Reinforcement Learning Approximate and Deep

Odalric-Ambrym Maillard

2023

Inria Scool

Today's lecture

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

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

Table of contents

Value Iteration

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**
    $V_{n+1}(s) = \max_a \sum_{s'} \mathcal{T}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_n(s')]$
  end for
  $n \leftarrow n + 1$
end while
for $s \in S$ **do**
  $\pi(s) = \text{argmax}_{a \in A} \sum_{s'} \mathcal{T}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_n(s')]$
end for
return V_n, π

Policy Iteration

Policy iteration algorithm

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

$n \leftarrow 0$

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

  solve (Evaluation phase)

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

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

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

  end for

  $n \leftarrow n + 1$

end while

return V_n, π_{n+1}

Monte Carlo Methods

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 infinitly and average: $V^\pi(s) = E^\pi[R_t]$

Monte Carlo Methods

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 infinitly 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 infinitly 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 :

$$\forall s \in S \quad V^*(s) = \max_{a \in A} E(r(s,a) + \gamma \sum_{s' \in S} \mathcal{T}^a_{ss'} V^*(s'))$$

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

Temporal Differences (TD) I

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

New Evaluation method for V

Widrow-Hoff like update rule:

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

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

Q-Learning

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 ← 0 until N_tot − 1 do
    s_n ← StateChoice
    a_n ← ActionChoice
    (s'_n, r_n) ← Simuler(s_n, a_n)
    % Update Q_n
    begin
        Q_{n+1} ← Q_n
        δ_n ← r_n + γ max_b Q_n(s'_n, b) − Q_n(s_n, a_n)
        Q_{n+1}(s_n, a_n) ← Q_n(s_n, a_n) + α_n(s_n, a_n)δ_n
    end
end for
return Q_{N_tot}
```

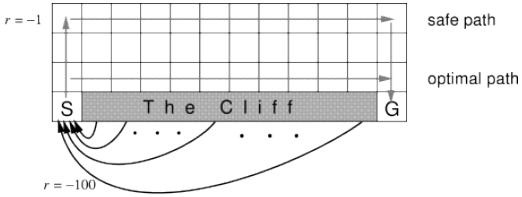
SARSA

Same for Q

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

SARSA
Init Q_0
for $n \leftarrow 0$ **until** $N_{tot} - 1$ **do**
  $s_n \leftarrow$ StateChoice
  $a_n \leftarrow$ 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 $Q_{N_{tot}}$

Q-Learning


r = −1
safe path
optimal path
S
The Cliff
G
r = −100


Sarsa
Q-learning
Reward per epsiode
−25
−50
−75
−100
0
100
200
300
400
500
Episodes

Exploration Management

Action selection

- Greedy Selection : $a = a^* = \text{argmax}_a\, Q(s, a)$
- 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

MDP Recap

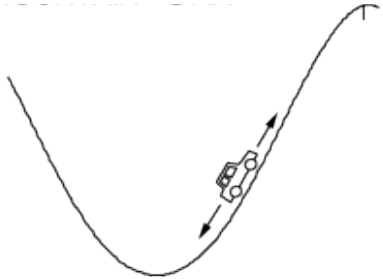
- $\mathcal{S}$ States, $\mathcal{A}$ Actions
- $R : \mathcal{S} \times \mathcal{A} \to \mathcal{P}([0,1])$ **Reward** function (mean m)
- $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ **Transition** function
- $\pi : \mathcal{S} \times \mathcal{A}$ **Policy**, or **class of policies**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

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

Challenge

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

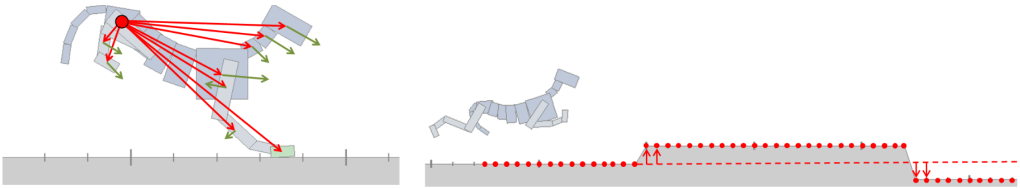
Example: Mountain-car



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

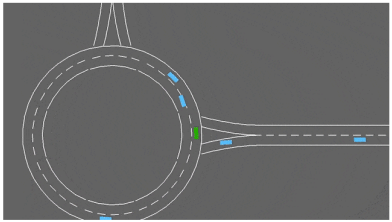


States: relative positions (red points) and velocities.
Actions: acceleration at each joint
Rewards: not falling (height of anchor point).

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

Some examples

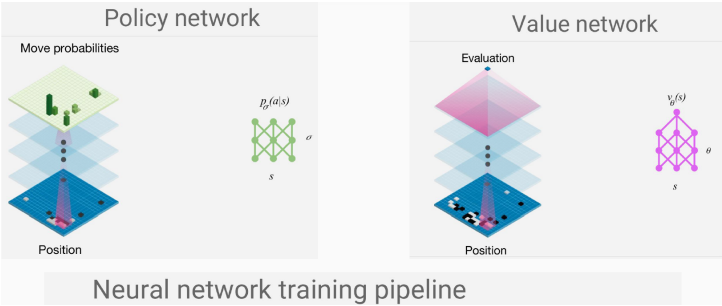
- BackGammon: 10^{20} states [Tesauro, 1995]
- Chess: 10^{50} states
- Go: 10^{170} states, 400 actions [Silver et al., 2016]
- Atari: 240x160 continuous dimensions [Mnih et al., 2015]
- Robotics: multiple degrees of freedom
- Language: very large discrete action space

Tabular RL

Complexity is polynomial. Doesn't scale up.

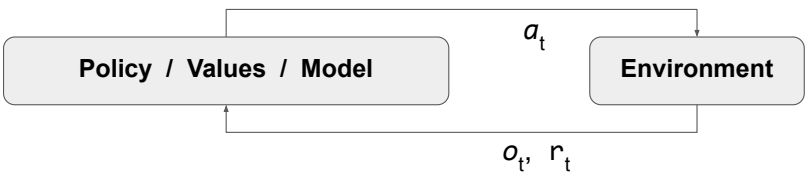
- DP (finite-horizon), VI, PI (greedy op.) require to know the model P, R and to **represent** the functions V, π.
- TD methods need to **represent** V, π but do not require P, R.
- Q-learning needs to **represent** Q, greedy operator **no longer** needs P, R.

- **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

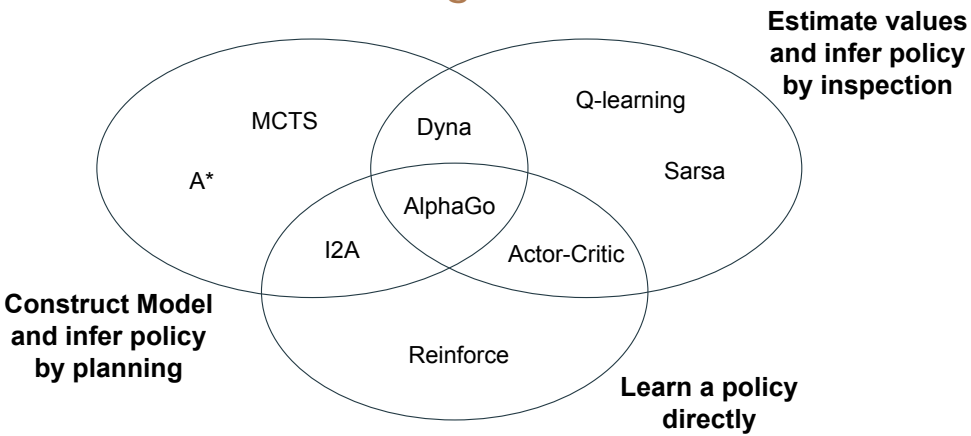


RL Solutions: The Big Picture

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

Deep Learning

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

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**)

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

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

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

- **Bellman evaluation operator** $T_\pi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$
  - 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: $v_\pi(s) = Q_\pi(s, \pi(s))$.

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

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

- **Bellman evaluation operator** $T_\pi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$
  - 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** $T_* : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$
  - 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 $V^\pi = Q_\pi(s, \pi(s))$:

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

  - resp. to $V^\star$:

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

  - 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

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

  - sampled **Bellman evaluation** operator

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

  - sampled **Bellman optimality** operator

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

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

- 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.

- 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 $f : \mathcal{X} \to \mathbb{R}$ as **weighted sum** of **known** functions $\varphi_1, \varphi_2, \ldots, \varphi_D : \mathcal{X} \to \mathbb{R}$ with **unknown** parameters $\theta_1, \ldots, \theta_D \in \mathbb{R}$:

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

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

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

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 $\theta^* = \text{argmin}_\theta J(\theta)$ from samples.

Types of parameterizations I

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

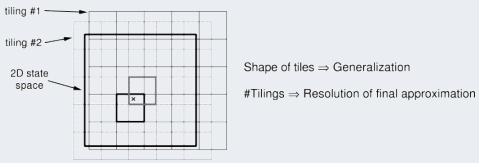
Types of parameterizations II

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



- Regression trees, neirest neighbours etc.

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

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

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

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

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

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

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

  - 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.

- **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 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 $(x_n, y_n)_{n \leqslant N}$ find parameter θ by optimizing

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

- **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 $(x_n, y_n)_{n \leqslant N}$ find parameter θ by optimizing

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

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

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

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

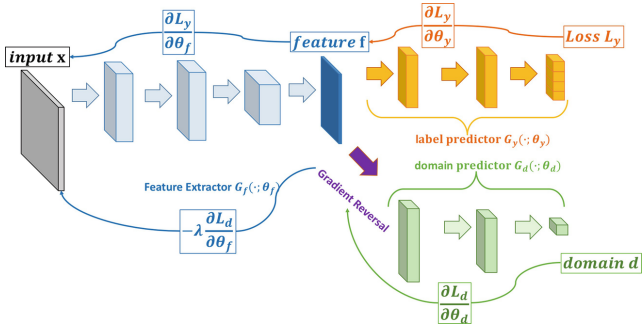
$$G_N \theta = \Phi_N^\top Y_N, \text{ where } G_N = \Phi_N^\top \Phi_N \,.$$

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

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

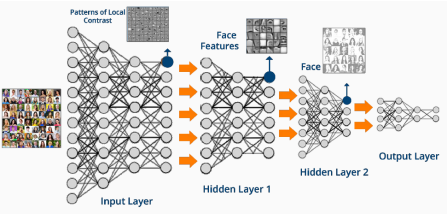
▷ To find parameter θ in **Non-linear** representation we also use **differentiation**: **gradient** methods.

Deep Learning revolution = **Automated differentiation** tools



Warning

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



DL: Computer Vision breakthrough and AutoGrad

+

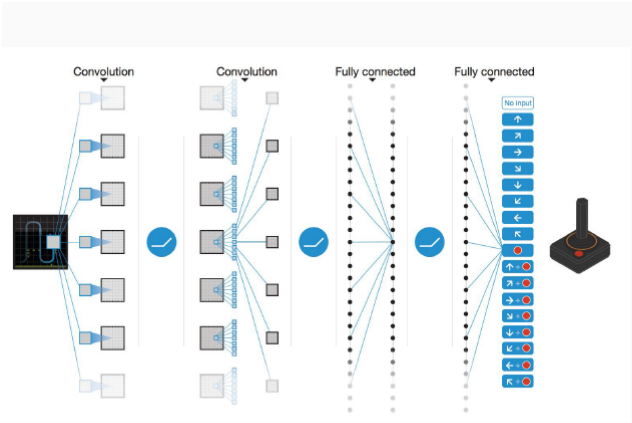
Parallelize &
Accelerate time (simulation)

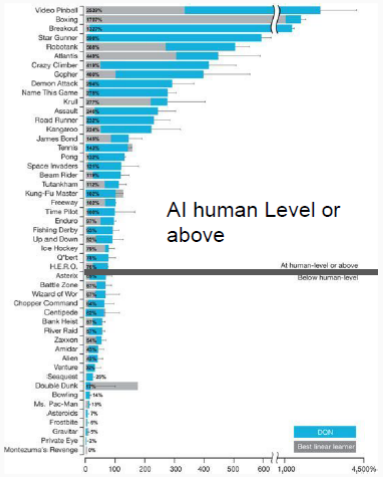
Computational Power

+

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

- Representation from visual input (raw pixels)





Q-value approximation: Deep Q-Network (DQN)

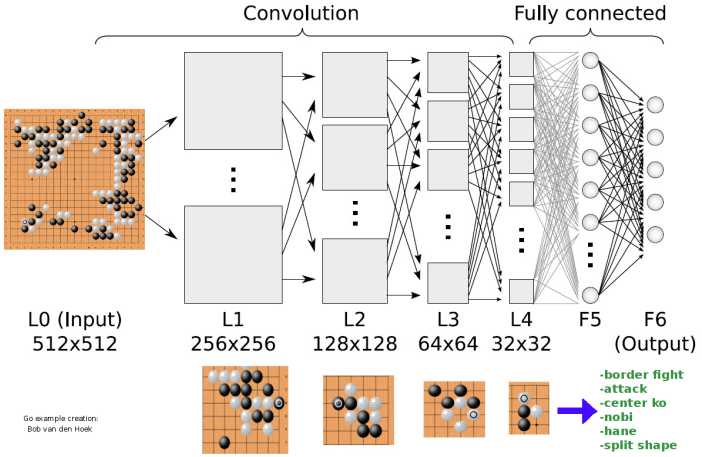


Table of contents

Least-Square Temporal Differences I

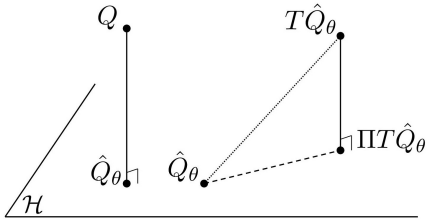
General idea (batch method)

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

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

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

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



Least-Square Temporal Differences II

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 [Bradtke and Barto, 1996, Boyan, 1999]

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

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

Approximate SARSA

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**
  $s_n \leftarrow$ StateChoice
  $a_n \leftarrow$ 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)

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

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

- Solve $\underset{\pi \in \mathcal{F}}{\operatorname{argmin}} (\max_a Q(\cdot, a) - Q(\cdot, \pi))$: **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:

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

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

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

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

Algorithm 1 Approximate value iteration

Input: A dataset $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)_{1 \leqslant i \leqslant n}\}$, 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:

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

3: Solve the regression problem with inputs (s_i, a_i) and outputs $(\widehat{T} Q_k)(s_i, a_i)$ 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:

$$\mathbf{Q_{k+1}} = \mathbb{A}\widehat{\mathbf{T}}\mathbf{Q_k}.$$

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

- Approximate value iteration:

$$\mathbf{Q_{k+1}} = \mathbb{A}\widehat{\mathbf{T}}\mathbf{Q_k}.$$

- $\mathbb{A}$ is an abstract approximation operator
- Problem: $\mathbb{A}\widehat{T}$ should be a **contraction**!
  - Otherwise divergence can (will) occur;
  - Not the case for $\Pi\widehat{T}$...
  - 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:

$$\mathbf{Q_{k+1}} = \mathbb{A}\widehat{\mathbf{T}}\mathbf{Q_k}.$$

- $\mathbb{A}$ is an abstract approximation operator
- Problem: $\mathbb{A}\widehat{T}$ should be a **contraction**!
  - Otherwise divergence can (will) occur;
  - Not the case for $\Pi\widehat{T}$...
  - 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** $\mathcal{T}$.

- Idea: Instead of considering $\mathcal{T}$, consider $\mathcal{T}_\pi$ **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 $\pi_0 \in \mathcal{A}^{\mathcal{S}}$ (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: 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 $Q_\theta \approx T_\pi Q_\theta$?

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

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

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

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

  - 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:

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

  - 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:

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

- 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:

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

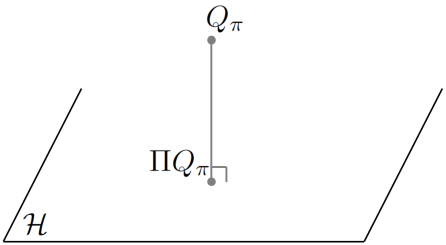
$$= \min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \left(r_i + \gamma Q_\theta(s_i', \pi(s_i')) - Q_\theta(s_i, a_i) \right)^2 .$$

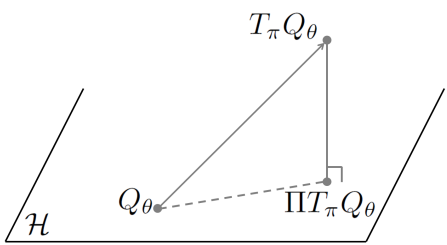
- Idea 2: **minimize the residual** $\|Q_\theta - T_\pi Q_\theta\|$ for some norm.
- With an ℓ_2-loss, parametric representation and sampled operator:

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

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

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





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

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

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

- (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 $\theta = w_\theta$:

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

- 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:

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

3: policy improvement:

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

4: **end for**

5: **return** the policy π_{K+1}

Proposition (Williams and Baird, 93)

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

Proposition (Tsitsiklis and Van Roy, 97)

Let ρ be the stationary distribution of P_π, then

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

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

Proposition

Assume $\|\varepsilon_k\|_\infty \leqslant \varepsilon$. The loss due to running π_k instead of $\pi_\star$ satisfies

$$\limsup_{k\to\infty} \|V_\star - v_{\pi_k}\|_\infty \leqslant \frac{2\gamma}{(1-\gamma)^2}\varepsilon\,.$$

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

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

Deep Q-Network I

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 $\theta^- \leftarrow \theta$

Ref: Watkins et al.

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*:

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

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:

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

Target: do not back-prop gradients here

Ref: Lin et al.

Experience Replay

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

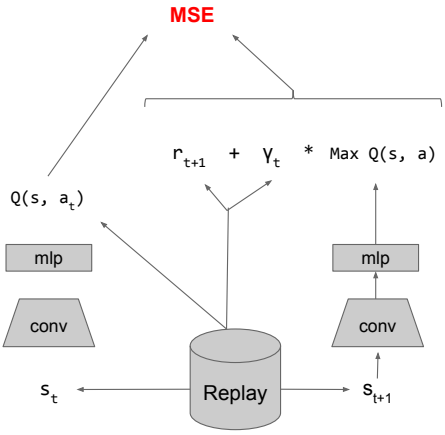
In Reinforcement Learning this is largely false

- 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**

DQN

Ref: Mnih et al.



Ref: Schaul et al.

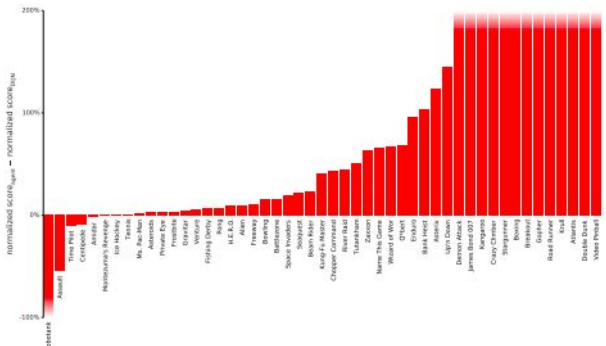
Prioritized Experience Replay

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 = loss

Rank-based
p_i = $rank_i$

Ref: van Hasselt et al.

Overestimation Bias

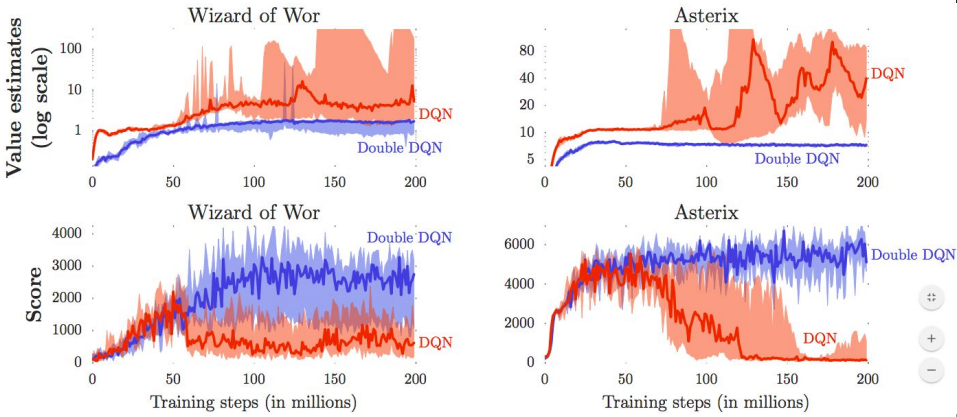
At the heart of the Q-learning operator there is the process of selecting the max value among all actions to construct the target.

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

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

$Q_{\theta}(s, argmax_{a'} \, Q_{\eta}(s, a')))$ ← **Double Q-learning**

Overestimation Bias



Improvements I

Double DQN [van Hasselt et al., 2016]

DQN:

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

Double DQN

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

Decorrelates selection and evaluation and avoid overestimation
`https://www.youtube.com/watch?v=0JYRcogPcfY`

Improvements II

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**:

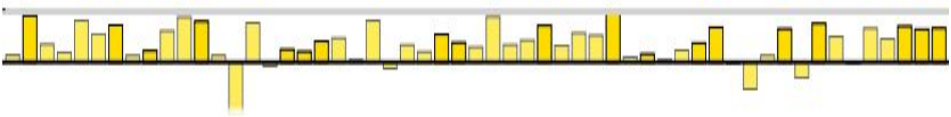
$$R_t + \gamma R_{t+1} + \ldots + \gamma^{K-1} R_{t+K-1} + \gamma^K \max_a Q_\theta(S_{t+1}, A_{t+1})$$

Ref: Hessel et al.

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

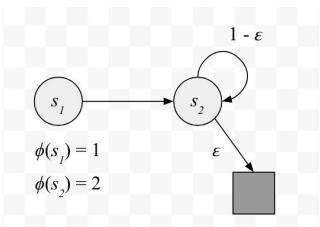
1. **Off-Policy**
    a. Estimate values of a policy on data generated from a different policy
2. **Bootstrapping**
    a. Using estimates to compute the targets for updating those same estimates
3. **Function Approximation**
    a. E.g. using neural networks

Ref: Tsitsiklis and Van Roy

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

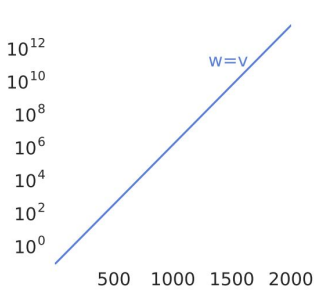
$$\left(R_t + \gamma v_\theta(S_{t+1}) - v_\theta(S_t)\right)^2$$



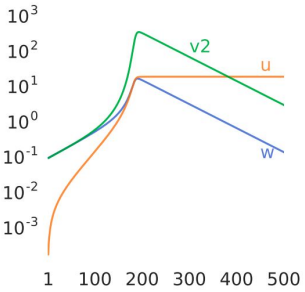
If I update **off-policy,** for instance because I use an experience replay, I may update **v(s_1) more often than v(s_2)** making **w** blow to infinity instead of converging to the optimal value of 0.

Ref: van Hasselt et al.

Deep RL and the Deadly Triad



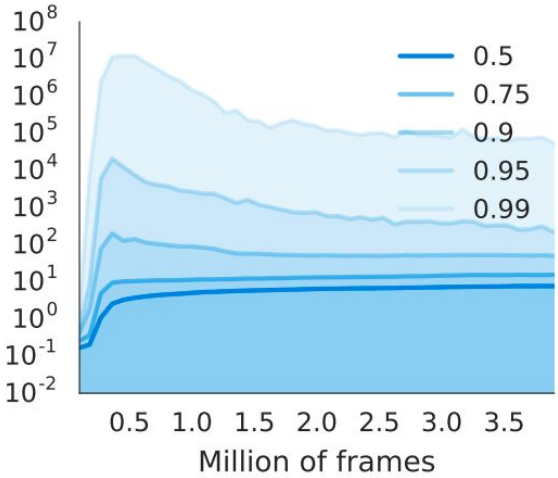
(b) $v(s) = w\phi(s)$ diverges.



(c) $v(s) = w(\phi(s) + u)$ converges.

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:

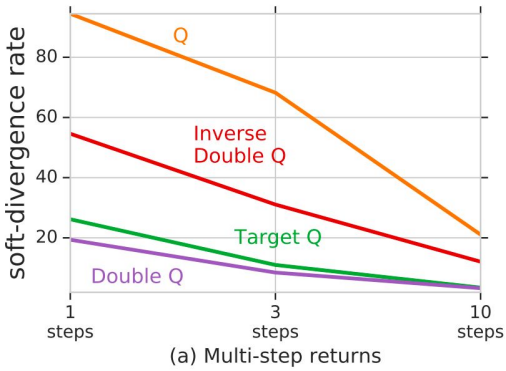
- The parameters of one are updated at each step,
- The parameters of the other are a slow copy of the first one.

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

Ref: van Hasselt et al.

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**: $\theta \leftarrow \alpha(r + \gamma Q_{\theta-}(s', \mathrm{argmax}_b\, Q_{\theta-}(s, b)) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)$
- **DDQN**: $\theta \leftarrow \alpha(r + \gamma Q_{\theta-}(s', \mathrm{argmax}_b\, Q_\theta(s, b)) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a)$
  - 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 $G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$

- 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).

Policy Iteration: recap

Let π_0 be an arbitrary stationary policy
while $k = 1, \ldots, K$ **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^{π_k} $\Longrightarrow$ Is it still converging?
Continuous actions?

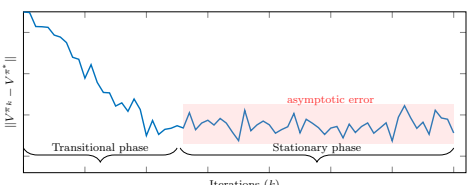
Approximate Policy Iteration

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:

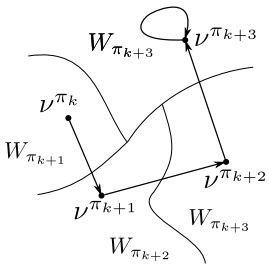
$$\limsup_{k\to+\infty} \underbrace{\|v^\star - v^{\pi_k}\|_\infty}_{\text{performance loss}} \leq \frac{2\gamma}{(1-\gamma)^2} \limsup_{k\to+\infty} \underbrace{\|v_k - v^{\pi_k}\|_\infty}_{\text{approximation error}}$$

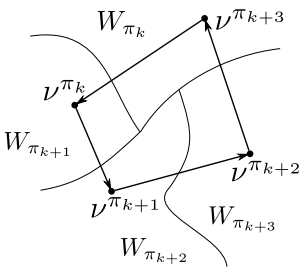


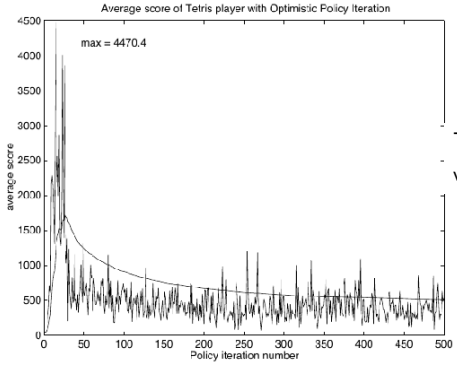
Approximate Policy Iteration: Issues

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

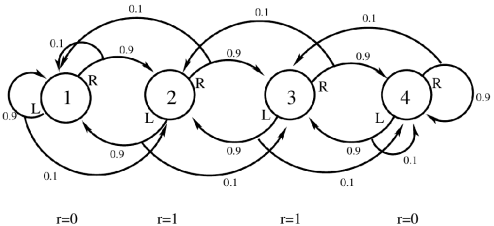


Figure 9: The problematic MDP.

From Policy Iteration to Policy Search

- Approximate a *stochastic policy* directly using function approximation

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

- 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

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

- 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

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

- 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)

▷ We can redefine **Episodic** J_e using **trajectories** (rollouts)
$\tau = (s_1, a_1, r_1, \ldots, s_{H+1})$

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

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

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

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

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

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

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

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

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

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

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

Theorem (Williams, 2992; Sutton et al. 2000)

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

- 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

$$\begin{aligned} \pi_\theta(a|s) &= \frac{e^{-\kappa Q_\theta(s,a)}}{\sum_{a' \in \mathcal{A}} e^{-\kappa Q_\theta(s,a')}} \\ \nabla_\alpha \log \pi_\theta(a|s) &= \kappa \nabla_\theta Q_\theta(s,a) - \kappa \sum_{a' \in \mathcal{A}} \pi(a'|s) \nabla_\theta Q_\theta(s,a') \end{aligned}$$

REINFORCE

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

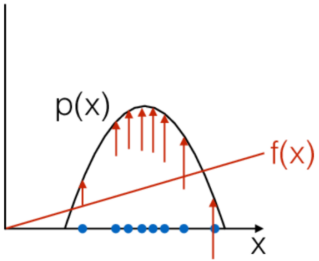
$$\theta_{k+1} = \theta_k + \alpha_k \widehat{\nabla_\theta J}(\pi_{\theta_k})$$

3. Return last policy π_{θ_K}

REINFORCE: Intuition

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

- $R(\tau_i)$ measures how *good* is sample τ_i
- Moving in the direction of $\widehat{g}_i$ 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

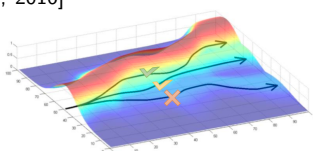


image from "CS 294-112: Deep Reinforcement Learning" slides by S. Levine

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

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

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

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 $h_n = \{s_1^n, a_1^n, r_1^n, \ldots, s_H^n, a_H^n, r_H^n\} \sim \pi_{\theta^n}$
  $PG_\theta = 0$
  for $t = 1$ to H **do**
    $R_t = \sum_{t'=t}^{H} r_{t'}^n$
    $PG_\theta \mathrel{+}= \nabla_\theta \log \pi_{\theta^n}(s_t, a_t) R_t$
  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

- 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**

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

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 $q^\pi(s,a) = \lim_{T\to\infty} \mathbb{E}\Big[\sum_{t=1}^{T} \gamma^{t-1} r(s_t,a_t)\Big]$

Going beyond the finite-horizon case

Theorem

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

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

- 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

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

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

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

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

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

Policy Gradient via Automatic Differentiation

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

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

"Pseudo loss": weighted maximum likelihood

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

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 approximatio [Sutton and Barto, 2018]

Gradient in practice

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

- **!** 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 approximatio [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?)

- 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* $\| \cdot \|$

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

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* $\|\cdot\|$

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

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* $\|\cdot\|$

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

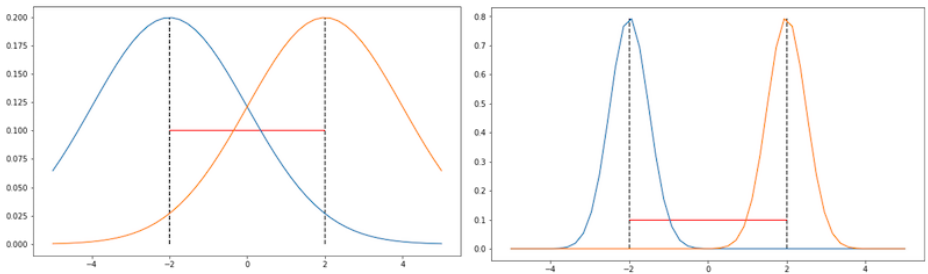
Is the Euclidean norm the best metric?
Can we use an alternative definition of (*local*) distance?

$\Longrightarrow$ 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

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

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

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

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

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

(proofs)

Natural Gradient in ML

[Martens, 2014]

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

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

Natural Gradient in ML

[Martens, 2014]

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

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

$$A = \frac{1}{2} F \quad \Longrightarrow \quad -\sqrt{2} \frac{\widetilde{\nabla} h}{\|\nabla h\|_{F^{-1}}} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \underset{d:D_{KL}(p(x|\theta)\|p(x|\theta+d)) \leq \epsilon^2}{\arg\min} \{h(\theta + d)\}$$

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. $\|x\|_B = \sqrt{x^\mathsf{T} B x}$)

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

$$A = \frac{1}{2}F \quad \Longrightarrow \quad -\sqrt{2}\frac{\widetilde{\nabla} h}{\|\nabla h\|_{F^{-1}}} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \underset{d:D_{KL}(p(x|\theta)\|p(x|\theta+d)) \leq \epsilon^2}{\arg\min} \{h(\theta + d)\}$$

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 $d \to 0$, and so will be (approximately) symmetric in a local neighborhood [Martens, 2014]

! $\widetilde{\nabla} h$ is be *invariant* to the choice of parameterization

Natural Policy Gradient

Trust-region objective

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

Approximate objective and KL

$$L_{\theta_k}(\theta) \approx L_{\theta_k}(\theta_k) + g^\intercal(\theta - \theta_k)$$
$$\overline{D}_{KL}(\theta \| \theta_k) \approx \frac{1}{2}(\theta - \theta_k)^\intercal F(\theta - \theta_k)$$

$\Longrightarrow$

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

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., $span(g, Hg, \dots H^{j-1}g)$)

$\Longrightarrow$ *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:

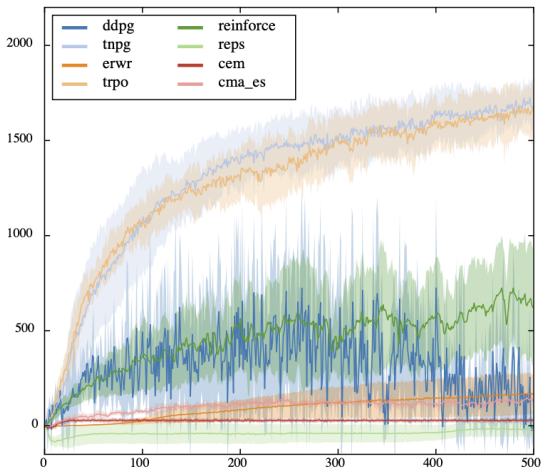
- Use conjugate gradient to compute $F^{-1}g$ without inverting F [Pascanu and Bengio, 2013]
- With j iterations, CG solves systems of equations $Hx = g$ for x by finding projection onto Krylov subspace (i.e., $span(g, Hg, \dots H^{j-1}g)$)

$\implies$ *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]

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

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]

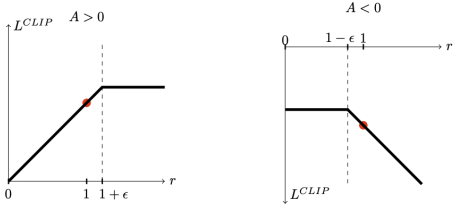
2 *Clipped Objective*

- Recall surrogate objective

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

- Form a lower bound via clipped importance ratios

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



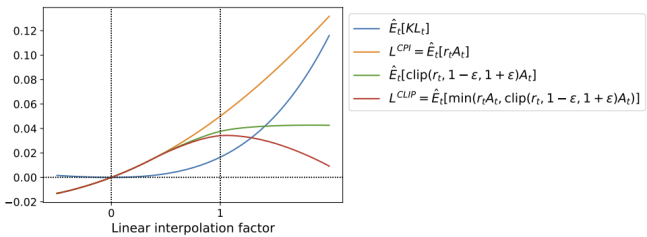
- $\pi_{k+1} = \arg\max_\pi L_{\pi_k}^{\mathsf{CLIP}}(\pi)$

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}

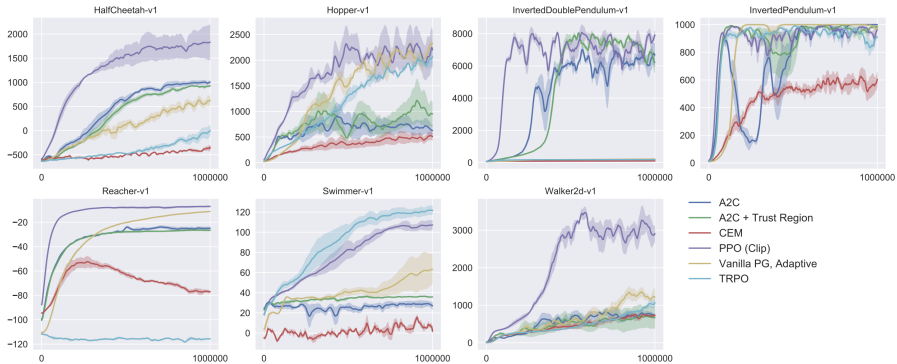


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

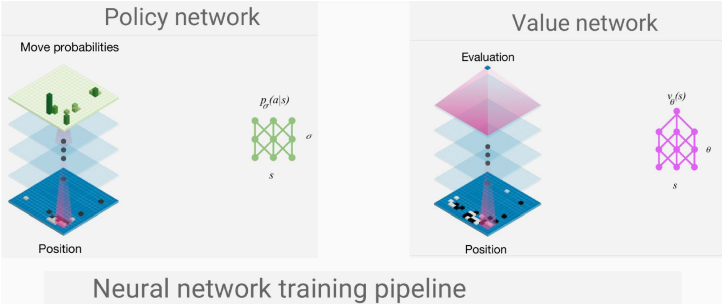
- **Trust Region Policy Optimization:**

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

- **Proximal Policy Optimization:**

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

- **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**: $\theta \leftarrow \theta + \alpha\Big(\sum_{t=1}^{H} \nabla \log \pi_\theta(s_t, a_t) \sum_{t'=t}^{H} r_t\Big)$
- **PPO**: $\theta \leftarrow \underset{\theta'}{\operatorname{argmax}}\, J_\theta(\theta') - \lambda \mathbb{E} D_{KL}(\theta', \theta)$
  - **TRPO**: $\theta \leftarrow \underset{\theta'}{\operatorname{argmax}}\, J_\theta(\theta')$ s.t. $\mathbb{E} D_{KL}(\theta', \theta) \leqslant \delta$
  - **REPS**: plus entropy constraint $H(\theta') - H(\theta) \leqslant \gamma$ (see DRLII: robotics).

"*The more applied you go, the stronger theory you need*"

Merci

odalricambrym.maillard@inria.fr
odalricambrymmaillard.wordpress.com