Basics of Reinforcement Learning

Ioana Veronica Chelu

August 29, 2018

Acknoledgements

Special thanks to the organizers, speakers and lab assistants of the **TMLSS - Transylvanian Machine Learning Summer School** for the content of the lectures and labs presented during the school, which serve as basis for this lecture and slides.

Table of contents

Introduction

How to build intelligent machines?

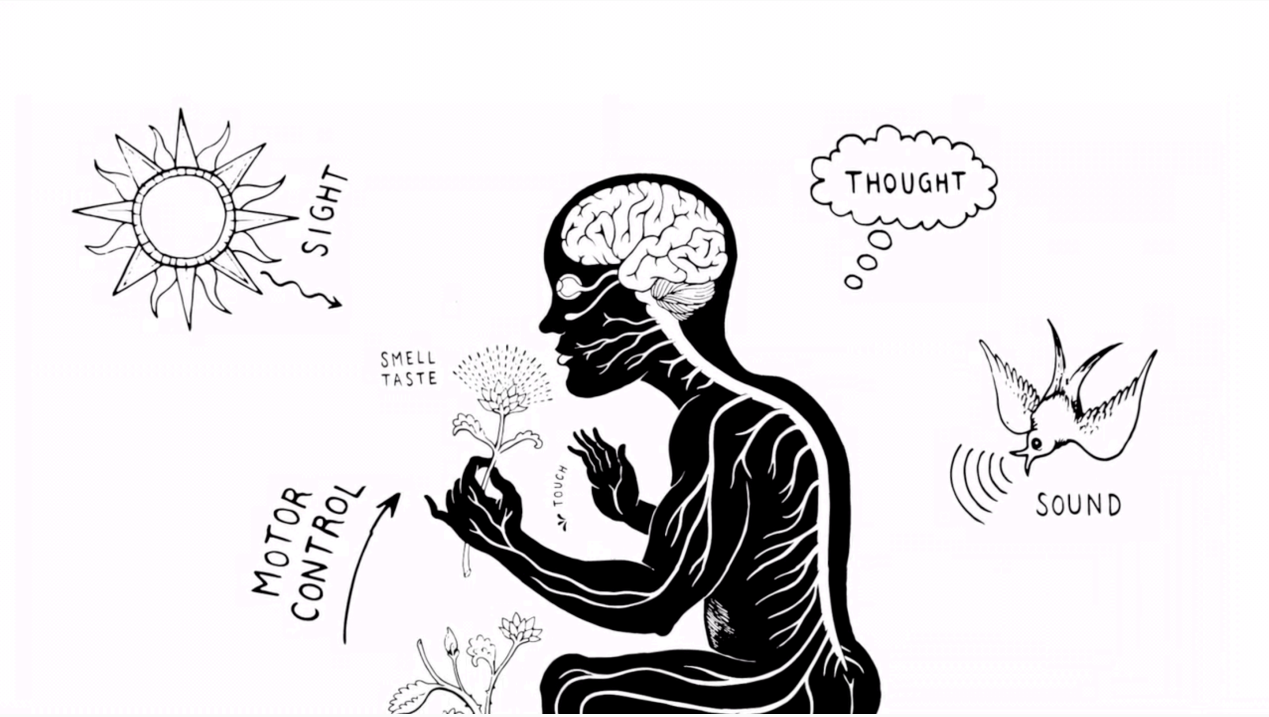
- Some innate aptitudes (e.g the inborn ability to walk which horses have immediately after being born, recognizing faces in babies, swimming in dogs, elephants)
- Some things we can only learn (e.g. driving a car, playing instruments).
- A spectrum of skills impossible to have been programmed to do

Learning as the basis of intelligence

- Our learning mechanism(s) are likely powerful enough to do everything we associate with intelligence

Reinforcement learning = a general purpose algorithm

- Interpret rich sensory inputs (NN)
- Choose complex actions (RL)



Why deep reinforcement learning?

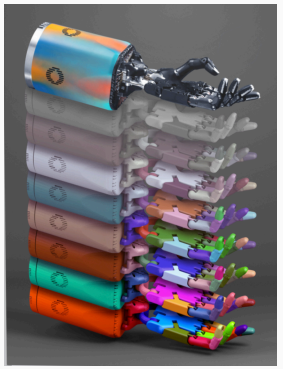
- **Deep** = can process complex sensory input and also compute really complex functions
- **Reinforcement learning** = can choose complex actions

What can deep learning & RL do well now?

- Proficiency in domains governed by simple, known rules

- Learn simple skills with raw sensory inputs, given enough experience

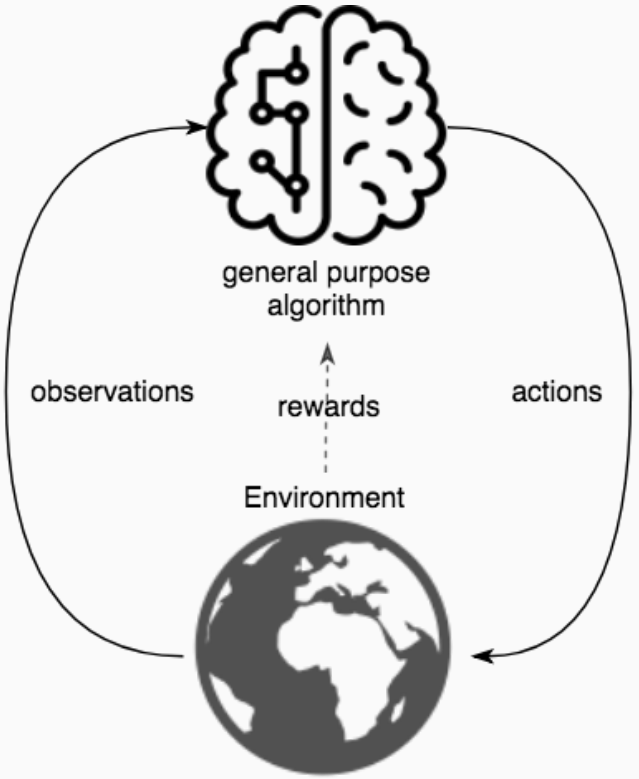
- Learn from imitating enough human provided expert behavior



Adaptive learning-based mechanism for complex behavior

* as a **goal-directed agent** interacting with an **uncertain environment** through trail and error.

"Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain."
- **Alan Turing**

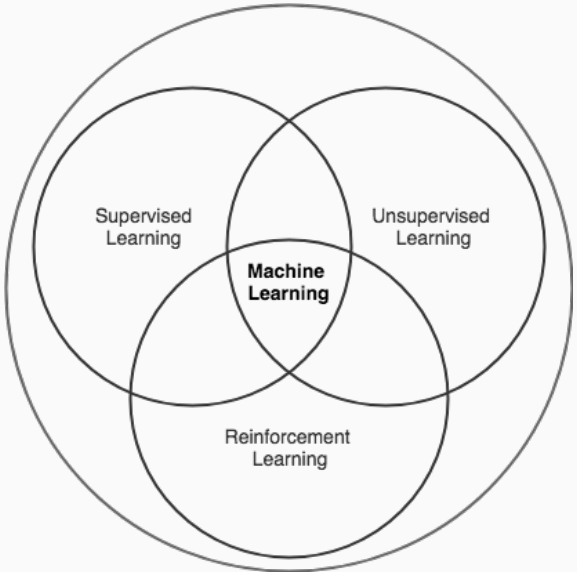


Sequential Decision Making

About RL

Features

- no supervisor, only a reward signal
- delayed feedback
- sequential stream of data, not i.i.d
- non-stationary system
- trial and error search



Examples

- navigate mazes/cities
- play Atari games better than humans
- defeat the world's best Go player
- make a humanoid robot walk
- manage an investment portfolio
- control a power station

Reinforcement Learning as Markov process

Humans and animals

- direct sensorimotor connection to the environment
- a lot of cause and effect information
- incorporate sense, actions and goals

Markov decision process

- is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P} >$:
  - $\mathcal{S}$ is the set space,
  - $\mathcal{A}$ is the action space,
  - $\mathcal{P}(s', r|s, a)$ - the dynamics model. distribution. p specifies a probability distribution for each choice of s, a and r, where $r \in \mathcal{R}$ is the reward.

Markov Property

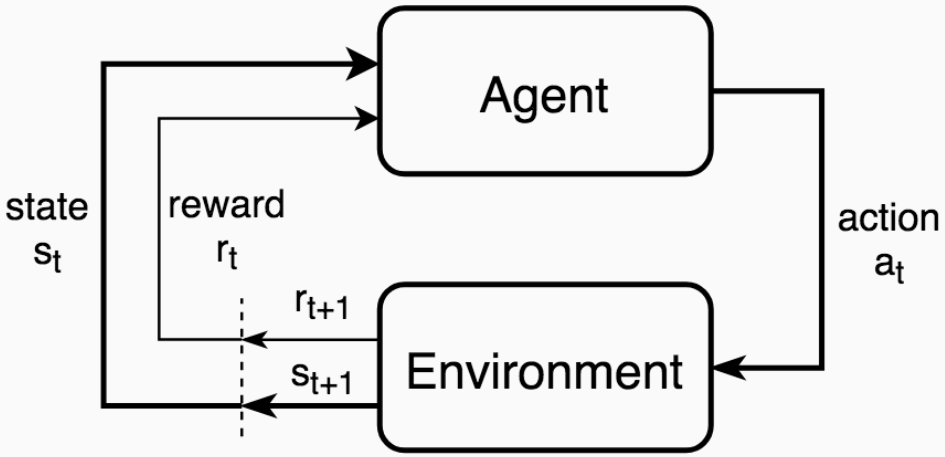
"The future is independent of the past given the present"

Markov state

$P[St + 1|St] = P[St + 1|S1, ..., St]$

- The state captures all relevant information about the past
- In POMDPS this doesn't hold
- With function approximation and recurrent models we can ignore this

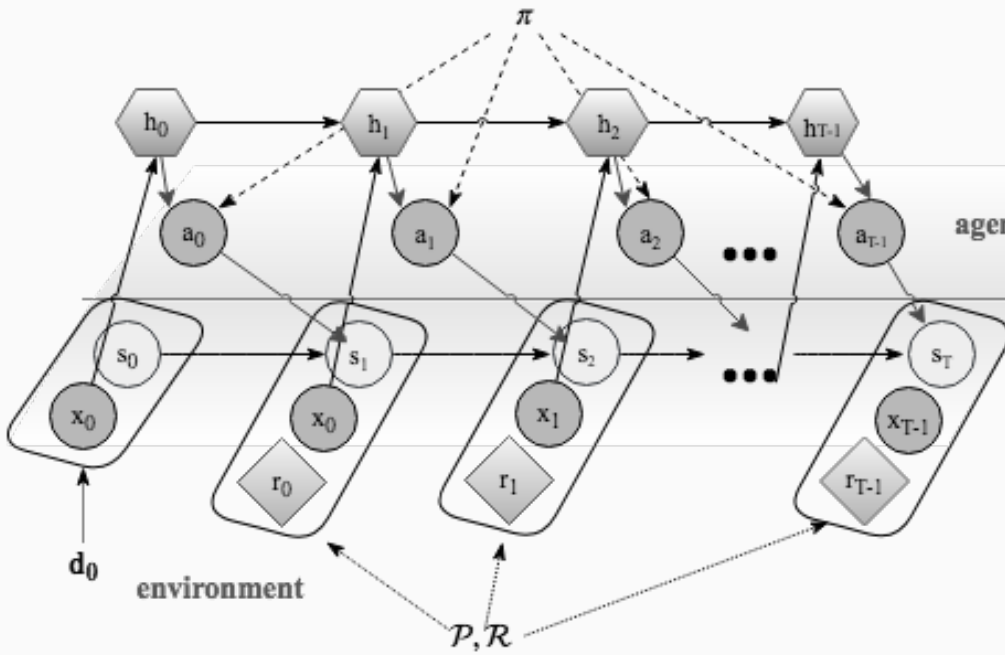
Sequential decision making



- at each step t:
  - the agent
    - executes action A_t
    - receives observation S_t
    - receives scalar reward R_t
  - the environment:
    - receives action A_t
    - emits observation S_{t+1}
    - emits scalar reward R_{t+1}
- t++

POMDP - as a stochastic graph

$$
\begin{aligned}
& s_0, x_0 \sim d_0 \\
& a_0 \sim \pi(a_0|h_0) \\
& s_1, x_1, r_0 \sim \mathcal{P}(s_1, x_1, r_0|s_0, a_0) \\
& a_1 \sim \pi(a_1|h_1) \\
& s_2, x_2, r_1 \sim \mathcal{P}(s_2, x_2, r_1|s_1, a_1) \\
& \dots \\
& a_{T-1} \sim \pi(a_{T-1}|h_{T-1}) \\
& s_T, x_T, r_{T-1} \sim \mathcal{P}(s_T, x_T, r_{T-1}|s_{T-1}, a_{T-1})
\end{aligned}
\tag{1}
$$



The RL problem

Elements of Reinforcement Learning

An RL agent may include one or more of these elements:

- **Policy** - defines the agent's behaviour
- **Value functions** - indicate how good it is to be in a particular situation (long term)
- **Model** - simulated representation of the environment

Policies

The policy π is represented by a function describing a mapping from states to actions:

- deterministic: $a = \pi(s)$
- stochastic: $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$

Goal is to find a policy π that maximizes the expected future sum of cumulative reward.

$$maximize_{\pi} \mathbb{E}_{\pi}[G_t],$$

$$where\ G_t = R_{t+1} + \gamma R_{t+2} + \ \ldots \ + \gamma^{T-1} R_T = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2}$$

T is the length of the episode, in the case of episodic tasks.

Value functions

State value function

$$v^{\pi}(s) : \mathcal{S} \rightarrow \mathcal{R} \tag{3}$$
$$v^{\pi}(s) = \mathbf{E}_{\pi}[G_t | S_t = s]$$

The state-action value function

$$q^{\pi}(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R} \tag{4}$$
$$q^{\pi}(s, a) = \mathbf{E}_{\pi}[G_t | S_t = s, A_t = a]$$

Greedy policy / optimal policy

$\pi(a) = argmax_{a\prime} q(s, a\prime)$.

In a discrete MDP, there $\exists\ |\pi*| \geq 1$ greedy with respect to its own q^*.

Discounted setting

- γ - the discount factor.
- the agent values more rewards received in the near future that those received far into the future.
- rewards are discounted by $\gamma^{\tau - t}$, where $\gamma \in [0, 1]$

Model

Model

- representation of the environment's dynamics
- can be an **explicit** probabilistic model - expected transitions
- or it can be an **implicit** model - sample transitions
- is used to simulate the environment and produce simulated experience

model $\xrightarrow{\text{planning}}$ *policy*

model $\xrightarrow{\text{simulated experience}}$ *value functions policy*

Types of RL agents

- Value Based
  - Policy - implicit
  - Value Function
- Policy Based
  - Policy - explicit
  - No Value Function
- Actor Critic
  - Policy - explicit
  - Value Function

- Model Free
- Model Based
  - *background planning*
    - build model for the entire state-space
    - simulate experience
    - improve policy/value function
  - *decision planning*
    - build model for the current state
    - ”roll” the model ahead to see how things play out
    - decide the best action for current state

Exploration-Exploitation dilemma

- **Exploration** - find more information about the environment
- **Exploitation** - exploit known information to maximize reward
- Typically we need have equal need of both

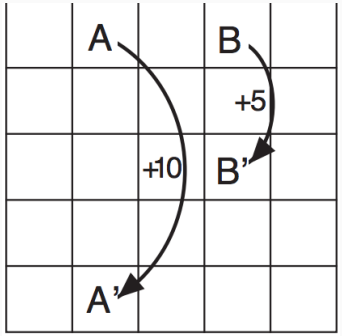
Prediction & Control

Prediction and Control - overview

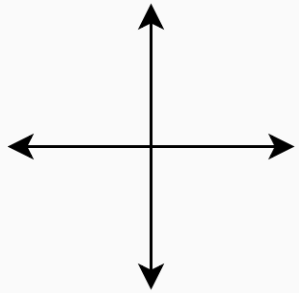
- **Prediction** - evaluate the future under a given policy
- **Control** - optimize on future behavior in to find the best policy

Gridworld Example

Prediction - what is the value function v^π for some policy π?



(a) gridworld

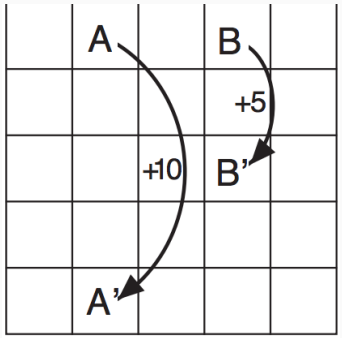


(b) actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(c) v^π

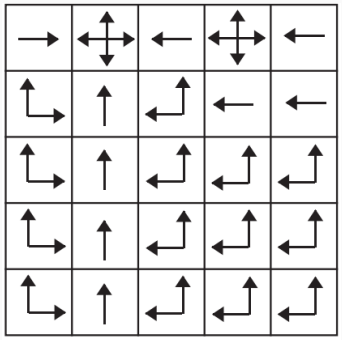
Control - What is the optimal value function $v*$?
What is the optimal policy? $\pi*$?



(a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

(b) $v*$



(c) $pi*$

Bellman Expectation Equation

- math operator - iteratively applied to value functions.
- a single step $\rightarrow$ the value function closer to its real value $|\pi$.
- iterating this process $\rightarrow$ convergence to the real values of π.

$$T^{\pi}v(s) = r(s, \pi(s)) + \gamma \sum \mathcal{P}(s'|s, \pi(s))v(s') \tag{5}$$

State value function

$$\begin{aligned} v^{\pi}(s) &= \mathbf{E}_{\pi}[R_{t+1} + \gamma v^{\pi}(S_{t+1})|S_t = s] \\ v^{\pi}(s) &= \sum_{a \in \mathcal{A}} \pi(s, a)[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a)v^{\pi}(s')] \\ V^{\pi} &= \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} V^{\pi} \end{aligned} \tag{6}$$

The state-action value function

$$\begin{aligned} q^{\pi}(s, a) &= \mathbf{E}_{\pi}[R_{t+1} + \gamma q^{\pi}|S_t = s, A_t = a] \\ q^{\pi}(s, a) &= \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a)[r(s, a) + \sum_{a' \in \mathcal{A}} q^{\pi}(s', a')] \end{aligned} \tag{7}$$

Bellman Optimality Equations

State value function

$$
\begin{aligned}
v^*(s) &= \max_{a \in \mathcal{A}} q^*(s,a) \\
v^*(s) &= \max_{a \in \mathcal{A}} [r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) v^*(s')] \\
v^* &= \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v^*
\end{aligned}
\tag{8}
$$

The state-action value function

$$
\begin{aligned}
q^*(s,a) &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) v^*(s')] \\
q^*(s,a) &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}} q^*(s',a')]
\end{aligned}
\tag{9}
$$

Dynamic Programming

- General solution method for problems with:
  - optimal solution can be decomposed into subproblems
  - overlapping subproblems
- Markov decision processes satisfy these
- Bellman equation gives recursive decomposition
- Value functions store and reuse solutions

Learning in a known MDP with Dynamic Programming

- assume full knowledge of the MDP - **model-based**
- plan in an MDP
- use the idea of **expected estimates**
  - **Prediction**:
    - Input: MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma >$, π
    - Output: value function v^{π}
  - **Control**:
    - Input: MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma >$,
    - Output: optimal value function v^*

Solutions

- Policy Iteration
- Value Iteration

Policy Iteration

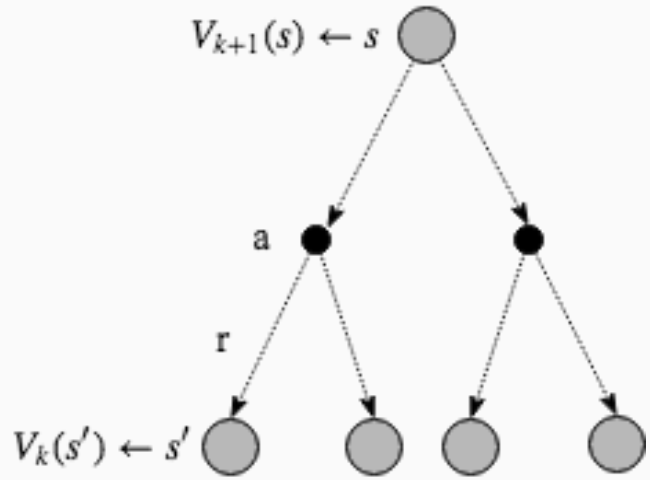
Two interleaved steps:

- policy evaluation
- policy improvement

Iterative Policy Evaluation

Policy evaluation

- Find $v|\pi$
- Apply **Bellman expectation operator** iteratively to back up the value function:
  $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_\pi$



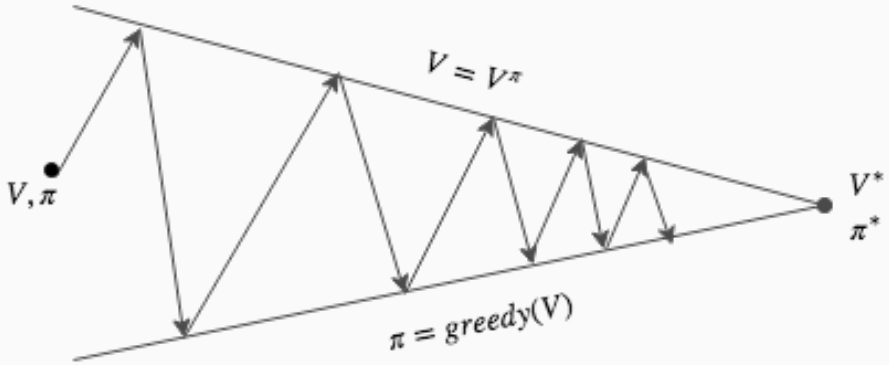
$$v^{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(s,a)[r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,\pi(s))v^k(s')]$$

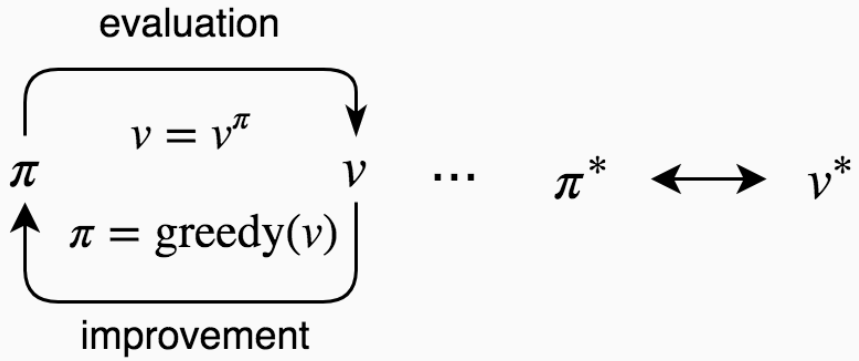
$$v^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v^k \tag{10}$$

Improving the policy

Policy improvement

- Given v_π
- Improve the policy by **acting greedily** with respect to it: $\pi' = greedy V^\pi$



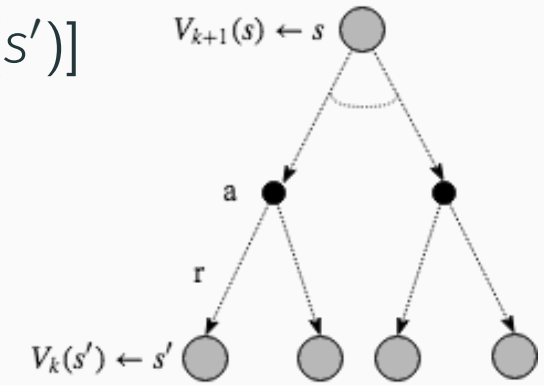


Value iteration

- Doesn't search for an explicit policy
- Apply the **Bellman optimality** backup iteratively:
  $V_1 \rightarrow V_2 \rightarrow ... \rightarrow V*$.

$$v^{k+1}(s) = \max_{a \in \mathcal{A}} [r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) V^k(s')]$$

$$v^{k+1} = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v^k \tag{11}$$



Learning in an unknown MDP

- model-free prediction = **estimate** the value function of an unknown MDP
- model-free control = **optimize** the value function of an unknown MDP

Prediction

- Monte-Carlo methods
- Temporal-difference methods

Monte-Carlo learning

- Learn directly from episodes of experience (**sample estimates**)
- **Model-free**: no knowledge of MDP transitions / rewards
- Learn from **complete episodes: no bootstrapping**
- Centered around the idea: **value = mean return**
- Can only be applied to episodic MDPs (episodes must terminate)
- The more samples $\rightarrow$ less error in the estimate
- A lot of variance (noise) in the samples

Monte-Carlo policy evaluation

- Estimate v_π from episodes of experience under policy π:
  $S_1, A_1, R_2, ..., S_k \sim \pi$
- Return is the total discounted reward:
  $G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T1} R_T$
- Value function is the *empirical mean* return: $v_\pi(s) = \sum_{t=1}^{t=T} G_t$

$$
\begin{aligned}
N(St) &\leftarrow N(S_t) + 1 \\
V(S_t) &\leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))
\end{aligned}
\tag{12}
$$

- In non-stationary problems $\Rightarrow$ a running mean (forget old episodes)

$$
\begin{aligned}
N(St) &\leftarrow N(S_t) + 1 \\
V(S_t) &\leftarrow V(S_t) + \alpha(G_t - V(S_t))
\end{aligned}
\tag{13}
$$

Temporal-Difference Learning

- Learn directly from episodes of experience (**sample estimates**)
- **Model-free**: no knowledge of MDP transitions / rewards
- Learns from **incomplete episodes, by bootstrapping**
- Can be applied to non-episodic tasks
- **Update a guess from a guess** (bias) after grounding in one - TD(0), or more - n-step TD, TD(λ) - interactions of experience with the environment

Temporal-difference policy evaluation

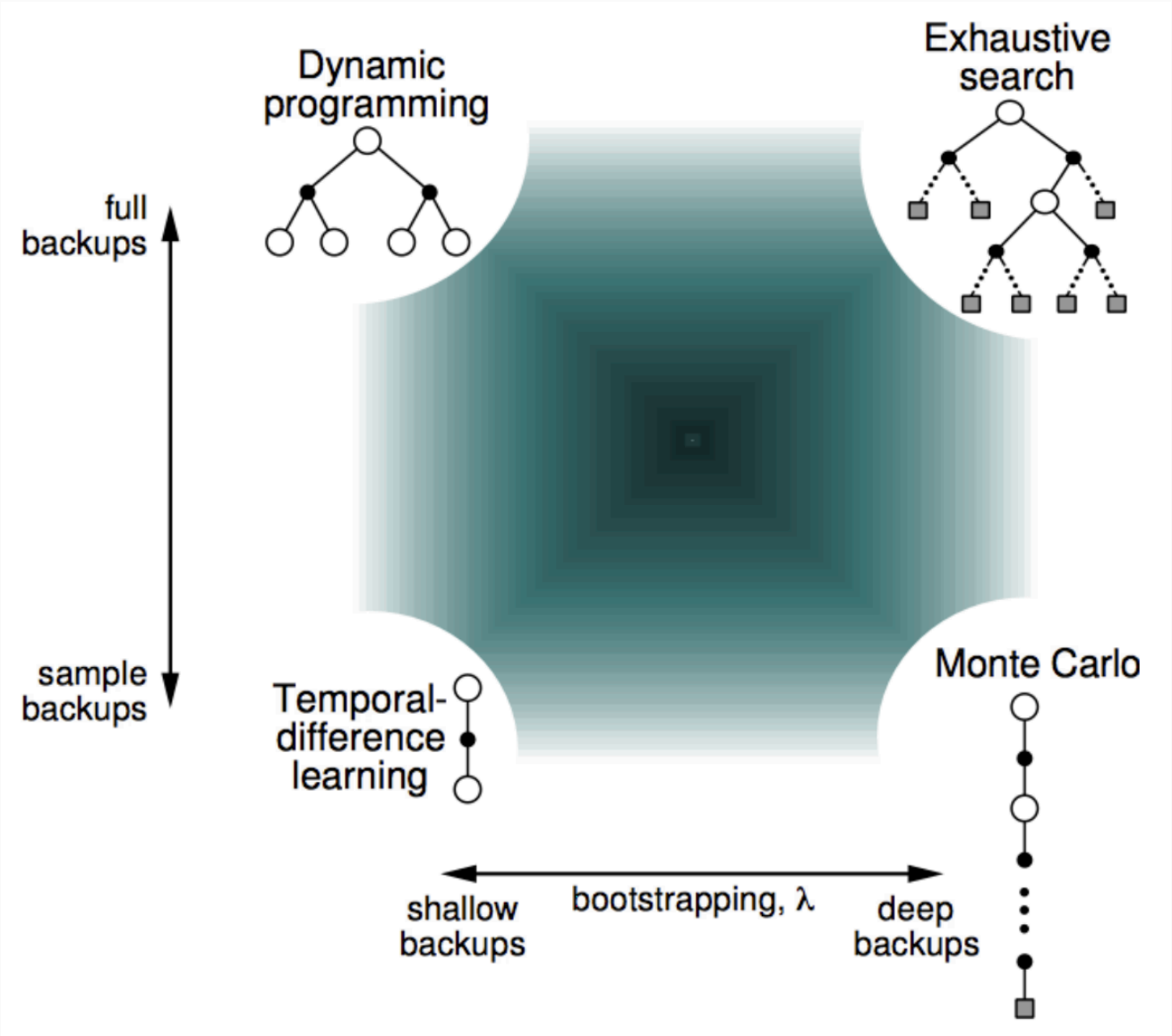
- Update value $V(S_t)$ toward estimated return: $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{14}$$

MC vs. TD

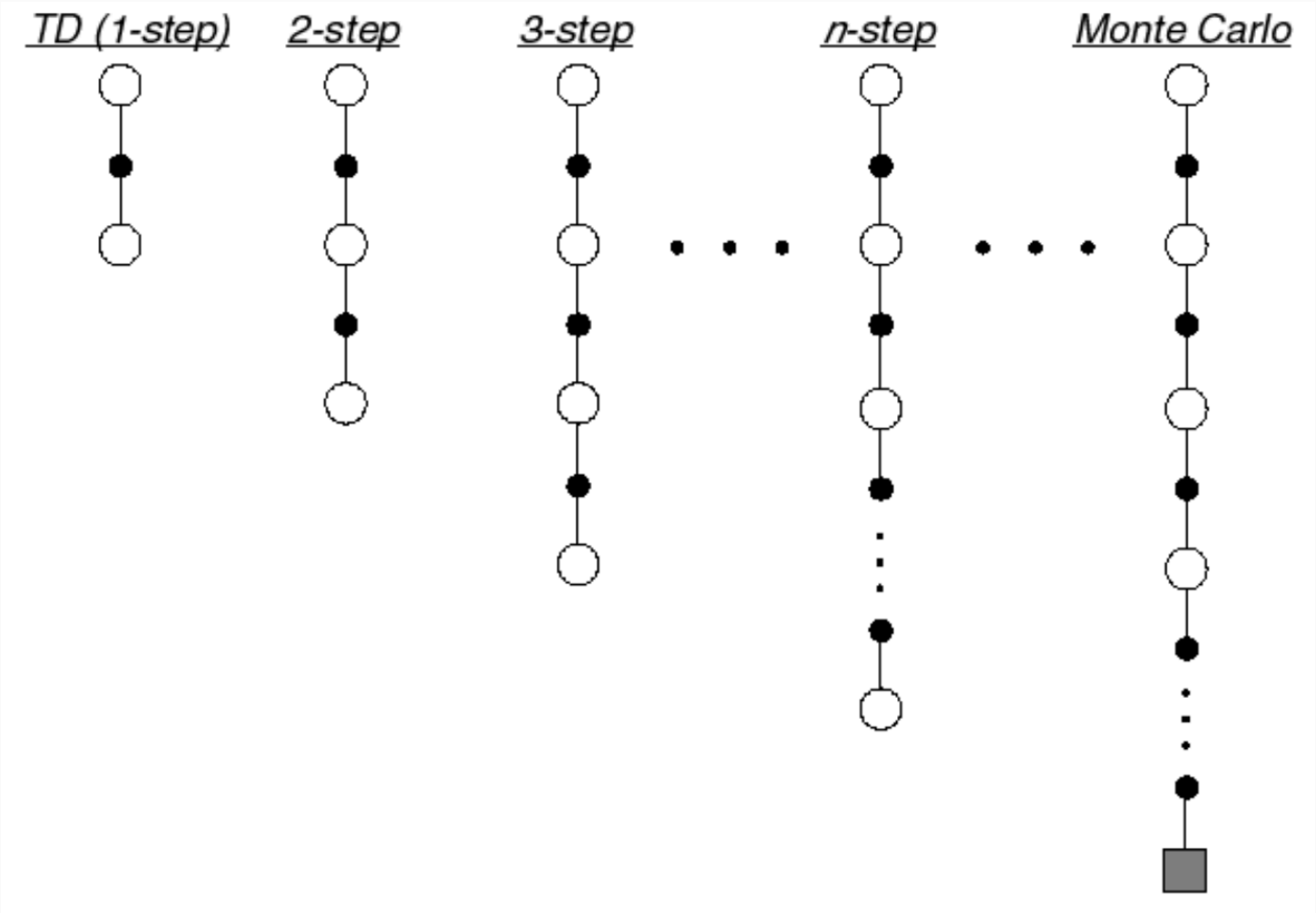
- MC has high variance, zero bias
- Good convergence properties (even with function approximation)
- Not very sensitive to initial value
- TD has low variance, some bias
- Usually more efficient than MC
- TD(0) converges to $v_\pi(s)$ (but not always with function approximation)
- More sensitive to initial value

Unified view of RL



n-step Prediction

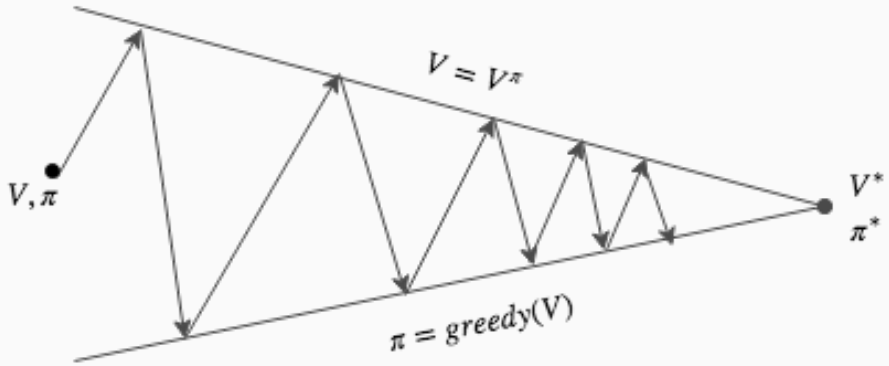
TD target - after grounding in n steps of reality

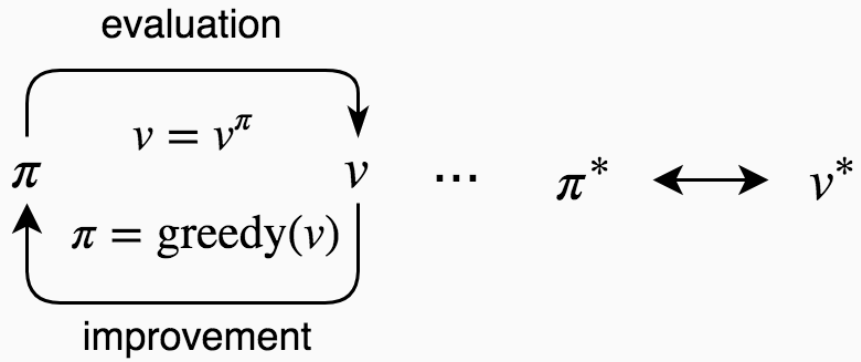


Model-Free Control

- **Policy evaluation** $\rightarrow$ estimate v_π - iterative policy evaluation
- **Policy improvement** $\rightarrow \pi' >= \pi$ - greedy policy improvement

Generalised Policy Iteration (GPI)





- On-policy vs Off-policy:
  - On-policy learning = learn about policy π from experience sampled from π
  - Off-policy learning = learn about policy π from experience sampled from μ
- Solutions
  - Vanilla MC (on-policy/off policy with importance sampling)
  - Q-Learning (off-policy)
  - Sarsa (on-policy/off policy with importance sampling)
  - Expected Sarsa (on-policy/off policy with importance sampling)

GPI With Monte-Carlo Evaluation

- **Policy evaluation** Monte-Carlo policy evaluation - $Q \approx q_\pi$
- **Policy improvement** ϵ**-greedy policy** improvement

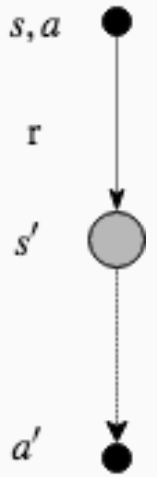
ϵ-greedy policy

- Ensure continual exploration
- With probability 1 – choose the greedy action
- With probability choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a* = \text{argmax}_{a \in \mathcal{A}} Q(s,a), \\ \epsilon/m, & \text{otherwise.} \end{cases} \tag{15}$$

Sarsa

- On-policy control with TD-learning of the action-value function Q
- **Policy evaluation** $\Rightarrow$ Sarsa
- **Policy improvement** - greedy policy improvement

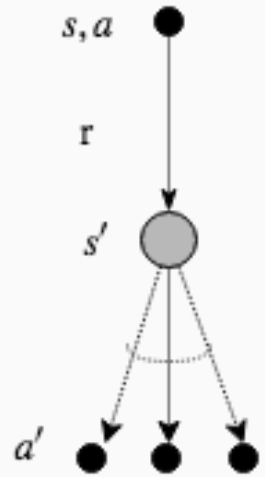


$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s',a')) \tag{16}$$

$$\begin{aligned} q_t^{(n)} &= r_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, a_{t+n}) \\ Q(s_t,a_t) &\leftarrow Q(s_t,a_t) + \alpha(q_t^{(n)} - Q(S_t,A_t)) \end{aligned} \tag{17}$$

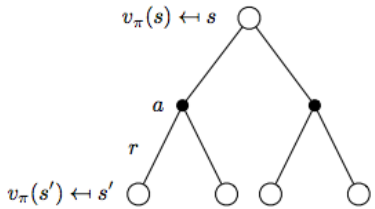
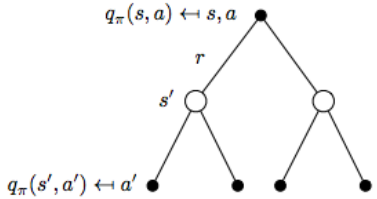
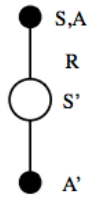
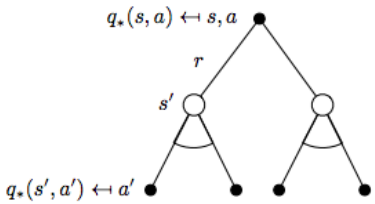
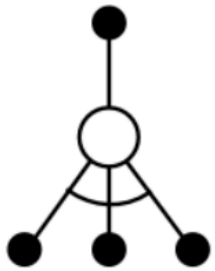
Q-learning

- Off-policy control with TD-learning of the action-value function Q
- **Target policy** π is greedy w.r.t. $Q(s, a)$
- **Behaviour policy** μ is more exploratory (ϵ-greedy w.r.t. $Q(s, a)$)



$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma max_{a'} Q(s', a') - Q(s', a')) \tag{18}$$

Unified view - DP vs TD

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s,a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s,a)$	 Q-Value Iteration	 Q-Learning

Function approximation

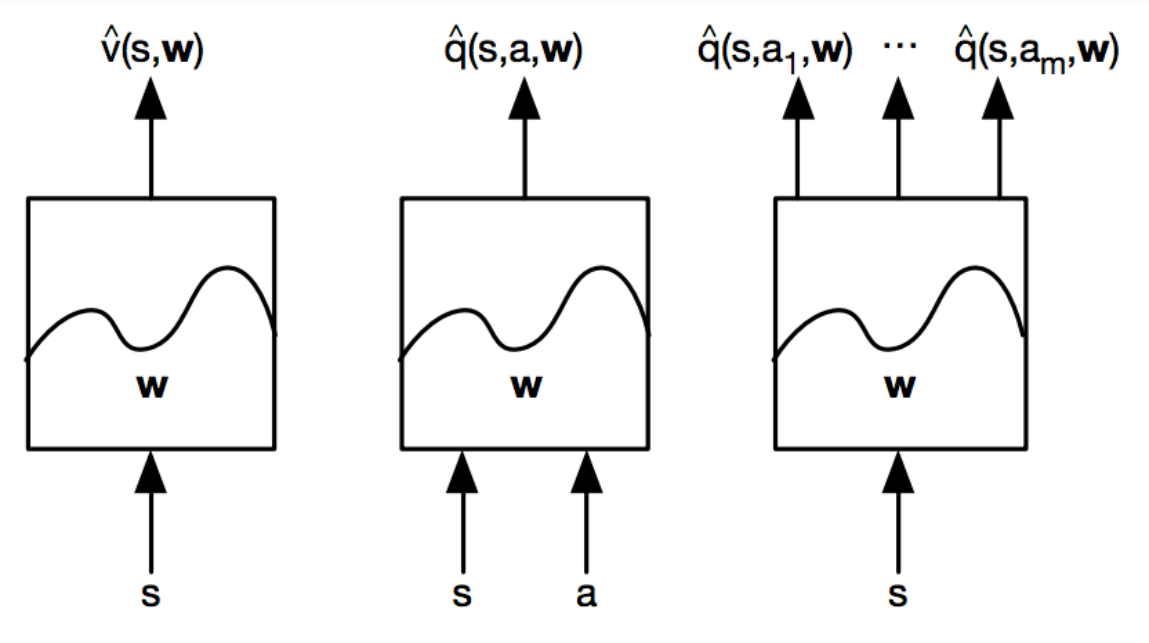
Function approximation

- **Previously...**
  - each state is simply an index or a one-hot encoding
  - each state s has an entry V(s) (state-action pair s, a - Q(s, a))
- Intractable when we deal with large or infinite state spaces
- Agent is incapable of sharing experience between states
- There are too many states and/or actions to store in memory
- It is too slow to learn the value of each state individually
- **Now...**
  - Function approximation = the value of a function is completely described in terms of its parameters (weights & biases of a neural network).
  - Minimize losses (e.g TD-error loss) - gradients of the function with respect to its parameters.
  - Generalize from seen states to unseen states

Value-function approximation

- Estimate value function with function approximation

$$\hat{v}(s; w) \approx v_\pi(s)$$
$$\hat{q}(s, a; w) \approx q_\pi(s, a) \tag{19}$$



Value-function approx by SGD

- Performance objective for value function approximation

$$J(w) = \mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S; w))^2] \tag{20}$$

- Find w that minimizes the objective - gradient descent

$$\begin{aligned} \nabla_w &= \frac{1}{2}\alpha \nabla_w J(w) \\ &= \alpha \mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S; w))\nabla_w \hat{v}(S; w)] \end{aligned} \tag{21}$$

- Stochastic gradient descent samples the gradient

$$\nabla_w = \alpha(v_\pi(S) - \hat{v}(S; w))\nabla_w \hat{v}(S; w) \tag{22}$$

- Expected update is equal to full gradient update

Model-free Prediction by SGD

- When we don't have an oracle for the target v_π:
- **Monte-Carlo** - target = return (G_t)

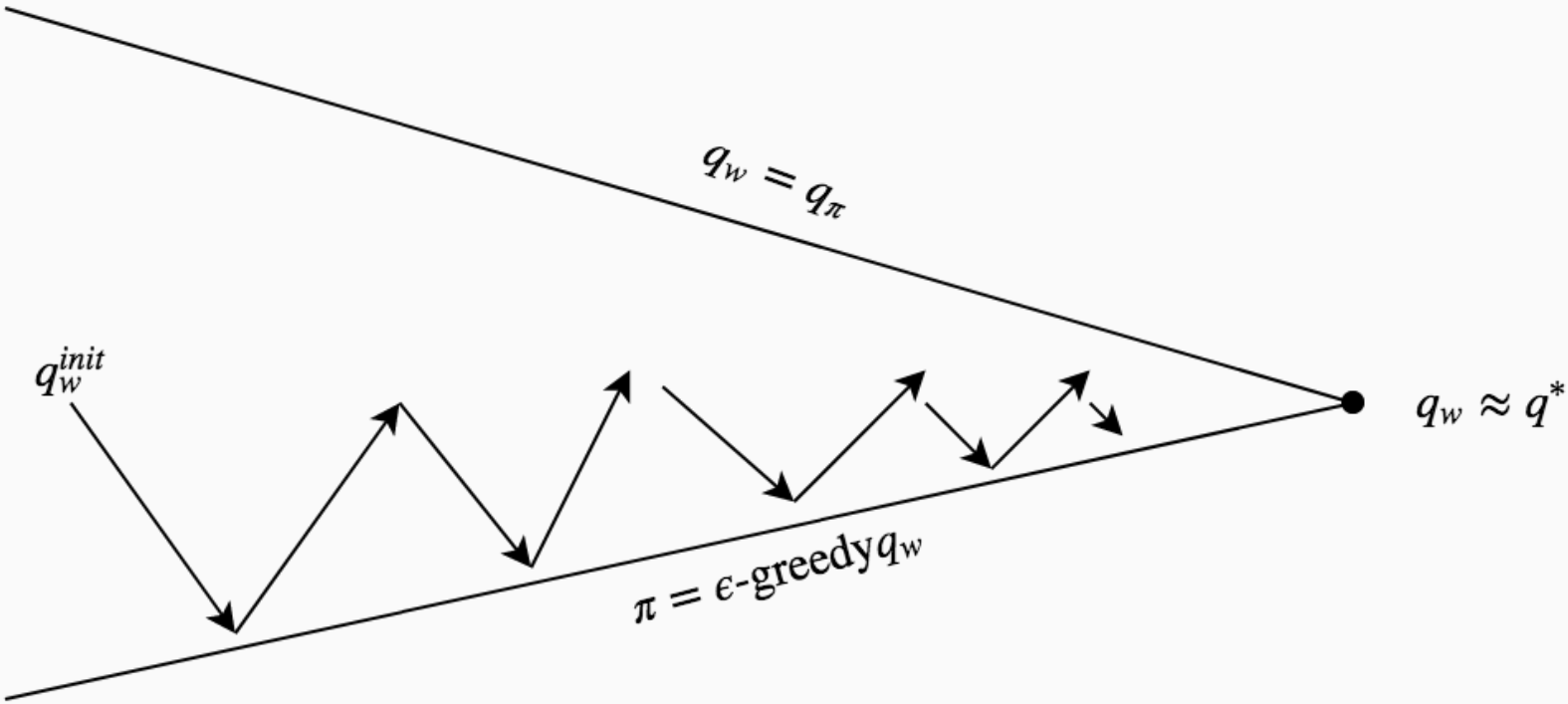
$$\nabla_w = \alpha(G_t - \hat{v}(S; w))\nabla_w \hat{v}(S; w) \tag{23}$$

- **TD(0)** - target = TD target ($R_{t+1} + \gamma \hat{v}(S_{t+1}; w)$)

$$\nabla_w = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}; w) - \hat{v}(S; w))\nabla_w \hat{v}(S; w) \tag{24}$$

Control with Value Function Approximation

- **Policy evaluation** - approximate parametric action-value function: $\hat{q}(\cdot, \cdot; w) \approx q_\pi$
- **Policy improvement** - ϵ-greedy policy improvement



SGD with Experience Replay

- Collect experience - tuples (state, value) pairs:

$$\mathcal{D} = (s_1, v_\pi), (s_2, v_\pi), ..., (s_T, v_\pi) \tag{25}$$

- Repeat:
  - Sample (state, value) from experience $s, v_\pi \sim \mathcal{D}$
  - Apply SGD update:

$$\nabla_w = \alpha(v_\pi(s) - \hat{v}(S; w))\nabla_w \hat{v}(s; w) \tag{26}$$

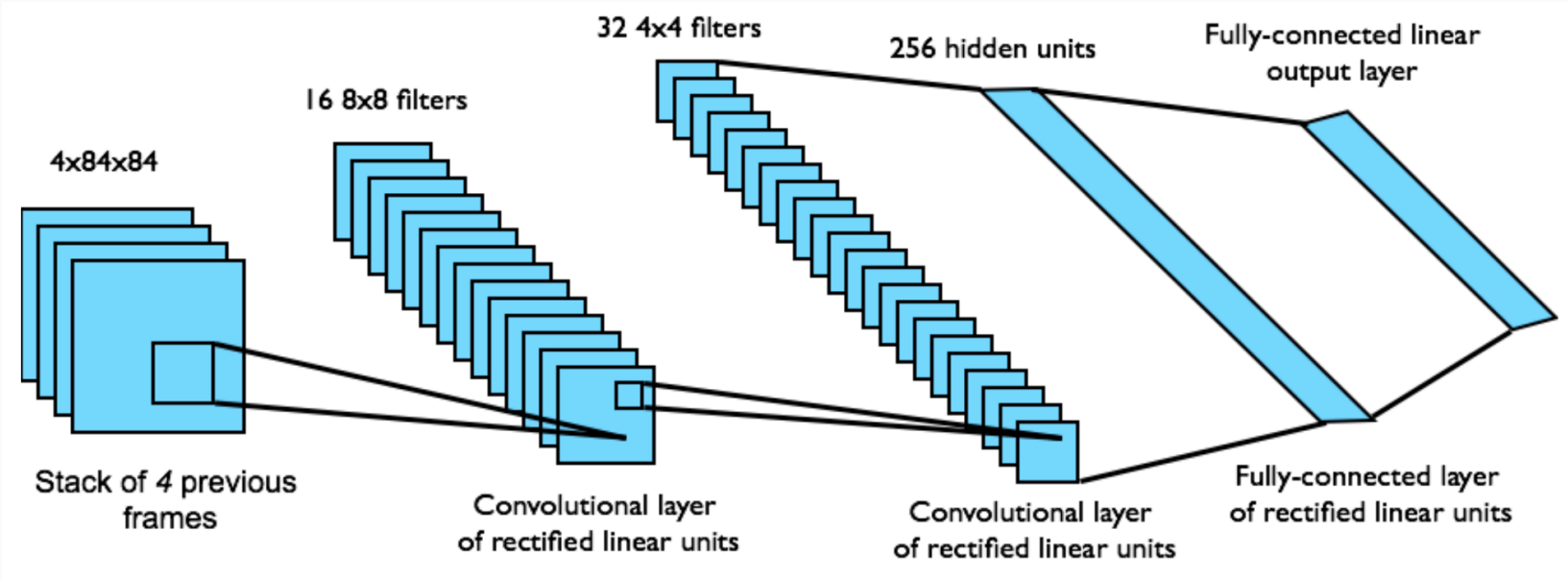
Experience Replay in Deep Q-Networks (DQN)

- DQN uses experience replay and fixed Q-targets
- Take action at 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
- Optimize MSE between Q-network and Q-learning targets using SGD:

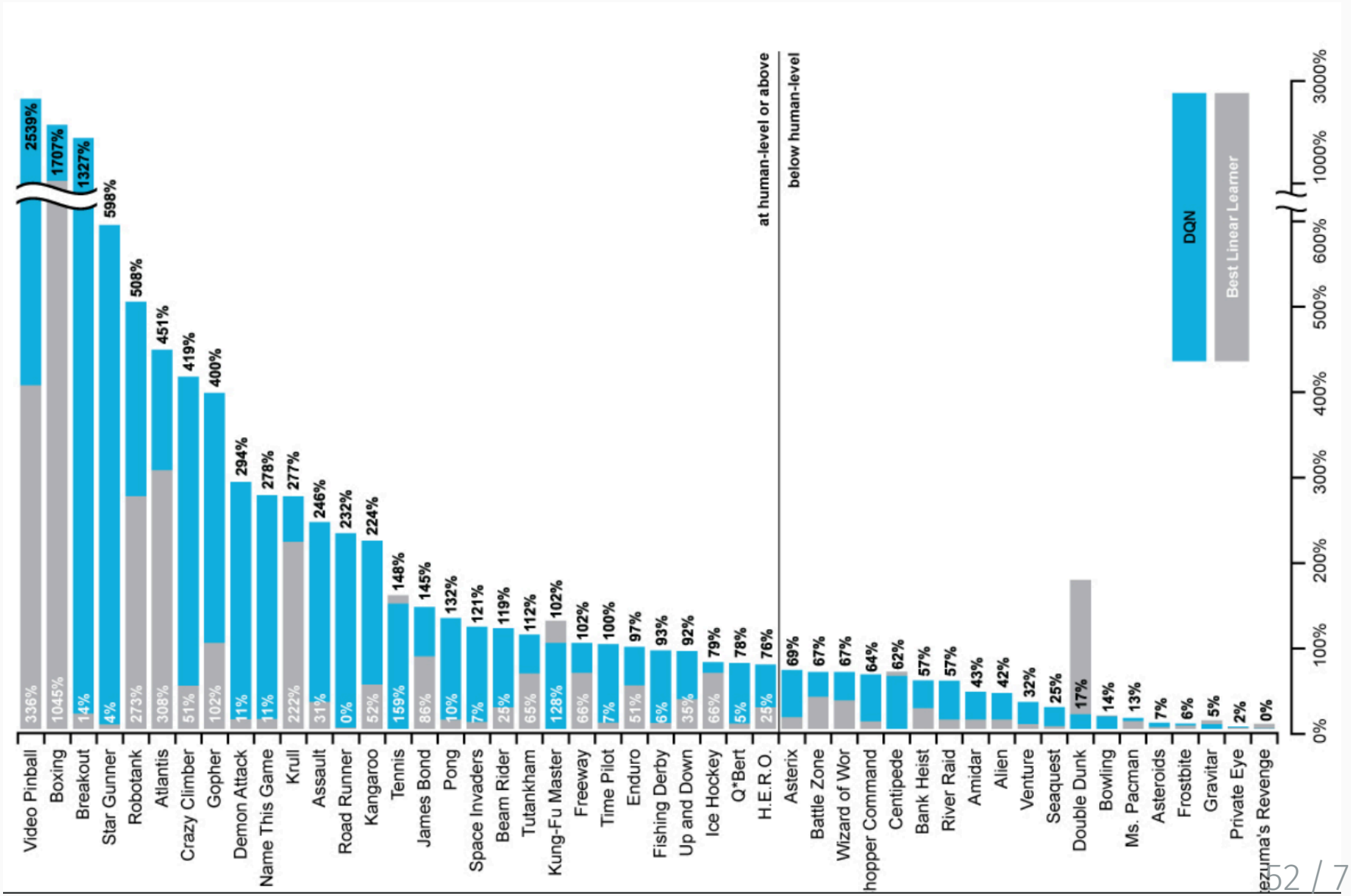
$$L_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}[r + \gamma \max a' Q(s', a'; \overline{w_i}) Q(s, a; w_i)^2] \qquad (27)$$

DQN in Atari

- **End-to-end learning** of values Q(s, a) from pixels s
- State s is a 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 in Atari



Policy Gradients

Policy Parametrization Methods

- **Previously...**
  - learn value functions, implicit policy (ϵ-greedy) w.r.t. value function
  - assume discrete action-space since we are unable to take the max of a continuous space of actions
- **Now...**
  - represent the policy explicitly - parameterized stochastic policies $\pi_\theta(a|s)$, where θ represents the parameters of the policy.
  - do not make any assumptions on the action space
  - parameterizing the policy might be a simpler function to approximate
  - action probabilities change smoothly as a function of the learned parameter (vs. ϵ-greedy selection - change dramatically for an arbitrary small change in the estimated action values)

Policy Based Methods

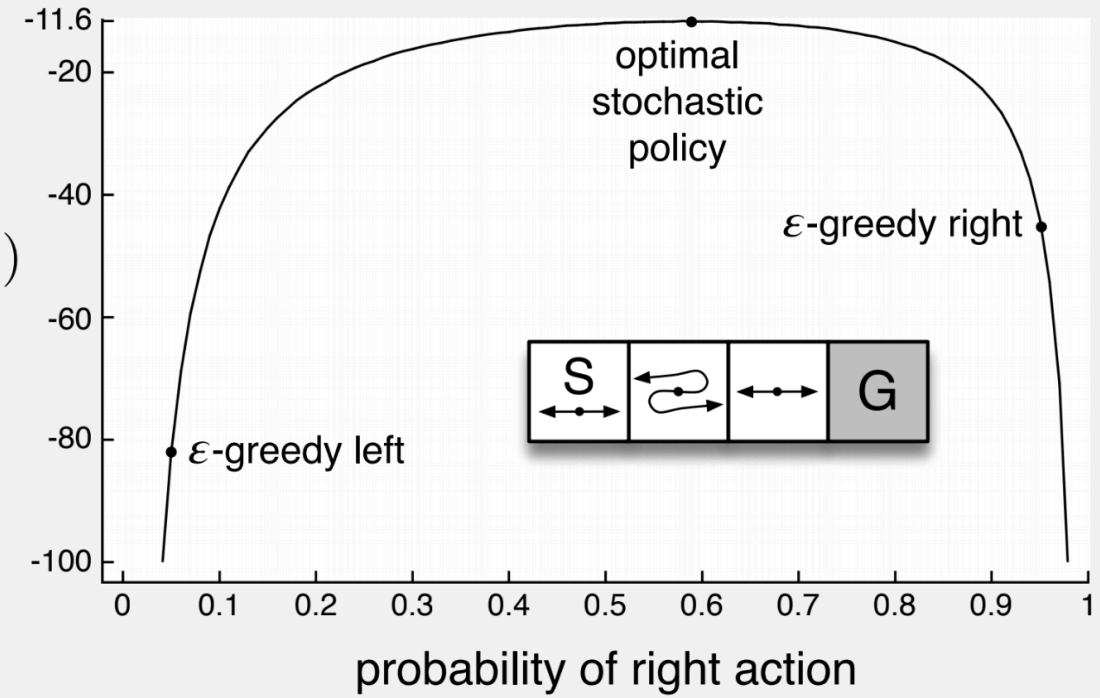
- No value function
- Parametrized policy

Actor-Critic

- Parametrized value function
- Parametrized policy

Example: Short corridor with switched actions

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(\mathrm{S})$$



The Policy Gradient Theorem - episodic case

- Performance measure = the value of the start state of the episode:

$$J(\theta) = v_{\pi_\theta}(s_0)$$
$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \tag{28}$$

- The constant of proportionality is the length of the episode, and is 1 in the continuing case (equality)
- μ = the on-policy distribution under π

Proof Sketch of the Policy Gradient Theorem

$$
\begin{aligned}
\nabla v_\pi(s) &= \nabla[\sum_a \pi(a|s) q_\pi(s,a)], \text{for } s \in \mathcal{S} \\
&= \sum_a [\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a)] \text{ (product rule of calculus)} \\
&= \sum_a [\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla \sum_{s,r} p(s',r|s,a)(r + v_\pi(s'))] \\
&= \sum_a [\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_s p(s'|s,a) \nabla v_\pi(s')] \\
&= \sum_a [\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_s p(s'|s,a) \text{ (unrolling)} \\
&\sum_{a'} [\nabla \pi(a'|s') q_\pi(s',a') + \pi(a'|s') \sum_{s''} p(s'|s,a) \nabla v_\pi(s'')]] \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} Pr(s \to x, k, \pi) \sum a \nabla \pi(a|x) q_\pi(x,a) \\
&\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \qquad (29)
\end{aligned}
$$

REINFORCE: Monte Carlo Policy Gradient

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a)$$

$$\nabla J(\theta) = \mathbb{E}_\pi[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\theta)]$$

$$\nabla J(\theta) = \mathbb{E}_\pi[\sum_a \pi(a|S_t,\theta) q_\pi(S_t,a) \frac{\nabla \pi(a|S_t,\theta)}{\pi(a|S_t,\theta)}]$$

$$\nabla J(\theta) = \mathbb{E}_\pi[q_\pi(S_t,A_t) \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}] \text{ replacing a by the sample} A_t \sim \pi)$$

$$\nabla J(\theta) = \mathbb{E}_\pi[G_t \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}]$$

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}$$

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla ln \pi(A_t|S_t,\theta_t)$$

(30)

REINFORCE - Vanilla PG algorithm

REINFORCE

Input: a differentiable policy parameterization $\pi(a|s,\theta)$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\theta \in \mathcal{R}^d$ (e.g., to 0)
for *each episode* **do**
  Generate an episode $S_0, A_0, R_1, .., S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\theta)$
  for *each each step of the episode* $t = 0, 1, ..., T-1$ **do**
    $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta ln \pi_\theta(A_t|S_t,\theta)$
  end
end

REINFORCE with Baseline

- The policy gradient theorem can be generalized to include a comparison of the action-value to an arbitrary baseline b(s) that does not depend on the actions:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s)(q_\pi(s,a) - b(s)) \tag{31}$$

- the equation remains valid because the subtracted quantity is zero:

$$\sum_a b(s) \nabla \pi(a|s, theta) = b(s) \nabla \sum_a \pi(a|s, theta) = b(s) \nabla 1 = 0 \tag{32}$$

- the update rule for of REINFORCE that includes a general baseline:

$$\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t)) \nabla ln \pi(A_t|S_t, \theta_t) \tag{33}$$

Actor-Critic Methods

- Baseline - used as a critic, it **bootstraps** (updating the value estimate for a state from the estimated values of subsequent states)
- Reduces variance, introduces bias
- **Critic** updates the action-value function parameters w,
- **Actor** updates the policy parameters θ, in the direction suggested by the critic.
- One-step Actor-Critic:

$$\begin{aligned} \nabla_\theta J(\theta) &\approx \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) Q_w(s,a)] \\ \nabla_\theta &= \nabla_\theta - \alpha \nabla_\theta log \pi_\theta(s,a) Q_w(s,a) \end{aligned} \tag{34}$$

- The value function parameters w minimise the mean-squared error:

$$\mathbb{E}_{\pi_\theta}(Q^{\pi_\theta}(s,a) - Q_w(s,a))^2 \tag{35}$$

Reducing the variance of Actor-Critic methods

- **Advantage function**, denoted with A:

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \tag{36}$$

- One-step Actor-Critic with advantage function:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) A^{\pi_\theta}(s,a))] \tag{37}$$

- TD error

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s) \tag{38}$$

- One-step Actor-Critic with TD error:

$$\begin{aligned} &\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) A^{\pi_\theta}(s,a))] \\ &\mathbb{E}_{\pi_\theta}[\delta^{\pi_\theta}|s,a] = \mathbb{E}_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s')|s,a] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s)(s,a) \end{aligned} \tag{39}$$

Frontiers

The challenge of perception

- Humans can reuse past knowledge
- Transfer learning in deep RL is an open problem
- How sensitive are RL policies to shadows, motion blur, viewpoint change, and clutter?
- How to disentangle perception representation from policy representation?
- How to learn generalizable feature spaces?
- How to compactly summarize information about the distant past?

The challenge of building models of the world

- Relieve the pressure of data efficiency
- Simulation should be an invaluable asset
- Simulators are ”doomed to succeed”
- How to introduce uncertainty in the dynamics model estimation?
- How to switch between model-free, model-based, maybe build a framework that explains both types of behavior encountered in humans and animals.

The challenge of reward engineering

- Reward signals are arbitrarily sparse, not structured, non-informative, sometimes completely absent!
- Not clear what the reward function should be
- Hard to engineer, even harder to make foolproof
- How do I know if the robot reaches the goal?
- How do I know if the block is lifted?
- How do I know if the legos are stacked?
- Inverse reinforcement learning.
- Learning from demonstrations/curriculum learning

The challenge of task scaling and complexity

- We don't want to perform just simple short tasks
- Few methods attempt to learn long, complex, multi-parted tasks
- Constitutionality, generalization between tasks, abstraction

The challenge of data efficiency

- Humans can learn incredibly quickly
- Deep RL methods are usually slow - they learn from millions of sample interactions with an environment

Intrinsic motivation: Exploration, Curiosity, Surprise

- Should solve the challenge of reward shaping and data efficiency
- Learning solely from external rewards is unnatural and unnecessary
- Explore intrinsic motivations such as curiosity in a general way
- The ”reinforcement signal” (TD-error) has grounds in neuroscience research as a driver for phasic activation of neurons.

Continual (lifelong) learning and Meta-learning

- Should solve the challenge of data efficiency, task scaling and complexity
- A learning framework or algorithm that enables forward transfer (learning new tasks faster)
- Not forgetting (retaining previous skills)

Hierarchical RL

- Should solve the challenge of data efficiency, task scaling and complexity
- The objective of HRL is to build temporal abstraction.
- Encapsulate primitive actions in higher-level abstract actions/behavior (making coffee, having a shower)
- Enables long-term credit assignment, high-level planning, skill composition

Learning from Demonstrations/Imitation learning

- Should solve the challenge of reward shaping, task scaling and complexity.
- Human and animal learning relies heavily on imitation learning.
- A necessary component of any general learning approach.
- Inverse reinforcement learning.

Building generative models of the world

- Planning
- Exploration
- Guarantee safety
- Generalization for different goals

Thank you.
Questions?

References

- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. `http://incompleteideas.net/book/the-book-2nd.html`
- CS 294: Deep Reinforcement Learning, Spring 2017 `http://rll.berkeley.edu/deeprlcoursesp17/`
- UCL Course on RL `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.`
- Mnih et al - *Human-level control through deep reinforcement learning*.