STAT 453: Introduction to Deep Learning and Generative Models

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching

Lecture 18
Introduction to Generative Adversarial Networks

arXiv.org > stat > arXiv:1406.2661

Search...

Help | Advanced S

Statistics > Machine Learning

[Submitted on 10 Jun 2014]

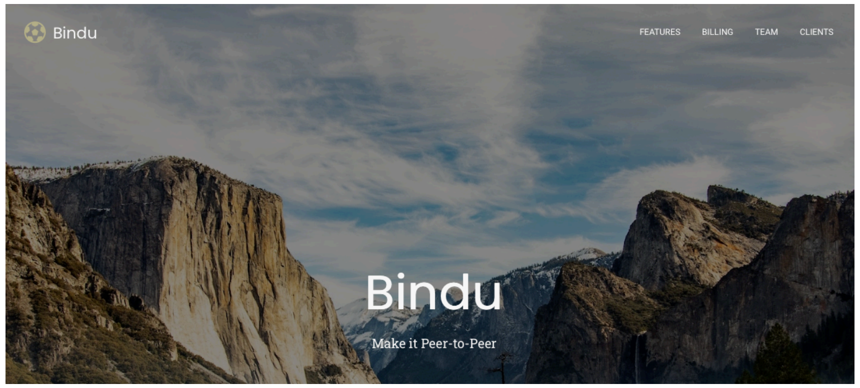
Generative Adversarial Networks

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: 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. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to 1/2 everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

https://arxiv.org/abs/1406.2661

https://thiscatdoesnotexist.com



https://thisstartupdoesnotexist.com

https://thispersondoesnotexist.com

https://thisponydoesnotexist.net

Lecture Overview

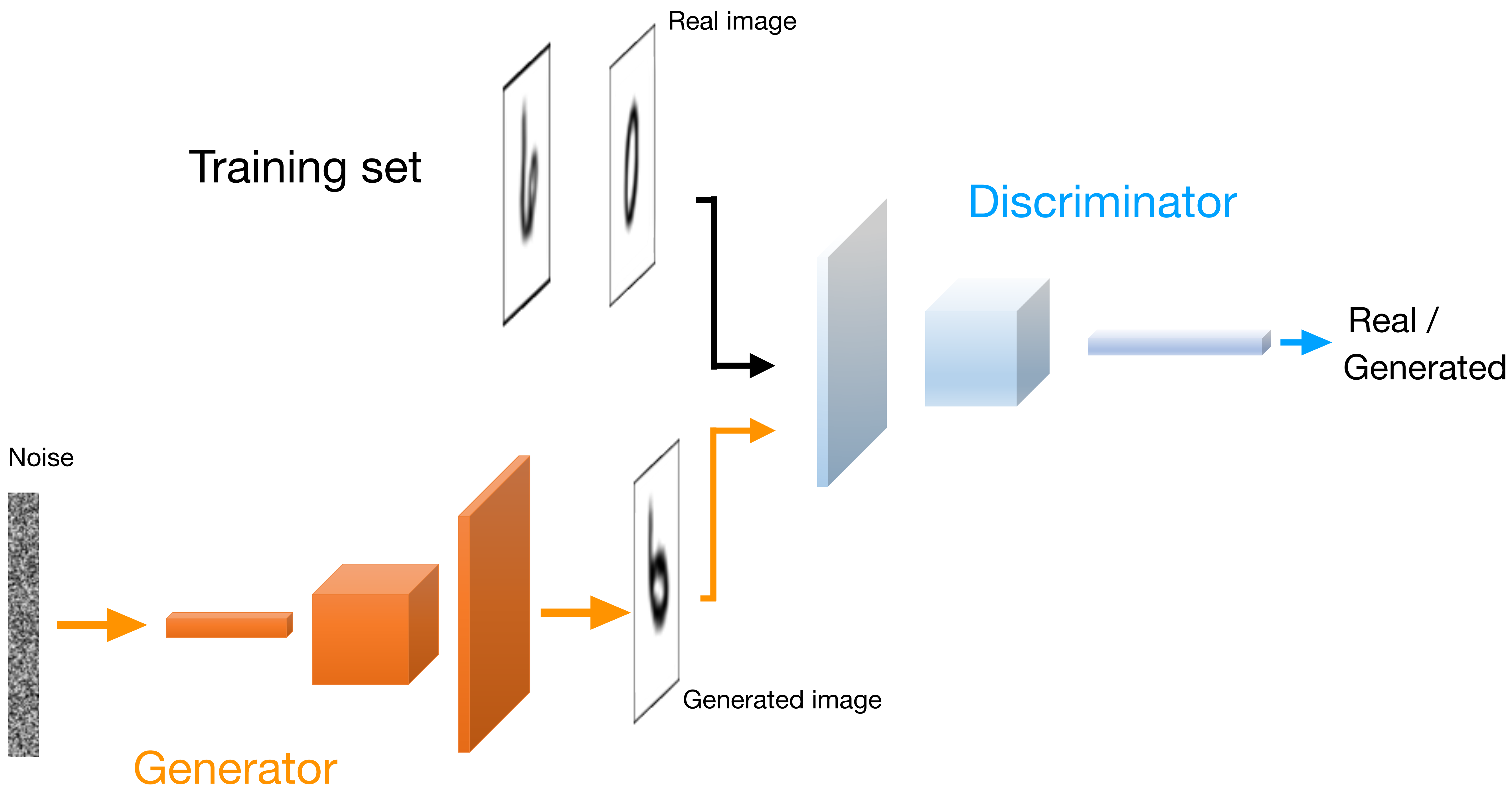
1. The Main Idea Behind GANs
2. The GAN Objective
3. Modifying the GAN Loss Function for Practical Use
4. A Simple GAN Generating Handwritten Digits in PyTorch
5. Tips and Tricks to Make GANs Work
6. A DCGAN for Generating Face Images in PyTorch

Letting two neural networks compete with each other

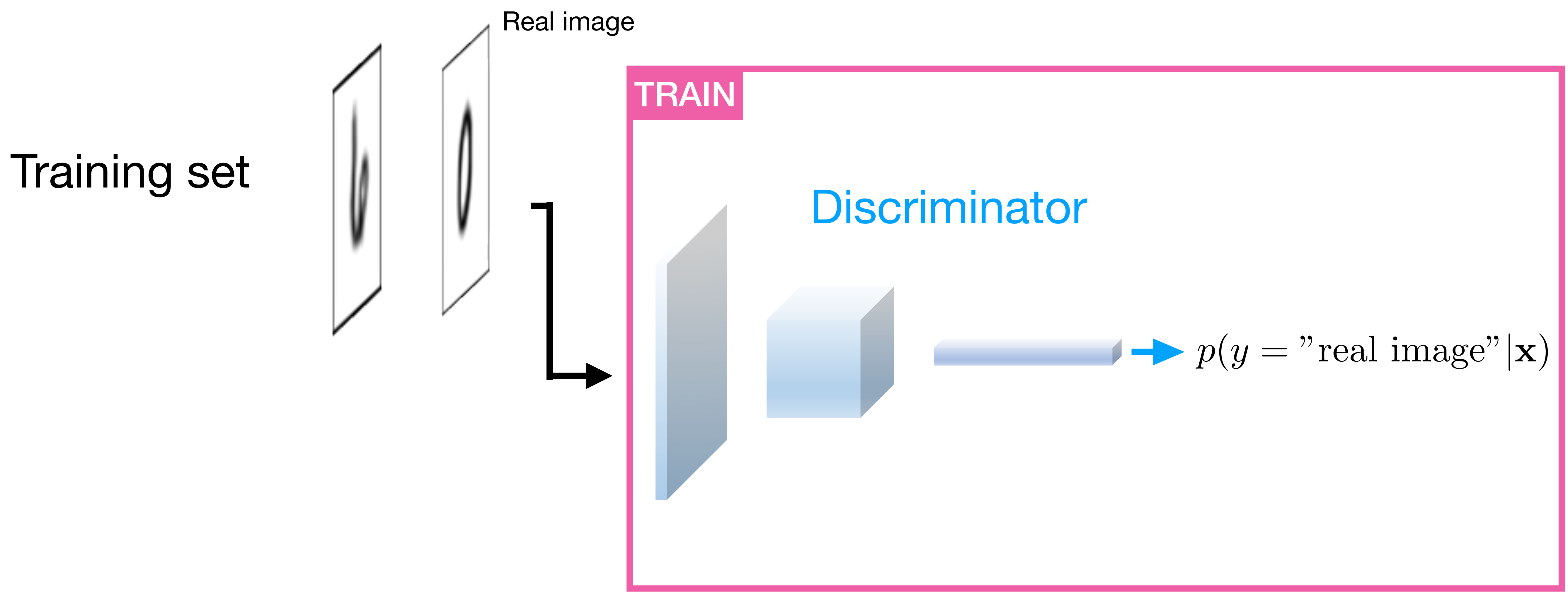
Generative Adversarial Networks (GAN)

- The original purpose is to generate new data
- Classically for generating new images, but applicable to wide range of domains
- Learns the training set distribution and can generate new images that have never been seen before
- Similar to VAE, and in contrast to e.g., autoregressive models or RNNs (generating one word at a time), GANs generate the whole output all at once

Deep Convolutional GAN (DCGAN or just GAN)

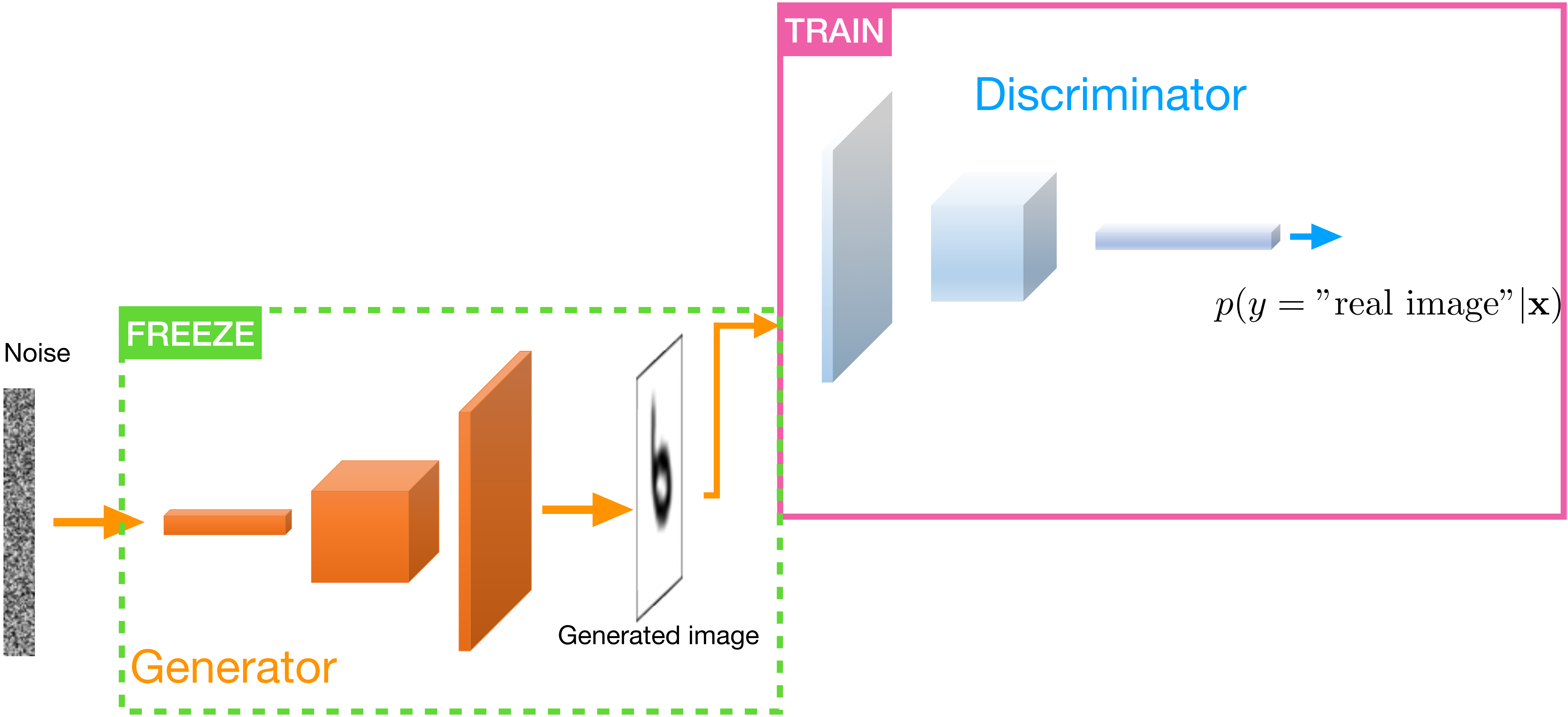


Step 1.1: Train Discriminator



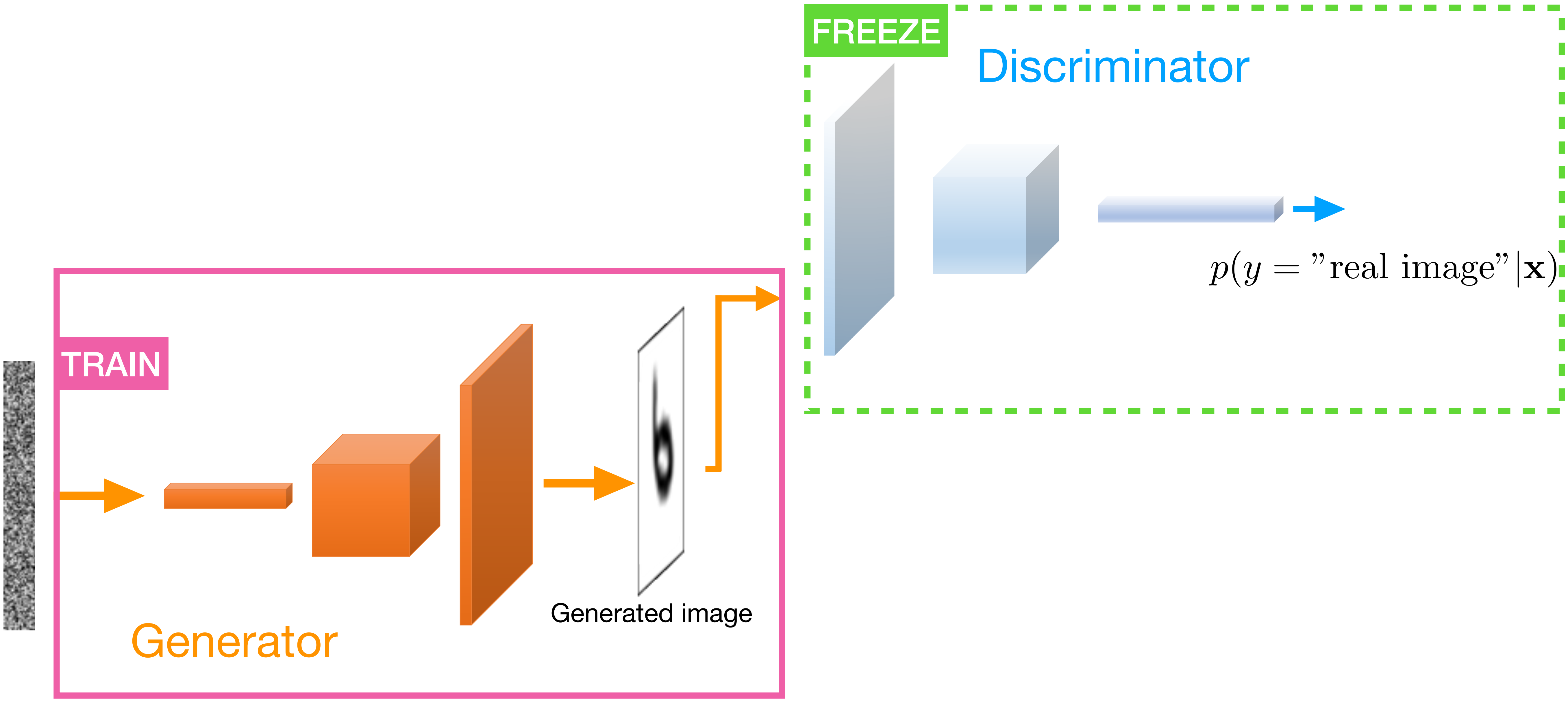
Train to predict that real image is real

Step 1.2: Train Discriminator



Train to predict that fake image is fake

Step 2: Train Generator



Train to predict that fake image is <u>real</u>

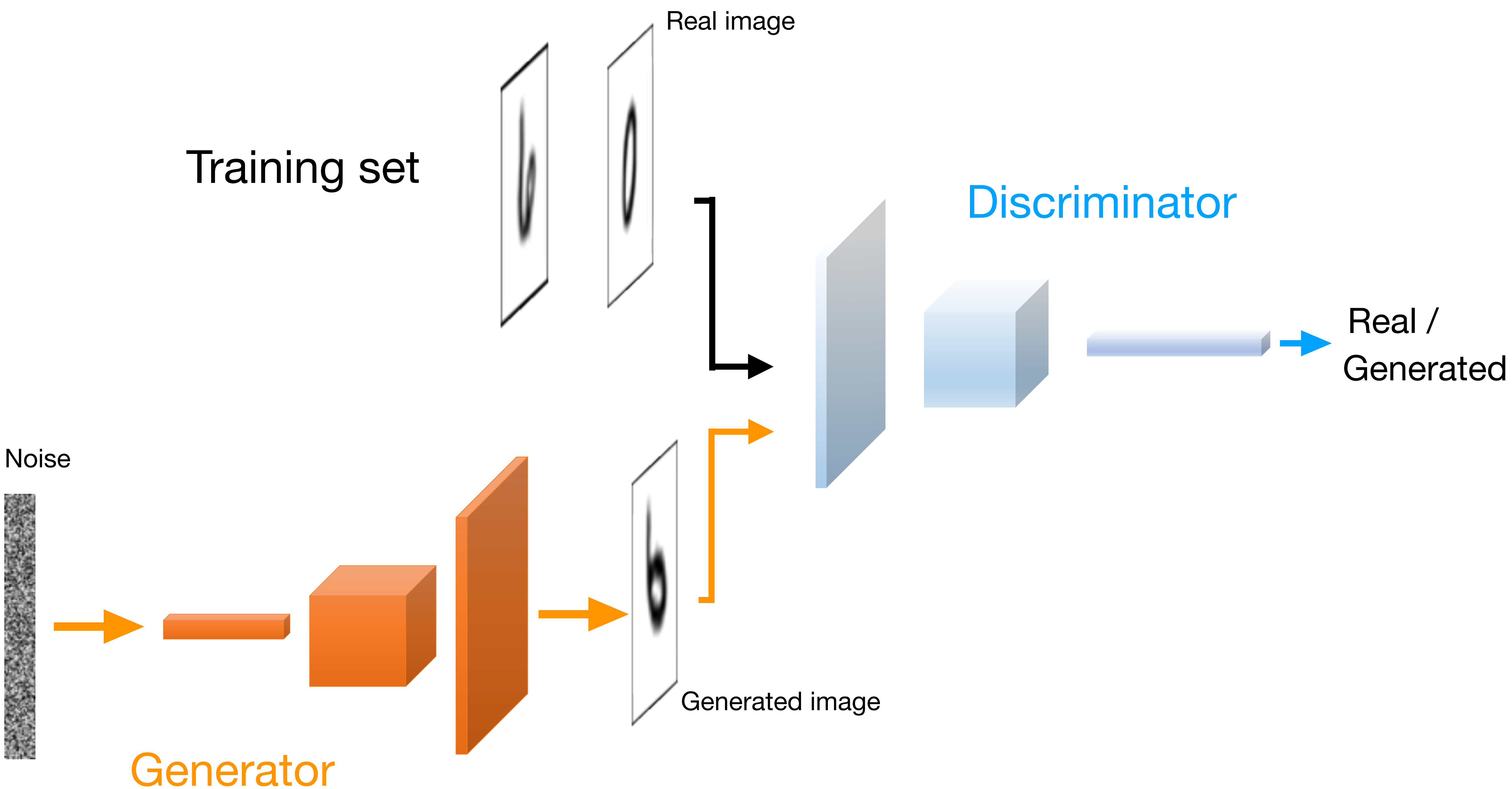
Adversarial Game

Discriminator: learns to become better at distinguishing real from generated images

Generator: learns to generate better images to fool the discriminator

How do the loss functions look like?

When Does a GAN Converge?



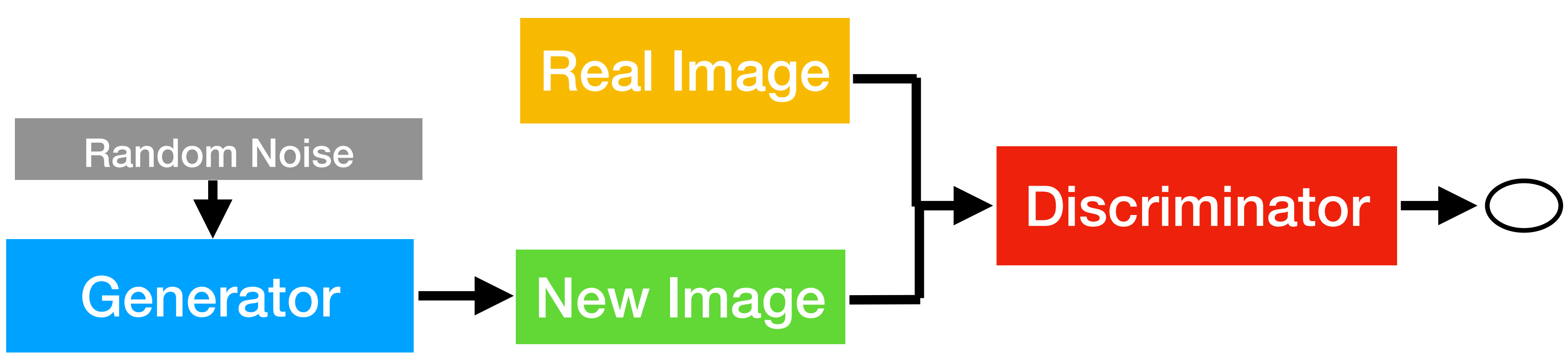
GAN Objective

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Discriminator gradient for update (gradient ascent):

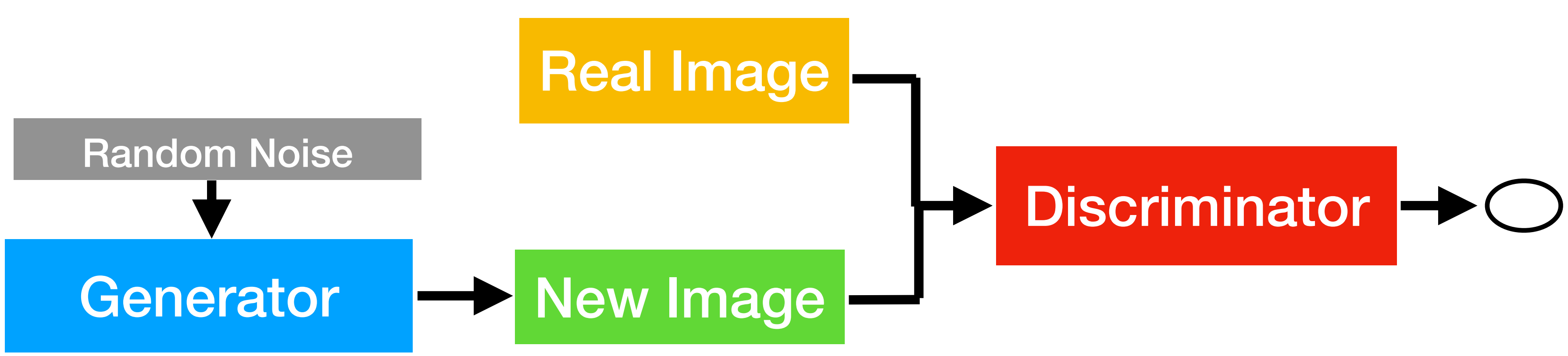
$$\nabla_{\mathbf{W}_D} \frac{1}{n} \sum_{i=1}^{n} \left[\log \overbrace{D\left(\boldsymbol{x}^{(i)}\right)}^{\substack{\text{predict well on real images} \\ \text{=> want probability close to 1}}} + \log \left(1 - \overbrace{D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)}^{\substack{\text{predict well on fake images} \\ \text{=> want probability close to 0}}}\right) \right]$$



$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Discriminator gradient for update (gradient ascent):

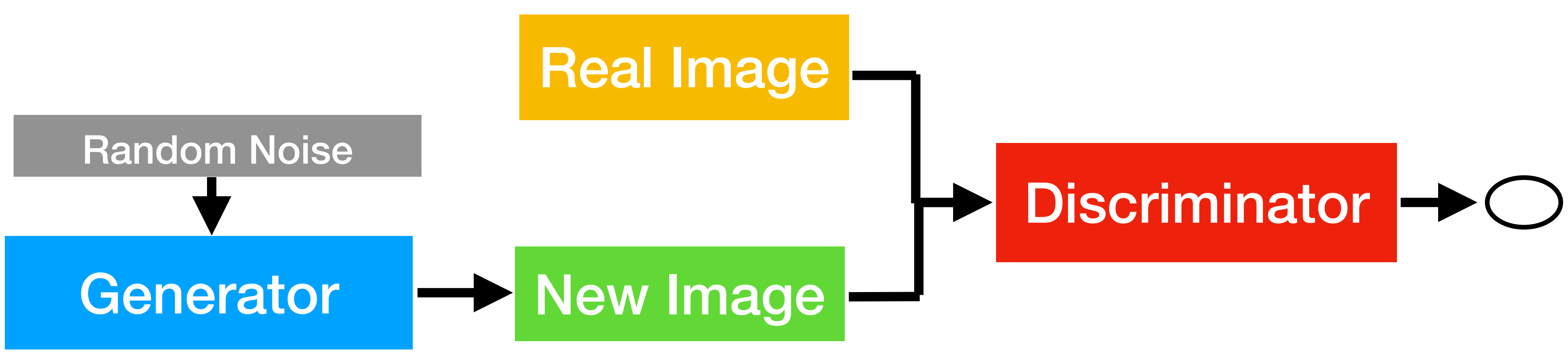
$$\nabla_{\mathbf{W}_D} \frac{1}{n} \sum_{i=1}^{n} \left[\overbrace{\log D\left(\boldsymbol{x}^{(i)}\right)}^{\substack{\text{predict well on real images} \\ \text{=> want probability close to 1}}} + \overbrace{\log \left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)}^{\substack{\text{predict well on fake images} \\ \text{=> want probability close to 0}}} \right]$$



$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Generator gradient for update (gradient descent):

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(1 - \overbrace{D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)}^{\substack{\text{predict badly on fake images} \\ \text{=> want probability close to 1}}}\right)$$



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.

- Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets." In *Advances in Neural Information Processing Systems*, pp. 2672-2680. 2014.

GAN Convergence

- Converges when Nash-equilibrium (Game Theory concept) is reached in the minmax (zero-sum) game

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- Nash-equilibrium in Game Theory is reached when the actions of one player won't change depending on the opponent's actions
- Here, this means that the GAN produces realistic images and the discriminator outputs random predictions (probabilities close to 0.5)

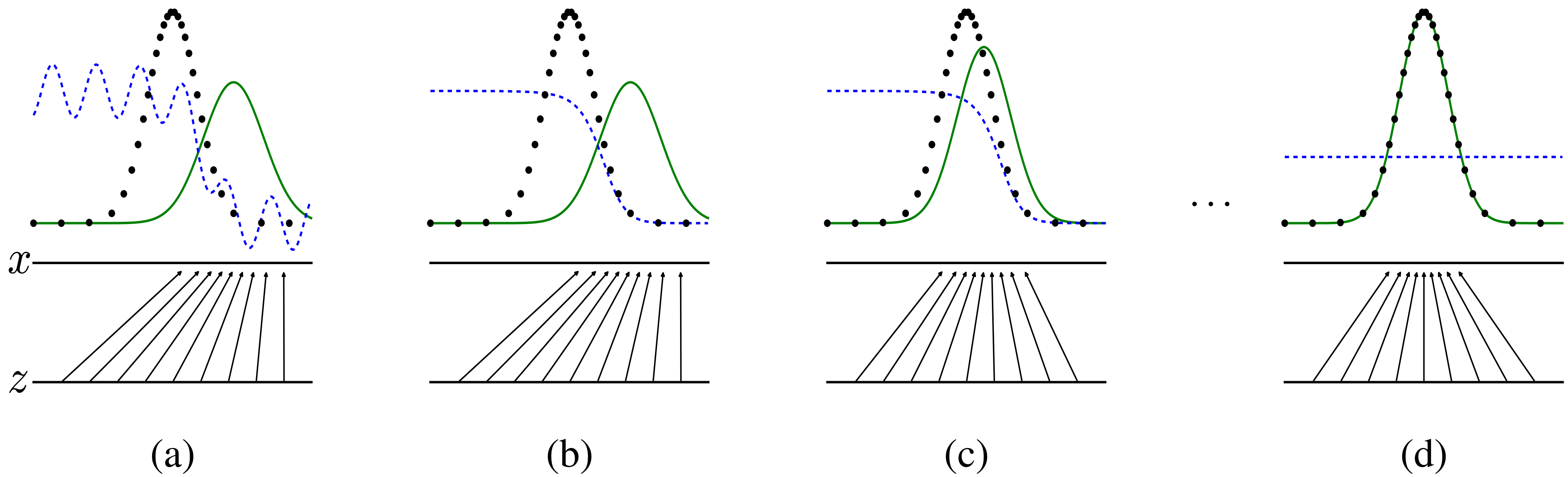


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

- Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets." In *Advances in Neural Information Processing Systems*, pp. 2672-2680. 2014.

Improving stochastic gradient descent for the generator

1. The Main Idea Behind GANs

2. The GAN Objective

3. Modifying the GAN Loss Function for Practical Use

4. A Simple GAN Generating Handwritten Digits in PyTorch

5. Tips and Tricks to Make GANs Work

6. A DCGAN for Generating Face Images in PyTorch

GAN Training Problems

- Oscillation between generator and discriminator loss
- Mode collapse (generator produces examples of a particular kind only)
- Discriminator is too strong, such that the gradient for the generator vanishes and the generator can't keep up
- Discriminator is too weak, and the generator produces non-realistic images that fool it too easily (rare problem, though)

GAN Training Problems

- Discriminator is too strong, such that the gradient for the generator vanishes and the generator can't keep up
- Can be fixed as follows:

Instead of gradient descent with

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

Do gradient ascent with

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

GAN Loss Function in Practice
(will be more clear in the code examples)

Discriminator

- Maximize prediction probability of classifying real as real and fake as fake
- Remember maximizing log likelihood is the same as minimizing negative log likelihood (i.e., minimizing cross-entropy)

Generator

- Minimize likelihood of the discriminator to make correct predictions (predict fake as fake; real as real), which can be achieved by maximizing the cross-entropy
- This doesn't work well in practice though because of small gradient issues
- Better: flip labels and minimize cross entropy (force the discriminator to output high probability for real if an image is fake)

gradient ascent

$$\nabla_{\mathbf{W}_D} \frac{1}{n} \sum_{i=1}^{n} \left[\underbrace{\log D\left(\boldsymbol{x}^{(i)}\right)}_{\substack{\text{predict well on real images} \\ \text{=> want probability close to 1}}} + \log \underbrace{\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)}_{\substack{\text{predict well on fake images} \\ \text{=> want probability close to 0}}} \right]$$

<u>Discriminator objective</u> in the neg. log-likelihood (binary cross entropy) perspective:

Real images, $y = 1$

$$\mathcal{L}(\mathbf{w}) = \boxed{-y^{(i)} \log\left(\hat{y}^{(i)}\right)} - \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

Want, $\hat{y} = 1$

Fake images, $y = 0$

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) \boxed{- \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)}$$

Want, $\hat{y} = 0$

gradient descent with

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

<u>Generator objective</u> in the neg. log-likelihood (binary cross entropy) perspective:

Fake images, $y = 0$

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) \boxed{-\left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)}$$

Want, $\hat{y} = 0$

Flip sign to "+" so that it turns into "want $\hat{y} = 1$"

It is better to flip the labels instead of the sign

gradient descent with

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

<u>Generator objective</u> in the neg. log-likelihood (binary cross entropy) perspective:

Fake images, $y = 0$

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) \boxed{- \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)}$$

Want, $\hat{y} = 0$

Flip sign to "+" so that it turns into "want $\hat{y} = 1$"

Do gradient ascent with

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(D \left(G \left(\boldsymbol{z}^{(i)} \right) \right) \right)$$

And flip labels

<u>Generator objective</u> in the neg. log-likelihood (binary cross entropy) perspective:

fake image label flipped -> real image label, $y = 1$

$$\mathcal{L}(\mathbf{w}) = \boxed{-y^{(i)} \log \left(\hat{y}^{(i)} \right)} - \left(1 - y^{(i)} \right) \log \left(1 - \hat{y}^{(i)} \right)$$

Want, $\hat{y} = 1$

Implementing our first GAN

1. The Main Idea Behind GANs
2. The GAN Objective
3. Modifying the GAN Loss Function for Practical Use
4. **A Simple GAN Generating Handwritten Digits in PyTorch**
5. Tips and Tricks to Make GANs Work
6. A DCGAN for Generating Face Images in PyTorch

How do the loss functions look like?

1. The Main Idea Behind GANs

2. The GAN Objective

3. Modifying the GAN Loss Function for Practical Use

4. A Simple GAN Generating Handwritten Digits in PyTorch

5. Tips and Tricks to Make GANs Work

6. A DCGAN for Generating Face Images in PyTorch

https://github.com/soumith/ganhacks

How do the loss functions look like?

1. The Main Idea Behind GANs

2. The GAN Objective

3. Modifying the GAN Loss Function for Practical Use

4. A Simple GAN Generating Handwritten Digits in PyTorch

5. Tips and Tricks to Make GANs Work

6. A DCGAN for Generating Face Images in PyTorch

Deep Convolutional GAN

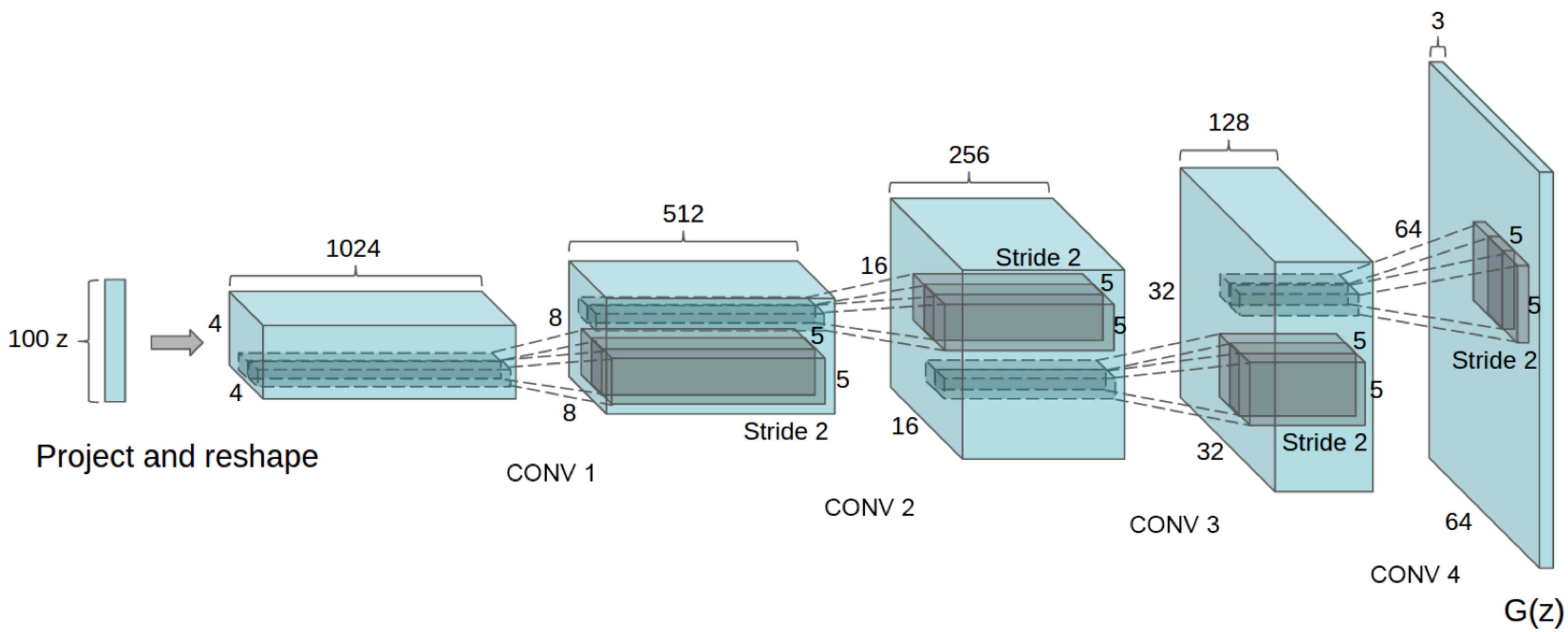


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)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

Radford, A., Metz, L., & Chintala, S. (2015). [Unsupervised representation learning with deep convolutional generative adversarial networks](). arXiv preprint arXiv:1511.06434.