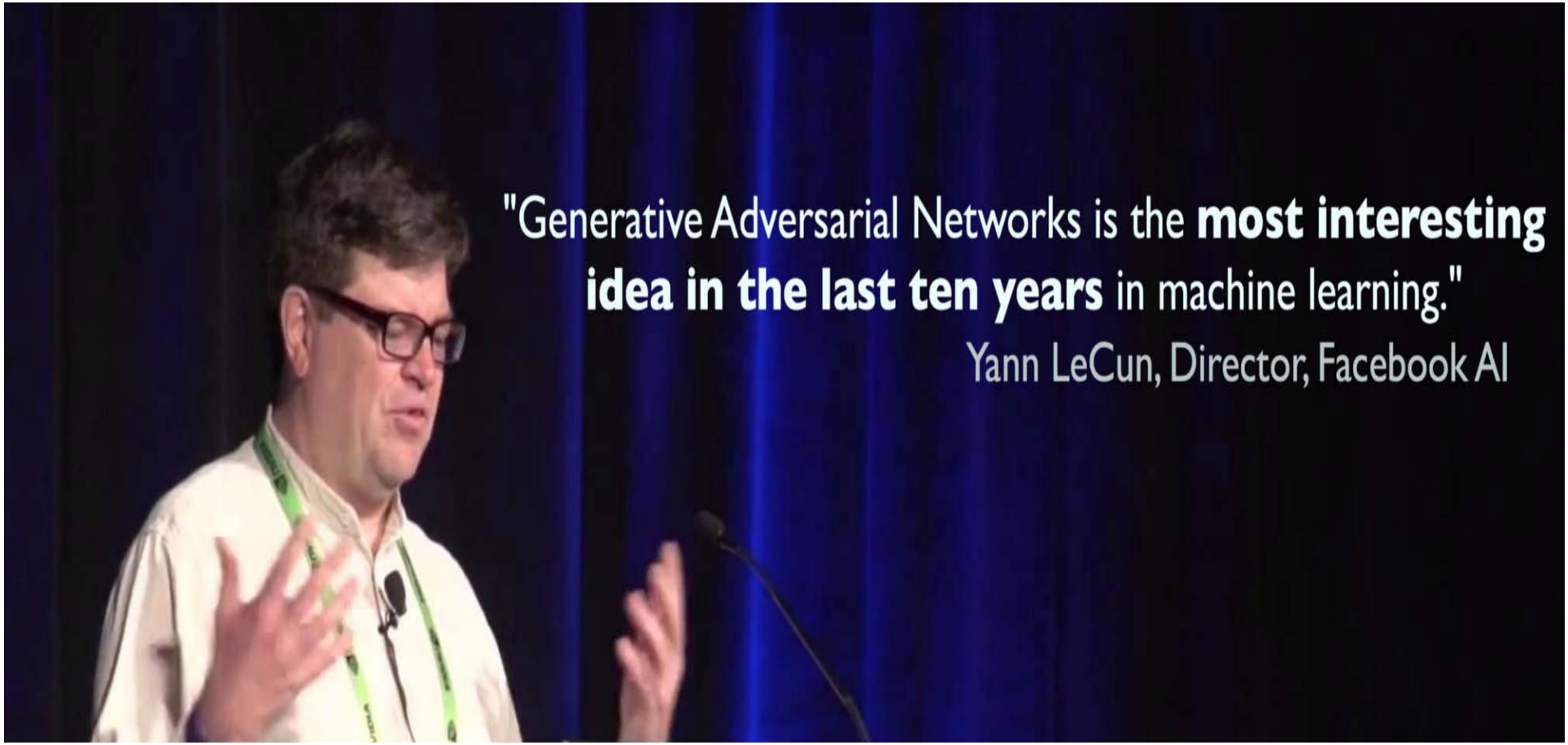
Lecture 3
GANs and Their Applications in Image Generation

Lin ZHANG, PhD
School of Software Engineering
Tongji University
Fall 2017


"Generative Adversarial Networks is the most interesting idea in the last ten years in machine learning."
Yann LeCun, Director, Facebook AI

Outline

- Introduction
- Theoretical Part
- Application Part
- Existing Implementations Using Tensorflow

Introduction

- Deep CNNs mentioned in our last lectures actually are kinds of supervised learning tools
  - Supervised learning can solve many problems
  - But they need labeled data which is quite expensive to obtain
- Supervised deep learning is quite powerful; that is because it can learn a good **representation** of data

Q: Is there any other ways to learn representations of data?
A: Unsupervised learning. Using unsupervised learning, good intermediate representations could be learned from unlimited amount of unlabeled data, which can then be used in a variety of supervised learning tasks such as image classification

Introduction

The rapid progress of AI in the last few years are largely the result of advances in deep learning and neural nets, combined with the availability of large datasets and fast GPUs. We now have systems that can recognize images with an accuracy that rivals that of humans. This will lead to revolutions in several domains such as autonomous transportation and medical image understanding. But all of these systems currently use supervised learning in which the machine is trained with inputs labeled by humans. The challenge of the next several years is to let machines learn from raw, unlabeled data, such as video or text. This is known as unsupervised learning. AI systems today do not possess "common sense", which humans and animals acquire by observing the world, acting in it, and understanding the physical constraints of it. Some of us see unsupervised learning as the key towards machines with common sense.

--Yann LeCun 2016 at CMU

Introduction

- GANs, short for Generative Adversarial Networks
  - It is an unsupervised learning theory
  - They can learn to create data that is similar to data that we give them
  - The intuition behind this is that if we can get a model to write high-quality news articles for example, then it must have also learned a lot about news articles in general. Or in other words, the model should also have a good internal **representation** of news articles. We can then hopefully use this representation to help us with other related tasks, such as classifying news articles by topic
- Based on GANs, a lot interesting works can be done

Outline

- Introduction
- Theoretical Part
  - Generative Adversarial Networks (GAN)
  - Deep Convolution GAN (DCGAN)
  - Wasserstein GAN (WGAN)
  - Conditional GANs (cGANs)
- Application Part
- Existing Implementations Using Tensorflow

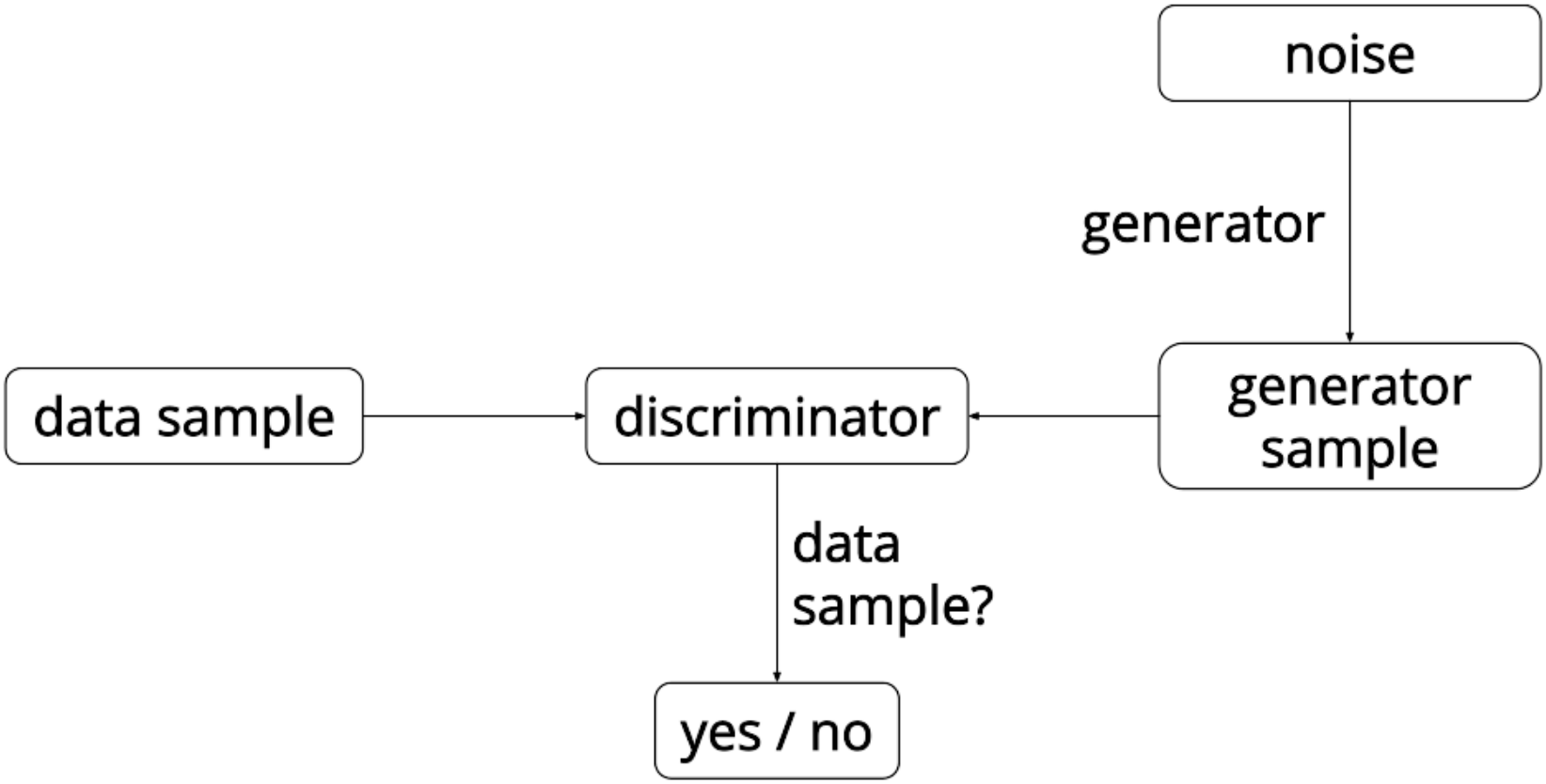
Generative Adversarial Networks[1]

- The main idea behind a GAN is to have two competing neural network models
  - One takes noise as input and generates samples (and so is called the generator)
  - The other model (called the discriminator) receives samples from both the generator and the training data, and has to be able to distinguish between the two sources
  - These two networks play a continuous game, where the generator is learning to produce more and more realistic samples, and the discriminator is learning to get better and better at distinguishing generated data from real data
  - These two networks are trained simultaneously, and the hope is that the competition will drive the generated samples to be indistinguishable from real data

[1] I. J. Goodfellow et al., Generative adversarial nets, NIPS, 2014

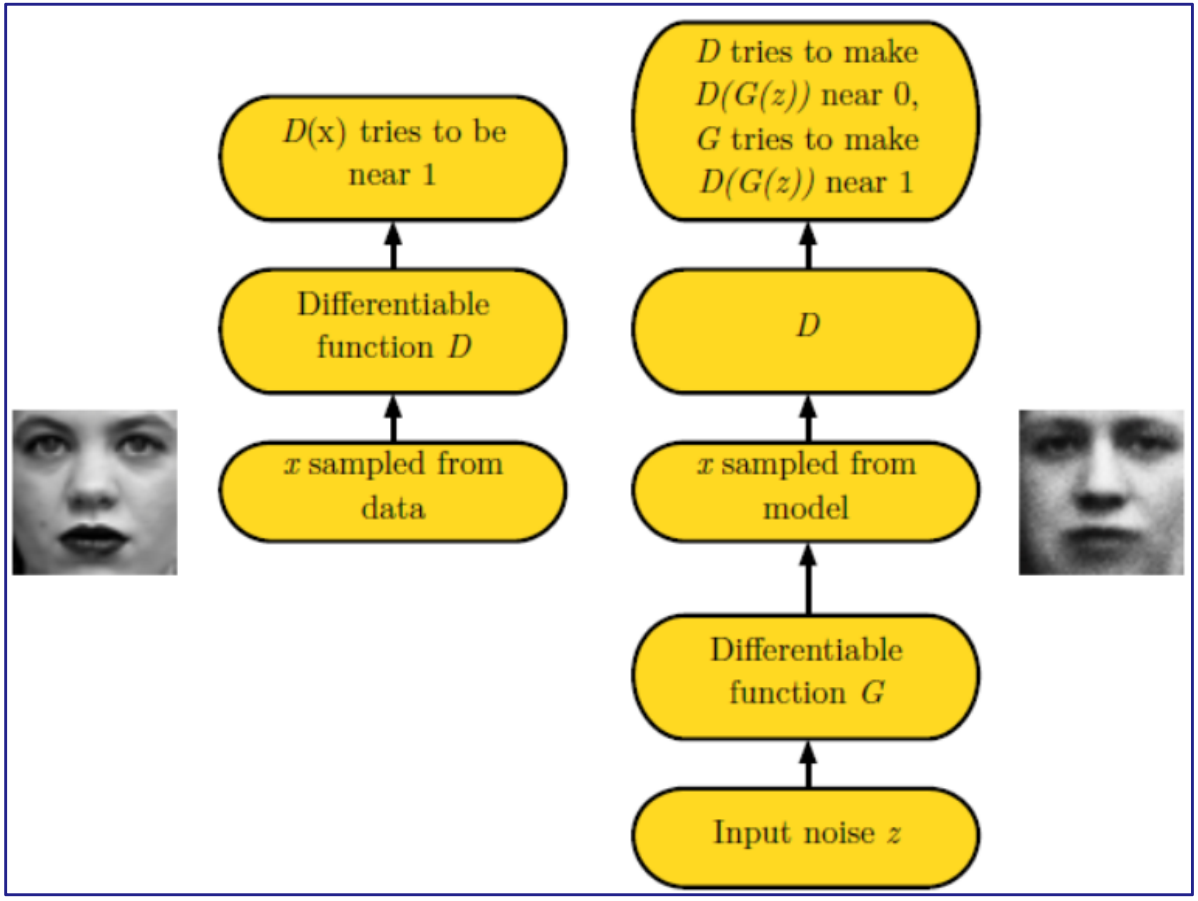
Generative Adversarial Networks

- A coarse architecture of GAN



Generative Adversarial Networks

- A more formal description



Generative Adversarial Networks

- A more formal description
  - G is the generator, mapping the input noise variable $\mathbf{z}$ to data space as $G(\mathbf{z};\theta_g)$
  - D is the discriminator $D(\mathbf{x};\theta_d)$, outputting a single scalar; $D(\mathbf{x})$ represents the probability that $\mathbf{x}$ came from the data
  - We train D to maximize the probability of assigning the correct label to both training samples and samples from G
  - We simultaneously train G to minimize $\log\big(1-D(G(\mathbf{z}))\big)$

D and G play the two-player minimax game with the value function $V(G, D)$,

$$\min_G \max_D \Big\{V(D,G) = E_{\mathbf{x}\sim p_{data}(\mathbf{x})}\big[\log D(\mathbf{x})\big] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\big[\log(1-D(G(\mathbf{z})))\big]\Big\}$$

Generative Adversarial Networks

- Algorithm for training

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

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

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Generative Adversarial Networks

- Some notes
  - G and D are trained alternatively
  - When D is trained, parameters related to G are fixed
  - When G is trained, parameters related to D are fixed

Generative Adversarial Networks—Example

Problem: We want to train a $G(z)$, mapping a given random number z to a number conforms to $N(\mu, \sigma^2)$

Solution

Prepare real data: generate a set $\{x_i\}_{i=1}^{N}$, and each x_i is drawn from $N(\mu, \sigma^2)$

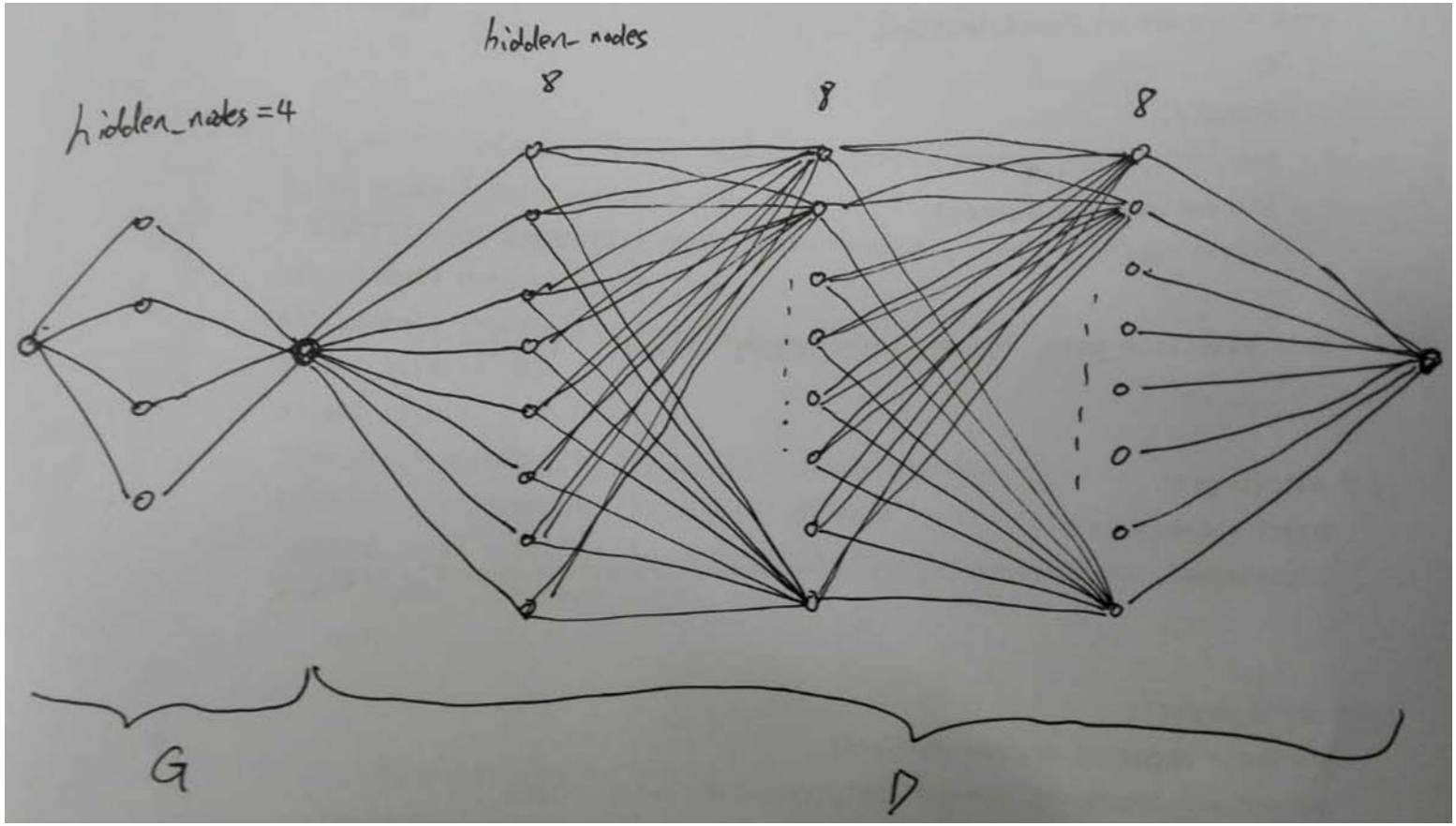
Train the GAN, and finally we can get a G, which can map any given z to $G(z)$, satisfying the requirement

This example and the code for Ubuntu+TensorFlow+Python can be found at

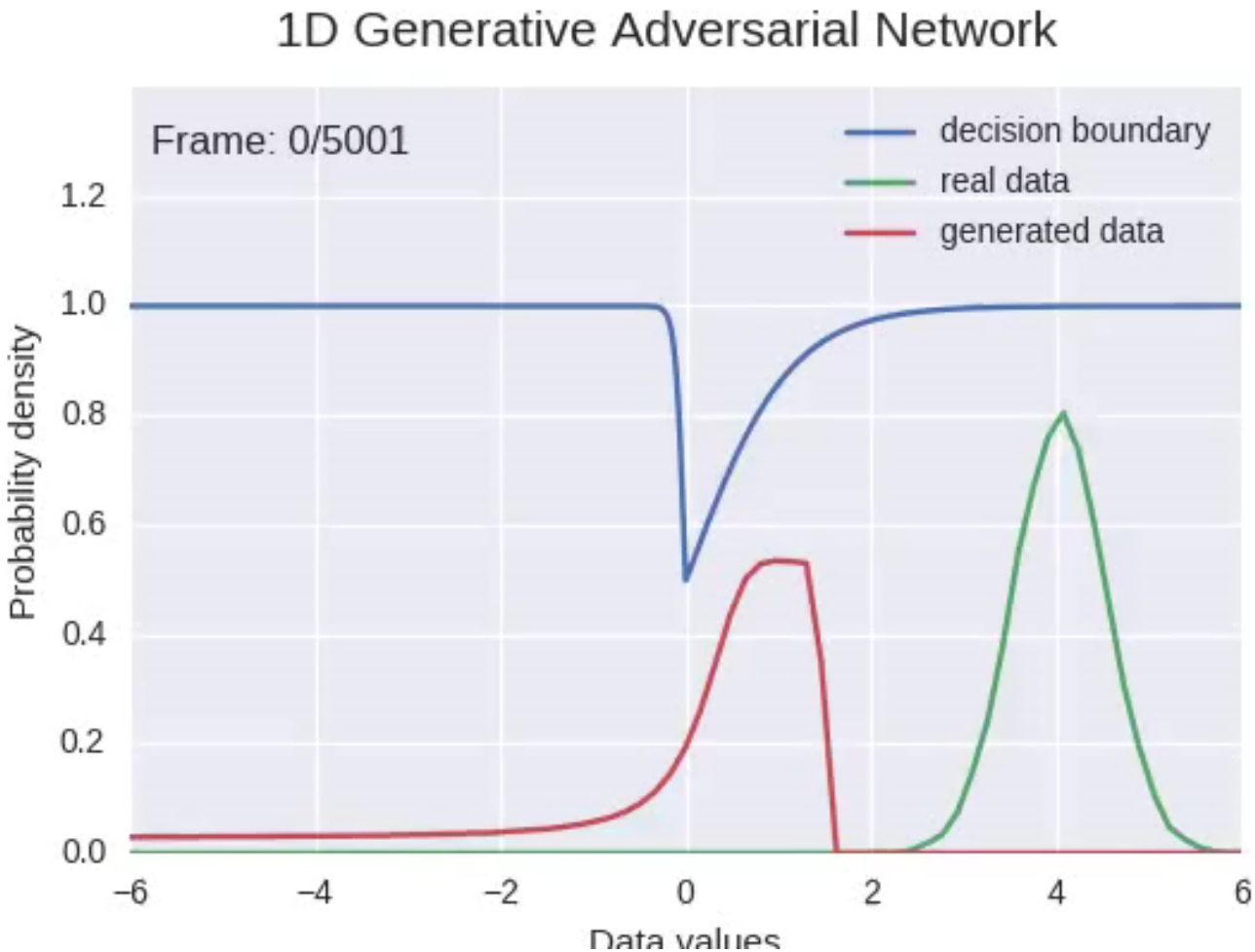
http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/

Generative Adversarial Networks—Example

Network structures of G and D of this example (I highly recommend you read through the codes)



Generative Adversarial Networks—Example



Problems of Goodfellow's GAN

- There are some drawbacks of the original Goodfellow's GAN
  - In general, they are notoriously difficult to train (refer to the gif in previous slide)
  - Controlling the image diversity of the generated samples is difficult
  - In many cases, the generated images suffer from being noisy and incomprehensible
  - Balancing the convergence of the discriminator and of the generator is a challenge
  - Easily suffer from modal collapse, a failure mode in which just one image is learned

Try to solve these problems

Researchers proposed different GAN variants

Outline

- Introduction
- Theoretical Part
  - Generative Adversarial Networks (GAN)
  - Deep Convolutional GAN (DCGAN)
  - Wasserstein GAN (WGAN)
  - Conditional GANs (cGANs)
- Application Part
- Existing Implementations Using Tensorflow

Deep Convolutional GAN (DCGAN)[1]

- DCGAN is the first successful work combining GANs with deep convolutional neural networks
- It can be trained to generate higher resolution and sharper images than Goodfellow's GAN
- DCGAN can be more easily and robustly trained than the original GAN
- The authors demonstrate that the intermediate representations learned by DCGAN can be successfully used for supervised tasks

[1] A. Radford et al., Unsupervised representation learning with deep convolutional generative adversarial networks, ICLR, 2016

Deep Convolutional GAN (DCGAN)

- Architecture guidelines for DCGAN
  - Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator)
  - Use batchnorm in both the generator and the discriminator
  - Remove fully connected hidden layers for deeper architectures
  - Use ReLU activation in generator for all layers except for the output, which uses Tanh
  - Use LeakyReLU activation in the discriminator for all layers

Deep Convolutional GAN (DCGAN)

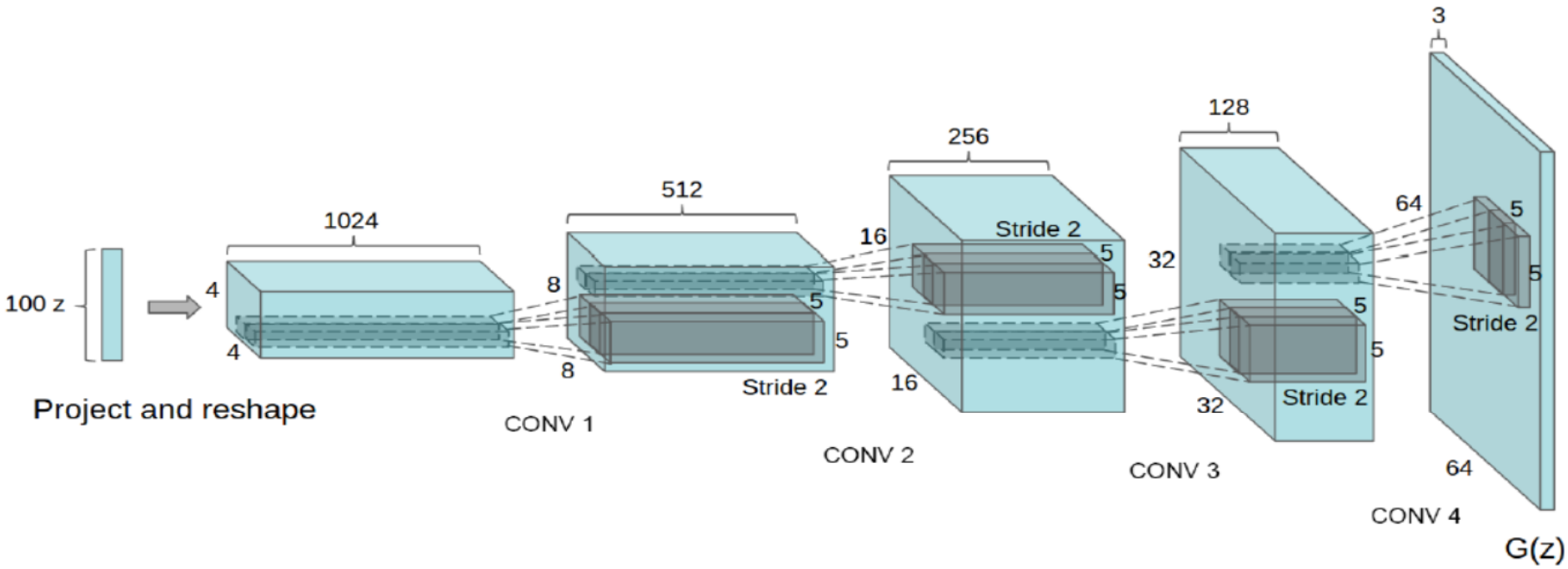


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

Face images generated by DCGAN

Outline

- Introduction
- Theoretical Part
  - Generative Adversarial Networks (GAN)
  - Deep Convolutional GAN (DCGAN)
  - Wassertein GAN (WGAN)
  - Conditional GANs (cGANs)
- Application Part
- Existing Implementations Using Tensorflow

Problems of Goodfellow's GAN[1]

- Goodfellow's GAN is difficult to train
  - The training process is quite unstable
  - Generated samples have a low diversity
  - There is no a general loss function to indicate the training progress, i.e., there is no indicator to tell you when to stop training

What is the root reason?

[1] M. Arjovsky et al., Towards principled methods for training generative adversarial networks, ICLR, 2017

Problems of Goodfellow's GAN—Prerequisite

- Kullback–Leibler divergence (KL-divergence)
  - It is also called relative entropy
  - It is used to measure the distance between two distributions
  - KL divergence is asymmetric

$$KL(P_1 \| P_2) = E_{x\sim P_1}\left[\log \frac{P_1}{P_2}\right]$$

In discrete case it is, $KL(P_1 \| P_2) = \sum P_1(x) \log \frac{P_1(x)}{P_2(x)}$

In continuous case it is, $KL(P_1 \| P_2) = \int p_1(x) \log \frac{p_1(x)}{p_2(x)} dx$

  - It has a unique minimum at $P_1(x) = P_2(x)$

Note that, usually we define: $0 \log \frac{0}{0} = 0$, $0 \log \frac{0}{q} = 0$, $p \log \frac{p}{0} = +\infty$

Problems of Goodfellow's GAN—Prerequisite

- Jensen–Shannon divergence (JS-divergence)
  - It is defined based on KL-divergence
  - Different from KL-divergence, it is symmetric

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

Problems of Goodfellow's GAN

When training the discriminator, we need to minimize

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

where P_r is the distribution of the real data and P_g is the distribution of the generated data

For a specified sample x, the loss is

$$-P_r(x)\log(D(x)) - P_g(x)\log(1-D(x))$$

Thus, the optimal discriminator D^* should be of the form,

$$D^*(x) = \frac{P_r(x)}{P_r(x)+P_g(x)} \qquad (1)$$

When training the generator, we need to minimize

$$L(g_\theta) = E_{x\sim P_r}\left[\log D(x)\right] + E_{x\sim P_g(x)}\left[\log(1-D(x))\right] \qquad (2)$$

When D is optimal, (2) becomes

Problems of Goodfellow's GAN

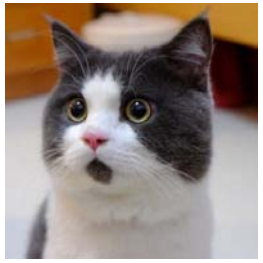
$$L(g_\theta) = E_{x\sim P_r}\left[\log\frac{P_r(x)}{\frac{1}{2}\left(P_r(x)+P_g(x)\right)}\right] + E_{x\sim P_g(x)}\left[\log\frac{P_g(x)}{\frac{1}{2}\left(P_r(x)+P_g(x)\right)}\right] - 2\log 2 \qquad (3)$$

$$= KL\left(P_r \,\|\, \frac{P_r+P_g}{2}\right) + KL\left(P_g \,\|\, \frac{P_r+P_g}{2}\right) - 2\log 2$$

$$= 2JS\left(P_r \,\|\, P_g\right) - 2\log 2$$

Thus, when training the generator, the JS divergence between P_r and P_g is being minimized. It seems reasonable.

However, actually it is nearly impossible to minimize $JS\left(P_r \,\|\, P_g\right)$



Surprised? Why?

Problems of Goodfellow's GAN

Lemma 3. *Let $\mathcal{M}$ and $\mathcal{P}$ be two regular submanifolds of $\mathbb{R}^d$ that don't perfectly align and don't have full dimension. Let $\mathcal{L} = \mathcal{M} \cap \mathcal{P}$. If $\mathcal{M}$ and $\mathcal{P}$ don't have boundary, then $\mathcal{L}$ is also a manifold, and has strictly lower dimension than both the one of $\mathcal{M}$ and the one of $\mathcal{P}$. If they have boundary, $\mathcal{L}$ is a union of at most 4 strictly lower dimensional manifolds. In both cases, $\mathcal{L}$ has measure 0 in both $\mathcal{M}$ and $\mathcal{P}$.*

In our case, P_r and P_g are two regular submanifolds that don't perfect align and don't have full dimensions, thus the intersection of P_r and P_g has the measure 0

Thus,

$$JS\left(P_r \,\|\, P_g\right) = \frac{1}{2}\int P_r(x)\log\frac{P_r(x)}{\frac{1}{2}\left(P_r(x)+P_g(x)\right)}dx + \frac{1}{2}\int P_g(x)\log\frac{P_g(x)}{\frac{1}{2}\left(P_r(x)+P_g(x)\right)}dx$$

$$= \frac{1}{2}\int P_r(x)\log\frac{P_r(x)}{\frac{1}{2}P_r(x)}dx + \frac{1}{2}\int P_g(x)\log\frac{P_g(x)}{\frac{1}{2}P_g(x)}dx$$

$$= \frac{1}{2}\log 2 + \frac{1}{2}\log 2 = \log 2$$

Problems of Goodfellow's GAN

- Finally, we have the following observations
  - When the discriminator is optimal, optimizing the generator equals minimizing $JS\left(P_r \| P_g\right)$; however, since the overlap between P_r and P_g is negligible, $JS\left(P_r \| P_g\right)$ is actually a constant; that means, for SGD based optimization methods, the gradient is zero!
  - When the discriminator is far from optimal, the gradient information obtained when training the generator will be less indicative, i.e., they are not quite useful to train a strong generator
  - Only when the discriminator is either not too bad or not too good, the training process can be continued; that is the root cause why the training process for GAN is difficult to control

Any solutions?

Wasserstein GAN (WGAN)—Pre-requisite

- Earth-mover distance (EMD)

Assume that signature P has m clusters with $P = \{(p_1, w_{p1}), (p_2, w_{p2}), \ldots, (p_m, w_{pm})\}$, where p_i is the cluster representative and w_{pi} is the weight of the cluster. Similarly another signature $Q = \{(q_1, w_{q1}), (q_2, w_{q2}), \ldots, (q_n, w_{qn})\}$ has n clusters. Let $D = [d_{i,j}]$ be the ground distance between clusters p_i and q_j.

We want to find a flow $F = [f_{i,j}]$, with $f_{i,j}$ the flow between p_i and q_j, that minimizes the overall cost.

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j} d_{i,j}$$

subjected to the constraints:

$$f_{i,j} \geq 0, 1 \leq i \leq m, 1 \leq j \leq n$$

$$\sum_{j=1}^{n} f_{i,j} \leq w_{pi}, 1 \leq i \leq m$$

$$\sum_{i=1}^{m} f_{i,j} \leq w_{qj}, 1 \leq j \leq n$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j} = \min \left\{ \sum_{i=1}^{m} w_{pi}, \quad \sum_{j=1}^{n} w_{qj} \right\}$$

The optimal flow F is found by solving this linear optimization problem. The earth mover's distance is defined as the work normalized by the total flow:

$$EMD(P, Q) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j} d_{i,j}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j}}$$

Wasserstein GAN (WGAN)—Pre-requisite

- Earth-mover distance (EMD)
  - Note that: in general, P and Q in EMD can have different integrals (The total quantity of supplied goods and the total volume of the warehouses can be different)
  - When the two distributions have the same integral, as in normalized histograms or probability density functions, the EMD is equivalent to the **Wasserstein distance** between the two distributions

Wasserstein GAN (WGAN)—Pre-requisite

- Wasserstein distance

Suppose $P_1(x)$ and $P_2(y)$ are two distributions

$\gamma \in \Pi(P_1, P_2)$, which is the set of all distributions whose marginals are $P_1(x)$ and $P_2(y)$ respectively

That is, $\sum_x \gamma(x,y) = P_2(y), \sum_y \gamma(x,y) = P_1(x)$

The Wasserstein distance (equal to EMD) is defined as,

$$W(P_1, P_2) = \inf_{\gamma \in \Pi} \left(\sum_{x,y} \|x - y\| \gamma(x,y) \right) = \inf_{\gamma \in \Pi} \left[E_{(x,y)\sim\gamma} \left(\|x - y\| \right) \right]$$

Wasserstein GAN (WGAN)—Pre-requisite

- Wasserstein distance
  - It is better than KL-divergence or JS-divergence in that even though the two distributions do not overlap, it can still measure their difference
  - Thus, it can used to substitute the JS divergence used in Goodfellow's GAN

Wasserstein GAN (WGAN)—Pre-requisite

- Wasserstein distance

Example 1 (Learning parallel lines). Let $Z \sim U[0,1]$ the uniform distribution on the unit interval. Let $\mathbb{P}_0$ be the distribution of $(0, Z) \in \mathbb{R}^2$ (a 0 on the x-axis and the random variable Z on the y-axis), uniform on a straight vertical line passing through the origin. Now let $g_\theta(z) = (\theta, z)$ with θ a single real parameter. It is easy to see that in this case,

- $W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|$,
- $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0\,, \\ 0 & \text{if } \theta = 0\,, \end{cases}$
- $KL(\mathbb{P}_\theta \| \mathbb{P}_0) = KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0\,, \\ 0 & \text{if } \theta = 0\,, \end{cases}$

Wasserstein GAN (WGAN)—Pre-requisite

- Wasserstein distance

Based on Kantorovich-Rubinstein duality,

$$W(P_r, P_\theta) = \sup_{\|f\|_L \le 1} \left(E_{x \sim P_r}[f(x)] - E_{x \sim P_\theta}[f(x)] \right)$$

where the supremum is over all the 1-Lipschitz functions $f : \chi \to R$.

Therefore, if we have a parameterized family of functions $\{ f_w \}_{w \in W}$ that are all K-Lipschitz for some K, the following expression,

$$\max_{w \in W} \left(E_{x \sim P_r}[f_w(x)] - E_{z \sim p(z)}[f_w(g_\theta(z))] \right)$$

would yield a calculation of $W(P_r, P_\theta)$ up to a multiplicative constant K

Wasserstein GAN (WGAN)[1]

- Wasserstein GAN
  - When training the discriminator f_w(in WGAN, it is called as critic), we want to maximize (w is the parameter need to be trained)

$$L = E_{x\sim P_r}[f_w(x)] - E_{z\sim p(z)}[f_w(g_\theta(z)]$$

    That is, we want to accurately estimate the Wasserstein distance of the current two distributions and then at the next round, the generator will try to minimize it
  - To make sure f_w is K-Lipschitz, a clipping technique is used, i.g., all the weights of f_w is constrained to [-c, c]
  - When training the generator, we can minimize,

$$L = E_{x\sim P_r}[f_w(x)] - E_{z\sim p(z)}[f_w(g_\theta(z)]$$

    Or equivalently by minimizing, $L = -E_{z\sim p(z)}[f_w(g_\theta(z)]$

    where w is fixed and θ is the generator's parameters that need to be updated

[1] M. Arjovsky et al., Wasserstein GANs, arXiv, 2017

Wasserstein GAN (WGAN)

- Wasserstein GAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

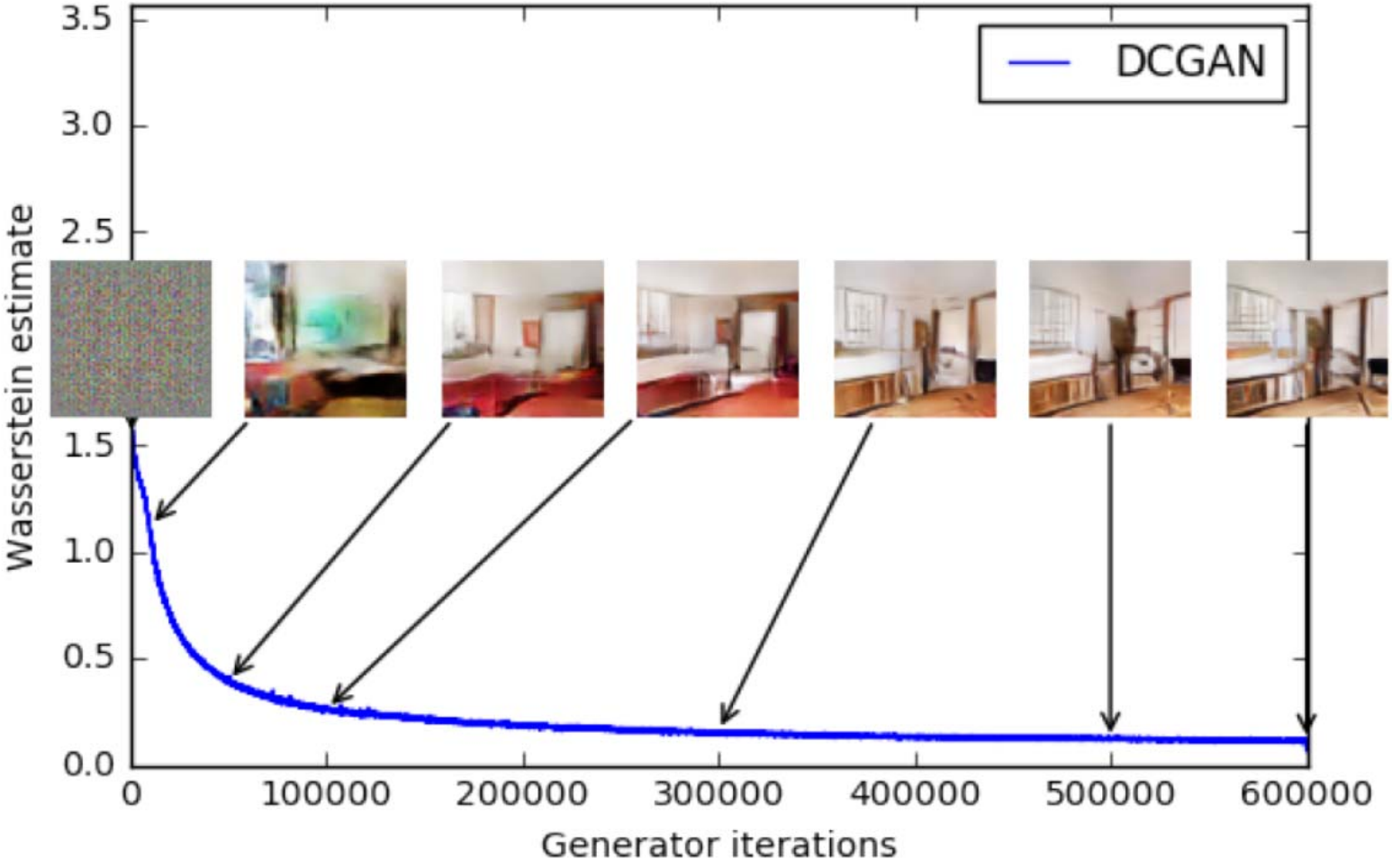
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Wasserstein GAN (WGAN)

- WGAN has a very excellent property: A meaningful loss metric
  - When training WGAN, we can get the current Wasserstein distance between the two distributions P_r and P_θ
  - Intuitively, a lower W-distance indicates that P_θ is closer to P_r , i.e., the current generator is better

On next page, you can see the relationship between the W-distance and the quality of the generator

Wasserstein GAN (WGAN)



WGAN with Gradient Penalty (WGAN-GP)[1]

- In original WGAN, to make sure the critic function is K-Lipschitz, it uses a weight-clipping technique
- Weight-clipping will lead to vanishing or exploding gradients
- Thus, a new technique is proposed to make the critic function K-Lipschitz, which is called 'gradient penalty'

  The new critical loss becomes

$$L = \underbrace{E_{\tilde{x}\sim P_g}\left[D(\tilde{x})\right] - E_{x\sim P_r}\left[D(x)\right]}_{\text{original critic}} + \underbrace{\lambda E_{\bar{x}\sim P_{\bar{x}}}\left[\left(\left\|\nabla_{\bar{x}} D(\bar{x})\right\|_2 - 1\right)^2\right]}_{\text{gradient penalty}}$$

[1] I. Gulrajani et al., Improved training of Wasserstein GANs, arXiv, 2017

WGAN with Gradient Penalty (WGAN-GP)

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.

Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Outline

- Introduction
- Theoretical Part
  - Generative Adversarial Networks (GAN)
  - Deep Convolutional GAN (DCGAN)
  - Wassertein GAN (WGAN)
  - Conditional GANs (cGANs)
- Application Part
- Existing Implementations Using Tensorflow

Conditional Generative Adversarial Nets (cGANs)[1]

- In an unconditioned generative model, there is no control on modes of the data being generated
- By conditioning the model on additional information it is possible to direct the data generation process
- Such conditioning could be based on class labels, or even data from different modality (such as images)
- cGAN is a very general idea; that means the conditions can be of various form and the ways to add conditions can also vary

[1] M. Mirza et al., Conditional generative adversarial nets, arXiv, 2014

Conditional Generative Adversarial Nets (cGANs)

- The unconditioned GAN,

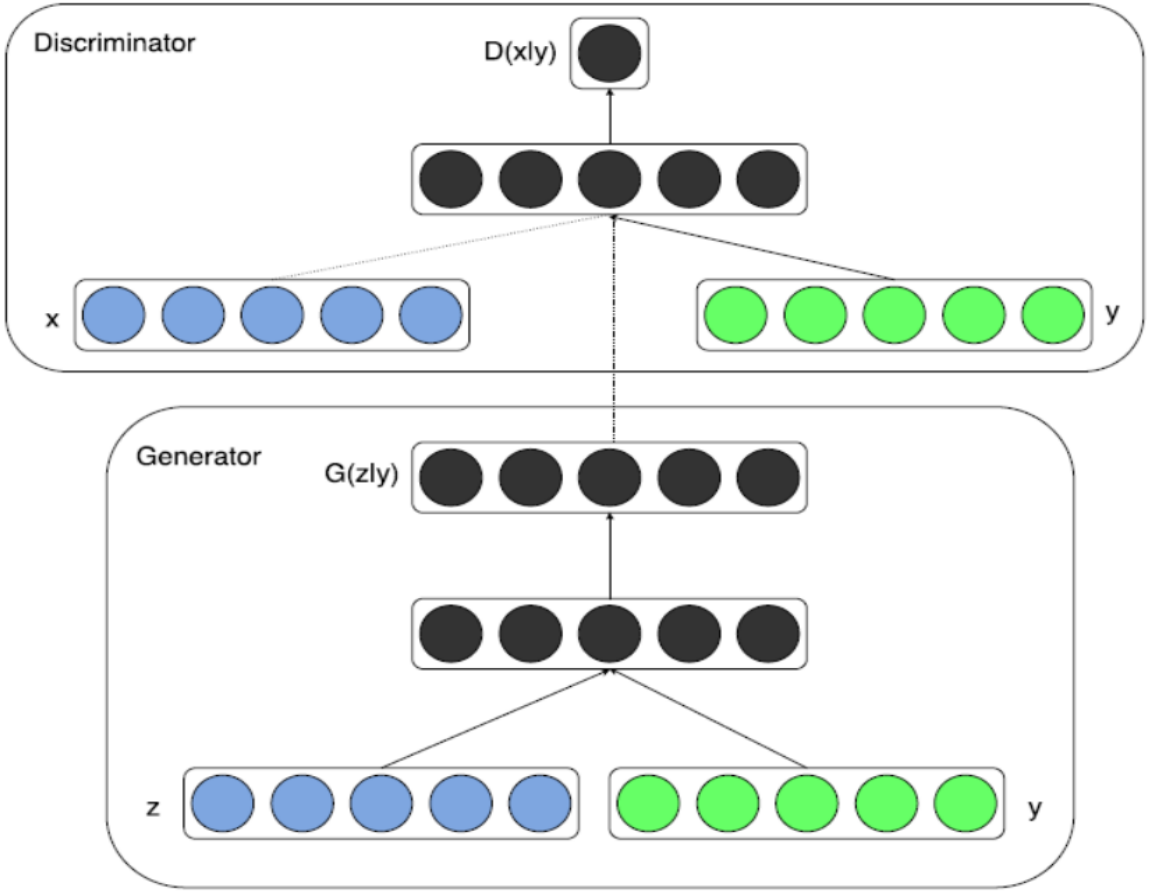
$$\min_G \max_D \left\{ V(D,G) = E_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log(1-D(G(\mathbf{z})))\right] \right\}$$

- GAN can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information ***y***,

$$\min_G \max_D \left\{ V(D,G) = E_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[\log D(\mathbf{x}\,|\,\mathbf{y})\right] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log(1-D(G(\mathbf{z}\,|\,\mathbf{y})))\right] \right\}$$

Conditional Generative Adversarial Nets (cGANs)

- The architecture of a simple conditional adversarial net



Outline

- Introduction
- Theoretical Part
- Application Part
  - Paired image-to-image translation (pix2pix)
  - Unpaired image-to-image translation (cycleGAN)
  - Image superresolution (SRGAN)
  - S+U learning (SimGAN)
- Existing Implementations Using Tensorflow

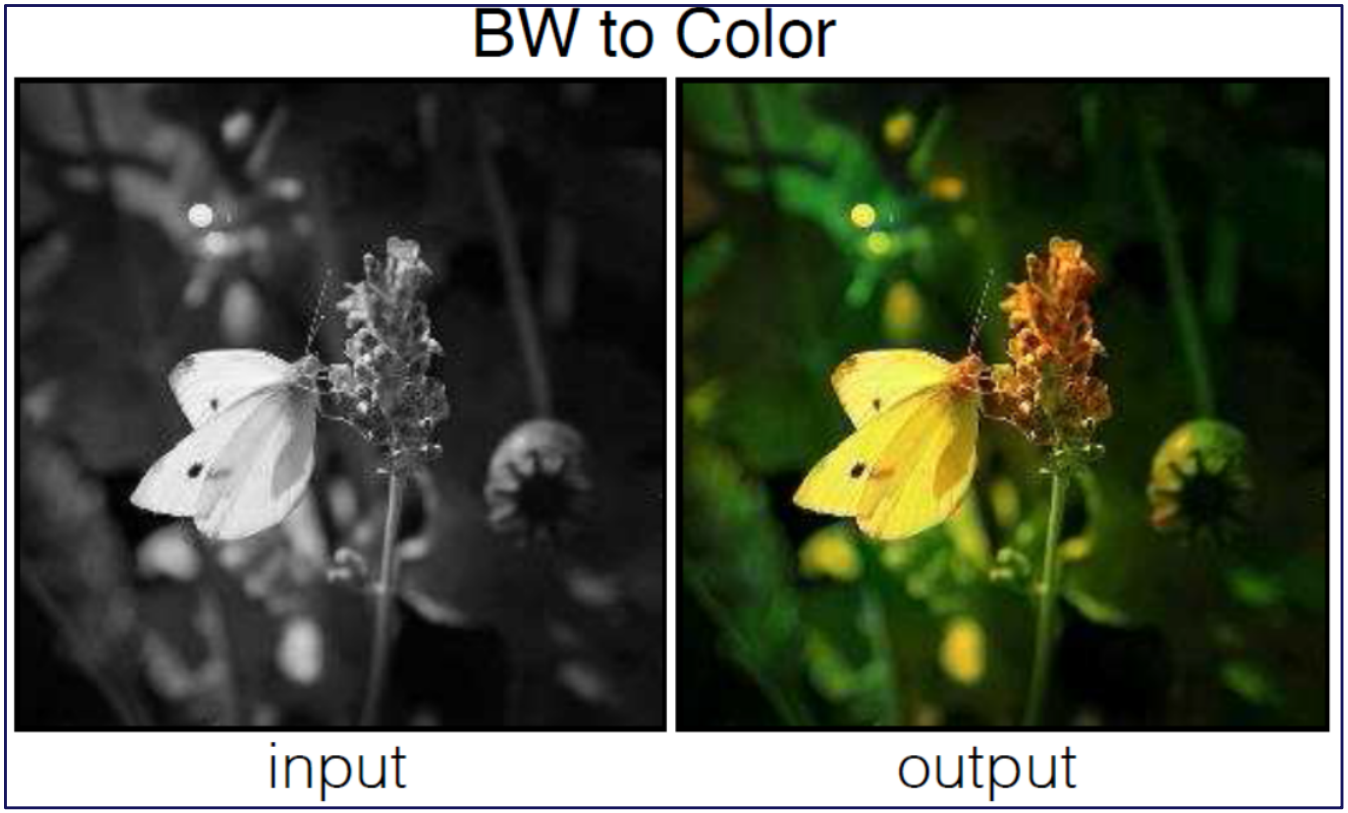
Paired Image-to-image translation (pix2pix) [1]

- Many problems in image processing, computer graphics, and CV can be posed as translating an input image into a corresponding output image

[1] P. Isola, J. Zhu, T. Zhou, and A.A. Efros, Image-to-image translation with conditional adversarial networks, CVPR, 2017

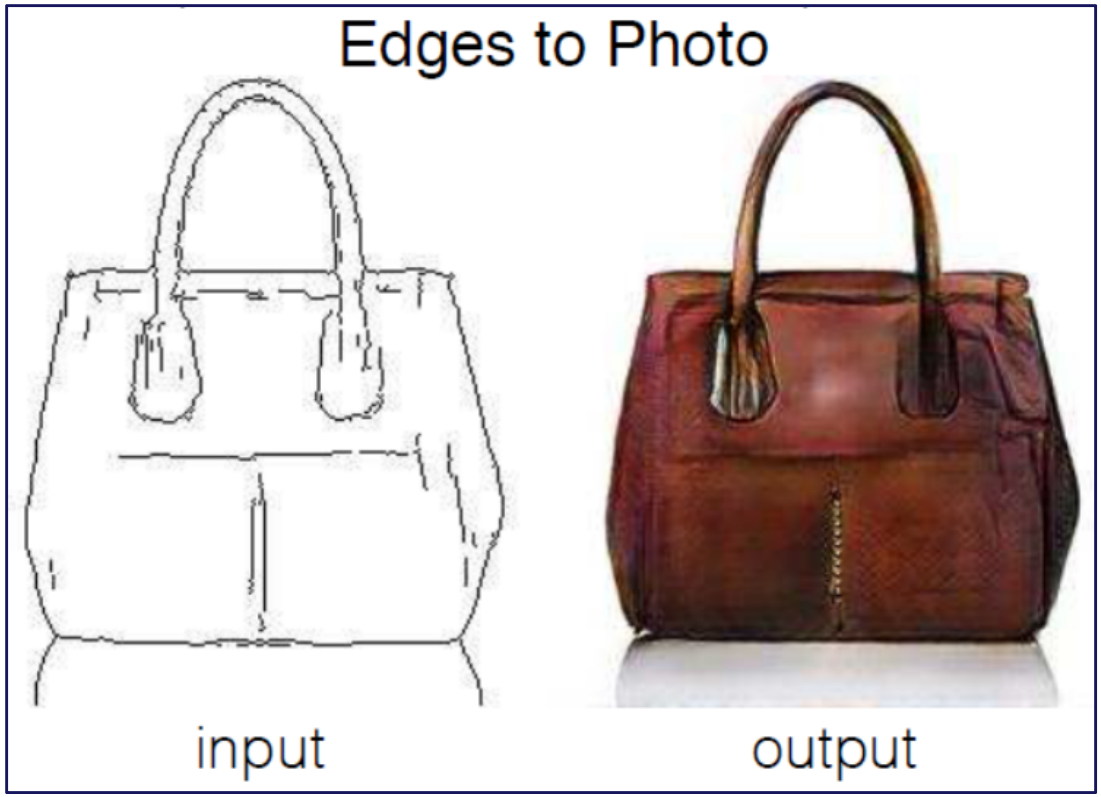
Paired Image-to-image translation (pix2pix)

- Many problems in image processing, computer graphics, and CV can be posed as translating an input image into a corresponding output image



Paired Image-to-image translation (pix2pix)

- Many problems in image processing, computer graphics, and CV can be posed as translating an input image into a corresponding output image

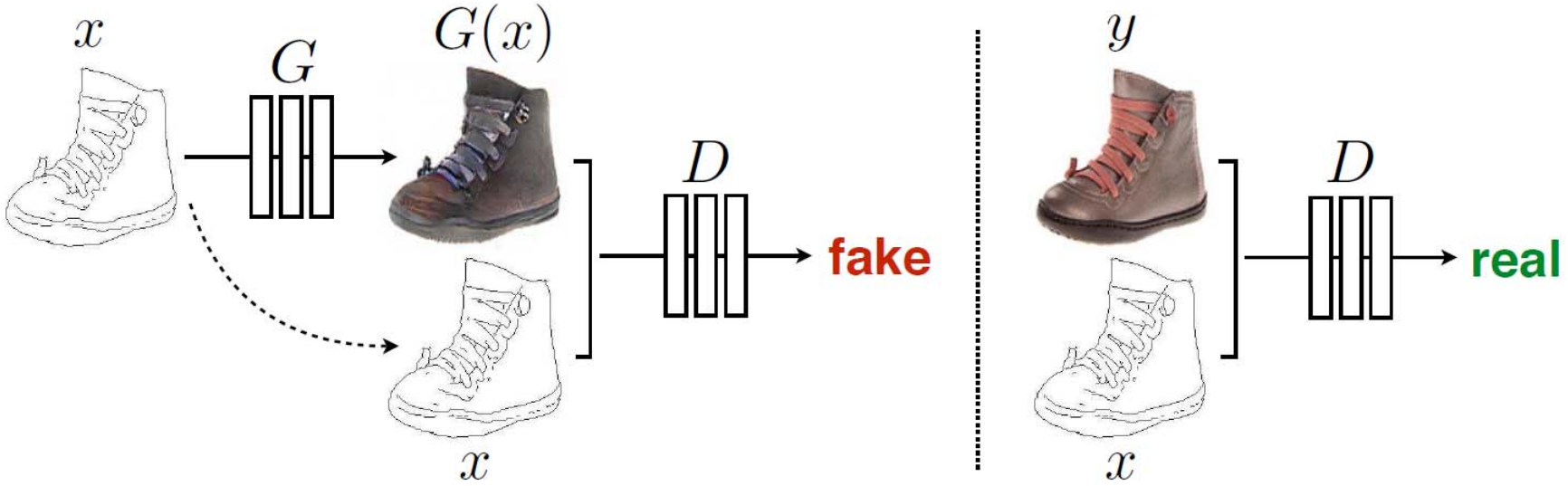


Paired Image-to-image translation (pix2pix)

- Many problems in image processing, computer graphics, and CV can be posed as translating an input image into a corresponding output image
- This work proposed a unified solution for these kinds of problems based on cGANs, which takes the input image as the condition

Paired Image-to-image translation (pix2pix)

Suppose that x is the input image, y is the corresponding output image
The generator G is trained to produce outputs that cannot be distinguished from real images by an adversarially trained discriminator D



Paired Image-to-image translation (pix2pix)

Suppose that x is the input image, y is the corresponding output image
The generator G is trained to produce outputs that cannot be distinguished from real images by an adversarially trained discriminator D

When updating the discriminator,

$$\theta_D^* = \arg\max_{\theta_D} E_{x:y}\left[\log D_{\theta_D}(x,y)\right] + E_{x,G(x)}\left[\log\left(1 - D_{\theta_D}(x, G_{\theta_G}(x))\right)\right]$$

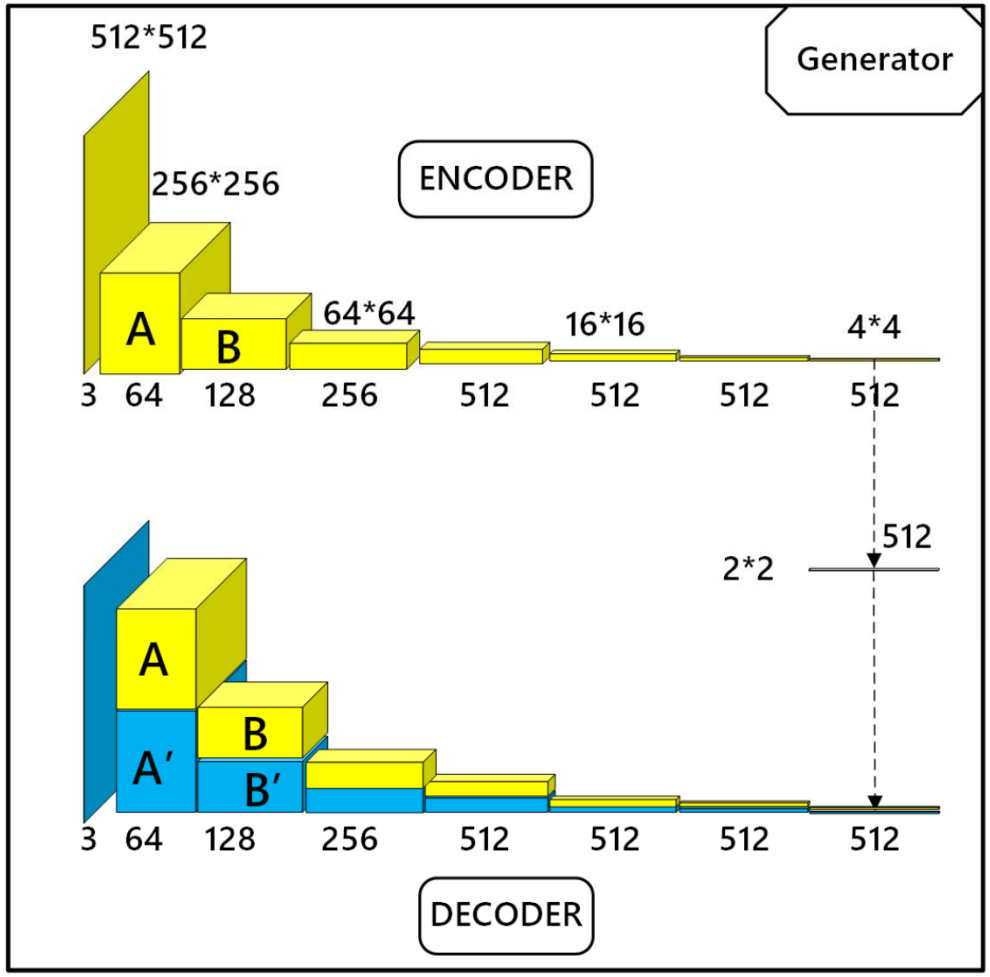
When updating the generator,

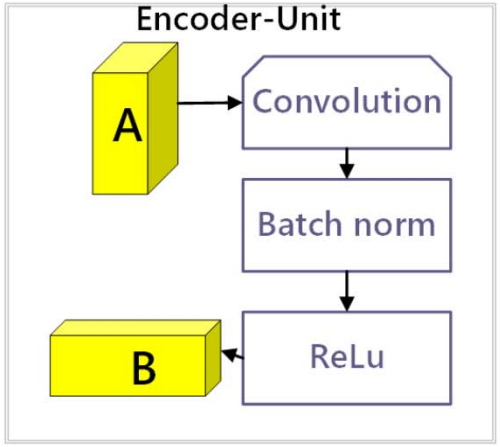
$$\theta_G^* = \arg\min_{\theta_G} E_{x,G(x)}\left[\log\left(1 - D_{\theta_D}(x, G_{\theta_G}(x))\right)\right] + \lambda_1 E_{x:y}\left[\left\|y - G_{\theta_G}(x)\right\|_1\right]$$

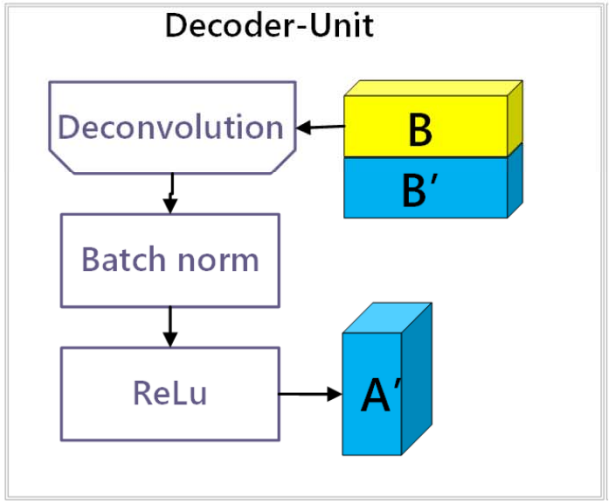
Paired Image-to-image translation (pix2pix)

- Implementation details
  - Input to the generator is an image
  - The generator uses a U-shaped encoder-decoder network, in which the corresponding decoder features will be concatenated with the feature maps from the encoder

Paired Image-to-image translation (pix2pix)



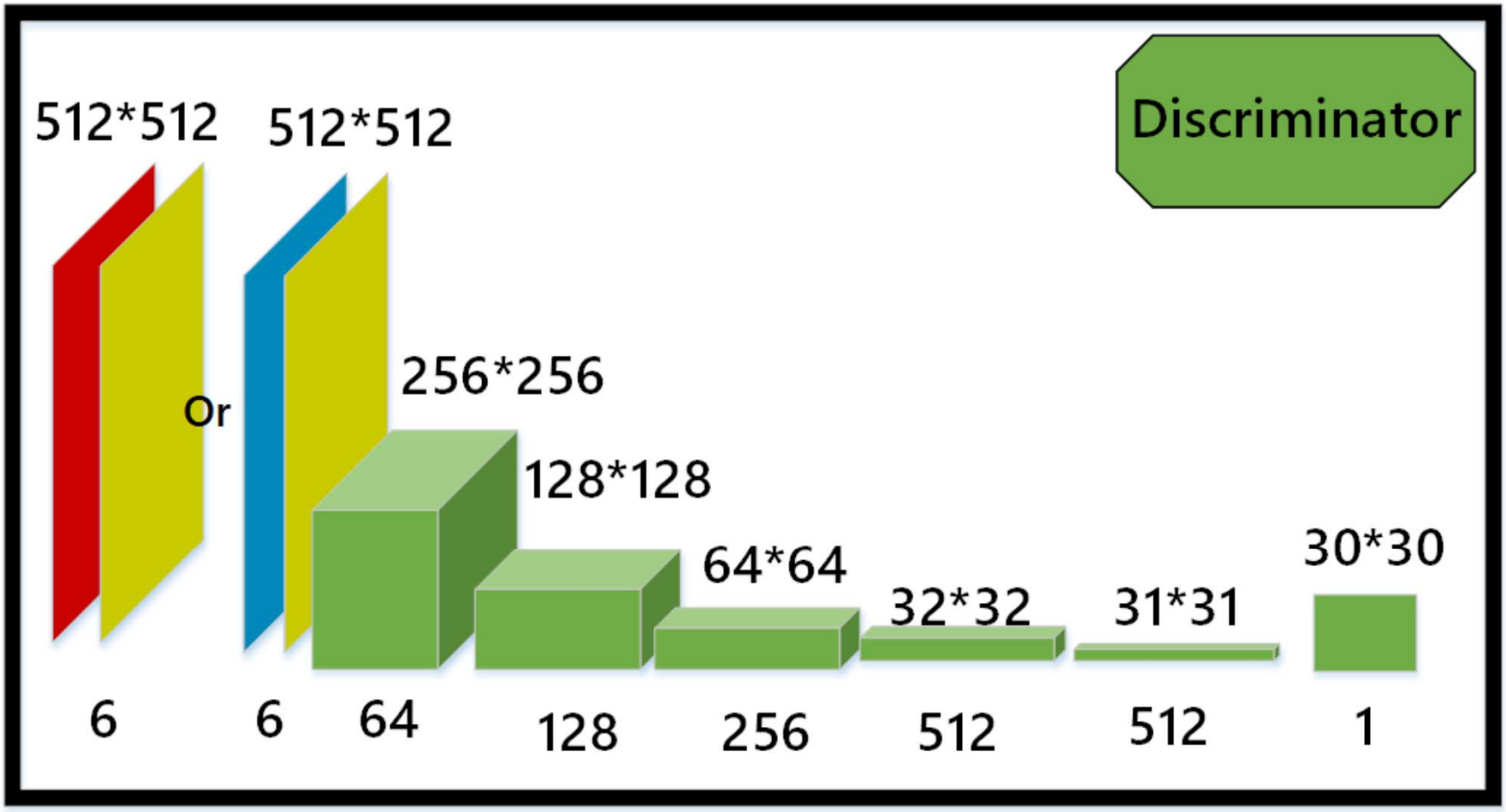




Paired Image-to-image translation (pix2pix)

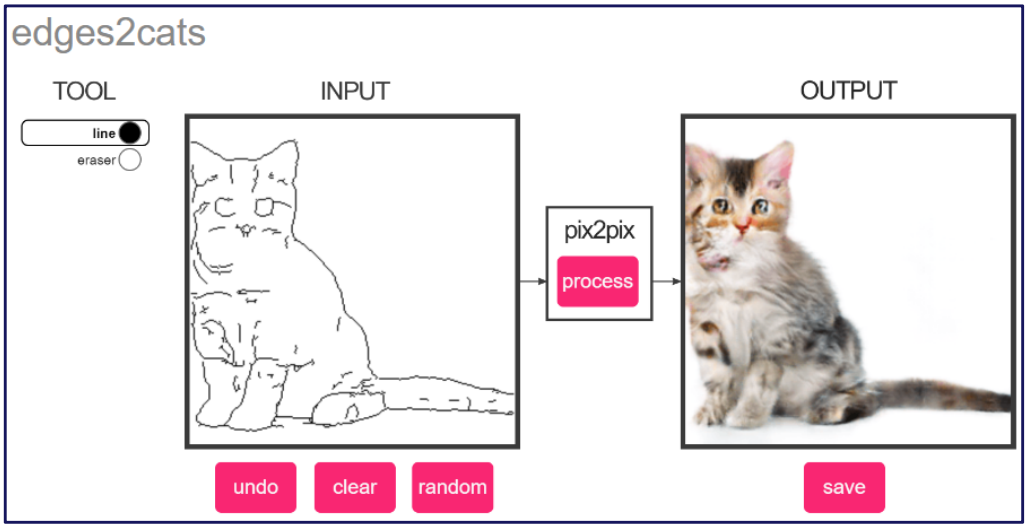
- Implementation details
  - Input to the generator is an image
  - The generator uses a U-shaped encoder-decoder network, in which the corresponding decoder features will be concatenated with the feature maps from the encoder
  - The output of the last layer of the discriminator is a matrix instead of a scalar; such a matrix can indicate the ‘fakeness’ or ‘realness’ at the patch level

Paired Image-to-image translation (pix2pix)



Paired Image-to-image translation (pix2pix)

Have Fun! You can try the online Image-to-Image demo implemented by Christopher Hesse https://affinelayer.com/pixsrv/



Outline

- Introduction
- Theoretical Part
- Application Part
  - Paired image-to-image translation (pix2pix)
  - Unpaired image-to-image translation (CycleGAN)
  - Image superresolution (SRGAN)
  - S+U learning (SimGAN)
- Existing Implementations Using Tensorflow

Unpaired image-to-image translation (CycleGAN) [1]

- Problem definition

This is one Monet's artwork

What was the real scene when Monet draw this picture?

[1] J. Zhu et al., Unpaired image-to-image translation using cycle-consistent adversarial networks, arXiv, 2017

Unpaired image-to-image translation (CycleGAN)[1]

- Problem definition

This is one Monet's artwork

This is a computer generated image

[1] J. Zhu et al., Unpaired image-to-image translation using cycle-consistent adversarial networks, arXiv, 2017

Unpaired image-to-image translation (CycleGAN)

- Problem definition
  - It is also an image-to-image translation problem
  - However, for training, we do not have corresponding input pairs

Source domain X, target domain Y, no paired examples
Goal: learning a mapping $G : X \rightarrow Y$, such that the distribution of images from $G(X)$ is indistinguishable from the distribution of Y using an adversarial loss

Unpaired image-to-image translation (CycleGAN)

- Key idea: Cycle consistent
  - E.g., if we translate a sentence from Chinese to English, and then translate it back from English to Chinese, we should arrive back at the original sentence
  - Mathematically, if we have a translator $G : X \to Y$ and another translator $F : Y \to X$ then G and F should be inverses of each other
  - Cycle consistency loss encourages $F(G(x)) \approx x, G(F(y)) \approx y$

Unpaired image-to-image translation (CycleGAN)

- Key idea: Cycle consistent

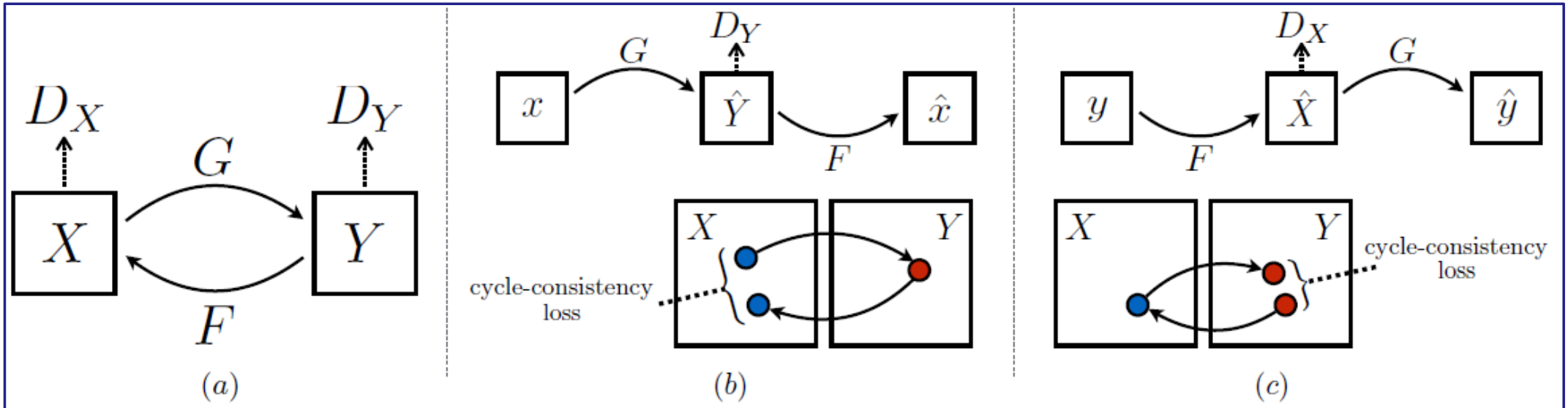


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

Unpaired image-to-image translation (CycleGAN)

- Key idea: Cycle consistent

The objective is, $G^*, F^* = \arg\min_{G,F} \max_{D_X, D_Y} L(G, F, D_X, D_Y)$

where,

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cyc}(G, F)$$

$$L_{GAN}(G, D_Y, X, Y) = E_{y \sim p_{data}(y)}\left[\log D_Y(y)\right] + E_{x \sim p_{data}(x)}\left[\log(1 - D_Y(G(x)))\right]$$

$$L_{GAN}(F, D_X, Y, X) = E_{x \sim p_{data}(x)}\left[\log D_X(x)\right] + E_{y \sim p_{data}(y)}\left[\log(1 - D_X(F(y)))\right]$$

$$L_{cyc}(G, F) = E_{x \sim p_{data}(x)}\left[\left\|F(G(x)) - x\right\|_1\right] + E_{y \sim p_{data}(y)}\left[\left\|G(F(y)) - y\right\|_1\right]$$

Unpaired image-to-image translation (CycleGAN)—Examples

image

Van Gogh style

Unpaired image-to-image translation (CycleGAN)—Examples

Outline

- Introduction
- Theoretical Part
- Application Part
  - Paired image-to-image translation (pix2pix)
  - Unpaired image-to-image translation (cycleGAN)
  - Image superresolution (SRGAN)
  - S+U learning (SimGAN)
- Existing Implementations Using Tensorflow

Image Super-resolution (SRGAN)

- Problem definition
  - The task of super-resolution is to estimate a high-resolution (HR) image from its low-resolution counterpart
  - It is a highly ill-posed problem
  - In some occasions, it is also referred to as image interpolation
  - ‘Nearest neighbor’, ‘bilinear’, and ‘bicubic’ are the three most common interpolation methods provided in Image Processing packages, such as OpenCV and Matlab
  - Modern techniques to solve this problem is based on machine learning technologies; the essence is to learn a function, mapping from the low-resolution image space to the high-resolution image space

Image Super-resolution (SRGAN)[1]

- Key ideas of SRGAN
  - It uses a GAN architecture to make sure the SR images are in the manifold of natural HR images
  - The loss used for updating the generator consists of two parts: the adversarial loss and the content loss

[1] C. Ledig et al., Photo-realistic single image super-resolution using a generative adversarial network, CVPR, 2017

Image Super-resolution (SRGAN)

When updating the discriminator,

$$\theta_D^* = \arg\max_{\theta_D} E_{I^{HR}\sim p_{train}(I^{HR})}\left[\log D_{\theta_D}(I^{HR})\right] + E_{I^{LR}\sim p_G(I^{LR})}\left[\log\left(1 - D_{\theta_D}\left(G_{\theta_G}\left(I^{LR}\right)\right)\right)\right]$$

When updating the generator,

$$\theta_G^* = \arg\min_{\theta_G} E_{I^{LR}\sim p_G(I^{LR})}\left[\log\left(1 - D_{\theta_D}\left(G_{\theta_G}\left(I^{LR}\right)\right)\right)\right]$$

$$+\lambda_1 E_{I^{LR}\sim p_G(I^{LR})}\left[\left\| I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y}\right\|^2\right]$$

$$+\lambda_2 E_{I^{LR}\sim p_G(I^{LR})}\left[\left\| \phi_{i,j}\left(I^{HR}\right)_{x,y} - \phi_{i,j}\left(G_{\theta_G}(I^{LR})\right)_{x,y}\right\|^2\right]$$

where $\phi_{i,j}$ indicates the feature map obtained by the *j*-th convolution before the *i*-th max-pooling layer with the VGG19 network

Image Super-resolution (SRGAN)

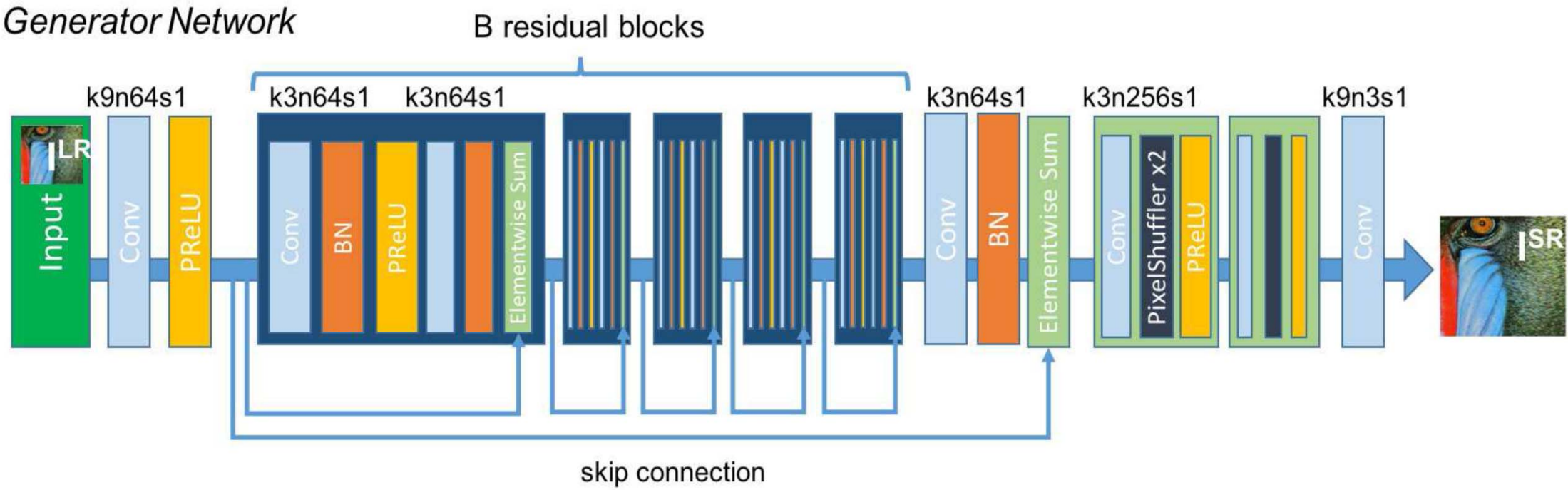


Image Super-resolution (SRGAN)

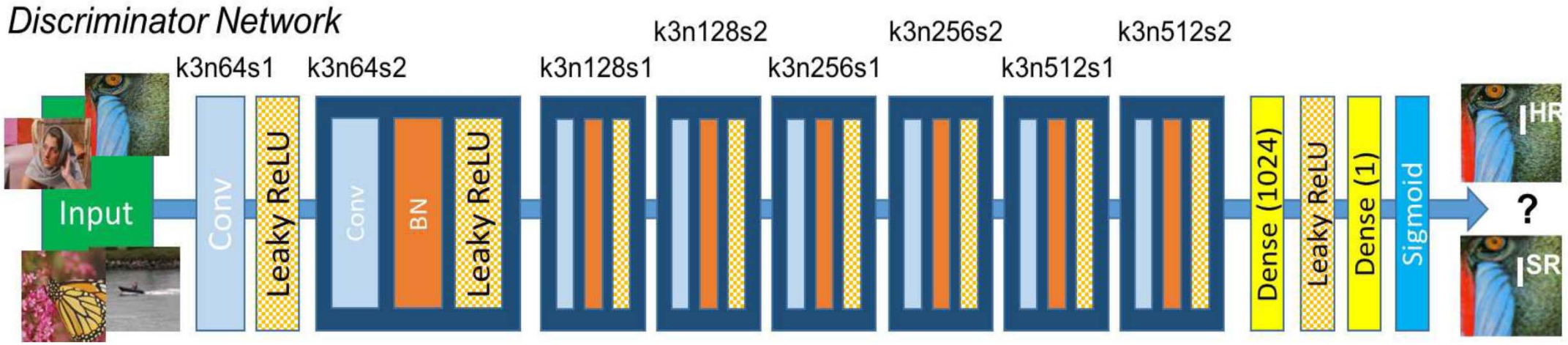
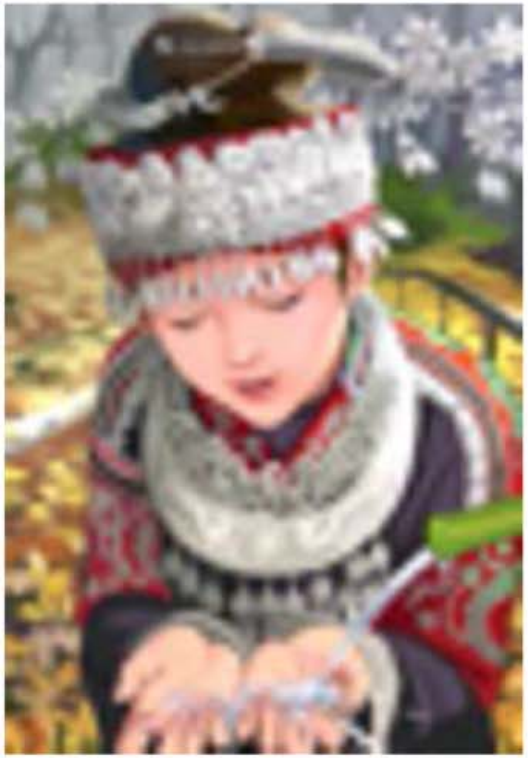


Image Super-resolution (SRGAN)--Sample Results

bicubic
(21.59dB/0.6423)



SRResNet
(23.53dB/0.7832)

SRGAN
(21.15dB/0.6868)

original

Outline

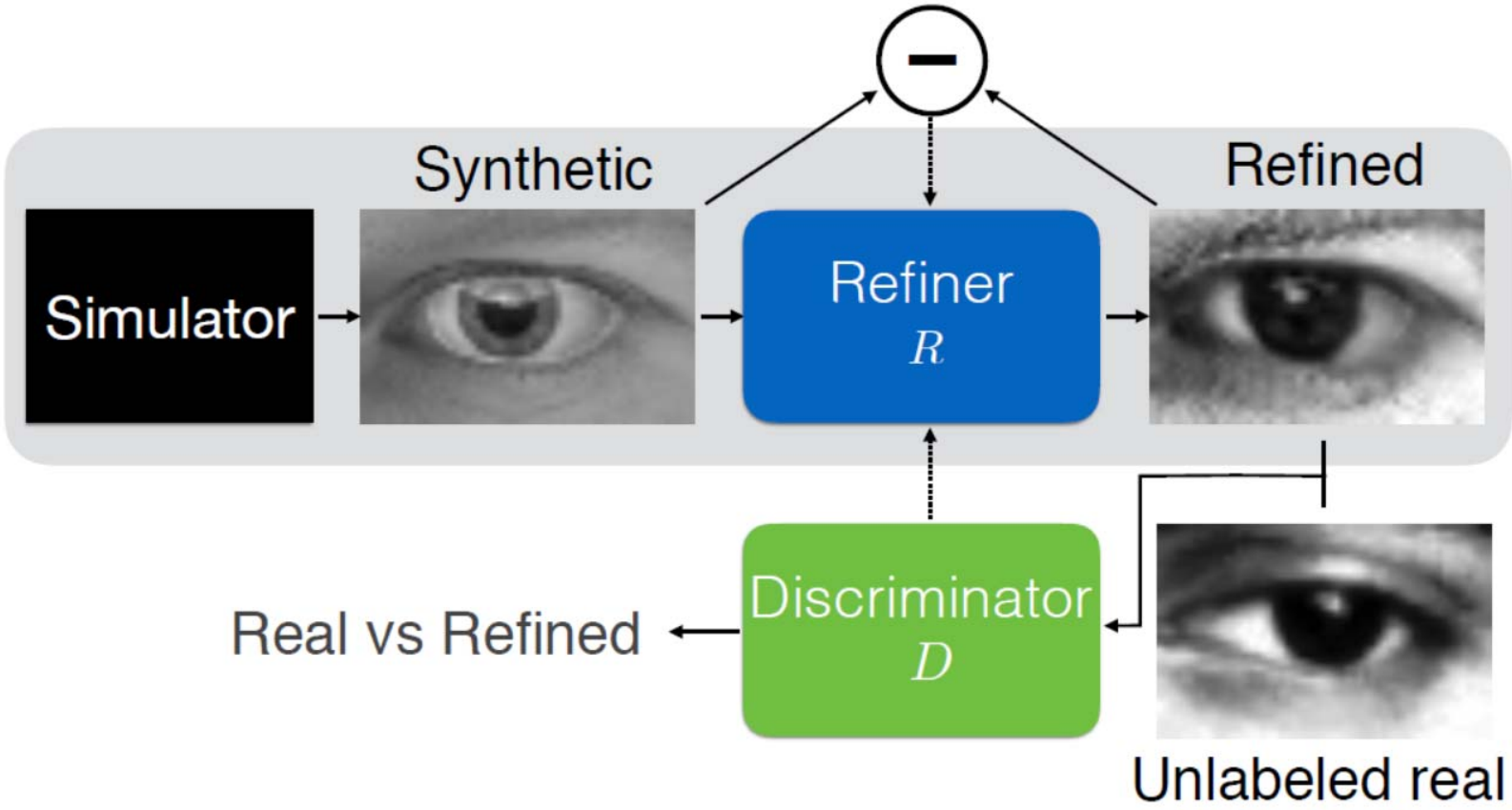
- Introduction
- Theoretical Part
- Application Part
  - Paired image-to-image translation (pix2pix)
  - Unpaired image-to-image translation (cycleGAN)
  - Image superresolution (SRGAN)
  - S+U learning (SimGAN)
- Existing Implementations Using Tensorflow

S+U Learning (SimGAN)[1]

- Apple's first public AI paper
- Problem formulation
  - Recently, it has become tractable to train models on synthetic images, to avoid the need for expensive annotations
  - However, learning from synthetic images may not achieve the desired performance due to a gap between synthetic and real image distributions
  - To reduce this gap, the authors proposed Simulated + Unlabeled (S+U) learning, where the task is to improve the realism of a simulator's output using unlabeled real data while preserving the annotation information from the simulator

[1] A. Shrivastava et al., Learning from simulated and unsupervised images through adversarial training, CVPR, 2017

S+U Learning (SimGAN)



Overview of SimGAN

S+U Learning (SimGAN)

- The goal of SimGAN is to use a set of unlabeled real images $y_i \in \mathbf{Y}$ to learn a refiner $R_\theta(x)$ that refines a synthetic image x
- The refined image is denoted by $x^* := R_\theta(x)$. The key requirement for S+U learning is that the refined image x^* should look like a real image in appearance while preserving the annotation information from the simulator
- The key idea is that the loss for the generator contains two parts, the adversarial loss and a self regularization term that penalizes large changes between the synthetic and refined images

S+U Learning (SimGAN)

When updating the discriminator,

$$\phi^* = \arg\max_{\phi} E_{y\sim \mathbf{Y}}\left[\log D_\phi(y)\right] + E_{x\sim p_{syn}(x)}\left[\log\left(1 - D_\phi\left(R_\theta\left(x\right)\right)\right)\right]$$

When updating the generator,

$$\theta^* = \arg\min_{\theta} E_{x\sim p_{syn}(x)}\left[\log\left(1 - D_\phi\left(R_\theta\left(x\right)\right)\right)\right] + \lambda E_{x\sim p_{syn}(x)}\left[\left\|x - R_\theta(x)\right\|_1\right]$$

S+U Learning (SimGAN)

- Other implementation details
  - They limit the discriminator's receptive field to local regions instead of the whole image, resulting in multiple local adversarial loss per image (the same as the patchGAN idea in pix2pix)

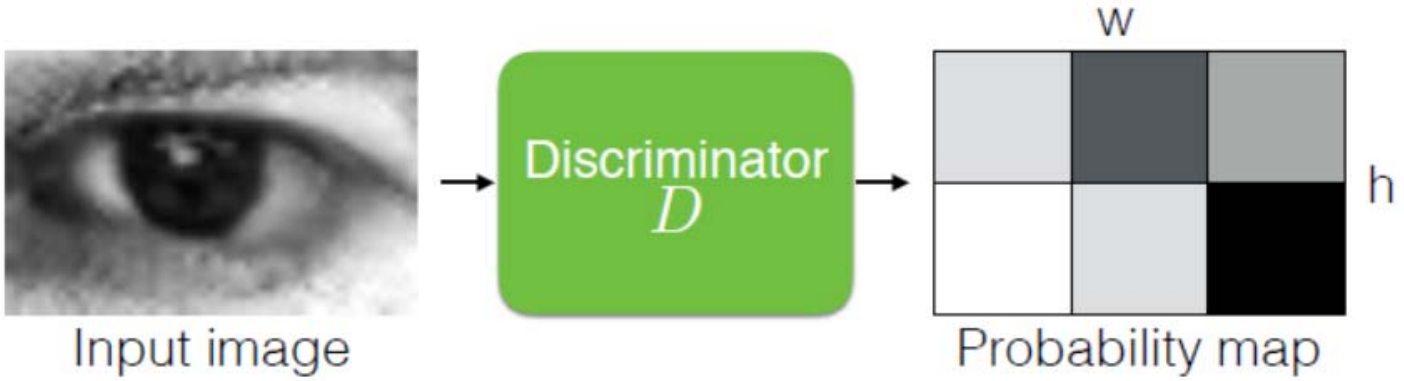
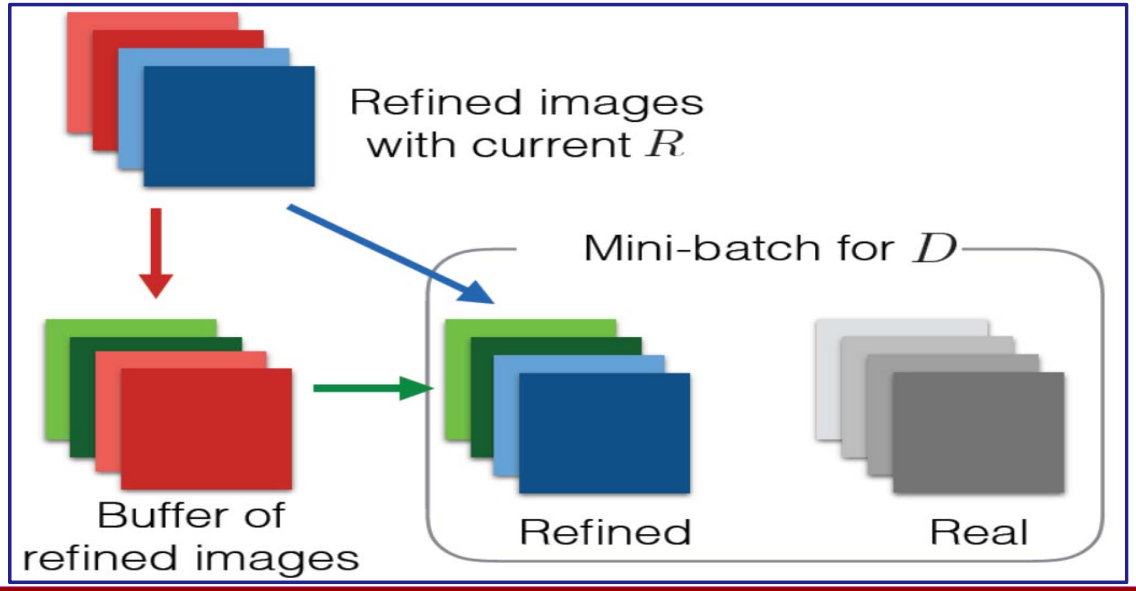


Figure 3. Illustration of local adversarial loss. The discriminator network outputs a $w \times h$ probability map. The adversarial loss function is the sum of the cross-entropy losses over the local patches.

S+U Learning (SimGAN)

- Other implementation details
  - They limit the discriminator's receptive field to local regions instead of the whole image, resulting in multiple local adversarial loss per image
  - They introduce a method for improving the stability of training by updating the discriminator using a history of refined images rather than only the ones from the current refiner network



S+U Learning (SimGAN)—Examples

Outline

- Introduction
- Theoretical Part
- Application Part
- Existing Implementations Using Tensorflow

Existing Implementations Using Tensorflow

- DCGAN
  - https://github.com/carpedm20/DCGAN-tensorflow
- Pix2Pix
  - The best implementation is https://github.com/affinelayer/pix2pix-tensorflow
- CycleGAN
  - I haven't found a perfect Tensorflow Implementation; you may try the original Pytorch implementation
- SRGAN
  - https://github.com/zsdonghao/SRGAN
- SimGAN
  - https://github.com/carpedm20/simulated-unsupervised-tensorflow

Thanks!