

Deep Learning Optimization - REINFORCEMENT Learning

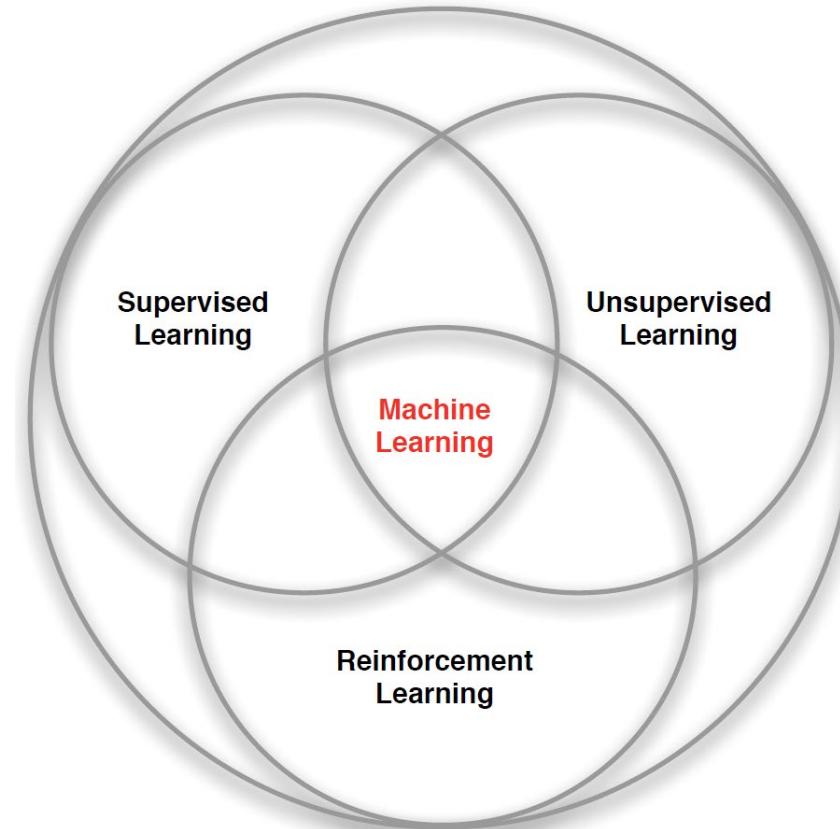
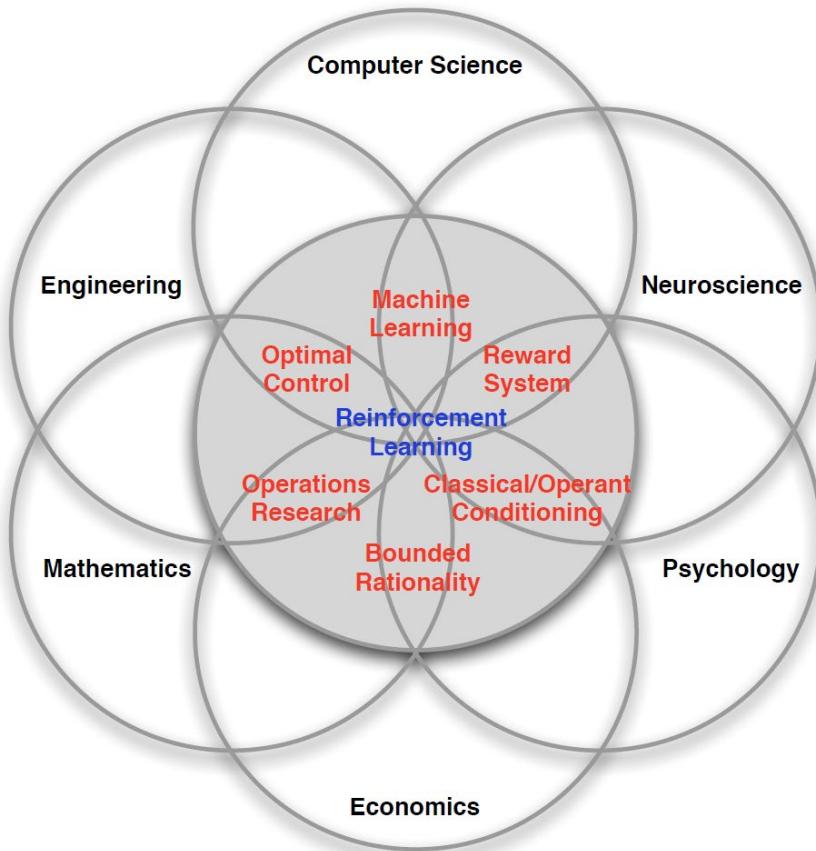
April 3, 2023

Eunhyeok Park

Notice

- This class introduces only the parts of reinforcement learning to understand the relevant optimization techniques
 - RL for discrete/continuous space optimization
 - SARSA / Q-Learning / policy gradient
 - RL covers much more general and diverse topics
 - Highly recommend you listen to the following class
 - <https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver>

REINFORCEMENT Learning



Characteristics of RL

What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a *reward* signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Keywords

- Value / Policy
- Environment
- Agent
- History H_t
- State S_t
- Reward R_t
- Action A_t
- Return G_t
- Bellman Equation
- Policy Gradient

History and State

- The **history** is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, \dots, A_{t-1}, O_t, R_t$$

- i.e. all observable variables up to time t
- i.e. the sensorimotor stream of a robot or embodied agent
- What happens next depends on the history:
 - The agent selects actions
 - The environment selects observations/rewards
- **State** is the information used to determine what happens next
- Formally, state is a function of the history:

$$S_t = f(H_t)$$

Reward

- A **reward** R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward

Reinforcement learning is based on the **reward hypothesis**

Definition (Reward Hypothesis)

All goals can be described by the maximisation of expected cumulative reward

Return

Definition

The *return* G_t is the total discounted reward from time-step t .

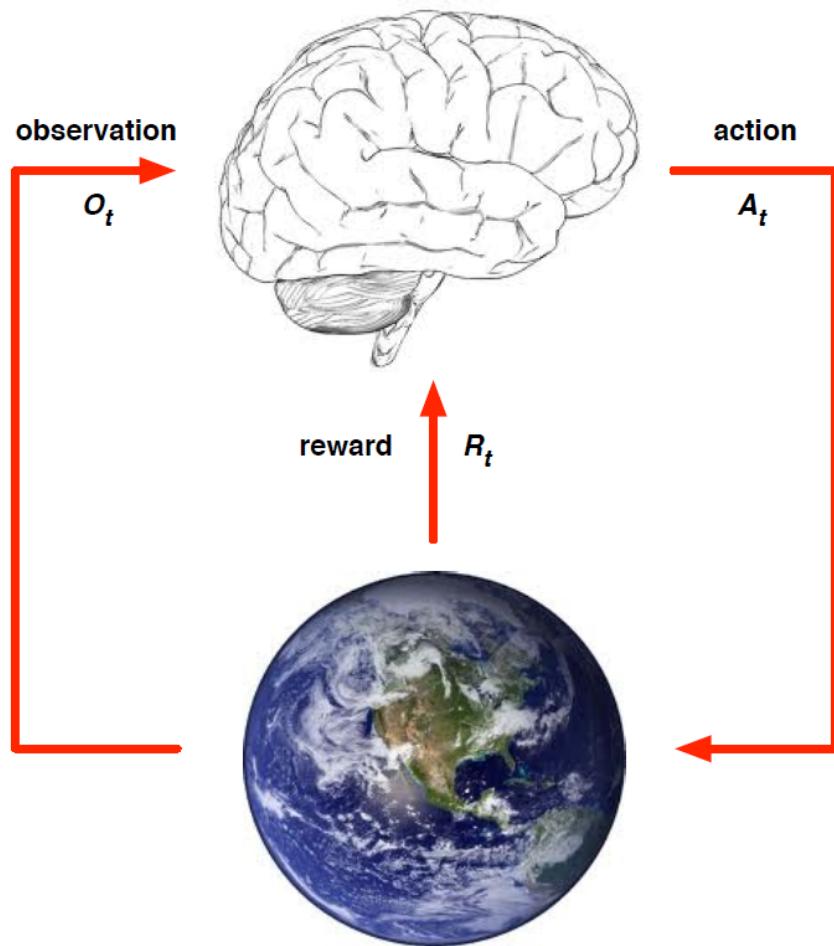
$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
 - γ close to 0 leads to "myopic" evaluation
 - γ close to 1 leads to "far-sighted" evaluation

Sequential Decision Making

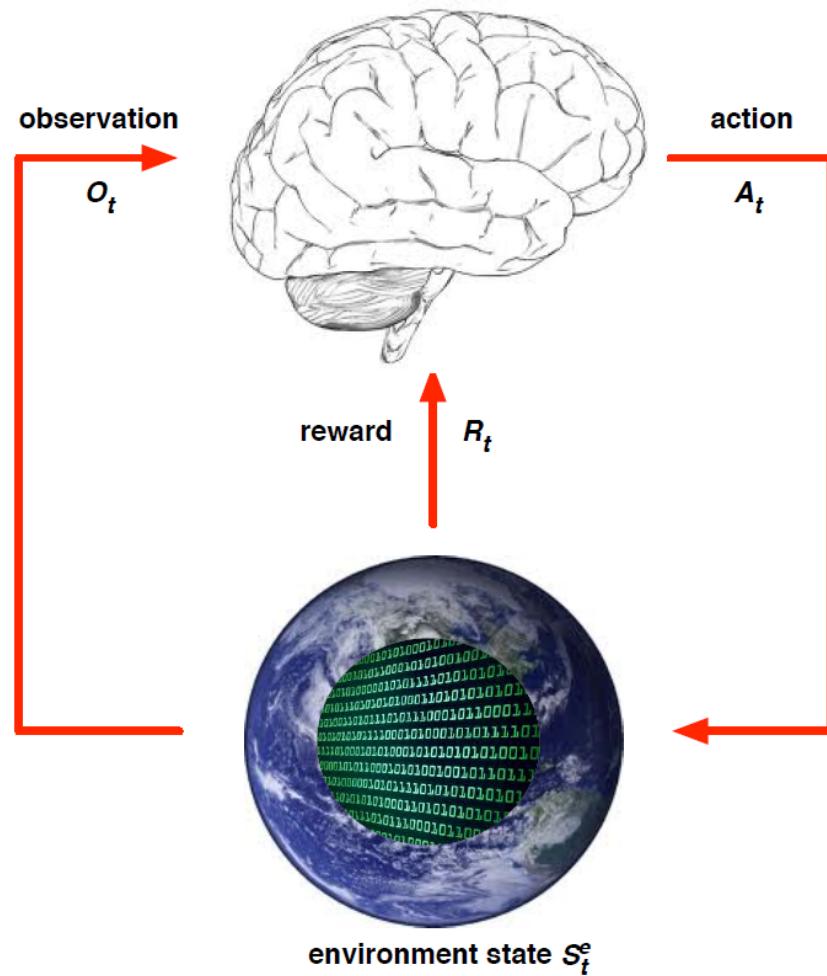
- Goal: *select actions to maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
 - A financial investment (may take months to mature)
 - Refuelling a helicopter (might prevent a crash in several hours)
 - Blocking opponent moves (might help winning chances many moves from now)

Agent and Environment



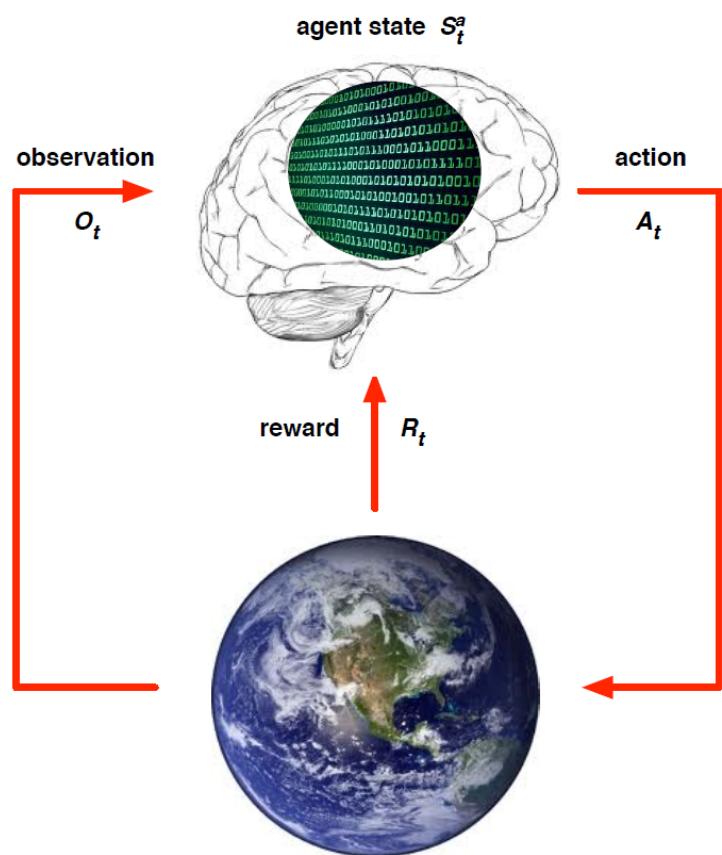
- At each step t the agent:
 - Executes action A_t
 - Receives observation O_t
 - Receives scalar reward R_t
- The environment:
 - Receives action A_t
 - Emits observation O_{t+1}
 - Emits scalar reward R_{t+1}
- t increments at env. step

Environment State



- The **environment state** S_t^e is the environment's private representation
- i.e. whatever data the environment uses to pick the next observation/reward
- The environment state is not usually visible to the agent
- Even if S_t^e is visible, it may contain irrelevant information

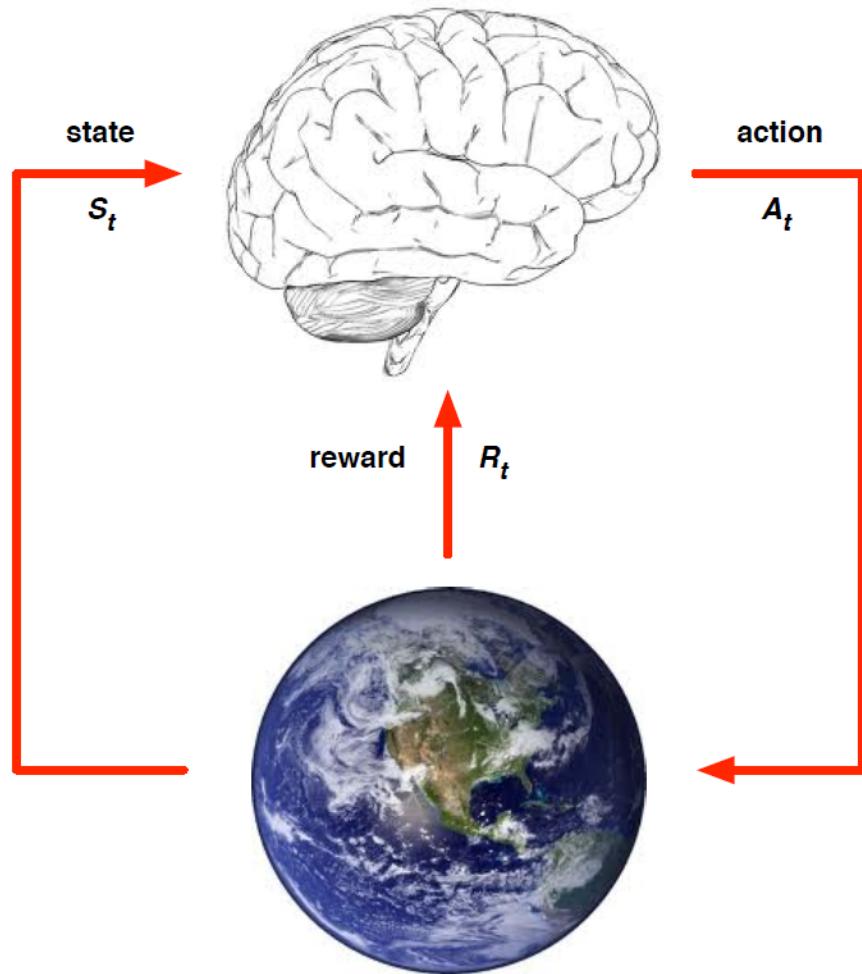
Agent State



- The **agent state** S_t^a is the agent's internal representation
- i.e. whatever information the agent uses to pick the next action
- i.e. it is the information used by reinforcement learning algorithms
- It can be any function of history:

$$S_t^a = f(H_t)$$

Fully Observable State



Full observability: agent **directly** observes environment state

$$O_t = S_t^a = S_t^e$$

- Agent state = environment state = information state
- Formally, this is a **Markov decision process** (MDP)
- (Next lecture and the majority of this course)

Exploration and Exploitation

- Reinforcement Learning:
 - The environment is initially unknown
 - The agent interacts with the environment
 - The agent improves its policy
- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way
- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is usually important to explore as well as exploit

Major Component of an RL Agent

- An RL agent may include one or more of these components:
 - Policy: agent's behaviour function
 - Value function: how good is each state and/or action
 - Model: agent's representation of the environment

Policy

- A **policy** is the agent's behaviour
- It is a map from state to action, e.g.
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

Model

- A **model** predicts what the environment will do next
- \mathcal{P} predicts the next state
- \mathcal{R} predicts the next (immediate) reward, e.g.

$$\begin{aligned}\mathcal{P}_{ss'}^a &= \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] \\ \mathcal{R}_s^a &= \mathbb{E}[R_{t+1} | S_t = s, A_t = a]\end{aligned}$$

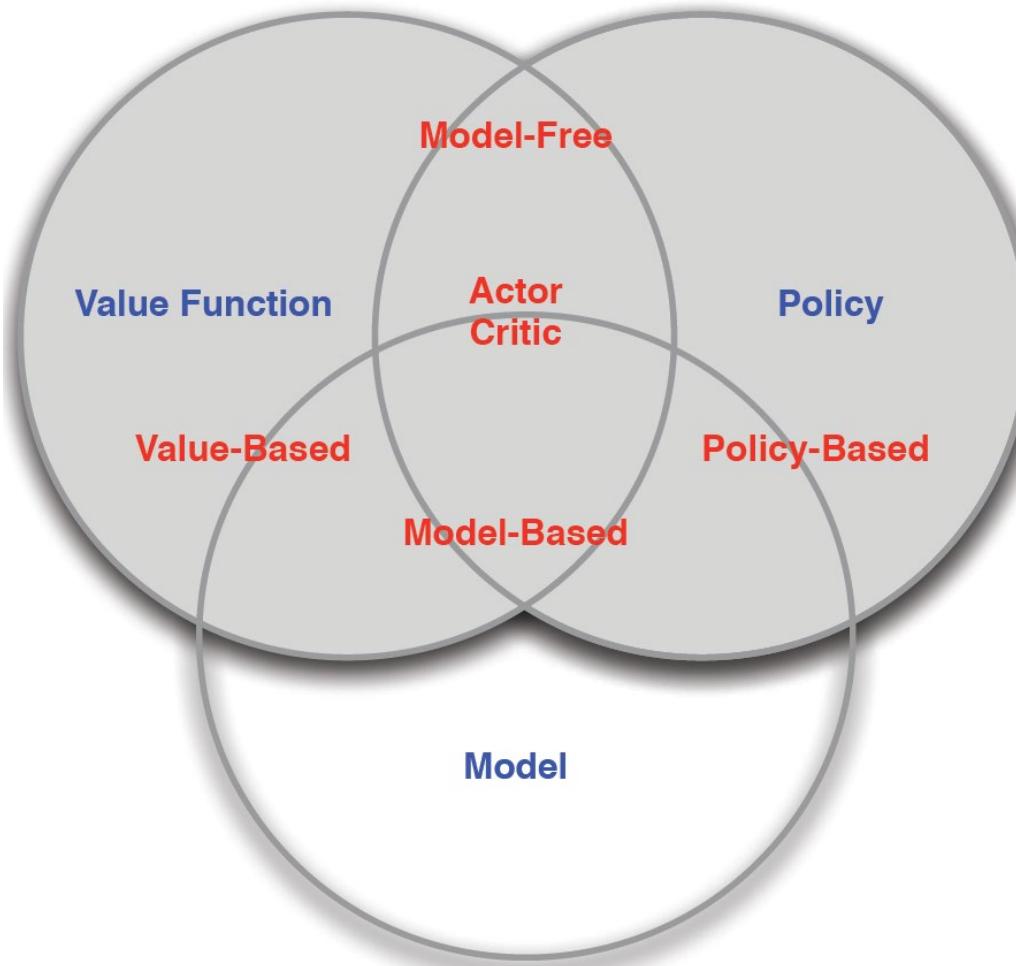
Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

$$v_\pi(s) = \mathbb{E}_\pi [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

Category of RL Agent

- Value Based
 - No Policy (Implicit)
 - Value Function
- Policy Based
 - Policy
 - No Value Function
- Actor Critic
 - Policy
 - Value Function



- Model Free
 - Policy and/or Value Function
 - No Model
- Model Based
 - Policy and/or Value Function
 - Model

Introduction to MDPs

- *Markov decision processes* formally describe an environment for reinforcement learning
- Where the environment is *fully observable*

“The future is independent of the past given the present”

Definition

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away

Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states S_1, S_2, \dots with the Markov property.

Definition

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- \mathcal{S} is a (finite) set of states
- \mathcal{P} is a state transition probability matrix,
$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

Markov Reward Process

A Markov reward process is a Markov chain with values.

Definition

A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- \mathcal{S} is a finite set of states
- \mathcal{P} is a state transition probability matrix,
 $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$
- \mathcal{R} is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

Value Function

The value function $v(s)$ gives the long-term value of state s

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E} [G_t \mid S_t = s]$$

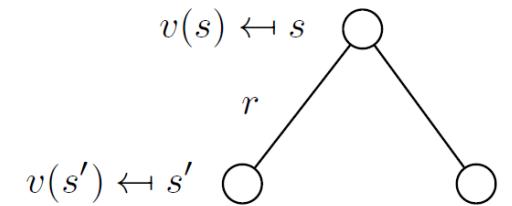
Bellman Equation for MRPs

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

$$v(s) = \mathbb{E} [R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]$$

$$\begin{aligned} v(s) &= \mathbb{E} [G_t \mid S_t = s] \\ &= \mathbb{E} [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s] \\ &= \mathbb{E} [R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \dots) \mid S_t = s] \\ &= \mathbb{E} [R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E} [R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s] \end{aligned}$$



$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- \mathcal{S} is a finite set of states
- \mathcal{A} is a finite set of actions
- \mathcal{P} is a state transition probability matrix,
$$\mathcal{P}_{ss'}^{\textcolor{red}{a}} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = \textcolor{red}{a}]$$
- \mathcal{R} is a reward function, $\mathcal{R}_s^{\textcolor{red}{a}} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = \textcolor{red}{a}]$
- γ is a discount factor $\gamma \in [0, 1]$.

Policy

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
- The state sequence S_1, S_2, \dots is a Markov process $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- The state and reward sequence S_1, R_2, S_2, \dots is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
- where

$$\mathcal{P}_{s,s'}^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}_{ss'}^a$$

$$\mathcal{R}_s^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}_s^a$$

Action-Value Function

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s , and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s , taking action a , and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a]$$

Value Function based RL

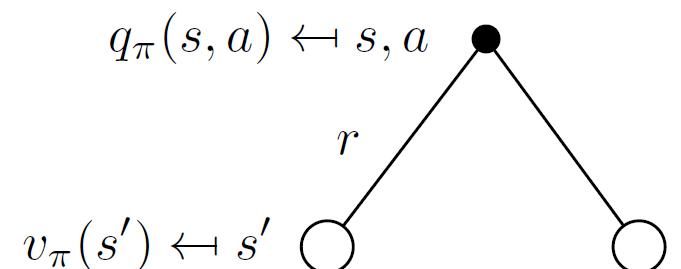
Bellman Expectation Equation

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}_\pi [R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

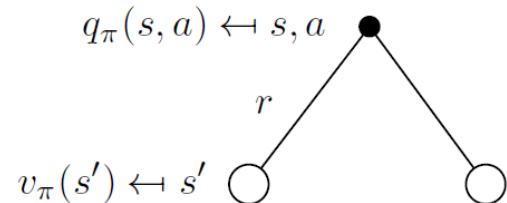
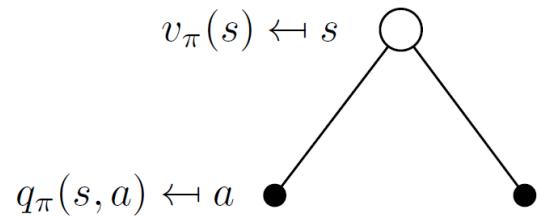
The action-value function can similarly be decomposed,

$$q_\pi(s, a) = \mathbb{E}_\pi [R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$



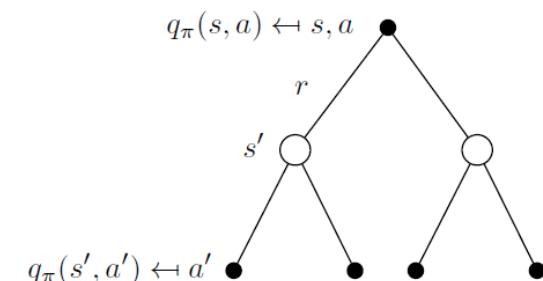
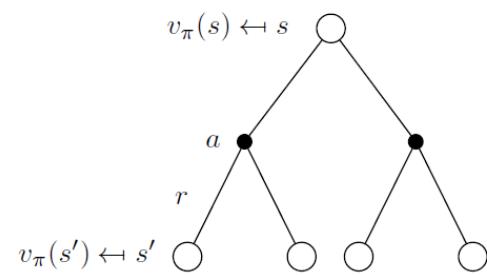
$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Bellman Expectation Equation for V^π, Q^π



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)$$

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a')$$

Optimal Value Function

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_{\pi} v_{\pi}(s)$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is “solved” when we know the optimal value fn.

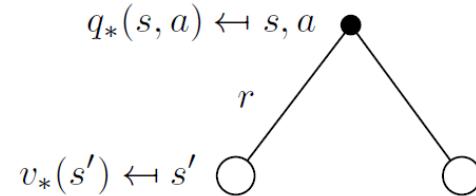
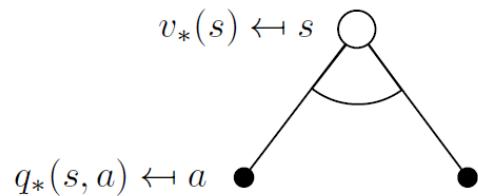
Finding an Optimal Policy

An optimal policy can be found by maximising over $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

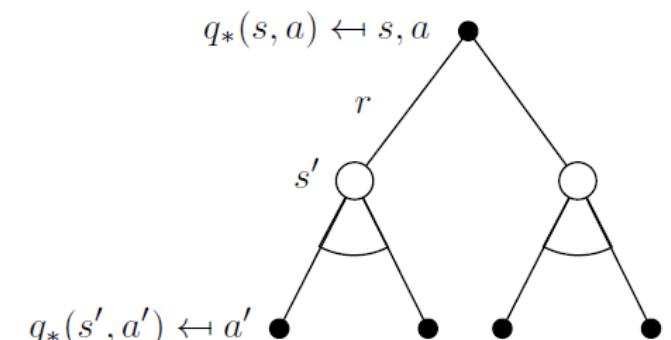
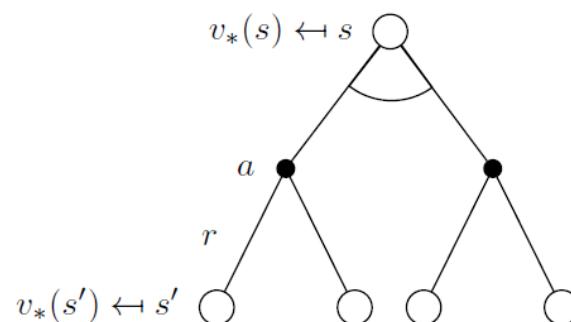
- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Bellman Optimality Equation for V^*, Q^*



$$v_*(s) = \max_a q_*(s, a)$$

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$



$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a')$$

How to Improve a Policy

- Given a policy π

- Evaluate the policy π

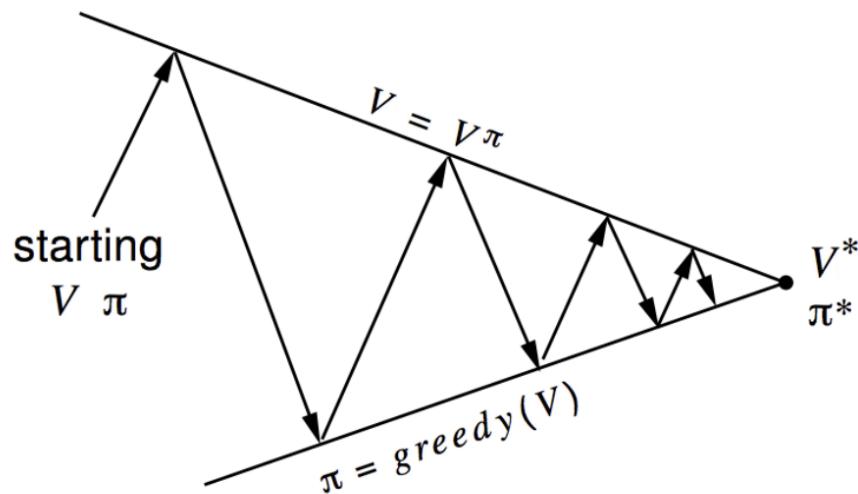
$$v_\pi(s) = \mathbb{E} [R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

- Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to π^*

How to Improve a Policy

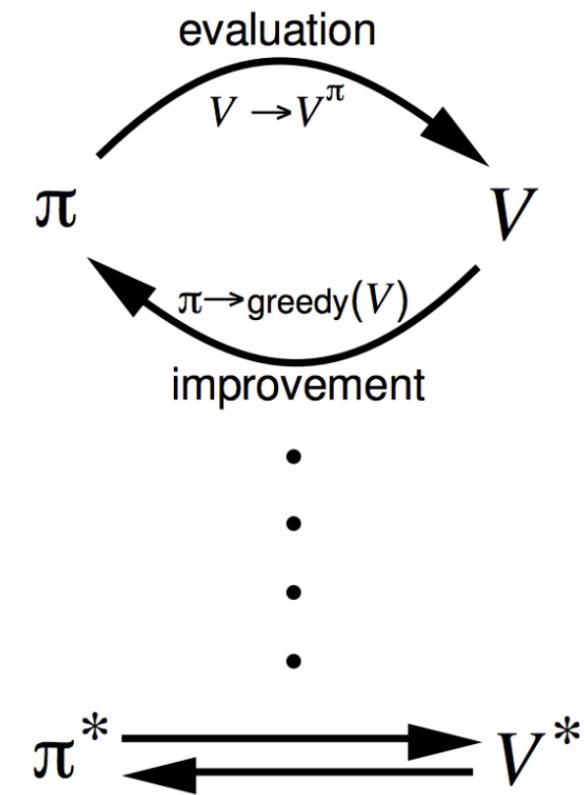


Policy evaluation Estimate v_π

Iterative policy evaluation

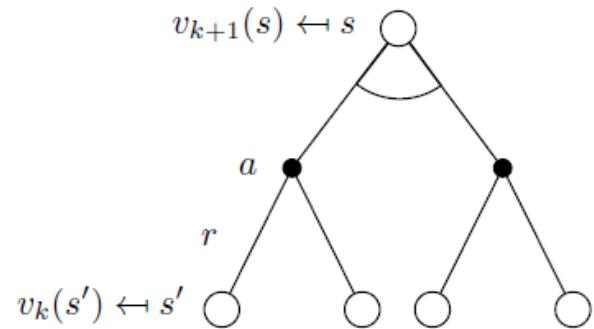
Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement



Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- Using synchronous backups
 - At each iteration $k + 1$
 - For all states $s \in \mathcal{S}$
 - Update $v_{k+1}(s)$ from $v_k(s')$
- Convergence to v_* will be proven later
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

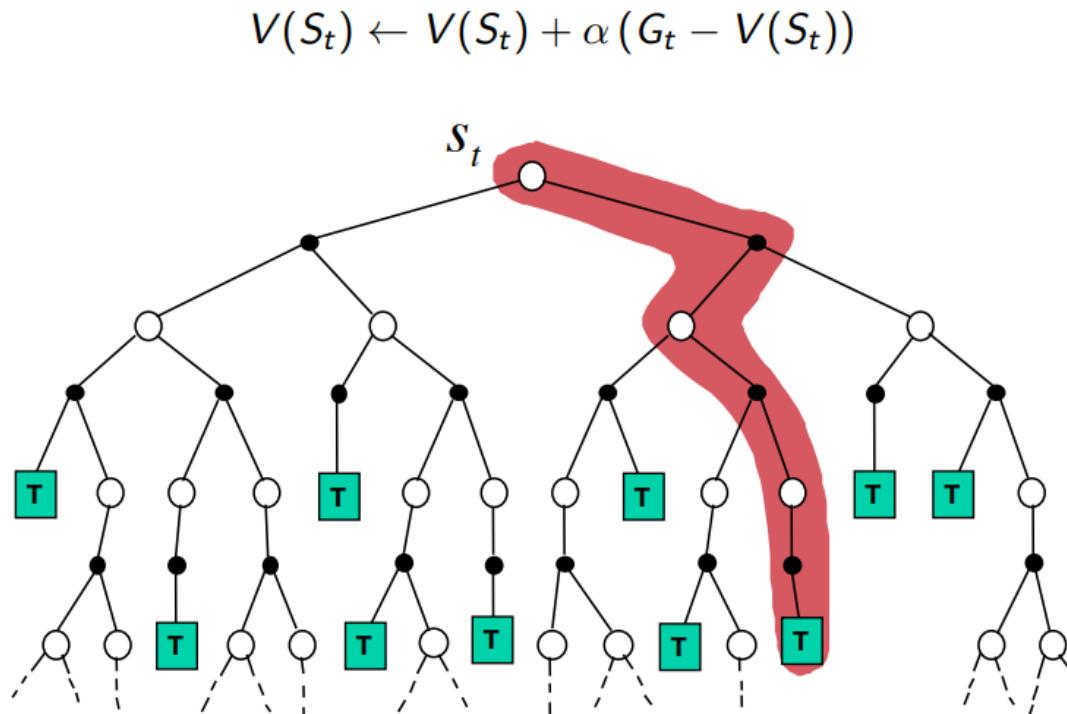


$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$
$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a \mathbf{v}_k$$

Monte-Carlo vs Temporal-Difference Learning

- Monte-Carlo

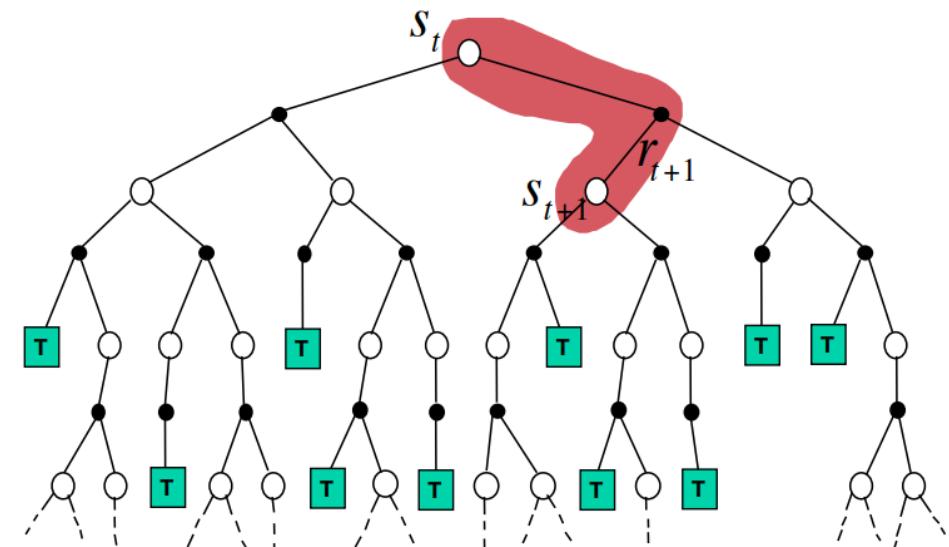
$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



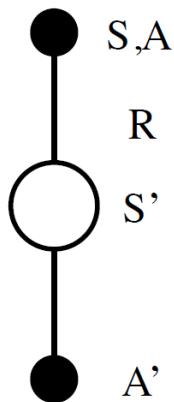
- Temporal-Difference

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



Updating Action-Value Functions with Sarsa



Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ε -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

$$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$$

N-Step SARSA

- Consider the following n -step returns for $n = 1, 2, \infty$:

$$\begin{array}{lll} n = 1 & (\text{Sarsa}) & q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}) \\ n = 2 & & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\ \vdots & & \vdots \\ n = \infty & (\text{MC}) & q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T \end{array}$$

- Define the n -step Q-return

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

- n -step Sarsa updates $Q(s, a)$ towards the n -step Q-return

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{(n)} - Q(S_t, A_t) \right)$$

Q-Learning

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

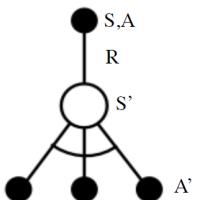
 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$;

 until S is terminal



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

Deep Q-Network (DQN)

DQN uses **experience replay** and **fixed Q-targets**

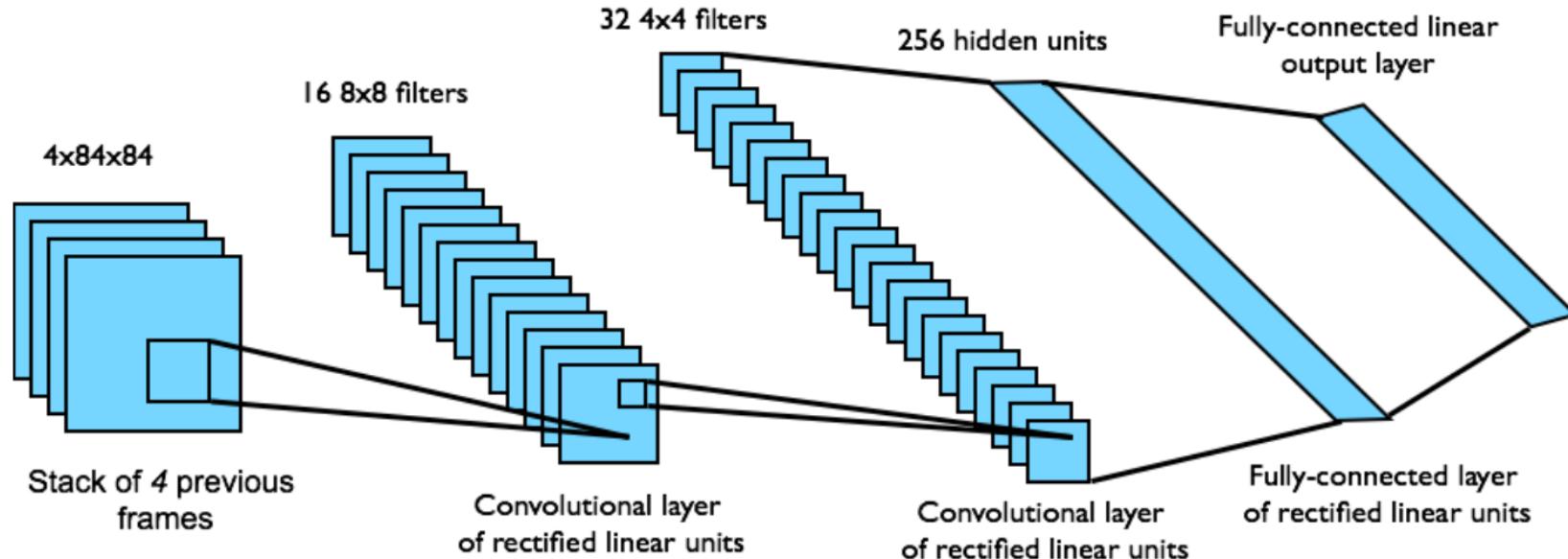
- Take action a_t according to ϵ -greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory \mathcal{D}
- Sample random mini-batch of transitions (s, a, r, s') from \mathcal{D}
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right]$$

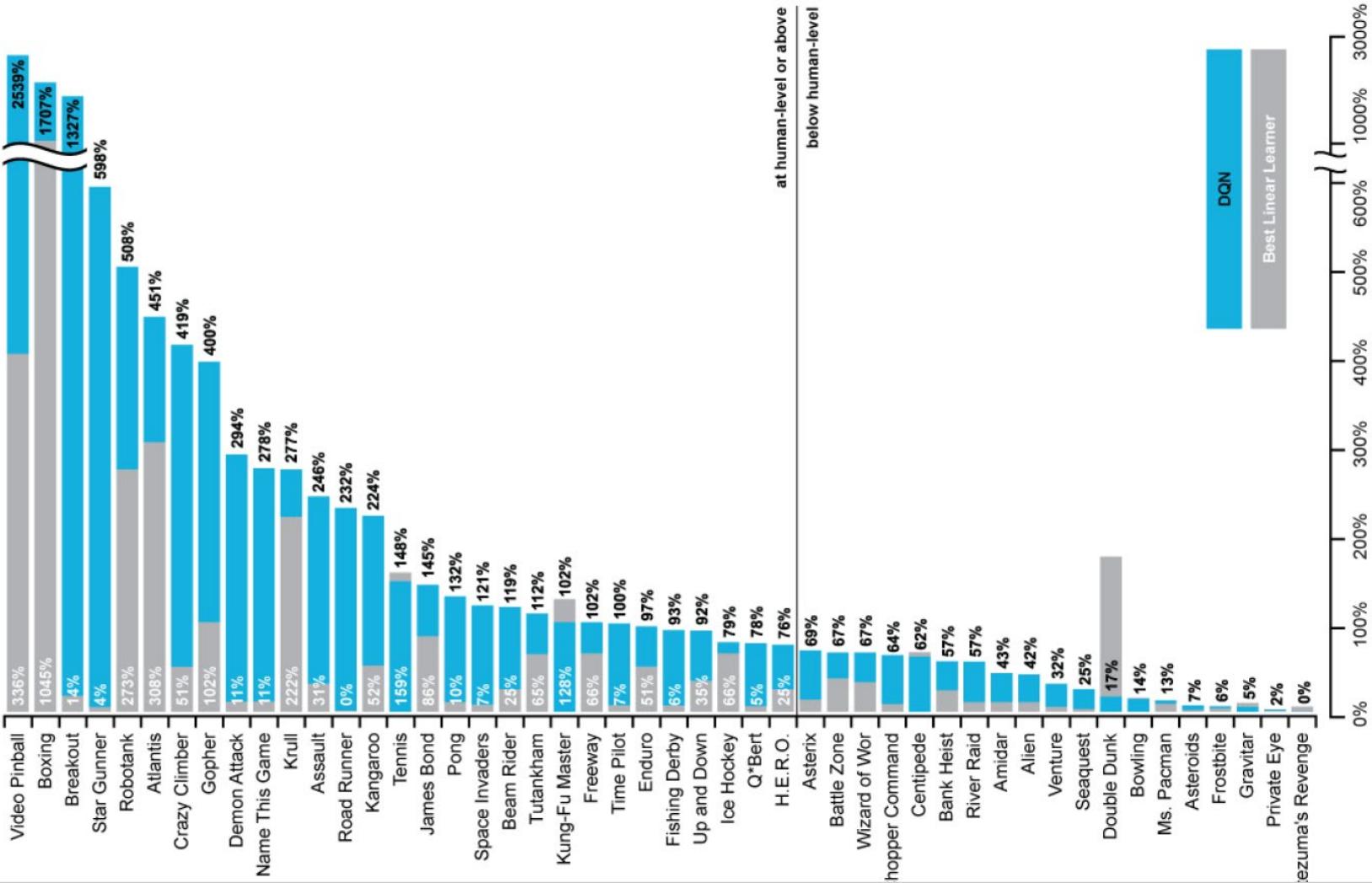
- Using variant of stochastic gradient descent

DQN for Atari Game

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



DQN Results



Policy Gradient based RL

Policy Gradient based RL

- Previous methods are based on the value or action-value function

$$V_\theta(s) \approx V^\pi(s)$$

$$Q_\theta(s, a) \approx Q^\pi(s, a)$$

- A policy was generated directly from the value function
 - e.g. using ϵ -greedy
- In this lecture we will directly parametrise the **policy**

$$\pi_\theta(s, a) = \mathbb{P}[a | s, \theta]$$

- We will focus again on **model-free** reinforcement learning

Policy Objective Functions

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ , find best θ
- But how do we measure the quality of a policy π_θ ?
- In episodic environments we can use the **start value**

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta} [v_1]$$

- In continuing environments we can use the **average value**

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or the **average reward per time-step**

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where $d^{\pi_\theta}(s)$ is **stationary distribution** of Markov chain for π_θ

Policy Optimization

- Policy based reinforcement learning is an **optimisation** problem
- Find θ that maximises $J(\theta)$
- Some approaches do not use gradient
 - Hill climbing
 - Simplex / amoeba / Nelder Mead
 - Genetic algorithms
- Greater efficiency often possible using gradient
 - Gradient descent
 - Conjugate gradient
 - Quasi-newton
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

One-Step MDPs

- Consider a simple class of **one-step** MDPs
 - Starting in state $s \sim d(s)$
 - Terminating after one time-step with reward $r = \mathcal{R}_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$J(\theta) = \mathbb{E}_{\pi_\theta} [r]$$

$$= \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s, a) \mathcal{R}_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) \mathcal{R}_{s,a}$$

$$= \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) r]$$

- **Likelihood ratios** exploit the following identity

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

Policy Gradient Theorem

- The policy gradient theorem generalises the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $Q^\pi(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem

*For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, or $\frac{1}{1-\gamma}J_{avV}$,
the policy gradient is*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$$

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

```
function REINFORCE
    Initialise  $\theta$  arbitrarily
    for each episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$  do
        for  $t = 1$  to  $T - 1$  do
             $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$ 
        end for
    end for
    return  $\theta$ 
end function
```

Reducing Variance Using a Critic

- Monte-Carlo policy gradient still has high variance
- We use a **critic** to estimate the action-value function,

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain *two* sets of parameters
 - Critic Updates action-value function parameters w
 - Actor Updates policy parameters θ , in direction suggested by critic
- Actor-critic algorithms follow an *approximate* policy gradient

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) \ Q_w(s, a)] \\ \Delta\theta &= \alpha \nabla_\theta \log \pi_\theta(s, a) \ Q_w(s, a)\end{aligned}$$

Action-Value Actor-Critic (A2C)

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s, a) = \phi(s, a)^\top w$
 - Critic Updates w by linear TD(0)
 - Actor Updates θ by policy gradient

```
function QAC
    Initialise  $s, \theta$ 
    Sample  $a \sim \pi_\theta$ 
    for each step do
        Sample reward  $r = \mathcal{R}_s^a$ ; sample transition  $s' \sim \mathcal{P}_{s, a}$ .
        Sample action  $a' \sim \pi_\theta(s', a')$ 
         $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$ 
         $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$ 
         $w \leftarrow w + \beta \delta \phi(s, a)$ 
         $a \leftarrow a', s \leftarrow s'$ 
    end for
end function
```

Reducing Variance Using a Baseline

- We subtract a baseline function $B(s)$ from the policy gradient
- This can reduce variance, without changing expectation

$$\begin{aligned}\mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) B(s)] &= \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) B(s) \\ &= \sum_{s \in \mathcal{S}} d^{\pi_\theta} B(s) \nabla_\theta \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \\ &= 0\end{aligned}$$

- A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
- So we can rewrite the policy gradient using the **advantage function** $A^{\pi_\theta}(s, a)$

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$$

Reading Assignments

- Deep Deterministic Policy Gradient (DDPG)
- Asynchronous A2C (A3C)