Autonomous
Systems Lab

384.195 Robot Learning
Deep Reinforcement Learning Part 2

Dongheui Lee, Yashuai Yan

Overview

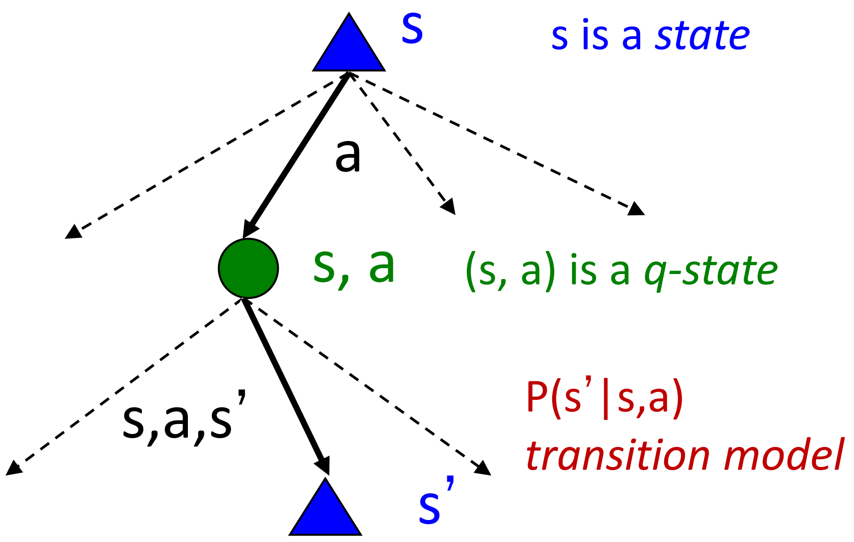
- Summary of value-based RL methods
- Policy-based RL methods
  - Policy Approximation
  - Policy Gradient Theorem
  - REINFORCE
- Actor-Critic RL methods
  - Advantage Actor Critic (A2C)
  - GAE
  - Proximal Policy Optimization (PPO)

Markov Decision Processes

- Markov decision processes:
  - Set of states S
  - Set of actions A
  - Transitions P(s’|s,a)
  - Rewards R(s,a,s’) (and discount γ)

- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards

$$V([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ \ldots$$



MDP Algorithms

- Value Iteration: Compute optimal values

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a} \sum_{s'} P(s'|s,a)V_k(s')$$

- Policy Evaluation: Compute values for a particular policy

$$V^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s,\pi(s))V^{\pi}(s')$$

- Policy Exaction: use your values to turn into a policy

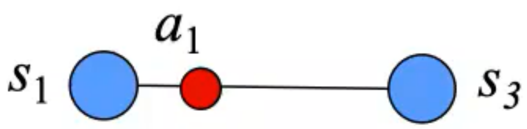
$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \left[R(s) + \gamma \sum_{s'} P(s'|s,a)V(s') \right]$$ actions that maximize Value-Fct

- Policy Improvement:

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \left[R(s) + \gamma \sum_{s'} P(s'|s,a)V^{\pi_i}(s') \right]$$ Iteration

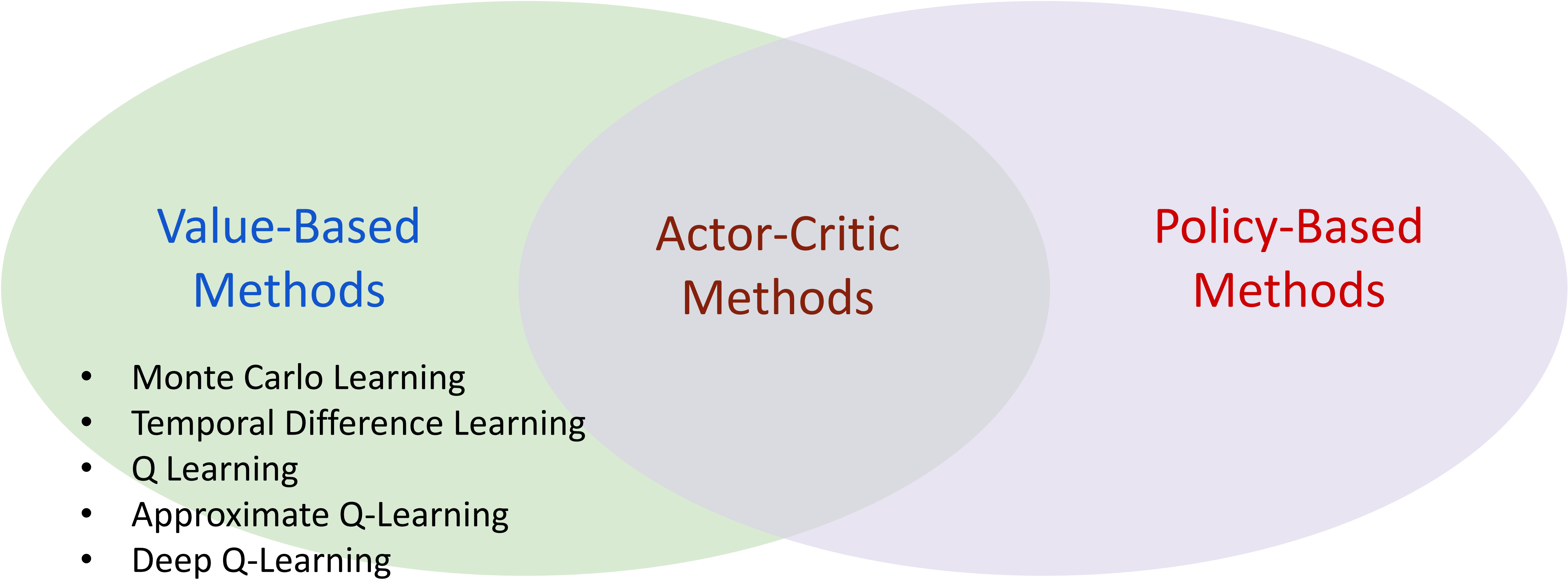
Reinforcement Learning

- RL involves making a series of optimal actions, it is considered a **sequential decision problem** and can be modeled using Markov Decision Process.
  - A set of states s $\in$ S
  - A set of actions (per state) A
  - A model P(s'|s,a)
  - A reward function R(s,a,s') (and discount γ)
  - Still looking for a policy π(s)
- Unknown model of transition P(s'|s,a) and reward R(s,a,s')
  - **Sample Backup**: the value of a state is updated from experiences collected from the interaction with the environment.



$$V(s_1) \leftarrow f\left(r(s_1, a_1) + \gamma V(s_3)\right)$$

RL methods


Value-Based Methods
Actor-Critic Methods
Policy-Based Methods
• Monte Carlo Learning
• Temporal Difference Learning
• Q Learning
• Approximate Q-Learning
• Deep Q-Learning

Monte Carlo Learning

- This is called direct evaluation or Monte Carlo on Policy Evaluation.
- Goal: learn the (state) values for each state under π
- Idea: Average together observed sample values

- It eventually computes the correct average values, using just sample transitions (unbiased estimator, but high variance)
- It must wait until the end of episode for any update. It takes long to learn.
- It wastes information about state connections. Each state must be learned separately.

Temporal Difference Value Learning

- learn from every experience of a transition (s, a, s', r)!
- Learn V(s) like Bellman Update

$$V^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s,\pi(s))V^{\pi}(s')$$

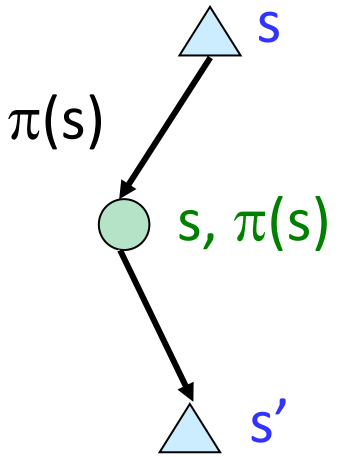
- Temporal difference learning of values
  - Move values toward the value of whatever successor occurs: running average

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

- However, if we want to turn values into a (new) policy, we're stuck:



Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_k(s',a')$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate of Q(s,a) value:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$ ← find value for next best state a'

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

- off-policy learning: Q-learning converges to optimal policy -- even if you're acting suboptimally!

From Tabular Q-Learning to Approximate Q-Learning

- Basic Q-Learning keeps a table of all q-values. In realistic situations, it may not feasible. Too many states or continuous state space.
- Approximate Q-Learning
  - Q-Learning with linear feature functions

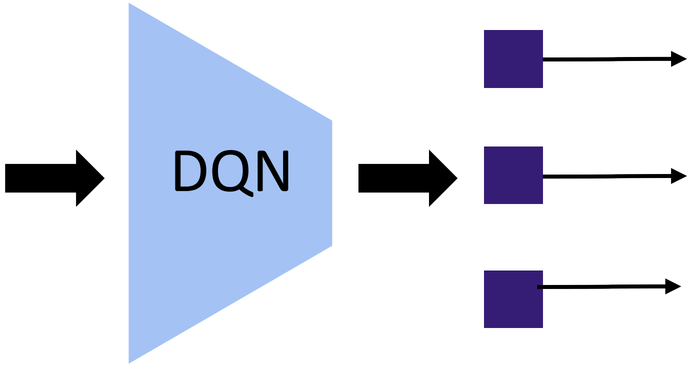
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \, [\text{difference}] \qquad \text{Exact Q's}$$

$$w_i \leftarrow w_i + \alpha \, [\text{difference}] \, f_i(s,a) \qquad \text{Approximate Q's}$$

- Deep Q Network

State s

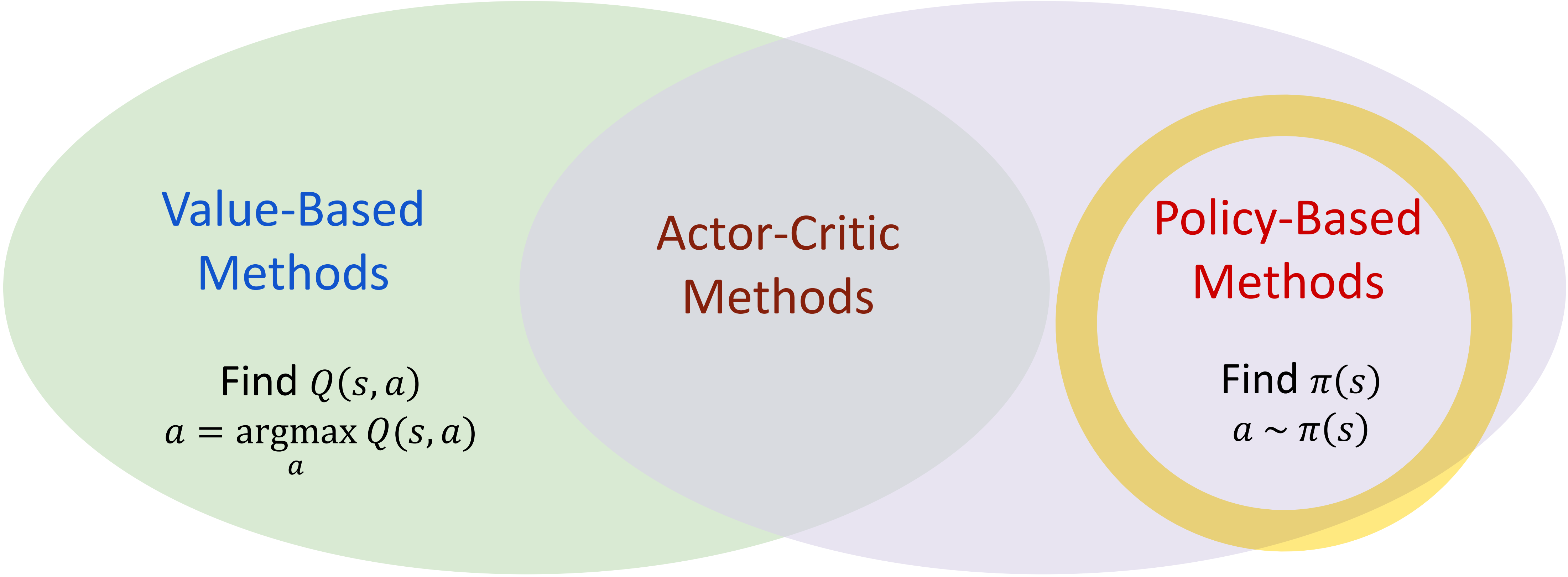


$$\hat{Q}(s, \text{"left"}; \mathbf{w})$$

$$\hat{Q}(s, \text{"right"}; \mathbf{w})$$

$$\hat{Q}(s, \text{"up"}; \mathbf{w})$$

RL methods


Value-Based Methods
Find $Q(s,a)$
$a = \underset{a}{\operatorname{argmax}}\, Q(s,a)$
Actor-Critic Methods
Policy-Based Methods
Find $\pi(s)$
$a \sim \pi(s)$

Deep Q Networks (DQN)

- Use NN to learn Q-function and then use to infer the optimal policy

find optimal action

State s

DQN

$Q(s, a_1) = 3$

$Q(s, a_2) = 35$

$Q(s, a_3) = 10$

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a) = a_2$$

deterministic approach: select action

Policy Gradient

- DQN: Approximate Q-function and use to infer the optimal policy $\pi(s)$
- Policy Gradient: Directly optimize the policy $\pi(s)$

State s

DQN

$$\sum_i P(a_i|s) = 1$$

$P(a_1|s) = 0.1$

$P(a_2|s) = 0.8$

$P(a_3|s) = 0.1$

$$\pi(s) \sim P(a|s) = a_2$$

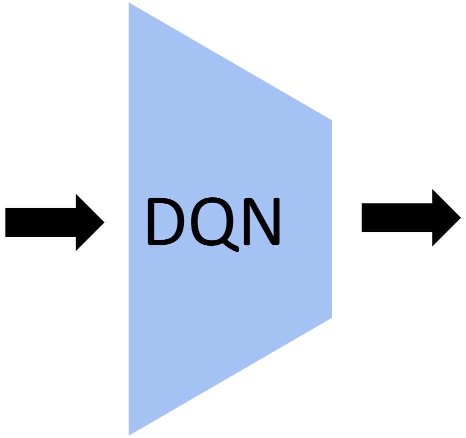
not deterministic approach: 10% a_3, 10% a_1, 80% a_2

What are the advantages of this formulation?

Discrete vs Continuous Action Spaces

- Discrete action space: which direction should I move?

State s

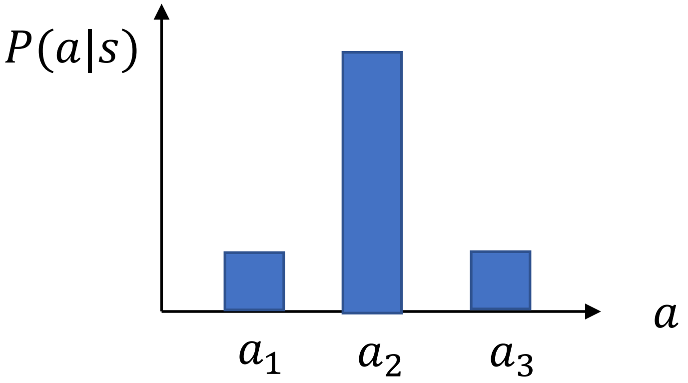


$P(a_1|s) = 0.1$

$P(a_2|s) = 0.8$

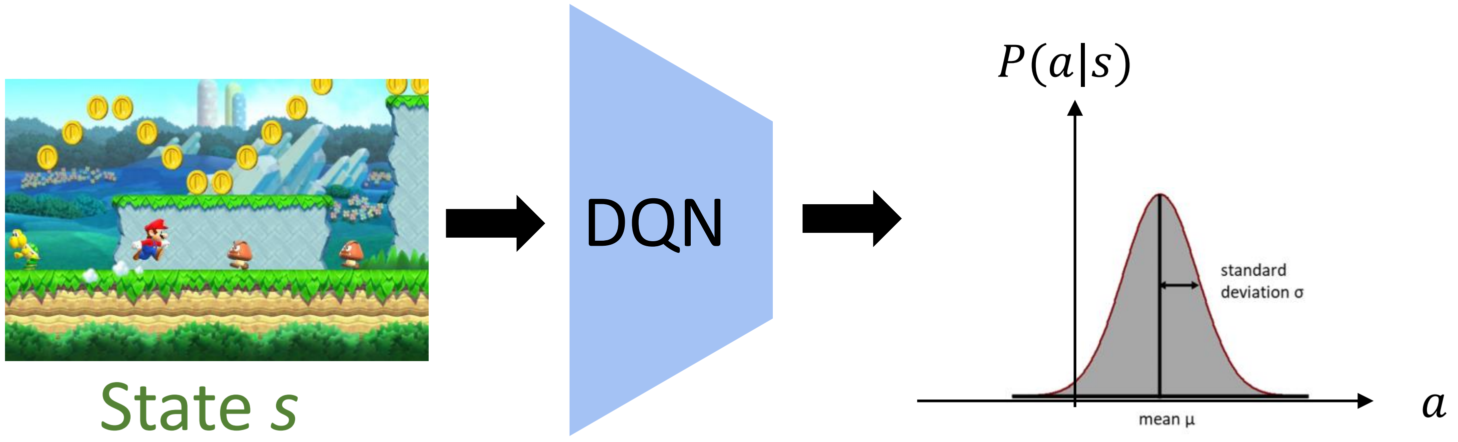
$P(a_3|s) = 0.1$

$$\sum_i P(a_i|s) = 1$$



Discrete vs Continuous Action Spaces

- Discrete action space: which direction should I move?
- Continuous action space: how fast should I move? e.g. Gaussian

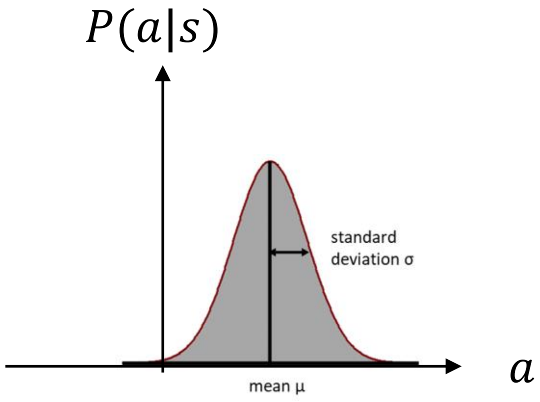


$$a \sim \mathcal{N}(\mu(s), \sigma^2)$$

Policy Gradient

- Policy Gradient: Enables modeling of continous action space

State s → DQN → Mean μ, Variance σ^2



$$a \sim \mathcal{N}(\mu(s), \sigma^2)$$

$$\int P(a|s) = 1$$

Downsides of value based reinforcement learning (e.g., DQN)

- $V(s)$ does not prescribe actions
  - Need dynamics model and compute 1 bellman back-up.
- $Q(s,a)$ needs to be able to efficiently solve $\underset{a}{\operatorname{argmax}}\, Q(s,a)$
  - Complexity: Challenge for continuous & high-dimensional action spaces
- Flexibility
  - Policy is deterministically computed from the Q function by maximizing the reward → cannot learn stochastic policies

To address these, consider a new class of RL training algorithms:
Policy gradient methods

- Often π can be simpler than $Q(s,a)$ or $V(s)$

Advantages of Policy-based RL

- Better convergence properties
- Efficient in high-dimensional or continuous action spaces
- Can learn stochastic policies

Stochastic Policy

Probability to take action a in state s.

- Consider stochastic $\pi_\theta(a|s) = \pi(a|s;\theta) = P(a|s;\theta)$ parametrized by θ.
- Finitely many discrete actions

  Softmax $\pi_\theta(a|s) = \frac{\exp(h(s,a;\theta))}{\sum_{a'} \exp(h(s,a';\theta))}$

  ($\frac{a}{\sum_{a'} a'}$ $\frac{\text{value for action}}{\text{all actions}}$)

  where $h(s,a;\theta)$ might be

  linear $h(s,a;\theta) = \sum_i \theta_i f_i(s,a)$

  or **non-linear** $h(s,a;\theta) = \text{NeuralNet}(s,a;\theta)$

- Continuous actions:

  Gaussian $\pi_\theta(a|s) = N(a|\mu(s;\theta), \Sigma(s;\theta))$

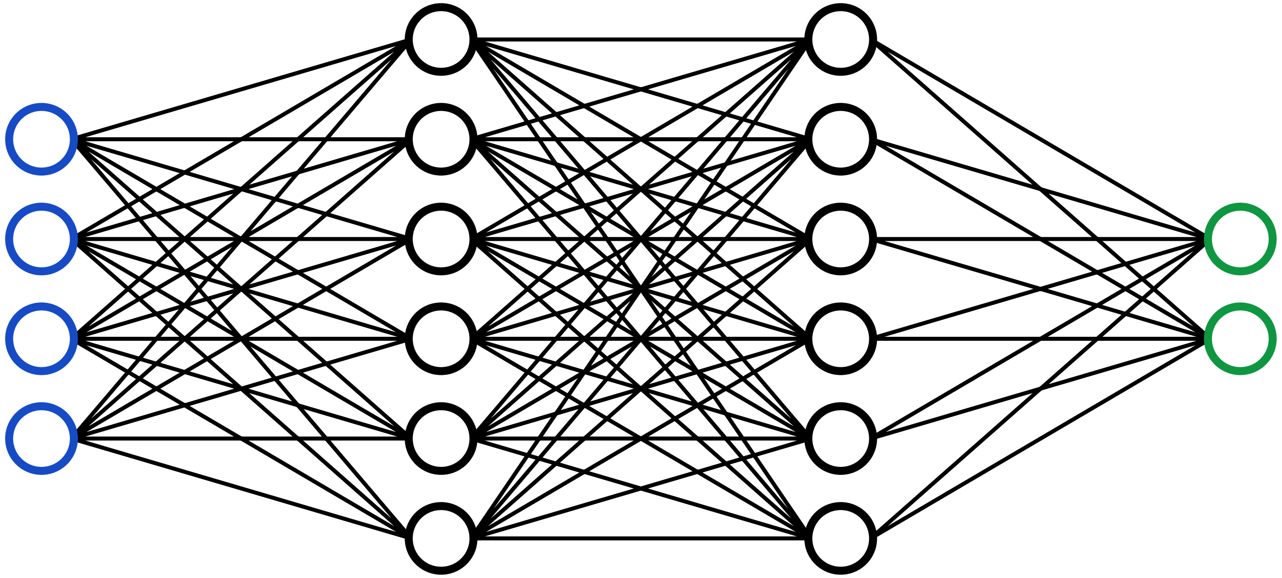
Intuition of Policy Gradient RL

Case study – Autonomous Driving Vehicle

- Agent: Vehicle
- Rewards: distance traveled

State

Camera
GPS
Lidar
…



Action

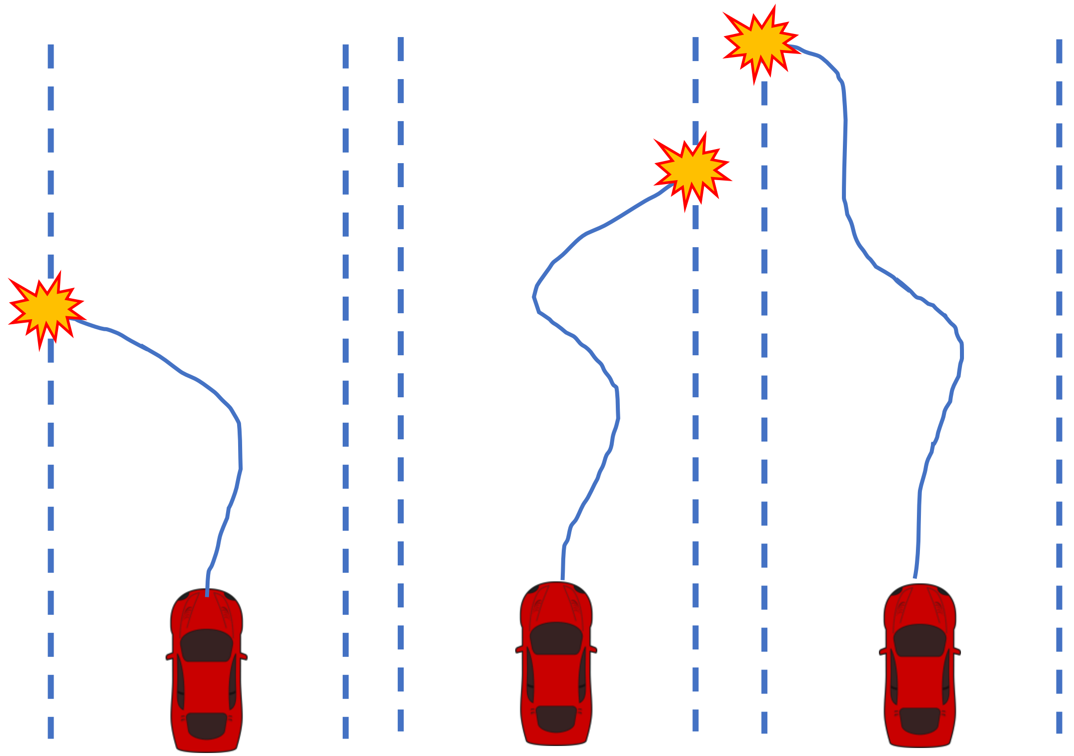
Steering wheel angle
Acceleration
Brake

Intuition of Policy Gradient RL

Case study – Autonomous Driving Vehicle

Policy Gradient Training algorithm

- Initialize the agent
- Run a policy until termination
- Record all states, actions, rewards
- Decrease probability of actions that resulted in low reward → actions near to crash
- Increase probability of actions that resulted in high rewards → actions far away from crash



How to improve the policy?

perf. measure for recorded policy $\pi(\theta)$

- Assume an episodic case. We can define the performance measure $J(\theta)$ as the value of the start state of the episode.

How to update θ? policy gradient ascent. $\theta \leftarrow \theta + \alpha \nabla J(\theta)$

$$\theta \leftarrow \theta + \alpha \cdot \boxed{\frac{\partial V(s;\theta)}{\partial \theta}}$$

learning rate policy gradient

How to get performance?

- Challenge: The performance depends on both action selection and the distribution of states (in which those selections are made). Both are affected by the policy parameter.
- How can we estimate the performance gradient wrt the policy parameter when the gradient depends on the unknown effect of policy changes on the state distribution?

Policy Gradient Theorem

Policy gradient: Derivative of $V(s;\theta)$ w.r.t. θ .

- $$\frac{\partial V(s;\theta)}{\partial\theta} = \frac{\partial \sum_a \pi(a\,|\,s;\theta)\cdot Q_\pi(s,\,a)}{\partial\theta}$$

no θ parameter

$$= \sum_a \frac{\partial \pi(a\,|\,s;\theta)\cdot Q_\pi(s,\,a)}{\partial\theta}$$

(gradient of sum is sum of gradient)

$$= \sum_a \frac{\partial \pi(a\,|\,s;\theta)}{\partial\theta}\cdot Q_\pi(s,\,a)$$

$\cdot \frac{\pi(a,s)}{\pi(a,s)}$

(multiplz by one)

(by property of the gradient of log)

$$= \sum_a \pi(a\,|\,s;\theta)\boxed{\frac{\partial \log \pi(a\,|\,s;\theta)}{\partial\theta}}\cdot Q_\pi(s,\,a)$$

$$\frac{\partial \log \pi(\theta)}{\partial\theta} = \frac{1}{\pi(\theta)}\cdot\frac{\partial \pi(\theta)}{\partial\theta}$$

$$= \mathbb{E}_{A\sim\pi(\cdot\,|\,s;\theta)}\left[\frac{\partial \log \pi(A\,|\,s;\theta)}{\partial\theta}\cdot Q_\pi(s,\,A)\right]$$

(by definition of the expectation)

Policy Update using Policy Gradient Estimate

- **Policy update** $\theta \leftarrow \theta + \alpha \cdot \frac{\partial V(s;\theta)}{\partial \theta}$ gradient ascent

- **Policy Gradient**: Derivative of $V(s;\theta)$ w.r.t. θ.

$$\frac{\partial V(s;\theta)}{\partial \theta} = \underline{\mathbb{E}_{A\sim\pi(\cdot\,|\,s;\theta)}}\left[\frac{\partial \log \pi(A\,|\,s;\theta)}{\partial \theta} \cdot Q_\pi(s,\ A)\right]$$

computing expectation directly is usually infeasible.

How to calculate policy gradient?

- Monte-Carlo Estimation: estimate expectation from random samples.

Monte Carlo Policy Gradient (REINFORCE)

REINFORCE(s_0, π_θ)

Initialize π_θ to anything

Loop forever (for each episode)

Generate episode $s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T$ with π_θ

Loop for each step of the episode $n = 0, 1, \ldots, T$

Q-fct. → $\boxed{G_n \leftarrow \sum_{t=0}^{T-n} \gamma^t r_{n+t}}$ Using return as an unbiased sample of $Q^\pi(s, a)$

Update policy: $\boxed{\theta \leftarrow \theta + \alpha\, \gamma^n G_n \nabla \log \pi_\theta(a_n | s_n)}$

Return π_θ

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem

Quiz 1

1. What is the fundamental idea behind policy gradient methods in RL?
   A. Maximizing immediate rewards
   B. Minimizing action entropy
   C. Directly optimizing the policy
   D. Learning Q-values

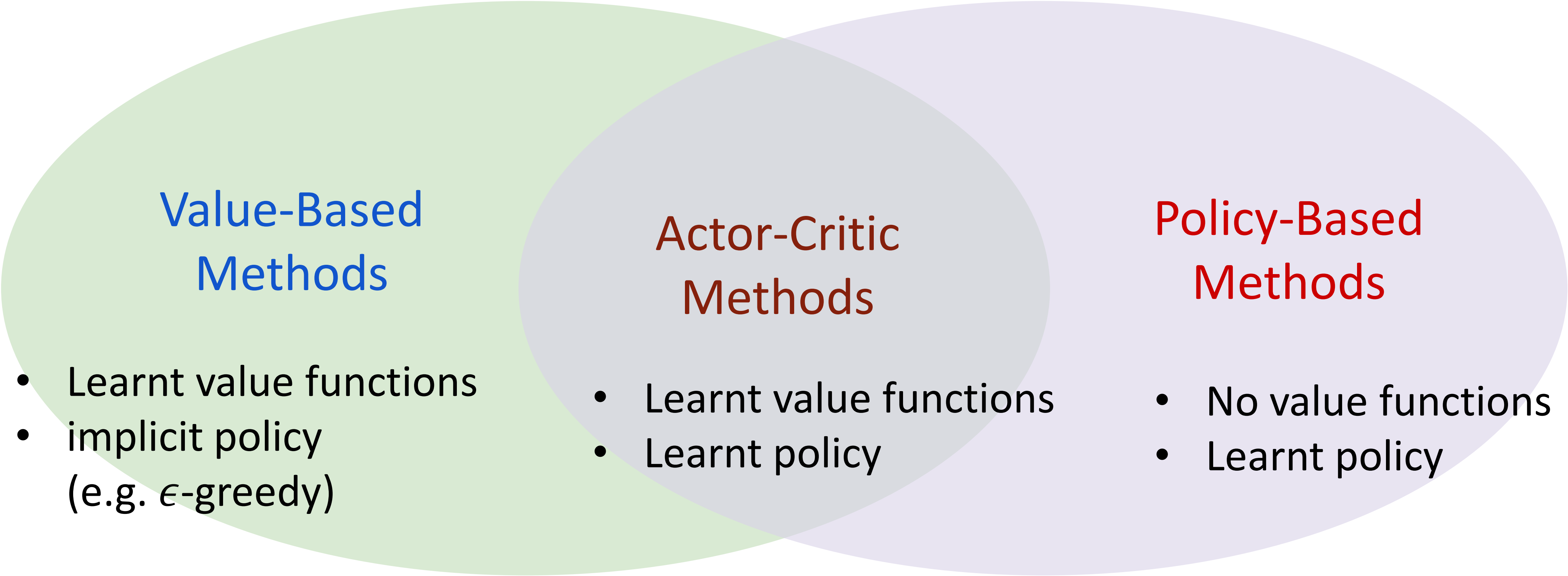
2. In policy gradient methods, what is the objective function that is typically maximized to improve the policy?
   A. Q-values
   B. Maximum likelihood of actions
   C. Expected cumulative reward
   D. Value function approximation

Quiz

3. How is the policy typically represented in policy gradient methods?
   A. As a fixed set of rules
   B. Using a decision tree
   C. As a parameterized probability distribution
   D. With a lookup table

4. How do policy gradient methods balance exploration and exploitation?
   A. By always selecting the action with highest probability
   B. By encouraging stochastic policy
   C. By using epsilon-greedy exploration
   D. By minimizing entropy of the policy

RL methods


Value-Based Methods
• Learnt value functions
• implicit policy (e.g. ϵ-greedy)
Actor-Critic Methods
• Learnt value functions
• Learnt policy
Policy-Based Methods
• No value functions
• Learnt policy

References

- Sutton and Barto, Reinforcement Learning: An Introduction, Chapter 13