Reinforcement Learning - Lecture 6

Profa. Dra. Esther Luna Colombini
esther@ic.unicamp.br

Prof. Dr. Alexandre Simoes
alexandre.simoes@unesp.br

LaRoCS – Laboratory of Robotics and Cognitive Systems

UNICAMP

Advanced Institute for Artificial Intelligence

Summary

- Value-function approximation for prediction
  - MC
  - TD
- Value-function approximation for control
  - MC
  - TD

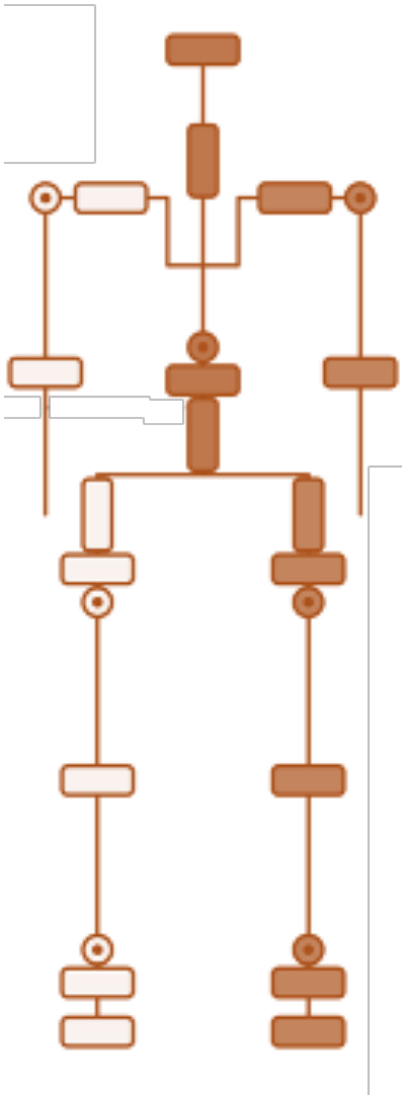
Disclaimer

- These slides were built upon David Silver's Lecture notes on Reinforcement Learning

- How to scale-up tabular RL to deal with real problems?
  - Function approximation
- Remember the elements RL are concerned
  - Optimization
  - Delayed consequences
  - Exploration
  - **Generalization**

Large-scale RL problems

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter: continuous state space
  - Robot control: continuous state space
    - If we decide to discretize
      - $2{,}5*10^{180}$ (25 joints discretized each 1o)
- How can we scale up the model-free methods for prediction and control from the last lectures?



Large-scale RL problems

- We do not want to explicitly store or learn for every single state a
  - Dynamics or reward model
  - Value
  - State-action value
  - Policy
- Want more compact representation that generalizes across state or states and actions
- Benefits
  - Reduces memory needed to store (P, R) / V / Q / π
  - Reduces computation needed to compute (P, R) / V / Q / π
  - Reduces experience needed to find a good P, R / V / Q / π

Value-function function approximation

- The methods studied so far have represented the value function by a lookup table
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation

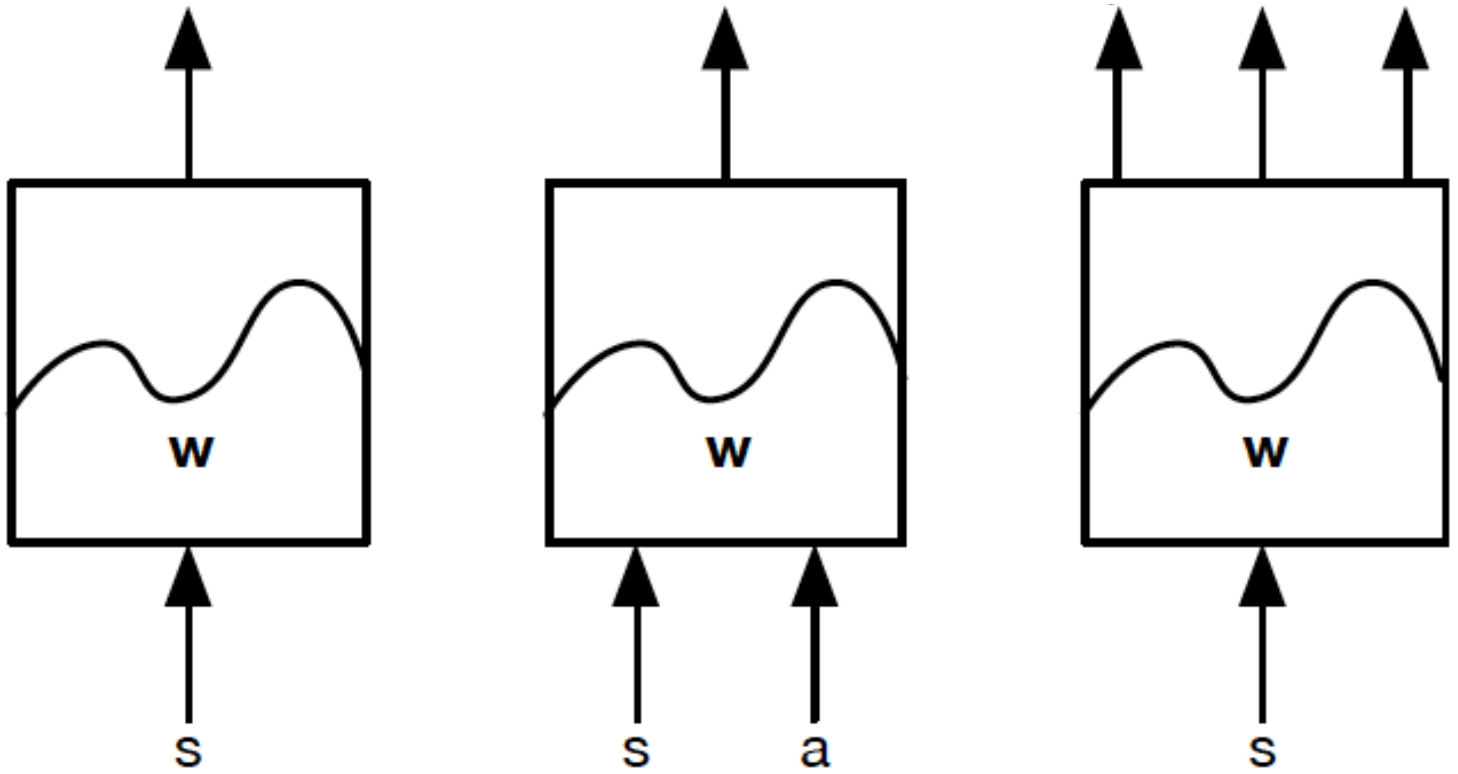
$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s), \text{ or}$$
$$\hat{q}(s, \mathbf{a}, \mathbf{w}) \approx q_\pi(s, a)$$

  - Generalize from seen states to unseen states
  - Update parameter $\mathbf{w}$ using MC or TD learning

Value-function function approximation types

- Represent a (state-action/state) value function with a parameterized function instead of a table

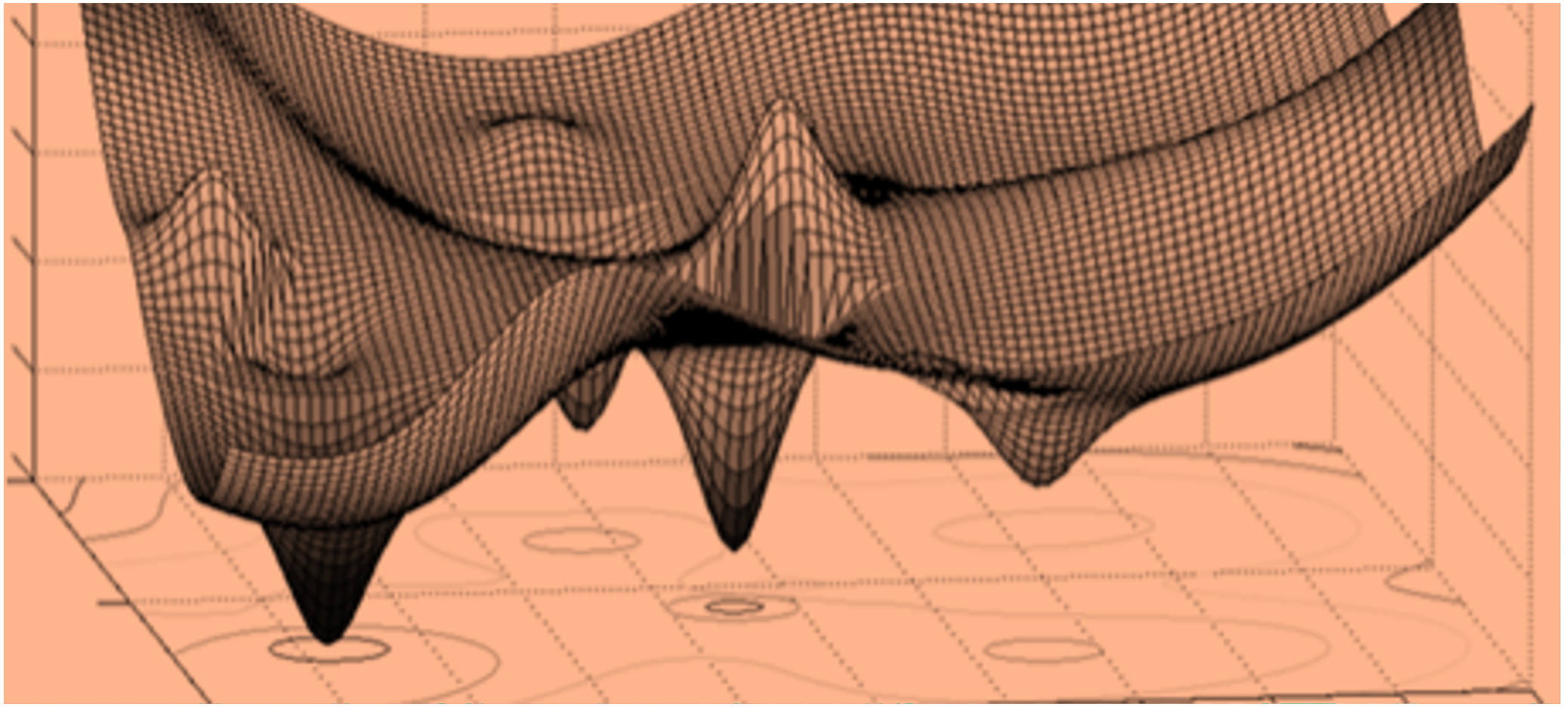
$\hat{v}(s, \mathbf{w})$ $\hat{q}(s, a, \mathbf{w})$ $\hat{q}(s, a_1, \mathbf{w})$ ... $\hat{q}(s, an, \mathbf{w})$



- Which function approximator?

Which Function Approximator?

- Many possible function approximators including
  - **Linear combination of features**
  - **Neural networks**
  - Decision trees
  - Nearest neighbors
  - Fourier/ wavelet bases
  - …
- But, we will consider differentiable function approximators
- Furthermore, we require a training method that is suitable for **non-stationary, non-iid** data



Incremental Methods

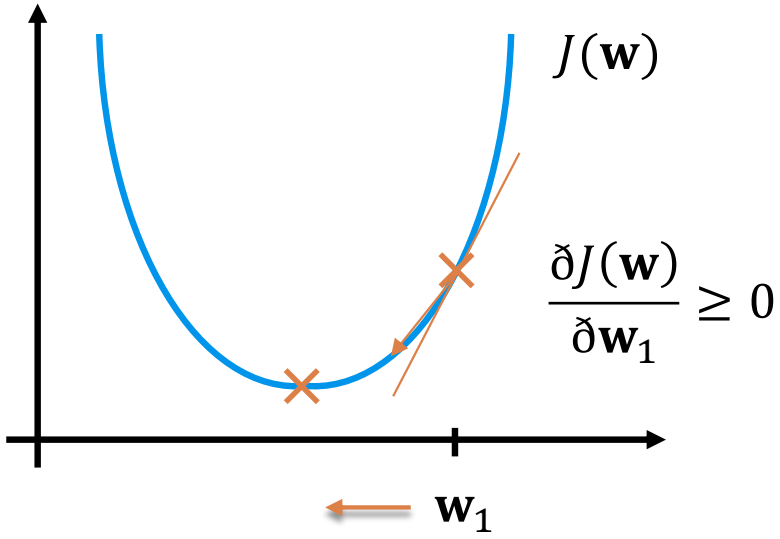
- The **gradient** is another name for derivative, or the rate of change of a **function**
  - It is a vector pointing in the direction of greatest increase of a **function**
    - Is zero at a local maximum or local minimum
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the gradient of $J(\mathbf{w})$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

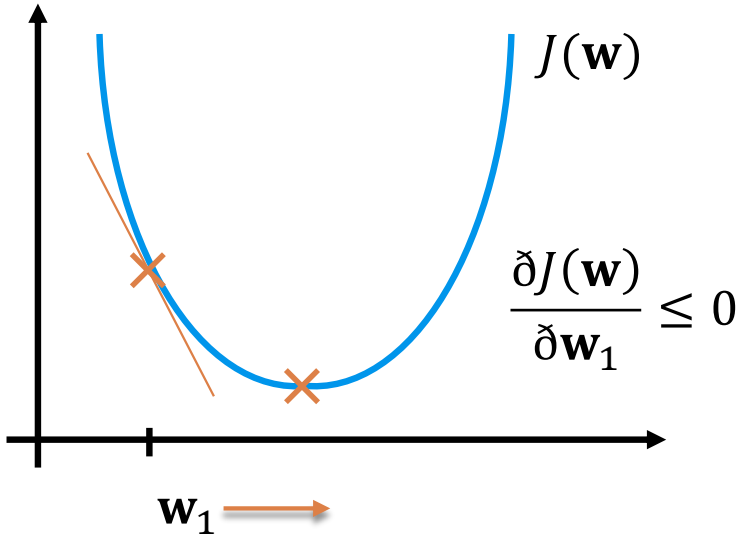
- To find a local minimum of $J(\mathbf{w})$
- Adjust $\mathbf{w}$ in direction opposite do gradient

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

- Where α is the learning rate



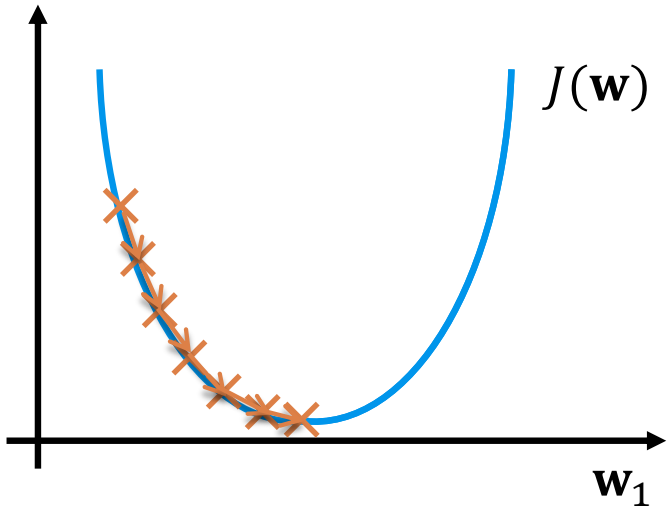
$\mathbf{w}_1 = \mathbf{w}_1 - \alpha.(positive\ number)$



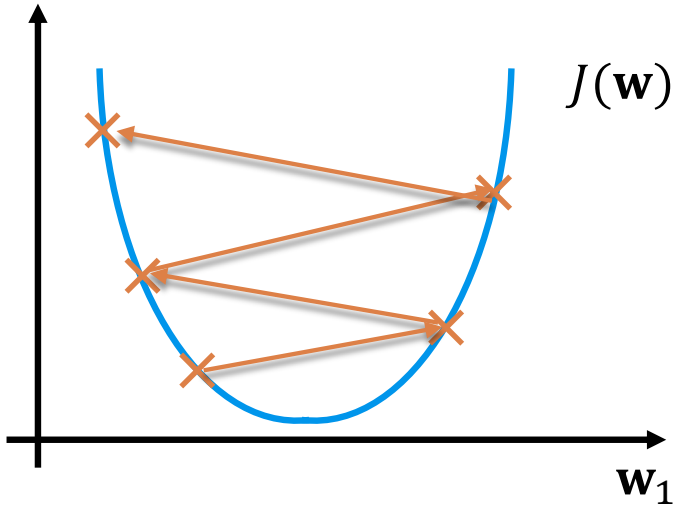
$\mathbf{w}_1 = \mathbf{w}_1 - \alpha.(negative\ number)$

Gradient Descent

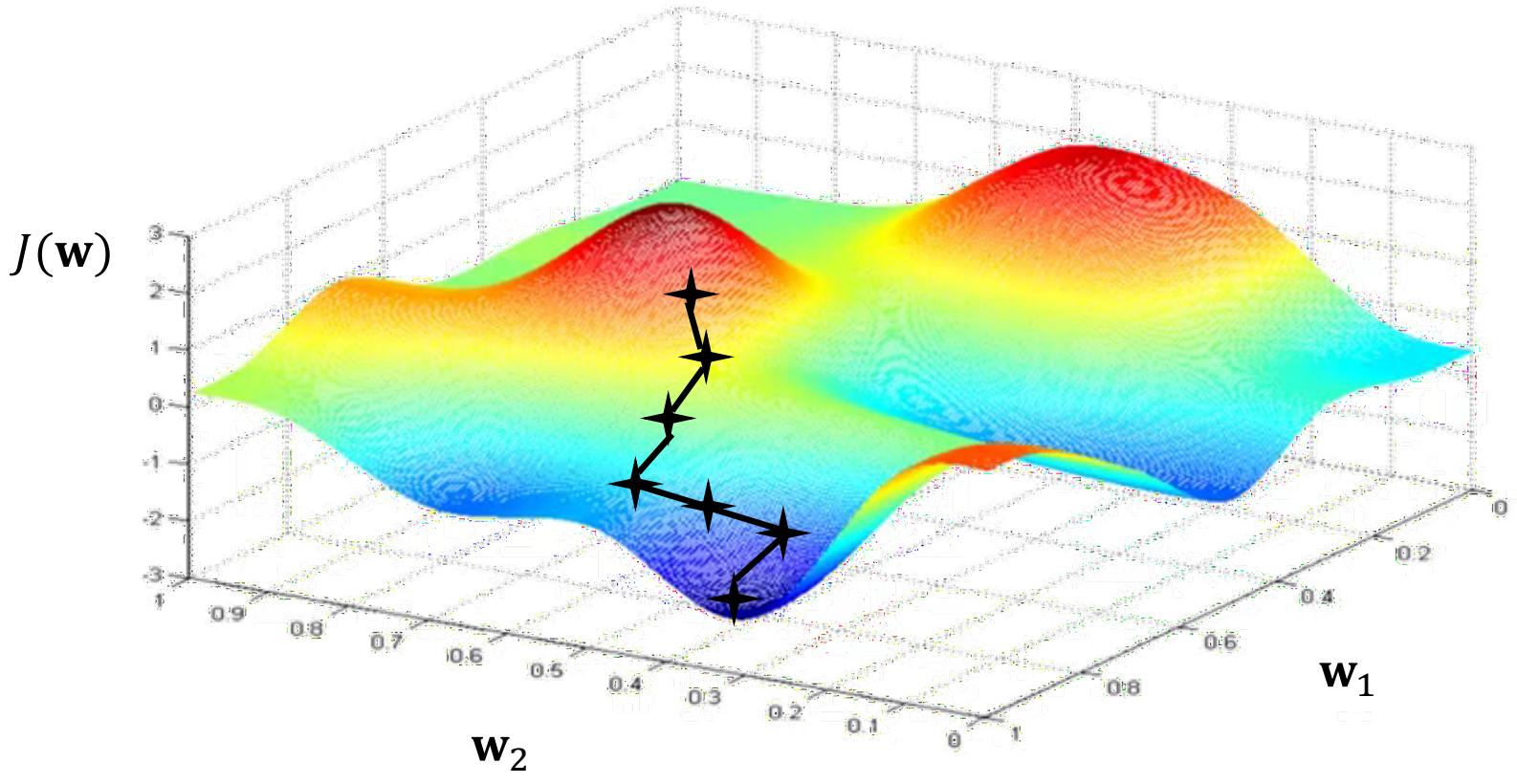
- Be careful with α
  - If α is too small, gradient descent can be slow



  - If α is too large, gradient descent can diverge



Gradient Descent



Value Function Approximation by SGD

- Goal: find parameter vector w minimizing mean-squared error between approximate value function $\hat{v}(s, \mathbf{w})$ and true value $v_\pi(s)$ over some distribution of the states

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(S) - \hat{v}(S, \mathbf{w})\right)^2\right]$$

- Gradient descent finds a local minimum

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

$$= \alpha \mathbb{E}_\pi\left[\left(v_\pi(S) - \hat{v}(S, \mathbf{w})\right)\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})\right]$$

- Stochastic gradient descent samples the gradient

$$\Delta \mathbf{w} = \alpha \left[\left(v_\pi(S) - \hat{v}(S, \mathbf{w})\right)\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})\right]$$

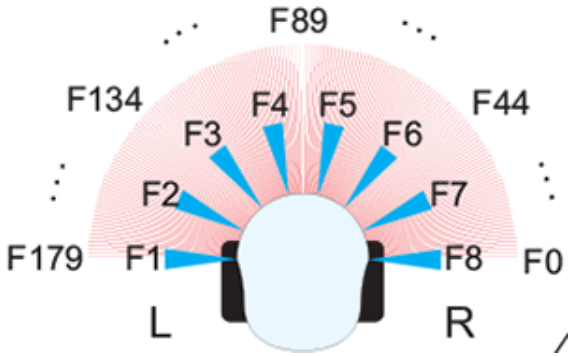
- Expected update is equal to full gradient update

Feature Vectors

- Represent state by a feature vector

$$x(S) = \begin{pmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{pmatrix}$$

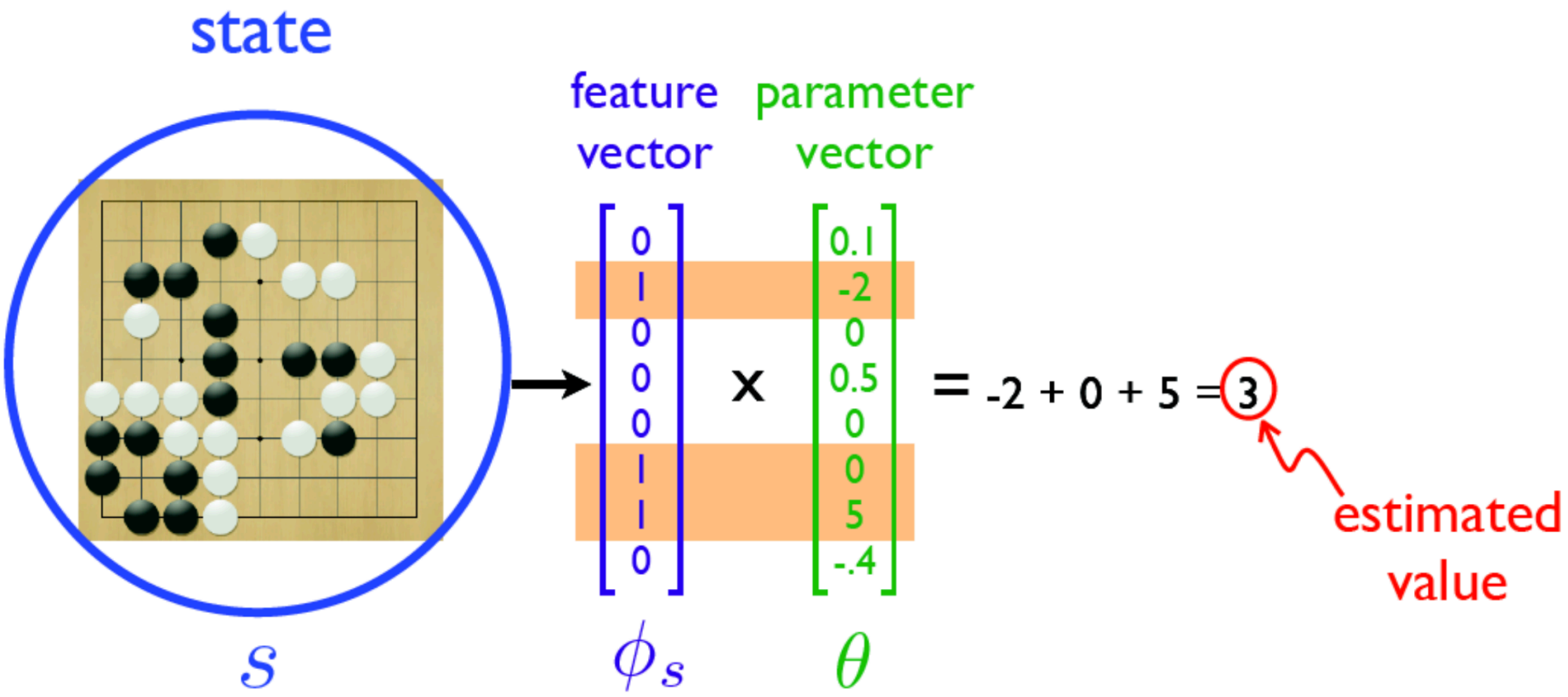
- Examples:
  - Distance of robot from landmarks
  - Trends in the stock market
    Piece and pawn configurations in chess



$$x_1(S) = \begin{pmatrix} F0 \\ \vdots \\ F179 \end{pmatrix} \qquad x_2(S) = \begin{pmatrix} Obstacle\ Right \\ \vdots \\ No\ Obstacle \end{pmatrix}$$

Feature Vectors

Sutton, presentation at ICML 2009



(10^{35} states, 10^5 binary features and parameters.)

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = x(S)^T \mathbf{w} = \sum_{j=1}^{n} \boldsymbol{x_j}(S) \boldsymbol{w_j}$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = \mathbb{E}_\pi[(v_\pi(S) - x(S)^T \mathbf{w})^2]$$

- Stochastic gradient descent converges on global optimum
- Update rule is particularly simple (step-size x prediction error × feature value)

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

$$\Delta \mathbf{w} = \alpha \left[\left(v_\pi(S) - \hat{v}(S, \mathbf{w}) \right) x(S) \right]$$

Table Lookup Features

- Table lookup is a special case of linear value function approximation
- Using table lookup features

$$\mathrm{x}^{\mathrm{table}}(\mathrm{S}) = \begin{pmatrix} \mathbf{1}(S = \mathrm{x}_1) \\ \vdots \\ \mathbf{1}(S = \mathrm{x}_n) \end{pmatrix}$$

- Parameter vector $\mathbf{w}$ gives value of each individual state

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = \mathrm{s}_1) \\ \vdots \\ \mathbf{1}(S = \mathrm{s}_n) \end{pmatrix} \cdot \begin{pmatrix} \mathrm{w}_1 \\ \vdots \\ \mathrm{w}_n \end{pmatrix}$$

Incremental Prediction Algorithms

- Recall model-free policy evaluation
  - Following a fixed policy π (or had access to prior data)
  - Goal is to estimate v_π and/or q_π
- Maintained a lookup table to store estimates v_π and/or q_π
- Updated these estimates after each episode (Monte Carlo methods) or after each step (TD methods)
- Now: in value function approximation, change the estimate update step to include **fitting the function approximator**

- We assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a target for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{G_t} - \hat{v}(S_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \right]$$

  - For TD(0), the target is the TD target $\boldsymbol{R_{t+1}} + \boldsymbol{\gamma \hat{v}(S_{t+1}, \mathbf{w})}$

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{R_{t+1}} + \boldsymbol{\gamma \hat{v}(S_{t+1}, \mathbf{w})} - \hat{v}(S_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \right]$$

  - For TD(λ), the target is the λ-return G_t^λ

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{G_t^\lambda} - \hat{v}(S_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \right]$$

- Return G_t is an unbiased but noisy sample of the true expected return $v_\pi(st)$
- Therefore, can reduce MC VFA to doing supervised learning on a set of (state,return) pairs:

$$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, \ldots, \langle S_T, G_T \rangle$$

- Use G_t for the true $v_\pi(s)$ when fit function approximator
- Concretely when using linear VFA for policy evaluation

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{G_t} - \hat{v}(S_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \right]$$

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{G_t} - \hat{v}(S, \mathbf{w}) \right) x(S) \right]$$

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

Monte Carlo Value Function Approximation

Algorithm: MC Linear Value Function Approximation for Policy Evaluation

1: $\mathbf{w}=0,, k = 1$

2: ***repeat***

3: Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,T})$ given π

4: **for** $(t=1,\ldots, L_k)$ **do**

5: **if** first visit to s in episode k then

6: $G_t(s) = \sum_{j=t}^{L_k} r_{k,j}$

7: *Update weights:*

8: $\mathbf{w} = \mathbf{w} + \alpha\,[(G_t - x(S)^T\mathbf{w})x(S)]$

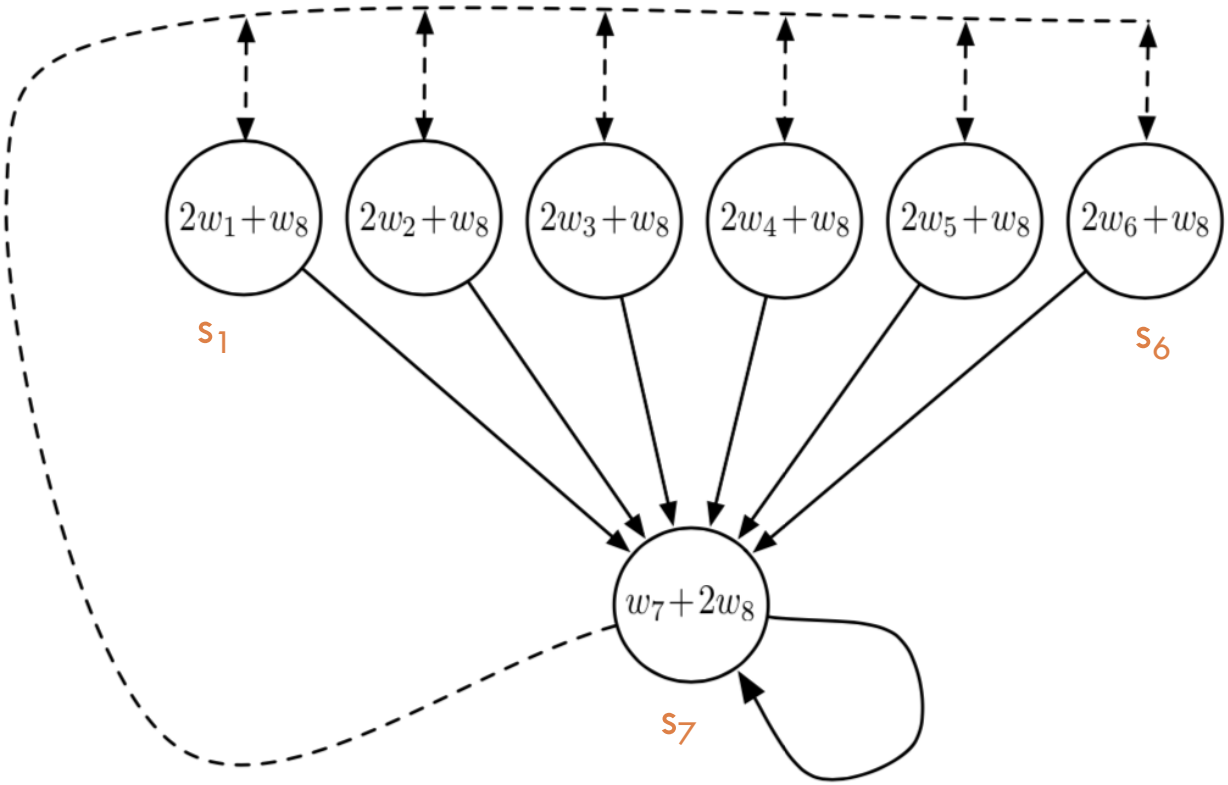
9: $k=k+1$

10: **until last episode**

Baird (1995)-Like Example with MC Policy Evaluation

s_1 =[2 0 0 0 0 0 0 1]
s_2 =[0 2 0 0 0 0 0 1]
s_3 =[0 0 2 0 0 0 0 1]
…
s_7 =[0 0 0 0 0 0 1 2]

w_0 =[1 1 1 1 1 1 1 1]
a_1 _____ lines
a_2 -------- lines
Rewards 0



- Small prob s_7 goes to terminal state, $x(s_7)^T$ =[0 0 0 0 0 0 1 2]
- Let's consider the sequence: $s_1, a_1, 0, s_7, a_1, 0, s_7, a_1, 0, T$
- $G(s_1)=0$
- $w=w+\alpha[0- x(s_1)^T\mathbf{w}]x(s_1)$ as, $x(s_1)\mathbf{w}=2+1=3$
- w= [1 1 1 1 1 1 1 1] +0.5[0-3][2 0 0 0 0 0 0 1]
- w = [1 1 1 1 1 1 1 1] -1.5[2 0 0 0 0 0 0 1] = [-2 1 1 1 1 1 1 -0.5]

- Uses bootstrapping and sampling to approximate true v_π
- The TD-target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$ is a biased sample of true value $v_\pi(S_t)$
- Can still apply supervised learning to "training data":

$$\langle S_1, R_2 +, \hat{v}(S_2, \boldsymbol{w}) \rangle, \langle S_2, R_3 +, \hat{v}(S_3, \boldsymbol{w}) \rangle, \ldots, \langle S_{T-1}, R_T \rangle$$

- For example, using linear TD(0)

$$\Delta \mathbf{w} = \alpha \left[(R \quad + \gamma \hat{v}(S', \boldsymbol{w}) - \hat{v}(S \quad, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S \quad, \mathbf{w}) \right]$$
$$= \alpha \delta \, \mathbf{x}(S)$$

- Linear TD(0) converges (close) to global optimum

TD(0) Linear VFA for Policy Evaluation

Algorithm: TD(0) Linear Value Function Approximation for Policy Evaluation

1: $\mathbf{w}=0,, k = 1$

2: ***repeat***

3: Sample tuple (s_k, a_k, r_k, s_{k+1}) given π

4: *Update weights:*

5: $\mathbf{w} = \mathbf{w} + \alpha\,[(r \;\; + \gamma x(S')^T\mathbf{w} - x(S)^T\mathbf{w})]x(S)$

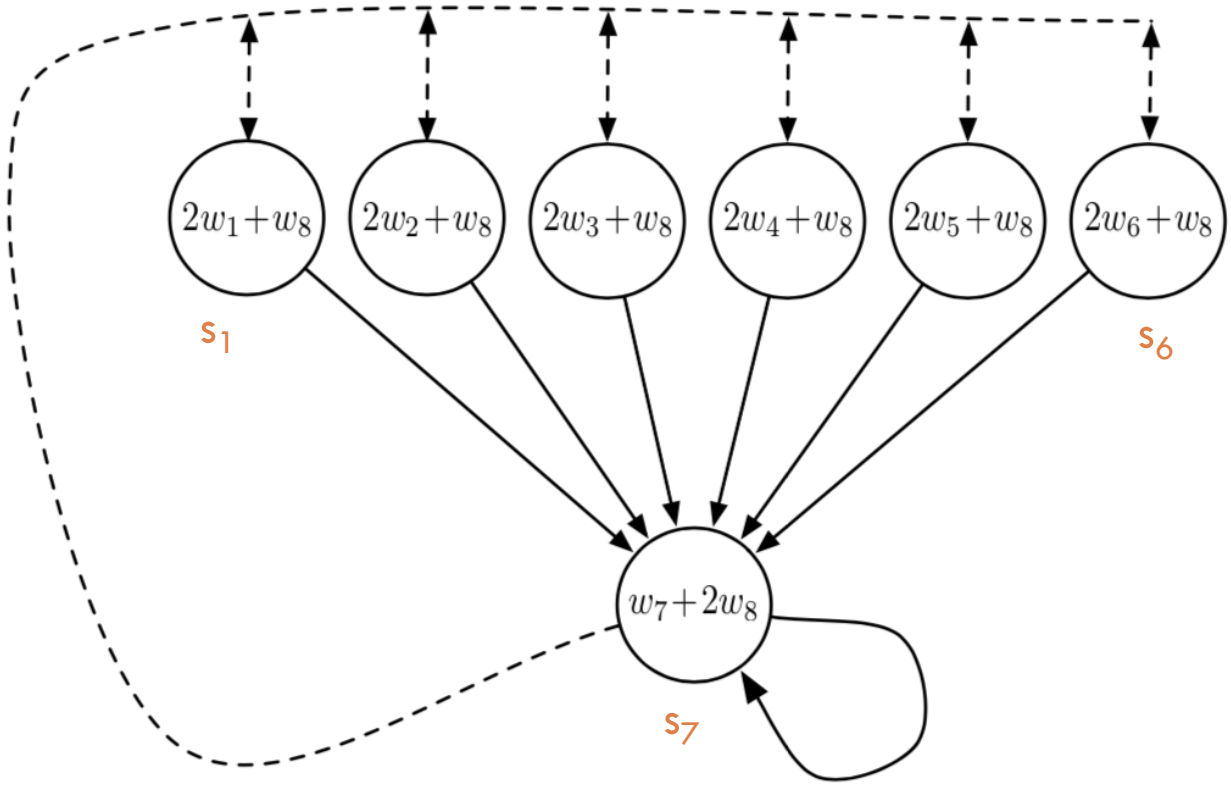
6: k=k+1

7: ***end***

Baird (1995)-Like Example with TD(0)

s_1 =[2 0 0 0 0 0 0 1]
s_2 =[0 2 0 0 0 0 0 1]
s_3 =[0 0 2 0 0 0 0 1]
…
s_7 =[0 0 0 0 0 0 1 2]

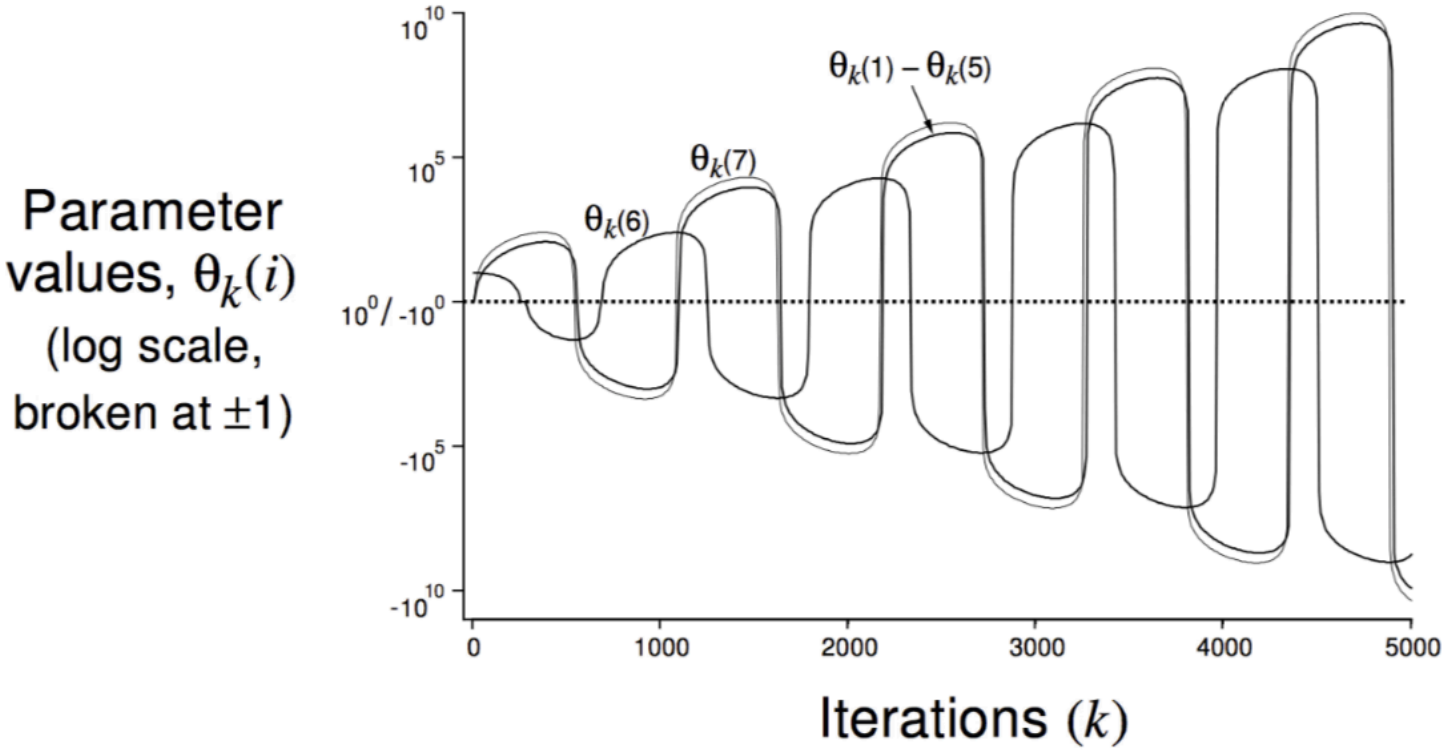
w_0 =[1 1 1 1 1 1 1 1]
a_1 _____ lines
a_2 -------- lines
Rewards 0
$\gamma = 0.9$



- Small prob s_7 goes to terminal state, $x(s_7)^T$ =[0 0 0 0 0 0 1 2]
- Let's consider the sequence: $s_1, a_1, 0, s_7$
- $w = w + \alpha[0 + \gamma x(s_7)^T \mathbf{w} - x(s_1)^T \mathbf{w}]x(s_1)$ as, $x(s_1)^T \mathbf{w}=2+1=3$ and $x(s_7)^T \mathbf{w}=3$
- $w = w + 0.5[0+0.9.3-3]\ x(s_1)$ -> [1 1 1 1 1 1 1 1] - 0.15. [2 0 0 0 0 0 0 1]
- w = [0.7 1 1 1 1 1 1 0.85]

Parameter Divergence in Baird's Counterexample

- If P(s) is uniform (≠from the true stationary distribution of the Markov chain), then the asymptotic bahaviour becomes unstable



TD(λ) with Value Function Approximation

- The target return G_t^λ is also a biased sample of true value v_π
- Can again apply supervised learning to "training data":

$$\langle S_1, G_1^\lambda \rangle, \langle S_2, G_2^\lambda \rangle, \ldots, \langle S_{T-1}, G_{T-1}^\lambda \rangle$$

- Forward view linear TD(λ)

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{G_t^\lambda} - \hat{v}(S_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \right]$$

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{G_t} - \hat{v}(S_t, \mathbf{w}) \right) x(S_t) \right]$$

- Backward view linear TD(λ)

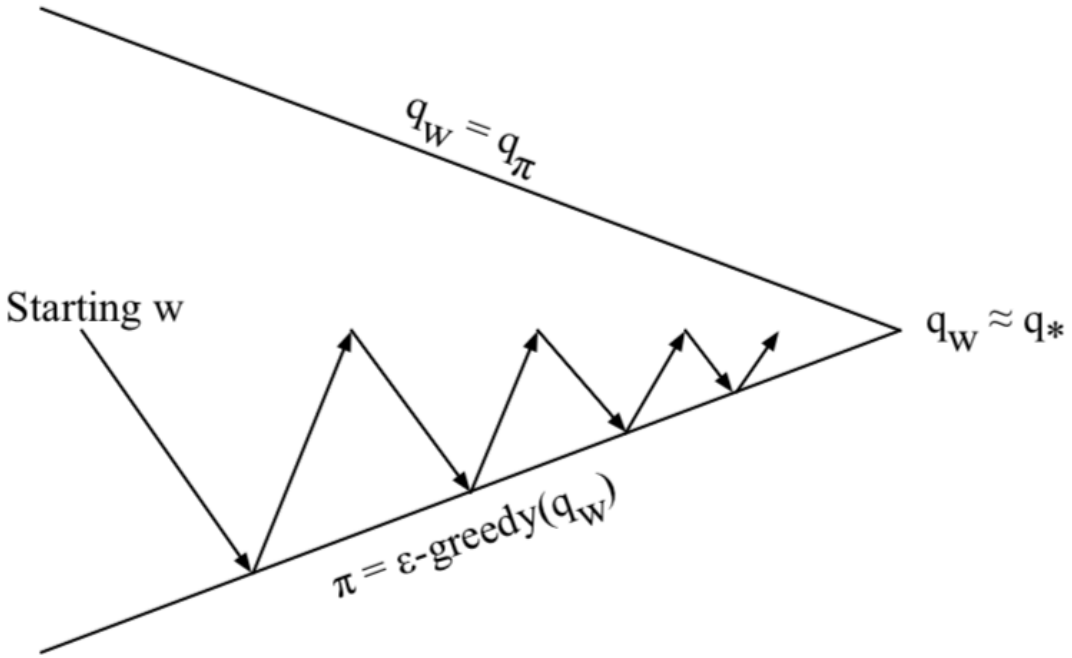
$$\delta_t = (R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \mathbf{w}))$$

$$E_t = \gamma \lambda \, E_{t-1} + x(S_t)$$

$$\Delta \mathbf{w} = \alpha \delta_t E_t$$

- Forward view and backward view linear TD(λ) are equivalent

Control with Value Function Approximation



- Policy evaluation
  - Approximate Monte-Carlo policy evaluation, $\hat{q}(\cdot,\cdot,\mathbf{w}) \approx q_\pi$
- Policy improvement
  - ϵ -greedy policy improvement

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(\mathrm{S}, \mathrm{A}, \mathbf{w}) \approx q_\pi(\mathrm{S}, \mathrm{A})$$

- Minimize mean-squared error between approximate action-value $\hat{q}(\mathrm{S}, \mathrm{A}, \mathbf{w})$ and true action-value $q_\pi(\mathrm{S}, \mathrm{A})$

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})\right)^2\right]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w}) = \left(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})\right) \nabla_{\mathbf{w}}\, \hat{q}(S, A, \mathbf{w})$$

$$\Delta\mathbf{w} = \alpha\left(\left(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})\right) \nabla_{\mathbf{w}}\, \hat{q}(S, A, \mathbf{w})\right)$$

Linear Action-Value Function Approximation

- Represent state and action by a feature vector

$$\mathrm{x}(\mathrm{S},\mathrm{A}) = \begin{pmatrix} \mathrm{x}_1(S,A) \\ \vdots \\ \mathrm{x}_n(S,A) \end{pmatrix}$$

- Represent action-value fn by linear combination of features

$$\hat{q}(\mathrm{S},\mathrm{A},\mathbf{w}) = \mathrm{x}(\mathrm{S},\mathrm{A})^{\intercal}\,\mathbf{w} = \sum_{j=1}^{n} \boldsymbol{x_j}(S,A)\boldsymbol{w_j}$$

- Stochastic gradient descent update

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

$$\Delta\mathbf{w} = \alpha((q_\pi(S,A) - \hat{q}(S,A,\mathbf{w}))\;\mathrm{x}(\mathrm{S},\mathrm{A})$$

Incremental Control Algorithms

- Like prediction, we must substitute a target for $q_\pi(S, A)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{G_t} - \hat{q}(S_t, A_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}) \right]$$

  - For SARSA, the target is the TD target $\boldsymbol{R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w})}$

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w})} - \hat{q}(S_t, A_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}) \right]$$

  - For Q-learning instead use a TD target $\boldsymbol{R_{t+1} + \gamma max_{a'} \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w})}$ which leverages the max of the current function approximation value

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{R_{t+1} + \gamma max_{a'} \hat{q}(S_{t+1}, a', \mathbf{w})} - \hat{q}(S_t, A_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}) \right]$$

  - For forward-view TD(λ), the target is the action-value λ-return

$$\Delta \mathbf{w} = \alpha \left[\left(\boldsymbol{q_t^\lambda} - \hat{q}(S_t, A_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}) \right]$$

  - For backward-view TD(λ), equivalent update is

$$\delta_t = \left(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{q}(S_t, A_t, \mathbf{w}) \right)$$

$$E_t = \gamma \lambda \, E_{t-1} + x(S_t, A_t)$$

$$\Delta \mathbf{w} = \alpha \delta_t E_t$$

SARSA Algorithm

Algorithm: SARSA (on-policy TD control with VFA)

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize value-function weights $\mathbf{w}$ arbitrarily

repeat (for each episode)

    Initialize S,A

    repeat (for each step of episode)

        Take action A, observe R, S'

        if ***S'****is terminal*

            $\boldsymbol{w} = \boldsymbol{w} + \alpha \left[\left(R + \boldsymbol{\gamma \hat{q}}(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{w})\right) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})\right]$

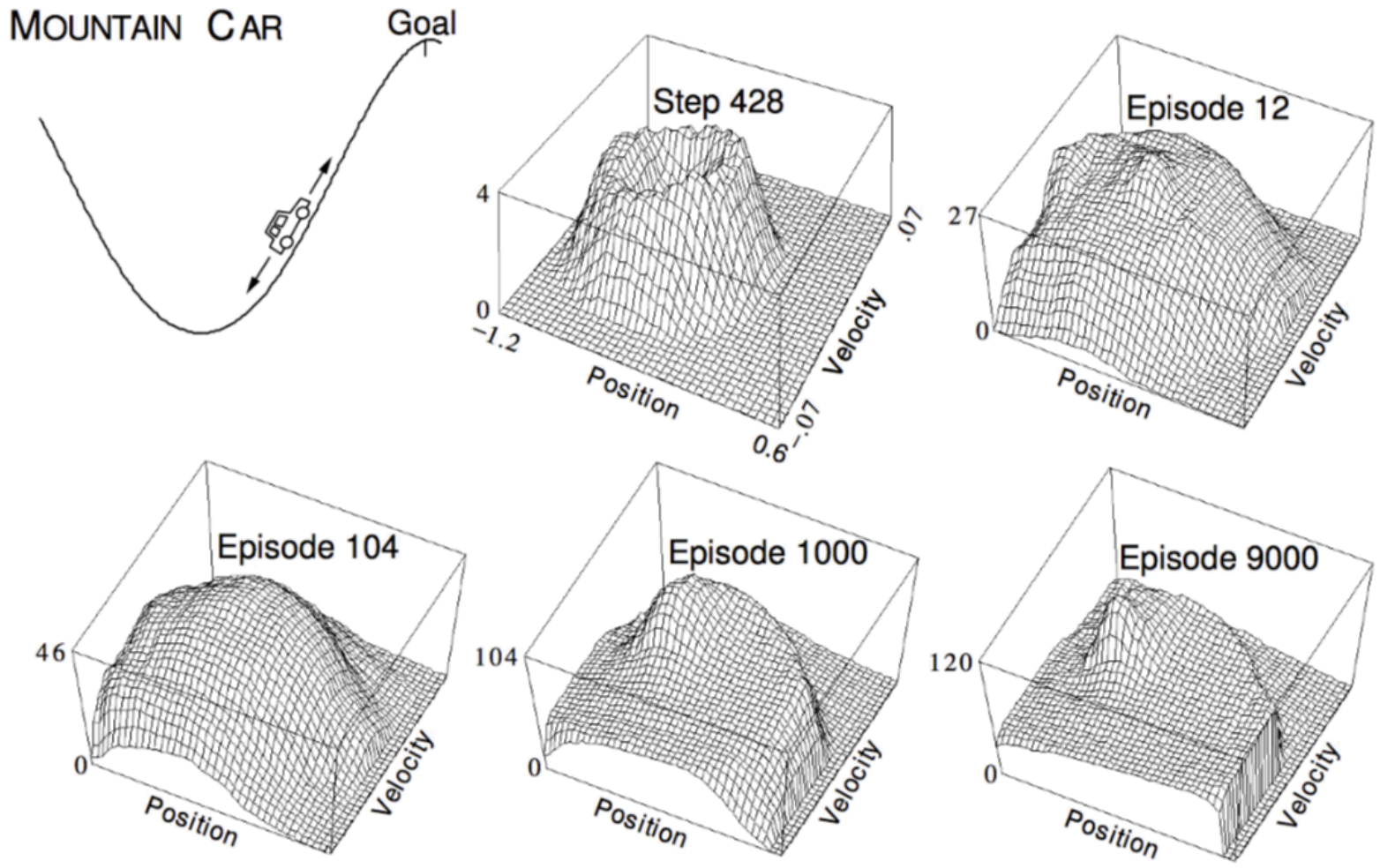
            Go to next expisode

        Choose A' as a function derived from $\hat{q}(S', \cdot, \mathbf{w})$ (e.g. ***ϵ-greedy)***

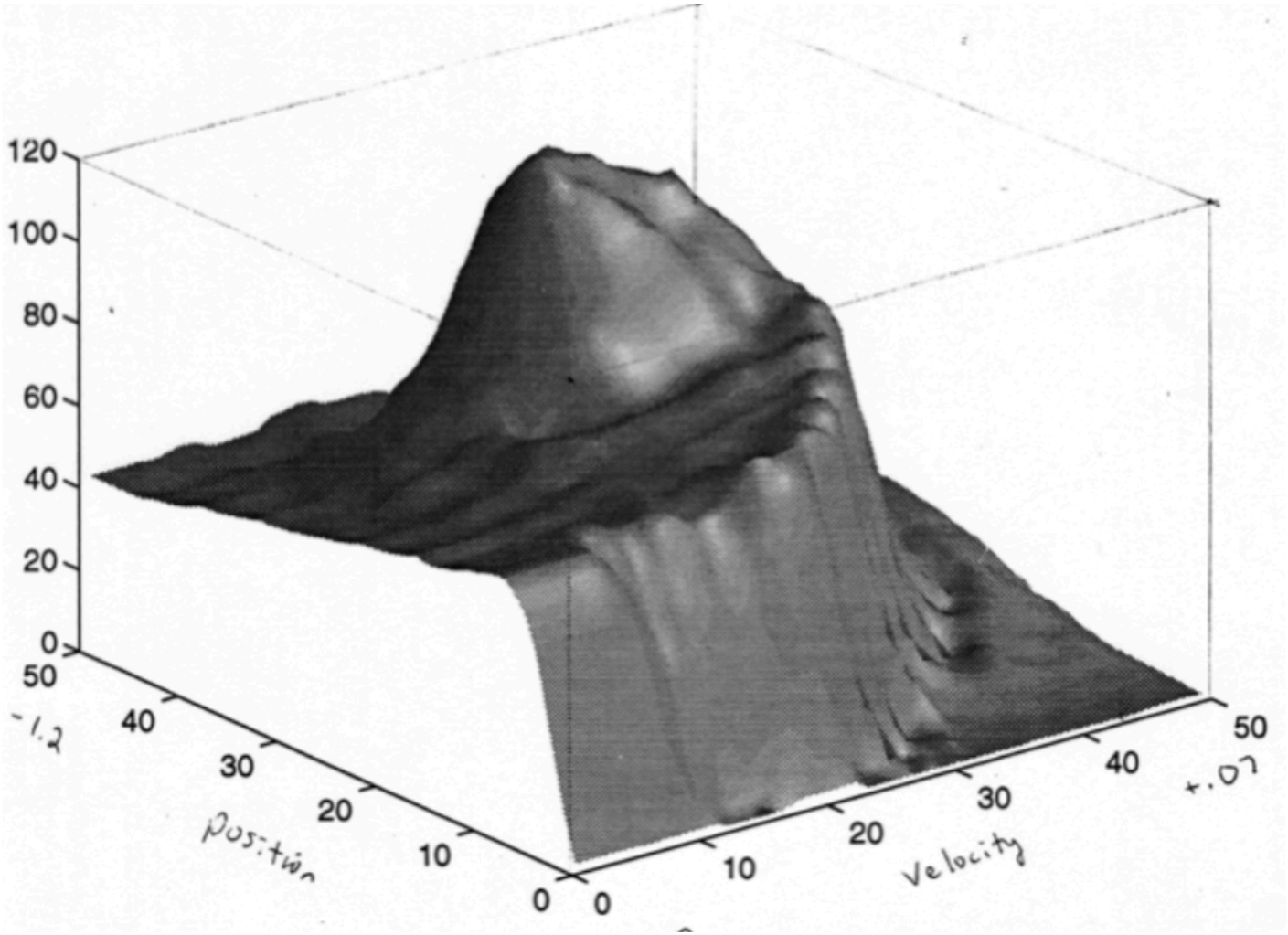
        $\boldsymbol{w} = \boldsymbol{w} + \alpha \left[\left(R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\right) \nabla_{w} \hat{q}(S, A, \boldsymbol{w})\right]$

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

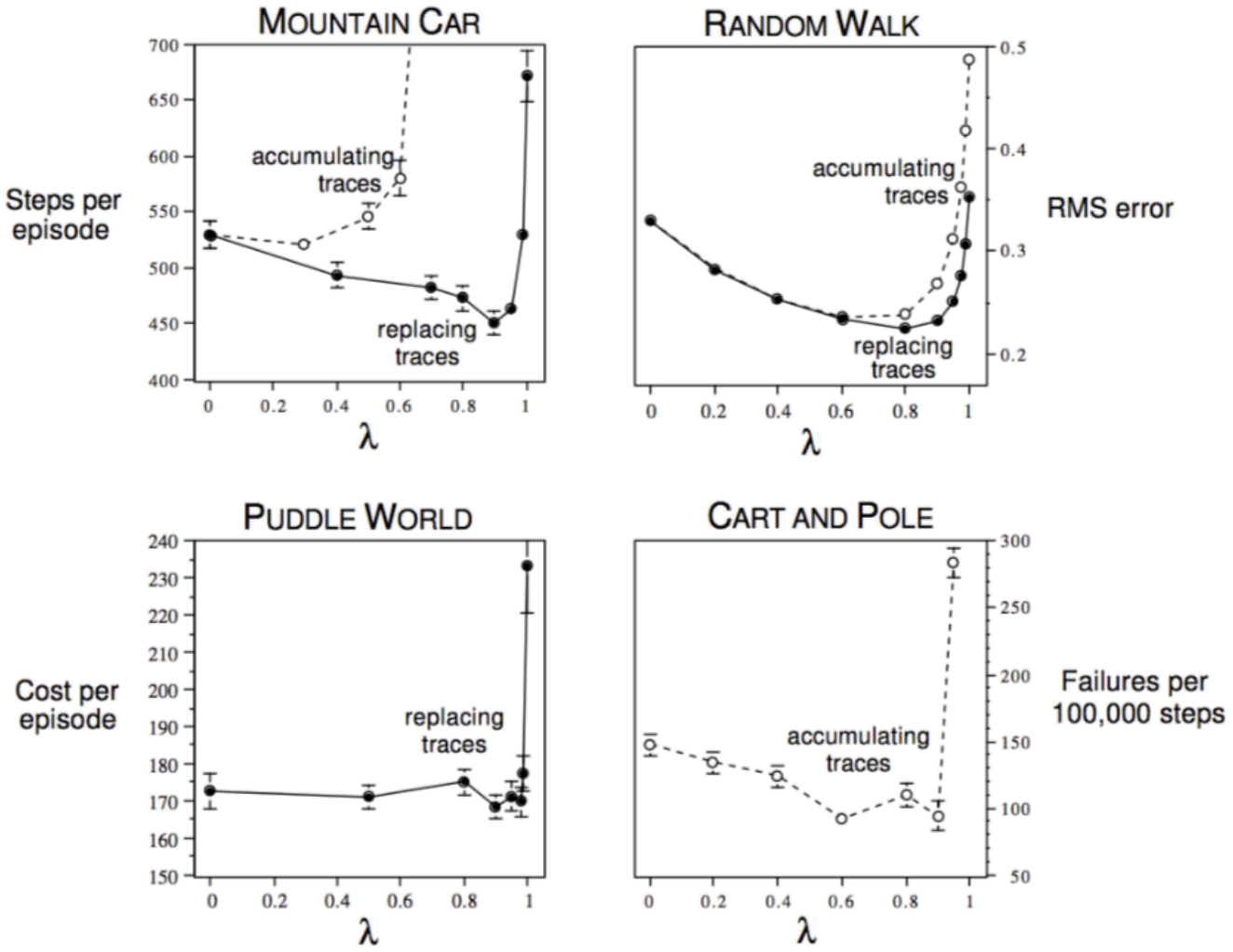
Linear Sarsa with Coarse Coding in Mountain Car



Linear Sarsa with Radial Basis Functions in Mountain Car



Study of λ: Should We Bootstrap?



Convergence of Prediction Algorithms

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Convergence of Prediction Algorithms

- TD does not follow the gradient of any objective function
- This is why TD can diverge when off-policy or using non-linear function approximation
- Gradient TD follows true gradient of projected Bellman error

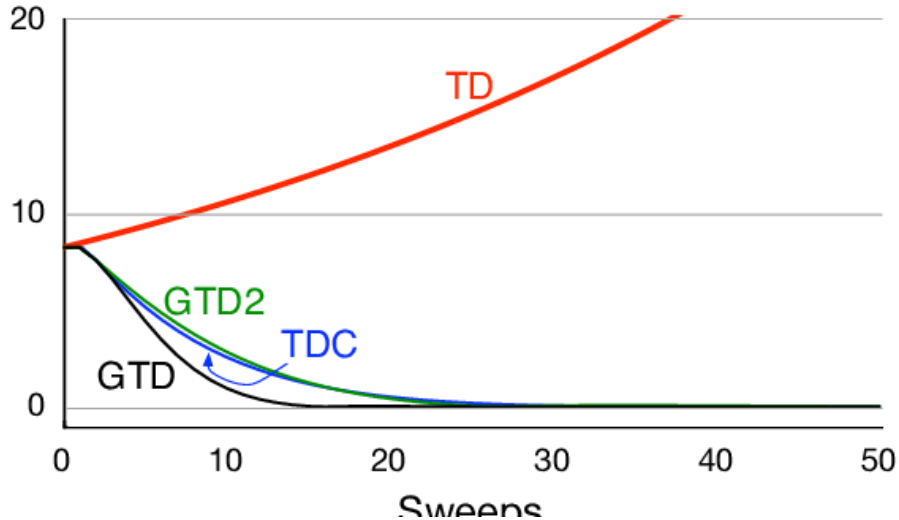
On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD	✓	✓	✗
	Gradient TD	✓	✓	✓
Off-Policy	MC	✓	✓	✓
	TD	✓	✗	✗
	Gradient TD	✓	✓	✓

Convergence of Control Algorithms

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

(✓) = chatters around near-optimal value function

- Gradient temporal difference learning
  - GTD (gradient temporal difference learning)
    - $\mathbf{w}_k$ is a new set of weights whose purpose is to estimate $\mathbf{w}$ for a fixed value of the θ parameter
    - The parameter vector θ is updated on a "slower" timescale
      - $\theta_{t+1} = \theta_t + \alpha_t(x(S_t) - \gamma x(S_{t+1}))$ $x(S_t)^\intercal \mathbf{w}_t \pi(s_t, a_t)$
      - $\mathbf{w}_{t+1} = \mathbf{w}_t + \beta_t(\delta_t x(S_t) - \mathbf{w}_t) \pi(s_t, a_t)$
  - GTD2 (gradient temporal difference learning, version 2)
  - TDC (temporal difference learning with gradient corrections.)



Convergence of Control Algorithms

- Gradient TD algorithms with linear function approximation problems are guaranteed convergent under both general on- and off-policy training
- The computational complexity is only O(n) (n is the number of features)
- the curse of dimensionality is removed

Off Policy Function Approximation Convergence

- Extensive work in better TD-style algorithms with value function approximation, some with convergence guarantees: see Chp 11 SB
- Exciting recent work on batch RL that can converge with nonlinear VFA (Dai et al. ICML 2018): uses primal dual optimization
- An important issue is not just whether the algorithm converges, but what solution it converges too
- Critical choices: objective function and feature representation

Your knowledge by now

- Be able to implement TD(0) and MC on policy evaluation with linear value function approximation
- Be able to define what TD(0) and MC on policy evaluation with linear VFA are converging to and when this solution has 0 error and non-zero error.
- Be able to implement Q-learning and SARSA and MC control algorithms
- List the 3 issues that can cause instability and describe the problems qualitatively: function approximation, bootstrapping and off policy learning

- Next time
  - Deep Reinforcement Learning
    - Value-based methods
    - Policy Gradient

Lecture 5

- Reading:
  - RUSSELL, S. NORVIG, P. Artificial Intelligence. 3a edição. Chapter 21.
  - BARTO, A., SUTTON, R. Reinforcement Learning: An Introduction. Second Edition. Freely Available at: http://www.incompleteideas.net/book/RLbook2020.pdf

References

Lecture 5

- BARTO, A., SUTTON, R. Reinforcement Learning: An Introduction. Second Edition.
- MURPHY, R. R. Introduction to AI robotics. MIT Press, 2002.
- Lex Fridman, MIT Deep Learning Course, MIT, 2019.
- DUDEK, G.; JENKIN, M. Computational Principles of mobile robotics. Cambridge Press, 2000.
- ROMERO, R. A. F.; PRESTES, E.; OSÓRIO, F.; WOLF, D. (Orgs) Robótica móvel. LTC, 2014.
- BROOKS, R. Intelligence without representation. Artificial Intelligence, 47:139-159, 1991.
- RUSSEL, S. NORVIG, P. Artificial Intelligence: a modern approach. Prentice Hall, 2002.
- BRATKO, I. PROLOG: programming for artificial intelligence. Addison Wesley, 2nd edition, 1990.

This material is part of the Machine Learning Course

By Esther Colombini and Alexandre Simões