Generative Adversarial Network

Dr. Josif Grabocka

ISMLL, University of Hildesheim

Generative Adversarial Network

Generative Models

Learn to generate data that looks as close as possible to a real dataset:

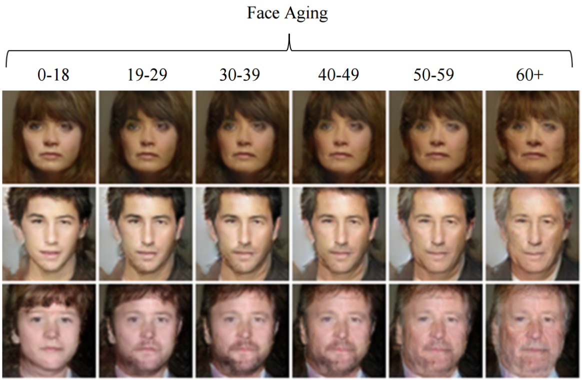

Figure 1: Conditional Generation, Source: Antipov 2017

Generation as a Maximum Likelihood Task

- Maximize the likelihood of observing the data using parameters θ:

$$\begin{aligned}\theta^* &= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} p_{\text{model}}(x^{(i)}; \theta) \\ &= \underset{\theta}{\operatorname{argmax}} \log \prod_{i=1}^{n} p_{\text{model}}(x^{(i)}; \theta) \\ &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log p_{\text{model}}(x^{(i)}; \theta)\end{aligned}$$

- Alternatively think it as a minimization of the distance between the generated distribution p_{model} and observed data p_{model}

$$\theta^* = \underset{\theta}{\operatorname{argmax}} D_{KL}\left(p_{\text{data}}(x) \,||\, p_{\text{model}}(x; \theta)\right)$$

Minimax - Game Theory

- A zero-sum game: Sum of the outcomes (loss or gain) among participants is zero. E.g.: Board games as Chess, ...

- **Minimax** is a strategy to win a zero-sum game by:
  - **Max**imizing the **Min**imum gain of an action
  - Maximizing the worst-case outcome without knowing the future moves of other players

- Let the g_i denote the minimax gain of player i when:
  - player i plays an action $a_i \in \mathcal{A}$ among many feasible actions $\mathcal{A}$
  - player(s) j followed with action(s) a_j
  - $g(a_i, a_j)$ denote the gain of player i as a result of the round of actions

$$g_i = \max_{a_i \in \mathcal{A}} \min_{a_j \in \mathcal{A}} \; g_i\left(a_i, a_j\right)$$

Minimax - Example

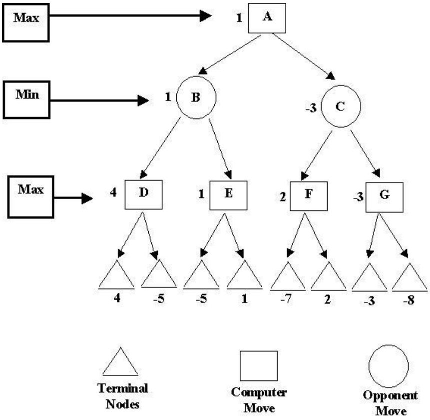

Figure 2: Minimax for Game Playing, Source: `http://www.cs.nott.ac.uk`

Generative Adversarial Networks (GAN)

- Two agents play a minimax game:
  - **Generator**: Generate synthetic data aiming to make them as similar as possible to real data
  - **Discriminator**: Distinguish if an input sample comes from the real data distribution

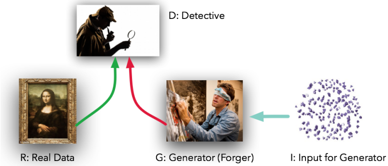

Figure 3: GAN, Courtesy of Dev Nag

- *Generator* **MIN**imizes the following:
  - *Discriminator* **MAX**imizes the accuracy of counterfeit detection

GAN - Problem

- **Generator**:
  - Needs to learn distribution p_g over data instances $x \in \mathbb{R}^M$
  - $G(z, \theta_g) : \mathbb{R}^L \to \mathbb{R}^M$ is a neural network
  - z is a noise fed into the generator following a prior on $z \sim p_z(z)$
- **Discriminator**:
  - $D(x, \theta_d) : \mathbb{R}^M \to [0, 1]$ is a neural network
  - $D(x)$ is the probability that x comes from real data rather than p_g
- GAN aims at learning θ_g and θ_d which optimize an objective V as $\min_G \max_D V(D, G)$, by:

$$\min_{\theta_g} \max_{\theta_d} \; \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x, \theta_d)] + \mathbb{E}_{z \sim p_z(z)} [\log (1 - D(G(z, \theta_g), \theta_d))]$$

GAN - Optimization

Algorithm 1: GAN Optimization

1: **for** $1, \ldots, \text{numIters}$ **do**
2: **for** $1, \ldots, K$ **do**
3: Sample n noise samples: $\{z^{(1)}, \ldots, z^{(n)}\}$ from $p_z(z)$
4: Sample n real samples: $\{x^{(1)}, \ldots, x^{(n)}\}$ from $p_{\text{data}}(x)$
5: Update discriminator parameters θ_d using gradient **ascent**:

$$\nabla_{\theta_d} \frac{1}{n} \sum_{i=1}^{n} \log D(x^{(i)}, \theta_d) + \log \left(1 - D(G(z^{(i)}, \theta_g), \theta_d)\right)$$

6: Sample n noise samples: $\{z^{(1)}, \ldots, z^{(n)}\}$ from $p_z(z)$
7: Update generator parameters θ_g using gradient **descent**:

$$\nabla_{\theta_g} \frac{1}{n} \sum_{i=1}^{n} \log \left(1 - D(G(z^{(i)}, \theta_g), \theta_d)\right)$$

GAN - Optimization (II)

- How to compute the derivatives w.r.t. the discriminator and generator weights? [check on board]

- What happens to the gradients ∇_{θ_g} in the early iterations?

Optimal Discriminator

- Given any generator G, an optimal discriminator D maximizes:

$$
\begin{aligned}
V(G, D) &= \int_x p_{\text{data}}(x) \log(D(x))dx + \int_z p_z(z) \log(1 - D(G(z)))dz \\
&= \int_x p_{\text{data}}(x) \log(D(x)) + p_g(x) \log(1 - D(x))dx
\end{aligned}
$$

- The maximum of $a \log y + b \log (1 - y)$ is $\frac{a}{a+b}$ for $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$
- Therefore, for a fixed G the optimal discriminator is:

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

Reformulate the objective

- Knowing the optimal D for a G, we can rewrite the optimization as:

$$
\begin{aligned}
C(G) &= \max_D V(G, D) \\
&= \mathbb{E}_{x \sim p_{\text{data}}} [\log \; D^*(x)] + \mathbb{E}_{z \sim p_x} [\log \; (1 - D^* G(z))] \\
&= \mathbb{E}_{x \sim p_{\text{data}}} [\log \; D^*(x)] + \mathbb{E}_{x \sim p_g} [\log \; (1 - D^*(x))] \\
&= \mathbb{E}_{x \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}\right] + \mathbb{E}_{x \sim p_g} \left[\log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)}\right]
\end{aligned}
$$

Optimality when generator meets real data

- For $p_{\text{data}}(x) = p_g(x)$ then $C(G) = -\log 4$, which is the minimum of $C(G)$ if there is no difference among the distributions of $p_{\text{data}}(x)$and $p_g(x)$, or:

$$C(G) = -\log(4) + KL\left(p_{\text{data}} || \frac{p_{\text{data}}(x) + p_g(x)}{2}\right) + KL\left(p_g || \frac{p_{\text{data}}(x) + p_g(x)}{2}\right)$$

- Which turns out to be the Jensen-Shannon divergence:

$$C(G) = -\log(4) + 2\ JSD(p_{\text{data}}(x)||p_g(x))$$

- JSD is known to be zero only for $p_{\text{data}}(x) = p_g(x)$, thus the minimum loss is when the generator matches exactly the true data distribution

DCGAN

- Replace pooling with strided convolutions (discriminator) and fractional-strided convolutions (generator)
- Use batchnorm in both generator and discriminator
- Remove fully connected hidden layers
- Use ReLU in generator for all layers, except output (tanh)
- Use LeakyReLU in discriminator for all layers

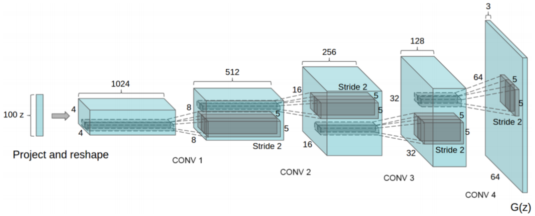

Figure 4: DCGAN Generator Architecture, Source: Radford et al., ICLR 2016

DCGAN (II)

Figure 5: DCGAN Generated Images discriminated against the LSUN dataset, Source: Radford et al., ICLR 2016