REINFORCEMENT LEARNING

Seminar @ UPC TelecomBCN Barcelona (2nd edition). Spring 2020.

Instructors

Josep Vidal

Margarita Cabrera

Xavier Giró-i-Nieto

Organizers

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecom BCN

Supporters

Google Cloud

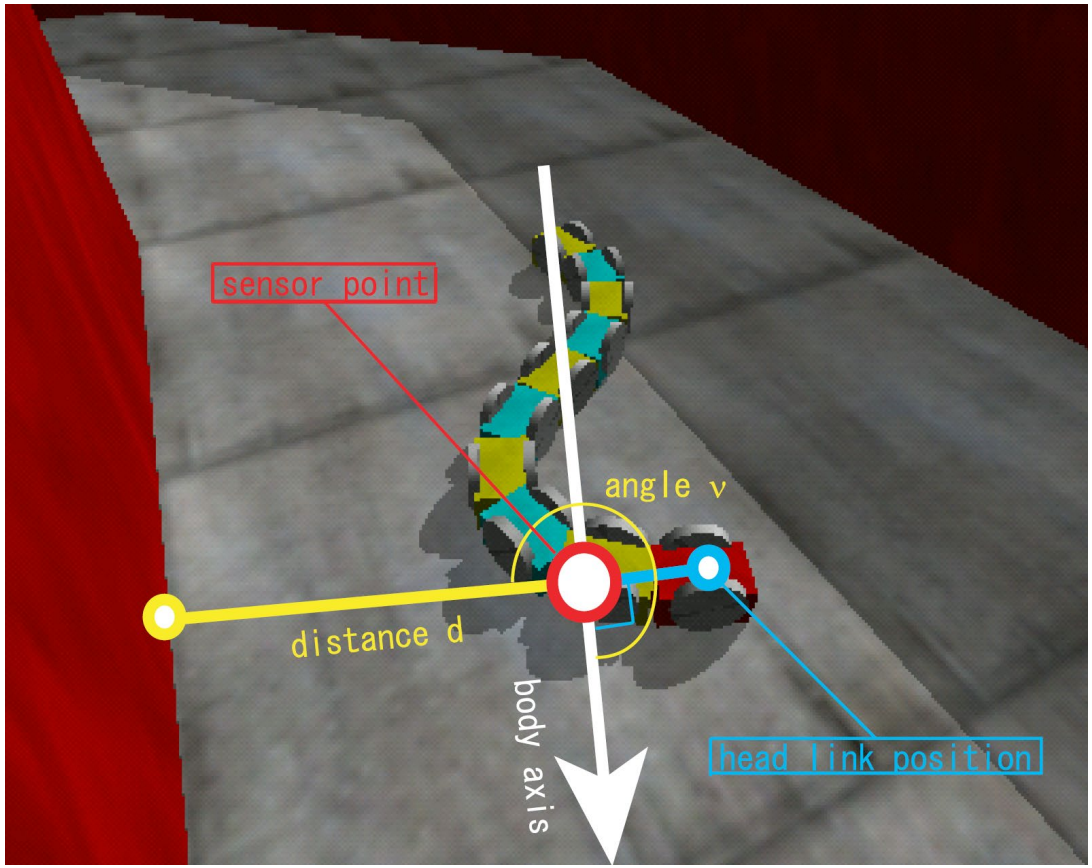
GitHub Education

+ info: http://bit.ly/upcrl-2020

telecos BCN

UPC

6. Value Function Approximation and Policy Gradient



Motivation

Problem: Estimating the optimal action-value function $q^*(s,a)$ might be feasible if a few state-action pairs are possible, but impossible for large spaces, e.g.

- Backgammon: 10^{20} states
- Atari Space Invaders: 10^{82} states are pixel values in a 210×160 screenshot (more states than atoms in the universe)
- Computer Go: 10^{170} states
- Helicopter: continuous state space

Until now we have worked with a look-up table where

- Every state s has an entry $v(s)$.
- Every state-action pair (s,a) has an entry $q(s,a)$.

- A look-up table is not practical when having a lot of states:
  - Many states / many actions to store in memory.
  - A finite amount of experience is available.
  - Too slow to learn the value of each state individually.

- **Solution:** Learn an approximation of the optimal action-value function $q(s,a,\mathbf{w})$ using a parametric function.
- We can adapt the model free methods and use a parametric function approximation with a set of weights $\mathbf{w}$ independent of the number of states:

$$\hat{v}(s,\mathbf{w}) \cong v_\pi(s) \qquad \hat{q}(s,a,\mathbf{w}) \cong q_\pi(s,a)$$

$$\mathbf{w} \in \mathbb{R}^d$$

Our method should achieve a generalization from seen states to unseen states.

Which function approximators?

- Linear combination of features
- Polynomial/Fourier bases
- Neural networks
- Decision trees
- …

We need differentiable in **w** function approximators, and a training method that is suitable for non-stationary data.

Summary for the lecture

- Value Function Approximation RL
  - Learned Value Function
- Policy Gradient RL
  - Learned Policy
- Actor – Critic RL
  - Jointly learn Value Function and Policy

VFA: Stochastic Gradient Descent Algorithm

- Select a differentiable function as for instance a linear function, a Fourier series or a Neural Network (NN):

$$\hat{v}(s, \mathbf{w})$$

- Define a goal function to be minimized as the MSE:

$$J(\mathbf{w}) = E_\pi \left\{ \left(v_\pi(s) - \hat{v}(s, \mathbf{w}) \right)^2 \right\}$$

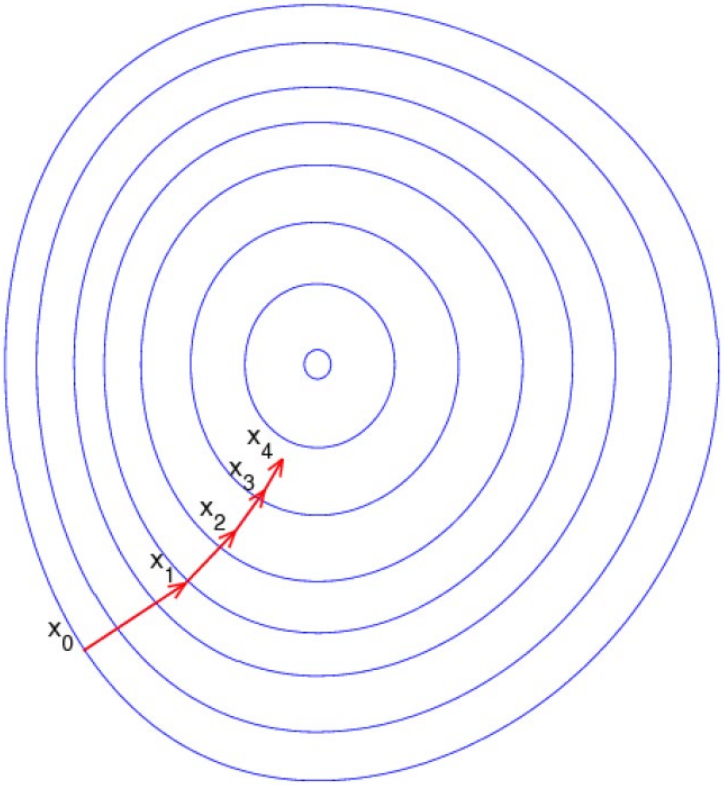
- Obtain the gradient w.r.t. $\mathbf{w}$

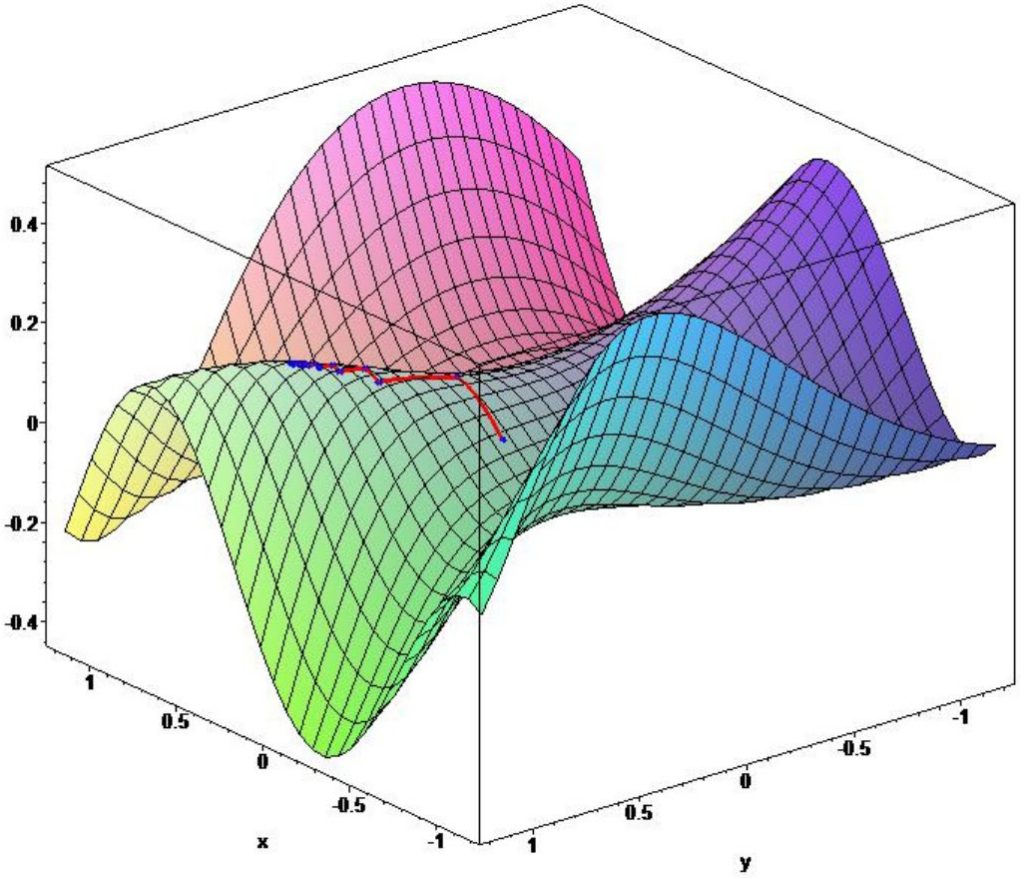
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = -2 E_\pi \left\{ \left(v_\pi(s) - \hat{v}(s, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) \right\}$$

- Stochastic Gradient Descent Algorithm: substitute the gradient by its sampled instantaneous value:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha E_\pi \left\{ \left(v_\pi(s) - \hat{v}(s, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) \right\}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(v_\pi(s) - \hat{v}(s, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$





From David Silver, Course on RL, 2015

VFA: Linear value function approximation

- Let us take the simplest example. Assume each state is represented by a number of features (e.g. distance of a robot to landmarks):

$$\mathbf{x}(s)^T = \begin{bmatrix} x_1(s) & \cdots & x_n(s) \end{bmatrix}$$

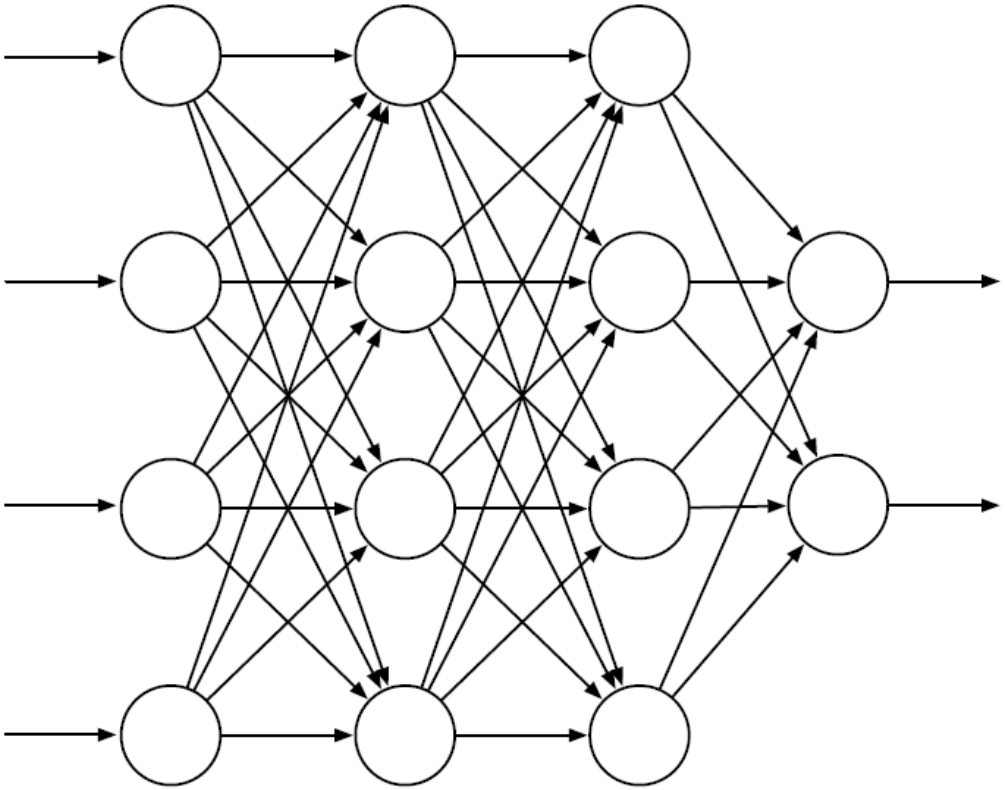
- The value function is a linear combination of these features

$$\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w}$$

- In this case

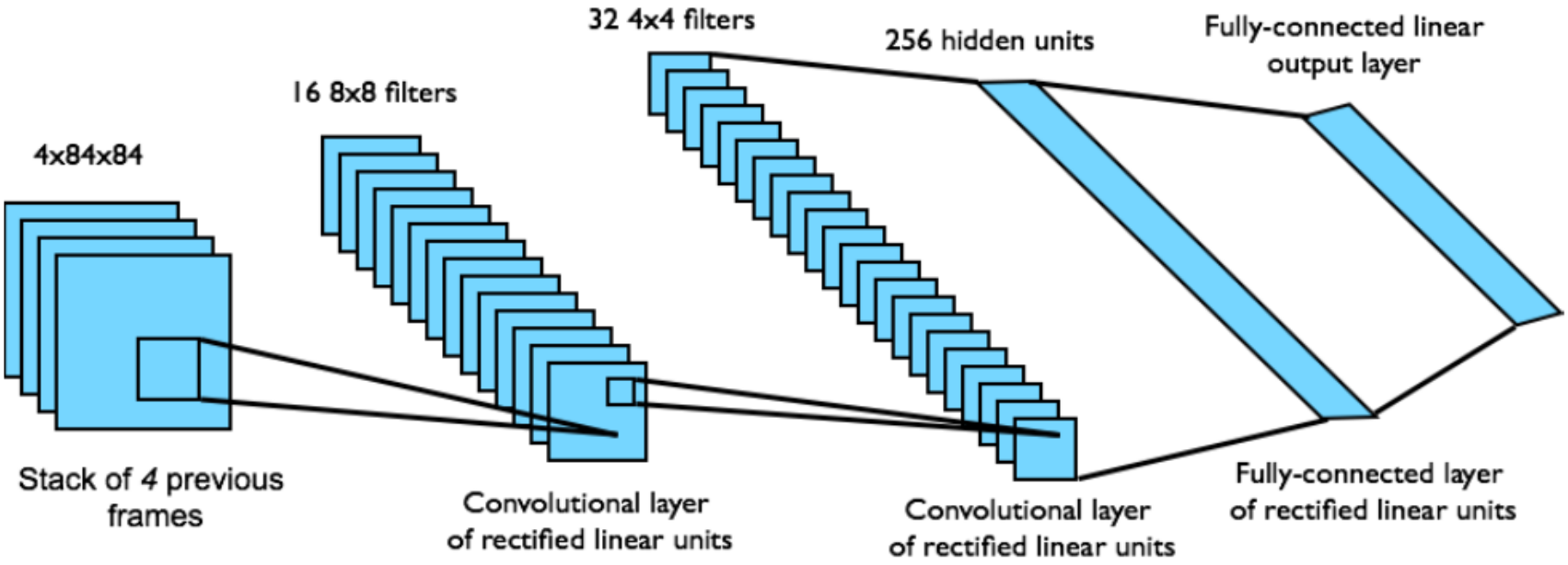
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(v_\pi(s) - \hat{v}(s, \mathbf{w}) \right) \mathbf{x}(s)$$

VFA: Other models…



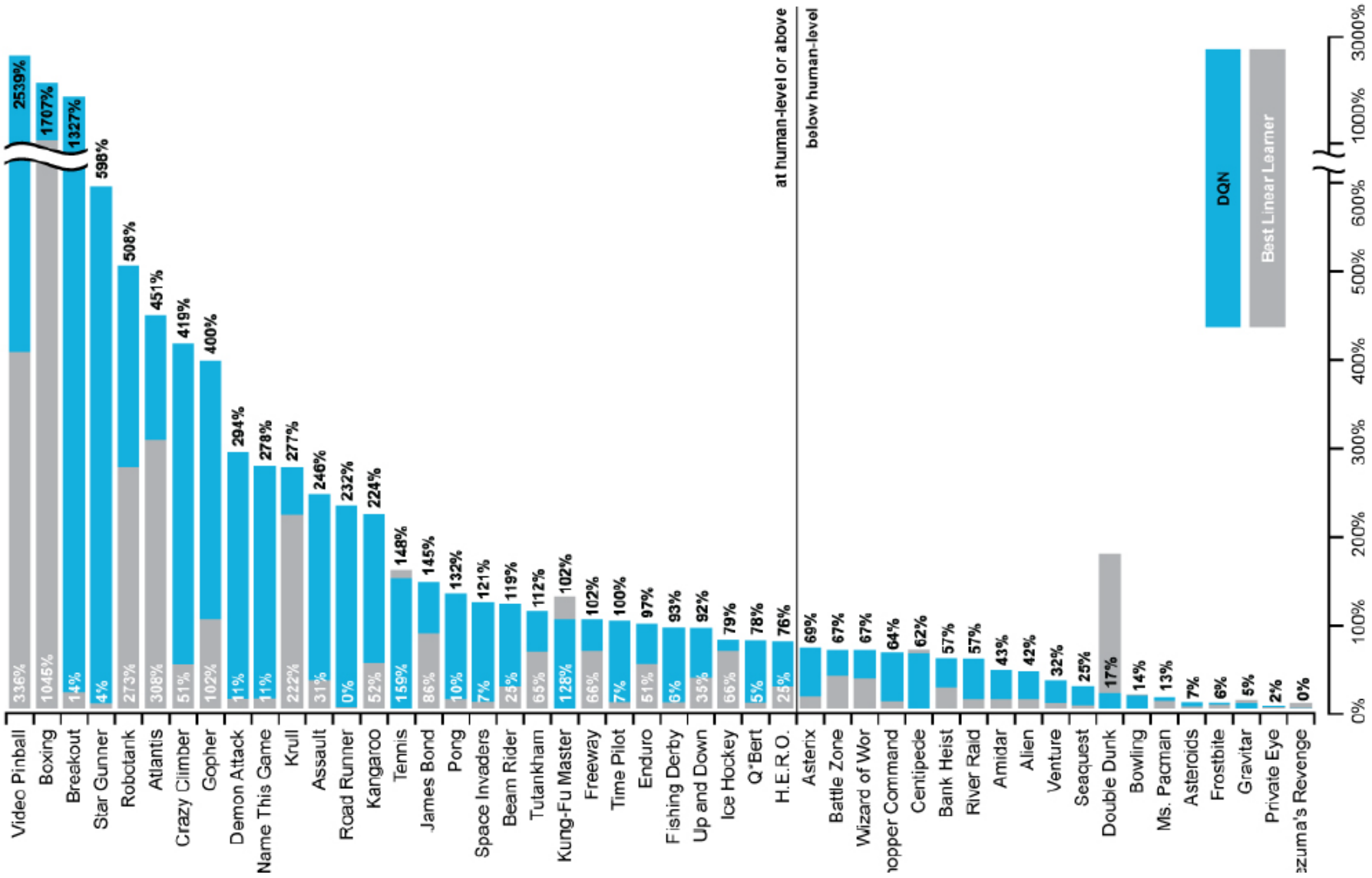
A generic feedforward artificial neural network is a universal function approximator

A deep neural network for q-learning… more in next chapter.



From David Silver, Course on RL, 2015

Performance of NN in some Atari games



From David Silver, Course on RL, 2015

VFA: Incremental Prediction Algorithms

In RL the true value function $v_\pi(s)$ is not known, so it has to be computed based on observed rewards:

- In Montecarlo G_t is an unbiased estimation of $v_\pi(s)$. It converges to a local minimum:

$$\text{MC:} \quad \mathbf{w} \leftarrow \mathbf{w} + \alpha \left(G_t - \hat{v}(s_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(s_t, \mathbf{w})$$

- In Time Difference the estimation of the true value is a biased sample. When used with a linear model, TD converges to the global minimum.

$$\text{SARSA:} \quad \mathbf{w} \leftarrow \mathbf{w} + \alpha \left(R_{t+1} + \gamma \hat{v}(s_{t+1}, \mathbf{w}) - \hat{v}(s_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(s_t, \mathbf{w})$$

This cannot be claimed in general, as the target function depends also on $\mathbf{w}$ (semi-gradient method). On the positive side, it has lower variance than MC because it depends only on next state not on all states until the end of the episode.

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S' is terminal

It can be easily adapted to Expected SARSA or Q-learning.

VFA: Incremental Control Algorithms

Control with VFA is straightforward, the q-function is approximated:

$$\hat{q}(s,a,\mathbf{w}) \cong q_\pi(s,a) \qquad \mathbf{w} \in \mathbb{R}^d$$

Updating the parameters can be done using a gradient method:

$$\text{MC:} \quad \mathbf{w} \leftarrow \mathbf{w} + \alpha\left(G_t - \hat{q}(s_t,a_t,\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{q}(s_t,a_t,\mathbf{w})$$

$$\text{SARSA:} \quad \mathbf{w} \leftarrow \mathbf{w} + \alpha\left(R_{t+1} + \gamma\hat{q}(s_{t+1},a_{t+1},\mathbf{w}) - \hat{q}(s_t,a_t,\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{q}(s_t,a_t,\mathbf{w})$$

The chosen action can be determined using an ε-greedy technique, as seen earlier:

$$a_t^* = \arg\max_a \hat{q}(s_t,a,\mathbf{w})$$

VFA: Incremental Control Algorithms

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

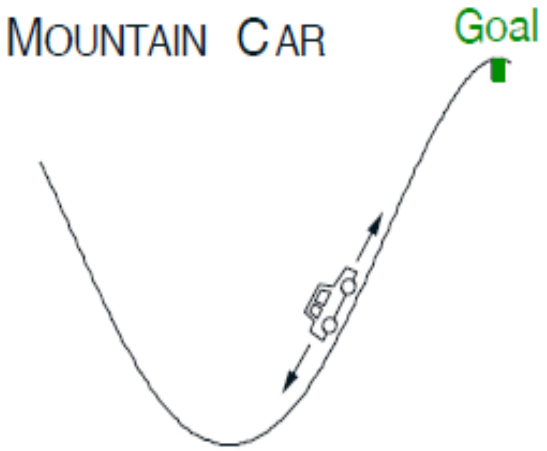
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

It can be easily adapted to Expected SARSA or Q-learning.

Example 6.1: The mountain car task (1)

Task of driving an underpowered car up a steep mountain road.

- **Problem:** Gravity is stronger than the car's engine, and even at full throttle the car cannot accelerate up the steep slope.
- **Solution:** to first move away from the goal and up the opposite slope on the left. Then, by applying full throttle the car can build up enough inertia to carry it up the steep slope.
- **Reward:** -1 on all time steps until the car moves past its goal position at the top of the mountain, which ends the episode.
- **Three actions**: full throttle forward (+1), full throttle reverse (-1), and zero throttle (0).



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

Example 6.1: The mountain car task (2)

The car moves according to a simplified physics.

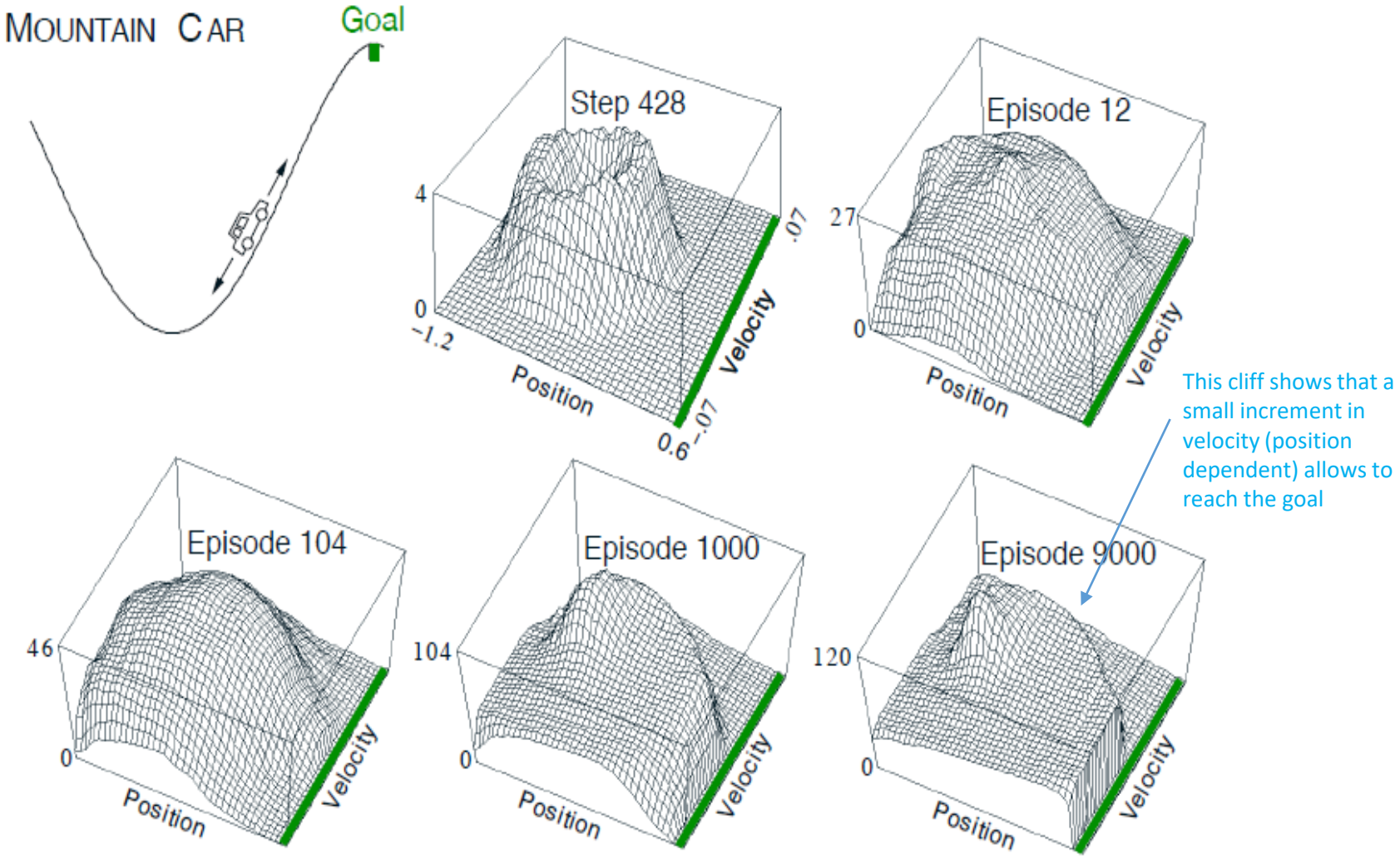
- Position $x_{t+1} \doteq \min(\max(-1.2; x_t + \dot{x}_{t+1}); +0.5)$
- Velocity $\dot{x}_{t+1} \doteq \min(\max(-0.7; \dot{x}_t + 0.001 a_{t+1} - 0.0025\cos(3x_t)); +0.7)$

- When x_{t+1} reaches the left bound, x'_{t+1} is reset to zero.
- When x_{t+1} reaches the right bound, the goal is reached and the **episode is terminated**.
- Each episode starts from a random position x_t in [-0.6;+0.4) and zero velocity.
- Feature vector: $[x_t \;\; \dot{x}_t]^T$
- State-action value function approximation:

$$\hat{q}(s, a, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s, a) \qquad \mathbf{w} \in \mathbb{R}^2$$

Example 6.1: The mountain car task (3)

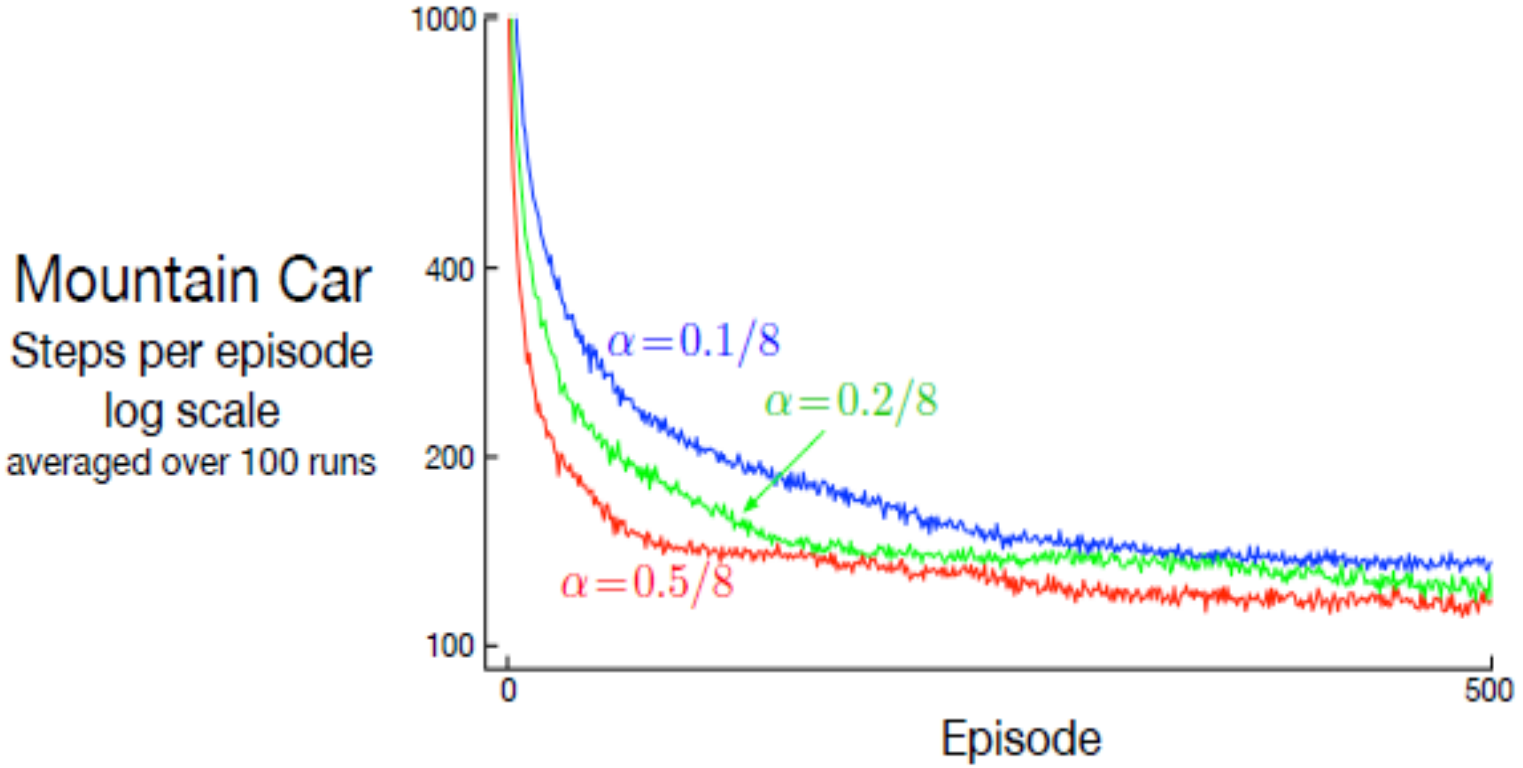
Results: negative of the value function (the cost-to-go function).



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

Example 6.1: The mountain car task (4)

Learning curves for the semi-gradient SARSA method with tile-coding function approximation and greedy action selection…



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

VFA: Batch Prediction Algorithms

- Gradient descent is simple and valuable for non-stationary environments, but it is not sample-efficient. In batch prediction we seek to find the best fitting value function over an entire episode.

- We seek to minimize the Least Square function:

$$\mathcal{L}(\mathbf{w}) = \sum_{t=1}^{T} \left(v_t^{\pi} - \hat{v}(s_t, \mathbf{w}) \right)^2$$

given the measured state-values set: $D = \left\{ \left\langle s_1, v_1^{\pi} \right\rangle, \left\langle s_2, v_2^{\pi} \right\rangle, \ldots, \left\langle s_T, v_T^{\pi} \right\rangle \right\}$

- The LS solution is found as: $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \mathbf{0}$

VFA: Batch Prediction Algorithms

In the linear case $\hat{v}(s_t, \mathbf{w}) = \mathbf{x}^T(s_t)\mathbf{w}$ a closed solution is easily computed:

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}^T(s_t) \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi}$$

requiring $O(N^3)$ operations for N features. In practice the value of v_t^{π} has to be a noisy or biased version:

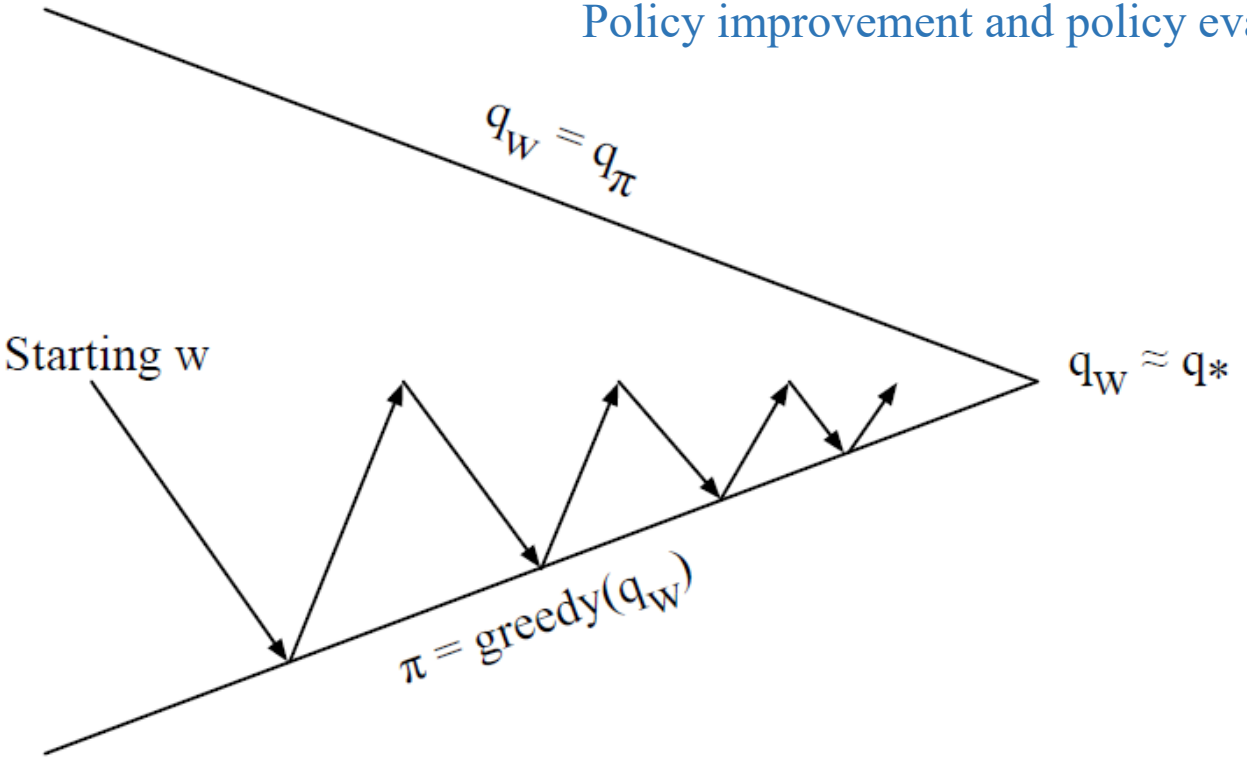
MC: $$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}^T(s_t) \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) G_t$$

SARSA: $$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(s_t)\left(\mathbf{x}^T(s_t) - \gamma \mathbf{x}^T(s_{t+1})\right) \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) R_{t+1}$$

Likewise, we can use least squares to approximate $q_\pi(s,a)$ using a linear approximation, given the experienced data set:

$$D = \left\{ \left\langle (s_1, a_1), v_1^\pi \right\rangle, \left\langle (s_2, a_2), v_2^\pi \right\rangle, \ldots, \left\langle (s_T, a_T), v_T^\pi \right\rangle \right\}$$

Policy improvement and policy evaluation



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

VFA: Batch Control Algorithms

We are interested in improving the policy, so some kind of exploration (off-policy) is needed. Let us adopt the same ε-greedy idea as in SARSA or Q-learning:

- Use the experience generated by an old policy π
- Consider an alternate action $a' = \pi_{new}(s_{t+1})$
- Update the estimated $\hat{q}(s, a, \mathbf{w})$ using the value of the alternative action:

$$R_{t+1} + \gamma \hat{q}(s_{t+1}, a', \mathbf{w})$$

VFA: Batch Control Algorithms

Example of LS-TD-Q in the linear case…

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(s_t, a_t) \left(\mathbf{x}^T(s_t, a_t) - \gamma \mathbf{x}^T \left(s_{t+1}, \pi\left(s_{t+1}\right) \right) \right) \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t, a_t) R_{t+1}$$

The following code re-evaluates the experience $\mathcal{D}$ with different policies…

Policy selection for LS-TD-Q with experience $\mathcal{D}$

Function LSPI-TD ($\mathcal{D}$, π_0)
    $\pi' \leftarrow \pi_0$
    Repeat
        $\pi \leftarrow \pi'$
        Update **w** as given by the LS-TD-Q linear case
        Update $Q(s,a)$
        $\pi'(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q(s,a)$, for all s in $\mathcal{S}$
    Until $\pi \simeq \pi'$
    Return π
End function

Downsides of VFA:

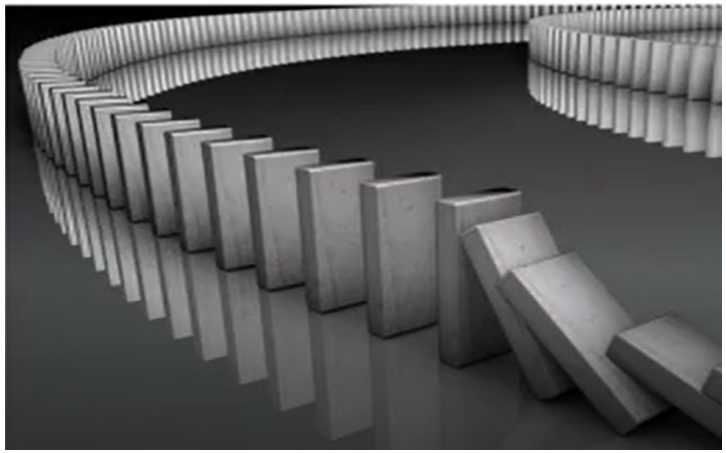
$q(s,a)$ often changes abruptly in s and a, problem depending:

- Close states could be arbitrary far in value
- As a consequence, gradients are unstable, a lot of data may be needed
- Numerical problems may appear
- In TD approaches, errors in the estimate may propagate

Non-stationarity:

- The use of GPI (change in policy) lead to invalid targets in MC and TD
- Small changes in q-values may lead to large changes in policy, in training data, in gradients, and hence large updates in q-values, large changes in policy → oscillating behaviour
- The environment can be non-stationary

Divergence is an issue!



Our final goal (policy) is replaced by better approximation

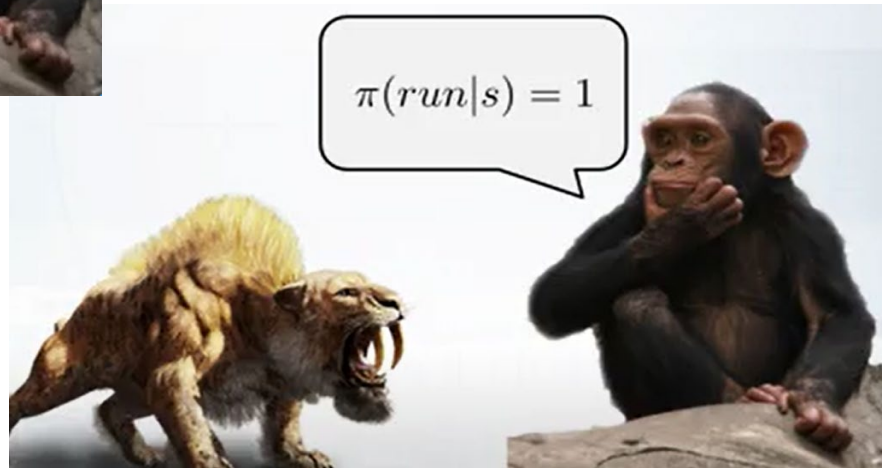
Which prediction is better A or B?

A simple 2-state world

	True	A	B
$Q(s_0,a_0)$	1	1	2
$Q(s_0,a_1)$	2	2	1
$Q(s_1,a_0)$	3	3	3
$Q(s_1,a_1)$	100	50	100
		better policy	lower MSE

Q-learning will prefer B, but better approximation of $q(s,a)$ does not always lead to better policies.

Computing $q(s,a)$ is often harder than picking optimal actions!
Why not learning agent's policy directly?



Policy Gradient

PG methods aim at selecting actions without previously checking the value function:

$$\hat{\pi}_{\boldsymbol{\theta}}(a \mid s, \boldsymbol{\theta}) = \Pr\left\{ A_t = a \middle| S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta} \right\} \qquad \boldsymbol{\theta} \in \mathbb{R}^{d'}$$

- Advantages
  - Better convergence properties than VFA methods
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies

- Drawbacks
  - Typically converge to a local rather than to the global optimum
  - Evaluating a policy is typically inefficient and shows high variance

Policy Gradient

- A performance measure $J(\boldsymbol{\theta})$ is needed: the value function for the policy

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0)$$

so that a Stochastic Gradient Ascent Algorithm can be applied:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \hat{\nabla} J(\boldsymbol{\theta}_t)$$

- The parameterized policy has to be differentiable w.r.t $\boldsymbol{\theta}$.

If the action-space is discrete (continuous) the natural function is the exponential softmax (Gaussian) distribution:

Instead of linear, it can also be a NN

$$\hat{\pi}_{\boldsymbol{\theta}}(a \mid s, \boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{\theta}^T \mathbf{x}(s,a)\right)}{\sum_{a' \in A} \exp\left(\boldsymbol{\theta}^T \mathbf{x}(s,a')\right)} \quad \text{or} \quad N\left(\boldsymbol{\theta}^T \mathbf{x}(s,a), \sigma^2(s,a)\right)$$

Policy Gradient

- Unlike ε-greedy strategies in VFA, the optimal policies may change smoothly and not be deterministic.
- How can we estimate the performance gradient w.r.t. θ if the gradient depends on the unknown effect of policy changes on the state distribution because it is a function of the environment?
- The policy gradient theorem gives the answer…

Policy Gradient Theorem

Policy Gradient Theorem. We can express the performance function as

We have to compute the gradient wrt $\boldsymbol{\theta}$, but the policy π is also present in the return $q(s,a)$!

$$J(\boldsymbol{\theta}) = E_s\left\{\sum_{a\in A}\pi\left(a|s,\boldsymbol{\theta}\right)q_\pi\left(s,a\right)\right\} = \sum_{\substack{a\in A\\ s\in S}} p(s)\pi\left(a|s,\boldsymbol{\theta}\right)q_\pi\left(s,a\right)$$

Using elementary calculus we can state (Sutton, section 13.2)…

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= E_s\left\{\sum_{a\in A} q_\pi\left(s,a\right)\nabla_{\boldsymbol{\theta}}\pi\left(a|s,\boldsymbol{\theta}\right)\right\} + \cancel{E_s\left\{\sum_{a\in A}\pi\left(a|s,\boldsymbol{\theta}\right)\nabla_{\boldsymbol{\theta}} q_\pi\left(s,a\right)\right\}}\\ &= E_{\pi_{\boldsymbol{\theta}}}\left\{q_\pi\left(s,a\right)\nabla_{\boldsymbol{\theta}}\ln\pi\left(a|s,\boldsymbol{\theta}\right)\right\}\\ &= \sum_{\substack{a\in A\\ s\in S}} p(s)\pi\left(a|s,\boldsymbol{\theta}\right)\nabla_{\boldsymbol{\theta}}\ln\pi\left(a|s,\boldsymbol{\theta}\right)q_\pi\left(s,a\right)\end{aligned}$$

In practice, replace this expectation…

Notes on the Policy Gradient Theorem…

1. Replace the expectation by an average over episodes

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \simeq \frac{1}{N} \sum_{i=1}^{N} \sum_{\substack{a \in A \\ s \in S}} \nabla_{\boldsymbol{\theta}} \ln \pi\left(a \middle| s, \boldsymbol{\theta}\right) q_{\pi}\left(s, a\right)$$

or by an instantaneous value $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \simeq \nabla_{\boldsymbol{\theta}} \ln \pi\left(a \middle| s, \boldsymbol{\theta}\right) q_{\pi}\left(s, a\right)$

2. The gradient expression is used in the second equality…

$$\nabla_{\boldsymbol{\theta}} \ln f(\boldsymbol{\theta}) = \frac{1}{f(\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \quad \Rightarrow \quad f(\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \ln f(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$$

PG: The REINFORCE algorithm

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t,\boldsymbol{\theta})$ ← Gradient update

$q(s,a)$ is directly observable

Learning rate

Discount factor

Actor - Critic

We have seen two main types of RL methods:

- Value-Based:
  - Approximate the $q(s,a)$, which is a mapping between an action and a value.
  - More sample efficient and steady.
- Policy-Based:
  - Find the optimal policy directly without $q(s,a)$ as a middleman.
  - Better for continuous and stochastic policies, have a faster convergence.

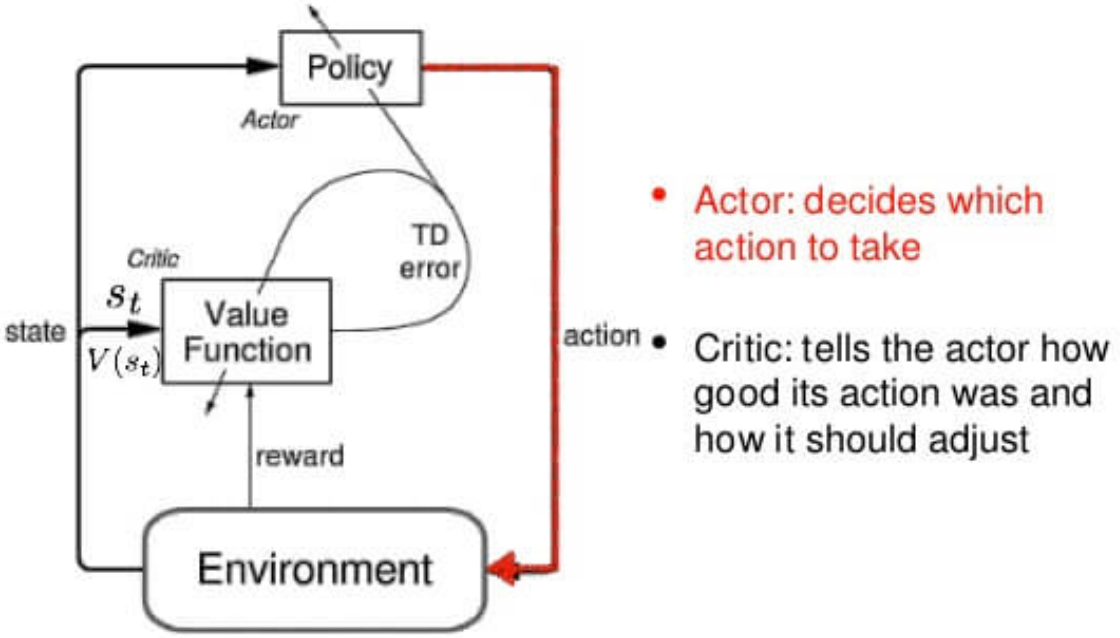
If these two algorithmic families are merged: Actor-Critic

- By reducing the variance of policy-based methods, this procedure will learn to play more efficiently than the two methods separately.
- Tight integration of the q-value knowledge.
- Actor-Critic algorithms are the base behind almost every modern RL.

Actor - Critic

The model is split in two: one for computing an action based on a state and another one to produce the $q(s,a)$ values of the action.

- The actor takes as input the state and outputs the best action (policy-based).
- The critic evaluates the action by computing $q(s,a)$ (value based).



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

Actor - Critic

- We use a critic to estimate the action-value function,

$$\hat{q}(s,a,\mathbf{w}) \cong q_{\pi_{\boldsymbol{\theta}}}(s,a) \qquad \mathbf{w} \in \mathbb{R}^{d}, \boldsymbol{\theta} \in \mathbb{R}^{d'}$$

- Actor-critic algorithms maintain two sets of parameters:
  - Critic updates action-value function parameters $\mathbf{w}$
  - Actor updates policy parameters $\boldsymbol{\theta}$ in direction suggested by critic

- Actor-critic algorithms compute an approximate policy gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \cong E_{\pi_{\boldsymbol{\theta}}} \left\{ \hat{q}(s,a,\mathbf{w}) \nabla_{\boldsymbol{\theta}} \log \pi\left(a \middle| s, \boldsymbol{\theta}\right) \right\}$$

Observed G in REINFORCE algorithm

Actor-Critic algorithm

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$
Repeat forever:
  Initialize S (first state of episode), sample $A \sim \pi(a|s,\boldsymbol{\theta})$
  $I \leftarrow 1$
  While S is not terminal:
    Observe R, S'
    Sample $A' \sim \pi(a|S',\boldsymbol{\theta})$
    $\delta \leftarrow R + \gamma \hat{q}(S',A',\mathbf{w}) - \hat{q}(S,A,\mathbf{w})$
Critic → $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I\, \delta\, \nabla_{\mathbf{w}} \hat{q}(S,A,\mathbf{w})$
Actor → $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\, \hat{q}(S,A,\mathbf{w})\, \nabla_{\boldsymbol{\theta}} \ln \pi\left(A|S,\boldsymbol{\theta}\right)$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$
    $A \leftarrow A'$

… back to VFA, gradient update based on MMSE criterion. For the linear approximation:

$$\nabla_{\mathbf{w}} \hat{q}(S,A,\mathbf{w}) = \mathbf{x}(S,A)$$

(if S' is terminal, then $\hat{q}(S',A,\mathbf{w}) \doteq 0$)

Gradient updates for the parameters

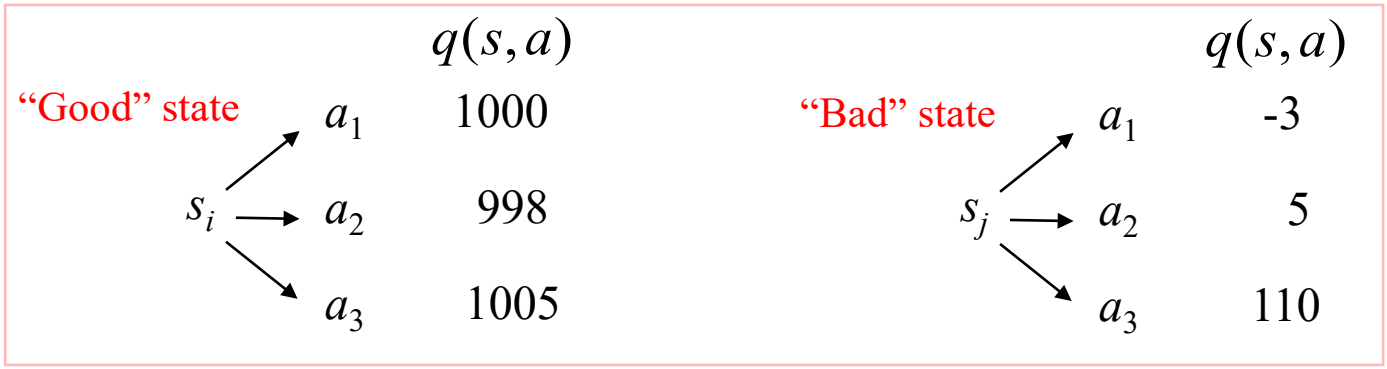
Discount factor

Different learning rates for critic and actor

Can AC be further improved?

What is better for learning:

- Turn around random actions of a good state? → marginal improvement
- Discover a great action in a bad state? → big improvement



Instead of using $\hat{q}(s,a,\mathbf{w})$ in AC, use an "advantage" measure to guide the algorithm towards the improvement.

AC with baseline

The policy gradient theorem can be generalized to include a comparison of the action value to an arbitrary baseline.

- Use a baseline function:

It cannot depend on *a* so the optimum policy is not modified

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E_{\pi_{\boldsymbol{\theta}}} \left\{ \left(\hat{q}(s,a,\mathbf{w}) - b(s) \right) \nabla_{\boldsymbol{\theta}} \log \pi \left(a \middle| s, \boldsymbol{\theta} \right) \right\}$$

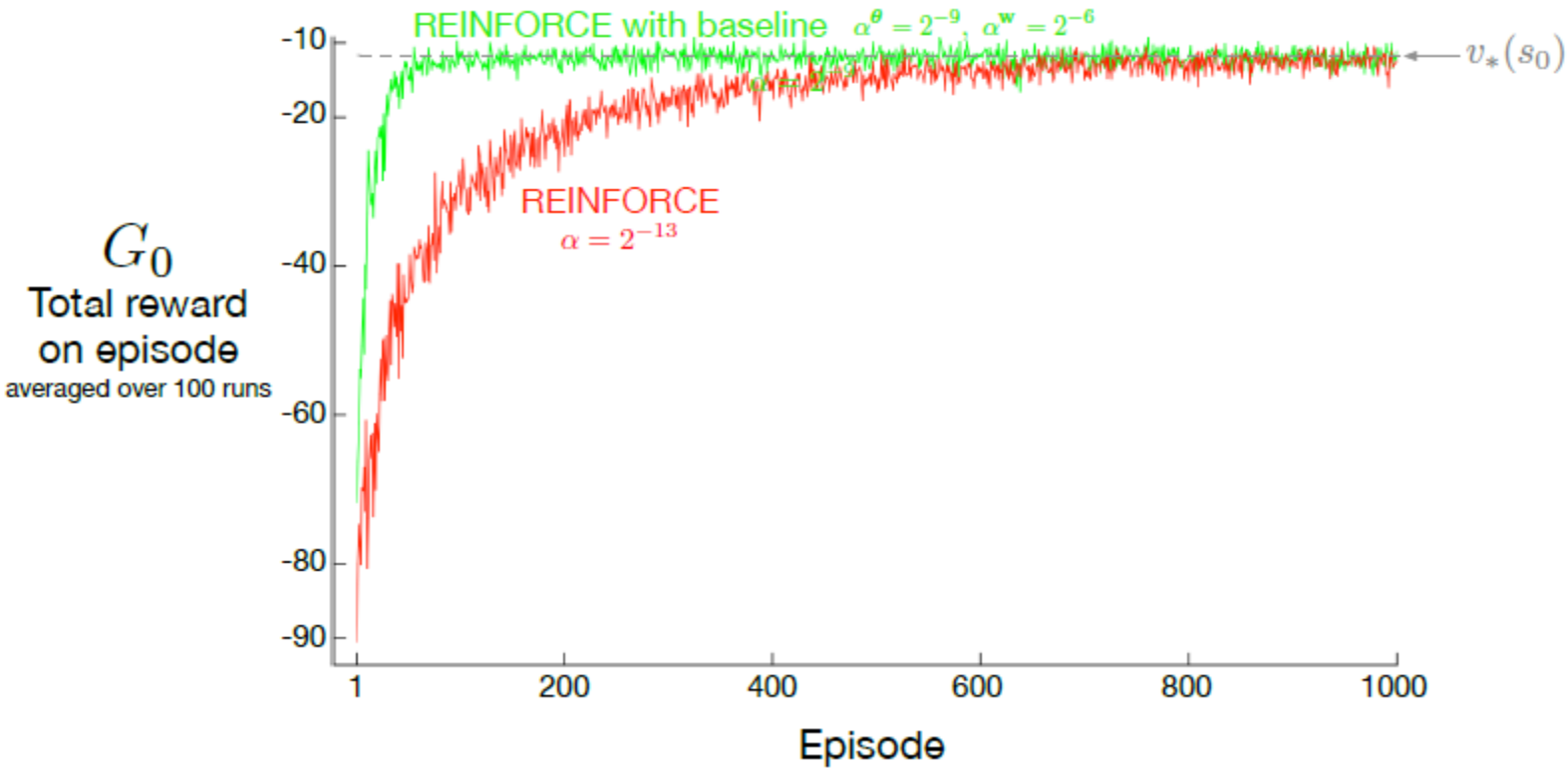
- One natural choice for the baseline is an estimate of the state value:

$$b(s) = \hat{v}(s,\mathbf{w})$$

The average performance of the state *s* over possible actions

Baseline speeds up convergence and reduces variance, as it guides the algorithm on the direction of more effective improvement.

If value function is poorly estimated, the algorithm performs at least as REINFORCE.



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

AC with baseline

Using $v(s)$ as the baseline, the advantage function can be defined as

$$A(s,a) = q(s,a) - v(s)$$
$$q(s,a) = r + \gamma v(s')$$
$$A(s,a) = r + \gamma v(s') - v(s)$$

An unbiased estimate of $A(s,a)$ is the TD error: $\delta = r + \gamma v_{\pi_\theta}(s') - v_{\pi_\theta}(s)$

$$E_{\pi_\theta}\left\{\delta \middle| s,a\right\} = E_{\pi_\theta}\left\{r + \gamma v_{\pi_\theta}(s') \middle| s,a\right\} - v_{\pi_\theta}(s)$$
$$= q_{\pi_\theta}(s,a) - v_{\pi_\theta}(s) = A(s,a)$$

and $v(s)$ can be estimated by averaging the approximated q-function $\hat{q}(s,a,\mathbf{w})$

$$\hat{v}(s) = \sum_{a \in \mathcal{A}} \pi\left(a \middle| s, \boldsymbol{\theta}\right) \hat{q}(s,a,\mathbf{w})$$

Actor-critic with baseline

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$
Repeat forever:
  Initialize S (first state of episode), sample $A \sim \pi(a|s,\boldsymbol{\theta})$
  $I \leftarrow 1$
  While S is not terminal:
    Observe R, S'
    Sample $A' \sim \pi(a|S',\boldsymbol{\theta})$
    $\delta \leftarrow R + \gamma \hat{q}(S',A',\mathbf{w}) - \hat{q}(S,A,\mathbf{w})$ (if S' is terminal, then $\hat{q}(S',A,\mathbf{w}) \doteq 0$)
Advantage → $\beta \leftarrow R + \sum_{a \in \mathcal{A}} \left(\gamma \pi\left(a|S',\boldsymbol{\theta}\right) \hat{q}(S',a,\mathbf{w}) - \pi\left(a|S,\boldsymbol{\theta}\right) \hat{q}(S,a,\mathbf{w}) \right)$
Critic → $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla_{\mathbf{w}} \hat{q}(S,A,\mathbf{w})$
Actor → $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \beta \nabla_{\boldsymbol{\theta}} \ln \pi\left(A|S,\boldsymbol{\theta}\right)$
    Gradient updates for the parameters
    $I \leftarrow \gamma I$
    $S \leftarrow S'$
    $A \leftarrow A'$

Discount factor

Different learning rates for critic and actor

Summary

Value-based	Policy-based + Actor-critic
Q-learning, SARSA, value-iteration	REINFORCE, Baseline Actor-Critic methods

What are the differences?

Value-based	Policy-based + Actor-critic
Harder problem	Easier solution
Explicit exploration, using ε-greedy	Innate exploration and stochasticicy (non deterministic policies)
Evaluates states and actions	Easier for random policies and continuous actions
Easier to train off-policy	Can accommodate supervised learning

Popular implementations of AC use deep neural network agents. Next chapter!