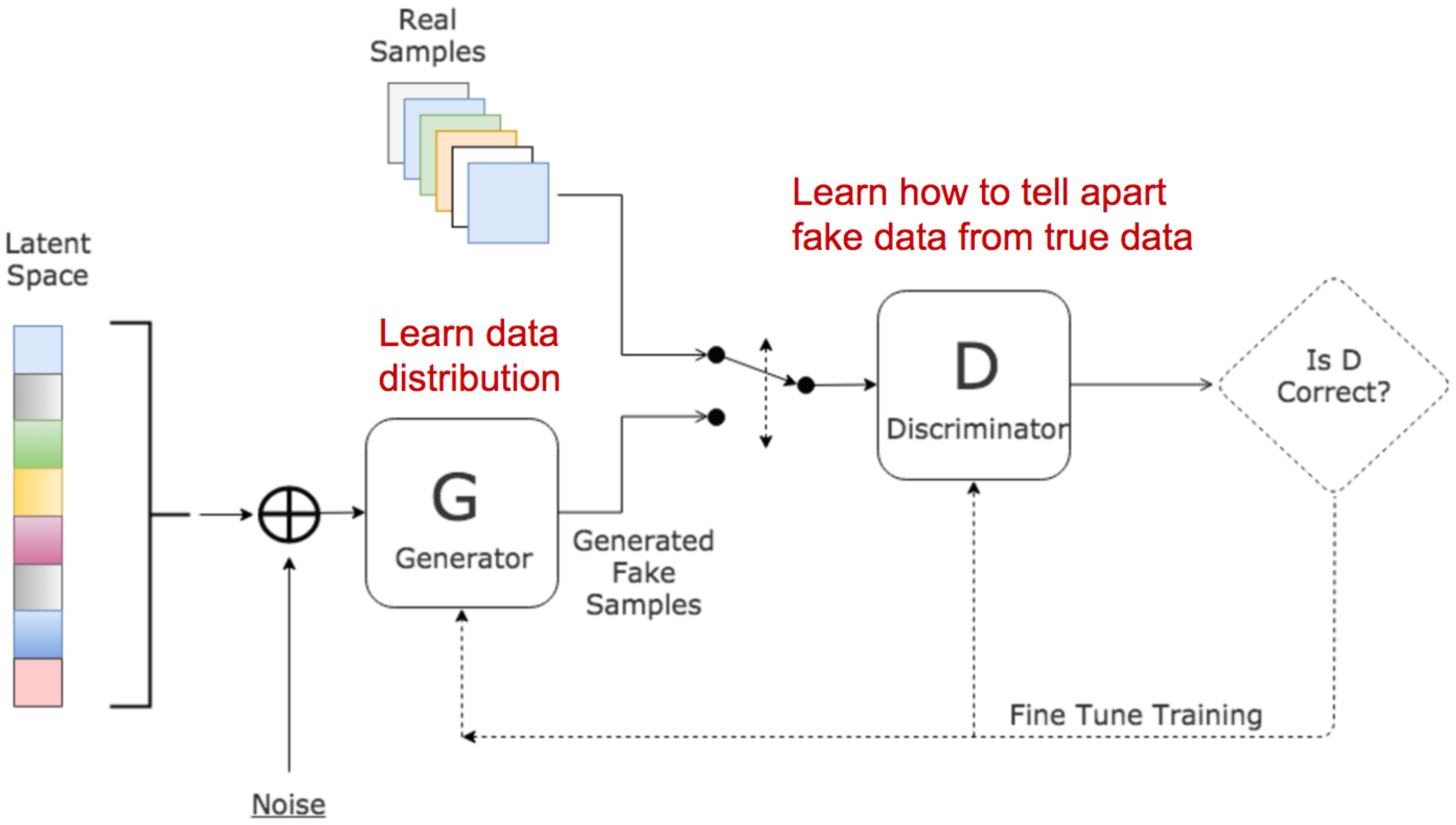
The problem of original GAN

CONTENT

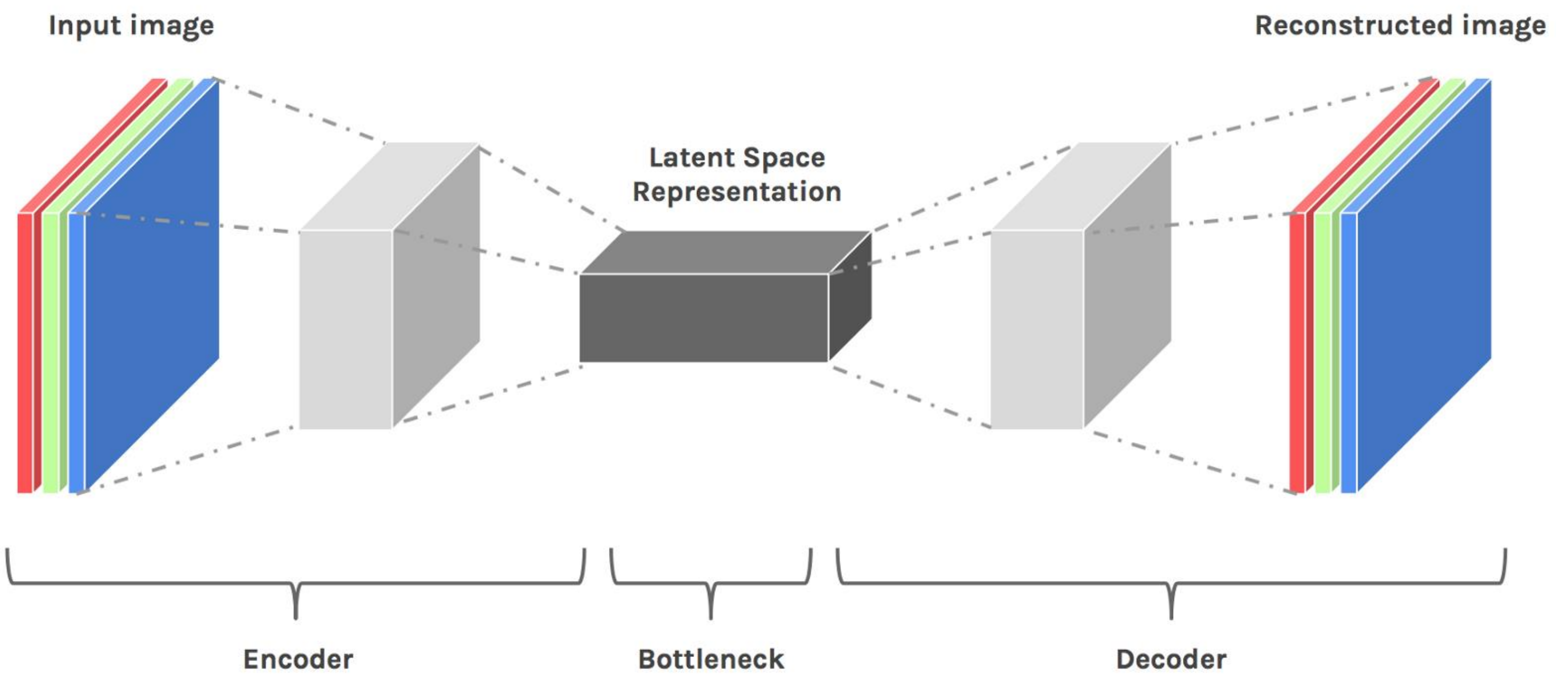
Formula

PART ONE

The Structure of GAN

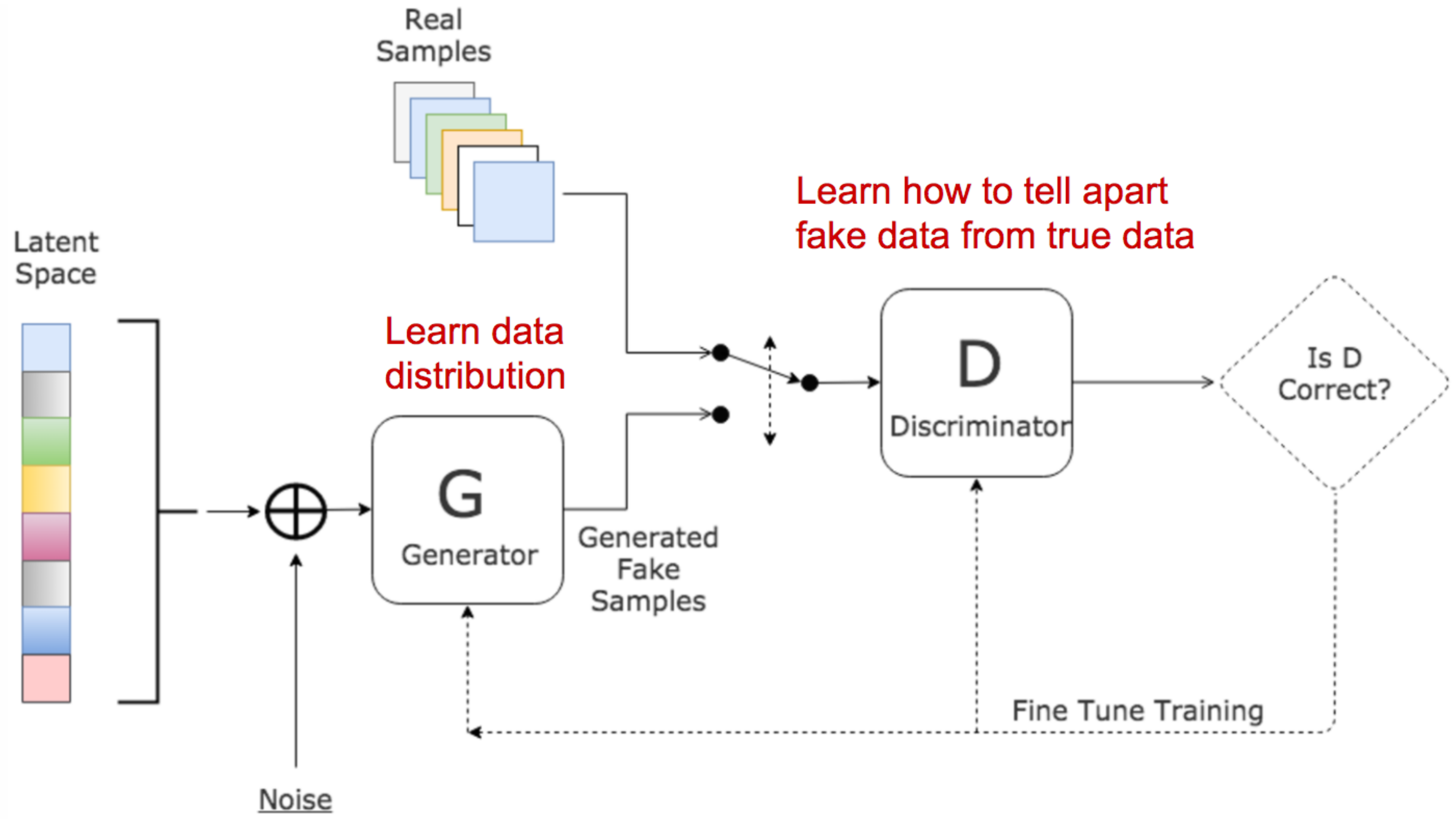


The latent space



- **Encoder brings the data from a high dimension input to a bottleneck layer**
- **Decoder takes this encoded input and converts it back to the original input shape**

The Structure of GAN



The optimizing function of original GAN

To learn the generator' distribution p_g over data x, we define a prior input noise variables $p_z(z)$,

$$L(G) = E_{z\sim p_z}\left[log\left(1 - D\left(G(z)\right)\right)\right] \quad (1)$$

The discriminator's target function is:

$$L(D) = -E_{x\sim P_r}\left[log\left(D(x)\right)\right] - E_{z\sim P_z}\left[log\left(1 - D\left(G(z)\right)\right)\right] \quad (2)$$

where

p_r **is the distribution of real data**

p_z **is the distribution of latent space**

$G(z)$ **is the distribution of generated images, or the mapping from latent space to generator**

$D(x)$ **is the probability distribution that x comes from the data rather than generator, D(x) = 1 → real data D(x) = 0 → fake data**

The optimizing function of original GAN

For simple calculation, we can calculate L(D) and L(G)as :

$$L(G) = E_{z\sim P_z}\left[log\left(1 - D\left(G(z)\right)\right)\right] \tag{3}$$

$$\Downarrow$$

$$L(G) = E_{x\sim P_r}\left[log\left(D(x)\right)\right] + E_{z\sim P_z}\left[log\left(1 - D\left(G(z)\right)\right)\right] \tag{4}$$

$$L(D) = -E_{x\sim P_r}\left[log\left(D(x)\right)\right] - E_{z\sim P_z}\left[log\left(1 - D\left(G(z)\right)\right)\right] \tag{5}$$

D and G plays the following two-player zero sum MinMax game with value function V(G,D):

$$\min_G \max_D V(G,D) = E_{x\sim P_r}\left[log\left(D(x)\right)\right] + E_{z\sim P_z}\left[log\left(1 - D\left(G(z)\right)\right)\right] \tag{6}$$

Discriminator update

Fixed generator, we use stochastic gradient ascent to update discriminator:

$$L(D) = -E_{x\sim P_r}[log(D(x))] - E_{z\sim P_z}\left[log\left(1 - D(G(z))\right)\right] \quad (7)$$

For simple calculation:

$$L(D) = -E_{x\sim P_r}[log(D(x))] - E_{x\sim P_g}[log(1 - D(x))] \quad (8)$$

where

p_g is the distribution of generated images

$$L(D) = -\int_x \{P_r[log(D(x))]\}\mathrm{dx} - \int_x \{P_g[log(1 - D(x))]\}\mathrm{dx} \quad (9)$$

Discriminator update

Fixed generator, we use stochastic gradient ascent to update discriminator:

$$L(D) = -E_{x\sim P_r}[log(D(x))] - E_{z\sim P_z}\left[log\left(1 - D(G(z))\right)\right] \quad (10)$$

For simple calculation:

$$L(D) = -E_{x\sim P_r}[log(D(x))] - E_{x\sim P_g}[log(1 - D(x))] \quad (11)$$

where

p_g is the distribution of generated images

In each iteration, we compute the gradient of the discriminator's loss function:

$$\frac{\partial L(D)}{\partial D} = -\frac{\partial}{\partial D}\int \{p_r log(D(x)) - p_g log(1 - D(x))\}\, dx \quad (12)$$

Discriminator update

$$\frac{\partial L(D)}{\partial D} = -\frac{\partial}{\partial D}\int \{p_r log(D(x)) + p_g log(1 - D(x))\}\, dx \qquad (13)$$

The closed-form solution of the **best discriminator** is:

$$\frac{\partial}{\partial D}[p_r log D(x) + p_g log(1 - D(x))] = -\left[\frac{p_r}{D(x)} - \frac{p_g}{1 - D(x)}\right] = 0 \qquad (14)$$

$$\frac{p_r}{D(x)} = \frac{p_g}{1 - D(x)} \qquad (15)$$

$$D^*(x) = \frac{p_r}{p_r + p_g} \qquad (16)$$

When $p_r = p_g$, the **best discriminator** is:

$$D^*(x) = \frac{1}{2} \qquad (17)$$

Generator Update

Under the best discriminator condition:

$$D^*(x) = \frac{p_r}{p_r + p_g} \tag{18}$$

$$L(G) = E_{x\sim P_r}\left[log\frac{p_r}{p_r + p_g}\right] + E_{z\sim P_z}\left[log\frac{p_g}{p_r + p_g}\right] \tag{19}$$

We use Kullback-Leibler divergence and Jensen-Shannon divergence to denote it:

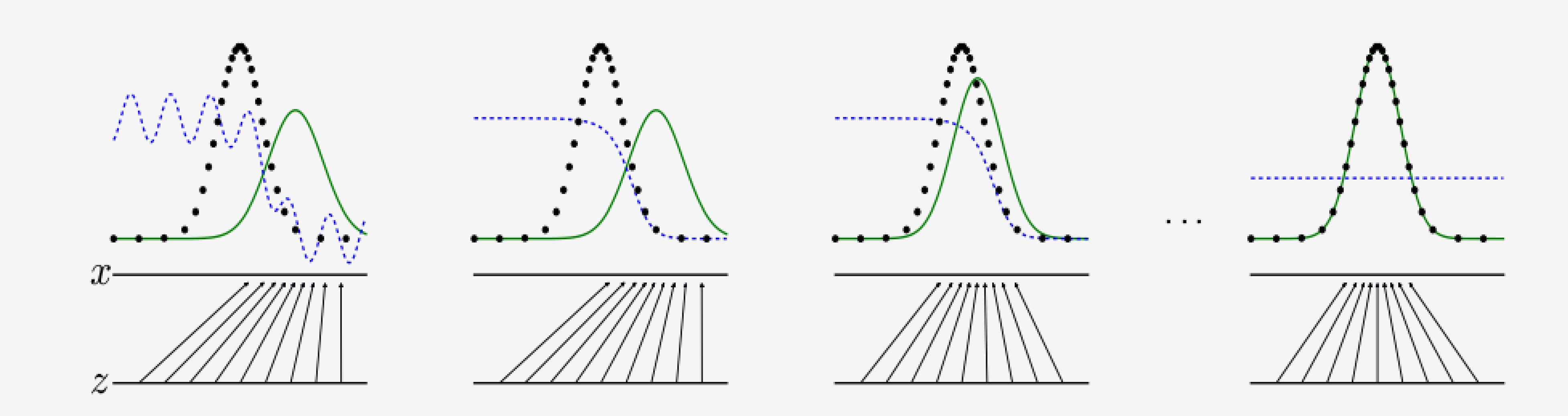
$$KL(P_1 \parallel P_2) = E_{x\sim P_1} log\frac{P_1}{P_2} \tag{20}$$

$$JS(P_1 \parallel P_2) = \frac{1}{2}KL\left(P_1 \parallel \frac{P_1 + P_2}{2}\right) + \frac{1}{2}KL\left(P_2 \parallel \frac{P_1 + P_2}{2}\right) \tag{21}$$

The loss function of generator can be written as follow:

$$L(G) = 2JS(P_r \parallel P_g) - 2log2 \tag{22}$$

Discriminator/Generator Update



D

p_{data}

$p_{generator}$(generated by G(z))

Start from intersection between D and G .Gradient of D has guided G(z) to flow to regions that are more likely to be classified as real data.

The Algorithm of GAN

for number of training iterations **do**

for k steps **do**

Sample minibatch of m noise samples $\{z^{(1)},\ldots,z^{(m)}\}$ from noise prior $p_g(z)$

Sample minibatch of m examples $\{z^{(1)},\ldots,z^{(m)}\}$ from data generating distribution $p_{data}(x)$

Update the discriminator by **ascending** its stochastic gradient:

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

Update Discriminator

end for

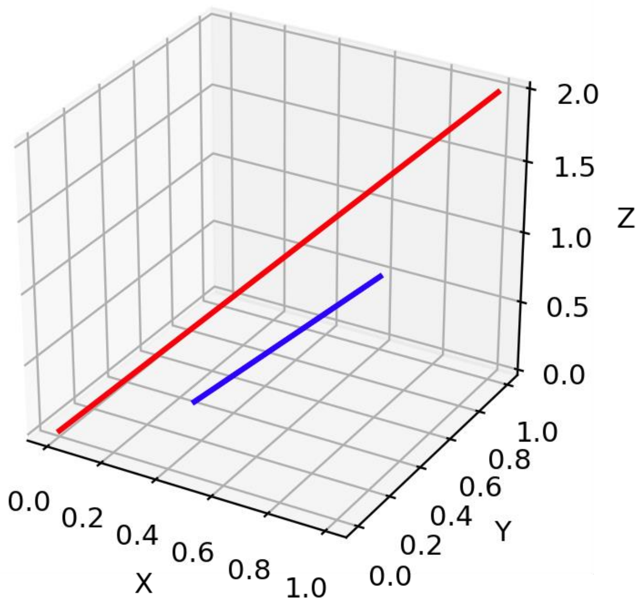
Sample minibatch of m noise sample $\{z^{(1)},\ldots,z^{(m)}\}$ from noise prior $p_g(z)$

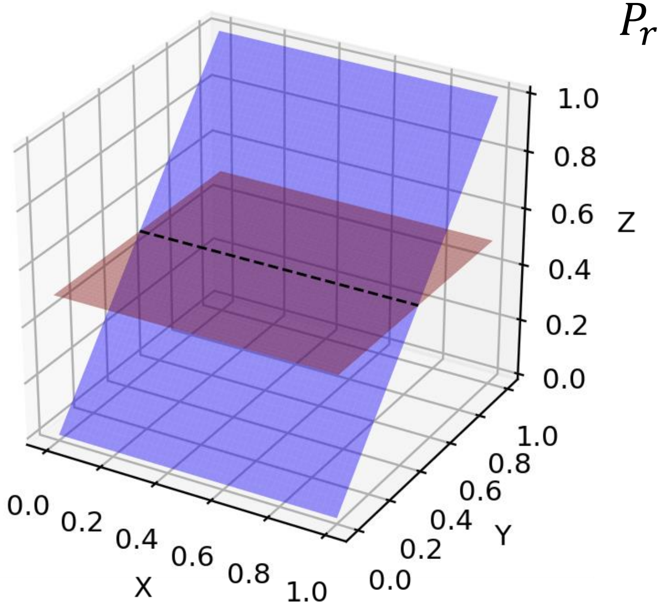
Update the generator by **descending** its stochastic gradient:

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

Update Generator

1 Lower dimension manifold Problem





$$P_r = 0 \; and \; P_g = 0 \Rightarrow JS = No \; contraibution$$

$$P_r = 0 \; and \; P_g = 0 \Rightarrow JS = 0$$

$$P_r \neq 0 \; and \; P_g = 0 \Rightarrow JS = log2$$

$$P_r = 0 \; and \; P_g = 0 \Rightarrow JS = log2$$

JS散度作为优化目标不合理！！！！！

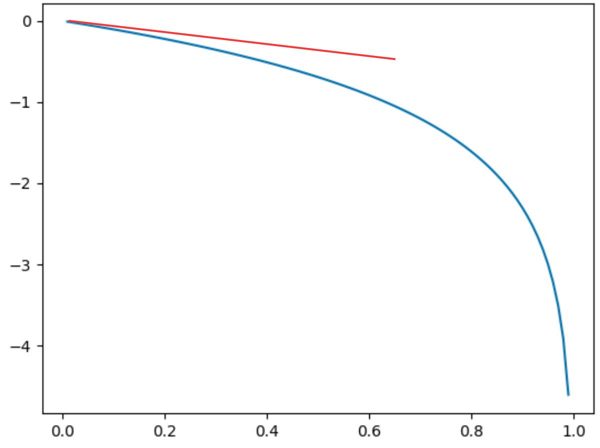
$$L(G) = 2JS(P_r \parallel P_g) - 2log2 \tag{23}$$

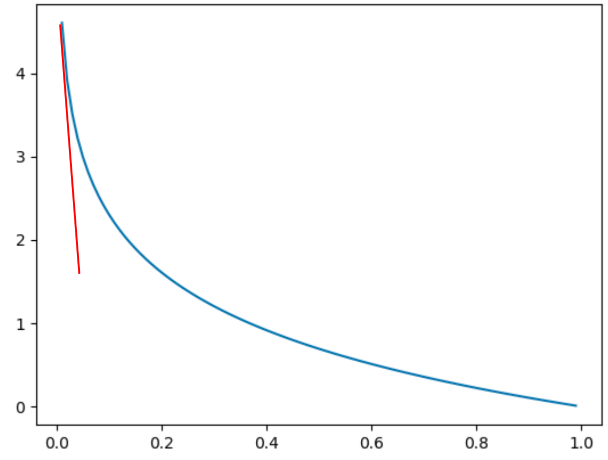
$$JS(P_1 \parallel P_2) = \frac{1}{2}KL\left(P_1 \parallel \frac{P_1 + P_2}{2}\right) + \frac{1}{2}KL\left(P_2 \parallel \frac{P_1 + P_2}{2}\right) \tag{24}$$

$$KL(P_1 \parallel P_2) = E_{x\sim P_1} log\frac{P_1}{P_2} \tag{25}$$

$$L(G) = E_{x\sim P_r} log\frac{P_r}{0.5(P_r + P_g)} + E_{x\sim P_g} log\frac{P_g}{0.5(P_r + P_g)} \tag{26}$$

2 Gradient Saturation Problem





$$L(G) = E_{z\sim P_z}\left[log\left(1 - D\left(G(z)\right)\right)\right] \qquad \text{L}(G)\text{新} = -E_{z\sim P_z}\left[log\left(D\left(G(z)\right)\right)\right]$$

$$0 \leq D(x) \leq 1$$

$$L(G) = 2JS\left(P_r \parallel P_g\right) - 2log2 \tag{30}$$

$$\text{L}(G)\text{新} = KL\left(P_g \parallel P_r\right) - 2JS\left(P_r \parallel P_g\right) + 2log2 + E_{x\sim P_r}[logD^*(x)] \tag{31}$$

2 Gradient Saturation Problem

$$L(G)_{新} = KL(P_g \parallel P_r) - 2JS(P_r \parallel P_g) + 2log2 + E_{x\sim P_r}[logD^*(x)] \quad (32)$$

$$KL(P_g \parallel P_r) = E_{x\sim P_g} log\frac{P_g}{P_r} \quad (33)$$

KL divergence is unbalance!

$$P_g(x) \to 0 \; and \; P_r \to 1, P_g log\frac{P_g}{P_r} \to 0 \quad (34)$$

$$P_g(x) \to 1 \; and \; P_r \to 0, P_g log\frac{P_g}{P_r} \to +\infty \quad (35)$$

生成器没能生成真实的样本,惩罚微小

生成器生成了不真实的样本,惩罚巨大

3 Model collapse Problem

$$P_g(x) \to 0 \ and \ P_r \to 1, P_g log\frac{P_g}{P_r} \to 0 \qquad (36)$$

$$P_g(x) \to 1 \ and \ P_r \to 0, P_g log\frac{P_g}{P_r} \to +\infty \qquad (37)$$

生成器没能生成真实的样本,惩罚微小

生成器生成了不真实的样本,惩罚巨大

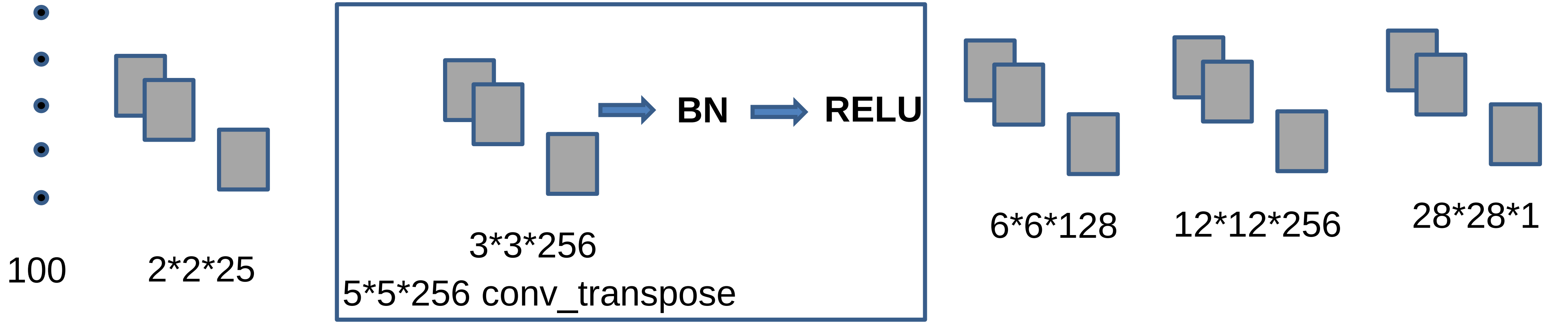
这一放一打之下，生成器宁可多生成一些重复但是很“安全”的样本，也不愿意去生成多样性的样本，因为那样一不小心就会产生第二种错误，得不偿失。这种现象就是大家常说的collapse mode。

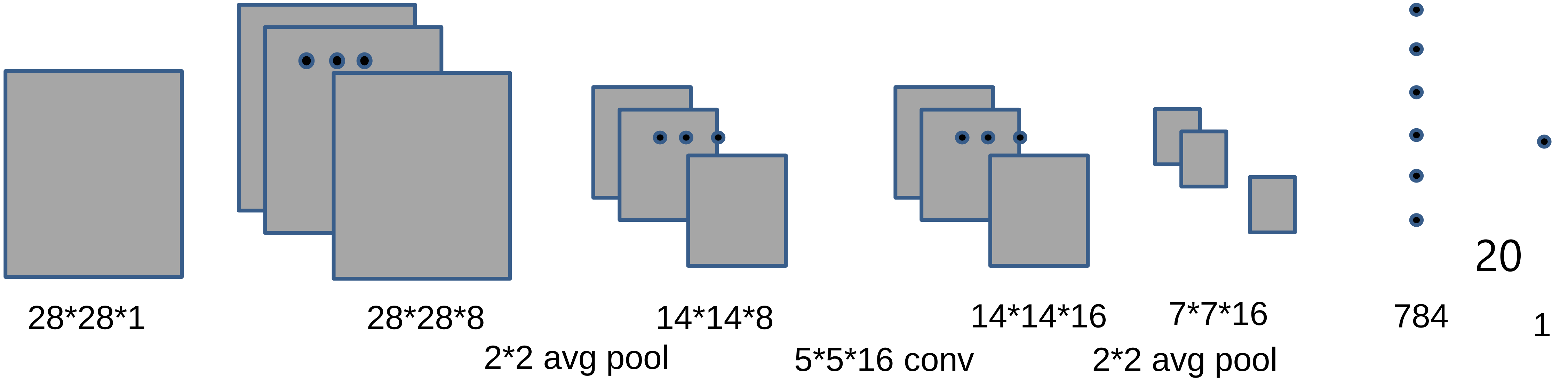
Experiment
PART TWO

PART TWO Experiment

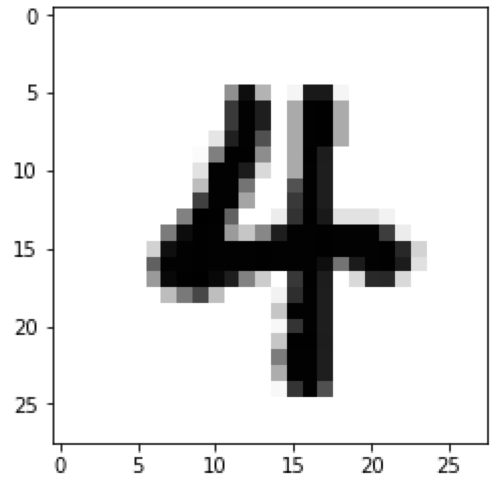
1 My GAN' structure

MINST
dataset

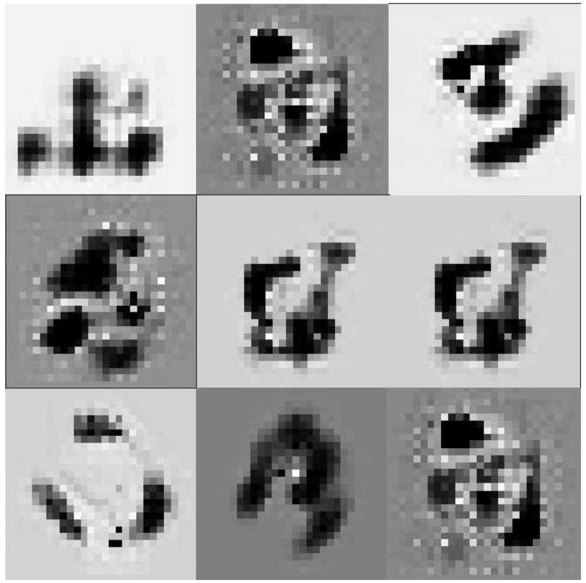




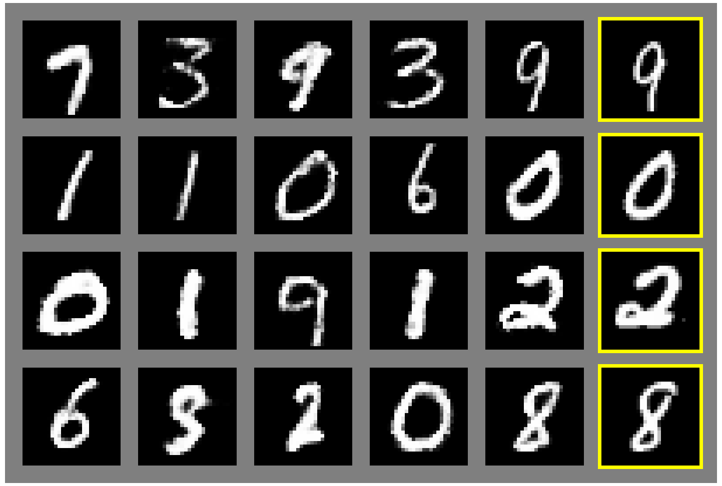
2 The Experiment result



Real Data



My Generated image



Original paper's image

设计此网络的目的是为了验证由于原始GAN模型出现的生成模型训练困难导致出现的生成图片不真实的情况，实际效果如上二图所示，但是在GAN原始论文中给出的模型训练效果比较真实，为了探究产生此问题的原因，下周，我会复现一下GAN原始论文中的代码。

Plan

Features Business Explore Marketplace Pricing

Search GitHub

Sign in or Sign up

Create your own GitHub profile

Sign up for your own profile on GitHub, the best place to host code, manage projects, and build software alongside 28 million developers.

Sign up

Dismiss

Ian Goodfellow

goodfeli

Staff Research Scientist at Google Brain

Block or report user

Google

Mountain View

Sign in to view email

http://www.iangoodfellow.com

Organizations

Overview Repositories 18 Stars 5 Followers 6.6k Following 4

Popular repositories

adversarial
Code and hyperparameters for the paper "Generative Adversarial Networks"
Python ★ 1.6k 612

dlbook_exercises
Exercises for the Deep Learning textbook at www.deeplearningbook.org
TeX ★ 629 173

theano_exercises
Exercises for my tutorials on Theano
Python ★ 595 309

dlbook_notation
LaTeX files for the Deep Learning book notation
TeX ★ 582 120

galatea
Ian Goodfellow's private research codebase
Python ★ 60 21

forgetting
Repository of code for the experiments for the ICLR submission "An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Networks"
Python ★ 25 13

355 contributions in the last year

PART THREE Plan

goodfeli Merge pull request #9 from jimmyahacker/correct-typo ··· Latest commit 1720041 on Jun 6

tfd_pretrain	new	4 years ago
.gitignore	Initial commit	4 years ago
LICENSE	Initial commit	4 years ago
README.md	Copy the code and hyperparameters from galatea	4 years ago
__init__.py	avoid underflowing the division	4 years ago
cifar10_convolutional.yaml	Copy the code and hyperparameters from galatea	4 years ago
cifar10_fully_connected.yaml	Copy the code and hyperparameters from galatea	4 years ago
deconv.py	Copy the code and hyperparameters from galatea	4 years ago
ll.py	avoid underflowing the division	4 years ago
ll_mnist.py	disable mnist likelihood	4 years ago
mnist.yaml	sped up mnist yaml file by monitoring few channels	4 years ago
parzen_ll.py	* Fix a typo	a month ago
sgd.py	fix use of data	4 years ago
sgd_alt.py	Copy the code and hyperparameters from galatea	4 years ago
show_gen_weights.py	Copy the code and hyperparameters from galatea	4 years ago
show_inpaint_samples.py	Copy the code and hyperparameters from galatea	4 years ago
show_samples.py	Copy the code and hyperparameters from galatea	4 years ago
show_samples_cifar_conv_paper.py	Copy the code and hyperparameters from galatea	4 years ago
show_samples_cifar_full_paper.py	Copy the code and hyperparameters from galatea	4 years ago
show_samples_inpaint.py	Copy the code and hyperparameters from galatea	4 years ago
show_samples_mnist_paper.py	Copy the code and hyperparameters from galatea	4 years ago
show_samples_tfd.py	Copy the code and hyperparameters from galatea	4 years ago
show_samples_tfd_paper.py	Copy the code and hyperparameters from galatea	4 years ago
test_deconv.py	Copy the code and hyperparameters from galatea	4 years ago

README.md

42 commits | 1 branch | 0 releases | 9 contributors | BSD-3-Clause

Branch: master | New pull request | Find file | Clone or download

martinarjovsky committed on Aug 24, 2017 Update README.md — Latest commit `f81eafd` on Aug 24, 2017

imgs	add readme	2 years ago
models	Fixing wrong variable name	a year ago
LICENSE.md	add license	2 years ago
README.md	Update README.md	11 months ago
main.py	de-normalize images for displaying	a year ago
requirements.txt	push code	2 years ago

README.md

Wasserstein GAN

Code accompanying the paper "Wasserstein GAN"

A few notes

- The first time running on the LSUN dataset it can take a long time (up to an hour) to create the dataloader. After the first run a small cache file will be created and the process should take a matter of seconds. The cache is a list of indices in the lmdb database (of LSUN)
- The only addition to the code (that we forgot, and will add, on the paper) are the lines 163-166 of main.py. These lines act only on the first 25 generator iterations or very sporadically (once every 500 generator iterations). In such a case, they set the number of iterations on the critic to 100 instead of the default 5. This helps to start with the critic at optimum even in the first iterations. There shouldn't be a major difference in performance, but it can help, especially when visualizing learning curves (since otherwise you'd see the loss going up until the critic is properly trained). This is also why the first 25 iterations take significantly longer than the rest of the training as well.
- If your learning curve suddenly takes a big drop take a look at this. It's a problem when the critic fails to be close to optimum, and hence its error stops being a good Wasserstein estimate. Known causes are high learning rates and momentum, and anything that helps the critic get back on track is likely to help with the issue.