



REINFORCEMENT LEARNING APPROXIMATE AND DEEP

Odalric-Ambrym Maillard

2021-2022

Inria Scool

Slides are mix of O. Pietquin, A. Lazaric, M. Pirotta, H. Van Hasselt, G. Neuholdmann

- ▷ LSTD LSPI, Fitted-Sarsa, Fitted Q-learning.
- ▷ DQN, DDQN
- ▷ REINFORCE
- ▷ PPO, TRPO

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS



Value Iteration

RL

Olivier
Pietquin

Introduction

MDP

Dynamic
Programming

Bellman
Equations

Algorithms

Value iteration algorithm

initialize $V_0 \in \mathcal{V}$

$n \leftarrow 0$

while $\|V_{n+1} - V_n\| > \varepsilon$ **do**

for $s \in S$ **do**

$$V_{n+1}(s) = \max_a \sum_{s'} \mathcal{T}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_n(s')]$$

end for

$n \leftarrow n + 1$

end while

for $s \in S$ **do**

$$\pi(s) = \operatorname{argmax}_{a \in A} \sum_{s'} \mathcal{T}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_n(s')]$$

end for

return V_n, π



Policy Iteration

RL

Olivier
Pietquin

Introduction

MDP

Dynamic
Programming

Bellman
Equations

Algorithms

Policy iteration algorithm

Init $\pi_0 \in \mathcal{D}$

$n \leftarrow 0$

while $\pi_{n+1} \neq \pi_n$ **do**

solve (Evaluation phase)

$$V_{n+1}(s) = \sum_{s'} \mathcal{T}_{ss'}^{\pi(s)} [\mathcal{R}_{ss'}^a + \gamma V_n(s')] \quad (\text{Linear eq.})$$

for $s \in S$ **do** (Improvement phase)

$$\pi_{n+1}(s) = \operatorname{argmax}_{a \in A} \sum_{s'} \mathcal{T}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_n(s')]$$

end for

$n \leftarrow n + 1$

end while

return V_n, π_{n+1}



Monte Carlo Methods

RL

Olivier
Pietquin

Introduction

Problem
Definition

Monte Carlo
Methods

Temporal
Differences

Exploration
Management

Conclusion

Learning $V^\pi(s)$ through sampling

- Random choice of a starting state $s \in S$
- Follow the policy π and observe the cumulative gain R_t
- Do this infinitely and average: $V^\pi(s) = E^\pi[R_t]$



Monte Carlo Methods

RL

Olivier
Pietquin

Introduction

Problem
Definition

Monte Carlo
Methods

Temporal
Differences

Exploration
Management

Conclusion

Learning $V^\pi(s)$ through sampling

- Random choice of a starting state $s \in S$
- Follow the policy π and observe the cumulative gain R_t
- Do this infinitely and average: $V^\pi(s) = E^\pi[R_t]$

Learning $Q^\pi(s, a)$ by sampling

- Random choice of a starting state $s \in S$
- Random choice of an action $a \in A$ (*exploring starts*)
- Follow policy π and observe gain R_t
- Do that infinitely and average : $Q^\pi(s, a) = E^\pi[R_t]$
- Enhance the policy : $\pi(s) = \text{argmax}_{a \in A} Q^\pi(s, a)$



Problem

Dynamic Programming

- Requires knowing the system's dynamics
- But takes the structure into account :

$$\forall s \in S \quad V^*(s) = \max_{a \in A} E(r(s, a) + \gamma \sum_{s' \in S} T_{ss'}^a V^*(s'))$$

Monte Carlo

- No knowledge is necessary
- No consideration is made of the structure :
$$Q^\pi(s, a) = E^\pi[R_t]$$
- So, the agent has to wait until the end of the interaction to improve the policy
- High variance

RL

Olivier
Pietquin

Introduction

Problem
Definition

Monte Carlo
Methods

Temporal
Differences

Exploration
Management

Conclusion



Temporal Differences (TD) I

RL

Olivier
Pietquin

Introduction

Problem
Definition

Monte Carlo
Methods

Temporal
Differences

Q-Learning
Eligibility Traces

Exploration
Management

Conclusion

TD Principle

Ideal Case (deterministic) :

$$\begin{aligned}V(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \\&= r_t + \gamma V(s_{t+1})\end{aligned}$$

In practice :

$$\delta_t = [r_t + \gamma V(s_{t+1})] - V(s_t) \neq 0!$$

δ_t is the temporal difference error (TD error).

Note: $r(s_t, a_t) = r_t$

Note: target is now $r_t + \gamma V(s_{t+1})$ which is biased but with lower variance.



Temporal Differences (TD) II

RL

Olivier
Pietquin

Introduction

Problem
Definition

Monte Carlo
Methods

Temporal
Differences

Q-Learning
Eligibility Traces

Exploration
Management

Conclusion

New Evaluation method for V

Widrow-Hoff like update rule:

$$V^{t+1}(s_t) \leftarrow V^t(s_t) + \alpha (r_t + \gamma V^t(s_{t+1}) - V^t(s_t))$$

- α is the learning rate
- $V(s_t)$ is the target



Q-Learning

RL

Learn π^* following π_t (off-policy)

$$Q^{t+1}(s_t, a_t) \leftarrow Q^t(s_t, a_t) + \alpha(r_t + \gamma \max_b Q^t(s_{t+1}, b) - Q^t(s_t, a_t))$$

Q-learning Algorithm

for $n \leftarrow 0$ **until** $N_{tot} - 1$ **do**

$s_n \leftarrow \text{StateChoice}$

$a_n \leftarrow \text{ActionChoice}$

$(s'_n, r_n) \leftarrow \text{Simuler}(s_n, a_n)$

 % Update Q_n

begin

$Q_{n+1} \leftarrow Q_n$

$\delta_n \leftarrow r_n + \gamma \max_b Q_n(s'_n, b) - Q_n(s_n, a_n)$

$Q_{n+1}(s_n, a_n) \leftarrow Q_n(s_n, a_n) + \alpha_n(s_n, a_n)\delta_n$

end

end for

return $Q_{N_{tot}}$

Olivier
Pietquin

Introduction

Problem
Definition

Monte Carlo
Methods

Temporal
Differences

Q-Learning
Eligibility Traces

Exploration
Management

Conclusion



SARSA

Same for Q

$$Q^{t+1}(s_t, a_t) \leftarrow Q^t(s_t, a_t) + \alpha (r_t + \gamma Q^t(s_{t+1}, a_{t+1}) - Q^t(s_t, a_t))$$

SARSA

Init Q_0

for $n \leftarrow 0$ **until** $N_{tot} - 1$ **do**

$s_n \leftarrow \text{StateChoice}$

$a_n \leftarrow \text{ActionChoice} = f(Q^{\pi_t}(s, a))$

 Perform action a and observe s', r

begin

 Perform action $a' = f(Q^{\pi_t}(s', a'))$

$\delta_n \leftarrow r_n + \gamma Q_n(s'_n, a') - Q_n(s_n, a_n)$

$Q_{n+1}(s_n, a_n) \leftarrow Q_n(s_n, a_n) + \alpha_n(s_n, a_n)\delta_n$

$s \leftarrow s', a \leftarrow a'$ **end**

end for

return $Q_{N_{tot}}$

RL

Olivier
Pietquin

Introduction

Problem
Definition

Monte Carlo
Methods

Temporal
Differences

Q-Learning
Eligibility Traces

Exploration
Management

Conclusion



Q-Learning

RL

Olivier
Pietquin

Introduction

Problem
Definition

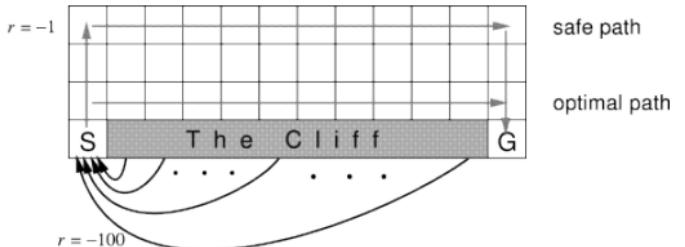
Monte Carlo
Methods

Temporal
Differences

Q-Learning
Eligibility Traces

Exploration
Management

Conclusion





Exploration Management

RL

Olivier
Pietquin

Introduction

Problem
Definition

Monte Carlo
Methods

Temporal
Differences

Exploration
Management

Conclusion

Action selection

- Greedy Selection : $a = a^* = \operatorname{argmax}_a Q(s, a)$
- ϵ -greedy selection : $P(a^*) = 1 - \epsilon$
- Softmax (Gibbs or Boltzmann) $P(a) = \frac{e^{Q(a)/\tau}}{\sum_{a'} e^{Q(a')/\tau}}$

Optimistic Initialization

- Initialize the value functions with high values so as to visit unseen states thanks to action selection rules.

Uncertainty and value of information

- Take uncertainty on the values into account.
- Compute the value of information provided by exploration.

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS

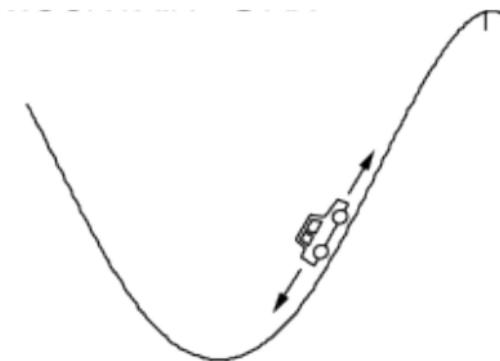
- ▷ \mathcal{S} States, \mathcal{A} Actions
- ▷ $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}([0, 1])$ **Reward** function (mean m)
- ▷ $P : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$ **Transition** function
- ▷ $\pi : \mathcal{S} \times \mathcal{A}$ **Policy**, or **class of policies** $\Pi = \{\pi_\theta : \theta \in \Theta\}$
- ▷ $V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s \right]$ **Value** function
- ▷ $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ **Quality** function

- ▷ \mathcal{S} States, \mathcal{A} Actions
- ▷ $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}([0, 1])$ **Reward** function (mean m)
- ▷ $P : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$ **Transition** function
- ▷ $\pi : \mathcal{S} \times \mathcal{A}$ **Policy**, or **class of policies** $\Pi = \{\pi_\theta : \theta \in \Theta\}$
- ▷ $V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s \right]$ **Value** function
- ▷ $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ **Quality** function

Challenge

How to **represent** R, P, π, Q, V_π when \mathcal{S}, \mathcal{A} are **huge** ? **continuous**?

EXAMPLE: MOUNTAIN-CAR



States: Position \times Speed of car (continuous).

Actions: Acceleration of car.

Transitions: Physics.

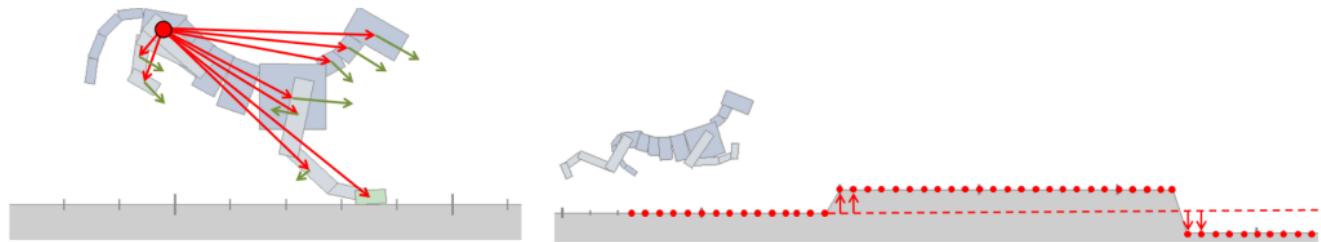
Rewards: 1 if reach top of mountain, 0 else.

Optimal policy: First go left to get momentum, then go right.

Continuous state-action space. For other classical tasks, check

https://gym.openai.com/envs/#classic_control.

EXAMPLE: MOTION-PLANNING



States: relative positions (red points) and velocities.

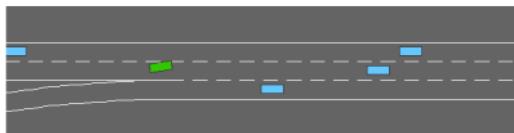
Actions: acceleration at each joint

Rewards: not falling (height of anchor point).

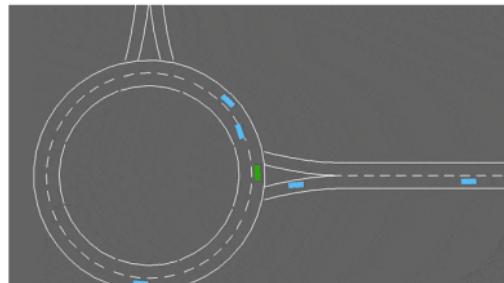
EXAMPLE: HIGH-WAY

Controlling an ego-vehicle in various road environments

<https://github.com/eleurent/highway-env>



The merge-v0 environment.



The roundabout-v0 environment.

Actions: acceleration, changing lane, etc.

Rewards: no collision, speed, etc.

Study safety and robustness (here with partially known behavior of other vehicles).

<http://edouardleurent.com/publication/phd-thesis/>



The Curse of Dimensionality (Bellman) I

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Control

Warnings

Deep
Q-Network

Some examples

- BackGammon: 10^{20} states [Tesauro, 1995]
- Chess: 10^{50} states
- Go: 10^{170} states, 400 actions [Silver et al., 2016]
- Atari: 240x160 continuous dimensions [Mnih et al., 2015]
- Robotics: multiple degrees of freedom
- Language: very large discrete action space

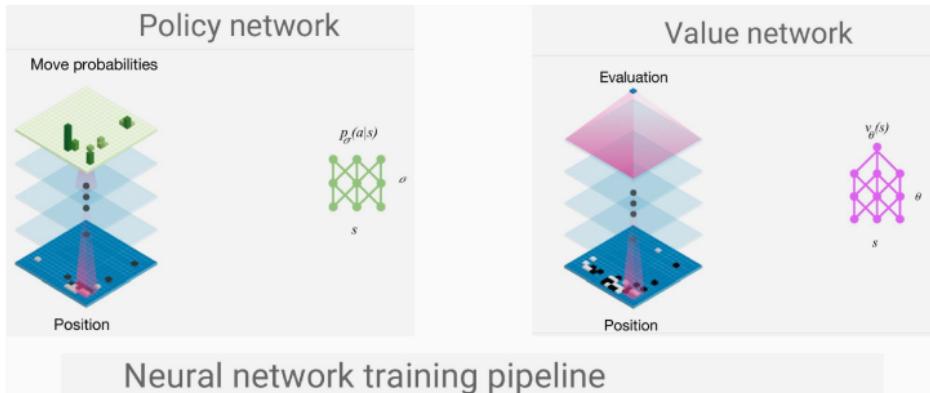
Tabular RL

Complexity is polynomial. Doesn't scale up.

- ▷ DP (finite-horizon), VI, PI (greedy op.) require to know the model P, R and to **represent** the functions V, π .
- ▷ TD methods need to **represent** V, π but do not require P, R .
- ▷ Q-learning needs to **represent** Q , greedy operator **no longer** needs P, R .

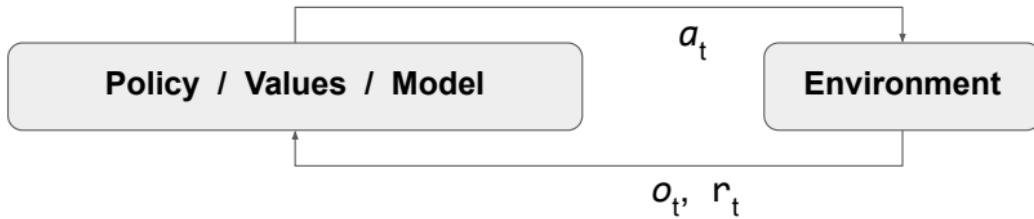
TYPES OF RL ALGORITHMS

- ▷ **Critic**: Value function oriented Q (SARSA, Q-learning, TD, etc.)
- ▷ **Actor**: Policy oriented π (Policy gradient, Reinforce, PPO, TRPO)
- ▷ **Actor-critic**: use both (e.g. A3C). e.g. AlphaGo

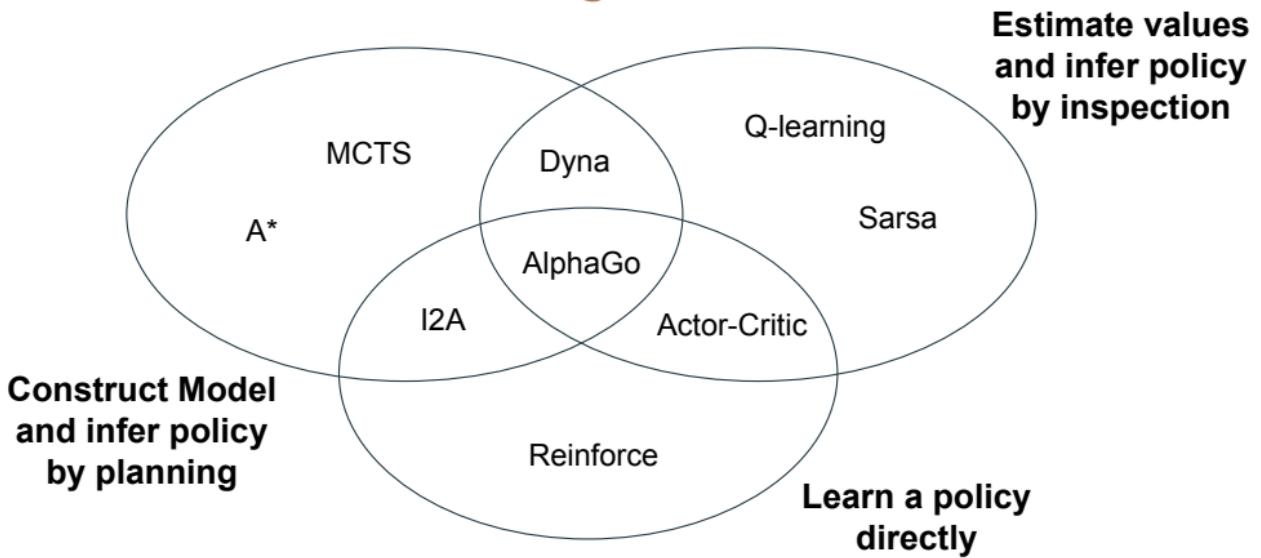


RL Solutions: The Big Picture

- Learn **policy** directly - **execute it**
- Estimate **values** of each action - infer policy by **inspection**
- Construct a **model** - infer policy by **planning**



RL Solutions: The Big Picture



Deep Reinforcement Learning

Policy $\pi : S_t \rightarrow p(A_t)$

Value $v : S_t \rightarrow V(S_t)$

Model $m : S_t, A_t \rightarrow S_{t+1}, R_{t+1}$

Ultimately these are just **functions**,
and we need a flexible way of **representing** and **learning** them

Deep Reinforcement Learning

Policy $\pi : S_t \rightarrow p(A_t)$

Value $v : S_t \rightarrow V(S_t)$

Model $m : S_t, A_t \rightarrow S_{t+1}, R_{t+1}$

Ultimately these are just **functions**,
and we need a flexible way of **representing** and **learning** them

Deep Learning



Why Deep Learning?

Deep learning is not the only answer, but it's a pretty convenient one

- **Table Look Up?**
 - Learn about each state separately
 - No **generalization**, does not scale to large state spaces
- **Linear?**
 - $y_t = \mathbf{w}^T S_t + b$
 - Can work very well if you have a rich state representation S_t
 - Hard-coding a suitable **rich state representation** S_t is difficult!

- ▶ state-action value function (aka **Q-function**, **quality function**)

$$Q_\pi(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) | S_0 = s, A_0 = a, S_{t+1} \sim p(\cdot | S_t, A_t), A_{t+1} = \pi(S_{t+1}) \right].$$

- ▶ **state-action value function** (aka **Q-function**, **quality function**)

$$Q_\pi(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) | S_0 = s, A_0 = a, S_{t+1} \sim \mathbf{p}(\cdot | S_t, A_t), A_{t+1} = \pi(S_{t+1}) \right].$$

- ▶ **Bellman evaluation operator** $T_\pi : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \rightarrow \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$

- ▶ definition: $[T_\pi Q](s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{p}(s' | s, a) Q(s', \pi(s'))$;
- ▶ Q_π is its unique fixed point: $T_\pi Q_\pi = Q_\pi$;
- ▶ link to v_π : $v_\pi(s) = Q_\pi(s, \pi(s))$.

RECAP: THE Q-FUNCTION

- ▶ **state-action value function** (aka **Q-function**, **quality function**)

$$Q_\pi(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) | S_0 = s, A_0 = a, S_{t+1} \sim \mathbf{p}(\cdot | S_t, A_t), A_{t+1} = \pi(S_{t+1}) \right].$$

- ▶ **Bellman evaluation operator** $T_\pi : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \rightarrow \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$

- ▶ definition: $[T_\pi Q](s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{p}(s'|s, a) Q(s', \pi(s'))$;
- ▶ Q_π is its unique fixed point: $T_\pi Q_\pi = Q_\pi$;
- ▶ link to v_π : $v_\pi(s) = Q_\pi(s, \pi(s))$.

- ▶ **Bellman optimality operator** $T_* : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \rightarrow \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$

- ▶ definition: $[T_* Q](s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{p}(s'|s, a) \max_{a' \in \mathcal{A}} Q(s', a')$;
- ▶ Q_* is its unique fixed point: $Q_* = T_* Q_*$;
- ▶ link to v_* : $v_*(s) = \max_{a \in \mathcal{A}} Q_*(s, a)$.

▷ Allows acting **greedily**:

▶ resp to $V^\pi = Q_\pi(s, \pi(s))$:

$$\begin{aligned} \pi' \in \mathcal{G}(V^\pi) &\Leftrightarrow \forall s \in \mathcal{S}, \quad \pi'(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{p}(s'|s, a) V^\pi(s') \right) \\ &\Leftrightarrow \forall s \in \mathcal{S}, \quad \pi'(s) \in \operatorname{argmax}_{a \in \mathcal{A}} Q_\pi(s, a). \end{aligned}$$

▶ resp. to V^* :

$$\star(s) \in \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a).$$

▶ resp. to any $Q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$:

$$\forall Q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}, \quad \pi \in \mathcal{G}(Q) \Leftrightarrow \forall s \in \mathcal{S}, \quad \pi(s) \in \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a).$$

- ▷ Allows **sampling easily** the related operators
 - ▶ recall the dataset

$$\mathcal{D} = \{(s_i, a_i, r_i, s'_i)_{1 \leq i \leq n}\}.$$

- ▶ sampled **Bellman evaluation** operator

$$[\hat{T}_\pi Q](s_i, a_i) = r_i + \gamma Q(s'_i, \pi(s'_i));$$

- ▶ sampled **Bellman optimality** operator

$$[\hat{T}_* Q](s_i, a_i) = r_i + \gamma \max_{a' \in \mathcal{A}} Q(s'_i, a').$$

- ▷ This motivates focusing on **representing the function Q .**

THE NEED FOR FUNCTION APPROXIMATION

- When \mathcal{S} or \mathcal{A} are discrete, we can represent $Q(s, a)$ or $\pi(s)$ with **tables**.

$Q(s, a)$	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
a_1	0	10	0	15	30	0	0	20	5
a_2	0	0	20	0	0	70	80	30	10
a_3	10	5	30	45	20	0	10	10	85

- When \mathcal{S} or \mathcal{A} are continuous (or too large), this is **no longer** possible.

THE NEED FOR FUNCTION APPROXIMATION

- When \mathcal{S} or \mathcal{A} are discrete, we can represent $Q(s, a)$ or $\pi(s)$ with **tables**.

$Q(s, a)$	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
a_1	0	10	0	15	30	0	0	20	5
a_2	0	0	20	0	0	70	80	30	10
a_3	10	5	30	45	20	0	10	10	85

- When \mathcal{S} or \mathcal{A} are continuous (or too large), this is **no longer** possible.

Need for **function representation**

Example: express a function $f : \mathcal{X} \rightarrow \mathbb{R}$ as **weighted sum** of **known** functions $\varphi_1, \varphi_2, \dots, \varphi_D : \mathcal{X} \rightarrow \mathbb{R}$ with **unknown** parameters $\theta_1, \dots, \theta_D \in \mathbb{R}$:

$$f(x) = \theta_1 \varphi_1(x) + \theta_2 \varphi_2(x) + \dots = \sum_{d=1}^D \theta_d \varphi_d(x)$$

$$f(x) = f_\theta(x) = \langle \theta, \varphi(x) \rangle$$

- Only need to store D values: $\theta = (\theta_1, \dots, \theta_D) \in \mathbb{R}^D$.



Value Function Approximation

Parametric approximation

The value function (or Q -function) will be expressed as a function of a set of parameters θ_i :

$$\hat{V}^\pi(s) = V_\theta(s) = V(s, \theta) \quad \hat{Q}^\pi(s, a) = Q_\theta(s, a) = Q(s, a, \theta)$$

where θ is the (column) vector of parameters: $[\theta_i]_{i=1}^p$

Method

Search in space $\mathcal{H} = \{V_\theta(s) (\text{resp. } Q_\theta(s, a)) | \theta \in \mathbb{R}^p\}$ generated by parameters θ_i for the best fit to $V^\pi(s)$ (resp. $Q^\pi(s, a)$) by minimizing an objective function $J(\theta)$.

Goal

Learn optimal parameters $\theta^* = \operatorname{argmin}_\theta J(\theta)$ from samples.

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Control

Warnings

Deep
 Q -Network



Types of parameterizations I

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Control

Warnings

Deep
Q-Network

Linear function approximation

$$V_\theta(s) = \sum_{i=0}^p \theta_i \phi_i(s) = \theta^\top \phi(s)$$

where $\phi_i(s)$ are called basis functions (or features) and define \mathcal{H} and $\phi(s) = [\phi_i(s)]_{i=1}^p$.

Look up table

- It is a special case of linear function approximation
- Parameters are the value of each state ($\theta_i = V(s_i)$ and $p = |S|$)
- $\phi(s) = \delta(s) = [\delta_i(s)]_{i=1}^{|S|}$ where $\delta_i(s) = 1$ if $s = s_i$ and 0 otherwise



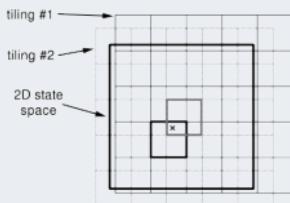
Types of parameterizations II

Neural networks

- θ is the vector of synaptic weights
- Inputs to the network is either s or (s, a)
- Either a single output for $V_\theta(s)$ or $Q_\theta(s, a)$ or $|A|$ outputs (one for each $Q_\theta(a_j, s)$)

Other approximations

- Tile Coding



Shape of tiles \Rightarrow Generalization

#Tilings \Rightarrow Resolution of final approximation

- Regression trees, nearest neighbours etc.

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Control

Warnings

Deep
Q-Network

- **Linear** representation of the Q-function $Q \in \mathcal{F}$:

$$\mathcal{F} = \left\{ f(x) = \sum_{d=1}^D \theta_d \varphi_d(x), \theta \in \mathbb{R}^D \right\}$$

- ▶ Linear regression, Wavelets, etc.
- ▶ Kernel regression (infinitely many base functions, but kernel trick).

- ▷ **Linear** representation of the Q-function $Q \in \mathcal{F}$:

$$\mathcal{F} = \left\{ f(x) = \sum_{d=1}^D \theta_d \varphi_d(x), \theta \in \mathbb{R}^D \right\}$$

- ▶ Linear regression, Wavelets, etc.
- ▶ Kernel regression (infinitely many base functions, but kernel trick).
- ▷ **Non-Linear** representation:

$$\mathcal{F} = \left\{ f(x) = \Psi \left(\sum_{d_1=1}^{D_1} \theta_{d_1}^{(1)} \varphi_{d_1}^{(1)} \left(\sum_{d_2=1}^{D_2} \theta_{d_2}^{(2)} \varphi_{d_2}^{(2)} \left(\dots \theta_{d_K}^{(K)} \varphi_{d_K}^{(K)}(x) \right) \right) \dots \right), \theta \in \mathbb{R}^D \right\}$$

- ▶ Several "layers" of weighted sums: **Neural networks**.
- ▶ **Known** functions ($\Psi, \varphi^{(1)}, \varphi^{(2)}$, etc) between layers.
- ▶ Large number $D = \sum_{k=1}^K D_k$ of **unknown** parameters, but structured.

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

Linear representation

Non-Linear representation

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS

LINEAR REPRESENTATION: LEAST-SQUARES

- **Linear space:** $\mathcal{F} = \left\{ f_\theta : f_\theta(x) = \sum_{d=1}^D \theta_d \varphi_d(x), \theta \in \mathbb{R}^D, \theta \in \Theta \right\}$.
Ex: $\varphi(x) = (1, x, x^2)$, $f_\theta(x) = 2 + \frac{1}{2}x - 2x^2$, $\theta = (2, 1/2, -2)$.

LINEAR REPRESENTATION: LEAST-SQUARES

► **Linear space:** $\mathcal{F} = \left\{ f_\theta : f_\theta(x) = \sum_{d=1}^D \theta_d \varphi_d(x), \theta \in \mathbb{R}^D, \theta \in \Theta \right\}$.

Ex: $\varphi(x) = (1, x, x^2)$, $f_\theta(x) = 2 + \frac{1}{2}x - 2x^2$, $\theta = (2, 1/2, -2)$.

► **Objective** : from $(x_n, y_n)_{n \leq N}$ find parameter θ by optimizing

$$\min_{\theta \in \Theta} \sum_{n=1}^N \left(y_n - \theta^\top \varphi(x_n) \right)^2. \quad (1)$$

LINEAR REPRESENTATION: LEAST-SQUARES

► **Linear space:** $\mathcal{F} = \left\{ f_\theta : f_\theta(x) = \sum_{d=1}^D \theta_d \varphi_d(x), \theta \in \mathbb{R}^D, \theta \in \Theta \right\}$.

Ex: $\varphi(x) = (1, x, x^2)$, $f_\theta(x) = 2 + \frac{1}{2}x - 2x^2$, $\theta = (2, 1/2, -2)$.

► **Objective :** from $(x_n, y_n)_{n \leq N}$ find parameter θ by optimizing

$$\min_{\theta \in \Theta} \sum_{n=1}^N \left(y_n - \theta^\top \varphi(x_n) \right)^2. \quad (1)$$

► Any solution to (1) must satisfy (taking derivative)

$$G_N \theta = \sum_{n=1}^N \varphi(x_n) y_n, \text{ where } G_N = \sum_{n=1}^N \varphi(x_n) \varphi(x_n)^\top \text{ (*D* } \times \text{ *D matrix*)}.$$

► Notations: $Y_N = (y_1, \dots, y_N)^\top$, $\Phi_N = (\varphi^\top(x_1), \dots, \varphi^\top(x_N))^\top$ (*N* \times *D*)

$$G_N \theta = \Phi_N^\top Y_N, \text{ where } G_N = \Phi_N^\top \Phi_N.$$

► When $\Theta = \mathbb{R}^d$ and G_N is invertible,

$$\text{(Ordinary Least-squares)} \quad \theta_N = G_N^{-1} \Phi_N^\top Y_N.$$

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

Linear representation

Non-Linear representation

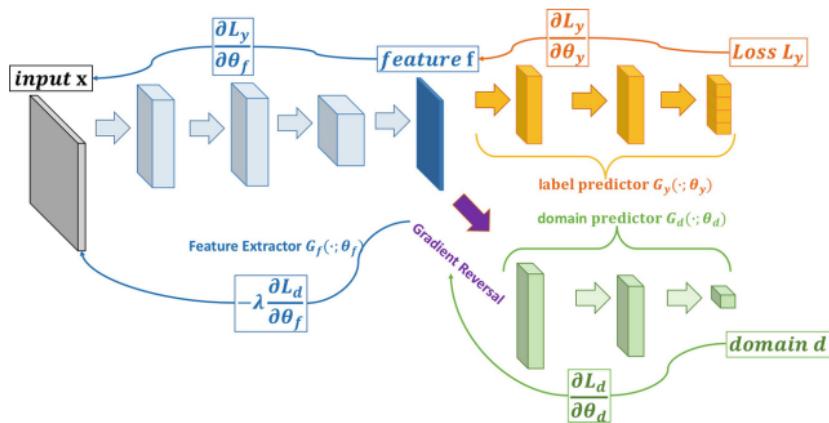
LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS

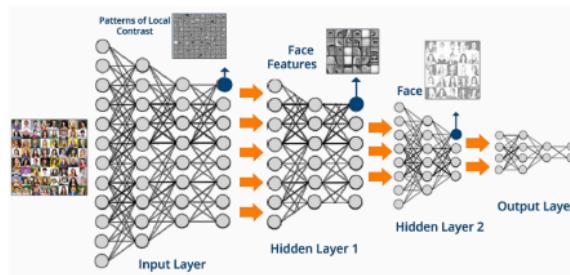
- To find parameter θ in **Non-linear** representation we also use **differentiation**: **gradient** methods.

Deep Learning revolution = **Automated differentiation** tools

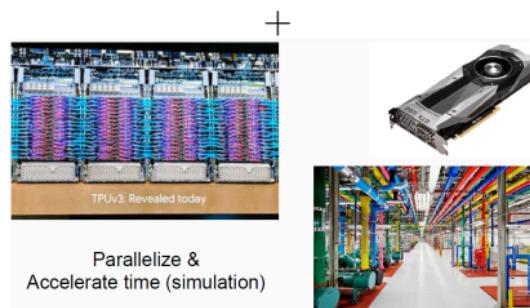


Warning

Auto-differentiation, gradient propagation requires a **fixed** architecture.



DL: Computer Vision breakthrough and AutoGrad



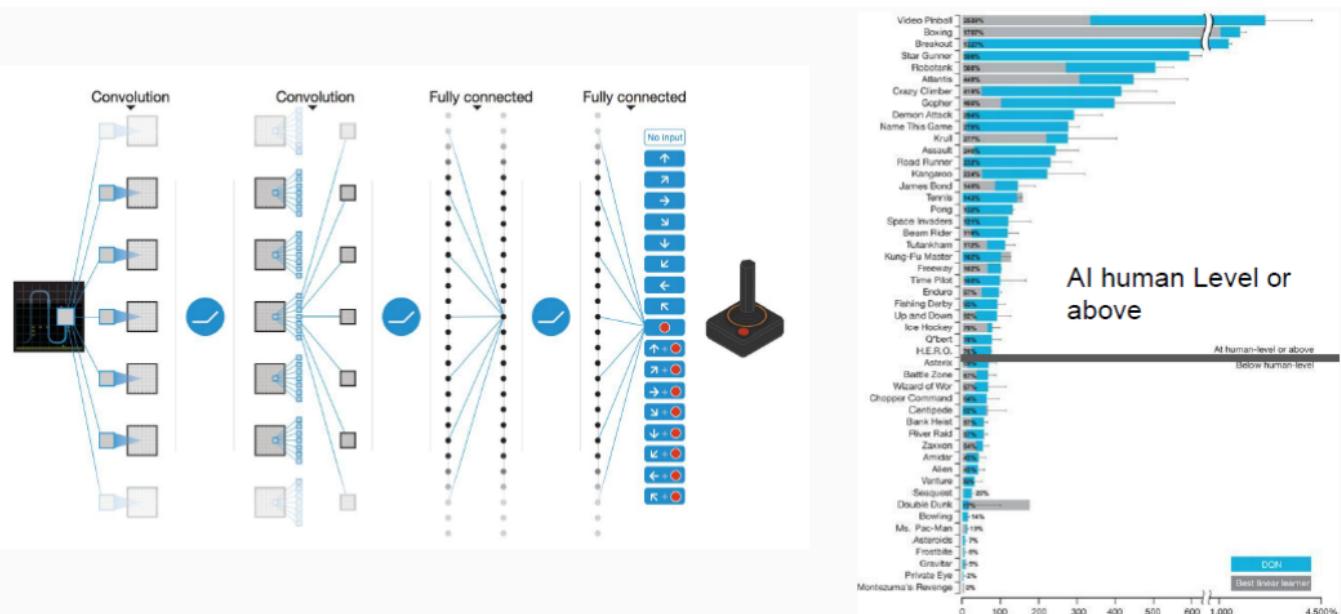
Computational Power



Reinforcement Learning Algorithm	Counterpart of Bellman Equation (top) Weight Change for Residual Algorithm (bottom)
TD(0)	$V(x) = \langle R + \gamma V(x') \rangle$ $\Delta w_r = -\alpha (R + \gamma V(x'_1) - V(x)) (\phi \gamma \frac{\partial}{\partial w} V(x'_2) - \frac{\partial}{\partial w} V(x))$
Value Iteration	$V(x) = \max_u \langle R + \gamma V(x') \rangle$ $\Delta w_r = -\alpha (\max_u \langle R + \gamma V(x') \rangle - V(x)) (\phi \frac{\partial}{\partial w} \max_u \langle R + \gamma V(x') \rangle - \frac{\partial}{\partial w} V(x))$
Q-learning	$Q(x, u) = \langle R + \gamma \max_{u'} Q(x', u') \rangle$ $\Delta w_r = -\alpha (R + \gamma \max_{u'} Q(x'_1, u') - Q(x, u)) (\phi \gamma \frac{\partial}{\partial w} \max_{u'} Q(x'_2, u') - \frac{\partial}{\partial w} Q(x, u))$
Advantage Learning	$A(x, u) = \left\langle R + \gamma^{\Delta t} \max_{u'} A(x', u') \right\rangle^{\frac{1}{\Delta t}} + (1 - \frac{1}{\Delta t}) \max_{u'} A(x, u')$ $\Delta w_r = -\alpha ((R + \gamma^{\Delta t} \max_{u'} A(x'_1, u'))^{\frac{1}{\Delta t}} + (1 - \frac{1}{\Delta t}) \max_{u'} A(x, u') - A(x, u))$ $\cdot (\phi \gamma^{\Delta t} \frac{\partial}{\partial w} \max_{u'} A(x'_2, u')^{\frac{1}{\Delta t}} + \phi (1 - \frac{1}{\Delta t}) \frac{\partial}{\partial w} \max_{u'} A(x, u') - \frac{\partial}{\partial w} A(x, u))$

Q-VALUE APPROXIMATION: DEEP Q-NETWORK (DQN)

- Representation from visual input (raw pixels)



Q-VALUE APPROXIMATION: DEEP Q-NETWORK (DQN)

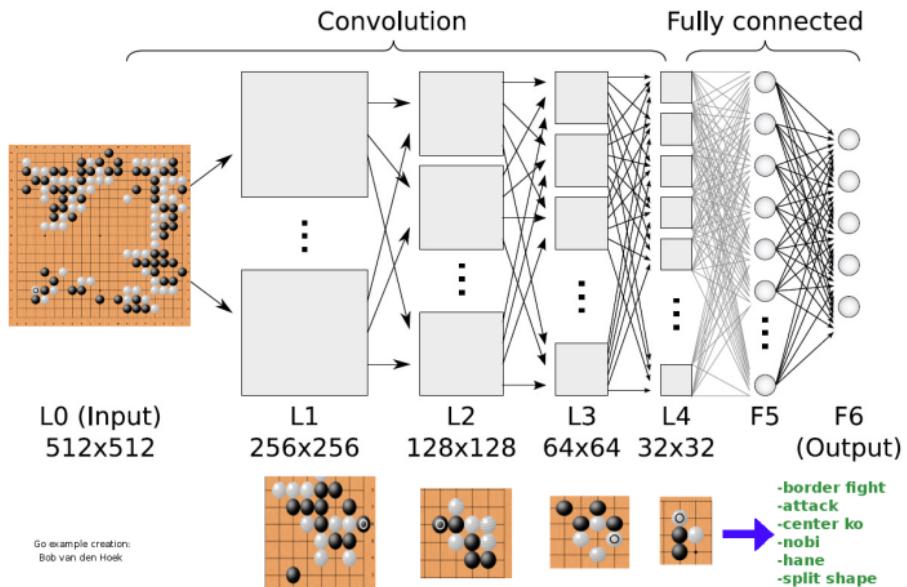


TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS



Least-Square Temporal Differences I

RL

Olivier
Pietquin

Introduction

Policy
EvaluationDirect methods
Residual
MethodsLeast-Square TD
Fitted-Value
Iteration

Control

Warnings

Deep
Q-Network

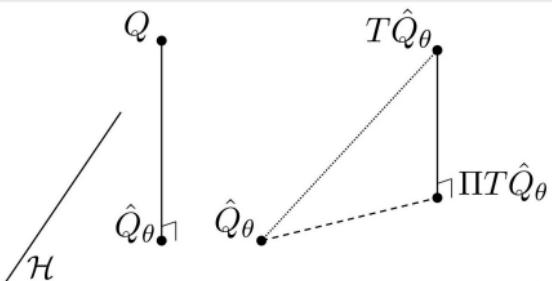
General idea (batch method)

Let's define Π as the projection operator such that:

$$\Pi V = \operatorname{argmin}_{V_\theta \in \mathcal{H}} \|V - V_\theta\|_{\nu, q}^q$$

Least-square TD minimizes the distance between the current estimate V_θ and the projection on \mathcal{H} of $T^\pi V_\theta(s)$

$$J(\theta) = \|V_\theta(s) - \Pi T V_\theta(s)\|_{\mu, p}^p$$





Least-Square Temporal Differences II

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Direct methods
Residual
Methods
Least-Square TD
Fitted-Value
Iteration

Control

Warnings

Deep
Q-Network

Two nested optimisation problems

$$J_1(\theta) = \frac{1}{N} \sum_{i=1}^N \|V_\theta(s_i) - V_\omega(s_i)\|^2$$

$$J_2(\omega) = \frac{1}{N} \sum_{i=1}^N \|V_\omega(s_i) - (r(s_i, a_i) + \gamma V_\theta(s_{i+1}))\|^2$$

Linear solution: LSTD [Bradtko and Barto, 1996, Boyan, 1999]

$$\theta^* = \left[\sum_{i=1}^N \phi(s_i) [\phi(s_i) - \gamma \phi(s_{i+1})]^\top \right]^{-1} \sum_{i=1}^N \phi(s_i) r(s_i, a_i).$$



Iterative projected fixed point

Fitted value iteration

Under some conditions, the composition of Π and T^π remains a contraction. The Fitted Value Iteration (FVI) procedure consists in iteratively applying the following rule:

$$V_\theta \leftarrow \Pi T V_\theta$$

In practice (batch method) [Gordon, 1995]

Collect a set of transitions with π : $\{s_i, a_i, r(s_i, a_i), s_{i+1}\}_{i=1}^N$

- Initialise θ_0
- Build a data set:
$$D_t = \{s_i, \hat{T}^\pi V_{\theta_t}(s_i)\} = \{s_i, r(s_i, a_i) + \gamma V_{\theta_t}(s_{i+1})\}_{i=1}^N$$
- Regress on D_t to find θ_{t+1}
- Iterate until convergence

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Direct methods
Residual
Methods
Least-Square TD
Fitted-Value
Iteration

Control

Warnings

Deep
Q-Network



Approximate SARSA

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Control

SARSA

LSPI
Fitted-Q

Warnings

Deep
Q-Network

Linear approximation of Q^π

$$Q_\theta(s, a) = \theta^\top \phi(s, a)$$

Linear SARSA

Init θ_0

for $n \leftarrow 0$ until $N_{tot} - 1$ do

$s_n \leftarrow \text{StateChoice}$

$a_n \leftarrow \text{ActionChoice} = f(Q_{\theta_t}(s_n, a))$

 Perform action a_n and observe $s_{n+1}, r(s_n, a_n)$

 begin

 Perform action $a_{n+1} = f(Q_{\theta_t}(s_{n+1}, a))$

$\delta_n \leftarrow r(s_n, a_n) + \gamma \theta_n^\top \phi(s_{n+1}, a_{n+1}) - \theta_n \phi(s_n, a_n)$

$\theta_{n+1} \leftarrow \theta_n + \alpha_n \delta_n \phi(s_n, a_n)$

$s_n \leftarrow s_{n+1}, a_n \leftarrow a_{n+1}$

 end

end for

return $\theta_{N_{tot}}$



Fitted- Q iteration

Replace V^π by Q^* [Riedmiller, 2005, Ernst et al., 2005]

Collect a set of transitions with π : $\{s_i, a_i, r(s_i, a_i), s_{i+1}\}_{i=1}^N$

- Initialise θ_0
- Build a data set: $D_t = \{(s_i, a_i), \hat{T}^* Q_{\theta_t}(s_i, a_i)\} = \{(s_i, a_i), r(s_i, a_i) + \gamma \max_b Q_{\theta_t}(s_{i+1}, b)\}_{i=1}^N$
- Regress on D_t to find θ_{t+1}
- (resample with $\pi = f(Q_\theta(s, a))$)
- Iterate until convergence

Output $\pi = \text{argmax } Q_\theta(s, a)$

Good point

There is no (yet) assumptions about parameterisation (no linear)

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Control
SARSA
LSPI
Fitted- Q

Warnings

Deep
Q-Network

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

Approximate value iteration

Approximate policy iteration

DEEP Q-NETWORK

POLICY GRADIENT METHODS

When \mathcal{S}, \mathcal{A} is huge, we may want to represent in Function spaces: m, p, V, Q or π .

1) Parameterize **Q function**

$$\mathcal{F} = \{Q_\theta(s, a) = \theta^\top \varphi(s, a), \theta \in \mathbb{R}^d\}$$

When \mathcal{S}, \mathcal{A} is huge, we may want to represent in Function spaces: m, p, V, Q or π .

1) Parameterize **Q function**

$$\mathcal{F} = \{Q_\theta(s, a) = \theta^\top \varphi(s, a), \theta \in \mathbb{R}^d\}$$

For VI, difficulty is to find $Q_\theta \simeq \mathcal{T}Q_\theta$: **Approximate VI**

For PI, difficulty is to find $Q_\theta \simeq \mathcal{T}_\pi Q_\theta$: **Approximate PI**

Different strategies:

- ▶ Minimize $\|Q_\theta - \mathcal{T}_\pi Q_\theta\|$: **Bellman residual minimization**.
- ▶ Solve $Q_\theta = \Pi_{\mathcal{F}} \mathcal{T}_\pi Q_\theta$: **Least-squares Temporal Differences**.

When \mathcal{S}, \mathcal{A} is huge, we may want to represent in Function spaces: m, p, V, Q or π .

1) Parameterize **Q function**

$$\mathcal{F} = \{Q_\theta(s, a) = \theta^\top \varphi(s, a), \theta \in \mathbb{R}^d\}$$

For VI, difficulty is to find $Q_\theta \simeq \mathcal{T}Q_\theta$: **Approximate VI**

For PI, difficulty is to find $Q_\theta \simeq \mathcal{T}_\pi Q_\theta$: **Approximate PI**

Different strategies:

- ▶ Minimize $\|Q_\theta - \mathcal{T}_\pi Q_\theta\|$: **Bellman residual minimization**.
- ▶ Solve $Q_\theta = \Pi_{\mathcal{F}} \mathcal{T}_\pi Q_\theta$: **Least-squares Temporal Differences**.

2) Parameterize **π function** (see next class)

- ▶ Solve $\operatorname{argmin}_{\pi \in \mathcal{F}} (\max_a Q(\cdot, a) - Q(\cdot, \pi))$: **Direct Policy Search**

- ▶ Difficulty to apply value iteration ($Q_{k+1} = \mathcal{T}Q_k$):
 - ▶ Max operator \mathcal{T} is **unknown**;
 - ▶ $Q_k \in \mathcal{F}$ may **not** imply $\mathcal{T}Q_k \in \mathcal{F}$.

- ▶ Difficulty to apply value iteration ($Q_{k+1} = \mathcal{T}Q_k$):
 - ▶ Max operator \mathcal{T} is **unknown**;
 - ▶ $Q_k \in \mathcal{F}$ may **not** imply $\mathcal{T}Q_k \in \mathcal{F}$.
- ▶ Example of naive **Approximate value iteration**
 - ▶ writing $Q_k = Q_{\theta_k}$, sampled operator:

$$(\hat{\mathcal{T}}Q_k)(s_i, a_i) = r_i + \gamma \max_{a' \in \mathcal{A}} Q_k(s'_i, a')$$

- ▶ search for $Q \in \mathcal{F}$ being the closest to $\hat{\mathcal{T}}Q_k$:

$$Q_{k+1} \in \operatorname{argmin}_{Q_\theta \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \left(Q_\theta(s_i, a_i) - (\hat{\mathcal{T}}Q_k)(s_i, a_i) \right)^2.$$

- ▶ summary: $Q_{k+1} = \Pi_{\mathcal{F}} \hat{\mathcal{T}}Q_k$.

Algorithm 1 Approximate value iteration

Input: A dataset $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)_{1 \leq i \leq n}\}$, the number K of iterations, a function approximator, an initial state-action value function Q_0 .

- 1: **for** $k = 0$ **to** K **do**
- 2: Apply the sampled Bellman operator to function Q_k :

$$(\hat{T}Q_k)(s_i, a_i) = r_i + \gamma \max_{a' \in \mathcal{A}} Q_k(s'_i, a').$$

- 3: Solve the regression problem with inputs (s_i, a_i) and outputs $(\hat{T}Q_k)(s_i, a_i)$ to get the Q-function Q_{k+1}
- 4: **end for**
- 5: **return** The greedy policy $\pi_{K+1} \in \mathcal{G}(Q_{K+1})$:

$$\forall s \in \mathcal{S}, \quad \pi_{K+1} \in \operatorname{argmax}_{a \in \mathcal{A}} Q_{K+1}(s, a).$$

- ▶ Approximate value iteration:

$$\mathbf{Q}_{k+1} = \mathbb{A} \hat{\mathbf{T}} \mathbf{Q}_k.$$

- ▶ \mathbb{A} is an abstract approximation operator

- ▶ Approximate value iteration:

$$\mathbf{Q}_{k+1} = \mathbb{A} \hat{\mathbf{T}} \mathbf{Q}_k.$$

- ▶ \mathbb{A} is an abstract approximation operator
- ▶ Problem: $\mathbb{A} \hat{\mathbf{T}}$ should be a **contraction**
 - ▶ Otherwise divergence can (will) occur;
 - ▶ Not the case for $\Pi \hat{\mathbf{T}} \dots$
 - ▶ Works if “**averagers**” are used for function approximation, such as
 - ▶ ensemble of trees
 - ▶ extremely randomized forests: **fitted-Q**
 - ▶ kernel averagers (Nadaraya-Watson)

Working with AVI is tricky due to **max Bellman operator** \mathcal{T} .

- ▶ Approximate value iteration:

$$\mathbf{Q}_{k+1} = \mathbb{A} \hat{\mathbf{T}} \mathbf{Q}_k.$$

- ▶ \mathbb{A} is an abstract approximation operator
- ▶ Problem: $\mathbb{A} \hat{\mathbf{T}}$ should be a **contraction**
 - ▶ Otherwise divergence can (will) occur;
 - ▶ Not the case for $\Pi \hat{\mathbf{T}} \dots$
 - ▶ Works if “**averagers**” are used for function approximation, such as
 - ▶ ensemble of trees
 - ▶ extremely randomized forests: **fitted-Q**
 - ▶ kernel averagers (Nadaraya-Watson)

Working with AVI is tricky due to **max Bellman operator** \mathcal{T} .

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

Approximate value iteration

Approximate policy iteration

DEEP Q-NETWORK

POLICY GRADIENT METHODS

- ▷ Idea: Instead of considering \mathcal{T} , consider \mathcal{T}_π **for a fixed π .**
- ▶ Policy iteration:
 - ① policy evaluation: solve the fixed-point equation $Q_{\pi_k} = T_{\pi_k} Q_{\pi_k}$;
 - ② policy improvement: compute the greedy policy $\pi_{k+1} = \mathcal{G}(Q_{\pi_k})$.
- ▶ **Approximate** policy iteration:
 - ① approximate policy evaluation: find a function $Q_k \in \mathcal{F}$ such that $Q_k \approx T_{\pi_k} Q_k$;
 - ② policy improvement: compute the greedy policy $\pi_{k+1} = \mathcal{G}(Q_{\pi_k})$.

Algorithm 2 Approximate policy iteration (generic form)

Input: An initial $\pi_0 \in \mathcal{A}^S$ (possibly an initial Q_0 and $\pi_0 \in \mathcal{G}(Q_0)$), number of iterations K

- 1: **for** $k = 0$ **to** K **do**
 - 2: approximate policy evaluation: find $Q_k \in \mathcal{F}$ such that $Q_k \approx T_{\pi_k} Q_k$.
 - 3: policy improvement: $\pi_{k+1} \in \mathcal{G}(Q_k)$.
 - 4: **end for**
 - 5: **return** the policy π_{K+1}
-

- ▷ How to find an **approximate fixed point** of T_π , that is a function $Q_\theta \in \mathcal{F}$ such that $Q_\theta \approx T_\pi Q_\theta$?

MONTE CARLO ROLLOUTS

- ▷ Assume that Q_π is known: simply a **regression** problem. E.g. linear least-squares:

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (Q_\pi(s_i, a_i) - Q_\theta(s_i, a_i))^2.$$

- ▶ Assume that Q_π is known: simply a **regression** problem. E.g. linear least-squares:

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (Q_\pi(s_i, a_i) - Q_\theta(s_i, a_i))^2.$$

- ▶ Q_π is (obviously) unknown...
- ▶ Idea 1: **Monte carlo rollout?**

- ▶ sample a full trajectory starting in s_i where action a_i is chosen first, all subsequent states being sampled according to the system dynamics and actions chosen according to π ;
- ▶ let q_i be the associated discounted cumulative reward;
- ▶ unbiased estimate: $\mathbb{E}[q_i | s_i, a_i] = Q_\pi(s_i, a_i)$

- ▶ Assume that Q_π is known: simply a **regression** problem. E.g. linear least-squares:

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (Q_\pi(s_i, a_i) - Q_\theta(s_i, a_i))^2.$$

- ▶ Q_π is (obviously) unknown...
- ▶ Idea 1: **Monte carlo rollout?**

- ▶ sample a full trajectory starting in s_i where action a_i is chosen first, all subsequent states being sampled according to the system dynamics and actions chosen according to π ;
- ▶ let q_i be the associated discounted cumulative reward;
- ▶ unbiased estimate: $\mathbb{E}[q_i | s_i, a_i] = Q_\pi(s_i, a_i)$
- ▶ replace $Q_\pi(s_i, a_i)$ by the unbiased estimate q_i :

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (q_i - Q_\theta(s_i, a_i))^2.$$

- ▶ Drawbacks: requires a simulator, rollouts can be quite noisy.

- ▶ Idea 2: **minimize the residual** $\|Q_\theta - T_\pi Q_\theta\|$ for some norm.
- ▶ With an ℓ_2 -loss, parametric representation and sampled operator:

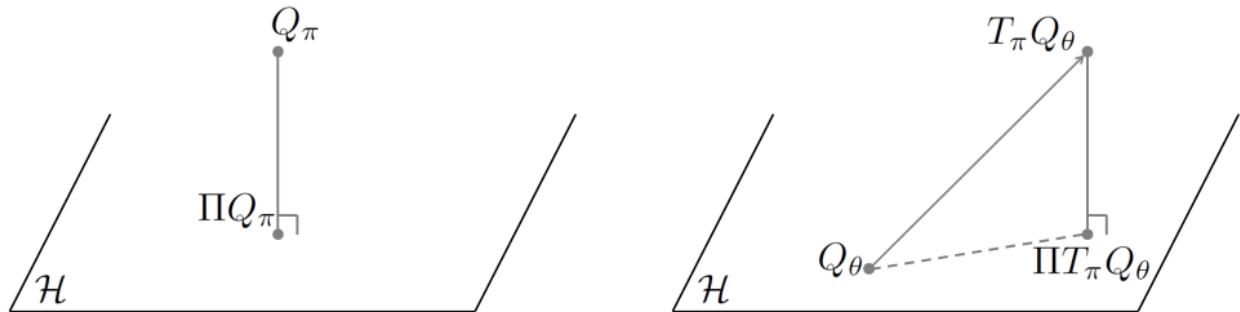
$$\begin{aligned} & \min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \left((\widehat{T}_\pi Q_\theta)(s_i, a_i) - Q_\theta(s_i, a_i) \right)^2 \\ &= \min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \left(r_i + \gamma Q_\theta(s'_i, \pi(s'_i)) - Q_\theta(s_i, a_i) \right)^2. \end{aligned}$$

- ▶ Idea 2: **minimize the residual** $\|Q_\theta - T_\pi Q_\theta\|$ for some norm.
- ▶ With an ℓ_2 -loss, parametric representation and sampled operator:

$$\begin{aligned} & \min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \left((\widehat{T}_\pi Q_\theta)(s_i, a_i) - Q_\theta(s_i, a_i) \right)^2 \\ &= \min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \left(r_i + \gamma Q_\theta(s'_i, \pi(s'_i)) - Q_\theta(s_i, a_i) \right)^2. \end{aligned}$$

- ▷ However, there is a **bias problem**:

$$\begin{aligned} & \mathbb{E}[((\widehat{T}_\pi Q_\theta)(s_i, a_i) - Q_\theta(s_i, a_i))^2 | s_i, a_i] \\ &= ((T_\pi Q_\theta)(s_i, a_i) - Q_\theta(s_i, a_i))^2 + \mathbb{V}((\widehat{T}_\pi Q_\theta)(s_i, a_i) | s_i, a_i) \\ &\neq ((T_\pi Q_\theta)(s_i, a_i) - Q_\theta(s_i, a_i))^2. \end{aligned}$$



► Idea 3: Solve $Q_\theta = \Pi T_\pi Q_\theta$. as a **nested optimization** problem:

$$w_\theta = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (r_i + \gamma Q_\theta(s'_i, \pi(s'_i)) - Q_w(s_i, a_i))^2 \quad (2)$$

$$\theta_n = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (Q_\theta(s_i, a_i) - Q_{w_\theta}(s_i, a_i))^2. \quad (3)$$

- ▶ (2) is ordinary least-squares in w . Solution:

$$w_\theta = \left(\sum_{i=1}^n \varphi(s_i, a_i) \varphi(s_i, a_i)^\top \right)^{-1} \sum_{i=1}^n \varphi(s_i, a_i) (r_i + \gamma \theta^\top \varphi(s'_i, \pi(s'_i))).$$

- (2) is ordinary least-squares in w . Solution:

$$w_\theta = \left(\sum_{i=1}^n \varphi(s_i, a_i) \varphi(s_i, a_i)^\top \right)^{-1} \sum_{i=1}^n \varphi(s_i, a_i) (r_i + \gamma \theta^\top \varphi(s'_i, \pi(s'_i))).$$

- (3) is minimized for $\theta = w_\theta$:

$$\begin{aligned} \theta_n = w_{\theta_n} &\Leftrightarrow \theta_n = \left(\sum_{i=1}^n \varphi(s_i, a_i) \varphi(s_i, a_i)^\top \right)^{-1} \sum_{i=1}^n \varphi(s_i, a_i) (r_i + \gamma \theta_n^\top \varphi(s'_i, \pi(s'_i))) \\ &\Leftrightarrow \theta_n = \left(\sum_{i=1}^n \varphi(s_i, a_i) (\varphi(s_i, a_i) - \gamma \varphi(s'_i, \pi(s'_i)))^\top \right)^{-1} \sum_{i=1}^n \varphi(s_i, a_i) r_i. \end{aligned}$$

- This API strategy with LSTD is called **LSPI (Least-Squares Policy Iteration)**.

- ▷ Representation: $Q_\theta(s, a) = \langle \theta, \varphi(s, a) \rangle$ with known φ .

Algorithm 3 Least-squares policy iteration

Input: An initial $\pi_0 \in \mathcal{A}^{\mathcal{S}}$ (possibly an initial Q_0 and $\pi_0 \in \mathcal{G}(Q_0)$), number of iterations K

1: **for** $k = 0$ **to** K **do**

2: approximate policy evaluation:

$$\theta_k = \left(\sum_{i=1}^n \varphi(s_i, a_i) (\varphi(s_i, a_i) - \gamma \varphi(s'_i, \pi_k(s'_i)))^\top \right)^{-1} \sum_{i=1}^n \varphi(s_i, a_i) r_i.$$

3: policy improvement:

$$\pi_{k+1} \in \mathcal{G}(Q_{\theta_k}).$$

4: **end for**

5: **return** the policy π_{K+1}

GUARANTEES FOR BRM, LSTD, LSPI

Proposition (Williams and Baird, 93)

$$\|V_\pi - \hat{V}_{BR}\|_\infty \leq \frac{1+\gamma}{1-\gamma} \|V_\pi - \hat{V}_{\text{best}}\|_\infty$$

Proposition (Tsitsiklis and Van Roy, 97)

Let ρ be the stationary distribution of P_π , then

$$\|V_\pi - \hat{V}_{LSTD}\|_{2,\rho} \leq \frac{1}{1-\gamma} \|V_\pi - \hat{V}_{\text{best}}\|_{2,\rho}$$

Approximate PI: $\pi_k = \mathcal{G}[v_{k-1}]$ and $v_k = v_{\pi_k} + \varepsilon_k$

Proposition

Assume $\|\varepsilon_k\|_\infty \leq \varepsilon$. The loss due to running π_k instead of π_* satisfies

$$\limsup_{k \rightarrow \infty} \|V_* - v_{\pi_k}\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \varepsilon.$$

Take-home critic strategies

Tabular value function:

- ▷ **Q-learning**: $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_t + \gamma \max_b Q_t(s_{t+1}, b) - Q_t(s_t, a_t))$
- ▷ **SARSA**: $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t))$

Parametric Linear Value function:

- ▷ **LSTD** for V_θ : $\theta_n = \left(\sum_{i=1}^n \varphi(s_i)(\varphi(s_i) - \gamma\varphi(s_{i+1})^\top) \right)^{-1} \sum_{i=1}^n \varphi(s_i)r_i$
- ▷ **LSPI** for Q_θ : $\theta_n = \left(\sum_{i=1}^n \varphi(s_i, a_i)(\varphi(s_i, a_i) - \gamma\varphi(s'_i, \pi(s'_i))^\top) \right)^{-1} \sum_{i=1}^n \varphi(s_i, a_i)r_i$

Parametric Non-Linear Value function: see next slides

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS

- ▷ **Slow updates** : Fast-learning plus Slow-learning networks.
- ▷ **Experience replay** with priority.
- ▷ **Double** Q-learning
- ▷ **Multi-step** Q-learning



Problems to use Neural Nets

- Correlated data (trajectories are made of state transitions conditioned on actions)
- Non stationary strategies (learning control while learning value)
- Extrapolate (bad for SARSA and Fitted- Q)
- Residual methods are more suited but cost function is biased



Deep Q-Network II

Solution [Mnih et al., 2015]

- Use two neural networks:
 - ① A slow-learning target network (θ^-)
 - ② A fast learning Q -network (θ)
- Use experience replay (fill in a replay buffer D with transitions generated by $\pi = f(Q_\theta(s, a))$)
- Shuffle samples in the replay buffer and minimize:

$$J(\theta) = \sum_{(s,a,r,s') \in D} \left[\left(r + \gamma \max_b Q_{\theta^-}(s', b) \right) - Q_\theta(s, a) \right]^2$$

$$\theta \leftarrow \alpha(r + \gamma \max_b Q_{\theta^-}(s', b) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)$$

- Every N training steps $\theta^- \leftarrow \theta$

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Control

Warnings

Deep
 Q -Network

Q-learning

Optimal policies and their values satisfy the **Bellman's optimality equation**:

$$Q^*(s, a) = E[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a]$$

Given estimates of the values Q , we can **sample** this equation and turn it into **incremental updates** to our estimates of Q^* :

$$Q^*(S_t, A_t) \leftarrow Q^*(S_t, A_t) + \alpha \underbrace{(R_t + \gamma \max_{a'} Q^*(S_{t+1}, A_{t+1}) - Q^*(S_t, A_t))}_{G_t}$$

Deep Q-learning

Previously we assumed a separate value estimate Q^* for each state action pairs s, a .

Instead, we can use a **neural network** to learn a **mapping** between state-action pairs (s, a) and their (optimal) values, and exploit **generalization**

The Q-learning updates reduces to **stochastic semi-gradient descent update** on:

$$Loss(\theta) = (R_t + \gamma \max_{a'} Q_\theta^*(S_{t+1}, A_{t+1}) - Q_\theta^*(S_t, A_t))^2$$

 Target: do not back-prop gradients here

Experience Replay

Stochastic gradient descent assumes that the updates we apply are **i.i.d.**

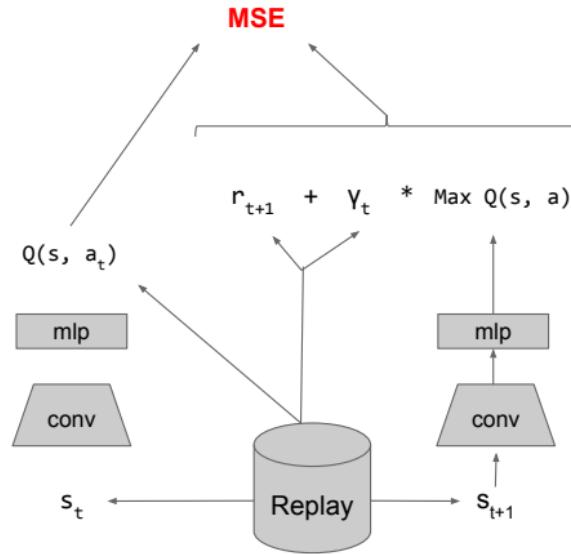
In Reinforcement Learning this is largely false

- e.g. values of states in the same episode of a game are **strongly correlated**.

Compute the Q-learning loss not on the most recent transition in the environment but instead on an old one sampled from a **large memory buffer**

Ref: Mnih et al.

DQN



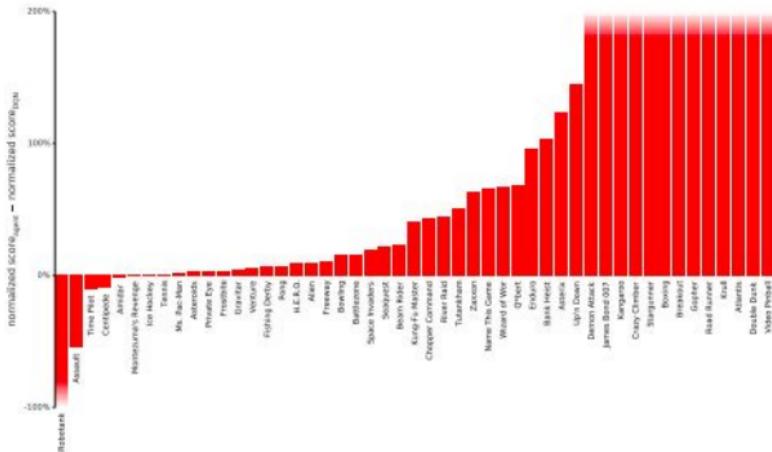
Prioritized Experience Replay

An experience replay is also useful because it allows:

- to use data multiple times to update our network
- to prioritize novel or “surprising” data from which there is much to learn over less interesting data that we already fully understand

As a result it can massively improve **data-efficiency**

Prioritized Replay



$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Proportional
 $p_i = \text{loss}$

Rank-based
 $p_i = \text{rank}_i$

Overestimation Bias

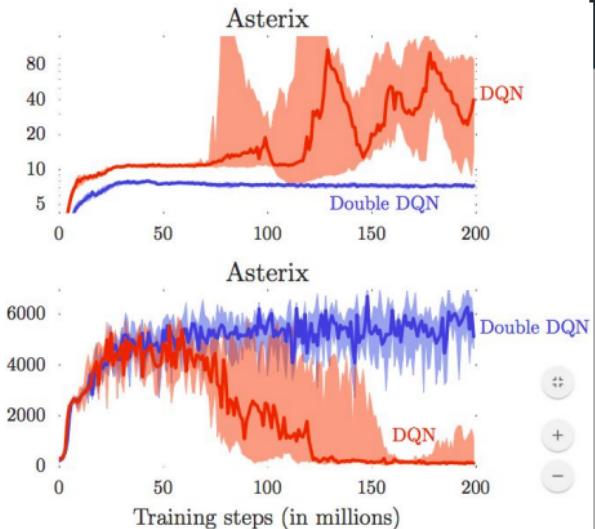
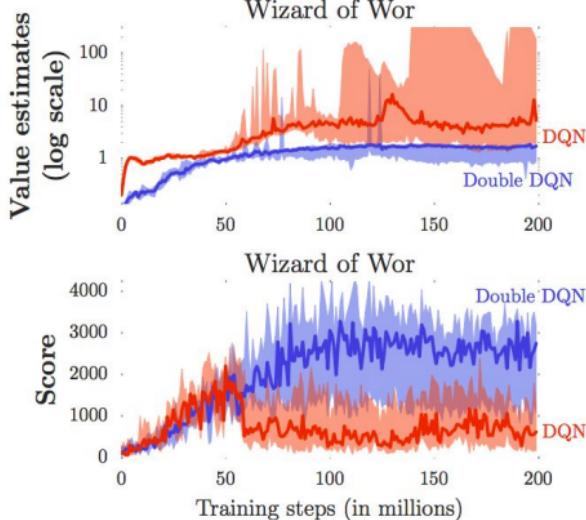
At the heart of the Q-learning operator there is the process of selecting the max value among all actions to construct the target.

$$\text{Loss}(\theta) = (R_t + \gamma \max_{a'} Q_\theta^*(S_{t+1}, A_{t+1}) - Q_\theta^*(S_t, A_t))^2$$

Danger! If the value estimates are noisy I will tend to select the noisiest estimate,
And over time I will end up **overestimating** the action values

$$Q_\theta(s, \operatorname{argmax}_{a'} Q_\eta(s, a')) \quad \longleftarrow \quad \textbf{Double Q-learning}$$

Overestimation Bias





Improvements I

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Control

Warnings

Deep
Q-Network

Double DQN [van Hasselt et al., 2016]

DQN:

$$\theta \leftarrow \alpha(r + \gamma \max_b Q_{\theta^-}(s', b) - Q_{\theta}(s, a)) \nabla_{\theta} Q_{\theta}(s, a)$$

Double DQN

$$\theta \leftarrow \alpha(r + \gamma \max_b Q_{\theta^-}(s', b) - Q_{\theta}(s, a)) \nabla_{\theta} Q_{\theta}(s, a)$$

Decorrelates selection and evaluation and avoid overestimation

<https://www.youtube.com/watch?v=0JYRcogPcfY>



Improvements II

RL

Olivier
Pietquin

Introduction

Policy
Evaluation

Control

Warnings

Deep
Q-Network

Prioritized Experience Replay

- Don't sample uniformly
- Sample with priority to high temporal differences:

$$\|r + \gamma \max_b Q_{\theta^-}(s', b) - Q_{\theta}(s, a)\|$$

Multi-step deep Q-learning

Using our own estimates Q_θ to construct targets $R_t + \gamma \max_a Q_\theta(S_{t+1}, A_{t+1})$ is called **bootstrapping**.

We don't however need to bootstrap necessarily after a single step.

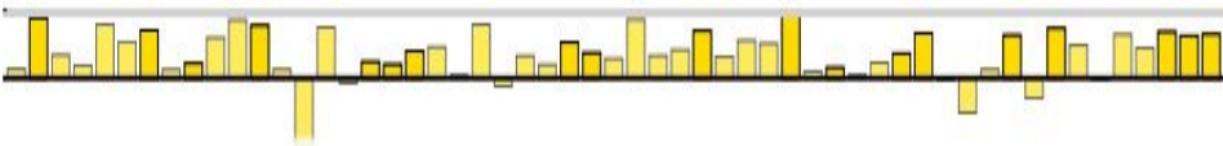
We can construct **multi-step targets**:

$$R_t + \gamma R_{t+1} + \dots + \gamma^{K-1} R_{t+K-1} + \gamma^K \max_a Q_\theta(S_{t+1}, A_{t+1})$$

Multi-step deep Q-learning

Multi-step updates propagate information about observed rewards faster through state space, and by choosing appropriately the bootstrap length we can trade-off **bias** and **variance**

Improvements of K=3 over K=1 in a variant of DQN on a collection of Atari games



The Deadly Triad

1. Off-Policy

- a. Estimate values of a policy on data generated from a different policy

2. Bootstrapping

- a. Using estimates to compute the targets for updating those same estimates

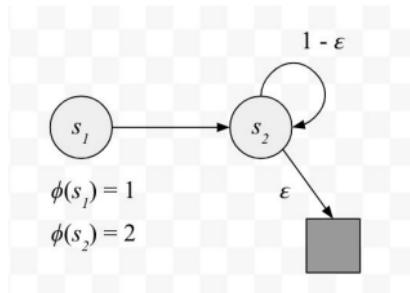
3. Function Approximation

- a. E.g. using neural networks

The Deadly Triad

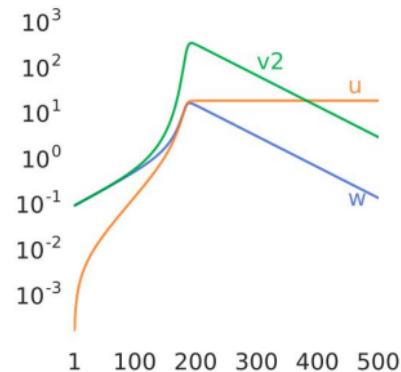
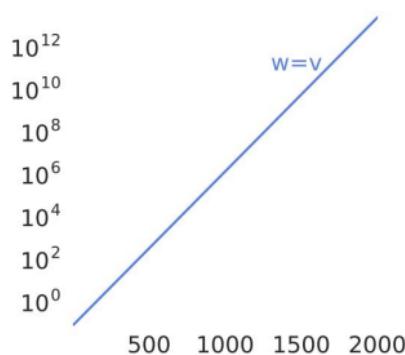
Sutton and Barto refer to the combination of these three features as the **Deadly Triad**, because can result in **divergence**, with the parameters of the network blowing up to infinity

$$(R_t + \gamma v_\theta(S_{t+1}) - v_\theta(S_t))^2$$



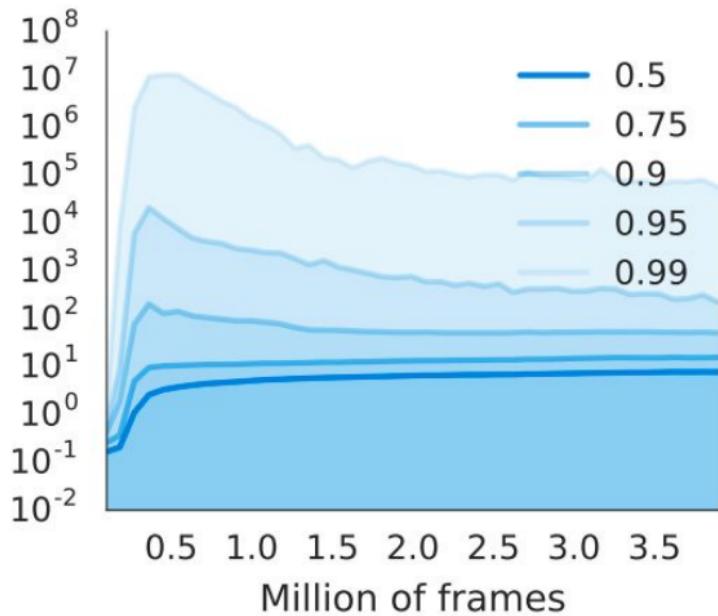
If I update **off-policy**, for instance because I use an experience replay, I may update $v(s_1)$ more often than $v(s_2)$ making w blow to infinity instead of converging to the optimal value of 0.

Deep RL and the Deadly Triad



Soft-divergence

This form of **soft-divergence**, where values grow to unreasonably high values but then converge back to sensible estimates, is more common in practice than actual divergence.



Target Networks

Target Networks are a common remedy to this type of **inappropriate generalization**

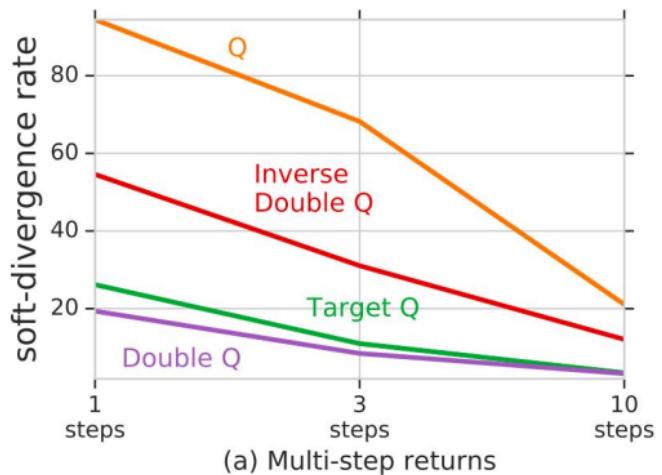
We can keep **two copies** of the neural network used to estimate Q values:

- The parameters of one are updated at each step,
- The parameters of the other are a slow copy of the first one.

This **reduces** the opportunity for **feedback loops**.

Multi-step returns and divergence

Alternatively we can also use Multi-step returns to reduce the reliance of our updates on bootstrapping:



Take-home critic strategies

Tabular value function:

- ▷ **Q-learning**: $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_t + \gamma \max_b Q_t(s_{t+1}, b) - Q_t(s_t, a_t))$
- ▷ **SARSA**: $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t))$

Parametric Linear Value function:

- ▷ **LSTD** for V_θ : $\theta_n = \left(\sum_{i=1}^n \varphi(s_i)(\varphi(s_i) - \gamma \varphi(s_{i+1})^\top) \right)^{-1} \sum_{i=1}^n \varphi(s_i)r_i$
- ▷ **LSPI** for Q_θ : $\theta_n = \left(\sum_{i=1}^n \varphi(s_i, a_i)(\varphi(s_i, a_i) - \gamma \varphi(s'_i, \pi(s'_i))^\top) \right)^{-1} \sum_{i=1}^n \varphi(s_i, a_i)r_i$

Parametric Non-Linear Value function:

- ▷ **DQN**: $\theta \leftarrow \alpha(r + \gamma Q_{\theta-}(s', \text{argmax}_b Q_{\theta-}(s, b)) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)$
- ▷ **DDQN**: $\theta \leftarrow \alpha(r + \gamma Q_{\theta-}(s', \text{argmax}_b Q_\theta(s, b)) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)$
- ▷ Exp.Replay: sample $(r, s, a, s') \propto |r + \gamma \max_b Q_{\theta-}(s', b) - Q_\theta(s, a)|$
- ▷ Multi-step variants

SAIL:

$$\theta \leftarrow \alpha(r + \gamma \max_b Q_{\theta-}(s', b) + \beta \max[G_t, Q_{\theta-}(s, a) - \max_a Q_{\theta-}(s, a)]) \nabla_\theta Q_\theta(s, a)$$

where $G_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS

- ▷ Approximate a **stochastic policy** π directly using $\pi \in \mathcal{F} = \{\pi_\theta : \theta \in \mathbb{R}^d\}$.
- ▷ Consider objective function $J(\theta)$ to optimize in θ :
 - ▷ **Total** return on **Episodic** task with horizon H :

$$J_e(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=1}^H r(s_t, a_t) \right] = V_H^{\pi_\theta}(s_1)$$

- ▷ **Average** value on **Continuing** tasks (infinite horizon):

$$J_c(\theta) = \sum_s f_{\pi_\theta}(s) \sum_a \pi_\theta(s, a) r(s, a),$$

with f_{π_θ} : stationary distribution of Markov chain induced by π_θ (frequency of visits of each state).

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS

Issues with Approximate Policy Iteration

Policy optimization

REINFORCE

PPO



Policy Iteration: recap

Let π_0 be an arbitrary stationary policy

while $k = 1, \dots, K$ **do**

Policy Evaluation: given π_k compute $v_k = v^{\pi_k}$

Policy Improvement: find π_{k+1} that is better than π_k

- e.g., compute the *greedy* policy

$$\pi_{k+1}(s) \in \arg \max_{a \in \mathcal{A}} \left\{ r(s, a) + \gamma \sum_y p(y|s, a) v^{\pi_k}(y) \right\}$$

return the last policy π_K

end

- Convergence is finite and monotonic [Bertsekas, 2007] (in exact settings)
- ?
- Issues: Function approximation for $v^{\pi_k} \implies$ Is it still converging?
Continuous actions?

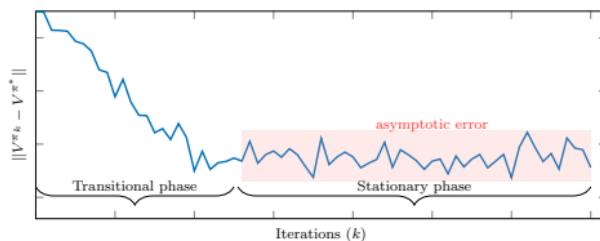
Approximate Policy Iteration

Issue: is no longer guaranteed to converge!

Proposition

The asymptotic performance of the policies π_k generated by the API algorithm is related to the approximation error as:

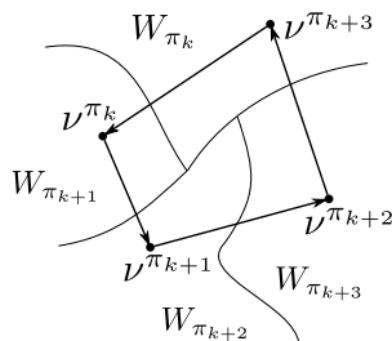
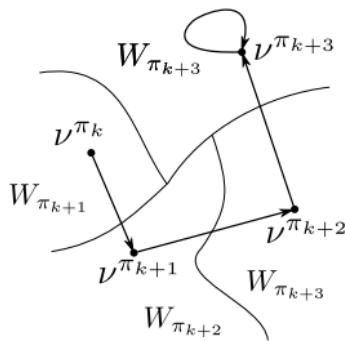
$$\limsup_{k \rightarrow +\infty} \underbrace{\|v^* - v^{\pi_k}\|_\infty}_{\text{performance loss}} \leq \frac{2\gamma}{(1-\gamma)^2} \limsup_{k \rightarrow +\infty} \underbrace{\|v_k - v^{\pi_k}\|_\infty}_{\text{approximation error}}$$

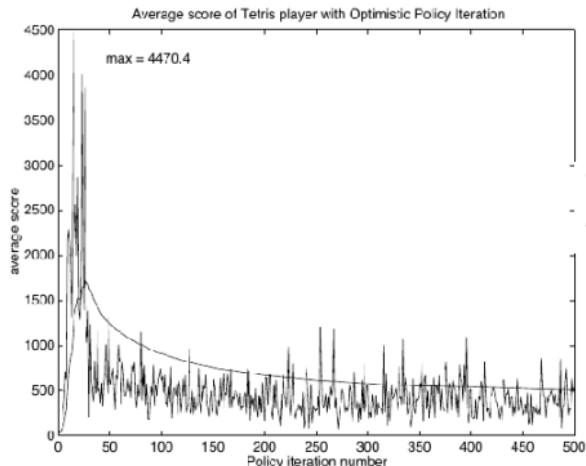


Approximate Policy Iteration: Issues

Potential pathologies in policy-iteration with function approximation

- 1 Exploration
- 2 Policy evaluation: bias, simulation bias/error
- 3 Policy improvement: policy oscillation
 - *local attractors*, e.g., local maxima





Tetris [Bertsekas and Ioffe, 1996]
very pathological [e.g., Scherrer et al., 2015]

Policy oscillation with linear function approximation [Koller and Parr, 2000, Lagoudakis and Parr, 2003a]

 poor policies

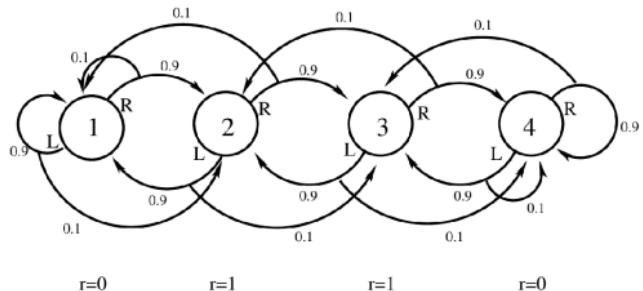


Figure 9: The problematic MDP.

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS

Issues with Approximate Policy Iteration

Policy optimization

REINFORCE

ppo



From Policy Iteration to Policy Search

- Approximate a *stochastic policy* directly using function approximation

$$\pi_\theta : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A}) \text{ with } \theta \in \mathbb{R}^d$$

- Let $J(\pi_\theta)$ denote the *policy performance* of policy π_θ

➤ *Policy optimization problem*

$$\max_{\pi_\theta} J(\pi_\theta)$$

From Policy Iteration to Policy Search

- Approximate a *stochastic policy* directly using function approximation

$$\pi_\theta : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A}) \text{ with } \theta \in \mathbb{R}^d$$

- Let $J(\pi_\theta)$ denote the *policy performance* of policy π_θ

➤ *Policy optimization problem*

$$\max_{\pi_\theta} J(\pi_\theta)$$

Solution 1: Policy Search/Black-box optimization:

Use global optimizers or gradient by finite-difference methods

Policy π_θ can also be *not differentiable* w.r.t. θ

From Policy Iteration to Policy Search

- Approximate a *stochastic policy* directly using function approximation

$$\pi_\theta : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A}) \text{ with } \theta \in \mathbb{R}^d$$

- Let $J(\pi_\theta)$ denote the *policy performance* of policy π_θ

➤ *Policy optimization problem*

$$\max_{\pi_\theta} J(\pi_\theta)$$

Solution 1: Policy Search/Black-box optimization:

Use global optimizers or gradient by finite-difference methods

Policy π_θ can also be *not differentiable* w.r.t. θ

Solution 2: Policy gradient optimization:

Compute the gradient $\nabla_\theta J(\theta)$ and follow the ascent direction

$\nabla_\theta \pi_\theta(s, a)$ should exist

Policy Gradient as Policy Update

Approximate Policy Iteration

$$\pi_{\theta_{k+1}} = \arg \max_{\pi_\theta} q^{\pi_\theta}(s, \pi_\theta(s))$$

Unstable (fast)

Policy Gradient

$$\theta_{k+1} = \theta_k + \alpha_k \nabla J(\theta_k)$$

Smooth, fine control (slow)

How do we compute $\nabla_\theta J(\theta)$?

(recap on optimality criteria)

FINITE HORIZON (EPISODIC TASK)

► We can redefine **Episodic** J_e using **trajectories** (rollouts)

$$\tau = (s_1, a_1, r_1, \dots, s_{H+1})$$

$$J_e(\theta) = \mathbb{E} \left[\sum_{t=1}^H r_t \right] = \mathbb{E}_\tau \left[R(\tau) \right] = \int p_{\pi_\theta}(\tau) R(\tau) d\tau$$

where $p_{\pi_\theta}(\tau)$: **probability** of observing trajectory τ ,

$$p_{\pi_\theta}(\tau) = p(s_1) \prod_{t=1}^H p(s_{t+1}|s_t, a_t) \pi_\theta(s_t, a_t) \quad \text{by Markov}$$

$R(\tau)$: **total reward** accumulated on trajectory τ .

$$R(\tau) = \sum_{t=1}^H r_t$$

- ▶ Key observation: $\nabla_{\theta} J_e(\theta) = \mathbb{E} \left[\nabla_{\theta} \log p_{\pi_{\theta}}(\tau) R(\tau) \right]$. Then
 - ▶ Gradient is **Independent** on dynamics (!):

$$\nabla_{\theta} \log p_{\pi_{\theta}}(\tau) = \sum_{t=1}^H \nabla_{\theta} \log \pi_{\theta}(s_t, a_t)$$

- ▶ Unbiased gradient estimator: Sample τ and compute $\nabla_{\theta} \log p_{\pi_{\theta}}(\tau) R(\tau)$

- ▷ Key observation: $\nabla_{\theta} J_e(\theta) = \mathbb{E} \left[\nabla_{\theta} \log p_{\pi_{\theta}}(\tau) R(\tau) \right]$. Then
- ▶ Gradient is **Independent** on dynamics (!):

$$\nabla_{\theta} \log p_{\pi_{\theta}}(\tau) = \sum_{t=1}^H \nabla_{\theta} \log \pi_{\theta}(s_t, a_t)$$

- ▶ Unbiased gradient estimator: Sample τ and compute $\nabla_{\theta} \log p_{\pi_{\theta}}(\tau) R(\tau)$
- ▷ In particular

Theorem (Williams, 1992; Sutton et al. 2000)

$$\nabla_{\theta} J_e(\theta) = \mathbb{E} \left[\sum_{t=1}^H \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) \sum_{t=1}^H r_t \right]$$

- ▷ Choose π such that $\nabla_{\theta} \log \pi_{\theta}$ is easy to compute.

- The objective is an expectation. We compute the gradient w.r.t. θ

$$\begin{aligned}\nabla_{\theta} J_e(\theta) &= \int \nabla_{\theta} p_{\pi_{\theta}}(\tau) R(\tau) d\tau \\&= \int p_{\pi_{\theta}}(\tau) \frac{\nabla_{\theta} p_{\pi_{\theta}}(\tau)}{p_{\pi_{\theta}}(\tau)} R(\tau) d\tau \\&= \mathbb{E} \left[\frac{\nabla_{\theta} p_{\pi_{\theta}}(\tau)}{p_{\pi_{\theta}}(\tau)} R(\tau) \right] \\&= \mathbb{E} \left[\nabla_{\theta} \log p_{\pi_{\theta}}(\tau) R(\tau) \right]\end{aligned}$$

How do we **represent** a policy?

- ▷ **Continuous** \mathcal{A} : **Normal** policy, $\theta = (\alpha, \beta)$

$$\pi_\theta(a|s) = \frac{1}{\sigma_\beta(s)\sqrt{2\pi}} e^{-\frac{(a-\mu_\alpha(s))^2}{2\sigma_\beta^2(s)}} \begin{cases} \nabla_\alpha \log \pi_\theta(a|s) &= \frac{(a-\mu_\alpha(s))}{\sigma_\beta^2(s)} \nabla_\alpha \mu_\alpha(s) \\ \nabla_\beta \log \pi_\theta(a|s) &= \frac{(a-\mu_\alpha(s))^2 - \sigma_\beta^2(s)}{\sigma_\beta^3(s)} \nabla_\beta \sigma_\beta(s) \end{cases}$$

- ▷ **Discrete** \mathcal{A} : **Gibbs** (softmax) policy

$$\pi_\theta(a|s) = \frac{e^{-\kappa Q_\theta(s,a)}}{\sum_{a' \in \mathcal{A}} e^{-\kappa Q_\theta(s,a')}}$$

$$\nabla_\alpha \log \pi_\theta(a|s) = \kappa \nabla_\theta Q_\theta(s, a) - \kappa \sum_{a' \in \mathcal{A}} \pi(a'|s) \nabla_\theta Q_\theta(s, a')$$

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS

Issues with Approximate Policy Iteration

Policy optimization

REINFORCE

PPO



REINFORCE

- 1 Let π_{θ_1} be an arbitrary policy
- 2 At each iteration $k = 1, \dots, K$
 - Sample m trajectory $\tau_i = (s_1, a_1, r_1, s_2, \dots, s_T, a_T, r_T, s_{T+1})$ following π_k
 - Compute unbiased gradient estimate

$$\widehat{\nabla_{\theta} J}(\pi_{\theta_k}) = \frac{1}{m} \sum_{i=1}^m \left(\sum_{t=1}^H r_t^i \right) \left(\sum_{t=1}^H \nabla_{\theta} \log \pi_{\theta_k}(s_t, a_t) \right)$$

- Update parameters

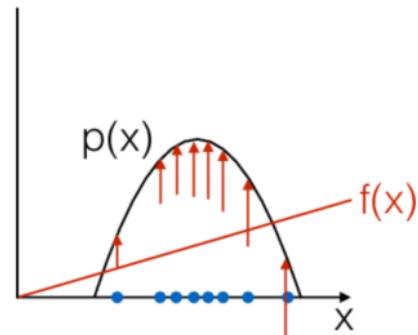
$$\theta_{k+1} = \theta_k + \alpha_k \widehat{\nabla_{\theta} J}(\pi_{\theta_k})$$

- 3 Return last policy π_{θ_K}

REINFORCE: Intuition

$$\hat{g}_i = R(\tau_i) \nabla_{\theta} \log \mathbb{P}(\tau_i | \pi_{\theta}, M)$$

- $R(\tau_i)$ measures how *good* is sample τ_i
- Moving in the direction of \hat{g}_i pushes up the log probability of the sample, in proportion to how good it is



[Schulman, 2016]

Interpretation: uses good trajectories as supervised examples

- *Like maximum likelihood* in supervised learning
- good stuff are made more likely while bad less (TO REMOVE)
- Trial and Error approach

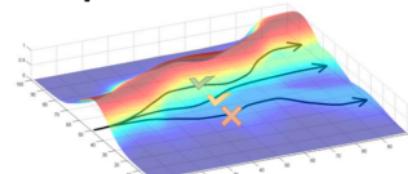


image from "CS 294-112: Deep Reinforcement Learning" slides by S. Levine

Levine

REINFORCE

Pros

- Easy to compute
- *Does not use Markov property!*
- Can be used in partially observable MDPs without modification

Issues

- Use an MC estimate of $q(s, a)$
- It has possibly a *very large variance*
- Needs many samples to converge

Policy Gradient: temporal structure

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E} \left[\sum_{t=1}^H \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) \sum_{t'=t}^H r_{t'} \right]$$

$$\begin{aligned} \mathbb{E}_{a \sim \pi_{\theta}} \left[\nabla_{\theta} \log \pi_{\theta}(s_t, a) \sum_{t'=1}^{t-1} r_i \middle| \tau_{1:t-1} \right] &= \left(\sum_{t'=1}^{t-1} r_i \right) \int \pi_{\theta}(s_t, a) \nabla_{\theta} \log \pi(s_t, a) da \\ &= \left(\sum_{t'=1}^{t-1} r_i \right) \int \nabla_{\theta} \pi(s_t, a) da \\ &= \left(\sum_{t'=1}^{t-1} r_i \right) \nabla_{\theta} \underbrace{\int \pi(s_t, a) da}_{:=1} = 0 \end{aligned}$$

in literature known as **G(PO)MDP** [Peters and Schaal, 2008b]



REINFORCE with PG theorem Algorithm II

Algorithm 1 REINFORCE with PG theorem Algorithm

Initialize θ^0 as random, Initialize step-size α_0

$n = 0$

while no convergence **do**

 Generate rollout $h_n = \{s_1^n, a_1^n, r_1^n, \dots, s_H^n, a_H^n, r_H^n\} \sim \pi_{\theta^n}$

$PG_\theta = 0$

for $t = 1$ to H **do**

$R_t = \sum_{t'=t}^H r_{t'}^n$

$PG_\theta += \nabla_\theta \log \pi_{\theta^n}(s_t, a_t) R_t$

end for

$n++$

$\theta^n \leftarrow \theta^{n-1} + \alpha_n PG_\theta$

 update α_n (if step-size scheduling)

end while

return θ^n

RL

Olivier
Pietquin

Introduction

Policy
Gradient

REINFORCE

Policy Gradient
Theorem

PG with baseline

Actor-Critic

INFINITE HORIZON (CONTINUING TASK)

- ▷ The γ -discounted average reward in infinite horizon is

$$J_c(\theta) = \sum_s d_{\pi_\theta}(s) \sum_a \pi_\theta(s, a) r(s, a),$$

with γ -discounted **visitation frequency**

$$d_{\pi_\theta}(s) = \lim_{T \rightarrow \infty} \sum_{t=1}^T \gamma^{t-1} \mathbb{P}(s_t = s | \pi)$$

Theorem (Policy gradient, infinite horizon)

$$\nabla_\theta J_c(\theta) = \sum_{s,a} d_{\pi_\theta}(s) \nabla_\theta \log \pi_\theta(s, a) q^{\pi_\theta}(s, a),$$

with γ -discounted Q function $q^\pi(s, a) = \lim_{T \rightarrow \infty} \mathbb{E} \left[\sum_{t=1}^T \gamma^{t-1} r(s_t, a_t) \right]$

Going beyond the finite-horizon case

Theorem

For an infinite horizon MDP (average or discounted), the policy gradient is

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{s \sim d^{\pi}} \mathbb{E}_{a \sim \pi_{\theta}(s, \cdot)} [\nabla_{\theta} \log \pi_{\theta}(s, a) q^{\pi}(s, a)]$$

- d^{π} is the stationary distribution
- q^{π} is the state-action value function

REINFORCE for infinite horizon

- 1 Collect m trajectories for policy π starting from $s_1 \sim \rho$
- 2 For each time t

$$\hat{q}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$$

(almost) unbiased estimate $\rightarrow \mathbb{E}[\hat{q}|s_t, a_t] = q^\pi(s_t, a_t)$

Then

$$\overline{\nabla_\theta J}(\pi_\theta) := \frac{1}{m} \sum_{i=1}^m \sum_{t=1}^T \gamma^{t-1} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t}) \sum_{t'=t}^T \gamma^{t'-t} r_{i,t'}$$

REINFORCE for infinite horizon

- Define $F_t := \hat{q}_t \nabla_{\theta} \log \pi_{\theta}(s_t, a_t)$

$$\begin{aligned}
 \mathbb{E} \left[\sum_{t=1}^{+\infty} \gamma^{t-1} F_t \right] &= \sum_{t=1}^{+\infty} \gamma^{t-1} \sum_s \mathbb{E}[F_t | s_t = s] \mathbb{P}(s_t = s | s_1 \sim \rho) \\
 &= \sum_{s,a} q^{\pi}(s,a) \nabla_{\theta} \pi(s,a) \underbrace{\sum_{t=1}^{+\infty} \gamma^{t-1} \mathbb{P}(s_t = s | s_1 \sim \rho)}_{:= d^{\pi}(s)} \\
 &= \nabla_{\theta} J(\pi)
 \end{aligned}$$

- Almost unbiased* (T vs. $+\infty$)
- We can introduce a *baseline* $b(s_t)$ also in this case

Policy Gradient via Automatic Differentiation

$$\overline{\nabla_{\theta} J}(\pi_{\theta}) := \frac{1}{m} \sum_{i=1}^m \sum_{t=1}^T \gamma^{t-1} \nabla_{\theta} \log \pi_{\theta}(s_{i,t}, a_{i,t}) \cdot \hat{q}_{i,t}$$

- Manually code the derivative can be tedious
 \implies use auto diff
- Define a graph such that its gradient is the policy gradient

“Pseudo loss”: weighted maximum likelihood

$$\tilde{J} = \frac{1}{m} \sum_{i=1}^m \sum_{t=1}^T \log \pi_{\theta}(s_{i,t}, a_{i,t}) \hat{q}_{i,t}$$

Gradient in Practice

Finite-Horizon γ -discounted setting

$$J_\gamma(\pi) = \mathbb{E} \left[\sum_{t=1}^H \gamma^{t-1} r_t \right]$$

$$\nabla_\theta J_\gamma(\pi) = \mathbb{E} \left[\sum_{t=1}^H \gamma^{t-1} \nabla_\theta \log \pi_\theta(s_t, a_t) q^\pi(s_t, a_t) \right]$$

Gradient in Practice

Finite-Horizon γ -discounted setting

$$J_\gamma(\pi) = \mathbb{E} \left[\sum_{t=1}^H \gamma^{t-1} r_t \right]$$

$$\nabla_\theta J_\gamma(\pi) = \mathbb{E} \left[\sum_{t=1}^H \gamma^{t-1} \nabla_\theta \log \pi_\theta(s_t, a_t) q^\pi(s_t, a_t) \right]$$

In practice

$$\nabla_\theta J^?(\pi) = \mathbb{E} \left[\sum_{t=1}^H \cancel{\gamma^{t-1}}^1 \nabla_\theta \log \pi_\theta(s_t, a_t) q^\pi(s_t, a_t) \right]$$

• $\nabla_\theta J^?(\pi)$ is a **semi-gradient** of the *undiscounted* objective $J(\pi)$

Gradient in practice

$$J(\pi) = \mathbb{E} \left[\sum_{t=1}^H r_t \right] \quad \mapsto \quad \nabla_\theta J(\pi) = \underbrace{\sum_s d_\gamma^\pi(s) \frac{\partial}{\partial \theta} v_\gamma^\pi(s)}_{:= \nabla_\theta J^?(\pi)} + \sum_s v_\gamma^\pi(s) \frac{\partial}{\partial \theta} d_\gamma^\pi(s)$$

- ! TD(0) step is also a semi-gradient of the mean squared Bellman error [Sutton and Barto, 2018, Chapter 9]
 - In *tabular settings*, semi-gradient TD(0) converges to a minimum of the mean squared error [Jaakkola et al., 1994]
 - Also *on-policy* TD with linear function approximation [Sutton and Barto, 2018]

Gradient in practice

$$J(\pi) = \mathbb{E} \left[\sum_{t=1}^H r_t \right] \quad \mapsto \quad \nabla_{\theta} J(\pi) = \sum_s d_{\gamma}^{\pi}(s) \frac{\partial}{\partial \theta} v_{\gamma}^{\pi}(s) + \sum_s v_{\gamma}^{\pi}(s) \frac{\partial}{\partial \theta} d_{\gamma}^{\pi}(s)$$

$\underbrace{\phantom{\sum_s d_{\gamma}^{\pi}(s) \frac{\partial}{\partial \theta} v_{\gamma}^{\pi}(s) + \sum_s v_{\gamma}^{\pi}(s) \frac{\partial}{\partial \theta} d_{\gamma}^{\pi}(s)}}_{:= \nabla_{\theta} J^?(\pi)}$

- ! TD(0) step is also a semi-gradient of the mean squared Bellman error [Sutton and Barto, 2018, Chapter 9]
 - In *tabular settings*, semi-gradient TD(0) converges to a minimum of the mean squared error [Jaakkola et al., 1994]
 - Also *on-policy* TD with linear function approximation [Sutton and Barto, 2018]

- ☛ Semi-policy gradient may converge to a **BAD policy** w.r.t. both discounted and undiscounted objectives
- Impossibility result [Nota and Thomas, 2019]:*

$$\nexists f(\pi) \in C \text{ such that } \nabla_{\theta} J^?(\pi) = \frac{\partial}{\partial \theta} f(\pi)$$

(Example?)

TABLE OF CONTENTS

MODEL-FREE LEARNING

LARGE STATE-ACTION SPACE

LINEAR REPRESENTATION

DEEP Q-NETWORK

POLICY GRADIENT METHODS

Issues with Approximate Policy Iteration

Policy optimization

REINFORCE

PPO



- ▷ Gradient descent usually considers **changes** measured in **Euclidean** distance.
- ▷ Better to use a **KL-based** metric instead of Euclidean
- ▷ TRPO computes a **constrained** objective.
- ▷ PPO computes a **regularized** objective.

Gradient Descent

Steepest descent direction of a function $h(\theta) \rightarrow -\nabla h(\theta)$

- It yields the *most reduction* in h per unit of change in θ
- Change is measured using the standard *Euclidean norm* $\|\cdot\|$

$$\frac{-\nabla h}{\|\nabla h\|} = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{d: \|d\| \leq \epsilon} \{h(\theta + d)\}$$

Gradient Descent

Steepest descent direction of a function $h(\theta) \rightarrow -\nabla h(\theta)$

- It yields the *most reduction* in h per unit of change in θ
- Change is measured using the standard *Euclidean norm* $\|\cdot\|$

$$\frac{-\nabla h}{\|\nabla h\|} = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{d: \|d\| \leq \epsilon} \{h(\theta + d)\}$$

Is the Euclidean norm the best metric?

Can we use an alternative definition of (*local*) distance?

Gradient Descent

Steepest descent direction of a function $h(\theta) \rightarrow -\nabla h(\theta)$

- It yields the *most reduction* in h per unit of change in θ
- Change is measured using the standard *Euclidean norm* $\|\cdot\|$

$$\frac{-\nabla h}{\|\nabla h\|} = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{d: \|d\| \leq \epsilon} \{h(\theta + d)\}$$

Is the Euclidean norm the best metric?

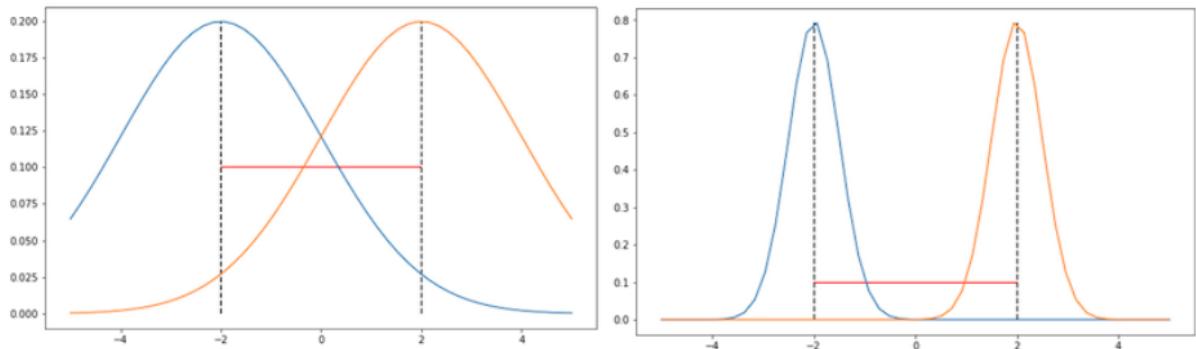
Can we use an alternative definition of (*local*) distance?

⇒ as suggested by [Amari, 1998] it is better to *define a metric* based not on the choice of the coordinates but rather *on the manifold these coordinates parametrize!*

(Example: gradient descent is not affine invariant)

Example

Consider a Gaussian parameterized by only its mean and keep the variance fixed to 2 and 0.5 for the first and second image respectively



The distance of those Gaussians are the same, i.e. 4, according to Euclidean metric (red line)

<https://wiseodd.github.io/techblog/2018/03/14/natural-gradient/>

Fisher Information Matrix

$$F = \mathbb{E}_{x \sim p(\cdot|\theta)} \left[\nabla \log p(x|\theta) \nabla \log p(x|\theta)^T \right]$$

Property 1: Fisher Information Matrix *is the Hessian of KL-divergence* between two distributions $p(x|\theta)$ and $p(x|\theta')$, with respect to θ' , evaluated at $\theta = \theta'$

$$H_{D_{KL}}(p(x|\theta) \| p(x|\theta')) = F$$

Fisher Information Matrix

$$F = \mathbb{E}_{x \sim p(\cdot|\theta)} \left[\nabla \log p(x|\theta) \nabla \log p(x|\theta)^T \right]$$

Property 1: Fisher Information Matrix *is the Hessian of KL-divergence* between two distributions $p(x|\theta)$ and $p(x|\theta')$, with respect to θ' , evaluated at $\theta = \theta'$

$$H_{D_{KL}}(p(x|\theta) \| p(x|\theta')) = F$$

Property 2: Second-order Taylor series expansion

$$D_{KL}(p(x|\theta) \| p(x|\theta + d)) = d^T F d + O(d^3)$$

(proofs)

Natural Gradient in ML

[Martens, 2014]

For a positive definite matrix A , we have [Ollivier et al., 2017] (def. $\|x\|_B = \sqrt{x^\top B x}$)

$$\frac{-A^{-1}\nabla h}{\|\nabla h\|_{A^{-1}}} = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{d: \|d\|_{A^{-1}} \leq \epsilon} \{h(\theta + d)\}$$

Natural Gradient in ML

[Martens, 2014]

For a positive definite matrix A , we have [Ollivier et al., 2017] (def. $\|x\|_B = \sqrt{x^\top B x}$)

$$\frac{-A^{-1}\nabla h}{\|\nabla h\|_{A^{-1}}} = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{d: \|d\|_{A^{-1}} \leq \epsilon} \{h(\theta + d)\}$$

$$A = \frac{1}{2}F \implies -\sqrt{2} \frac{\tilde{\nabla} h}{\|\nabla h\|_{F^{-1}}} = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{d: D_{KL}(p(x|\theta) \| p(x|\theta+d)) \leq \epsilon^2} \{h(\theta + d)\}$$

Negative natural gradient

- steepest descent direction *in the space of distributions*
- where distance is (*approximately*) measured in local neighborhoods by the KL divergence

Natural Gradient in ML

[Martens, 2014]

For a positive definite matrix A , we have [Ollivier et al., 2017] (def. $\|x\|_B = \sqrt{x^\top B x}$)

$$\frac{-A^{-1}\nabla h}{\|\nabla h\|_{A^{-1}}} = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{d: \|d\|_{A^{-1}} \leq \epsilon} \{h(\theta + d)\}$$

$$A = \frac{1}{2}F \implies -\sqrt{2} \frac{\tilde{\nabla} h}{\|\nabla h\|_{F^{-1}}} = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{d: D_{KL}(p(x|\theta) \| p(x|\theta+d)) \leq \epsilon^2} \{h(\theta + d)\}$$

Negative natural gradient

- steepest descent direction *in the space of distributions*
- where distance is (*approximately*) measured in local neighborhoods by the KL divergence
- ! $D_{KL}(p(x|\theta) \| p(x|\theta + d))$ is locally/asymptotically *symmetric* as $d \rightarrow 0$,
and so will be (approximately) symmetric in a local neighborhood [Martens, 2014]
- ! $\tilde{\nabla} h$ is be *invariant* to the choice of parameterization

Natural Policy Gradient

Trust-region objective

$$\begin{aligned}\pi_{k+1} = \arg \max_{\pi'} L_{\pi_k}(\pi') \\ \text{s.t. } \overline{D}_{KL}(\pi' \| \pi_k) \leq \delta\end{aligned}$$

Approximate objective and KL

$$\begin{aligned}L_{\theta_k}(\theta) &\approx L_{\theta_k}(\theta_k) + g^T(\theta - \theta_k) \\ \overline{D}_{KL}(\theta \| \theta_k) &\approx \frac{1}{2}(\theta - \theta_k)^T F(\theta - \theta_k)\end{aligned}$$

\implies

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T F^{-1} g}} \underbrace{F^{-1} g}_{:= \tilde{\nabla} J}$$

Algorithms [Kakade, 2002, Peters and Schaal, 2008a]

Truncated Natural Policy Gradient

Issues:

- $\theta \in \mathbb{R}^d$, d can be very large (e.g., thousands or millions)
- H or F have dimension d^2
- matrix inversion is $\mathcal{O}(d^3)$

Truncated Natural Policy Gradient

Issues:

- $\theta \in \mathbb{R}^d$, d can be very large (e.g., thousands or millions)
- H or F have dimension d^2
- matrix inversion is $\mathcal{O}(d^3)$

Solution:

- Use conjugate gradient to compute $F^{-1}g$ without inverting F [Pascanu and Bengio, 2013]
- With j iterations, CG solves systems of equations $Hx = g$ for x by finding projection onto Krylov subspace (i.e., $\text{span}(g, Hg, \dots H^{j-1}g)$)

⇒ *Truncated Natural Policy Gradient*

Truncated Natural Policy Gradient

Issues:

- $\theta \in \mathbb{R}^d$, d can be very large (e.g., thousands or millions)
- H or F have dimension d^2
- matrix inversion is $\mathcal{O}(d^3)$

Solution:

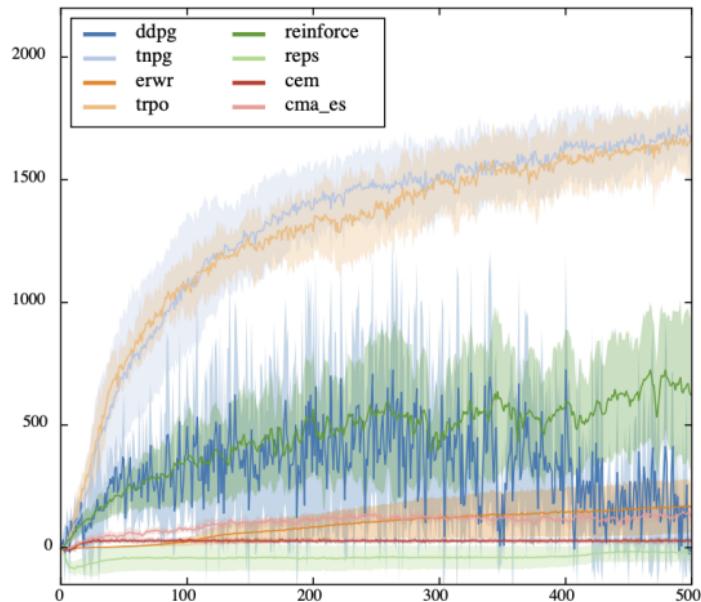
- Use conjugate gradient to compute $F^{-1}g$ without inverting F [Pascanu and Bengio, 2013]
- With j iterations, CG solves systems of equations $Hx = g$ for x by finding projection onto Krylov subspace (i.e., $\text{span}(g, Hg, \dots H^{j-1}g)$)

⇒ *Truncated Natural Policy Gradient*

Other solutions are possible: see ACKTR [Wu et al., 2017], [Ollivier, 2017]

Example: Walker-2d

[Duan et al., 2016]



Discussion

- Natural gradient contains second order informations
- Newton method?

Discussion

- Natural gradient contains second order informations
- Newton method?

The Hessian [Furmston and Barber, 2012, Shen et al., 2019]

$$\nabla^2 J(\theta) = \mathbb{E}_\tau \left[\nabla g(\theta, \tau) \nabla \log \mathbb{P}(\tau | \theta)^T + \nabla^2 g(\theta, \tau) \right]$$

with

$$g(\theta, \tau) = \sum_{h=1}^H \sum_{i=h}^H \gamma^i r(s_i, a_i) \log \pi_\theta(s_h, a_h)$$

Discussion

- [Furmston and Barber, 2012] noticed a connection between $\mathbb{E}[\nabla^2 g(\theta, \tau)]$ and the FIM!
- This hessian can be estimated using first-order information (leading to *quasi Newton approaches*) or *finite difference*
 - see [Shen et al., 2019] also for sample complexity

REINFORCE find an ϵ -approximate first-order stationary point in $O(1/\epsilon^4)$
Hessian aided policy gradient method [Shen et al., 2019] sample complexity of
 $O(1/\epsilon^3)$

Proximal Policy Optimization

[Schulman et al., 2017b]

- Avoid to compute the natural gradient
- Approximate the KL constraint

Proximal Policy Optimization

[Schulman et al., 2017b]

- Avoid to compute the natural gradient
- Approximate the KL constraint

1 Adaptive KL Penalty

- Consider regularized optimization problem

$$\theta_{k+1} = \arg \max_{\theta} L_{\theta_k}(\theta) - \lambda_k \mathbb{E}[D_{KL}(\theta \| \theta_k)]$$

- Adapt λ_k to enforce KL constraint

$$\lambda_{k+1} = \begin{cases} 2\lambda_k & \text{if } \mathbb{E}[D_{KL}(\theta \| \theta_k)] \geq 1.5\delta \\ \lambda_k/2 & \text{if } \mathbb{E}[D_{KL}(\theta \| \theta_k)] \leq \delta/1.5 \\ \lambda_k & \text{otherwise} \end{cases}$$

Proximal Policy Optimization

[Schulman et al., 2017b]

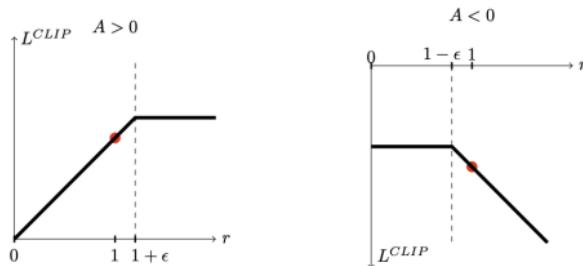
2 Clipped Objective

- Recall surrogate objective

$$L_{\pi}^{\text{IS}}(\pi') = \mathbb{E}_{s \sim d^{\pi}} \mathbb{E}_{a \sim \pi} \left[\frac{\pi'(s, a)}{\pi(s, a)} A^{\pi}(s, a) \right] = \mathbb{E}_{s \sim d^{\pi}} \mathbb{E}_{a \sim \pi} [r_{sa}(\pi') A^{\pi}(s, a)]$$

- Form a lower bound via clipped importance ratios

$$L_{\pi}^{\text{CLIP}}(\pi') = \mathbb{E}_{s \sim d^{\pi}} \mathbb{E}_{a \sim \pi} [\min \{r_{sa}(\pi') A^{\pi}(s, a), \text{clip}(r_{sa}(\pi'), 1 - \epsilon, 1 + \epsilon) A^{\pi}(s, a)\}]$$



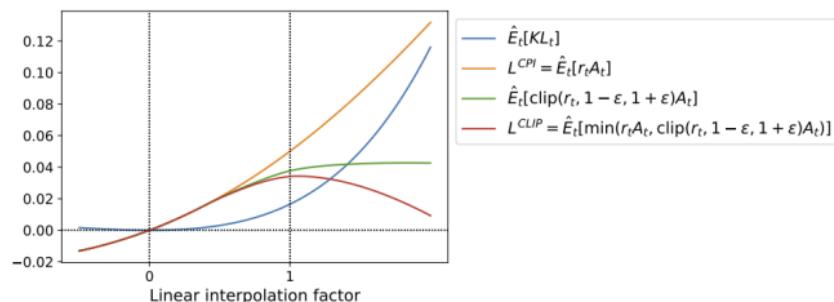
- $\pi_{k+1} = \arg \max_{\pi} L_{\pi_k}^{\text{CLIP}}(\pi)$

Proximal Policy Optimization

[Schulman et al., 2017b]

- Clipping prevents policy from moving too much away from θ_k
- Seems to work as well as PPO with KL penalty
- Much simpler to implement

How does it work?



Various objectives as a function of function of α between θ_k and θ_{k+1}

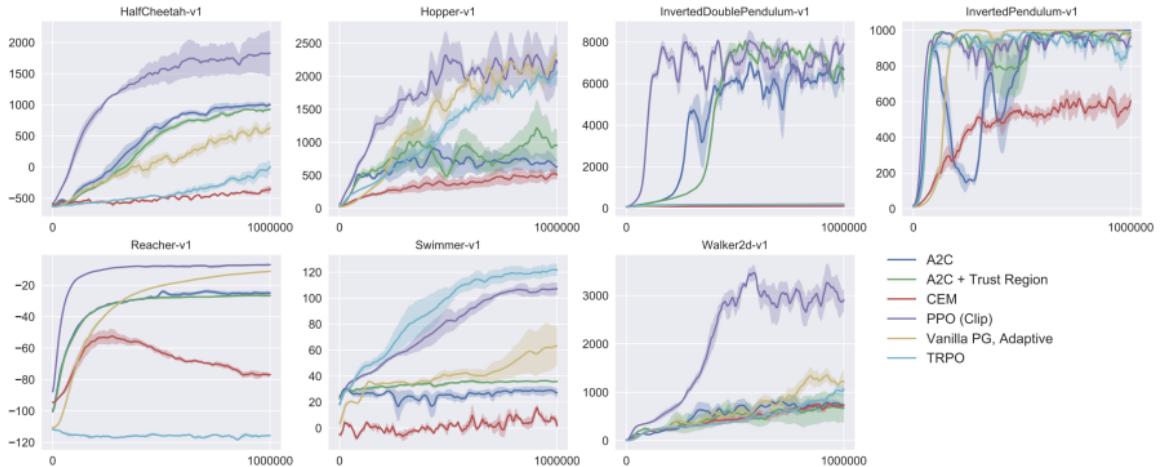


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

▷ Trust Region Policy Optimization

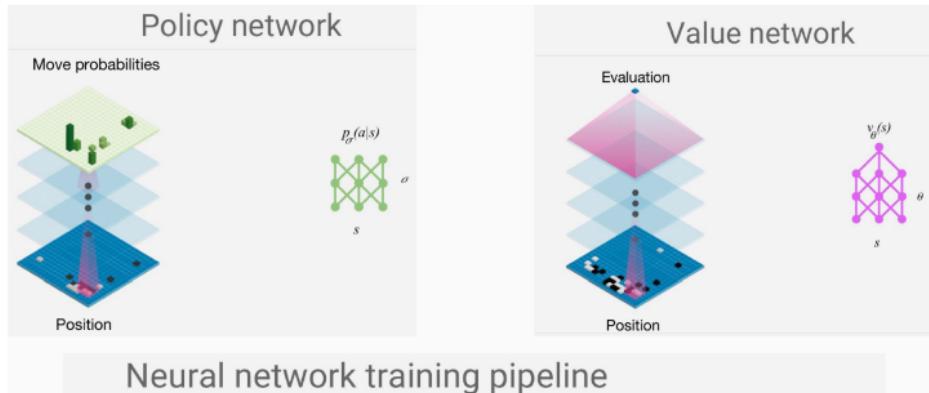
$$\pi_{k+1} = \operatorname{argmax}_{\pi'} L_{\pi_k}(\pi') \text{ s.t. } \mathbb{E}_s[D_{\text{KL}}(\pi', \pi_k)] \leq \delta$$

▷ Proximal Policy Optimization

$$\theta_{k+1} = \operatorname{argmax}_{\theta'} L_{\theta_k}(\theta') - \lambda_k \mathbb{E}_s[D_{\text{KL}}(\theta, \theta_k)]$$

TYPES OF RL ALGORITHMS

- ▷ **Critic**: Value function oriented (SARSA, Q-learning, TD, etc.)
- ▷ **Actor**: Policy oriented (Policy gradient, Reinforce, PPO, TRPO)
- ▷ **Actor-critic**: use both (e.g. A3C, SAC). e.g. AlphaGo



Take-home actor strategies

Parameterize policy π_θ :

- ▷ **Policy gradient:** $\nabla_\theta J(\theta) = \mathbb{E}_\tau[R(\tau)\nabla_\theta \log P(\tau|\theta)]$
- ▷ **REINFORCE:** $\theta \leftarrow \theta + \alpha \left(\sum_{t=1}^H \nabla \log \pi_\theta(s_t, a_t) \sum_{t'=t}^H r_t \right)$
- ▷ **PPO:** $\theta \leftarrow \operatorname{argmax}_{\theta'} J_\theta(\theta') - \lambda \mathbb{E} D_{KL}(\theta', \theta)$
- ▷ **TRPO:** $\theta \leftarrow \operatorname{argmax}_{\theta'} J_\theta(\theta') \text{ s.t. } \mathbb{E} D_{KL}(\theta', \theta) \leq \delta$
- ▷ **REPS:** plus entropy constraint $H(\theta') - H(\theta) \leq \gamma$ (see DRLII: robotics).

“The more applied you go, the stronger theory you need”

MERCI

odalricambrym.maillard@inria.fr

odalricambrymmaillard.wordpress.com