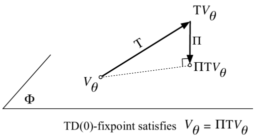
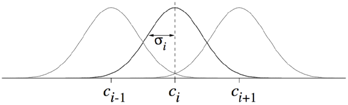
Reinforcement Learning

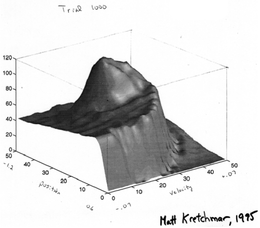
RL in continuous MDPs

Marcello Restelli

March–April, 2015







Large/Continuous MDPs

- Large/Continuous **state** space
  - Tabular representation cannot be used
- Large/Continuous **action** space
  - Maximization over action space is problematic
- Continuous **time**
  - Hamilton–Jacoby–Bellman equation

Function Approximation in RL: Why?

- Reinforcement learning can be used to solve **large** problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter: continuous state space
- How can we scale up the model-free methods for **prediction** and **control**?

Value Function Approximation

- So far we have represented 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**

$$\begin{aligned} V_\theta(s) &\approx V^\pi(s) \\ Q_\theta(s, a) &\approx Q^\pi(s, a) \end{aligned}$$

  - **Generalize** from seen states to unseen states
  - **Update** parameter θ using MC or TD learning

Which Function Approximation?

- There are **many** function approximators, e.g.
  - Artificial neural network
  - Decision tree
  - Nearest neighbor
  - Fourier/wavelet bases
  - Coarse coding
- **In principle**, any function approximator can be used. However, the choice may be affected by some properties of RL:
  - Experience is **not i.i.d.** – successive timesteps are correlated
  - During control, value function $V^\pi(s)$ is **non–stationary**
  - Agent's action **affect** the subsequent data it receives
  - Feedback is **delayed**, not instantaneous

Gradient Descent

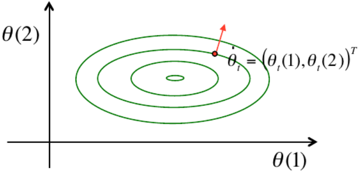
- Let $L(\theta)$ be a **differentiable** function of parameter vector θ
- Define the **gradient** of $L(\theta)$ to be

$$\nabla_\theta L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial L(\theta)}{\partial \theta_n} \end{bmatrix}$$

- To find a **local minimum** of $L(\theta)$
- Adjust the parameter θ in the direction of **negative gradient**

$$\Delta\theta = -\frac{1}{2}\alpha \nabla_\theta L(\theta)$$

where α is a stepsize parameter



Value Function Approximation by Stochastic Gradient Descent

- **Goal**: find parameter vector θ **minimizing** mean–squared error between approximate value function $V_\theta(s)$ and true value function $V^\pi(s)$

$$L(\theta) = \mathbb{E}_\pi[(V^\pi(s) - V_\theta(s))^2 | s_t = s]$$

- Gradient descent finds a **local** minimum

$$\begin{aligned}\Delta\theta &= -\frac{1}{2}\alpha\nabla_\theta L(\theta) \\ &= \alpha\mathbb{E}_\pi[(V^\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s) | s_t = s]\end{aligned}$$

- Stochastic gradient descent **samples** the gradient

$$\Delta\theta = \alpha(V^\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)$$

- **Expected** update is equal to **full** gradient update

Feature Vectors

- Represent state by a **feature vector**

$$\phi(s) = \begin{bmatrix} \phi_1(s) \\ \vdots \\ \phi_n(s) \end{bmatrix}$$

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

Linear Value Function Approximation

- Represent value function by a **linear combination** of features

$$V_\theta(s) = \phi(s)^\mathsf{T}\theta = \sum_{j=1}^{n} \phi_j(s)\theta_j$$

- Objective function is **quadratic**

$$J(\theta) = \mathbb{E}\left[(V^\pi(s) - \phi(s)^\mathsf{T}\theta)^2 | s_t = s\right]$$

- Stochastic gradient descent converges to **global** optimum
- **Update rule** is particularly simple

$$\begin{aligned} \nabla_\theta V_\theta(s) &= \phi(s) \\ \Delta\theta &= \alpha(V^\pi(s) - V_\theta(s))\phi(s) \end{aligned}$$

- Update = stepsize $\times$ prediction error $\times$ feature value

Table Lookup Features

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

$$\phi^{table}(s) = \begin{bmatrix} \mathbf{1}(s = s_1) \\ \vdots \\ \mathbf{1}(s = s_n) \end{bmatrix}$$

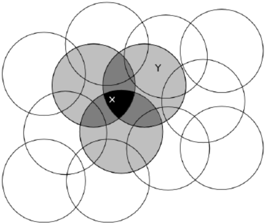
- Parameter vector θ gives value of **each individual state**

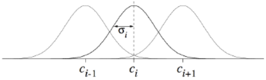
$$V(s) = \begin{bmatrix} \mathbf{1}(s = s_1) \\ \vdots \\ \mathbf{1}(s = s_n) \end{bmatrix}^{\mathrm{T}} \cdot \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

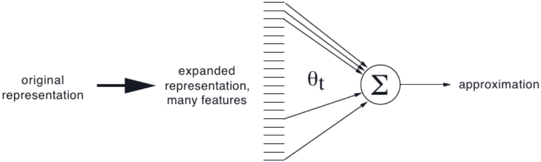
Coarse Coding

Example of linear value function approximation:

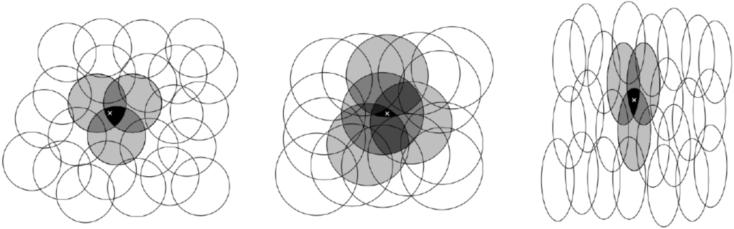
- Coarse coding provides **large** feature vector $\phi(s)$
- Parameter vector θ gives a value to **each feature**





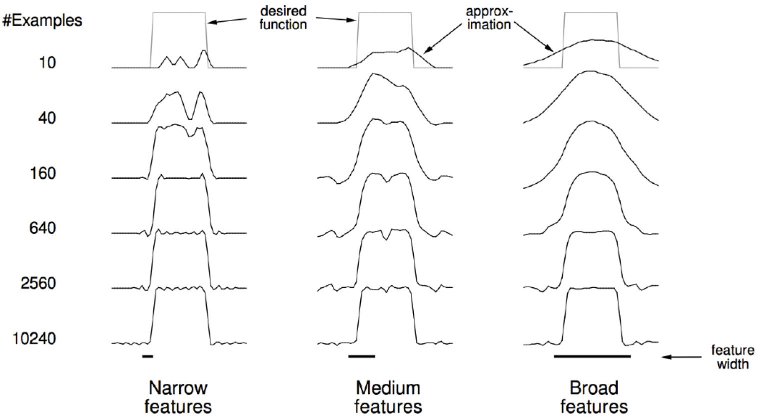


Generalization in Coarse Coding



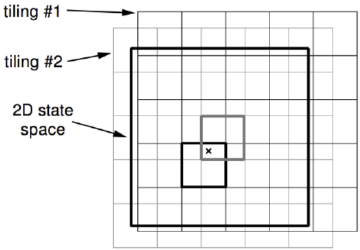
a) Narrow generalization b) Broad generalization c) Asymmetric generalization

Stochastic Gradient Descent with Coarse Coding



Tile Coding

- **Binary feature** for each tile
- Number of features present at any one time is **constant**
- Binary features means weighted sum **easy to compute**
- Easy to compute **indices** of the features present

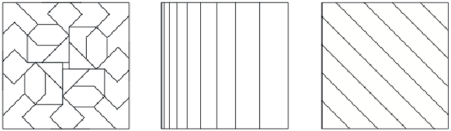


Shape of tiles ⇒ Generalization

#Tilings ⇒ Resolution of final approximation

Tile Coding

- Irregular tilings

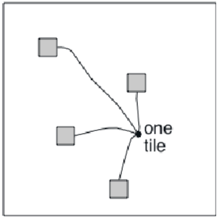


a) Irregular

b) Log stripes

c) Diagonal stripes

- Hashing

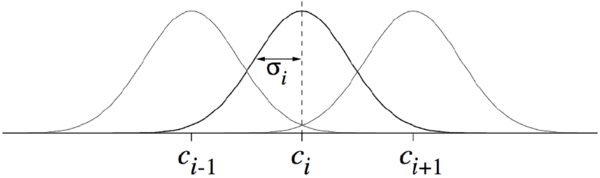


- CMAC [Albus, 1971] “Cerebellar Model Architecture Computer”

Radial Basis Functions (RBFs)

e.g., Gaussians

$$\phi_i(s) = \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right)$$



Incremental Prediction Algorithm

- Have 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 v_t

$$\Delta\theta = \alpha(v_t - V_\theta(s))\nabla_\theta V_\theta(s)$$

  - For TD(0), the target is the TD target $r + \gamma V(s')$

$$\Delta\theta = \alpha(r + \gamma V(s') - V_\theta(s))\nabla_\theta V_\theta(s)$$

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

$$\Delta\theta = \alpha(v_t^\lambda - V_\theta(s))\nabla_\theta V_\theta(s)$$

Monte–Carlo with Value Function Approximation

- The return v_t is an **unbiased**, noisy sample of true value $V^{\pi}(s)$
- Can therefore apply **supervised learning** to "training data":

$$\langle s_1, v_1\rangle, \langle s_2, v_2\rangle, \dots, \langle s_T, v_T\rangle$$

- For example, using **linear** Monte–Carlo **policy evaluation**

$$\begin{aligned} \Delta\theta &= \alpha(v_t - V_\theta(s))\nabla_\theta V_\theta(s) \\ &= \alpha(v_t - V_\theta(s))\phi(s) \end{aligned}$$

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

TD Learning with Value Function Approximation

- The TD–target $r_{t+1} + \gamma V_\theta(s_{t+1})$ is a **biased** sample of true value $V^\pi(s_t)$
- Can still apply supervised learning to "training data":

$$\langle s_1, r_2 + \gamma V_\theta(s_2)\rangle, \langle s_2, r_3 + \gamma V_\theta(s_3)\rangle, \ldots, \langle s_{T-1}, r_t\rangle$$

- For example, using **linear TD(0)**

$$\begin{aligned} \Delta\theta &= \alpha(r + \gamma V_\theta(s') - V_\theta(s))\nabla_\theta V_\theta(s) \\ &= \alpha\delta\phi(s) \end{aligned}$$

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

TD(λ) with Value Function Approximation

- The λ–return v_t^λ is also a **biased** sample of true value $V^\pi(s)$
- Can again apply supervised learning to "training data":

$$\langle s_1, v_1^\lambda \rangle, \langle s_2, v_2^\lambda \rangle, \ldots, \langle s_{T-1}, v_{T-1}^\lambda \rangle$$

- **Forward view** linear TD(λ)

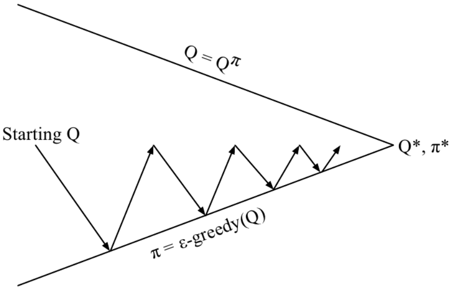
$$\begin{aligned} \Delta\theta &= \alpha(v_t^\lambda - V_\theta(s_t))\nabla_\theta V_\theta(s_t) \\ &= \alpha(v_t^\lambda - V_\theta(s_t))\phi(s_t) \end{aligned}$$

- **Backward view** linear TD(λ)

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \\ e_t &= \gamma\lambda e_{t-1} + \phi(s_t) \\ \Delta\theta &= \alpha\delta_t e_t \end{aligned}$$

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

Control with Value Function Approximation



- **Policy Evaluation**: Approximate policy evaluation, $Q_\theta \approx Q^\pi$
- **Policy Improvement**: ϵ–greedy policy improvement

Action–Value Function Approximation

- **Approximate** the action-value function

$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- **Minimize** the mean–squared error between approximate action–value function $Q_\theta(s,a)$ and true action–value function $Q^\pi(s,a)$
- Use **stochastic gradient descent** to find local minimum

$$\begin{aligned} -\frac{1}{2}\nabla_\theta J(\theta) &= (Q^\pi(s,a) - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a) \\ \Delta\theta &= \alpha(Q^\pi(s,a) - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a) \end{aligned}$$

Linear Action–Value Function Approximation

- Represent state **and** action by a **feature vector**

$$\phi(s,a) = \begin{bmatrix} \phi_1(s,a) \\ \vdots \\ \phi_n(s,a) \end{bmatrix}$$

- Represent action–value function by **linear combination of features**

$$Q_\theta(s,a) = \phi(s,a)^\top \theta = \sum_{j=1}^{n} \phi_j(s,a)\theta_j$$

- Stochastic gradient descent **update**

$$\begin{aligned} \nabla_\theta Q_\theta(s,a) &= \phi(s,a) \\ \Delta\theta &= \alpha(Q^\pi(s,a) - Q_\theta(s,a))\phi(s,a) \end{aligned}$$

Incremental Control Algorithms

- Like prediction, we must substitute a **target** for $Q^{\pi}(s, a)$
  - For MC, the target is the return v_t

$$\Delta\theta = \alpha(v_t - Q_\theta(s_t, a_t))\phi(s_t, a_t)$$

  - For TD(0), the target is the TD target $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$

$$\Delta\theta = \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t))\phi(s_t, a_t)$$

  - For forward–view TD(λ), target is λ–return v_t^λ

$$\Delta_\theta = \alpha(v_t^\lambda - Q_\theta(s_t, a_t))\phi(s_t, a_t)$$

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

$$\begin{aligned}
\delta_t &= r_{t+1} + \gamma Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t) \\
e_t &= \gamma\lambda e_{t-1} + \phi(s_t, a_t) \\
\Delta_\theta &= \alpha\delta_t e_t
\end{aligned}$$

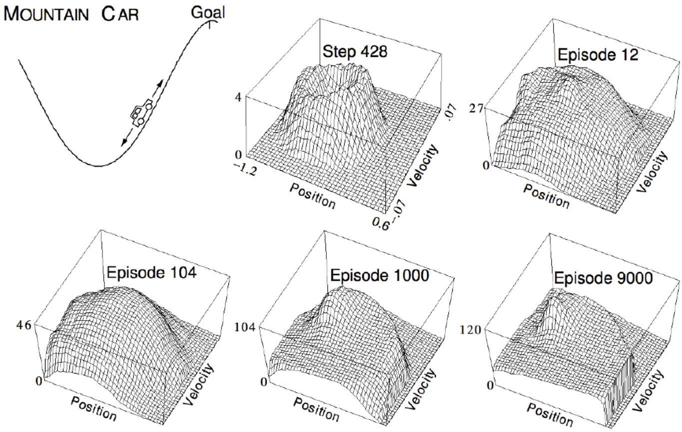
GPI Linear Gradient Descent SARSA(0)

Initialize $\boldsymbol{\theta}$ arbitrarily
loop
  $s, a \leftarrow$ initial state and action of episode
  $Q(s, a) = \boldsymbol{\theta}^{\mathrm{T}} \phi(s, a)$
  repeat
    $\nabla Q_a = \phi(s, a)$
    Take action a, observe reward, r, and next state s
    $\delta \leftarrow r - Q_a$
    Uniformly draw a number $\rho \in [0, 1]$
    if $\rho \leq 1 - \epsilon$ **then**
      for all $a \in \mathcal{A}(s)$ **do**
        $Q_a \leftarrow \boldsymbol{\theta}^{\mathrm{T}} \phi(s, a)$
      end for
      $a \leftarrow arg \max_a Q_a$
    else
      $a \leftarrow$ a random action $\in \mathcal{A}(s)$
    end if
    $Q_a \leftarrow \boldsymbol{\theta}^{\mathrm{T}} \phi(s, a)$
    $\delta \leftarrow \delta + \gamma Q_a$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \delta \nabla Q_a$
  until s is terminal
end loop

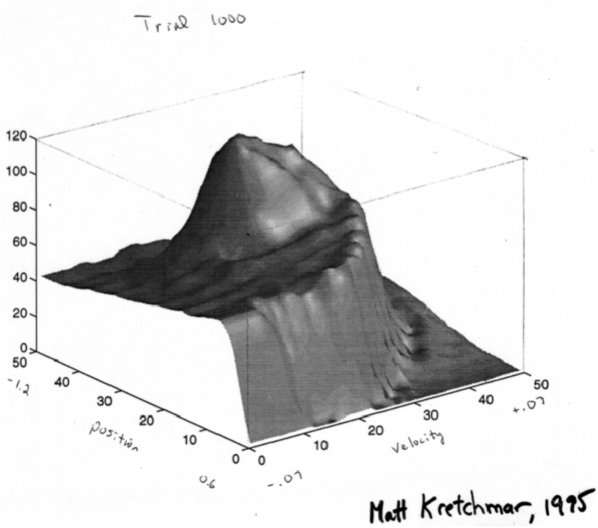
GPI Linear Gradient Descent Watkins' Q(0)

Initialize $\boldsymbol{\theta}$ arbitrarily
loop
    $s, a \leftarrow$ initial state and action of episode
    $Q(s, a) = \boldsymbol{\theta}^{\mathrm{T}} \phi(s, a)$
    repeat
        $\nabla Q_a = \phi(s, a)$
        Take action a, observe reward, r, and next state s
        $\delta \leftarrow r - Q_a$
        for all $a \in \mathcal{A}(s)$ **do**
            $Q_a \leftarrow \boldsymbol{\theta}^{\mathrm{T}} \phi(s, a)$
        end for
        $\delta \leftarrow \delta + \gamma \max_a Q_a$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \delta \nabla Q_a$
        Uniformly draw a number $\rho \in [0, 1]$
        if $\rho \leq 1 - \epsilon$ **then**
            for all $a \in \mathcal{A}(s)$ **do**
                $Q_a \leftarrow \boldsymbol{\theta}^{\mathrm{T}} \phi(s, a)$
            end for
            $a \leftarrow \arg\max_a Q_a$
        else
            $a \leftarrow$ a random action $\in \mathcal{A}(s)$
        end if
        $Q_a \leftarrow \boldsymbol{\theta}^{\mathrm{T}} \phi(s, a)$
    until s is terminal
end loop

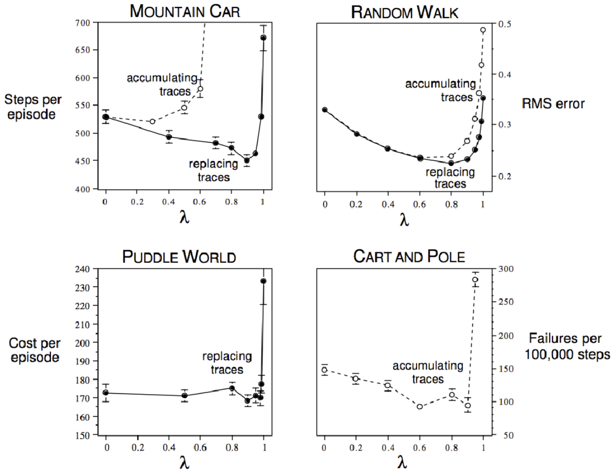
Linear SARSA with Coarse Coding in Mountain Car



Linear SARSA with Radial Basis Functions in Mountain Car



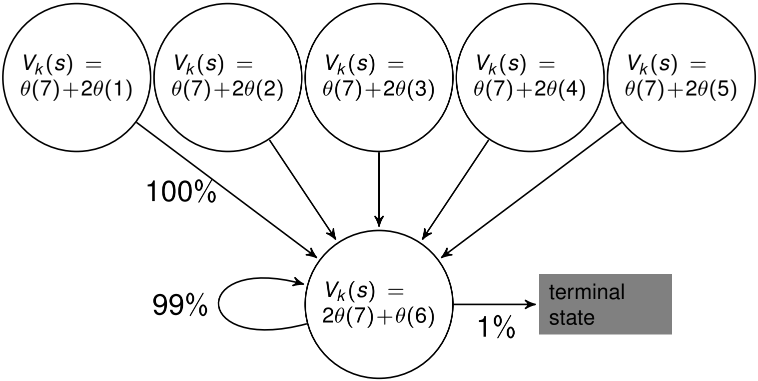
Study of λ: Should We Bootstrap?



Convergence Questions

- The previous results show it is desirable to **bootstrap**
- But now we consider **convergence issues**
- When do incremental prediction algorithms converge?
  - When using **bootstrapping** (i.e., TD with $\lambda < 1$)?
  - When using **linear** function approximation?
  - When using **off–policy** learning?
- Ideally, we would like algorithms that converge in **all cases**

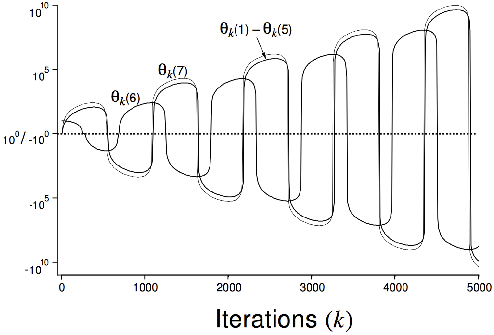
Baird's Counterexample



Reward is always **zero**

Parameter Divergence in Baird's Counterexample

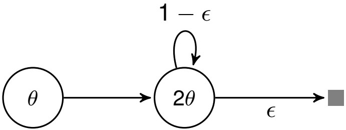
Parameter values, $\theta_k(i)$ (log scale, broken at ±1)
Using **uniform distribution**



$\gamma = 0.99$, $\alpha = 0.01$,
$\boldsymbol{\theta}_0 = [1, 1, 1, 1, 1, 10, 1]^{\mathsf{T}}$

Another Example

- Baird's counterexample has two simple **fixes**:
  - use **on–policy** distribution
  - Instead of taking small steps towards expected one–step returns, change value function to the best **least–squares approximation**
- This works if the feature vectors form a **linearly independent set**
- When an **exact solution** is not possible, it does not work even if we consider the **best approximation** at each iteration

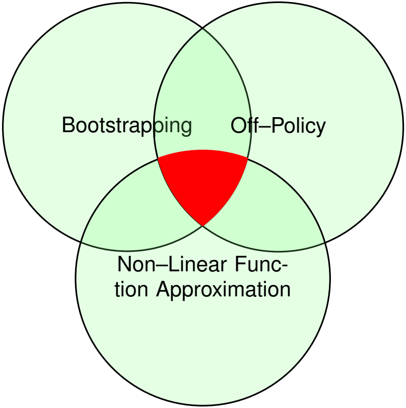


$$\begin{aligned}
\theta_{k+1} &= arg\min_{\theta\in\Re}\sum_{s\in\mathcal{S}}\{V_\theta(s) - \mathbb{E}_\pi[r_{t+1} + \gamma V_{\theta_k}(s_{t+1})|s_t = s]\} \\
&= arg\min_{\theta\in\Re}[\theta - 2\gamma\theta_k]^2 + [2\theta - 2(1-\epsilon)\gamma\theta_k]^2 = \frac{6-4\epsilon}{5}\gamma\theta_k
\end{aligned}$$

Convergence of Prediction Algorithms

On/Off–Policy	Algorithm	Table Lookup	Linear	Non–Linear
On–Policy	MC	OK	OK	OK
	TD(0)	OK	OK	KO
	TD(λ)	OK	OK	KO
Off–Policy	MC	OK	OK	OK
	TD(0)	OK	KO	KO
	TD(λ)	OK	KO	KO

Problematic Triumvirate



We have not **quite** achieved our ideal goal for prediction algorithms

Convergence of Control Algorithms

Algorithm	Table Lookup	Linear
Monte–Carlo Control	OK	OK
SARSA	OK	(OK)
Q–learning	OK	KO

(OK) = **chatters** around near–optimal value function

What Objective Function does TD(0) Optimize?

- Mean–squared error

$$MSE(\theta) = \|V_\theta - V^\pi\|_D^2 = \mathbb{E}_{s\sim D}[(V_\theta(s) - V^\pi(s))^2]$$

- Mean–squared **TD** error

$$MSTDE(\theta) = \mathbb{E}_{D,P,\pi}[\delta_t^2 | s_t = s]$$

- Mean–squared **Bellman** error

$$MSBE(\theta) = \|V_\theta - T^\pi V_\theta\|_D^2 = \mathbb{E}_{s\sim D}[\mathbb{E}_{P,\pi}[\delta_t | s_t = s]^2]$$

- Mean–squared **projected Bellman** error

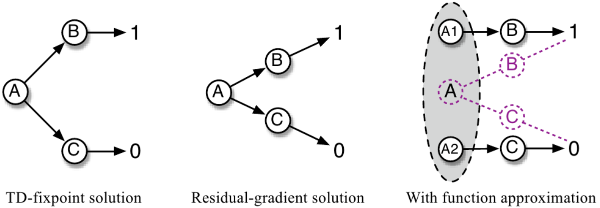
$$MSPBE(\theta) = \|V_\theta - \Pi T^\pi V_\theta\|_D^2$$

Split–A Example

Marcello Restelli

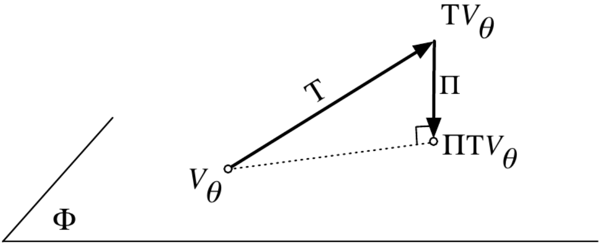
Incremental Methods

Batch methods



- Solution minimizing MSE: **Correct solution**
$$V(A) = 0.5, V(B) = 1, V(C) = 0$$
- Solution minimizing MSTDE (with table lookup):
$$V(A) = 0.5, V(B) = 0.75, V(C) = 0.25$$
- Solution minimizing MSBE (with function approximation):
$$V(A) = 0.5, V(B) = 0.75, V(C) = 0.25$$
- Solution minimizing MSPBE: **Correct solution**
$$V(A) = 0.5, V(B) = 1, V(C) = 0$$

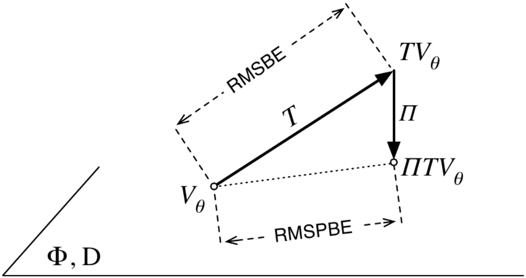
TD(0) Objective



TD(0)-fixpoint satisfies $V_\theta = \Pi T V_\theta$

- TD(0) finds the solution minimizing **MSPBE**
- But TD does not follow the gradient of **any** objective function
- This is why TD(0) can **diverge** when **off–policy** or using **non–linear** function approximation

Gradient Temporal–Difference Learning



- **Gradient** TD learning explicitly follows gradient of MSPBE

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta \| V_\theta - \Pi T^\pi V_\theta \|_D^2$$

- MSPBE is optimized at TD(0) **fixed point**
- Gradient descent of MSPBE finds TD(0) fixed point – robustly

Gradient Temporal–Difference Learning Algorithm

-

$$
\begin{aligned}
MSPBE(\theta) &= \| V_\theta - \Pi T^\pi V_\theta \|_D^2 \\
-\frac{1}{2}\nabla_\theta MSPBE(\theta) &= \mathbb{E}[\delta\phi] - \gamma\mathbb{E}[\phi'\phi^{\mathrm{T}}]\mathbb{E}[\phi\phi^{\mathrm{T}}]^{-1}\mathbb{E}[\delta\phi] \\
&\stackrel{\text{def}}{=} \mathbb{E}[\delta\phi] - \gamma\mathbb{E}[\phi'\phi^{\mathrm{T}}]w
\end{aligned}
$$

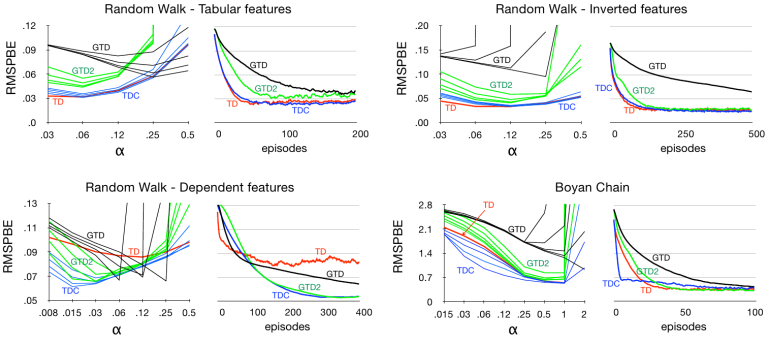
- This leads to the **linear TDC algorithm**

$$
\begin{aligned}
\Delta\theta &= \alpha(\delta\phi - \gamma\phi'(\phi^{\mathrm{T}} w)) \\
\Delta w &= \beta(\delta - \phi^{\mathrm{T}} w)\phi
\end{aligned}
$$

- The highlighted term is a **correction** of the TD(0) update

$$\Delta\theta = \alpha\delta\phi$$

Gradient Temporal Difference Learning Results



Convergence of Incremental Prediction Algorithms

On/Off–Policy	Algorithm	Table Lookup	Linear	Non–Linear
On–Policy	MC	OK	OK	OK
	TD(0)	OK	OK	KO
	TDC	**OK**	**OK**	**OK**
Off–Policy	MC	OK	OK	OK
	TD(0)	OK	KO	KO
	TDC	**OK**	**OK**	**OK**

Batch Reinforcement Learning

- Gradient descent is **simple** and appealing
- But it is **not** sample efficient
- Batch methods seek to find the **best fitting** value function
- Given the agent's **experience** ("training data")

Least Squares Prediction

- Given value function **approximation** $V_\theta(s) \approx V^\pi(s)$
- And **experience** $\mathcal{D}$ consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$$

- Which parameters θ give the **best fitting** value function $V_\theta(s)$?
- Least squares algorithms find parameter vector θ **minimizing sum–squared error** between $V_\theta(s_t)$ and target values v_t^π,

$$\begin{aligned} LS(\theta) &= \sum_{t=1}^{T} (v_t^\pi - V_\theta(s_t))^2 \\ &= \mathbb{E}_{\mathcal{D}}[(v^\pi - V_\theta(s))^2] \end{aligned}$$

Stochastic Gradient Descent with Experience Replay

- Given **experience** consisting of $\langle$*state*, *value*$\rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi\rangle, \langle s_2, v_2^\pi\rangle, \ldots, \langle s_T, v_T^\pi\rangle\}$$

- Repeat
  1. **Sample** state, value from experience

$$\langle s, v^\pi\rangle \sim \mathcal{D}$$

  2. Apply stochastic gradient descent **update**

$$\Delta\theta = \alpha(v^\pi - V_\theta(s))\nabla_\theta V_\theta(s)$$

- **Converges** to least squares solution

$$\theta^\pi = arg\min_\theta LS(\theta)$$

Linear Least Squares Prediction

- **Experience replay** finds least squares solution
- But it may take **many iterations**
- Using **linear** value function approximation $V_\theta(s) = \phi(s)^\top \theta$
- We can solve the least squares solution **directly**

Linear Least Squares Prediction (2)

- At minimum of $LS(\theta)$, the expected update must be **zero**

$$
\begin{aligned}
\mathbb{E}_{\mathcal{D}}[\Delta\theta] &= 0 \\
\sum_{t=1}^{T} \phi(s_t)(v_t^\pi - \phi(s_t)^\intercal \theta) &= 0 \\
\sum_{t=1}^{T} \phi(s_t) v_t^\pi &= \sum_{t=1}^{T} \phi(s_t)\phi(s_t)^\intercal \theta \\
\theta &= \left(\sum_{t=1}^{T} \phi(s_t)\phi(s_t)^\intercal \right)^{-1} \sum_{t=1}^{T} \phi(s_t) v_t^\pi
\end{aligned}
$$

- For N features, **direct** solution time is $O(N^3)$
- **Incremental** solution time is $O(N^2)$ using Shermann–Morrison

Linear Least Squares Prediction Algorithms

- We **do not know** true value v_t^π
- In practice, our "training data" must use **noisy** or **biased** samples of v_t^π
  - LSMC: Least Squares Monte–Carlo uses **return**

$$v_t^\pi \approx v_t$$

  - LSTD: Least Squares Temporal–Difference uses **TD target**

$$v_t^\pi \approx r_{t+1} + \gamma V_\theta(s_{t+1})$$

  - LSTD(λ): Least Squares TD(λ) uses λ**–return**

$$v_t^\pi \approx v_t^\lambda$$

- In each case solve directly for **fixed point** of MC/TD/TD(λ)

Linear Least Squares Prediction Algorithms (2)

- LSMC

$$
\begin{aligned}
0 &= \sum_{t=1}^{T} \alpha(v_t - V_\theta(s_t))\phi(s_t) \\
\theta &= \left(\sum_{t=1}^{T} \phi(s_t)\phi(s_t)^\mathrm{T}\right)^{-1} \sum_{t=1}^{T} \phi(s_t)v_t
\end{aligned}
$$

- LSTD

$$
\begin{aligned}
0 &= \sum_{t=1}^{T} \alpha(r_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t))\phi(s_t) \\
\theta &= \left(\sum_{t=1}^{T} \phi(s_t)(\gamma\phi(s_{t+1}) - \phi(s_t))^\mathrm{T}\right)^{-1} \sum_{t=1}^{T} \phi(s_t)r_{t+1}
\end{aligned}
$$

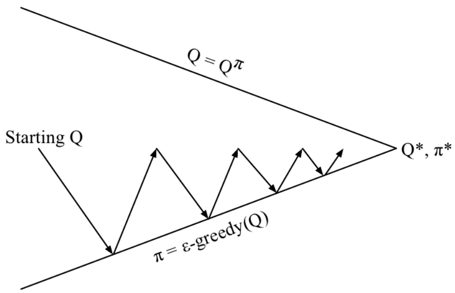
- LSTD(λ)

$$
\begin{aligned}
0 &= \sum_{t=1}^{T} \alpha\delta_t e_t \\
\theta &= \left(\sum_{t=1}^{T} e_t(\gamma\phi(s_{t+1}) - \phi(s_t))^\mathrm{T}\right)^{-1} \sum_{t=1}^{T} e_t r_{t+1}
\end{aligned}
$$

Convergence of Linear Least Squares Prediction Algorithms

On/Off–Policy	Algorithm	Table Lookup	Linear
On–Policy	LSMC	OK	OK
	LSTD(0)	OK	OK
	LSTD(λ)	OK	OK
Off–Policy	LSMC	OK	OK
	LSTD(0)	OK	OK
	LSTD(λ)	OK	OK

Least Squares Policy Iteration



- **Policy Evaluation**: Least Squares policy evaluation, $Q_\theta \approx Q^\pi$
- **Policy Improvement**: ϵ–greedy policy improvement

Least Squares Action–Value Function Approximation

- **Approximate** $Q(s, a)$ using linear combination of features

$$Q_\theta = \phi(s, a)^\mathsf{T}\theta \approx Q^\pi(s, a)$$

- **Minimize** least squares error between approximate action–value function $Q_\theta(s, a)$ and true action–value function $Q^\pi(s, a)$
- LSTDQ algorithm minimizes least squares **TD error**
- Expected SARSA update must be **zero** at **TD fixed point**

$$\begin{aligned} 0 &= \sum_{t=1}^{T} \alpha\delta_t\phi(s_t, a_t) = \sum_{t=1}^{T} \alpha(r_{t+1} + \gamma Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t))\phi(s_t, a_t) \\ \theta &= \left(\sum_{t=1}^{T} \phi(s_t, a_t)(\gamma\phi(s_{t+1}, a_{t+1}) - \phi(s_t, a_t))\right)^{-1} \sum_{t=1}^{T} \phi(s_t, a_t) r_{t+1} \end{aligned}$$

- Similarly for LSMCQ and LSTDQ(λ)

Least Squares Policy Iteration Algorithm

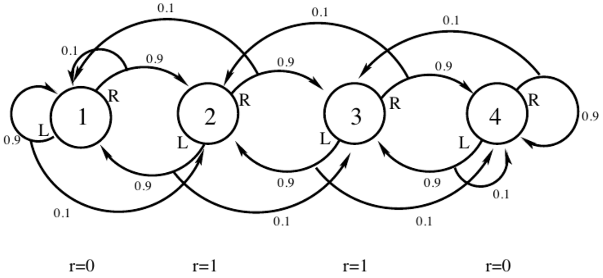
- The following pseudocode uses LSTDQ for policy evaluation
- It repeatedly re–evaluates experience $\mathcal{D}$ with different policies

function LSPI–TD($\mathcal{D}$,π_0)
    $\pi' \leftarrow \pi_0$
    repeat
        $\pi \leftarrow \pi'$
        $Q \leftarrow \text{LSTDQ}(\pi, \mathcal{D})$
        for all $s \in \mathcal{S}$ **do**
            $\pi'(s) \leftarrow arg\max_{a \in \mathcal{A}} Q(s, a)$
        end for
    until $\pi \approx \pi'$
    return π
end function

Convergence of Control Algorithms

Algorithm	Table Lookup	Linear
LSPI–MC	OK	OK
LSPI–TD(0)	OK	OK
LSPI–TD(λ)	OK	OK

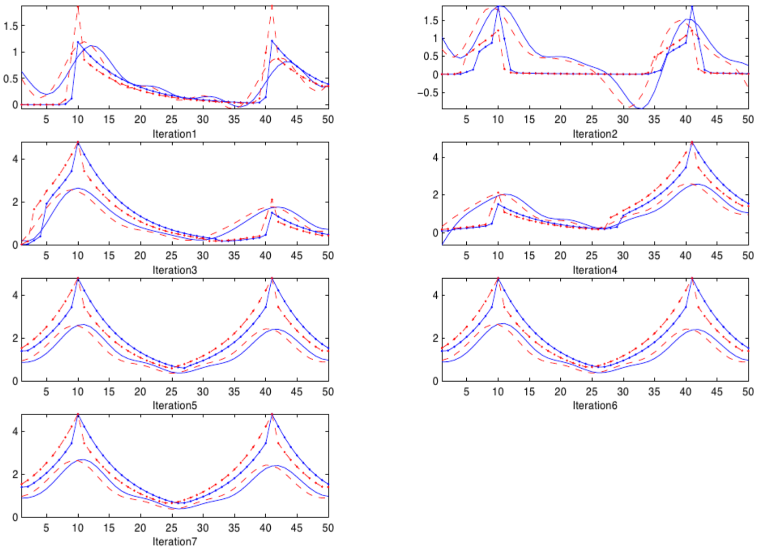
Chain Walk Example



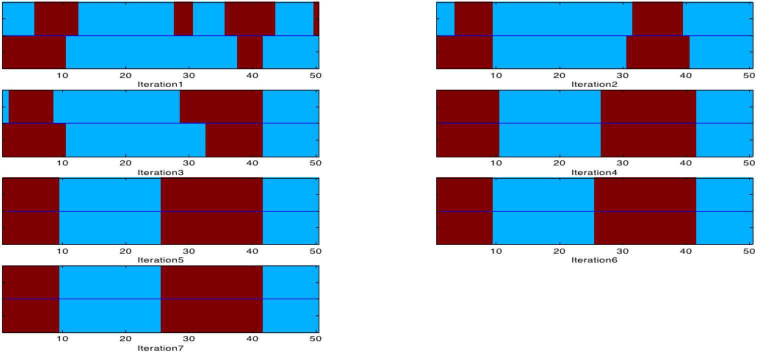
- Consider the **50–state version** of this problems
- **Reward** +1 in states 10 and 41, 0 elsewhere
- **Optimal policy**: R (1–9), L (10–25), R (26–41), L (42–50)
- **Features**: 10 evenly spaced Gaussians ($\sigma = 4$) for each action
- **Experience**: 10,000 steps from random walk policy

LSPI in Chain Walk

Action–Value Function



LSPI in Chain Walk
Policy



Fitted Q–Iteration

- Implements **fitted value iteration**
- Given a dataset of experience tuple $\mathcal{D}$, solve a **sequence of regression problems**
  - At iteration i, build an approximation $\hat{Q}_i$ over a dataset obtained by $(T^* Q_{i-1})(s, a)$
- Allows to use a large class of regression methods (**averagers**), e.g.
  - Kernel averaging
  - Regression trees
  - Fuzzy regression
- With other regression methods it **may diverge**
- In practice, good results also with **neural networks**

Fitted Q–Iteration Algorithm

Input: a set of four–tuples $\mathcal{D} = \{\langle s_i, a_i, r_{i+1}, s'_{i+1}\rangle\}_{i=1}^{L}$ and a regression algorithm
Initialize: set N to 0, $\hat{Q}_N(s,a) = 0, \quad \forall s, a$
repeat
  $N \leftarrow N + 1$
  Build a training set
$\mathcal{TS} = \{\langle s_i, a_i, r_{i+1} + \gamma \max_{a \in \mathcal{A}} \hat{Q}_{N-1}(s'_{i+1}, a)\rangle\}_{i=1}^{L}$
  Use the regression algorithm on $\mathcal{TS}$ to build $\hat{Q}_N(s,a)$
until Stopping condition

- Stopping condition:
  - **Fixed** number of iterations
  - When the **distance** between $\hat{Q}_N$ and $\hat{Q}_{N-1}$ drops below a **threshold**