Popular Machine Learning Methods: Idea, Practice and Math

Deep Generative Models

Yuxiao Huang

Data Science, Columbian College of Arts & Sciences
George Washington University

Spring 2023

Reference

- This set of slices was largely built on the following 7 wonderful books and a wide range of fabulous papers:

HML	Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition)
PML	Python Machine Learning (3rd Edition)
ESL	The Elements of Statistical Learning (2nd Edition)
PRML	Pattern Recognition and Machine Learning
NND	Neural Network Design (2nd Edition)
LFD	Learning From Data
RL	Reinforcement Learning: An Introduction (2nd Edition)

- For most materials covered in the slides, we will specify their corresponding books and papers for further reference.

Code Example

- See related code example in github repository: /p3_c3_s1_deep_generative_models/code_example

Table of Contents

Learning Objectives: Expectation

- It is **expected** to understand
  - the architecture and idea of Autoencoder (AE)
  - the architecture and idea of Variational Autoencoder (VAE)
  - the architecture and idea of Generative Adversarial Networks (GANs)

Learning Objectives: Recommendation

- It is **recommended** to understand
  - the two important extensions of GANs:
    - Wasserstein GAN (WGAN)
    - WGANs with Gradient Penalization (WGAN-GP)

Deep Generative Models

- *Deep Generative Models* are deep neural networks that can learn the latent representation of the input data, without any supervision (hence they belong to unsupervised learning).
- A key application of deep generative models is using the learned latent representation of the input data to generate new data (hence the name of these models).
- Below are the three most popular deep generative modes:
  - Autoencoder (AE)
  - Variational Autoencoder (VAE)
  - Generative Adversarial Networks (GANs)
- Here we will briefly discuss AE and VAE and focus on GANs, which have received the most attention among the three.
- See a more detailed discussion of AE and VAE in HML: Chap 17.

The Architecture of Autoencoder

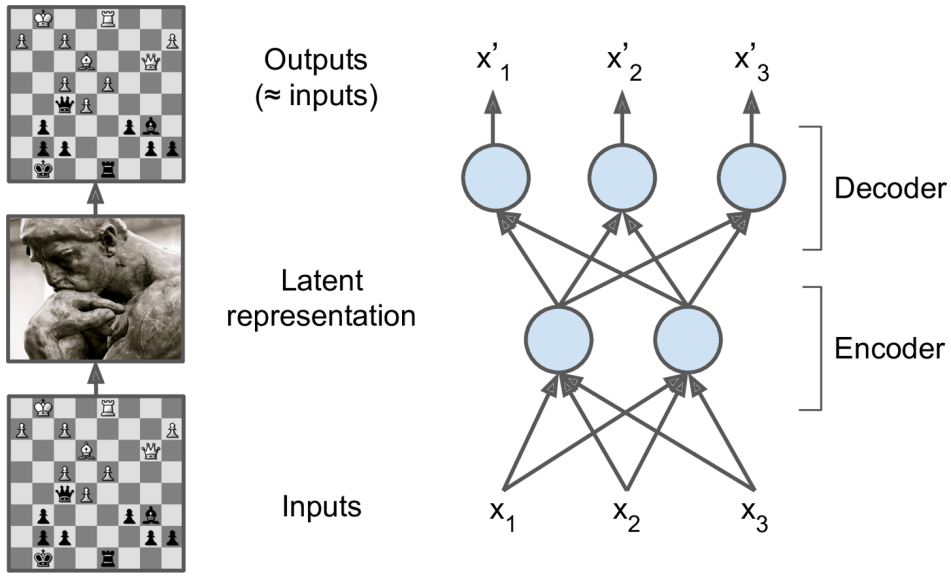

Figure 1: The architecture of AE. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- As shown in fig. 1, an AE has two parts:
  - an *Encoder* (a.k.a., *Recognition Network*) that transforms the input into its latent representation
  - an *Decoder* (a.k.a., *Generative Network*) that transforms the latent representation back to the input

The Architecture of Autoencoder

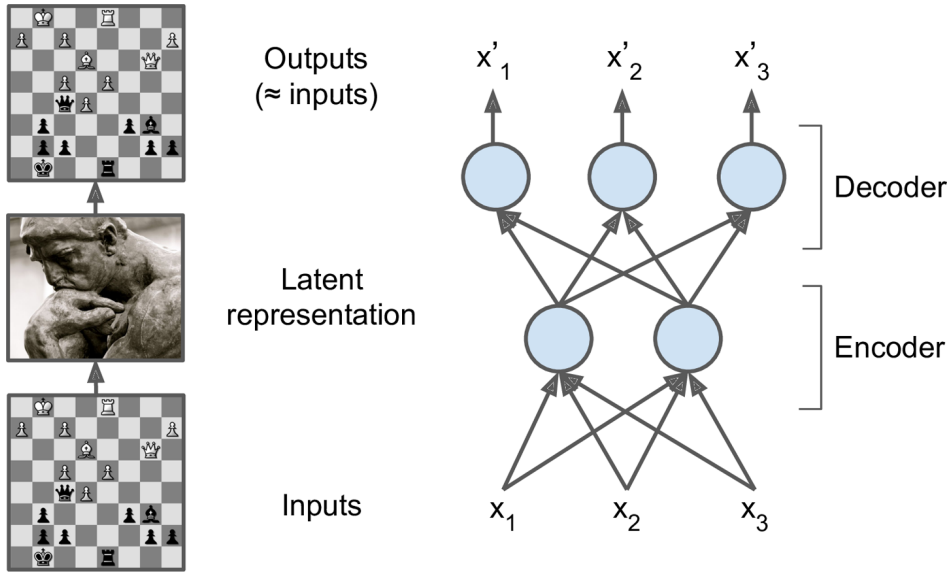

Figure 1: The architecture of AE. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition)*.

- AE is essentially a Multi-Layer Perceptron (see /p2_c2_s4_shallow_neural_networks):
  - the encoder is the hidden layer
  - the decoder is the output layer, where the number of perceptrons is the same as the number on the input layer (so as to reconstruct the input)
- AE reduces to Principal Component Analysis when:
  - the activations are identity function
  - the loss function is Mean Squared Error

The Architecture of Stacked Autoencoder

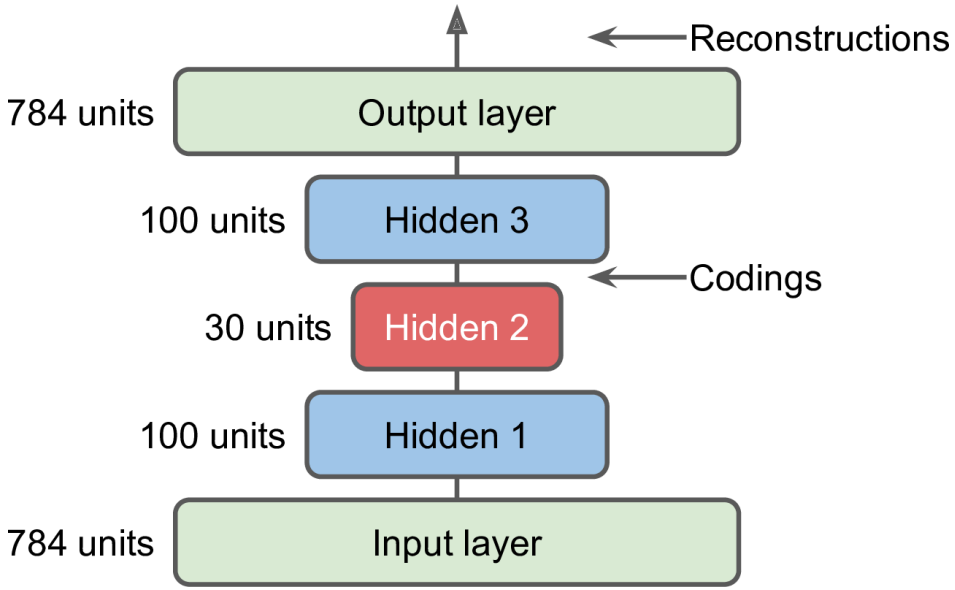

Figure 2: The architecture of stacked AE. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- Similar to Multi-Layer Perceptron, we can add more hidden layers to AE to make it deep.
- For an AE with multiple hidden layers, it is called *Stacked Autoencoder* (Stacked AE).
- As shown in fig. 2, the architecture of stacked AE is typically symmetrical with regard to the central hidden layer (the coding layer).

Transfer Learning with Stacked AE

- A key application of stacked AE is transfer learning.
- The idea is that, for a large training set where most of the data are unlabeled, we can:
  1. train a stacked AE on the whole (labeled and unlabeled) training data excluding the target (hence unsupervised learning)
  2. reuse the lower layers of the stacked AE as the base, and build a new model by adding layers on top of the base
  3. train the new model on the labeled training data (hence supervised learning):
     1. first freeze (some or all) reused layers and only train the remaining layers
     2. unfreeze the frozen layers and train the whole model

Good practice

- The steps for using stacked AE for transfer learning:

1. train a stacked AE on the whole (labeled and unlabeled) training data excluding the target (hence unsupervised learning)
2. reuse the lower layers of the stacked AE as the base, and build a new model by adding layers on top of the base
3. train the new model on the labeled training data (hence supervised learning):
   1. first freeze (some or all) reused layers and only train the remaining layers
   2. unfreeze the frozen layers and train the whole model

The Architecture of Variational Autoencoder

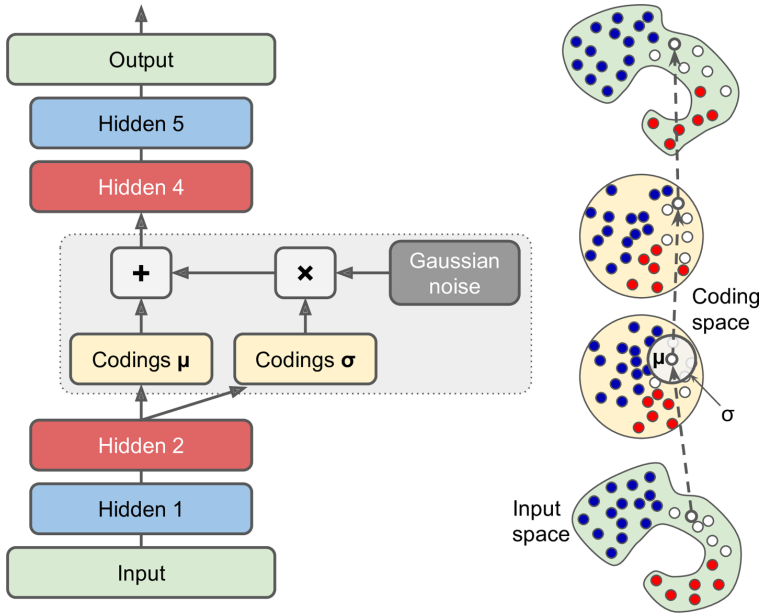

Figure 3: The architecture of variational AE. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition)*.

- A key extension of AE is *Variational Autoencoder* (VAE).
- The major differences between the architecture of AE and VAE are:
  - the encoder in AE models the latent representation of the input
  - the encoder in VAE models the latent representation of the mean (μ) and standard deviation (σ) of a Gaussian distribution, which is the distribubtion of the latent representation of the input

The Architecture of Variational Autoencoder

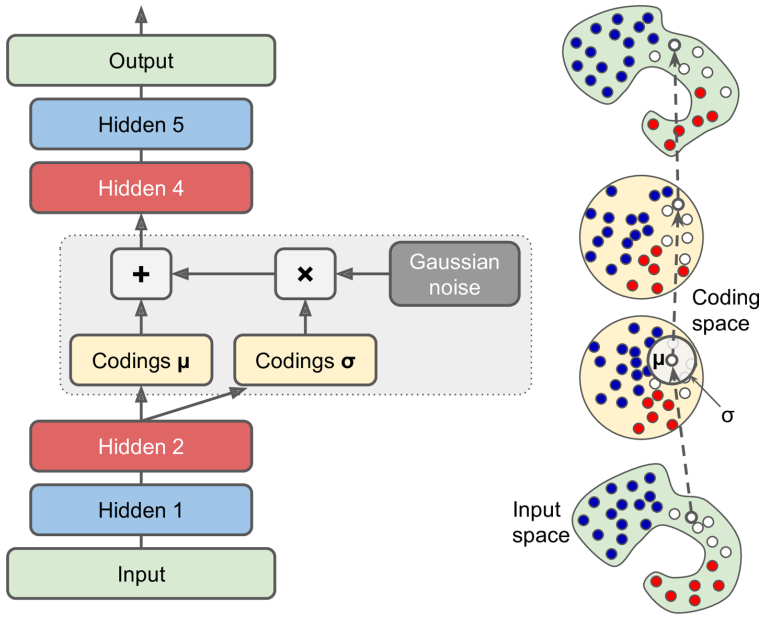

Figure 3: The architecture of variational AE. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition)*.

- A key merit of VAE is that, after training we can generate new input by:
  - sample the latent representation of the input from the Gaussian distribution
  - decode the latent representation to generate new input
- This also leads to another difference between AE and VAE:
  - the output of AE is deterministic (as there is no randomness in AE's forward pass)
  - the output of VAE is stochastic (as the latent representation of the input is sampled from a gaussian distribution)

The Idea

- Generative Adversarial Networks (GANs) [1] have received great attention for their good performance in generating artificial data resembling the real one.
- There are two components in GANs:
  - a *Generator* that tries to generate data similar to the real one
  - a *Discriminator* that tries to discriminate between the real data and the generated one

The Idea

- The idea of training GANs is based on an adversarial game between the two components:
  - the generator gets better (by deceiving the discriminator) only when the discriminator gets better (by debunking the generator)
  - the discriminator gets better (by debunking the generator) only when the generator gets better (by deceiving the discriminator)
- In theory, as training advances GANs eventually reach a *Nash Equilibrium* where:
  - the generator generates data that perfectly mimic the real one
  - the discriminator discriminates the two kinds of data with accuracy 0.5

The Adversarial Game

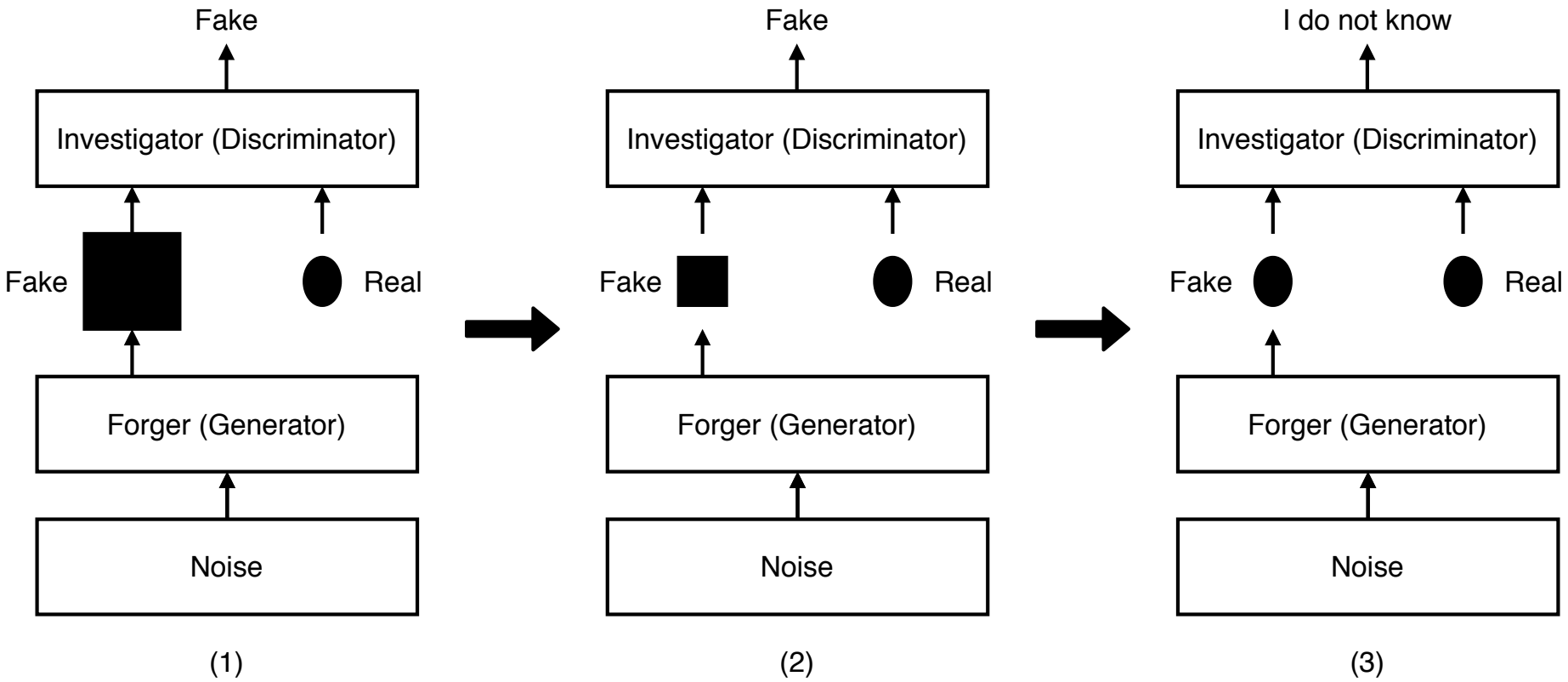

Figure 4: An example of the adversarial game between the generator and discriminator.

The Adversarial Game

- Figure. 4 shows an example of the adversarial game between the generator and discriminator.
- In this example, we use an adversarial game between a *Forger* and an *Investigator* to mimic the game between the generator and discriminator in GANs:
  - a forger (generator) attempts to make fake pearls and sell them at a high price
  - however, before selling them, the pearls have to be run by an investigator (discriminator), who decides whether the pearls are real
- Both the forger and investigator are in the early stage of their career, that is, neither of them is good at what they do:
  - the forger makes big square-shaped pearls
  - the investigator has no idea how to distinguish real pearls from fake ones
- The good news is, both of them are eager to learn.

The Adversarial Game

1. The investigator learns by comparing the fake pearls with the real ones.
2. The first thing caught the investigator's eye is the size: real ones are small but fake ones are big.
3. This insight was later leaked to the forger, he then starts making small square-shaped pearls to deceive the investigator.
4. This time, by comparing the fake pearls with the real ones, the investigator finds that he can also use the shape to separate the two kinds of pears: real ones are oval but fakes ones are square.
5. This insight was again leaked to the forger, he then starts making small oval-shaped pears to deceive the investigator.
6. By repeating this process, the investigator learns the differences between the real and fake pearls and the generator uses such information to narrow the differences.
7. Eventually, the fake pearls are so perfect such that the discriminator cannot tell any difference anymore.

The Loss Function

- The loss function of GANs, $V(\theta^{(D)}, \theta^{(G)})$, is

$$E_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log\left(1 - D\big(G(\mathbf{z})\big)\right)\right], \quad \text{where} \tag{1}$$

  - D is the discriminator
  - G is the generator
  - $\mathbf{x}$ is the real data
  - $\mathbf{z}$ is the latent vector and $G(\mathbf{z})$ the fake (generated) data
  - $D(\mathbf{x})$ is the probability of D correctly predicting $\mathbf{x}$ as real
  - $D\big(G(\mathbf{z})\big)$ is the probability of D wrongly predicting $G(\mathbf{z})$ as real
  - $E_{\mathbf{x}\sim p_{data}(\mathbf{x})}$ is the expectation with respect to the distribution of $\mathbf{x}$
  - $E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}$ is the expectation with respect to the distribution of $\mathbf{z}$
- In other words, the loss function is a sum of two expectations:
  - the first expectation, $E_{\mathbf{x}\sim p_{data}(\mathbf{x})}$, is the average of the probability of the discriminator correctly predicting real data ($\mathbf{x}$ in eq. (1)) as real
  - the second expectation, $E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}$, is the average of the probability of the discriminator correctly predicting fake data ($G(\mathbf{z})$ in eq. (1)) as fake
- It is worth noting that while the discriminator is related to both expectations (to predict the data as real or fake), the generator is only related to the second expectation (to generate the fake data).

The Optimization

- The optimization of GANs includes the following two steps (in order):
  1. maximize the loss function, $V(\theta^{(D)}, \theta^{(G)})$, with respect to the discriminator, D
  2. minimize the loss function, $V(\theta^{(D)}, \theta^{(G)})$, with respect to the generator, G
- The two steps above can be written as

$$\min_G \max_D V(\theta^{(D)}, \theta^{(G)}). \tag{2}$$

- At a glance, it is a bit strange that we optimize the loss using two completely opposite ways:
  - maximize it with respect to the discriminator
  - minimize it with respect to the generator
- Here is why the optimization makes sense.

Maximization regarding the Discriminator

- Eq. (1) shows the loss function of GANs, $V(\theta^{(D)}, \theta^{(G)})$:

$$E_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log\left(1 - D\big(G(\mathbf{z})\big)\right)\right]. \tag{1}$$

- In the first expectation of the loss, $D(\mathbf{x})$ is the probability of the discriminator correctly predicting the real data, $\mathbf{x}$, as real.
- The discriminator wants this probability as close to 1 (the maximum value of a probability) as possible, as 1 means the discriminator is certain that the real data is real.
- Since $\log D(\mathbf{x})$ is strictly increasing (i.e., the higher $D(\mathbf{x})$ the higher $\log D(\mathbf{x})$), making $D(\mathbf{x})$ as large as possible equates making $\log D(\mathbf{x})$ as large as possible, hence maximizing the first expectation.

Maximization regarding the Discriminator

- Eq. (1) shows the loss function of GANs, $V(\theta^{(D)}, \theta^{(G)})$:

$$E_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log\left(1 - D\big(G(\mathbf{z})\big)\right)\right]. \tag{1}$$

- In the second expectation of the loss, $D\big(G(\mathbf{z})\big)$ is the probability of the discriminator wrongly predicting the fake data, $G(\mathbf{z})$, as real.
- The discriminator wants this probability as close to 0 (the minimum value of a probability) as possible, as 0 means the discriminator is certain that the fake data is not real.
- Since $\log\left(1 - D\big(G(\mathbf{z})\big)\right.$ is strictly increasing (i.e., the higher $1 - D\big(G(\mathbf{z})$ the higher $\log\left(1 - D\big(G(\mathbf{z})\big)\right)$, making $D\big(G(\mathbf{z})\big)$ as small as possible equates making $1 - D\big(G(\mathbf{z})$ as large as possible, which in turn, equates making $\log\left(1 - D\big(G(\mathbf{z})\big)\right.$ as large as possible, hence maximizing the second expectation.

Maximization regarding the Discriminator

- To sum up, since we want to maximize both the first and the second expectation in the loss function, we want to maximize the whole loss function with respect to the discriminator.
- This is why we have the maximization in eq. (2)

$$\min_{G} \max_{D} V(\theta^{(D)}, \theta^{(G)}). \tag{2}$$

Minimization regarding the Generator

- Eq. (1) shows the loss function of GANs, $V(\theta^{(D)}, \theta^{(G)})$:

$$E_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log\left(1 - D\big(G(\mathbf{z})\big)\right)\right]. \tag{1}$$

- Since only the second expectation in the loss is related to the generator, we only need to discuss why we should minimize the second expectation with respect to the generator.
- As mentioned earlier, in the second expectation $D\big(G(\mathbf{z})\big)$ is the probability of the discriminator wrongly predicting the fake data, $G(\mathbf{z})$, as real.
- The generator wants this probability as close to 1 as possible, as 1 means that the discriminator is certain that the fake data is real.
- Since $\log\left(1 - D\big(G(\mathbf{z})\big)\right)$ is strictly increasing, making $D\big(G(\mathbf{z})\big)$ as large as possible equates making $1 - D\big(G(\mathbf{z})\big)$ as small as possible, hence minimizing the second expectation.
- This is why we have the minimization in eq. (2)

$$\min_{G}\max_{D} V(\theta^{(D)}, \theta^{(G)}). \tag{2}$$

Training GANs

- A practical way of training GANs is to alternate between the two optimization steps (in order):
  1. freeze the parameters of the generator and update the parameters of the discriminator
  2. freeze the parameters of the discriminator and update the parameters of the generator

Training GANs

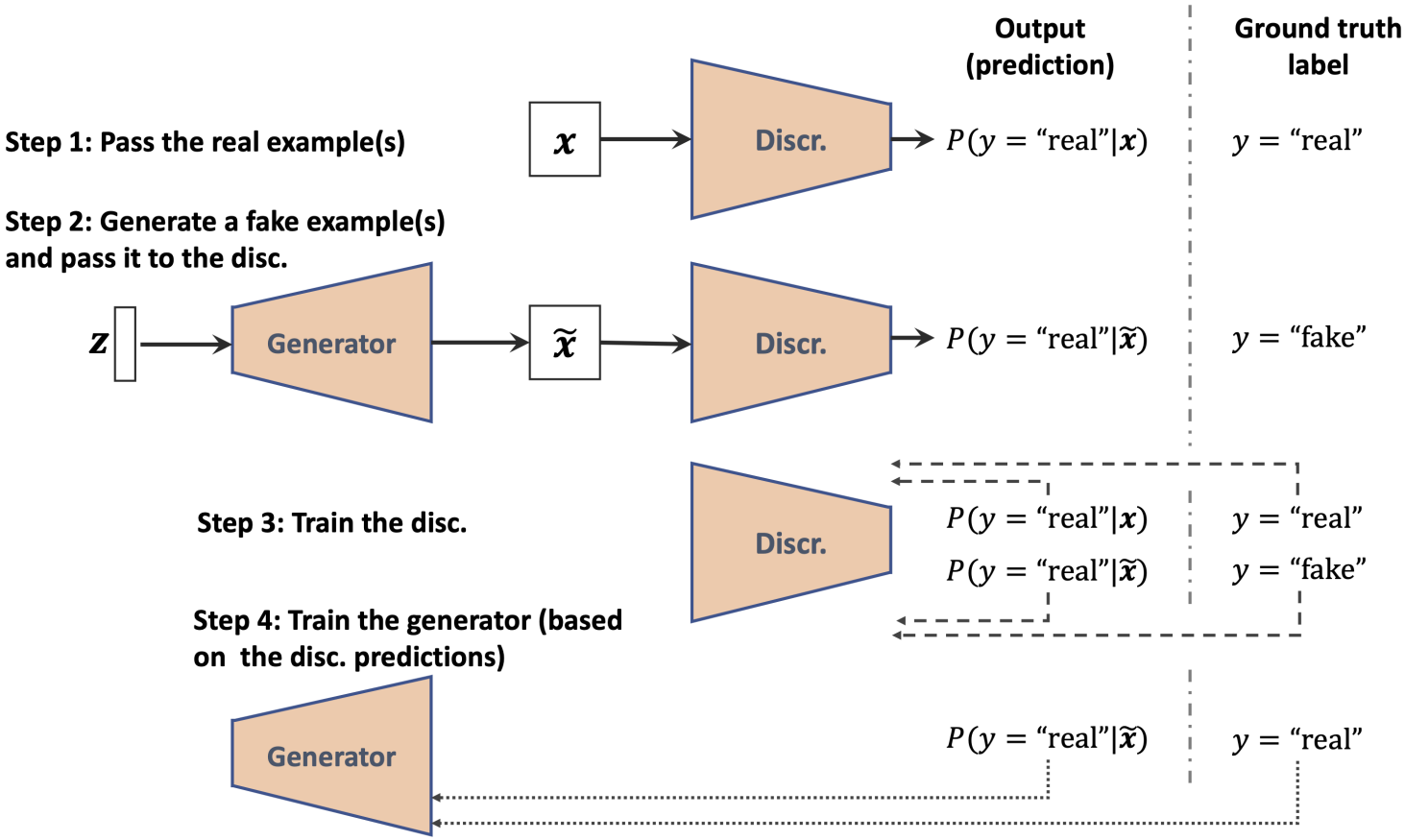

Figure 5: One iteration of training GANs. Picture courtesy of *Python Machine Learning (3rd Edition)*.

Deep Convolutional GANs

- GANs based on deep CNNs have been proposed to generate large images.
- One popular model is Deep Convolutional GANs (DCGANs) [2].

Good practice

- Follow the following good practices to build stable DCGANs:
  - replace any pooling layer with strided convolutions (in the discriminator) and transposed convolutions (in the generator)
  - use *Batch Normalization* in both the generator and the discriminator, except in the generator's output layer and discriminator's input layer
  - remove fully connected hidden layers
  - use `ReLU` activation in the generator for all layers except the output layer, which should use tanh
  - use `leaky ReLU` activation in the discriminator for all layers

Further Reading

- See a detailed discussion of GAN in a NeurIPS 2016 tutorial: https://arxiv.org/pdf/1701.00160.pdf
- See an important improvement of GAN, Wasserstein GAN (WGAN), in: https://arxiv.org/pdf/1701.07875.pdf
- See an important improvement of WGAN in: https://arxiv.org/pdf/1704.00028.pdf

Bibliography I

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio.

Generative Adversarial Nets.

In *NeurIPS*, pages 2672–2680, 2014.

A. Radford, L. Metz, and S. Chintala.

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.

arXiv preprint arXiv:1511.06434, 2015.