Generative Adversarial Networks, Reinforcement Learning

Junxian He
Apr 26, 2024
The GAN Model

\[ p(z) \]

The same as the VAE model, except that \( x \) is a deterministic function of \( z \), but it can be a distribution as well

Can VAE use a deterministic \( x = G(z) \)?

\[ X = G(z) \]

Sometimes we call GANs *implicit* generative models

You can draw samples, but hard to evaluate \( p(x) \)
Training GANs

Computation Graph

\[ p(z) \]

\[ Z \rightarrow X \]

\[ X = G(z) \]

\[ x \sim P_{data}(x) \]

1. Generator is trained to produce realistic examples to fool the discriminator
2. Discriminator is trained to discriminate real and fake examples
Training GANs

1. Generator is trained to produce realistic examples to fool the discriminator.
2. Discriminator is trained to discriminate real and fake examples.

The two objectives are against each other.

Adversarial Game

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

D(x) outputs the probability of x being the real example.

G(z) is trained to minimize the probability of G(z) recognized as “fake” by D.

D(x) is trained with a standard classification loss.
Theory of GANs

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

**Proposition 1.** For \( G \) fixed, the optimal discriminator \( D \) is

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

\[
C(G) = \max_D V(G, D)
\]

\[
= \mathbb{E}_{x \sim p_{\text{data}}}[\log D_G^*(x)] + \mathbb{E}_{z \sim p_z}[\log (1 - D_G^*(G(z)))]
\]

\[
= \mathbb{E}_{x \sim p_{\text{data}}}[\log D_G^*(x)] + \mathbb{E}_{x \sim p_g}[\log (1 - D_G^*(x))]
\]

\[
= \mathbb{E}_{x \sim p_{\text{data}}} \left[ \log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \right] + \mathbb{E}_{x \sim p_g} \left[ \log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)} \right]
\]
Theory of GANs

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

**Theorem 1.** The global minimum of the virtual training criterion \( C(G) \) is achieved if and only if \( p_g = p_{\text{data}} \). At that point, \( C(G) \) achieves the value \(- \log 4\).

\[
C(G) = - \log(4) + KL \left( p_{\text{data}} \parallel \frac{p_{\text{data}} + p_g}{2} \right) + KL \left( p_g \parallel \frac{p_{\text{data}} + p_g}{2} \right)
\]

\[
C(G) = - \log(4) + 2 \cdot JSD \left( p_{\text{data}} \parallel p_g \right)
\]

**Proposition 2.** If \( G \) and \( D \) have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given \( G \), and \( p_g \) is updated so as to improve the criterion

\[
\mathbb{E}_{x \sim p_{\text{data}}} [\log D_G^*(x)] + \mathbb{E}_{x \sim p_g} [\log(1 - D_G^*(x))]
\]

then \( p_g \) converges to \( p_{\text{data}} \)
Training GANs

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 \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
- Sample minibatch of \( m \) examples \( \{x^{(1)}, \ldots, x^{(m)}\} \) from data generating distribution \( p_{\text{data}}(x) \).
- Update the discriminator by ascending its stochastic gradient:

\[
\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D(x^{(i)}) + \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].
\]

**end for**

- Sample minibatch of \( m \) noise samples \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
- Update the generator by descending its stochastic gradient:

\[
\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right).
\]

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
Training GANs

1. GAN is a new algorithm to train a common generative model (like VAE)

2. GAN training is not MLE

What is it then?

Suppose the generator $G(x)$ is parameterized by $\theta$, then what is the gradient when updating $G(x)$?

\[
C(G) = - \log(4) + KL(p_{\text{data}} || \frac{p_{\text{data}} + p^*_g}{2}) + KL(p_g || \frac{p_{\text{data}} + p^*_g}{2})
\]

$p^*_g$ is from the solution of the discriminator, which is fixed when optimizing $\theta$

\[
\nabla_\theta C(G) = \nabla_\theta KL(p_g(x; \theta) || \frac{p_{\text{data}} + p^*_g(x)}{2})
\]
Training GANs

Recall that MLE is equivalent to minimizing $KL(p_{data}(x) \mid \mid p_g(x))$

For GANs, the generator is to minimize $KL(p_g(x; \theta) \mid \mid \frac{p_{data} + p_g^*(x)}{2})$

$$KL(p \mid \mid q) \neq KL(q \mid \mid p)$$

KL divergence is asymmetric, and GANs’ KL divergence is in the opposite direction with respect to MLE
GANs v.s. VAEs

GANs are widely demonstrated to show superiority to VAEs on generating realistic, vivid images. In contrast, VAEs’ generation is more blurred.

GANs’ generated images

GANs’ generation can “miss mode” of the data distribution, where the generated images are not diverse to cover all the data distributions (VAEs do not have this issue).

Brock et al. LARGE SCALE GAN TRAINING FOR HIGH FIDELITY NATURAL IMAGE SYNTHESIS. ICLR 2019.
Implication of the KL divergence

$KL(p_{data}(x) \mid \mid p_g(x))$ v.s. $KL(p_g(x) \mid \mid p_{data}(x))$

- $p_{data}$
- $p_{VAE}$
- $p_{GAN}$

Generates realistic images but lack diversity

Generates blurred/unrealistic images
Reinforcement Learning
Learning Tasks

- Supervised learning - $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - Regression - $y^{(i)} \in \mathbb{R}$
  - Classification - $y^{(i)} \in \{1, \ldots, C\}$
- Unsupervised learning - $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$
  - Clustering
  - Dimensionality reduction
- Reinforcement learning - $\mathcal{D} = \{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=1}^{T}$
RL Setup

In many cases, we cannot precisely define what the correct output is (think of we want to train a robot to walk)

Agent chooses actions which can depend on past
Environment can change state with each action
Reward (Output) depends on (Inputs) action and state of environment

Goal: maximize the total reward
Differences from Supervised Learning

- Maximize reward (rather than learn reward)
- Inputs are not iid – state & action depends on past

Supervised training is like imitation
RL Examples
RL Setup

• State space, $S$

• Action space, $\mathcal{A}$

• Reward function
  • Stochastic, $p(r \mid s, a)$
  • Deterministic, $R: S \times \mathcal{A} \rightarrow \mathbb{R}$

• Transition function
  • Stochastic, $p(s' \mid s, a)$
  • Deterministic, $\delta: S \times \mathcal{A} \rightarrow S$

In this lecture, we assume they are known

• Reward and transition functions can be known or unknown
RL Setup

- Policy, $\pi : S \rightarrow A$
  - Specifies an action to take in every state

- Value function, $V^\pi : S \rightarrow \mathbb{R}$
  - Measures the expected total reward of starting in some state $s$ and executing policy $\pi$, i.e., in every state, taking the action that $\pi$ returns
RL Example - gridworld

\[ S = \text{all empty squares in the grid} \]

\[ \mathcal{A} = \{\text{up, down, left, right}\} \]

Deterministic transitions

Rewards of +1 and -1 for entering the labelled squares

Terminate after receiving either reward
RL Example - gridworld

Is this policy optimal?
RL Example - gridworld

Optimal policy given a reward of -2 per step
RL Example - gridworld

Optimal policy given a reward of -0.5 per step

What would be the algorithm to find the optimal policy automatically?
Markov Decision Process

1. Start in some initial state $s_0$

2. For time step $t$:
   a. Agent observes state $s_t$
   b. Agent takes action $a_t = \pi(s_t)$  \textbf{Deterministic policy}
   c. Agent receives reward $r_t \sim p(r | s_t, a_t)$
   d. Agent transitions to state $s_{t+1} \sim p(s' | s_t, a_t)$

• MDPs make the \textbf{Markov assumption}: the reward and next state only depend on the current state and action.
Discounted Reward

Total reward is

\[ \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots \]

where \( 0 < \gamma < 1 \) is some discount factor for future rewards

Why discount?

- Mathematically tractable – total reward doesn’t explode
  
  \[ 1 + 1 + 1 + \ldots = \infty \text{ but } 1 + 0.8\times1 + (0.8)^2\times1 + \ldots = 5 \]

- Actions don’t have lasting impact
Key Challenges

• The algorithm has to gather its own training data

• The outcome of taking some action is often stochastic or unknown until after the fact

• Decisions can have a delayed effect on future outcomes (exploration-exploitation tradeoff)

  - explore decisions whose reward is uncertain
  - exploit decisions which give high reward
RL: Objective function

• Find a policy \( \pi^* = \arg\max_{\pi} V^\pi(s) \quad \forall \ s \in S \)

• \( V^\pi(s) = \mathbb{E}[\text{discounted total reward of starting in state} \ s \ \text{and executing policy} \ \pi \ \text{forever}] \)

\[
= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[R(s_t, \pi(s_t))]
\]

where \( 0 < \gamma < 1 \) is some discount factor for future rewards
Value Function

Bellman equations

\[ V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s_1 \in S} p(s_1 | s, \pi(s)) V^\pi(s_1) \]

Recursive form

Immediate reward

Expected (Discounted) Future reward
Solve Value Function

\[ V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s_1 \in S} p(s_1 \mid s, \pi(s)) V^\pi(s_1) \]

Given \( R \), transition function \( p \), and policy \( \pi(s) \), we can utilize this equation to solve \( V(s) \) for any \( s \)

How?

Suppose the state size is finite \(|S|\), you have \(|S|\) linear equations with \(|S|\) variables
Optimal value function and policy

• Optimal value function:

\[ V^*(s) = \max_{a \in \mathcal{A}} [R(s, a) + \gamma \sum_{s' \in S} p(s' | s, a)V^*(s')] \]

• System of \(|S|\) equations and \(|S|\) variables – nonlinear!

• Optimal policy:

\[ \pi^*(s) = \arg\max_{a \in \mathcal{A}} [R(s, a) + \gamma \sum_{s' \in S} p(s' | s, a)V^*(s')] \]

- Immediate reward
- Expected (Discounted) Future reward

• Insight: if you know the optimal value function, you can solve for the optimal policy!
Value Iteration

**Algorithm 4 Value Iteration**

1: For each state $s$, initialize $V(s) := 0$.
2: **for** until convergence **do**
3:  **for** every state, update
   
   $$V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s').$$

After find the optimal value function, we can find the optimal policy
Both value iteration and policy iteration are standard algorithms for solving MDPs, there isn’t universal agreement over which is better.
Next Questions

1. How to handle unknown state transition and reward functions?
2. How to handle continuous states and actions?