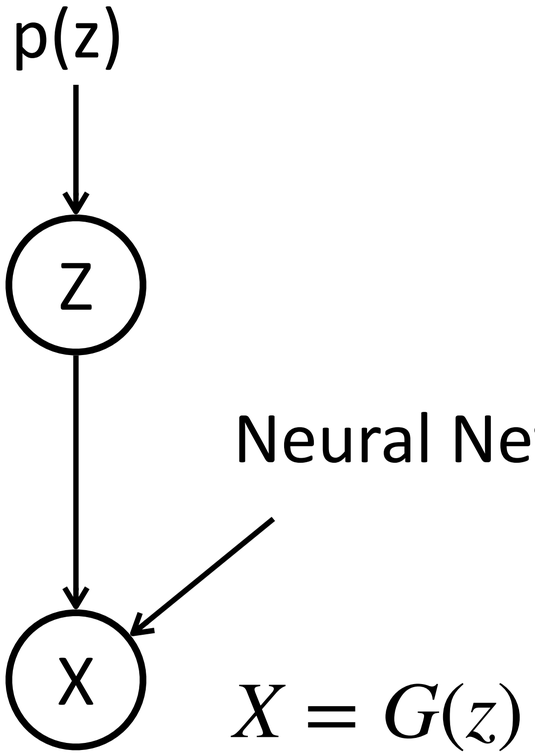
香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

COMP 5212
Machine Learning
Lecture 22

Generative Adversarial Networks, Reinforcement Learning

Junxian He
Apr 26, 2024

The GAN Model



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

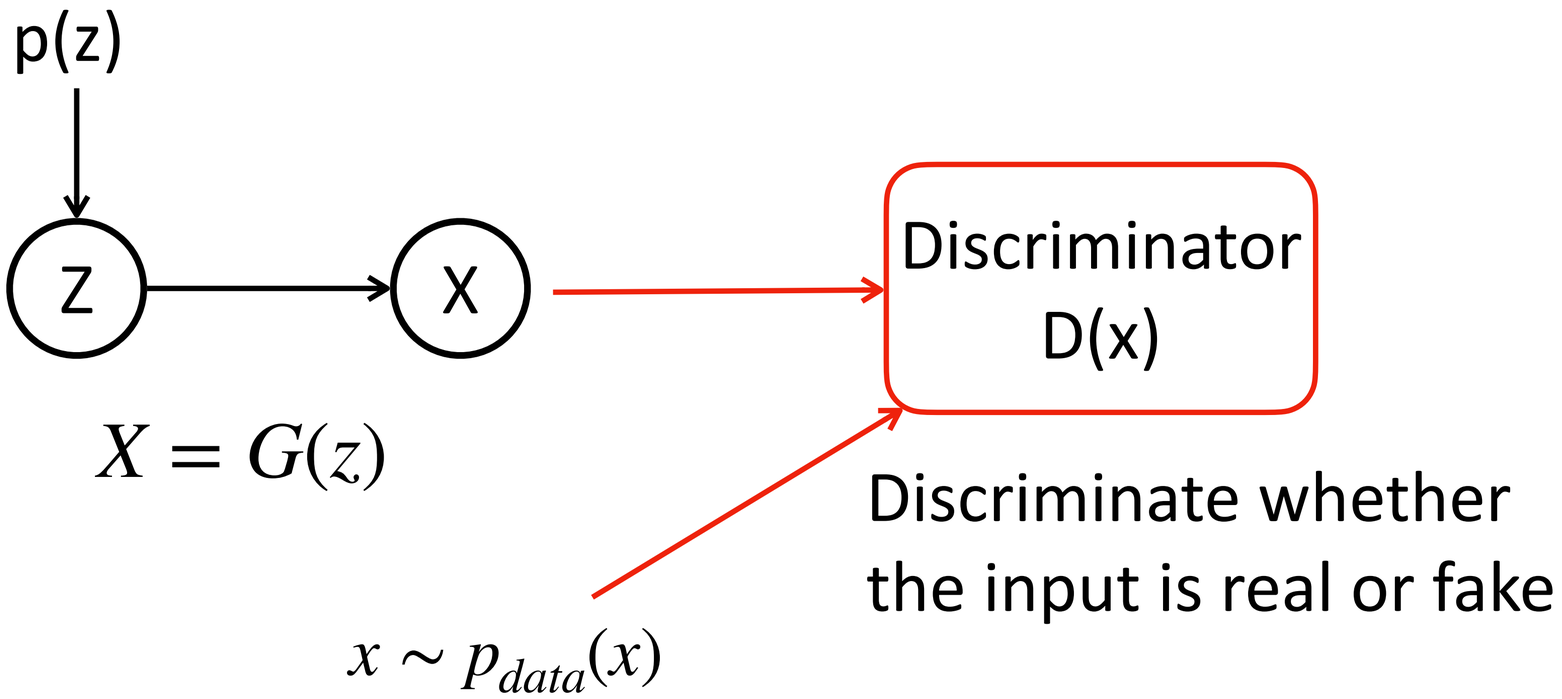
Can VAE use a deterministic x = G(z)?

Sometimes we call GANs *implicit* generative models

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

Training GANs

Computation Graph



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

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

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

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

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

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

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

Theory of GANs

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

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

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

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

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*

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

then p_g converges to p_{data}

Training GANs

Inner loop to update discriminator first

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

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

end for

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

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

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 θ, then what is the gradient when updating G(x)?

$$C(G) = -\log(4) + KL(p_{data} || \frac{p_{data} + p_g^*}{2}) + KL(p_g || \frac{p_{data} + p_g^*}{2})$$

p_g^* is from the solution of the discriminator, which is fixed when optimizing θ

$$\nabla_\theta C(G) = \nabla_\theta KL(p_g(x;\theta) || \frac{p_{data} + p_g^*(x)}{2})$$

Training GANs

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

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

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

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)

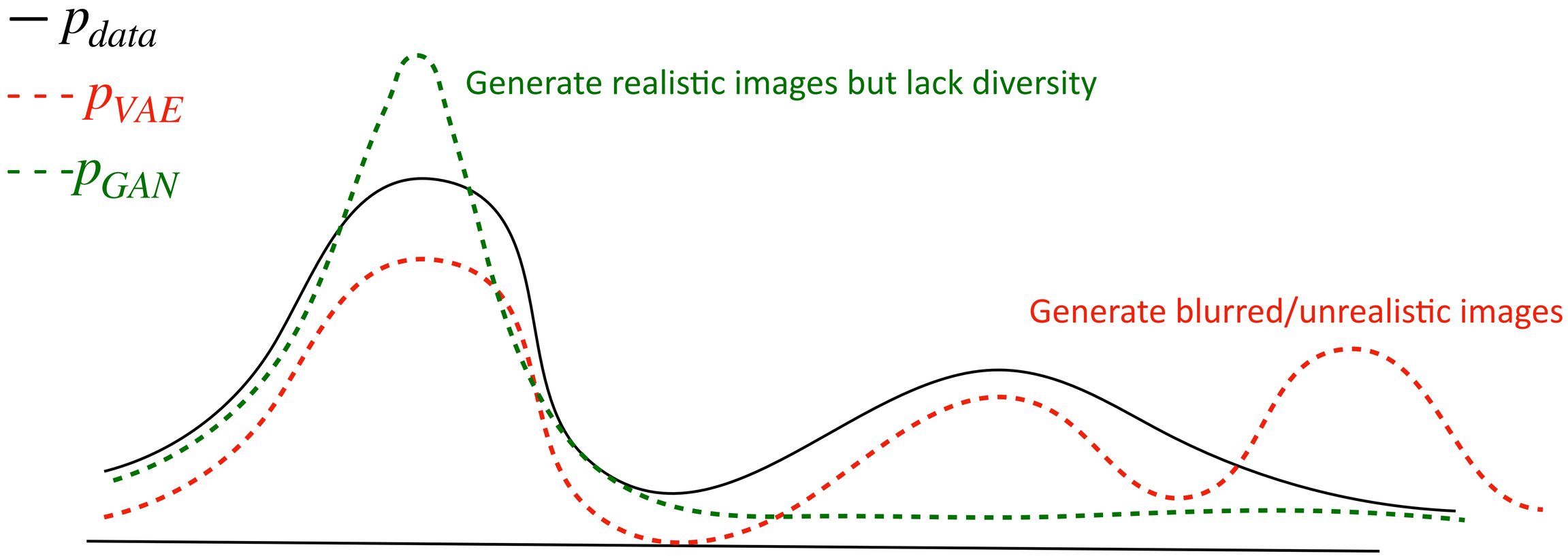
Brock et al. LARGE SCALE GAN TRAINING FOR HIGH FIDELITY NATURAL IMAGE SYNTHESIS. ICLR 2019.

Implication of the KL divergence

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

VAEs

GANs (approximately)



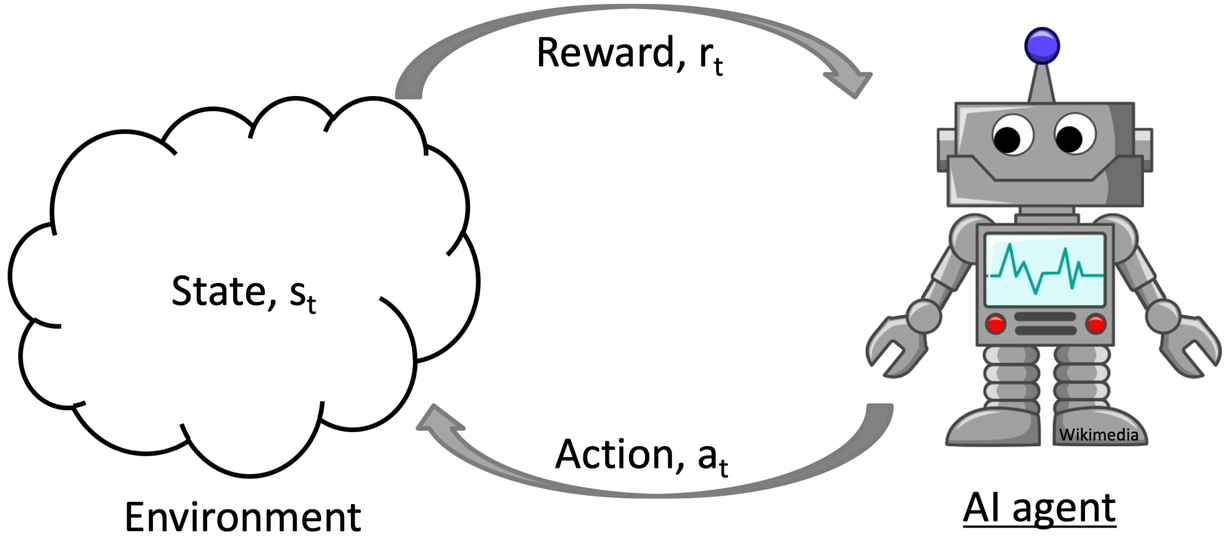
Reinforcement Learning

Learning Tasks

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

RL Setup

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



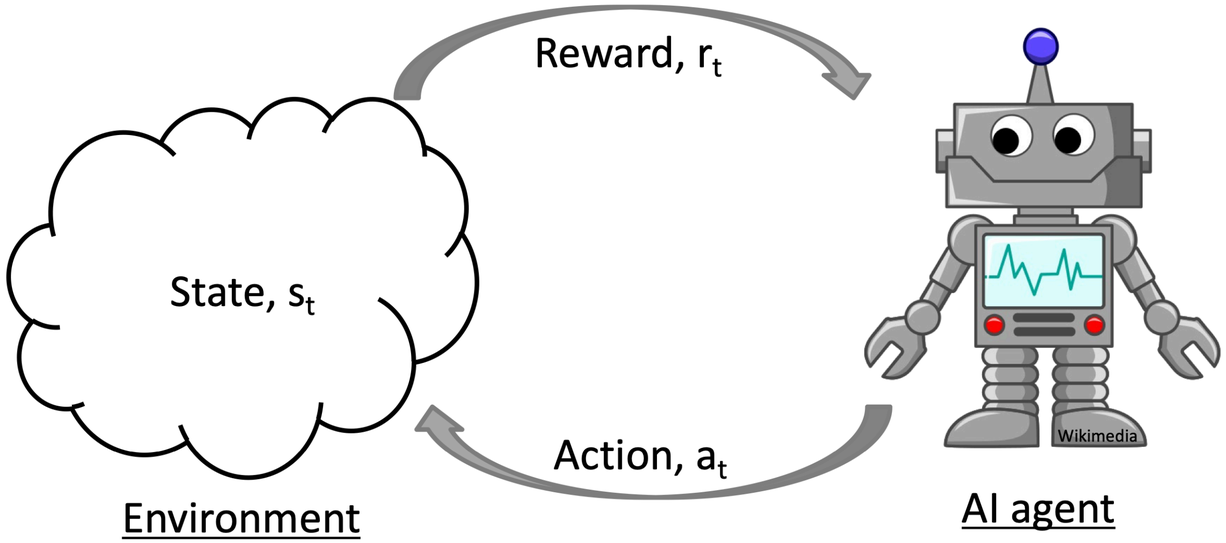
Agent chooses **actions** which can depend on past

Environment can change **state** with each action

Reward (Output) depends on (Inputs) action and state of environment

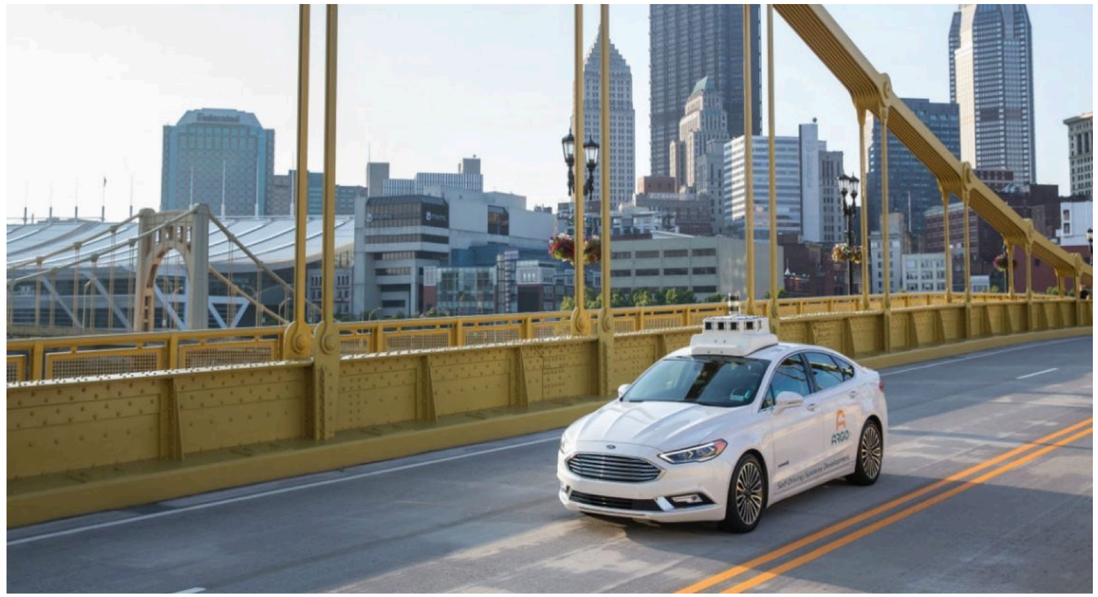
Goal: maximize the total reward

Differences from Supervised Learning



- Maximize reward (rather than learn reward) Supervised training is like imitation
- Inputs are not iid – state & action depends on past

RL Examples



RL Setup

- State space, $\mathcal{S}$
- Action space, $\mathcal{A}$
- Reward function
  - Stochastic, $p(r \mid s, a)$
  - Deterministic, $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Transition function
  - Stochastic, $p(s' \mid s, a)$
  - Deterministic, $\delta\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$

In this lecture, we assume they are known

- Reward and transition functions can be known or unknown

RL Setup

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

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

RL Example - gridworld

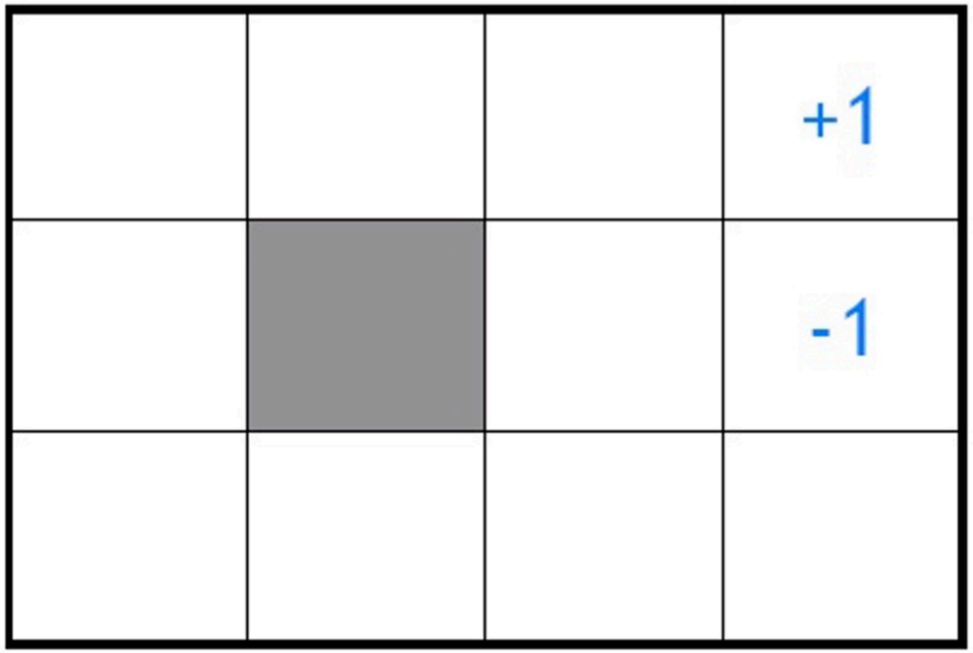
$\mathcal{S}$ = all empty squares in the grid

$\mathcal{A}$ = {up, down, left, right}

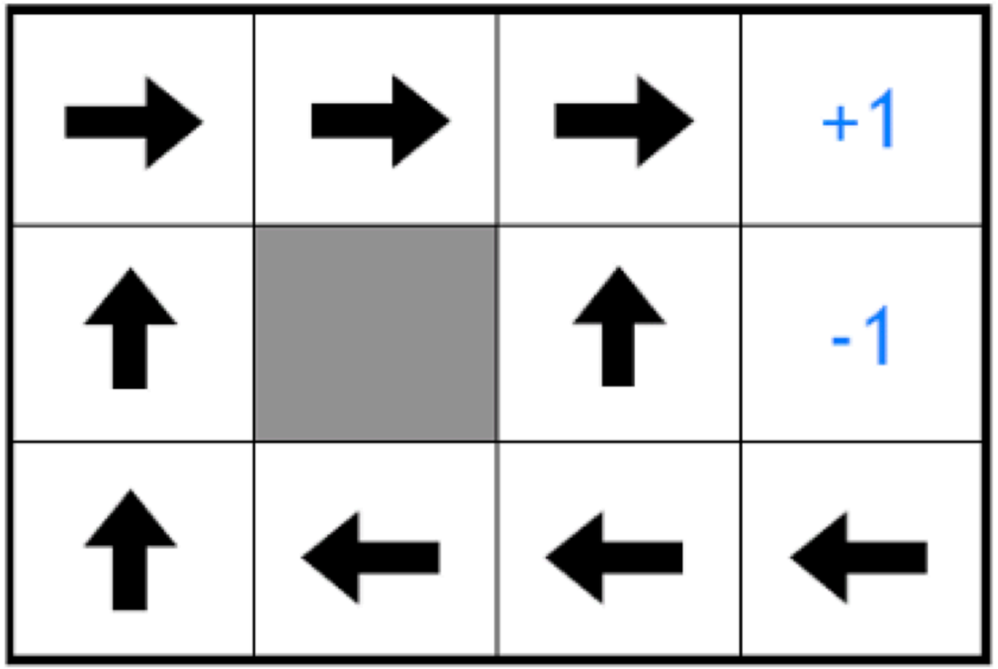
Deterministic transitions

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

Terminate after receiving either reward



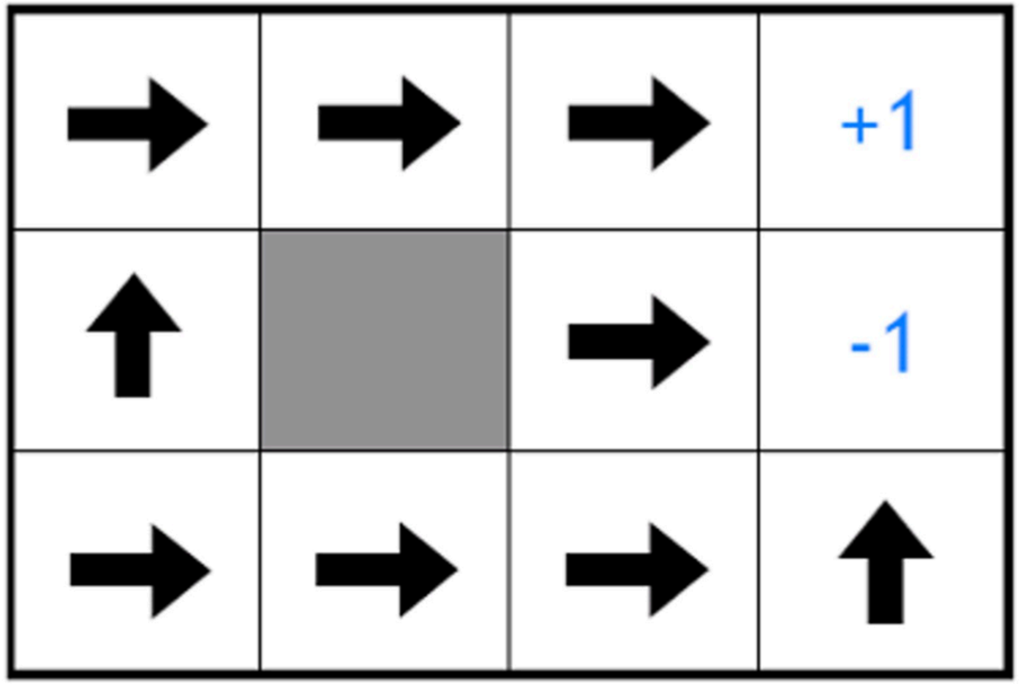
RL Example - gridworld



Is this policy optimal?

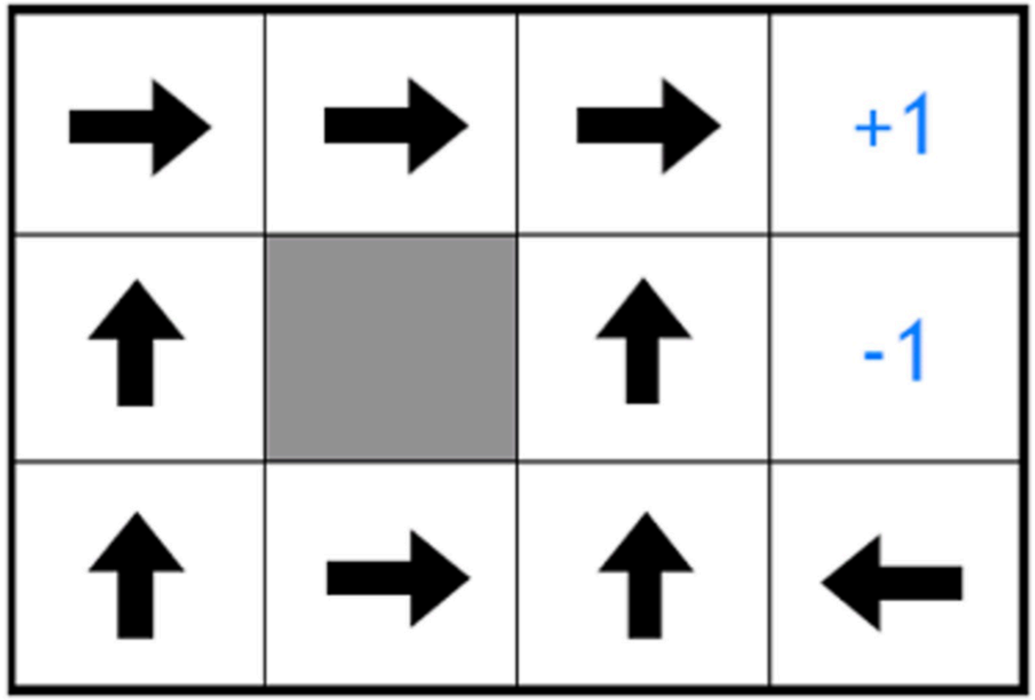
RL Example - gridworld

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 $a_t = \pi(s_t)$ Deterministic policy
   c. Agent receives reward $r_t \sim p(r \mid s_t, a_t)$
   d. Agent transitions to state $s_{t+1} \sim p(s' \mid s_t, a_t)$

- MDPs make the ***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

  $1 + 1 + 1 + \ldots = \infty$ but $1 + 0.8*1 + (0.8)^2*1 + \ldots = 5$

- 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 $\pi^* = \underset{\pi}{\text{argmax}}\ V^{\pi}(s)\ \forall\ s \in \mathcal{S}$
- $V^{\pi}(s) = \mathbb{E}[$*discounted* total reward of starting in state s and executing policy π forever$]$

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

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

Value Function

Bellman equations

$$V^{\pi}(s) = \underbrace{R\big(s, \pi(s)\big)}_{\text{Immediate reward}} + \gamma \underbrace{\sum_{s_1 \in S} p\big(s_1 \mid s, \pi(s)\big) V^{\pi}(s_1)}_{\text{Expected (Discounted) Future reward}}$$

Recursive form

Solve Value Function

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

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:

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

  - System of $|\mathcal{S}|$ equations and $|\mathcal{S}|$ variables – nonlinear!

- Optimal policy:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \underbrace{R(s,a)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')}_{\text{Expected (Discounted) Future reward}}$$

- 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

Policy Iteration

Algorithm 5 Policy Iteration

1: Initialize π randomly.
2: **for** until convergence **do**
3: Let $V := V^{\pi}$. ▷ typically by linear system solver
4: For each state s, let

$$\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s') V(s').$$

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?