Generative Adversarial Network

Deep Learning (ENGG*6600*01)

School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh

Summer 2023

Introduction

Introduction

- Suppose we have a **generative model** which takes a **random noise as input** and **generates a data point**.
- We want the generated data point to be of good quality; hence, we should somehow **judge its quality**.
- One way to judge it is to observe the generated sample and assess its quality visually. In this case, the **judge is a human**. However, we **cannot take derivative of human's judgment** for optimization.
- **Generative Adversarial Network (GAN)**, proposed in (2014) [1], has the same idea but it can take derivative of the judgment. For that, it uses a **classifier as the judge** rather than a human. Hence, we have a generator generating a sample and a **binary classifier (or discriminator)** to classify the generated sample as a real or generated sample.
- This classifier can be a **pre-trained network** which is already trained by some real and generated (fake) data points. However, GAN puts a step ahead and lets the **classifier be trained simultaneously** with training the generator. This is the core idea of adversarial learning where the classifier, also called the discriminator, and the generator compete with one another; hence, they **make each other stronger gradually by this competition** [2].

Introduction

- It is noteworthy that the term "**adversarial**" is used in two main streams of research in machine learning and they should not be confused. These two research areas are:
  - **Adversarial attack**, also called learning with adversarial examples or adversarial machine learning. This line of research inspects some examples which can be changed slightly but wisely to fool a trained learning model. For example, perturbation of some specific pixels in the input image may change the decision of learning model. The reason for this can be analyzed theoretically. Some example works in this area are [3, 4, 5, 6, 7].
  - **Adversarial learning** for generation. This line of research is categorized as generative models [8] and/or methods based on that. GAN is in this line of research. This paper focuses on this research area.

Adversarial Learning: The Adversarial Game

Adversarial Learning: The Adversarial Game

- The **original GAN**, also called the **vanilla GAN**, was proposed in (2014) [1].
- Consider a d-dimensional dataset with n data points, i.e., $\{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$. In GAN, we have a **generator** G which takes a p-dimensional random noise $\boldsymbol{z} \in \mathbb{R}^p$ as input and outputs a d-dimensional generated point $\boldsymbol{x} \in \mathbb{R}^d$. Hence, it is the mapping $G : \boldsymbol{z} \to \boldsymbol{x}$ where:

$$G(\boldsymbol{z}) = \boldsymbol{x}. \tag{1}$$

- Let the distribution of random noise be denoted by $\boldsymbol{z} \sim p_z(\boldsymbol{z})$. We want the generated $\widehat{\boldsymbol{x}}$ to be very similar to some original (or real) data point $\boldsymbol{x}$ in the dataset. We need a module to judge the quality of the generated point to see how similar it is to the real point. This module can be a human but we cannot take derivative of human's judgment for optimization! A good candidate for the judge is a **classifier**, also called the **discriminator**. The discriminator (also called the **critic**), denoted by $D : \boldsymbol{x} \to [0, 1]$, is a **binary classifier** which classifies the generated point as a real or generated point:

$$D(\boldsymbol{x}) := \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ is real,} \\ 0 & \text{if } \boldsymbol{x} \text{ is generated (fake).} \end{cases} \tag{2}$$

- The **perfect** discriminator outputs **one for real points** and **zero for generated points**.
- The discriminator's output is in the **range** $[0, 1]$ where the output for real data is closer to one and the output for fake data is closer to zero.
- If the generated point is very good and closely similar to a real data point, the classifier may make a mistake and output a value close to one for it. Therefore, if the classifier makes a mistake for the generated point, the generator has done a good job in generating a data point.

Adversarial Learning: The Adversarial Game

- The discriminator can be a **pre-trained classifier**, but we can make the problem more sophisticated, as explained in the following.
- Let us **train the discriminator simultaneously while we are training the generator**. This makes the discriminator D and the generator G stronger gradually while they compete each other.
- On the one hand, the **generator tries to generate realistic points to fool the discriminator** and make it a hard time to distinguish the generated point from a real point.
- On the other hand, the **discriminator tries to discriminate the fake (i.e., generated) point from a real point**.
- When one of them gets stronger in training, the other one tries to become stronger to be able to compete. Therefore, there is an **adversarial game** between the generator and the discriminator. This game is **zero-sum** because whatever one of them loses, the other wins.

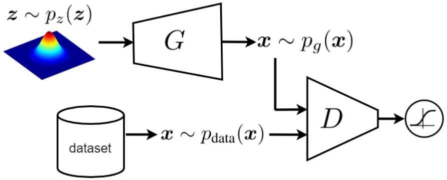
Optimization and Loss Function

Optimization and Loss Function

- We denote the probability distributions of dataset and noise by $p_{\text{data}}(\boldsymbol{x})$ and $p_z(\boldsymbol{z})$, respectively.
- As the figure shows, the **discriminator** is trained by real points from dataset as well as generated points from the generator.
- The discriminator and generator are **trained simultaneously**.
- The optimization **loss function** for both the discriminator and generator is:

$$\min_G \max_D \; V(D, G) := \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}\Big[\log\big(D(\boldsymbol{x})\big)\Big] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}\Big[\log\Big(1 - D\big(G(\boldsymbol{z})\big)\Big)\Big], \tag{3}$$

  where $\mathbb{E}[.]$ denotes the expectation operator and the loss function $V(D, G)$ is also called the **value function** of the game.


$\boldsymbol{z} \sim p_z(\boldsymbol{z})$
G
$\boldsymbol{x} \sim p_g(\boldsymbol{x})$
dataset
$\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$
D

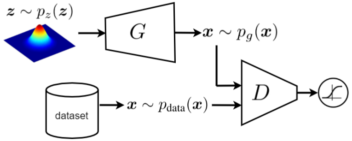
- In practice, we can use the Monte Carlo approximation [9] of expectation where the expectations are replaced with averages over the mini-batch.

Optimization and Loss Function

- We had:

$$\min_{G} \max_{D} \; V(D, G) := \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\Big[\log\big(D(\boldsymbol{x})\big)\Big] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\Big[\log\Big(1 - D\big(G(\boldsymbol{z})\big)\Big)\Big].$$

- The **first term** in Eq. (3) is expectation over the **real data**. This term is only used for the **discriminator** while it is a constant for the generator. According to Eq. (2), $D(\boldsymbol{x})$ outputs one (the larger label) for the real data; therefore, the **discriminator maximizes this term** because it assigns the **larger label to the real data**.
- The **second term** in Eq. (3) is expectation **over noise**. It inputs the noise $\boldsymbol{z}$ to the generator to have $G(\boldsymbol{z})$. The output of generator, which is the generated point, is fed as input to the discriminator to have $D\big(G(\boldsymbol{z})\big)$. The **discriminator wants to minimize** $D\big(G(\boldsymbol{z})\big)$ because the **smaller label is assigned to the generated data**, according to Eq. (2). In other words, the **discriminator wants to maximize** $1 - D\big(G(\boldsymbol{z})\big)$.
- As **logarithm** is a monotonic function, we can say that the **discriminator** wants to **maximize** $\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$ which is the second term in Eq. (3). As opposed to the discriminator, the **generator minimizes** $\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$ which is the second term in Eq. (3). This is because the generator wants to **fool the discriminator** to label the generated data as real data.


$\boldsymbol{z} \sim p_z(\boldsymbol{z})$
G
$\boldsymbol{x} \sim p_g(\boldsymbol{x})$
dataset
$\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$
D

Optimization and Loss Function

- We had:

$$\min_{G} \max_{D} \; V(D, G) := \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}\Big[\log\big(D(\boldsymbol{x})\big)\Big] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}\Big[\log\Big(1 - D\big(G(\boldsymbol{z})\big)\Big)\Big].$$

- The Eq. (3) is a **minimax** optimization problem [10] and can be solved using **alternating optimization** [11] where we optimize over D and over G iteratively until convergence (i.e., Nash equilibrium).
- The original GAN [1] uses **a step of stochastic gradient descent** [11] for **updates of each variable** in the alternating optimization. If we denote the loss function in Eq. (3) by $V(D, G)$, the alternating optimization is done as:

$$D^{(k+1)} := D^{(k)} + \eta^{(k)} \frac{\partial}{\partial D}\Big(V(D, G^{(k)})\Big), \tag{4}$$

$$G^{(k+1)} := G^{(k)} - \eta^{(k)} \frac{\partial}{\partial G}\Big(V(D^{(k+1)}, G)\Big), \tag{5}$$

  where k is the index of iteration and $\eta^{(k)}$ is the learning rate at iteration k.
- Throughout this slide deck, **derivatives w.r.t.** D **and** G mean the derivatives w.r.t. the parameters (weights) of D and G networks, respectively.
- Eqs. (4) and (5) are one step of **gradient ascent** and **gradient descent**, respectively. Note that the gradients here are the average of gradients in the **mini-batch**. Every mini-batch includes both real and generated data.
- The paper [1] suggests that Eq. (4) can be **performed for several times** before performing Eq. (5); however, the experiments of that paper perform Eq. (4) for **only one time** before performing Eq. (5).

Optimization and Loss Function

- Also note that another way to solve the optimization problem in GAN is **simultaneous optimization** [12] in which Eqs. (4) and (5) are performed at the same time and not one after the other.
- **Minimax versus maximin in GAN** [13, Section 5]: We saw in Eq. (3) that the optimization of GAN is a **minimax** problem:

$$\min_{G} \max_{D} \quad V(D, G). \tag{6}$$

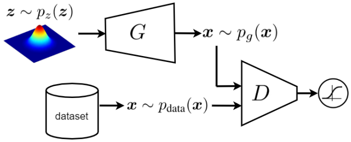
- By changing the order of optimization, one can see GAN as a **maximin** problem [13]:

$$\max_{D} \min_{G} \quad V(D, G). \tag{7}$$

- In fact, under some conditions, Eqs. (6) and (7) are equivalent [10].

Network Structure of GAN

- In practice, the discriminator and generator are two **(deep) neural networks**. The structure of GAN is depicted in this figure.
- The first layer of **discriminator network** is *d*-dimensional and its last layer is one dimensional with scalar output. In the original GAN, maxout activation function [14] is used for all layers except the last layer which has the sigmoid activation function to output a probability to model Eq. (2). The closer the output of *D* to one, the more probable its input is to be real.
- The **generator network** has a *p*-dimensional input layer for noise and a *d*-dimensional output layer for generating data. In the generator, a combination of ReLU [15] and sigmoid activation functions are used.
- The space of noise as the input to the generator is called the **latent space** or the **latent factor**.
- Each of the Eqs. (4) and (5) are performed using **backpropagation** in the neural networks.



Optimal Solution of GAN

Optimal Solution of GAN

- **Theorem**: [1, Proposition 1]: For a fixed generator G, the optimal discriminator is:

$$D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}, \tag{8}$$

  where $p_{\text{data}}(\boldsymbol{x})$ is the probability distribution of the real dataset evaluated at point $\boldsymbol{x}$ and $p_g(\boldsymbol{x})$ is the probability distribution of output of generator evaluated at point $\boldsymbol{x}$.

- **Proof**: According to the definition of expectation, the loss function in Eq. (3) can be stated as:

$$V(D, G) = \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))d\boldsymbol{x} + \int_{\boldsymbol{z}} p_z(\boldsymbol{z}) \log(1 - D(G(\boldsymbol{z})))d\boldsymbol{z}.$$

- According to Eq. (1), we have:

$$G(\boldsymbol{z}) = \boldsymbol{x} \implies \boldsymbol{z} = G^{-1}(\boldsymbol{x}) \implies d\boldsymbol{z} = (G^{-1})'(\boldsymbol{x})d\boldsymbol{x},$$

  where $(G^{-1})'(\boldsymbol{x})$ is the derivative of $(G^{-1})(\boldsymbol{x})$ with respect to (w.r.t.) $\boldsymbol{x}$. Hence:

$$V(D, G) = \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))d\boldsymbol{x} + \int_{\boldsymbol{z}} p_z(G^{-1}(\boldsymbol{x})) \log(1 - D(\boldsymbol{x}))(G^{-1})'(\boldsymbol{x})d\boldsymbol{x}.$$

Optimal Solution of GAN

- We found:

$$V(D,G) = \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))d\boldsymbol{x} + \int_{\boldsymbol{z}} p_z(G^{-1}(\boldsymbol{x})) \log(1 - D(\boldsymbol{x}))(G^{-1})'(\boldsymbol{x})d\boldsymbol{x}.$$

- The relation of distributions of input and output of generator is:

$$p_g(\boldsymbol{x}) = p_z(\boldsymbol{z}) \times G^{-1}(\boldsymbol{x}) = p_z(G^{-1}(\boldsymbol{x}))\, G^{-1}(\boldsymbol{x}), \tag{9}$$

where $G^{-1}(\boldsymbol{x})$ is the Jacobian of distribution at point $\boldsymbol{x}$. Hence:

$$\begin{aligned} V(D,G) &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))d\boldsymbol{x} + \int_{\boldsymbol{z}} p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))d\boldsymbol{x} \\ &= \int_{\boldsymbol{x}} \Big(p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))\Big) d\boldsymbol{x}. \end{aligned} \tag{10}$$

- For optimization in Eq. (3), taking derivative w.r.t. $D(\boldsymbol{x})$ gives:

$$\frac{\partial V(D,G)}{\partial D(\boldsymbol{x})} \overset{(a)}{=} \frac{\partial}{\partial D(\boldsymbol{x})}\Big(p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))\Big)$$

$$= \frac{p_{\text{data}}(\boldsymbol{x})}{D(\boldsymbol{x})} - \frac{p_g(\boldsymbol{x})}{1 - D(\boldsymbol{x})} = \frac{p_{\text{data}}(\boldsymbol{x})(1 - D(\boldsymbol{x})) - p_g(\boldsymbol{x})D(\boldsymbol{x})}{D(\boldsymbol{x})(1 - D(\boldsymbol{x}))} \overset{\text{set}}{=} 0$$

$$\implies p_{\text{data}}(\boldsymbol{x}) - p_{\text{data}}(\boldsymbol{x})D(\boldsymbol{x}) - p_g(\boldsymbol{x})D(\boldsymbol{x}) = 0 \implies D(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})},$$

where (a) is because taking derivative w.r.t. $D(\boldsymbol{x})$ considers a specific $\boldsymbol{x}$ and hence it removes the integral (summation). Q.E.D.

Optimal Solution of GAN

- **Theorem**: [1, Theorem 1] The optimal solution of GAN is when the distribution of generated data becomes equal to the distribution of data:

$$p_{g^*}(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x}). \tag{11}$$

- **Proof**: Putting the optimum $D^*(\boldsymbol{x})$, i.e. Eq. (8), in Eq. (10) gives:

$$\begin{aligned}
V(D^*, G) &= \int_{\boldsymbol{x}} \Big(p_{\text{data}}(\boldsymbol{x}) \log(D^*(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D^*(\boldsymbol{x})) \Big) d\boldsymbol{x} \\
&\overset{(8)}{=} \int_{\boldsymbol{x}} \Big[p_{\text{data}}(\boldsymbol{x}) \log\Big(\frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \Big) + p_g(\boldsymbol{x}) \log\Big(\frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \Big) \Big] d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} \Big[p_{\text{data}}(\boldsymbol{x}) \log\Big(\frac{p_{\text{data}}(\boldsymbol{x})}{2 \times \frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) + p_g(\boldsymbol{x}) \log\Big(\frac{p_g(\boldsymbol{x})}{2 \times \frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) \Big] d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} \Big[p_{\text{data}}(\boldsymbol{x}) \log\Big(\frac{p_{\text{data}}(\boldsymbol{x})}{\frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) + p_g(\boldsymbol{x}) \log\Big(\frac{p_g(\boldsymbol{x})}{\frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) \Big] d\boldsymbol{x} + \log(\frac{1}{2}) + \log(\frac{1}{2}) \\
&= \int_{\boldsymbol{x}} \Big[p_{\text{data}}(\boldsymbol{x}) \log\Big(\frac{p_{\text{data}}(\boldsymbol{x})}{\frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) + p_g(\boldsymbol{x}) \log\Big(\frac{p_g(\boldsymbol{x})}{\frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) \Big] d\boldsymbol{x} - \log(4) \\
&\overset{(a)}{=} \text{KL}\Big(p_{\text{data}}(\boldsymbol{x}) \Big\| \frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2} \Big) + \text{KL}\Big(p_g(\boldsymbol{x}) \Big\| \frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2} \Big) - \log(4),
\end{aligned} \tag{12}$$

where (a) is because of the definition of KL divergence.

Optimal Solution of GAN

- We found:

$$V(D^*, G) = \mathrm{KL}\Big(p_{\text{data}}(\boldsymbol{x}) \Big\| \frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}\Big) + \mathrm{KL}\Big(p_g(\boldsymbol{x}) \Big\| \frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}\Big) - \log(4).$$

- The Jensen-Shannon Divergence (JSD) is defined as [16]:

$$\mathrm{JSD}(P\|Q) := \frac{1}{2}\mathrm{KL}(P\|\frac{1}{2}(P+Q)) + \frac{1}{2}\mathrm{KL}(Q\|\frac{1}{2}(P+Q)), \tag{13}$$

  where P and Q denote the probability densities. In contrast to KL divergence, the JSD is symmetric.

- The obtained $V(D^*, G)$ can be restated as:

$$V(D^*, G) = 2\,\mathrm{JSD}\big(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_g(\boldsymbol{x})\big) - \log(4). \tag{14}$$

- According to Eq. (3), the generator minimizes $V(D^*, G)$. As the JSD is non-negative, the above loss function is minimized if we have:

$$\mathrm{JSD}\big(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_{g^*}(\boldsymbol{x})\big) = 0 \implies p_{\text{data}}(\boldsymbol{x}) = p_{g^*}(\boldsymbol{x}).$$

  Q.E.D.

Optimal Solution of GAN

- **Corollary**: [1, Theorem 1]: From Eqs. (11) and (14):

$$\begin{aligned} & p_{g^*}(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x}), \\ & V(D^*, G) = 2\,\text{JSD}\big(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_g(\boldsymbol{x})\big) - \log(4), \end{aligned}$$

we conclude that the optimal loss function in GAN is:

$$V(D^*, G^*) = -\log(4). \tag{15}$$

Convergence and Equilibrium Analysis of GAN

Convergence and Equilibrium Analysis of GAN

- **Theorem**: [1, Proposition 2] If the discriminator and generator have enough capacity and, at every iteration of the alternating optimization, the followings occur:
  - the discriminator is allowed to reach its optimum value as in Eq. (8), $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$, and
  - $p_g(\boldsymbol{x})$ is updated to minimize $V(D^*, G)$, as stated in Eq. (14), $V(D^*, G) = 2\,\text{JSD}\big(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_g(\boldsymbol{x})\big) - \log(4)$,

  then, $p_g(\boldsymbol{x})$ converges to $p_{\text{data}}(\boldsymbol{x})$, as stated in Eq. (11), $p_{g^*}(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$.

Convergence and Equilibrium Analysis of GAN

- **Proof**: The KL divergences in Eq. (12):

$$V(D^*, G) = \text{KL}\Big(p_{\text{data}}(\boldsymbol{x}) \Big\| \frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}\Big) + \text{KL}\Big(p_g(\boldsymbol{x}) \Big\| \frac{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}\Big) - \log(4).$$

  are convex functions w.r.t. $p_g(\boldsymbol{x})$. Hence, with sufficiently small updates of $p_g(\boldsymbol{x})$, it converges to $p_{\text{data}}(\boldsymbol{x})$. Note that Eq. (12), which we used here, holds if Eq. (8) holds, i.e., the discriminator is allowed to reach its optimum value. Q.E.D.

- The GAN loss, i.e. Eq. (3), can be restated as [17]:

$$\min_G \max_D \; V(D, G) := \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}\Big[f\big(D(\boldsymbol{x})\big)\Big] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}\Big[f\Big(-D\big(G(\boldsymbol{z})\big)\Big)\Big], \tag{16}$$

  where f is the negative logistic function, i.e., $f(x) := -\log(1 + \exp(-x))$. In fact, the function $f(.)$ can be any concave function. This formulation is slightly different from the original GAN in the sense that, here, the discriminator D outputs a real-valued scalar (without any activation function) while the discriminator of Eq. (3) outputs values in the range $(0, 1)$ after a sigmoid activation function.

- If D outputs 0.5 and 0, it means that it is completely confused in Eqs. (3) and (16), respectively. The Eq. (16) is a concave-concave loss function in most of the domain of discriminator [17, Proposition 3.1].

Convergence and Equilibrium Analysis of GAN

- **Theorem**: [17, Theorem 3.1]: After satisfying several reasonable assumptions (see [17] for details), a GAN with loss function of Eq. (16) is locally exponentially stable.
- **Lemma**: **Nash equilibrium** in GAN [18]: Nash equilibrium is the state of game where no player can improve its gain by choosing a different strategy. At the Nash equilibrium of GAN, we have:

$$V(D, G^*) \leq V(D^*, G^*) \leq V(D^*, G), \tag{17}$$

which is obvious because we are minimizing and maximizing $V(G, D)$ by the generator and discriminator, respectively, in Eq. (3):

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

- Empirical experiments have shown that GAN **may not reach its Nash equilibrium in practice** [18]. **Regularization** can help convergence of GAN to the Nash equilibrium [19]. It is shown in [19] that an effective regularization for GAN is **noise injection** [20] in which independent Gaussian noise is added to the training data points.

Conditional GAN

Conditional GAN

- As was explained before, in the original GAN, we randomly draw noise $\boldsymbol{z}$ from a prior distribution $p_z(\boldsymbol{z})$ and feed it to the generator. The generator outputs a point $\boldsymbol{x}$ from the noise $\boldsymbol{z}$. Assume that the dataset with which GAN is trained has c number of **classes**. The original GAN generates points from any class and we do **not have control to generate a point from a specific class**.
- Although, note that after GAN is trained, the latent space for $\boldsymbol{z}$ is meaningful in the sense that every part of the latent space results in generation of some specific points from some class. However, the **user cannot choose specifically what class to generate points from**.
- **Conditional GAN** (2014) [21], also called the conditional adversarial network, gives the user the opportunity to **choose the class of generation of points**. For the dataset $\{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$, let the one-hot encoded class labels be $\{\boldsymbol{y}_i \in \mathbb{R}^c\}_{i=1}^n$. In conditional GAN, we use the following loss function instead of Eq. (3):

$$\min_G \max_D \; V_C(D, G) := \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\Big[\log\big(D(\boldsymbol{x}|\boldsymbol{y})\big)\Big] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\Big[\log\Big(1 - D\big(G(\boldsymbol{z}|\boldsymbol{y})\big)\Big)\Big], \tag{18}$$

  where the discriminator and generator are both conditioned on the labels.
- In practice, for implementing the loss function (18), we **concatenate the one-hot encoded label $\boldsymbol{y}$ to the point $\boldsymbol{x}$ for the input to discriminator**. We also **concatenate the one-hot encoded label $\boldsymbol{y}$ to the noise $\boldsymbol{z}$ for the input to generator**. For these, the input layers of discriminator and generator are enlarged to accept the concatenated inputs.
- In the **test phase**, the user **chooses the desired class label** and the generator **generates a new point from that class**.

Deep Convolutional GAN (DCGAN)

Deep Convolutional GAN (DCGAN)

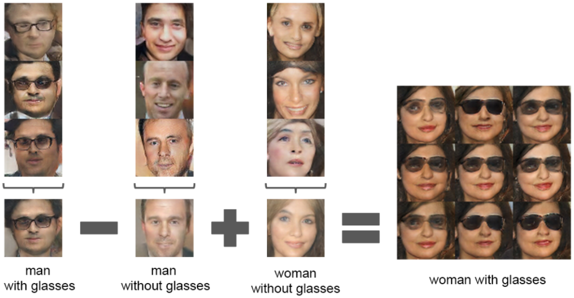
- **Deep Convolutional GAN (DCGAN)**, proposed in (2016) [22], made GAN **deeper** and generated **higher resolution** images than GAN.
- It also showed that it is **very hard to train a GAN**.
- In DCGAN, we use an **all-convolutional network** [23] which **replaces the pooling functions with strided convolutions**. In this way, the network **learns its own spatial downsampling**. This network is used for both generator and discriminator.
- In DCGAN, we also have only **convolutional layers** in the **input layer of generator** and **output layer of discriminator**, without any fully-connected layer. This **elimination of fully connected layers** is inspired by [24].
- We also apply **batch normalization** [25] to all layers **except the last layer of generative and the first layer of discriminator**. This is because batch normalization makes the mean of input to each neuron zero and its variance one; however, we **should learn the mean and variance of data** in the first layer of discriminator and the **mean and variance of data should be reproduced** by the last layer of generator.
- Batch normalization **reduces the problem of mode collapse**, which will be introduced later, with the price of **causing some fluctuations and instability** [22].
- **Virtual batch normalization** [26, Section 3.5]: Batch normalization has a problem; it makes the effect of every input $\boldsymbol{x}$ on network **dependent on other inputs in the mini-batch**. To not have this problem, Virtual Batch Normalization (VBN) **fixes the mini-batches initially once before start of training**; these mini-batches are called the reference batches. Every reference batch is normalized by only its own statistics (i.e., mean and covariance). VBN has been found to be effective in training of generator [26].

Deep Convolutional GAN (DCGAN)

- In DCGAN, the **last layer of generator** has the **hyperbolic tangent** activation function and its other layers have the **ReLU** activation function [15].
- As in GAN, the one-to-last layer of discriminator is flattened and connected to one neuron with the **sigmoid** activation function.
- In contrast to GAN, which uses the maxout activation function [14] for discriminator layers, DCGAN uses the **leaky rectified action function** for discriminator.

Vector Arithmetic in Latent Space

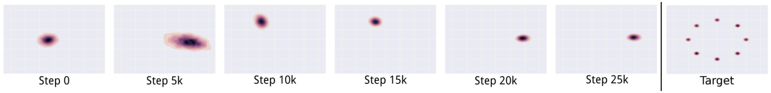
- DCGAN showed that we can **generate images from a specific domain if we train GAN on that domain**. For example, **bedroom images** were generated by DCGAN after being trained on a dataset of bedroom images.
- DCGAN also showed that the learned latent space is meaningful and we can do **vector arithmetic** in the latent space. Vector arithmetic in the latent space was previously used for showing the ability of **Word2Vec** [27]. DCGAN made it possible to do **vector arithmetic in the latent space for images**.
- An example of vector arithmetic by DCGAN is shown in this figure. In the latent space, the latent variables corresponding to man with glasses, man without glasses, and woman without glasses are used. Using one latent point for each of these does not work very well. An average of three latent vectors for each has worked properly in practice. The vector arithmetic works because **"man with glasses" minus "man without glasses" plus "woman without glasses" results in "woman with glasses"**.



Mode Collapse Problem in GAN

Mode Collapse Problem in GAN

- We expect from a GAN to learn a meaningful latent space of $\boldsymbol{z}$ so that **every specific value of $\boldsymbol{z}$ maps to a specific generated data point $\boldsymbol{x}$**. Also, **nearby $\boldsymbol{z}$ values in the latent space should be mapped to similar but a little different generations**.
- The **mode collapse problem** [28], also known as the **Helvetica scenario** [13], is a common problem in GAN models. It refers to when the **generator cannot learn a perfectly meaningful latent space** as was explained. Rather, it learns to **map several different $\boldsymbol{z}$ values to the same generated data point**.
- Mode collapse usually happens in GAN when the distribution of training data, $p_{\text{data}}(\boldsymbol{x})$, has **multiple modes**.
- An example of mode collapse is illustrated in this figure which shows training steps of a GAN model when the training data is a **mixture of Gaussians** [28]. In different training steps, GAN learns to **map all $\boldsymbol{z}$ values to one of the modes of mixture**. When the discriminator **learns to reject generation of some mode**, the generator **learns to map all $\boldsymbol{z}$ values to another mode**. However, it **never learns to generate all modes of the mixture**. We expect GAN to map some part, and not all parts, of the latent space to one of the modes so that all modes are covered by the whole latent space.



Mode Collapse Problem in GAN

- Another statement of the mode collapse is as follows [29, Fig. 1]. Assume $p_{\text{data}}(\boldsymbol{x})$ is **multi-modal** while the latent space $p_z(\boldsymbol{z})$ has only **one mode**.
- Consider two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ from two modes of data whose corresponding latent noises are $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$, respectively. According to the **mean value theorem**, there is a latent noise with the **absolute gradient value** $\|\boldsymbol{x}_2 - \boldsymbol{x}_1\| / \|\boldsymbol{z}_2 - \boldsymbol{z}_1\|$ where $\|.\|$ is a norm. As this gradient is Lipschitz continuous, when the two modes are **very far** resulting in a large $\|\boldsymbol{x}_2 - \boldsymbol{x}_1\|$, we face a problem. In this case, the latent noises between $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ generate data points between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ which are not in the modes of data and thus are not valid.
- There exist various methods which resolve the mode collapse problem in GAN and adversarial learning. Some of them **make the latent space a mixture distribution** to imitate generation of the multi-modal training data. Some of them, however, have other approaches.
- These methods are Minibatch GAN (2016) [26, Section 3.2], Unrolled GAN (2017) [28], Bourgain GAN (2018) [29], Mixture GAN (MGAN) (2018) [30], Dual Discriminator GAN (D2GAN) (2017) [31], and Wasserstein GAN (WGAN) (2017) [32].
- For more information on GAN and its variants, see our tutorial [33].

Some Applications of GAN

Some Applications of GAN

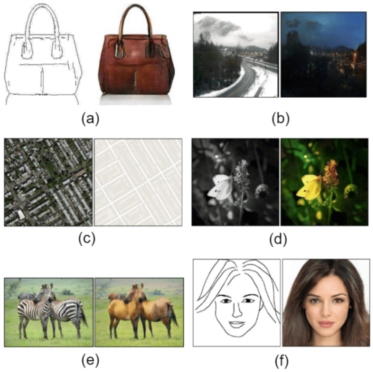
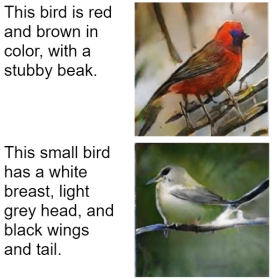


Figure 9. Image-to-image translation: (a) coloring a sketch, (b) changing daylight to night darkness in image, (c) changing an aerial image to a map, (d) coloring a black-and-white image, (e) transforming zebra to horse and vice versa, and (f) generating a facial image from a facial sketch. Transformations in (a), (b), (c), and (d) are by PatchGAN whose credits are for (Isola et al., 2017). Transformations in (e) and (f) are by CycleGAN (credit: (Zhu et al., 2017)) and DeepFaceDrawing (credit: (Chen et al., 2020)), respectively.

CycleGAN (credit: (2017) [34]) and DeepFaceDrawing (credit: (2020) [35])

Some Applications of GAN



This bird is red and brown in color, with a stubby beak.

This small bird has a white breast, light grey head, and black wings and tail.

Figure 10. Text-to-image translation by StackGAN. Images are from (Zhang et al., 2017).

credit: [36]

Some Applications of GAN

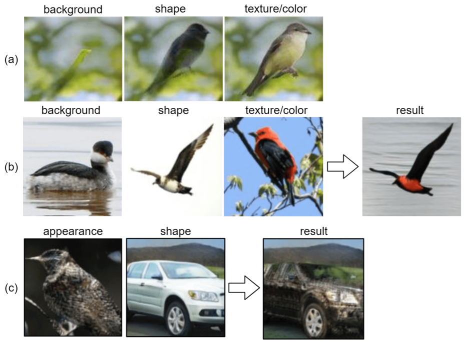


Figure 11. Mixing image characteristics using GAN: (a) Generating an image with background, shape, and color characteristics by FineGAN, (b) generating an image by borrowing its characteristics from three images using MixNMatch, and (c) generating an image by borrowing its characteristics from different domains using improved MixNMatch. Images are from (Singh et al., 2019), (Li et al., 2020), and (Ojha et al., 2021b), respectively.

Images are from [37], [38], and [39], respectively.

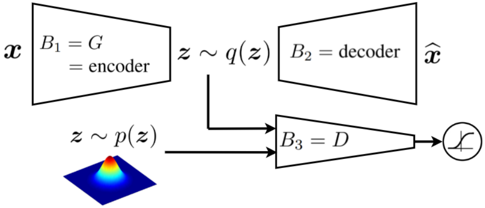
Adversarial Autoencoder

Adversarial Autoencoder

- Previously, **variational Bayes** was used in an autoencoder setting to have **variational autoencoder** [40, 41].
- Likewise, **adversarial learning** can be used in an autoencoder setting [42].
- Several **adversarial-based autoencoders** exist:
  - Adversarial Autoencoder (AAE) (2015) [43]
  - PixelGAN autoencoder (2017) [44]
  - Implicit Autoencoder (IAE) (2018) [45]

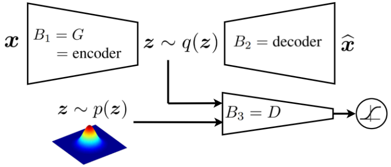
Adversarial Autoencoder

- **Adversarial Autoencoder (AAE)** was proposed in (2015) [43].
- The structure of AAE is depicted in this figure. Each of the blocks B_1, B_2, and B_3 in this figure has several network layers with nonlinear activation functions.


$\boldsymbol{x}$
$B_1 = G$
$= \text{encoder}$
$\boldsymbol{z} \sim q(\boldsymbol{z})$
$B_2 = \text{decoder}$
$\widehat{\boldsymbol{x}}$
$\boldsymbol{z} \sim p(\boldsymbol{z})$
$B_3 = D$

- AAE has an encoder (i.e., block B_1) and a decoder (i.e., block B_2). The input of encoder is a real data point $\boldsymbol{x} \in \mathbb{R}^d$ and the output of decoder is the reconstructed data point $\widehat{\boldsymbol{x}} \in \mathbb{R}^d$. One of the low-dimensional middle layers is the latent (or code) layer, denoted by $\boldsymbol{z} \in \mathbb{R}^p$, where $p \ll d$.
- The encoder and decoder model **conditional distributions** $p(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{x}|\boldsymbol{z})$, respectively. Let the distribution of the latent variable in the autoencoder be denoted by $q(\boldsymbol{z})$. This is the **posterior distribution** of latent variable.
- The blocks B_1 and B_3 are the **generator** G and **discriminator** D of adversarial network, respectively. We also have a **prior distribution**, denoted by $p(\boldsymbol{z})$, on the latent variable which is **chosen by the user**. This prior distribution can be a p-dimensional normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, for example.

Adversarial Autoencoder


$\boldsymbol{x}$
$B_1 = G$ $= \text{encoder}$
$\boldsymbol{z} \sim q(\boldsymbol{z})$
$B_2 = \text{decoder}$
$\widehat{\boldsymbol{x}}$
$\boldsymbol{z} \sim p(\boldsymbol{z})$
$B_3 = D$

- The **encoder** of the autoencoder (i.e., block B_1) is the generator G which generates the latent variable from the posterior distribution:

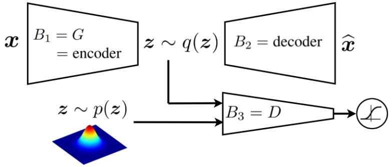
$$G(\boldsymbol{x}) = \boldsymbol{z} \sim q(\boldsymbol{z}). \tag{19}$$

- The discriminator D (i.e., block B_3) has a single output neuron with **sigmoid** activation function. It classifies the latent variable $\boldsymbol{z}$ to be a **real latent variable from the prior distribution** $p(\boldsymbol{z})$ or **a generated latent variable by the encoder** of autoencoder:

$$D(\boldsymbol{z}) := \begin{cases} 1 & \text{if } \boldsymbol{z} \text{ is real, i.e., } \boldsymbol{z} \sim p(\boldsymbol{z}) \\ 0 & \text{if } \boldsymbol{z} \text{ is generated, i.e., } \boldsymbol{z} \sim q(\boldsymbol{z}). \end{cases} \tag{20}$$

- As was explained, the **block** B_1 is shared between the **autoencoder and the adversarial network**.
- This adversarial learning makes both autoencoder and adversarial network stronger gradually because the **autoencoder tries to generate the latent variable which is very similar to the real latent variable from the prior distribution**. In this way, it tries to fool the discriminator. The discriminator, on the other hand, tries to become stronger not to be fooled by the encoder of autoencoder.

Adversarial Autoencoder



$\boldsymbol{x}$
$B_1 = G$
$= \text{encoder}$
$\boldsymbol{z} \sim q(\boldsymbol{z})$
$B_2 = \text{decoder}$
$\widehat{\boldsymbol{x}}$
$\boldsymbol{z} \sim p(\boldsymbol{z})$
$B_3 = D$

- In AAE, we have **alternating optimization** [11] where **reconstruction and regularization phases** are repeated iteratively.
- In the **reconstruction phase**, the mean squared error is minimized between the data $\boldsymbol{x}$ and the reconstructed data $\widehat{\boldsymbol{x}}$. In the **regularization phase**, the discriminator and generator are updated using the GAN approach.
- For each of these updates, we use stochastic gradient descent [11] with **backpropagation**. Overall, the two phases are performed as:

$$B_1', B_2^{(k+1)} := \arg\min_{B_1, B_2} \|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|_2^2, \quad (21)$$

$$\begin{cases} B_3^{(k+1)} := B_3^{(k)} - \eta^{(k)} \frac{\partial}{\partial B_3}\Big(V(B_3, B_1')\Big), \\ B_1^{(k+1)} := B_1' - \eta^{(k)} \frac{\partial}{\partial B_1}\Big(V(B_3^{(k+1)}, B_1)\Big), \end{cases} \quad (22)$$

  where $B_1 = G$ and $B_3 = D$. Eq. (21) is the **reconstruction phase** and Eq. (22) is the **regularization phase**.
- For **supervised and semi-supervised AAE**, refer to our tutorial paper [33].

Acknowledgment

- Some slides are based on our tutorial paper: “Generative adversarial networks and adversarial autoencoders: Tutorial and survey” [33]
- For more information on GAN and its variants, see our tutorial [33]. It covers various variants of GAN:
  - vanilla GAN, conditional GAN, DCGAN, the mode collapse problem, minibatch GAN, unrolled GAN, BourGAN, mixture GAN, D2GAN, Wasserstein GAN, f-GAN, adversarial variational Bayes, Bayesian GAN, feature matching in GAN, InfoGAN, GRAN, LSGAN, energy-based GAN, CatGAN, MMD GAN, LapGAN, progressive GAN, triple GAN, LAG, GMAN, AdaGAN, CoGAN, inverse GAN, BiGAN, ALI, SAGAN, Few-shot GAN, SinGAN, interpolation and evaluation of GAN, image-to-image translation (including PatchGAN, CycleGAN, DeepFaceDrawing, simulated GAN, interactive GAN), text-to-image translation (including StackGAN), mixing image characteristics (including FineGAN and MixNMatch), adversarial autoencoder, PixelGAN, implicit autoencoder.
- Some slides of this slide deck are inspired by teachings of Prof. Ali Ghodsi at University of Waterloo, Department of Statistics.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, vol. 27, 2014.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[3] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, “Adversarial machine learning,” in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pp. 43–58, 2011.

[4] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “DeepFool: a simple and accurate method to fool deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

[5] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” in *International Conference on Learning Representations*, 2017.

[6] A. Kurakin, I. Goodfellow, S. Bengio, *et al.*, “Adversarial examples in the physical world,” in *International Conference on Learning Representations, Workshop Track*, 2017.

[7] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *International Conference on Learning Representations*, 2018.

References (cont.)

[8] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes," in *Advances in neural information processing systems*, pp. 841–848, 2002.

[9] B. Ghojogh, H. Nekoei, A. Ghojogh, F. Karray, and M. Crowley, "Sampling algorithms, from survey sampling to Monte Carlo methods: Tutorial and literature review," *arXiv preprint arXiv:2011.00901*, 2020.

[10] D.-Z. Du and P. M. Pardalos, *Minimax and applications*, vol. 4. Springer Science & Business Media, 2013.

[11] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "KKT conditions, first-order and second-order optimization, and distributed optimization: Tutorial and survey," *arXiv preprint arXiv:2110.01858*, 2021.

[12] L. Mescheder, S. Nowozin, and A. Geiger, "The numerics of GANs," in *Advances in neural information processing systems*, 2017.

[13] I. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," in *Advances in neural information processing systems, Tutorial rack*, 2016.

[14] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *International conference on machine learning*, pp. 1319–1327, 2013.

[15] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *International Conference on Machine Learning*, 2010.

References (cont.)

[16] F. Nielsen, "A family of statistical symmetric divergences based on Jensen's inequality," *arXiv preprint arXiv:1009.4004*, 2010.

[17] V. Nagarajan and J. Z. Kolter, "Gradient descent GAN optimization is locally stable," in *Advances in neural information processing systems*, 2017.

[18] F. Farnia and A. Ozdaglar, "Do GANs always have Nash equilibria?," in *International Conference on Machine Learning*, pp. 3029–3039, 2020.

[19] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for GANs do actually converge?," in *International conference on machine learning*, pp. 3481–3490, PMLR, 2018.

[20] B. Ghojogh and M. Crowley, "The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial," *arXiv preprint arXiv:1905.12787*, 2019.

[21] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[22] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *International Conference on Learning Representations*, 2016.

[23] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *International Conference on Learning Representations, Workshop Track*, 2015.

References (cont.)

[24] A. Mordvintsev, C. Olah, and M. Tyka, "Inceptionism: Going deeper into neural networks." Google AI Blog, 2015.

[25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, 2015.

[26] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," *Advances in neural information processing systems*, vol. 29, pp. 2234–2242, 2016.

[27] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[28] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," in *International Conference on Learning Representations*, 2017.

[29] C. Xiao, P. Zhong, and C. Zheng, "BourGAN: Generative networks with metric embeddings," in *Advances in neural information processing systems*, 2018.

[30] Q. Hoang, T. D. Nguyen, T. Le, and D. Phung, "MGAN: Training generative adversarial nets with multiple generators," in *International Conference on Learning Representations*, 2018.

References (cont.)

[31] T. D. Nguyen, T. Le, H. Vu, and D. Phung, "Dual discriminator generative adversarial nets," *Advances in neural information processing systems*, 2017.

[32] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, pp. 214–223, 2017.

[33] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Generative adversarial networks and adversarial autoencoders: Tutorial and survey," *arXiv preprint arXiv:2111.13282*, 2021.

[34] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

[35] S.-Y. Chen, W. Su, L. Gao, S. Xia, and H. Fu, "DeepFaceDrawing: Deep generation of face images from sketches," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 72–1, 2020.

[36] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, "StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 5907–5915, 2017.

[37] K. K. Singh, U. Ojha, and Y. J. Lee, "FineGAN: Unsupervised hierarchical disentanglement for fine-grained object generation and discovery," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6490–6499, 2019.

References (cont.)

[38] Y. Li, K. K. Singh, U. Ojha, and Y. J. Lee, “MixNMatch: Multifactor disentanglement and encoding for conditional image generation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8039–8048, 2020.

[39] U. Ojha, K. K. Singh, and Y. J. Lee, “Generating furry cars: Disentangling object shape & appearance across multiple domains,” in *International Conference on Learning Representations*, 2021.

[40] D. P. Kingma and M. Welling, “Auto-encoding variational Bayes,” in *International Conference on Learning Representations*, 2014.

[41] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, “Factor analysis, probabilistic principal component analysis, variational inference, and variational autoencoder: Tutorial and survey,” *arXiv preprint arXiv:2101.00734*, 2021.

[42] A. Makhzani, *Unsupervised representation learning with autoencoders*. PhD thesis, University of Toronto, 2018.

[43] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, “Adversarial autoencoders,” *arXiv preprint arXiv:1511.05644*, 2015.

[44] A. Makhzani and B. Frey, “PixelGAN autoencoders,” in *Advances in neural information processing systems*, 2017.

[45] A. Makhzani, “Implicit autoencoders,” *arXiv preprint arXiv:1805.09804*, 2018.