Reinforcement Learning

Authors: Ta Viet Cuong, Ph. D

HMI laboratory, University of Engineering and Technology

April, 2025

Outline

1. Reinforcement Learning
2. Value and Policy Iteration
3. Deep RL

Reinforcement Learning

An agent interacts with the environment to achieve its goal

+ Agent receives an observation
+ Agent makes an action.
+ Agent receives a next observation and reward
+ And repeat …

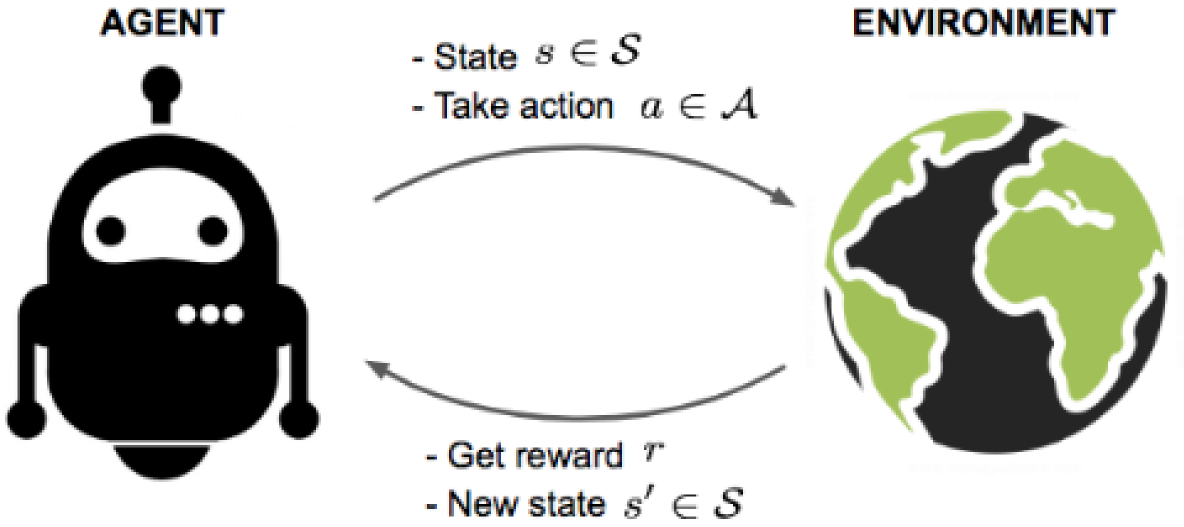


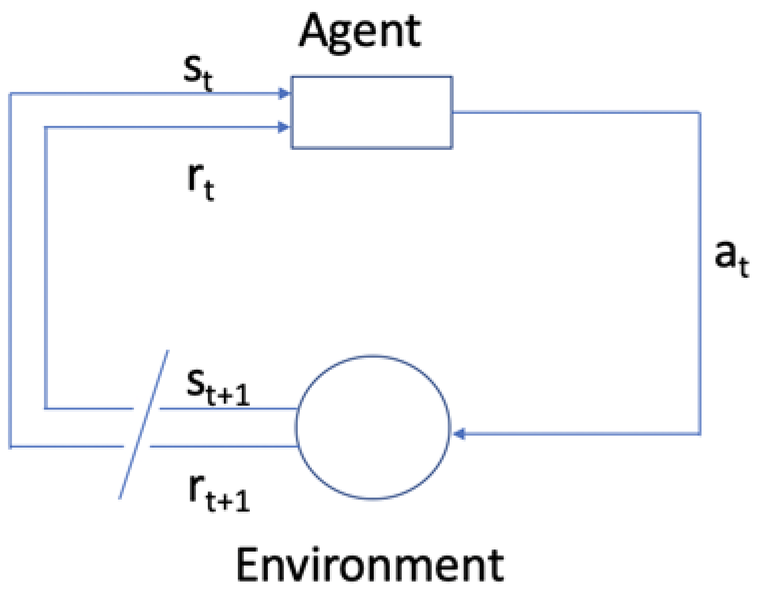
image credit to: Liliang Weng. https://lilianweng.github.io/

Reinforcement Learning: Formulation

RL uses a Markov Decision Process (MDP) definition:

$$<S, A, P, R, \gamma>$$

- S: the set of states
- A: the set of actions
- P: the transition rules, $P(s'|s, a)$
- R: the reward function, $R(s, a, s')$
- γ: Discount factor $0 < \gamma < 1$



Reinforcement Learning: Formulation

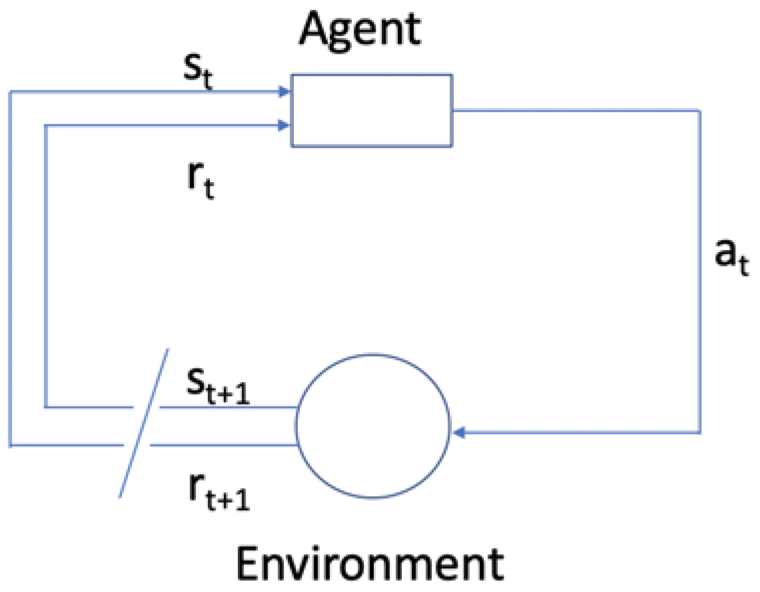
RL uses a Markov Decision Process (MDP) definition:

$$<S, A, P, R, \gamma>$$

+ S: the set of states
+ A: the set of actions
+ P: the transition rules, P(s'|s, a)
+ R: the reward function, R(s, a, s')
+ γ: Discount factor $0 < \gamma < 1$

Others:

- Initial distribution: d^0
- The set of ending state: E
- Max-time horizon: H



Reinforcement Learning: Formulation

RL uses a Markov Decision Process (MDP) definition:

$$<S, A, P, R, \gamma>$$

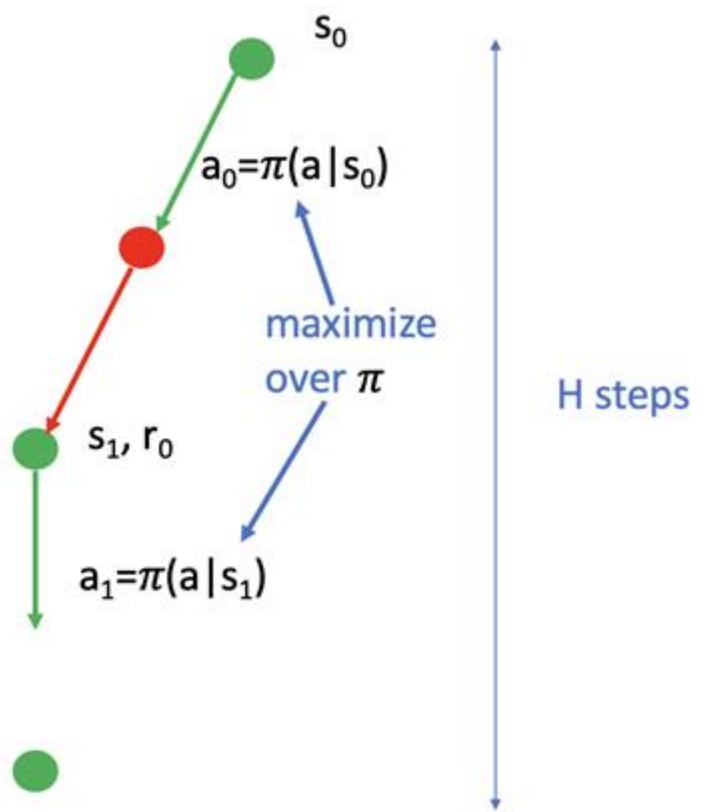
A trajectory τ is defined as with the Utility:

$$\tau = s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{H-1}, a_{H-1}, r_{H-1}, s_H(\text{terminated})$$

$$V(s_0) = U(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$$

Learning a policy π: S -> A which maximizes the expected rewards:

$$\pi^* = \max_\pi E[V(s) \mid \pi]$$



Reinforcement Learning: Formulation

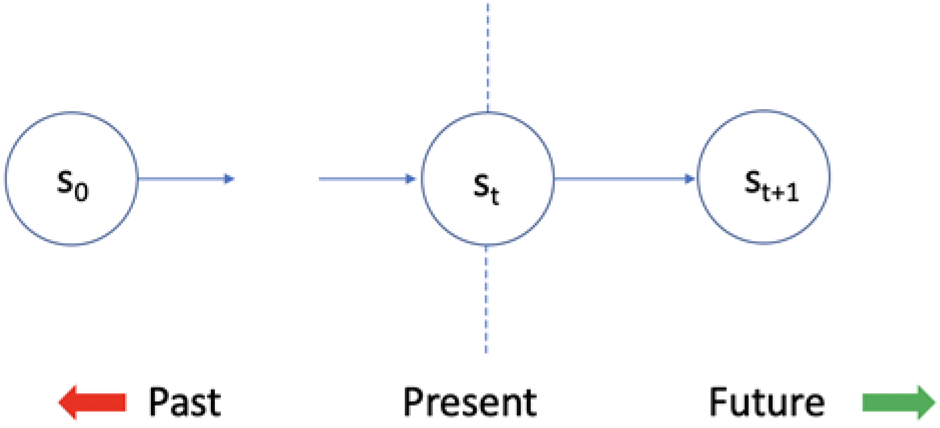
RL uses a Markov Decision Process (MDP) definition:

$$<S, A, P, R, \gamma>$$

Two assumptions:

- **MDP assumption**: $P(S_{t+1} | S_t) = P(S_{t+1} | S_t, \ldots, S_0)$

  "Future is not dependent on the Past given Present"



Reinforcement Learning: Formulation

RL uses a Markov Decision Process (MDP) definition:

$$<S, A, P, R, \gamma>$$

Two assumptions:

- **MDP assumption**: $P(S_{t+1}|S_t) = P(S_{t+1}|S_t,\ldots,S_0)$
- **Goal decomposition assumption:**

  " The goal can be achieved with a maximized reward policy"

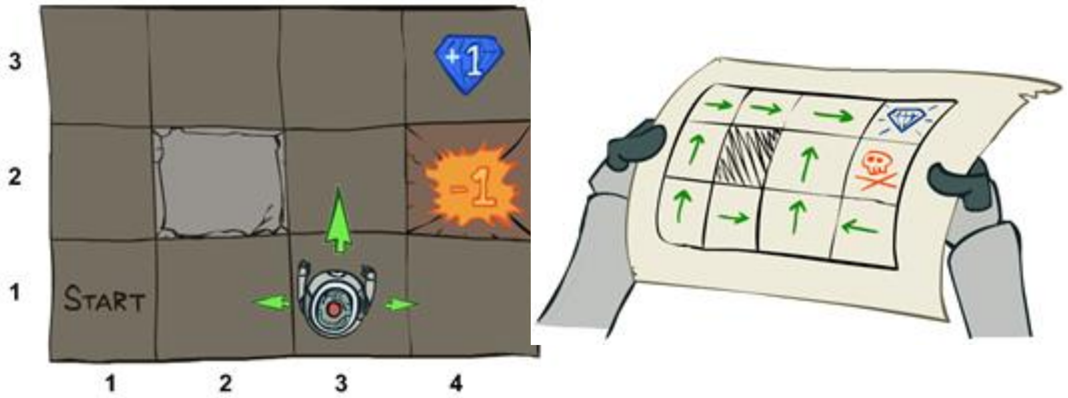


image credit to: CS188 Intro to AI at UC Berkeley (http://ai.berkeley.edu)

Reward scenario 1:
+ R(diamond) = +1
+ R(firepit) = -1
+ others = 0

Reward scenario 2:
+ R(diamond) = +1
+ R(firepit) = -1
+ others = -0.01 (Fuel Cost)

Reward scenario 3:
+ R(diamond) = +1
+ R(firepit) = -1
+ others = -0.5

Let $\gamma = 0.9$

Reinforcement Learning: Formulation

RL uses a Markov Decision Process (MDP) definition:

$$<S, A, P, R, \gamma>$$

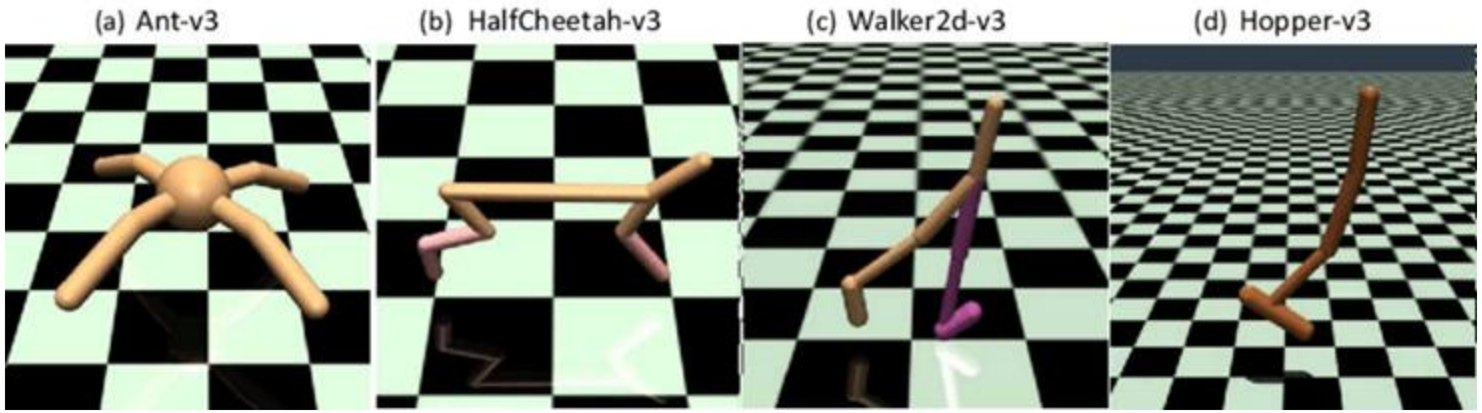
Two assumptions:

- MDP assumption
- Goal decomposition assumption:

Example problems:

- Playing games
- Robot control tasks

Atari Game - Pacman



OpenAI Gym-Mujoco

Reinforcement Learning: Formulation

RL uses a Markov Decision Process (MDP) definition:

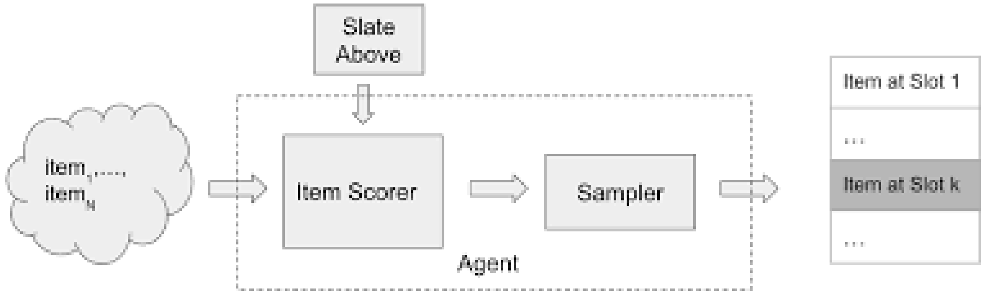
$$<S, A, P, R, \gamma>$$

Two assumptions:

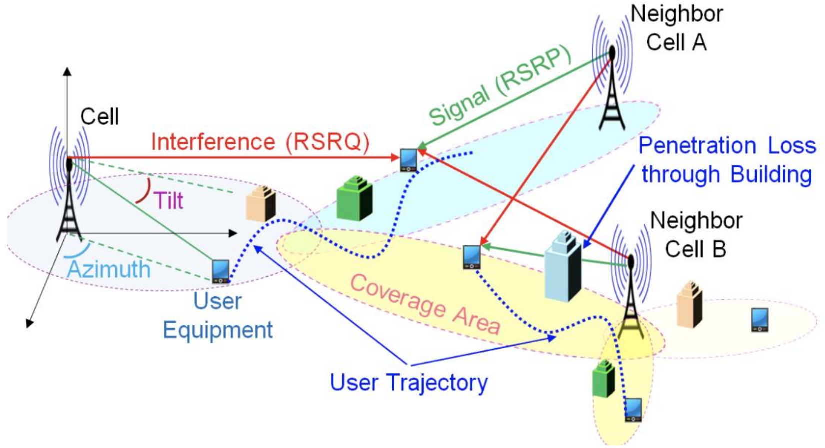
- MDP assumption
- Goal decomposition assumption:

Example problems:

- Robot control
- Playing game
- Product Recommendations
- Cell-phone coverage
- Training a “machine learning” model



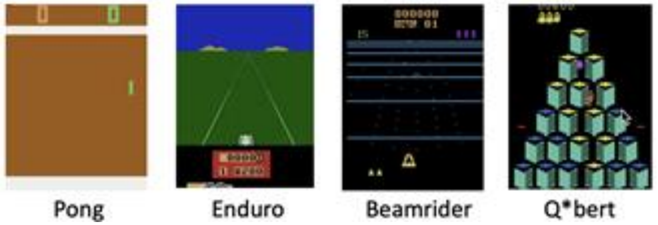
Agent for Item Ranking (image credit to: Netflix techblogs)



Cellphone control by RL
(Yongxi Tan et. al: https://arxiv.org/pdf/1802.06416.pdf)

Recent Advances in RL

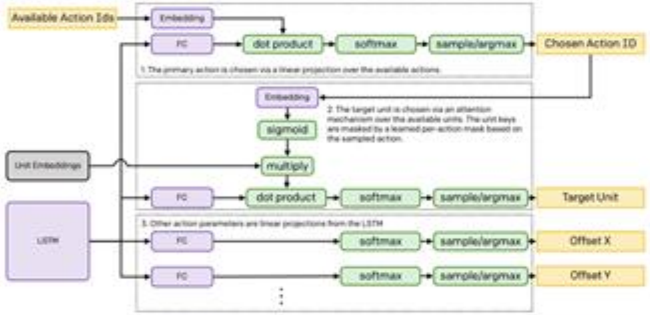
2013	Atari (DQN) [Deepmind]
2014	2D locomotion (TRPO) [Berkeley]
2015	AlphaGo [Deepmind]
2016	3D locomotion (TRPO+GAE) [Berkeley]
2016	Real Robot Manipulation (GPS) [Berkeley, Google]
2017	Dota2 (PPO) [OpenAI]
2018	DeepMimic [Berkeley]
2019	AlphaStar [Deepmind]
2019	**Rubik's Cube (PPO+DR) [OpenAI]**



Atari with Deep Q-learning (2013)

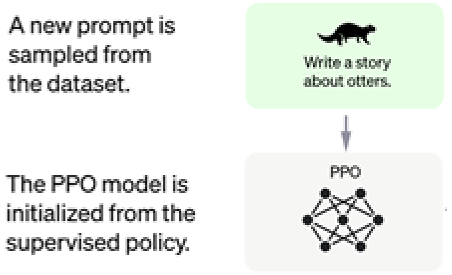
AlphaGo (2015)

OpenAI Dota2



OpenAI 5 Architecture (https://xlnwel.github.io/blog/)

(image credit to: Pieter Abeel. Foundations of Deep RL Series.)



ChatGPT=GPT3.5 +
Reinforcement Learning with Human Feedbacks

Solving RL tasks

- Exact method: Value Iteration and Policy Iteration

Solving RL: Value Iteration Methods

Exact methods: Assume |S| x |A| is tractable, and P and R is available

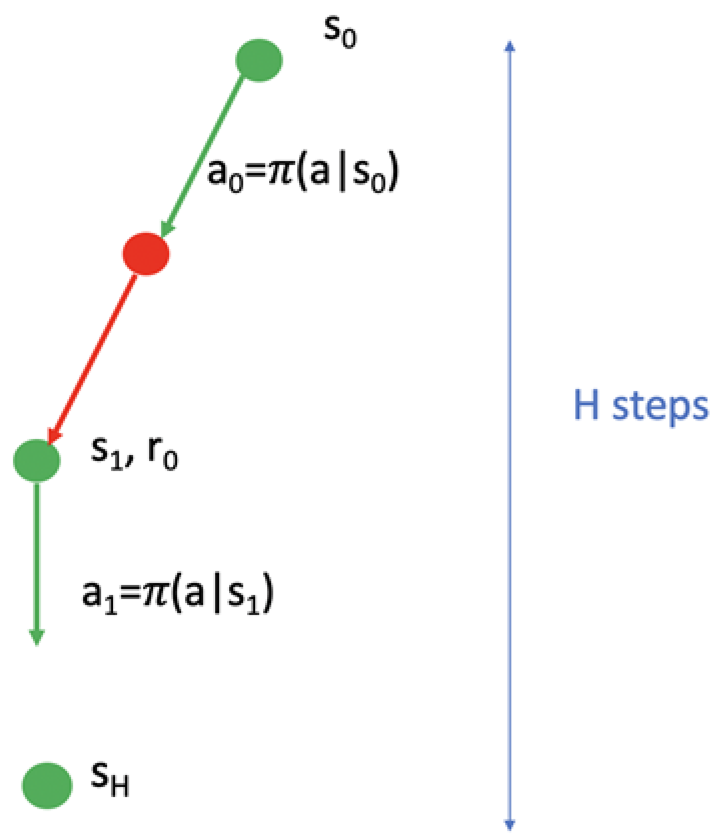
Optimal Value function is defined as:

$$V^*: S \rightarrow \mathbb{R}$$

with:

$$V^*(s) = \max_{\pi} E[U(s_0, a_0, r_0, s_1, a_1, r_1 \ldots, s_H) | \pi, s_0 = s)$$

$$U(s_0, a_0, s_1, a_1, \ldots, s_H) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$$



Solving RL: Value Iteration Methods

Exact methods: Assume |S| x |A| is tractable, and P and R is available

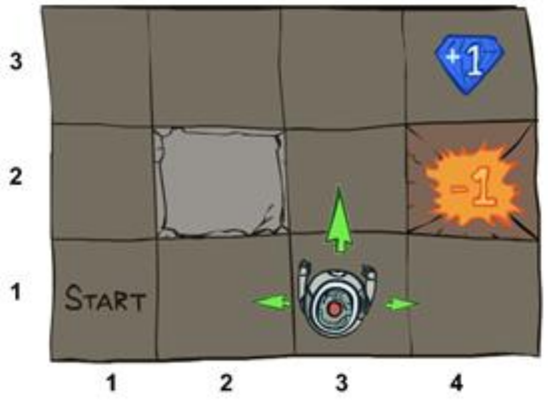
Optimal Value function is defined as:

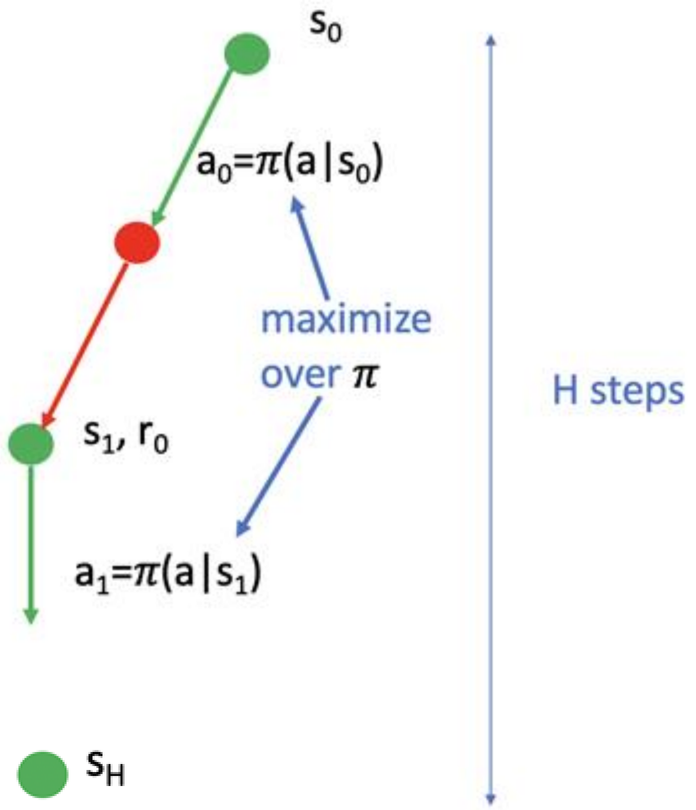
$$V^*: S \rightarrow \mathbb{R}$$

with:

$$V^*(s) = \max_{\pi} E[U(s_0, a_0, r_0, s_1, a_1, r_1 \ldots, s_H) | \pi, s_0 = s)$$

$$U(s_0, a_0, s_1, a_1, \ldots, s_H) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$$



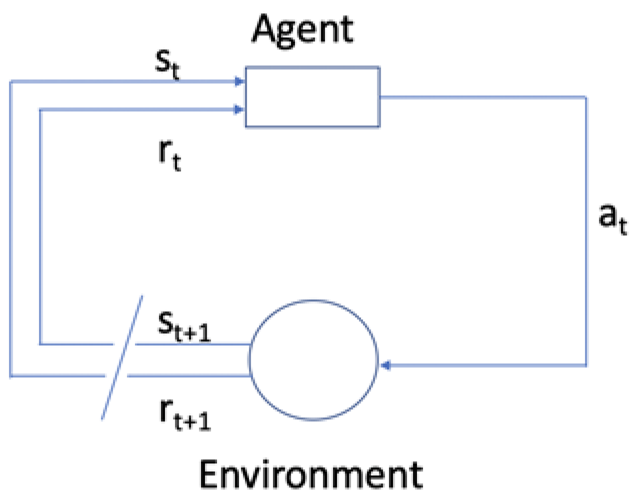


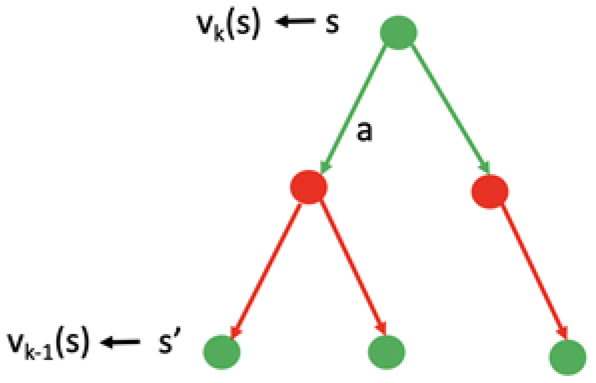
Solving RL: Value Iteration Methods

Finding V* by iterative methods:

- Start with V_0^*(s)= 0,
- Update with Bellman equation

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$





Solving RL: Value Iteration Methods

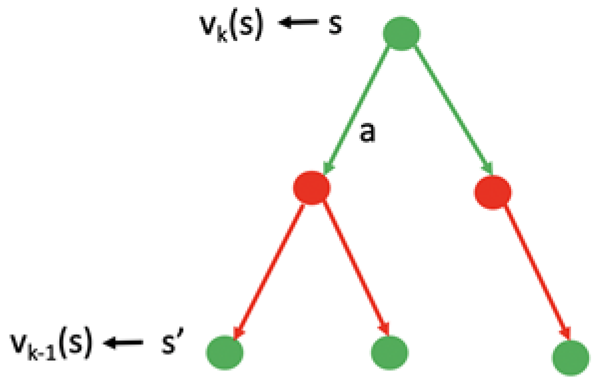
Finding V* by iterative methods:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

Convergence properties: if 0 < γ < 1, then V_k^* <- V^*

Optimal Policy: Can be extracted from V*, at any time step k

$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$



Solving RL problems: Policy Iteration

Policy Iteration: Similarly, we can define a value function associate with a policy

- Value Iteration steps:

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

(compare with Only Value)

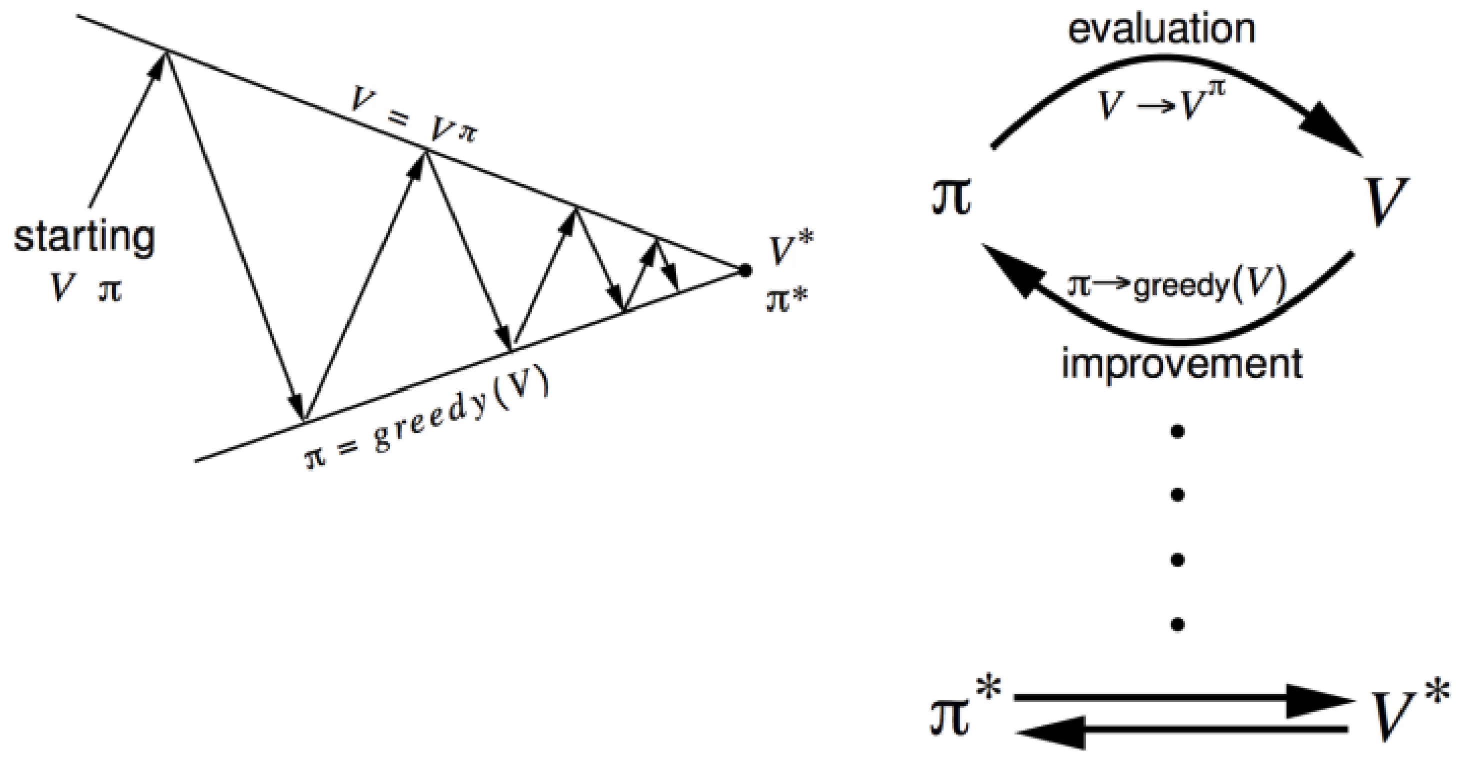
$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

- Policy Iteration steps:

$$\pi_{k+1}(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^{\pi_k}(s')\right]$$

Solving RL problems: Policy Iteration

Policy Iteration in short:



(image credit to: Deepmind. Reinforcement Learning Course Series 2021)

Deep Reinforcement Learning

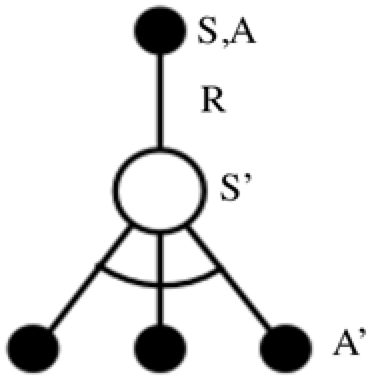
- Deep Q-Learning
- Policy Gradient
- TRPO/PPO

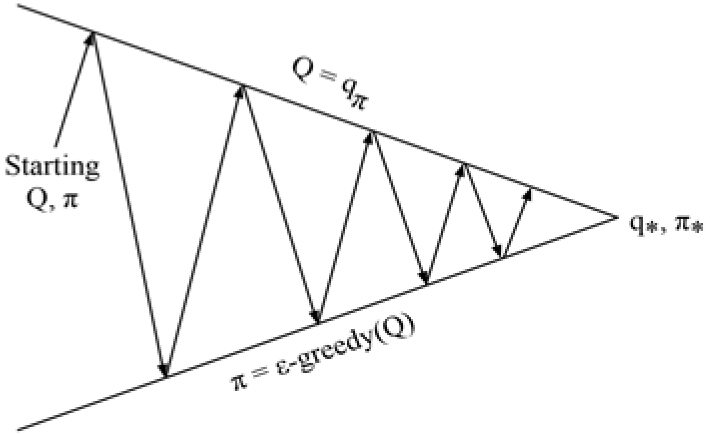
Q-learning

Q-value Iterations:

- Definitions: Q*(s, a) is the expected value starting from s do action a and acting optimally.
- Bellman equations:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$





(image credit to. Reinforcement Learning Course Series 2021)

Q-learning

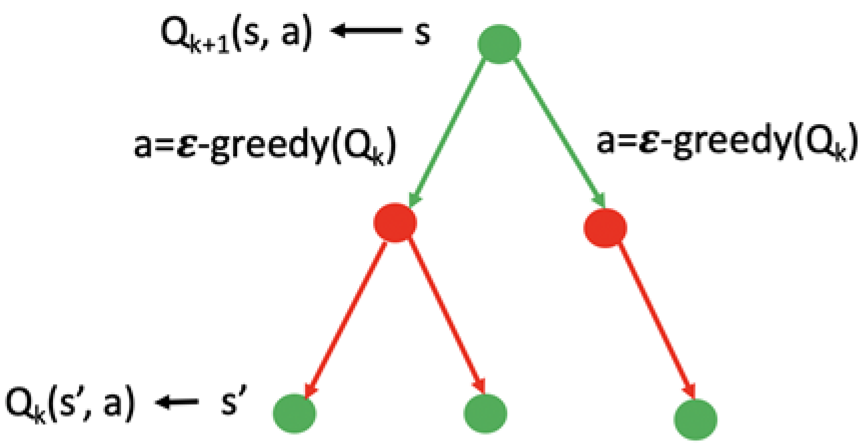
Q-value Iterations:

- Sampling based approach:

$$Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Assume an updated at state $Q_k(s, a)$ at iteration k^{th}:
  - Sample s', R(s, a, s') from ε-greedy $Q_k(s, a)$
  - Retrieve estimate $Q_k(s', a')$
  - The target value: target(s') = R(s, a, s') + γmax $Q_k(s', a')$
  - Update with a running average value α:

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\,[\text{target}(s')]$$



Approximate Q-learning

- Make a fast decision
- Generalize to similar state-action pairs

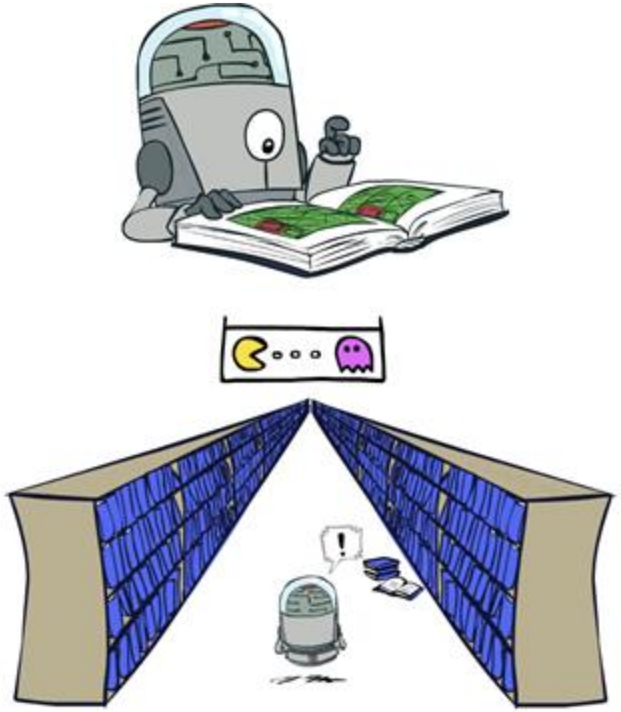
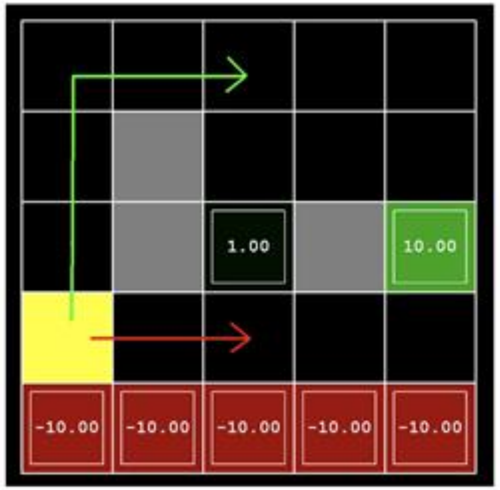


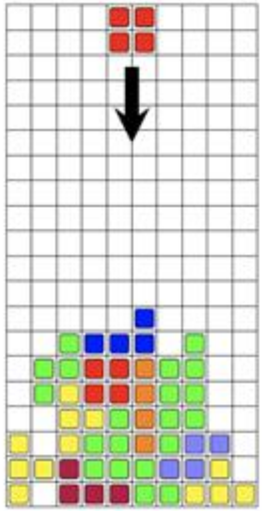
image credit to: CS188 Intro to AI at UC Berkeley (http://ai.berkeley.edu)

Approximate Q-learning

- Make a fast decision
- Generalize to similar state-action pairs



Gridworld
10^1



Tetris
10^60

Atari
10^308 (ram) 10^16992 (pixels)

(image credit to: Pieter Abeel. Foundations of Deep RL Series.)

Deep Q-learning

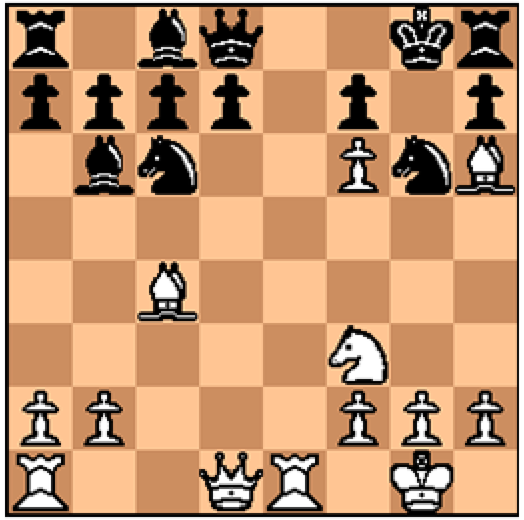
Approximate learning:

- Parameterized the Q-value by $Q_\theta(s, a)$

For examples:

- How many Queens, Knights, Rooks …
- Which are their positions …

Optimization meets Machine Learning



(image source: chessgames.com)

Deep Q-learning

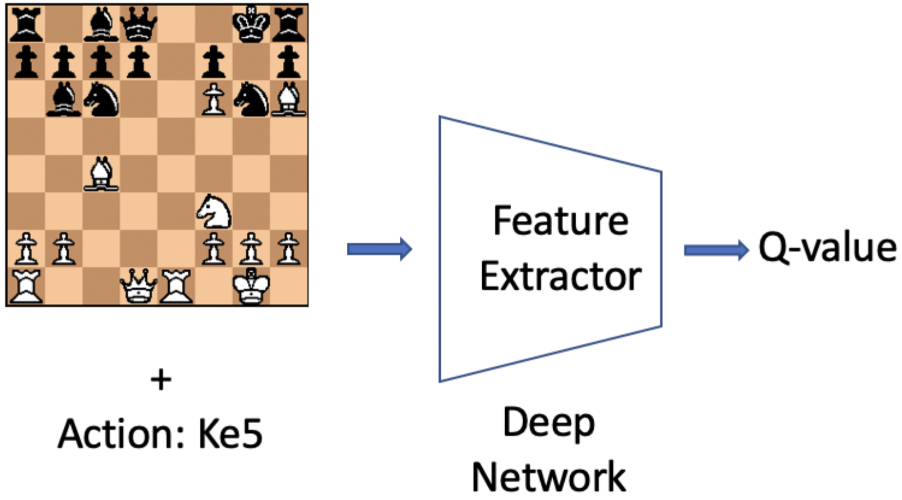
Approximate learning:

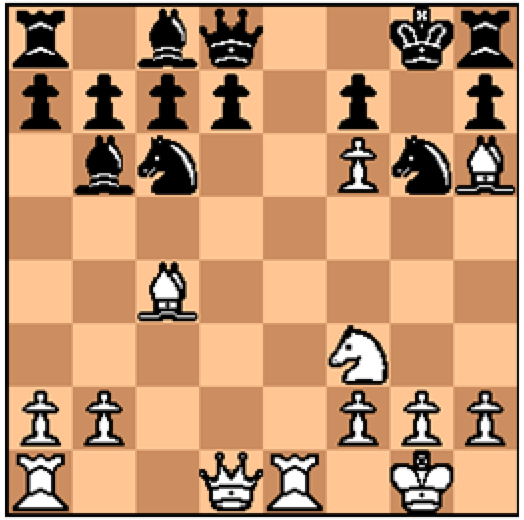
- Parameterized the Q-value by $Q_\theta(s, a)$

For examples:

- How many Queens, Knights, Rooks …
- Which are their positions …

Optimization meets Machine Learning





(image source: chessgames.com)

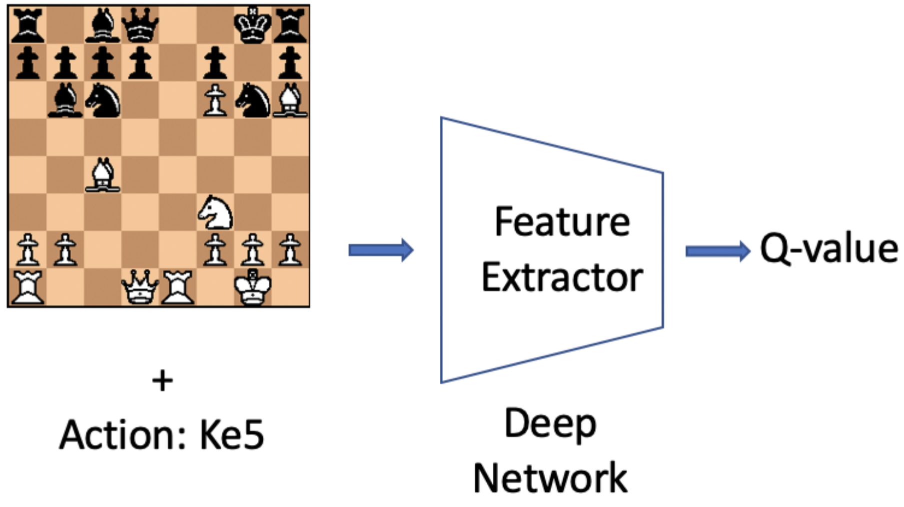
Deep Q-learning

Approximate learning:

- Parameterized the Q-value by $Q_\theta(s, a)$
- Optimization meets machine learning

What we want to achieve by θ:

- More efficient (memory, computing speeds) than store pair <s, a>
- Can make decision on key features of <s,a>
- Can have an efficient update mechanism



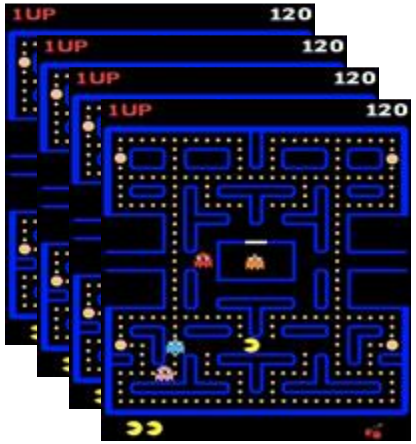
Dictionary Update: $$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left[\text{target}(s')\right]$$

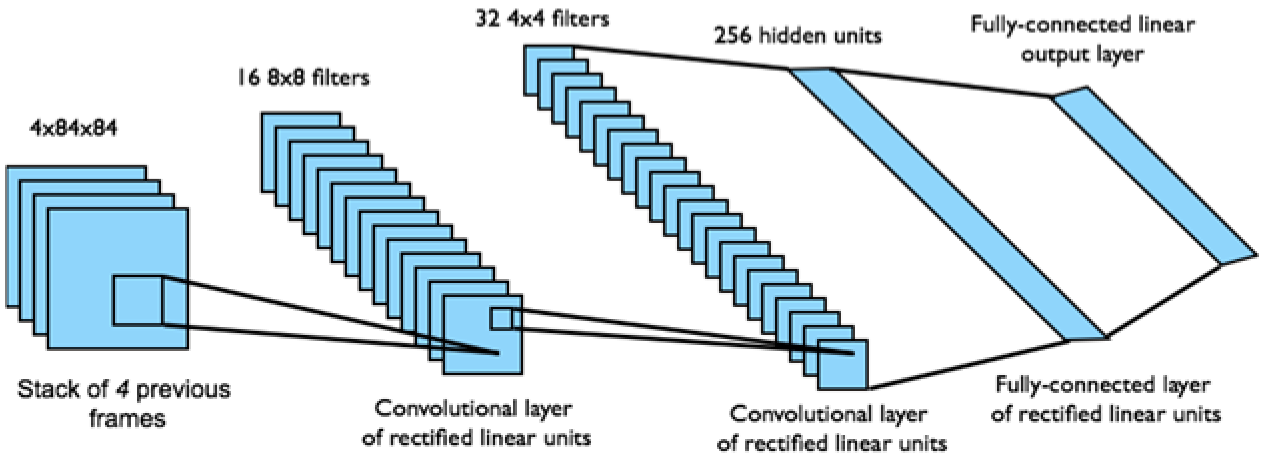
Gradient Descent Update: $$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2}(Q_\theta(s,a) - \text{target}(s'))^2\right]\Big|_{\theta=\theta_k}$$

Deep Q-learning

From the Nature paper [6]:

- End-to-end learning of values $Q_\theta(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q_\theta(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step





[6] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533

Policy Gradient methods

Policy Gradient theorem (Sutton et al. 1999): *Given the objective function as*

$$J(\pi) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t r_t\Big]$$
$$= \mathbb{E}_\pi[Q^\pi(s_0, a_0)]$$

Then the gradient of J(π) is

$$\nabla J(\pi) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t Q^\pi(s_t, a_t) \nabla \log \pi(a_t|s_t)\Big]$$

-> The policy gradient theorem provides the closed form expression for the derivative of the objective J, which allows one to directly optimize the objective by gradient ascend

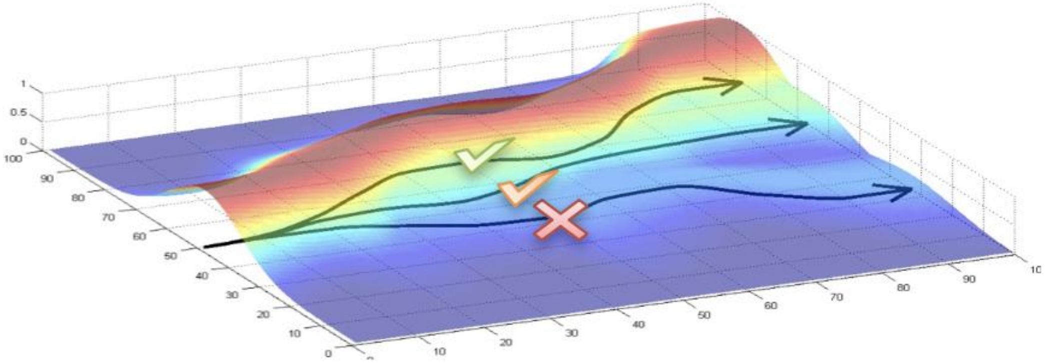
[7] Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." *Advances in neural information processing systems* 12 (1999).

Policy Gradient methods

The Policy Gradient $$\nabla J(\pi) = \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t Q^{\pi}(s_t, a_t) \nabla \log \pi(a_t | s_t)\Big]$$

How good of the Action

Probability of taking a Path



(image credit to: Pieter Abeel. Foundations of Deep RL Series.)

Can be improve by changing to: How good of the Action compare with a Baseline

In practice, the Actor-Critic employs the Advantage function:

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

Trust Region Policy Optimization (TRPO)

The difference lemma [9] tells us that the difference in performance between two any policies can be calculated as

$$J(\tilde{\pi}) - J(\pi) = \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a|s) A_{\pi}(s, a)$$

with

$$\rho_{\pi}(s) := \sum_{t=0}^{\infty} \gamma^t \mathbb{P}_{\pi}(s_t = s)$$

is called the (improper) marginal state distribution.

=> Given π, we can maximize J($\tilde{\pi}$) by maximizing

$$J(\tilde{\pi}) = J(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a|s) A_{\pi}(s, a)$$

How can we optimize this?

[8]Schulman, John, et al. "Trust region policy optimization." International conference on machine learning. PMLR, 2015.
[9] Agarwal, Alekh, et al. "Reinforcement learning: Theory and algorithms." CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep 32 (2019).

Trust Region Policy Optimization (cont)

$$J(\tilde{\pi}) = J(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$$

Difficult parts, because they rely on the samples from the distribution parameterized by the variable being optimized.

just replace $\tilde{\pi}$ with pi and wish for luck

replace with importance sampling

$$J(\tilde{\pi}) \approx L(\tilde{\pi}) := J(\pi) + \sum_s \rho_\pi(s) \mathbb{E}_{a\sim\pi} \frac{\tilde{\pi}(a|s)}{\pi(a|s)} A_\pi(s,a)$$

How good is this approximation?

Trust Region Policy Optimization (cont)

Monotonic improvement guarantee [8]:

$$J(\tilde{\pi}) \geqslant L(\tilde{\pi}) - \frac{4\gamma \max |A_\pi(s,a)|}{(1-\gamma)^2} \mathrm{D}_{KL}^{\max}(\pi, \tilde{\pi})$$

=> As the *distance* between π and $\tilde{\pi}$ *decreases*, the surrogate loss L($\tilde{\pi}$) becomes an increasingly accurate estimate of the actual performance J($\tilde{\pi}$)

=> Maximize the RHS of the above inequality, and transform it into a constrained optimization problem:

$$\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leq \delta.$$

This is TRPO's optimization objective

[8] Schulman, John, et al. "Trust region policy optimization." International conference on machine learning. PMLR, 2015.

Proximal Policy Optimization (PPO)

Proximal Policy optimization (PPO) [10] is simpler to implement than TRPO, but also has a trust-region-like mechanism.

TRPO objective

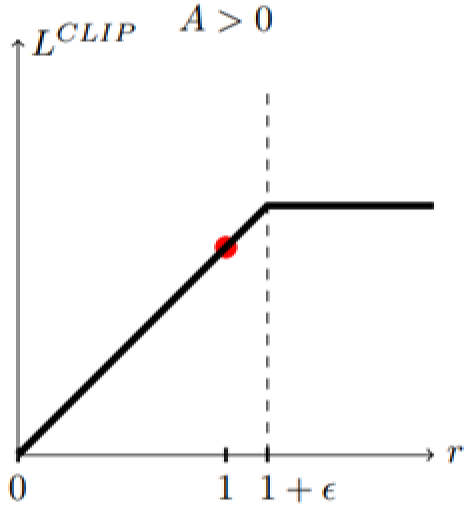
$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right]$$

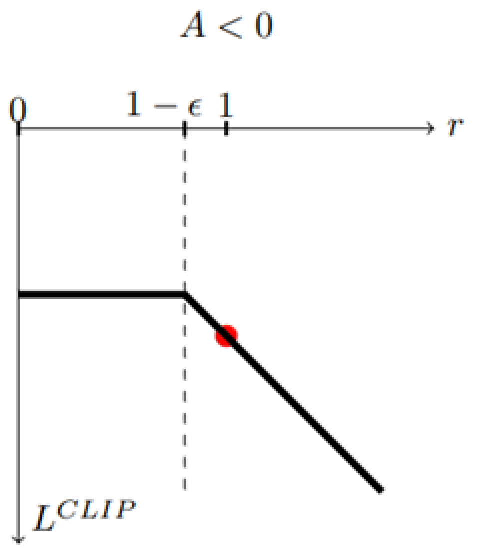
$$\text{subject to} \quad \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta.$$

PPO objective

importance sampling ratio

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$





[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017

PPO - Implementation that matters

To achieve the best performance of PPO, several small implementation details that might be overlooked are important to consider - without these, PPO can perform poorly. For example:

+ Value function clipping
+ Reward scaling
+ Orthogonal initialization

Many follow-up works of PPO use these choices as standard implementation.

See [11] for more information.

[11]Engstrom, Logan, et al. "Implementation matters in deep policy gradients: A case study on ppo and trpo." arXiv preprint arXiv:2005.12729 (2020).