Machine Learning: Algorithms and Applications

Philip O. Ogunbọna

Advanced Multimedia Research Lab
University of Wollongong

Artificial Neural Networks and Deep Learning: An Introduction
Autoencoders and GAN
Autumn

Outline

Autoencoders

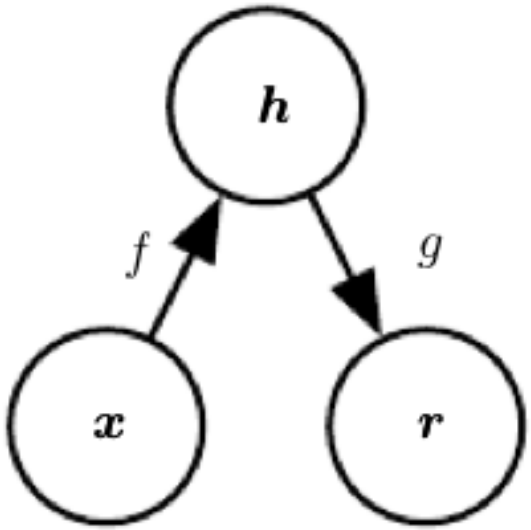

Figure 1: General structure of an autoencoder; input $\boldsymbol{x}$ maps to an output $\boldsymbol{r}$ (reconstruction) through internal representation or code $\boldsymbol{h}$ (Goodfellow et al. 2016)

- Conceptually an autoencoder is a feedforward network trained to copy its input to its output (albeit imperfectly)
- Structure (see Figure 1) has a hidden layer $\boldsymbol{h}$ describing the code representing the input
- Autoencoder has two parts: encoder function $\boldsymbol{h} = f(\boldsymbol{x})$ that generates the representative code of the input and decoder function $\boldsymbol{r} = g(\boldsymbol{h})$ that produces a reconstruction from the code
- Generalization of autoencoder to stochastic mappings: $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ and $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$
- Typical training strategy is similar to that used for feedforward networks - minibatch gradient descent

Stacked autoencoder

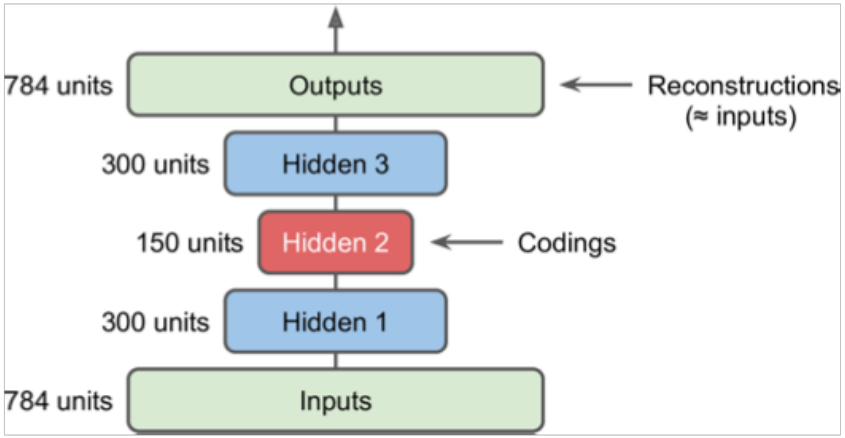

Figure 2: Example of stacked autoencoder used for the MNIST dataset; notice the 784 (28×28) input neurons; 300 hidden neurons; 150 central hidden neurons; a mirroring in the top layer (Géron 2019)

- Practical autoencoder is a stack
- Architecture of a stacked autoencoder is typically symmetrical with regards to the central hidden layer (the coding layer) (see Figure 2)

Undercomplete/Overcomplete autoencoders

- Constraining $\boldsymbol{h}$ to have smaller dimension than $\boldsymbol{x}$ results in an undercomplete autoencoder
- $\boldsymbol{h}$ captures the most salient features of input
- Learning entails minimizing a loss function

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) \tag{1}$$

  L penalizes $g(f(\boldsymbol{x}))$ being dissimilar to $\boldsymbol{x}$

- If dimension of code is greater than that of input we have overcomplete autoencoder
- Any architecture of autoencoder can be trained without the risk of over-capacity or learning a trivial identity, by using regularization
- Regularization can inpart properties to loss function:
  - sparsity of representation
  - smallness of derivative of representation
  - robustness to noise
  - robustness to missing data

Autoencoders and Principal Component Analysis (PCA)

- With a linear decoder $g(\boldsymbol{h})$ and mean squared error loss, an undercomplete autoencoder learns the same subspace as PCA
- With a nonlinear encoder and decoder (respectively, $f(\boldsymbol{x})$ and $g(\boldsymbol{h})$), an autoencoder can learn a more powerful generalization of PCA

Sparse autoencoders

- Sparse autoencoder has cost function used for training in the form of reconstruction error and sparsity penalty on the code layer $\boldsymbol{h}$:

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}) \tag{2}$$

  where $\boldsymbol{h}$ is the encoder output; $\boldsymbol{h} = f(\boldsymbol{x})$ typically (see Figure 1)

- Sparse autoencoders are useful in learning features that can be input for other tasks, e.g. classification (think about semi-supervised classification)

- Sparse autoencoders can be interpreted as approximating maximum likelihood training of generative model that has latent variables (in this case $\boldsymbol{h}$)

- In this respect, it is maximizing

$$\log p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x}) = \log p_{\text{model}}(\boldsymbol{h}) + \log p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h}) \tag{3}$$

  $\log p_{\text{model}}(\boldsymbol{h})$ can be sparsity-inducing

Denoising autoencoders

- Denoising aims to reduce the noise in signals
- Denoising autoencoders minimize

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))) \tag{4}$$

  where $\tilde{\boldsymbol{x}}$ is a copy of $\boldsymbol{x}$ corrupted by some form of noise
- Training process forces f and g to implictly learn the structure of $p_{\text{data}}(\boldsymbol{x})$
- Another form of regularization $\lambda \sum_i ||\nabla_x h_i||^2$ forces the learning of a function that does not change much when $\boldsymbol{x}$ changes slightly:

$$L(\boldsymbol{x}, f(g(\boldsymbol{x}))) + \lambda \sum_i ||\nabla_x h_i||^2 \tag{5}$$

Denoising autoencoders

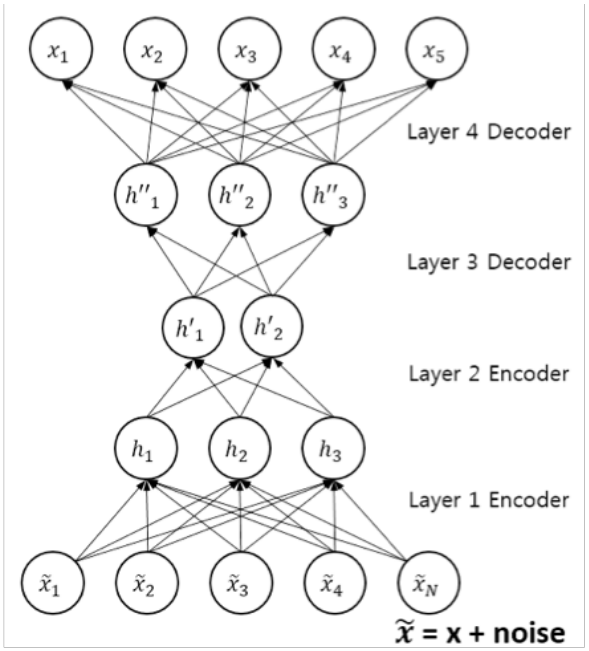

Figure 3: Stacked convolutional denoising autoencoder

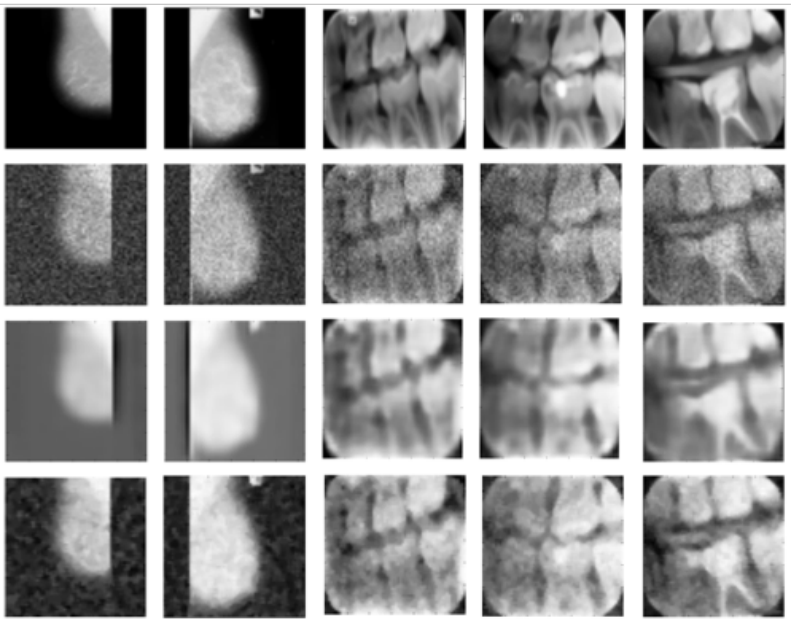

Figure 4: Comparison between output of stacked convolutional denoising autoencoder and median filter; Gaussian noise: $\mu = 0$, $\sigma = 1$

More autoencoders - cost functions

Contractive autoencoder

- Regularization is introduced on the code $h = f(\boldsymbol{x})$ to encourage derivatives of f to be as small as possible

$$\Omega(\boldsymbol{h}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2$$

- Contractive autoencoder and denoising autoencoder are related when input noise is small and Gaussian (Goodfellow et al. (2016))
  - denoising autoencoders make the reconstruction function resist small but finite-sized perturbations of the input;
  - contractive autoencoders make the feature extraction function resist infinitesimal perturbations of the input

Variational autoencoder

Variational autoencoder (Chollet 2021)

1. An encoder module turns the input sample, (e.g.input-img,) into two parameters in a latent space of representations, `z_mean` and `z_log_variance`.
2. You randomly sample a point z from the latent normal distribution that is assumed to generate the input image, via

$$\text{z} = \text{z\_mean} + \exp(\text{z\_log\_variance}) \times \text{epsilon}$$

   where `epsilon` is a random tensor of small values.
3. A decoder module maps this point in the latent space back to the original input image.

Variational autoencoder

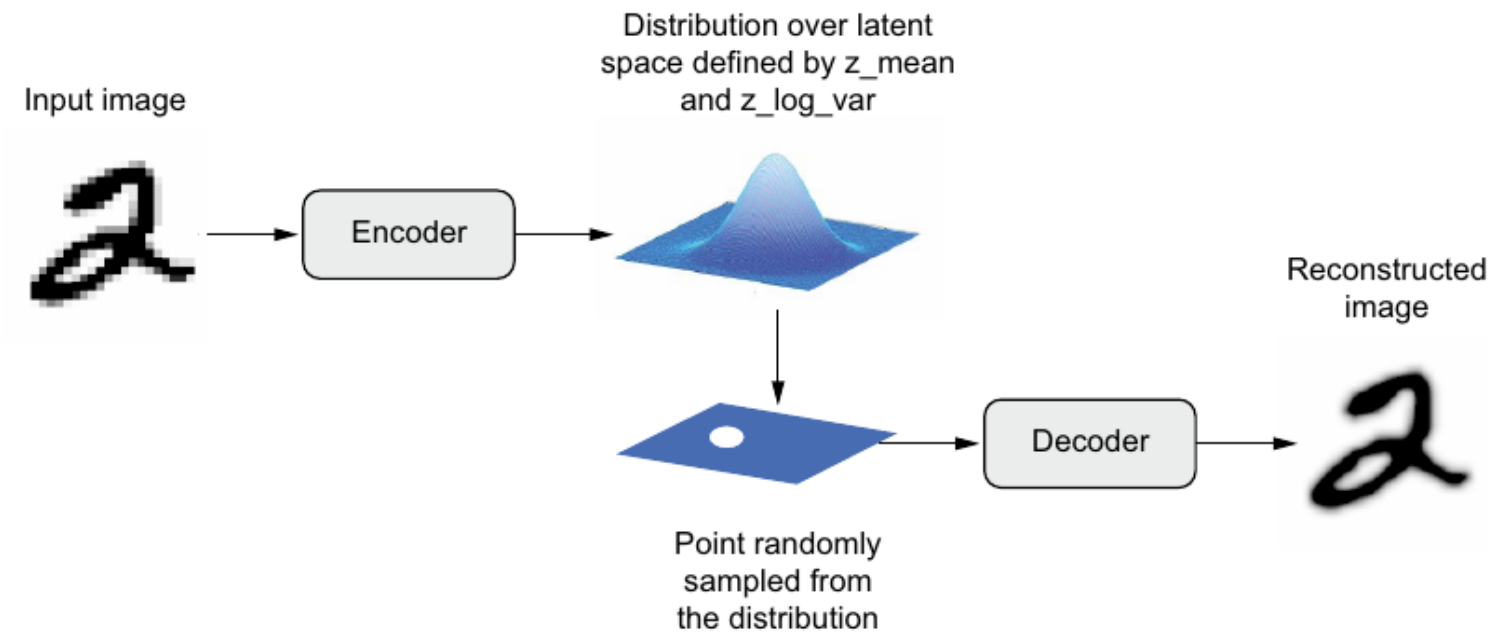

Figure 5: VAE maps an image to two vectors, z_mean and z_log_sigma, which define a probability distribution over the latent space, used to sample a point to decode (Chollet 2021)

- See accompanying jupyter notebook for a demonstration of VAE (Chollet 2021). The notebook from Géron (2019) is also a good source of demonstration.

Generative adversarial networks (GAN)

- Central problem addressed by GAN is density estimation; GAN implicitly captures the underlying data distribution
- GAN can be used in both unsupervised and semisupervised learning settings
- Characterised by training two networks in competition:
  - There is a network, named the generator ($\mathcal{G}$), trying to produce samples from a distribution that is learned from given data - mimicking, forging, synthetic data
  - There is a second network, the discriminator ($\mathcal{D}$), that is able to tell the synthetic samples from the real ones
- Objective is to be able to generate synthetic signals that are no different from the real ones

Generative adversarial networks (GAN)

Simple illustration of GAN operation

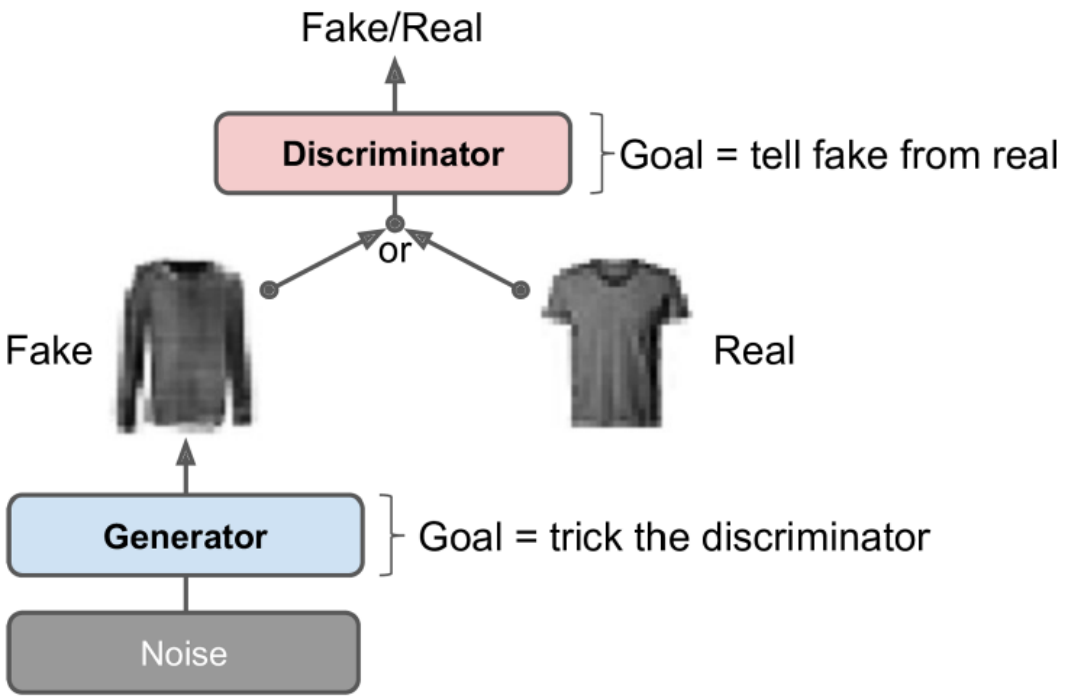

Figure 6: Simple illustration of GAN (Géron 2019)

Generative adversarial networks (GAN)

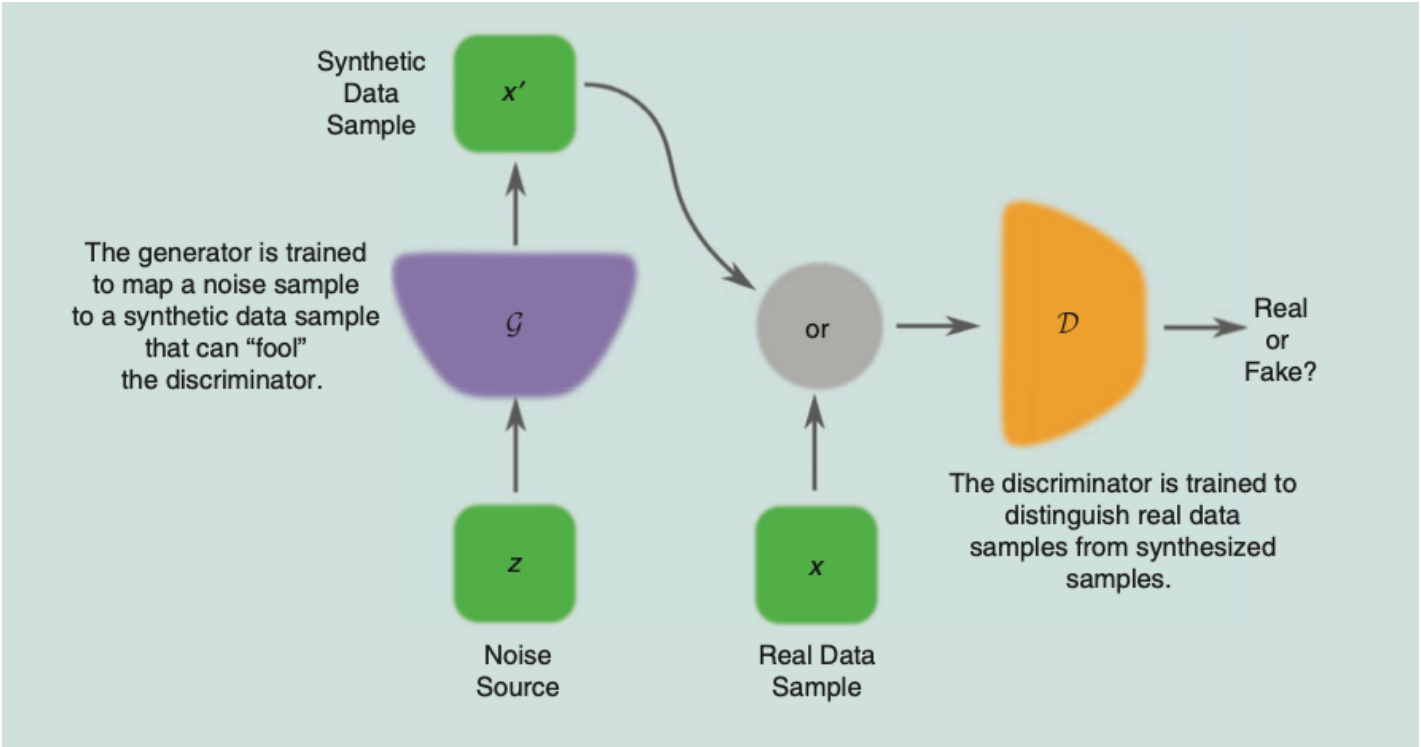

Figure 7: Two models are learned while training GAN; Discriminator ($\mathcal{D}$) and Generator ($\mathcal{G}$); models implemented using neural network, but any differentiable system (mapping) can also be used (Creswell et al. 2018)

Generative adversarial networks (GAN)

- In Figure 7, Generator network has no access to the real samples
- Generator network is a mapping from some representation space (latent space) to the data sample space:

$$\mathcal{G} : \mathcal{G}(\boldsymbol{z}) \rightarrow \mathcal{R}^{|\boldsymbol{x}|}$$

where $\boldsymbol{z} \in \mathcal{R}^{|\boldsymbol{x}|}$ is the data sample and $|\cdot|$ denotes the number of dimensions

- Discriminator network, $\mathcal{D}$, maps data sample to a probability that sample is from real data distribution and not generator distribution

$$\mathcal{D} : \mathcal{D}(\boldsymbol{x}) \rightarrow (0, 1)$$

- $p_{\text{data}}(\boldsymbol{x})$ represents the probability density function over the data samples (in $\mathcal{R}^{|\boldsymbol{x}|}$) and $p_g(\boldsymbol{x})$ distribution of the samples produced by the generator

Generative adversarial networks (GAN)

- During training we set objective functions for the generator ($J_{\mathcal{G}}(\Theta_{\mathcal{G}}; \Theta_{\mathcal{D}})$) and discriminator ($J_{\mathcal{D}}(\Theta_{\mathcal{D}}; \Theta_{\mathcal{G}})$)
- Note that $J_{\mathcal{G}}$ and $J_{\mathcal{D}}$ are co-dependent on the network parameters, $\Theta_{\mathcal{G}}$ and $\Theta_{\mathcal{D}}$ as the networks are iteratively trained

Generative adversarial networks (GAN)

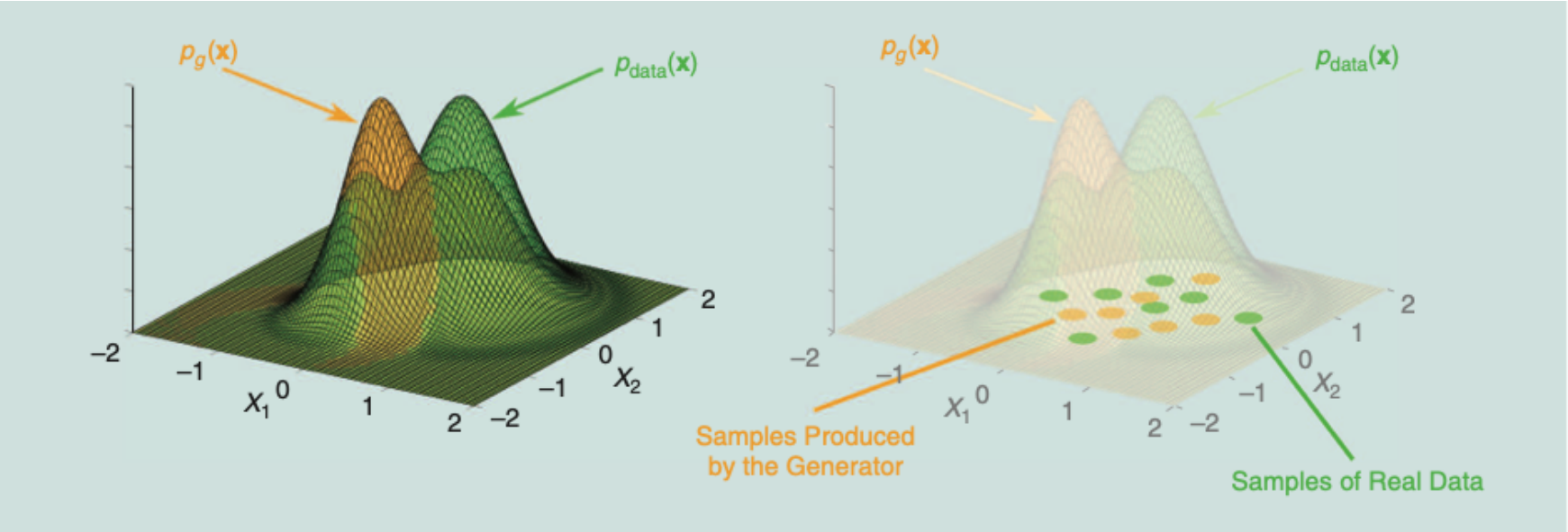

Figure 8: During GAN training, the generator is encouraged to produce a distribution of samples, $p_g(\boldsymbol{x})$ to match that of real data, $p_{\text{data}}(\boldsymbol{x})$ (Creswell et al. 2018)

Generative adversarial networks (GAN)

Training GAN

- We find parameters of a discriminator that maximize its classification accuracy and find the parameters of a generator that maximally confuses the discriminator
- Cost of training is evaluated using a value function; solve the following mini-max problem:

$$\max_{\mathcal{D}} \min_{\mathcal{G}} V(\mathcal{G}, \mathcal{D})$$

where

$$V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})} \log \mathcal{D}(\boldsymbol{x}) + \mathbb{E}_{p_g(\boldsymbol{x})} \log \left(1 - \mathcal{D}(\boldsymbol{x})\right)$$

- Parameters of one model are updated while the parameters of the other are fixed
- Optimal discriminator is unique (Goodfellow et al. 2014)

$$\mathcal{D}^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

- Generator is optimal when (Goodfellow et al. 2014)

$$p_g(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$$

- See accompanying jupyter notebook for demonstration (Géron 2019, Chollet 2021)

Generative adversarial networks (GAN)

Other GAN architectures

- Initial GAN architecture used fully connected neural network
- Difficult to train; successful only with a subset of datsets - stability issues
- Deep convolutional GAN provided more stability;
- Conditional GAN - both the generator and the discriminator networks are class-conditional (Figure 9)
- Conditional GANs can provide better representations for multimodal data generation
- InfoGAN decomposes the noise source into an incompressible source and a “latent code,”; attempt to discover latent factors of variation by maximizing the mutual information between the latent code and the generator’s output (Figure 10)

Generative adversarial networks (GAN)

Other GAN architectures

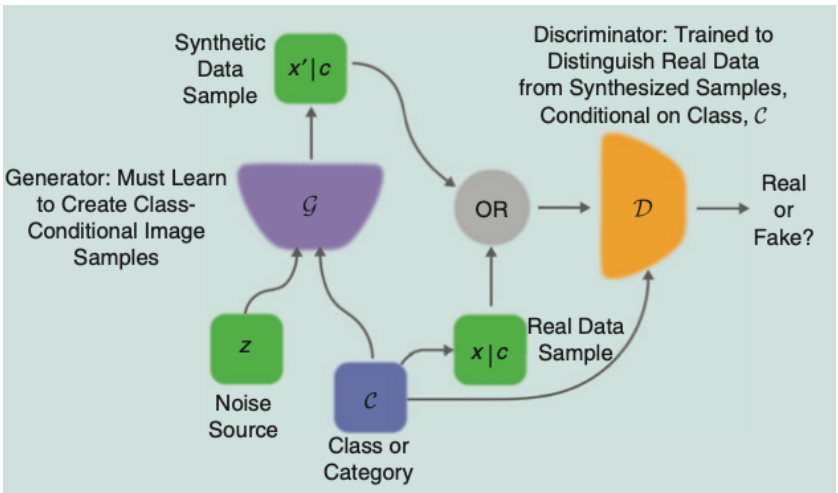

Figure 9: Conditional GAN

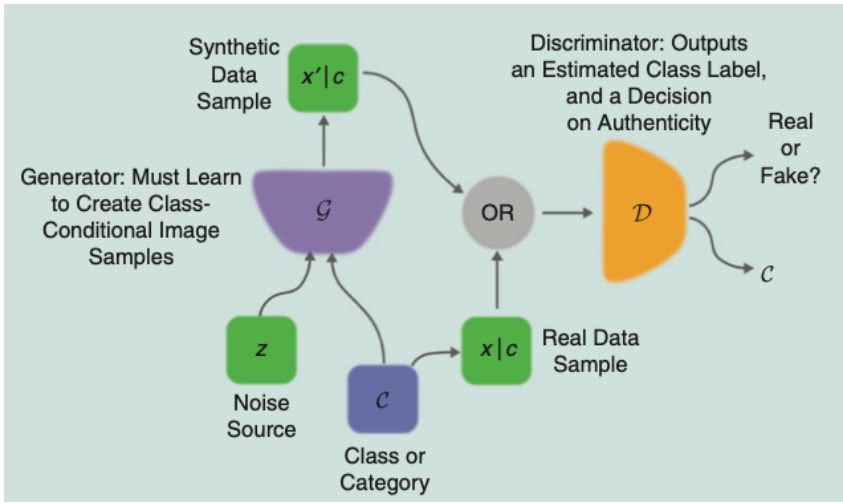

Figure 10: InfoGAN

Bibliography

Chollet, F. (2021), *Deep Learning with Python*, 2nd edn, Manning Publishing Co. Ltd., Shelter Island, NY, USA.

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B. & Bharath, A. A. (2018), 'Generative adversarial networks: An overview', *IEEE Signal Processing Magazine* **35**(1), 53 – 65.

Géron, A. (2019), *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media, Inc., Boston, USA.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014), Generative adversarial nets, *in* 'Proc. Advances Neural Information Processing Systems Conf', Montreal, Quebec, Canada, p. 2672–2680.