

A Guided Tour of Chapter 13: Batch RL, Experience-Replay, DQN, LSPI, Gradient TD

Ashwin Rao

ICME, Stanford University

Moving on to practically sophisticated algorithms

- Let's examine the core pattern of RL Algorithms we've learnt so far
- Experiences arrive one at a time, is used (for learning) and *discarded*
- Learning is *incremental*, with VF update after each unit of experience
- Are there alternative patterns we can employ? The answer is *Yes*
- We highlight 2 key patterns that yield a richer range of RL Algorithms
 - ① *Experience-Replay*: Store the data as it arrives, and re-use it
 - ② *Batch RL*: Learn the VF for an entire batch of data directly
- Experience-Replay and Batch RL can be combined in interesting ways

Incremental RL makes inefficient use of training data

- Incremental versus Batch RL in the context of fixed finite data
- Let's understand the difference for the simple case of MC Prediction
- Given fixed finite sequence of trace experiences yielding training data:

$$\mathcal{D} = [(S_i, G_i) | 1 \leq i \leq n]$$

- Incremental MC estimates $V(s; \mathbf{w})$ using $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ for each data pair:

$$\mathcal{L}_{(S_i, G_i)}(\mathbf{w}) = \frac{1}{2} \cdot (V(S_i; \mathbf{w}) - G_i)^2$$

$$\nabla_{\mathbf{w}} \mathcal{L}_{(S_i, G_i)}(\mathbf{w}) = (V(S_i; \mathbf{w}) - G_i) \cdot \nabla_{\mathbf{w}} V(S_i; \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha \cdot (G_i - V(S_i; \mathbf{w})) \cdot \nabla_{\mathbf{w}} V(S_i; \mathbf{w})$$

- n updates are performed in sequence for $i = 1, 2, \dots, n$
- Uses update method of FunctionApprox for each data pair (S_i, G_i)
- Incremental RL makes inefficient use of available training data \mathcal{D}
- Essentially each data point is “discarded” after being used for update

Batch MC Prediction makes efficient use of training data

- Instead we'd like to estimate the Value Function $V(s; \mathbf{w}^*)$ such that

$$\begin{aligned}\mathbf{w}^* &= \arg \min_{\mathbf{w}} \frac{1}{n} \cdot \sum_{i=1}^n \frac{1}{2} \cdot (V(S_i; \mathbf{w}) - G_i)^2 \\ &= \arg \min_{\mathbf{w}} \mathbb{E}_{(S, G) \sim \mathcal{D}} \left[\frac{1}{2} \cdot (V(S; \mathbf{w}) - G)^2 \right]\end{aligned}$$

- This is the solve method of FunctionApprox on training data \mathcal{D}
- This approach to RL is known as *Batch RL*
- solve by doing updates with repeated use of available data pairs
- Each update using random data pair $(S, G) \sim \mathcal{D}$

$$\Delta \mathbf{w} = \alpha \cdot (G - V(S; \mathbf{w})) \cdot \nabla_{\mathbf{w}} V(S; \mathbf{w})$$

- This will ultimately converge to desired value function $V(s; \mathbf{w}^*)$
- Repeated use of available data known as *Experience-Replay*
- This makes more efficient use of available training data \mathcal{D}

Batch TD Prediction makes efficient use of Experiences

- In Batch TD Prediction, we have experiences data \mathcal{D} available as:

$$\mathcal{D} = [(S_i, R_i, S'_i) | 1 \leq i \leq n]$$

- Where (R_i, S'_i) is the pair of reward and next state from a state S_i
- So, Experiences \mathcal{D} in the form of finite number of atomic experiences
- This is represented in code as an Iterable [TransitionStep [S]]
- Parameters updated with repeated use of these atomic experiences
- Each update using random data pair $(S, R, S') \sim \mathcal{D}$

$$\Delta \mathbf{w} = \alpha \cdot (R + \gamma \cdot V(S'; \mathbf{w}) - V(S; \mathbf{w})) \cdot \nabla_{\mathbf{w}} V(S; \mathbf{w})$$

- This is TD Prediction with Experience-Replay on Finite Experiences \mathcal{D}

Batch TD(λ) Prediction

- In Batch TD(λ) Prediction, given finite number of trace experiences

$$\mathcal{D} = [(S_{i,0}, R_{i,1}, S_{i,1}, R_{i,2}, S_{i,2}, \dots, R_{i,T_i}, S_{i,T_i}) | 1 \leq i \leq n]$$

- Parameters updated with repeated use of these trace experiences
- Randomly pick trace experience (say indexed i) $\sim \mathcal{D}$
- For trace experience i , parameters updated at each time step t :

$$\mathbf{E}_t = \gamma \lambda \cdot \mathbf{E}_{t-1} + \nabla_{\mathbf{w}} V(S_{i,t}; \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha \cdot (R_{i,t+1} + \gamma \cdot V(S_{i,t+1}; \mathbf{w}) - V(S_{i,t}; \mathbf{w})) \cdot \mathbf{E}_t$$

The Deep Q-Networks (DQN) Control Algorithm

DQN uses **Experience-Replay** and **fixed Q-learning targets**.

At each time t for each episode:

- Given state S_t , take action A_t according to ϵ -greedy policy extracted from Q-network values $Q(S_t, a; \mathbf{w})$
- Given state S_t and action A_t , obtain reward R_{t+1} and next state S_{t+1}
- Store atomic experience $(S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory \mathcal{D}
- Sample random mini-batch of atomic experiences $(s_i, a_i, r_i, s'_i) \sim \mathcal{D}$
- Update Q-network parameters \mathbf{w} using Q-learning targets based on “frozen” parameters \mathbf{w}^- of *target network*

$$\Delta \mathbf{w} = \alpha \cdot \sum_i (r_i + \gamma \cdot \max_{a'_i} Q(s'_i, a'_i; \mathbf{w}^-) - Q(s_i, a_i; \mathbf{w})) \cdot \nabla_{\mathbf{w}} Q(s_i, a_i; \mathbf{w})$$

- $S_t \leftarrow S_{t+1}$

Parameters \mathbf{w}^- of target network infrequently updated to values of Q-network parameters \mathbf{w} (hence, Q-learning targets treated as “frozen”)

Least-Squares RL Prediction

- Batch RL Prediction for general function approximation is iterative
- Uses Experience-Replay and Gradient Descent
- We can solve directly (without gradient) for linear function approx
- Define a sequence of feature functions $\phi_j : \mathcal{S} \rightarrow \mathbb{R}, j = 1, 2, \dots, m$
- Parameters w is a weights vector $\mathbf{w} = (w_1, w_2, \dots, w_m) \in \mathbb{R}^m$
- Value Function is approximated as:

$$V(s; \mathbf{w}) = \sum_{j=1}^m \phi_j(s) \cdot w_j = \phi(s)^T \cdot \mathbf{w}$$

where $\phi(s) \in \mathbb{R}^m$ is the feature vector for state s

Least-Squares Monte-Carlo (LSMC)

- Loss function for Batch MC Prediction with data $[(S_i, G_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2n} \cdot \sum_{i=1}^n \left(\sum_{j=1}^m \phi_j(S_i) \cdot w_j - G_i \right)^2 = \frac{1}{2n} \cdot \sum_{i=1}^n (\phi(S_i)^T \cdot \mathbf{w} - G_i)^2$$

- The gradient of this Loss function is set to 0 to solve for \mathbf{w}^*

$$\sum_{i=1}^n \phi(S_i) \cdot (\phi(S_i)^T \cdot \mathbf{w}^* - G_i) = 0$$

- \mathbf{w}^* is solved as $\mathbf{A}^{-1} \cdot \mathbf{b}$
- $m \times m$ Matrix \mathbf{A} is accumulated at each data pair (S_i, G_i) as:

$$\mathbf{A} \leftarrow \mathbf{A} + \phi(S_i) \cdot \phi(S_i)^T \text{ (i.e., outer-product of } \phi(S_i) \text{ with itself)}$$

- m -Vector \mathbf{b} is accumulated at each data pair (S_i, G_i) as:

$$\mathbf{b} \leftarrow \mathbf{b} + \phi(S_i) \cdot G_i$$

- Sherman-Morrison incremental inverse can be done in $O(m^2)$

Least-Squares Temporal-Difference (LSTD)

- Loss func for Batch TD Prediction with data $[(S_i, R_i, S'_i) | 1 \leq i \leq n]$:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2n} \cdot \sum_{i=1}^n (\phi(S_i)^T \cdot \mathbf{w} - (R_i + \gamma \cdot \phi(S'_i)^T \cdot \mathbf{w}))^2$$

- The semi-gradient of this Loss function is set to 0 to solve for \mathbf{w}^*

$$\sum_{i=1}^n \phi(S_i) \cdot (\phi(S_i)^T \cdot \mathbf{w}^* - (R_i + \gamma \cdot \phi(S'_i)^T \cdot \mathbf{w}^*)) = 0$$

- \mathbf{w}^* is solved as $\mathbf{A}^{-1} \cdot \mathbf{b}$
- $m \times m$ Matrix \mathbf{A} is accumulated at each atomic experience (S_i, R_i, S'_i) :

$$\mathbf{A} \leftarrow \mathbf{A} + \phi(S_i) \cdot (\phi(S_i) - \gamma \cdot \phi(S'_i))^T \text{ (note the Outer-Product)}$$

- m -Vector \mathbf{b} is accumulated at each atomic experience (S_i, R_i, S'_i) :

$$\mathbf{b} \leftarrow \mathbf{b} + \phi(S_i) \cdot R_i$$

- Sherman-Morrison incremental inverse can be done in $O(m^2)$

LSTD(λ)

- Likewise, we can do LSTD(λ) using Eligibility Traces
- Denote the Eligibility Traces of atomic experience i as \mathbf{E}_i
- Note: \mathbf{E}_i accumulates $\nabla_{\mathbf{w}} V(s; \mathbf{w}) = \phi(s)$ in each trace experience
- When accumulating, previous step's eligibility traces discounted by $\lambda\gamma$

$$\sum_{i=1}^n \mathbf{E}_i \cdot (\phi(S_i)^T \cdot \mathbf{w}^* - (R_i + \gamma \cdot \phi(S'_i)^T \cdot \mathbf{w}^*)) = 0$$

- \mathbf{w}^* is solved as $\mathbf{A}^{-1} \cdot \mathbf{b}$
- $m \times m$ Matrix \mathbf{A} is accumulated at each atomic experience (S_i, R_i, S'_i) :

$$\mathbf{A} \leftarrow \mathbf{A} + \mathbf{E}_i \cdot (\phi(S_i) - \gamma \cdot \phi(S'_i))^T \text{ (note the Outer-Product)}$$

- m -Vector \mathbf{b} is accumulated at each atomic experience (S_i, R_i, S'_i) as:

$$\mathbf{b} \leftarrow \mathbf{b} + \mathbf{E}_i \cdot R_i$$

- Sherman-Morrison incremental inverse can be done in $O(m^2)$

Convergence of Least Squares Prediction Algorithms

On/Off Policy	Algorithm	Tabular	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✗	-
	TD	✓	✗	✗
	LSTD	✓	✗	-

Least Squares RL Control

- To perform Least Squares RL Control, we do GPI with:
 - Policy Evaluation as Least-Squares Q-Value Prediction
 - Greedy (or ϵ -Greedy) Policy Improvement
- For On-Policy MC/TD Control, Q-Value Prediction (for policy π):

$$Q^\pi(s, a) \approx Q(s, a; \mathbf{w}^*) = \phi(s, a)^T \cdot \mathbf{w}^*$$

- Direct solve for \mathbf{w}^* using experiences data generated using policy π
- We are interested in Off-Policy Control with Least-Squares TD
- Using the same idea as Q-Learning and with Experience-Replay
- This technique is known as Least Squares Policy Iteration (LSPI)

Least Squares Policy Iteration (LSPI)

- Input is fixed finite data set \mathcal{D} consisting of (s, a, r, s') experiences
- Goal is to determine Optimal Q-Value Linear Function Approximation
- Each iteration of GPI starts with a deterministic target policy π_D
- π_D is made available from the previous iteration of GPI
- Goal of the iteration is to solve for weights \mathbf{w}^* to minimize:

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \sum_i (Q(s_i, a_i; \mathbf{w}) - (r_i + \gamma \cdot Q(s'_i, \pi_D(s'_i); \mathbf{w})))^2 \\ &= \sum_i (\phi(s_i, a_i)^T \cdot \mathbf{w} - (r_i + \gamma \cdot \phi(s'_i, \pi_D(s'_i))^T \cdot \mathbf{w}))^2\end{aligned}$$

- Solved using sampled mini-batch of experiences (s_i, a_i, r_i, s'_i) from \mathcal{D}
- This solved \mathbf{w}^* defines an updated Q-Value Function
- Iteration ends by setting the target policy π_D (for next iteration) as:

$$\pi_D(s) = \arg \max_a Q(s, a; \mathbf{w}^*)$$

Solving for weights \mathbf{w}^* with LSTDQ

- We set the semi-gradient of $\mathcal{L}(\mathbf{w})$ at $\mathbf{w} = \mathbf{w}^*$ to 0

$$\sum_i \phi(s_i, a_i) \cdot (\phi(s_i, a_i)^T \cdot \mathbf{w}^* - (r_i + \gamma \cdot \phi(s'_i, \pi_D(s'_i))^T \cdot \mathbf{w}^*)) = 0 \quad (1)$$

- \mathbf{w}^* is solved as $\mathbf{A}^{-1} \cdot \mathbf{b}$
- $m \times m$ Matrix \mathbf{A} is accumulated at each experience (s_i, a_i, r_i, s'_i) :

$$\mathbf{A} \leftarrow \mathbf{A} + \phi(s_i, a_i) \cdot (\phi(s_i, a_i) - \gamma \cdot \phi(s'_i, \pi_D(s'_i)))^T$$

- m -Vector \mathbf{b} is accumulated at each experience (s_i, a_i, r_i, s'_i) as:

$$\mathbf{b} \leftarrow \mathbf{b} + \phi(s_i, a_i) \cdot r_i$$

- Sherman-Morrison incremental inverse can be done in $O(m^2)$
- This least-squares solution of \mathbf{w}^* (Prediction) is known as *LSTDQ*
- GPI with LSTDQ and greedy policy improvement known as *LSPI*

Convergence of Control Algorithms

Algorithm	Tabular	Linear	Non-Linear
MC Control	✓	(✓)	✗
SARSA	✓	(✓)	✗
Q-Learning	✓	✗	✗
LSPI	✓	(✓)	-

(✓) means it chatters around near-optimal Value Function

LSPI for Optimal Exercise of American Options

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call
- When payoff is not path-dependent and state dimension is not large, we can do backward induction on a binomial/trinomial tree/grid
- Otherwise, the standard approach is [Longstaff-Schwartz algorithm](#)
- Longstaff-Schwartz algorithm combines 3 ideas:
 - Valuation based on Monte-Carlo simulation
 - Function approximation of continuation value for in-the-money states
 - Backward-recursive determination of early exercise states
- We consider LSPI as an alternative approach for American Pricing

LSPI as an alternative to Longstaff-Schwartz

- RL is straightforward if we clearly define the MDP
- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Stop) or Continue
- *Reward* always 0, except upon Exercise (= Payoff)
- *State*-transitions based on Underlying Security's Risk-Neutral Process
- Key is function approximation of state-conditioned continuation value
- Continuation Value \Rightarrow Optimal Stopping \Rightarrow Option Price
- We customize LSPI to Optimal Exercise of American Options
- Based on [this paper by Li, Szepesvari, Schuurmans](#)

LSPI customized for American Options Pricing

- 2 actions: $a = c$ (continue the option) and $a = e$ (exercise the option)
- Create function approx representation for $Q(s, a)$ only for $a = c$ since we know option payoff $g(s)$ for $a = e$, i.e., $Q(s, a) = g(s)$

$$\hat{Q}(s, a; \mathbf{w}) = \begin{cases} \phi(s)^T \cdot \mathbf{w} & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

for feature funcs $\phi(\cdot) = [\phi_i(\cdot) | i = 1, \dots, m]$ of only state & not action

- Each iteration of GPI starts with a deterministic target policy π_D
- π_D is greedy policy from previous iteration's solved $Q(s, a; \mathbf{w}^*)$
- Since we learn Q-Value function for only $a = c$, behavior policy μ generating experiences data for training is a constant func $\mu(s) = c$
- Also, for American Options, the reward for $a = c$ is 0
- So each atomic experiences for training is of the form $(s, c, 0, s')$
- So we represent each atomic experience for training as a 2-tuple (s, s')

LSPI customized for American Options Pricing

- This reduces LSPI Semi-Gradient Equation (1) to:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \mathbf{w}^* - \gamma \cdot \hat{Q}(s'_i, \pi_D(s'_i); \mathbf{w}^*)) = 0 \quad (2)$$

- We need to consider two cases for the term $\hat{Q}(s'_i, \pi_D(s'_i); \mathbf{w}^*)$
 - C1: If s'_i is non-terminal and $\pi_D(s'_i) = c$ (i.e., $\phi(s'_i)^T \cdot \mathbf{w} \geq g(s'_i)$):
Substitute $\phi(s'_i)^T \cdot \mathbf{w}^*$ for $\hat{Q}(s'_i, \pi_D(s'_i); \mathbf{w}^*)$ in Equation (2)
 - C2: If s'_i is a terminal state or $\pi_D(s'_i) = e$ (i.e., $g(s'_i) > \phi(s'_i)^T \cdot \mathbf{w}$):
Substitute $g(s'_i)$ for $\hat{Q}(s'_i, \pi_D(s'_i); \mathbf{w}^*)$ in Equation (2)
- So rewrite Equation (2) using indicator notation for cases C1, C2 as:

$$\sum_i \phi(s_i) \cdot (\phi(s_i)^T \cdot \mathbf{w}^* - \mathbb{I}_{C1} \cdot \gamma \cdot \phi(s'_i)^T \cdot \mathbf{w}^* - \mathbb{I}_{C2} \cdot \gamma \cdot g(s'_i)) = 0$$

- Factoring out \mathbf{w}^* , we get:

$$(\sum_i \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{C1} \cdot \gamma \cdot \phi(s'_i)))^T \cdot \mathbf{w}^* = \gamma \cdot \sum_i \mathbb{I}_{C2} \cdot \phi(s_i) \cdot g(s'_i)$$

LSPI customized for American Options Pricing

- This can be written in the familiar vector-matrix notation: $\mathbf{A} \cdot \mathbf{w}^* = \mathbf{b}$

$$\mathbf{A} = \sum_i \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{C1} \cdot \gamma \cdot \phi(s'_i))^T$$

$$\mathbf{b} = \gamma \cdot \sum_i \mathbb{I}_{C2} \cdot \phi(s_i) \cdot g(s'_i)$$

- $m \times m$ Matrix \mathbf{A} is accumulated at each atomic experience (s_i, s'_i) as:

$$\mathbf{A} \leftarrow \mathbf{A} + \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{C1} \cdot \gamma \cdot \phi(s'_i))^T$$

- m -Vector \mathbf{b} is accumulated at each atomic experience (s_i, s'_i) as:

$$\mathbf{b} \leftarrow \mathbf{b} + \gamma \cdot \mathbb{I}_{C2} \cdot \phi(s_i) \cdot g(s'_i)$$

- Sherman-Morrison incremental inverse of \mathbf{A} can be done in $O(m^2)$

Feature functions

- Li, Szepesvari, Schuurmans recommend Laguerre polynomials (first 4)
- Over $M_t = S_t/K$ where S_t is underlying price and K is strike
- $\phi_0(S_t) = 1, \phi_1(S_t) = e^{-\frac{M_t}{2}}, \phi_2(S_t) = e^{-\frac{M_t}{2}} \cdot (1 - M_t), \phi_3(S_t) = e^{-\frac{M_t}{2}} \cdot (1 - 2M_t + M_t^2/2)$
- They used these for Longstaff-Schwartz as well as for LSPI
- For LSPI, we also need feature functions for time
- They recommend
 $\phi_0^{(t)}(t) = \sin(\frac{\pi(T-t)}{2T}), \phi_1^{(t)}(t) = \log(T - t), \phi_2^{(t)}(t) = (\frac{t}{T})^2$

Deep Q-Learning for American Pricing

- LSPI is data-efficient/compute-efficient, but linearity is a limitation
- Alternative is (incremental) Q-Learning with neural network approx
- We employ the same set up as LSPI (including Experience-Replay)

$$\hat{Q}(s, a; \mathbf{w}) = \begin{cases} f(s; \mathbf{w}) & \text{if } a = c \\ g(s) & \text{if } a = e \end{cases}$$

where $f(s; \mathbf{w})$ is the deep neural network function approximation

- Q-Learning update for each atomic experience (s_i, s'_i)

$$\Delta \mathbf{w} = \alpha \cdot (\gamma \cdot \hat{Q}(s'_i, \pi(s'_i); \mathbf{w}) - f(s_i; \mathbf{w})) \cdot \nabla_{\mathbf{w}} f(s_i; \mathbf{w})$$

- When s'_i is a non-terminal state, the update is:

$$\Delta \mathbf{w} = \alpha \cdot (\gamma \cdot \max(g(s'_i), f(s'_i; \mathbf{w})) - f(s_i; \mathbf{w})) \cdot \nabla_{\mathbf{w}} f(s_i; \mathbf{w})$$

- When s'_i is a terminal state, the update is:

$$\Delta \mathbf{w} = \alpha \cdot (\gamma \cdot g(s'_i) - f(s_i; \mathbf{w})) \cdot \nabla_{\mathbf{w}} f(s_i; \mathbf{w})$$

Motivation for understanding Value Function Geometry

- Helps us better understand transformations of Value Functions (VFs)
- Across the various DP and RL algorithms
- Particularly helps when VFs are approximated, esp. with linear approx
- Provides insights into stability and convergence
- Particularly when dealing with the “Deadly Triad”
- Deadly Triad $:=$ [Bootstrapping, Func Approx, Off-Policy]
- **Leads us to Gradient TD**

- Assume finite state space $\mathcal{S} = \mathcal{N} = \{s_1, s_2, \dots, s_n\}$
- Action space \mathcal{A} consisting of finite number of actions
- This exposition can be extended to infinite/continuous spaces
- This exposition is for a fixed (often stochastic) policy denoted $\pi(s, a)$
- VF for a policy π is denoted as $\mathbf{V}^\pi : \mathcal{S} \rightarrow \mathbb{R}$
- m feature functions $\phi_1, \phi_2, \dots, \phi_m : \mathcal{S} \rightarrow \mathbb{R}$
- Feature vector for a state $s \in \mathcal{S}$ denoted as $\phi(s) \in \mathbb{R}^m$
- For linear function approximation of VF with weights $\mathbf{w} = (w_1, w_2, \dots, w_m)$, VF $\mathbf{V}_{\mathbf{w}} : \mathcal{S} \rightarrow \mathbb{R}$ is defined as:

$$\mathbf{V}_{\mathbf{w}}(s) = \phi(s)^T \cdot \mathbf{w} = \sum_{j=1}^m \phi_j(s) \cdot w_j \text{ for any } s \in \mathcal{S}$$

- $\mu_\pi : \mathcal{S} \rightarrow [0, 1]$ denotes the states' probability distribution under π

VF Geometry and VF Linear Approximations

- Consider n -dim space \mathbb{R}^n , with each dim corresponding to a state in \mathcal{S}
- Think of a VF (typically denoted \mathbf{V}): $\mathcal{S} \rightarrow \mathbb{R}$ as a vector in this space
- Each dimension's coordinate is the VF for that dimension's state
- Coordinates of vector \mathbf{V}^π for policy π are: $[\mathbf{V}^\pi(s_1), \dots, \mathbf{V}^\pi(s_n)]$
- Consider m independent vectors with j^{th} vector: $[\phi_j(s_1), \dots, \phi_j(s_n)]$
- These m vectors are the m columns of $n \times m$ matrix $\Phi = [\phi_j(s_i)]$
- Their span represents m -dim subspace within this n -dim space
- Spanned by the set of all $\mathbf{w} = [w_1, w_2, \dots, w_m] \in \mathbb{R}^m$
- Vector $\mathbf{V}_{\mathbf{w}} = \Phi \cdot \mathbf{w}$ in this subspace has coordinates $[\mathbf{V}_{\mathbf{w}}(s_1), \dots, \mathbf{V}_{\mathbf{w}}(s_n)]$
- Vector $\mathbf{V}_{\mathbf{w}}$ is fully specified by \mathbf{w} (so we often say \mathbf{w} to mean $\mathbf{V}_{\mathbf{w}}$)

Some more notation

- Denote $\mathcal{R}(s, a)$ as the Expected Reward upon action a in state s
- Denote $\mathcal{P}(s, a, s')$ as the probability of transition $s \rightarrow s'$ upon action a
- Define

$$\mathcal{R}^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot \mathcal{R}(s, a)$$

$$\mathcal{P}^\pi(s, s') = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot \mathcal{P}(s, a, s')$$

- Notation \mathcal{R}^π refers to vector $[\mathcal{R}^\pi(s_1), \mathcal{R}^\pi(s_2), \dots, \mathcal{R}^\pi(s_n)]$
- Notation \mathcal{P}^π refers to matrix $[\mathcal{P}^\pi(s_i, s_{i'})], 1 \leq i, i' \leq n$
- Denote $\gamma < 1$ as the MDP discount factor

Bellman operator B^π

- Bellman Policy Operator B^π for policy π operating on VF vector \mathbf{V} :

$$B^\pi(\mathbf{V}) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \mathbf{V}$$

- B^π is an affine transformation in vector space \mathbb{R}^n
- We lighten notation for application of B^π operator to $B^\pi \cdot \mathbf{V}$
- Note that \mathbf{V}^π is the fixed point of B^π , i.e.,

$$B^\pi \cdot \mathbf{V}^\pi = \mathbf{V}^\pi$$

- If we start with an arbitrary VF vector \mathbf{V} and repeatedly apply B^π , by Fixed-Point Theorem, we will reach the fixed point \mathbf{V}^π
- This is the Dynamic Programming Policy Evaluation algorithm
- Monte Carlo without func approx also converges to \mathbf{V}^π (albeit slowly)

Projection operator Π_{Φ}

- First we define “distance” $d(\mathbf{V}_1, \mathbf{V}_2)$ between VF vectors $\mathbf{V}_1, \mathbf{V}_2$
- Weighted by μ_{π} across the n dimensions of $\mathbf{V}_1, \mathbf{V}_2$

$$d(\mathbf{V}_1, \mathbf{V}_2) = \sum_{i=1}^n \mu_{\pi}(s_i) \cdot (\mathbf{V}_1(s_i) - \mathbf{V}_2(s_i))^2 = (\mathbf{V}_1 - \mathbf{V}_2)^T \cdot \mathbf{D} \cdot (\mathbf{V}_1 - \mathbf{V}_2)$$

where \mathbf{D} is the square diagonal matrix consisting of $\mu_{\pi}(s_i), 1 \leq i \leq n$

- Projection operator for subspace spanned by Φ is denoted as Π_{Φ}
- Π_{Φ} performs an orthogonal projection of VF vector \mathbf{V} on subspace Φ
- So, $\Pi_{\Phi}(\mathbf{V})$ is the VF in subspace Φ defined by $\arg \min_{\mathbf{w}} d(\mathbf{V}, \mathbf{V}_{\mathbf{w}})$
- This is a weighted least squares regression with solution:

$$\mathbf{w} = (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot \Phi^T \cdot \mathbf{D} \cdot \mathbf{V}$$

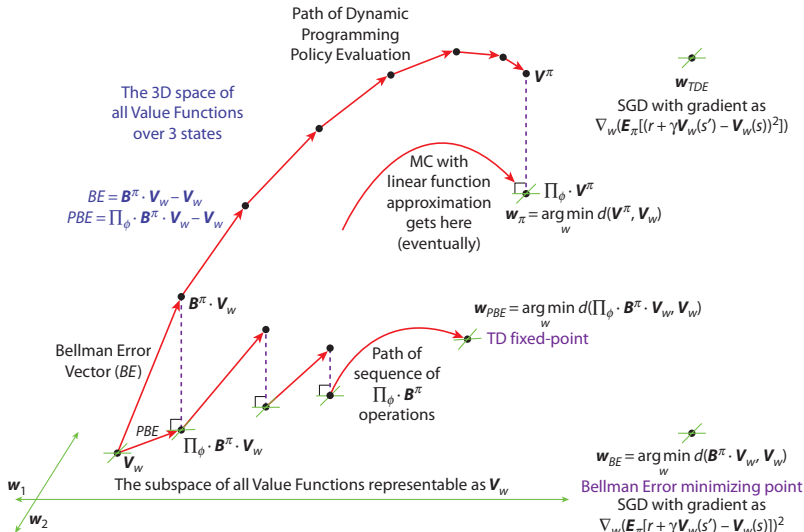
- So, we denote and treat Projection operator Π_{Φ} as a $n \times n$ matrix:

$$\Pi_{\Phi} = \Phi \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot \Phi^T \cdot \mathbf{D}$$

4 VF vectors of interest in the Φ subspace

Note: We will refer to the Φ -subspace VF vectors by their weights \mathbf{w}

- ① Projection $\Pi_{\Phi} \cdot \mathbf{V}^{\pi}$ yields $\mathbf{w}_{\pi} = \arg \min_{\mathbf{w}} d(\mathbf{V}^{\pi}, \mathbf{V}_{\mathbf{w}})$
 - This is the VF we seek when doing linear function approximation
 - Because it is the VF vector “closest” to \mathbf{V}^{π} in the Φ subspace
 - Monte-Carlo with linear func approx will (slowly) converge to \mathbf{w}_{π}
- ② Bellman Error (BE)-minimizing: $\mathbf{w}_{BE} = \arg \min_{\mathbf{w}} d(\mathbf{B}^{\pi} \cdot \mathbf{V}_{\mathbf{w}}, \mathbf{V}_{\mathbf{w}})$
- ③ Temporal Difference Error (TDE)-minimizing:
 $\mathbf{w}_{TDE} = \arg \min_{\mathbf{w}} \mathbb{E}_{\pi}[\delta^2]$
- ④ Projected Bellman Error (PBE)-minimizing:
 $\mathbf{w}_{PBE} = \arg \min_{\mathbf{w}} d((\Pi_{\Phi} \cdot \mathbf{B}^{\pi}) \cdot \mathbf{V}_{\mathbf{w}}, \mathbf{V}_{\mathbf{w}})$



Bellman Error (BE)-minimizing \mathbf{w}_{BE}

\mathbf{w}_{BE} is the vector in the Φ subspace for which the Bellman Error $\mathbf{B}^\pi \cdot \mathbf{V}_w - \mathbf{V}_w$ is minimized

$$\begin{aligned}\mathbf{w}_{BE} &= \arg \min_{\mathbf{w}} d(\mathbf{B}^\pi \cdot \mathbf{V}_w, \mathbf{V}_w) \\ &= \arg \min_{\mathbf{w}} d(\mathbf{V}_w, \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \mathbf{V}_w) \\ &= \arg \min_{\mathbf{w}} d(\Phi \cdot \mathbf{w}, \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \Phi \cdot \mathbf{w}) \\ &= \arg \min_{\mathbf{w}} d(\Phi \cdot \mathbf{w} - \gamma \mathcal{P}^\pi \cdot \Phi \cdot \mathbf{w}, \mathcal{R}^\pi) \\ &= \arg \min_{\mathbf{w}} d((\Phi - \gamma \mathcal{P}^\pi \cdot \Phi) \cdot \mathbf{w}, \mathcal{R}^\pi)\end{aligned}$$

This is a weighted least-squares linear regression of \mathcal{R}^π versus $\Phi - \gamma \mathcal{P}^\pi \cdot \Phi$ with weights μ_π , whose solution is:

$$\mathbf{w}_{BE} = ((\Phi - \gamma \mathcal{P}^\pi \cdot \Phi)^T \cdot \mathbf{D} \cdot (\Phi - \gamma \mathcal{P}^\pi \cdot \Phi))^{-1} \cdot (\Phi - \gamma \mathcal{P}^\pi \cdot \Phi)^T \cdot \mathbf{D} \cdot \mathcal{R}^\pi$$

Model-Free Learning of \mathbf{w}_{BE}

- Let us refer to $(\Phi - \gamma \mathcal{P}^\pi \cdot \Phi)^T \cdot \mathbf{D} \cdot (\Phi - \gamma \mathcal{P}^\pi \cdot \Phi)$ as \mathbf{A}
- Let us refer to $(\Phi - \gamma \mathcal{P}^\pi \cdot \Phi)^T \cdot \mathbf{D} \cdot \mathcal{R}^\pi$ as \mathbf{b}
- So that $\mathbf{w}_{BE} = \mathbf{A}^{-1} \cdot \mathbf{b}$
- Following policy π , each time we perform a model-free transition from s to s' getting reward r , we get a sample estimate of \mathbf{A} and \mathbf{b}
- Estimate of \mathbf{A} is the outer-product of vector $\phi(s) - \gamma \cdot \phi(s')$ with itself
- Estimate of \mathbf{b} is scalar r times vector $\phi(s) - \gamma \cdot \phi(s')$
- Average these estimates across many such model-free transitions
- However, this requires m (number of features) to not be too large

Residual Gradient Algorithm to solve for \mathbf{w}_{BE}

- \mathbf{w}_{BE} is the vector in the Φ subspace for which BE is minimized
- But BE for a state is the expected TD error δ in that state when following policy π
- So we want to do SGD with gradient of square of expected TD error

$$\begin{aligned}\Delta \mathbf{w} &= -\frac{1}{2}\alpha \cdot \nabla_{\mathbf{w}}(\mathbb{E}_{\pi}[\delta])^2 \\ &= -\alpha \cdot \mathbb{E}_{\pi}[r + \gamma \cdot \phi(s')^T \cdot \mathbf{w} - \phi(s)^T \cdot \mathbf{w}] \cdot \nabla_{\mathbf{w}}\mathbb{E}_{\pi}[\delta] \\ &= \alpha \cdot (\mathbb{E}_{\pi}[r + \gamma \cdot \phi(s')^T \cdot \mathbf{w}] - \phi(s)^T \cdot \mathbf{w}) \cdot (\phi(s) - \gamma \cdot \mathbb{E}_{\pi}[\phi(s')])\end{aligned}$$

- This is called the *Residual Gradient* algorithm
- Requires two independent samples of s' transitioning from s
- In that case, converges to \mathbf{w}_{BE} robustly (even for non-linear approx)
- But it is slow, and doesn't converge to a desirable place
- Cannot learn if we can only access features, and not underlying states

Temporal Difference Error (TDE)-minimizing \mathbf{w}_{TDE}

- \mathbf{w}_{TDE} is the vector in the Φ subspace for which the expected square of the TD error δ (when following policy π) is minimized

$$\mathbf{w}_{TDE} = \arg \min_{\mathbf{w}} \sum_{s \in \mathcal{S}} \mu_{\pi}(s) \sum_{r, s'} \mathbb{P}_{\pi}(r, s' | s) \cdot (r + \gamma \cdot \phi(s')^T \cdot \mathbf{w} - \phi(s)^T \cdot \mathbf{w})^2$$

- To perform SGD, we have to estimate the gradient of the expected square of TD error by sampling
- The weight update for each sample in the SGD will be:

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \cdot \nabla_{\mathbf{w}} (r + \gamma \cdot \phi(s')^T \cdot \mathbf{w} - \phi(s)^T \cdot \mathbf{w})^2 \\ &= \alpha \cdot (r + \gamma \cdot \phi(s')^T \cdot \mathbf{w} - \phi(s)^T \cdot \mathbf{w}) \cdot (\phi(s) - \gamma \cdot \phi(s')) \end{aligned}$$

- This algorithm (named *Naive Residual Gradient*) converges robustly, but not to a desirable place

Projected Bellman Error (PBE)-minimizing vector \mathbf{w}_{PBE}

- Consider the composition of Projection Operator Π_{Φ} and Bellman Policy Operator \mathbf{B}^{π} , i.e., $\Pi_{\Phi} \cdot \mathbf{B}^{\pi}$
- We call $\Pi_{\Phi} \cdot \mathbf{B}^{\pi}$ the Projected Bellman Operator
- Applying \mathbf{B}^{π} on a VF vector $\mathbf{V}_{\mathbf{w}}$ in the Φ subspace typically throws it out of the Φ subspace
- Then further applying Π_{Φ} brings it back into the Φ subspace
- Call this resultant VF vector in the Φ subspace as $\mathbf{V}_{\mathbf{w}'}$
- Define \mathbf{w}_{BE} as the \mathbf{w} for which $d(\mathbf{V}_{\mathbf{w}'}, \mathbf{V}_{\mathbf{w}})$ is minimized
- Projected Bellman Error (PBE)-minimizing:
$$\mathbf{w}_{PBE} = \arg \min_{\mathbf{w}} d((\Pi_{\Phi} \cdot \mathbf{B}^{\pi}) \cdot \mathbf{V}_{\mathbf{w}}, \mathbf{V}_{\mathbf{w}})$$
- The minimum is 0, i.e., $\Phi \cdot \mathbf{w}_{PBE}$ is the fixed point of $\Pi_{\Phi} \cdot \mathbf{B}^{\pi}$
- Starting with an arbitrary VF vector \mathbf{V} and repeatedly applying \mathbf{B}^{π} (potentially taking it out of the subspace) followed by Π_{Φ} (projecting it back to the subspace), we will reach the fixed point $\Phi \cdot \mathbf{w}_{PBE}$

Solution of \mathbf{w}_{PBE} with a Linear System Formulation

$\Phi \cdot \mathbf{w}_{PBE}$ is the fixed point of operator $\Pi_\Phi \cdot \mathbf{B}^\pi$. We know:

$$\Pi_\Phi = \Phi \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot \Phi^T \cdot \mathbf{D}$$

$$\mathbf{B}^\pi(\mathbf{V}) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \mathbf{V}$$

Therefore,

$$\Phi \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot \Phi^T \cdot \mathbf{D} \cdot (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \Phi \cdot \mathbf{w}_{PBE}) = \Phi \cdot \mathbf{w}_{PBE}$$

Since columns of Φ are assumed to be independent (full rank),

$$(\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot \Phi^T \cdot \mathbf{D} \cdot (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \Phi \cdot \mathbf{w}_{PBE}) = \mathbf{w}_{PBE}$$

$$\Phi^T \cdot \mathbf{D} \cdot (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \Phi \cdot \mathbf{w}_{PBE}) = \Phi^T \cdot \mathbf{D} \cdot \Phi \cdot \mathbf{w}_{PBE}$$

$$\Phi^T \cdot \mathbf{D} \cdot (\Phi - \gamma \mathcal{P}^\pi \cdot \Phi) \cdot \mathbf{w}_{PBE} = \Phi^T \cdot \mathbf{D} \cdot \mathcal{R}^\pi$$

This is a square linear system of the form $\mathbf{A} \cdot \mathbf{w}_{PBE} = \mathbf{b}$ whose solution is:

$$\mathbf{w}_{PBE} = \mathbf{A}^{-1} \cdot \mathbf{b} = (\Phi^T \cdot \mathbf{D} \cdot (\Phi - \gamma \mathcal{P}^\pi \cdot \Phi))^{-1} \cdot \Phi^T \cdot \mathbf{D} \cdot \mathcal{R}^\pi$$

Model-Free Learning of \mathbf{w}_{PBE}

- How do we construct matrix $\mathbf{A} = \Phi^T \cdot \mathbf{D} \cdot (\Phi - \gamma \mathcal{P}^\pi \cdot \Phi)$ and vector $\mathbf{b} = \Phi^T \cdot \mathbf{D} \cdot \mathcal{R}^\pi$ without a model?
- Following policy π , each time we perform a model-free transition from s to s' getting reward r , we get a sample estimate of \mathbf{A} and \mathbf{b}
- Estimate of \mathbf{A} is outer-product of vectors $\phi(s)$ and $\phi(s) - \gamma \cdot \phi(s')$
- Estimate of \mathbf{b} is scalar r times vector $\phi(s)$
- Average these estimates across many such model-free transitions
- This algorithm is called Least Squares Temporal Difference (LSTD)
- Alternative: Our usual Semi-Gradient TD descent with updates:

$$\Delta \mathbf{w} = \alpha \cdot (r + \gamma \cdot \phi(s')^T \cdot \mathbf{w} - \phi(s)^T \cdot \mathbf{w}) \cdot \phi(s)$$

- This converges to \mathbf{w}_{PBE} because $\mathbb{E}_\pi[\Delta \mathbf{w}] = 0$ yields

$$\begin{aligned} \Phi^T \cdot \mathbf{D} \cdot (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \Phi \cdot \mathbf{w} - \Phi \cdot \mathbf{w}) &= 0 \\ \Rightarrow \Phi^T \cdot \mathbf{D} \cdot (\Phi - \gamma \mathcal{P}^\pi \cdot \Phi) \cdot \mathbf{w} &= \Phi^T \cdot \mathbf{D} \cdot \mathcal{R}^\pi \end{aligned}$$

Gradient TD Algorithms to solve for \mathbf{w}_{PBE}

- For on-policy linear func approx, semi-gradient TD works
- For non-linear func approx or off-policy, we need Gradient TD
 - GTD: The original Gradient TD algorithm
 - GTD-2: Second-generation GTD
 - TDC: TD with Gradient correction
- We need to set up the loss function whose gradient will drive SGD

$$\mathbf{w}_{PBE} = \arg \min_{\mathbf{w}} d(\Pi_{\Phi} \cdot \mathbf{B}^{\pi} \cdot \mathbf{V}_{\mathbf{w}}, \mathbf{V}_{\mathbf{w}}) = \arg \min_{\mathbf{w}} d(\Pi_{\Phi} \cdot \mathbf{B}^{\pi} \cdot \mathbf{V}_{\mathbf{w}}, \Pi_{\Phi} \cdot \mathbf{V}_{\mathbf{w}})$$

- So we define the loss function (denoting $\mathbf{B}^{\pi} \cdot \mathbf{V}_{\mathbf{w}} - \mathbf{V}_{\mathbf{w}}$ as $\delta_{\mathbf{w}}$) as:

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= (\Pi_{\Phi} \cdot \delta_{\mathbf{w}})^T \cdot \mathbf{D} \cdot (\Pi_{\Phi} \cdot \delta_{\mathbf{w}}) = \delta_{\mathbf{w}}^T \cdot \Pi_{\Phi}^T \cdot \mathbf{D} \cdot \Pi_{\Phi} \cdot \delta_{\mathbf{w}} \\ &= \delta_{\mathbf{w}}^T \cdot (\Phi \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot \Phi^T \cdot \mathbf{D})^T \cdot \mathbf{D} \cdot (\Phi \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot \Phi^T \cdot \mathbf{D}) \cdot \delta_{\mathbf{w}} \\ &= \delta_{\mathbf{w}}^T \cdot (\mathbf{D} \cdot \Phi \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot \Phi^T) \cdot \mathbf{D} \cdot (\Phi \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot \Phi^T \cdot \mathbf{D}) \cdot \delta_{\mathbf{w}} \\ &= (\delta_{\mathbf{w}}^T \cdot \mathbf{D} \cdot \Phi) \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi) \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot (\Phi^T \cdot \mathbf{D} \cdot \delta_{\mathbf{w}}) \\ &= (\Phi^T \cdot \mathbf{D} \cdot \delta_{\mathbf{w}})^T \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot (\Phi^T \cdot \mathbf{D} \cdot \delta_{\mathbf{w}})\end{aligned}$$

TDC Algorithm to solve for \mathbf{w}_{PBE}

We derive the TDC Algorithm based on $\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w})$

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = 2 \cdot (\nabla_{\mathbf{w}}(\Phi^T \cdot \mathbf{D} \cdot \delta_{\mathbf{w}})^T) \cdot (\Phi^T \cdot \mathbf{D} \cdot \Phi)^{-1} \cdot (\Phi^T \cdot \mathbf{D} \cdot \delta_{\mathbf{w}})$$

Now we express each of these 3 terms as expectations of model-free transitions $s \xrightarrow{\pi} (r, s')$, denoting $r + \gamma \cdot \phi(s')^T \cdot \mathbf{w} - \phi(s)^T \cdot \mathbf{w}$ as δ

- $\Phi^T \cdot \mathbf{D} \cdot \delta_{\mathbf{w}} = \mathbb{E}[\delta \cdot \phi(s)]$
- $\nabla_{\mathbf{w}}(\Phi^T \cdot \mathbf{D} \cdot \delta_{\mathbf{w}})^T = \mathbb{E}[(\nabla_{\mathbf{w}}\delta) \cdot \phi(s)^T] = \mathbb{E}[(\gamma \cdot \phi(s') - \phi(s)) \cdot \phi(s)^T]$
- $\Phi^T \cdot \mathbf{D} \cdot \Phi = \mathbb{E}[\phi(s) \cdot \phi(s)^T]$

Substituting, we get:

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = 2 \cdot \mathbb{E}[(\gamma \cdot \phi(s') - \phi(s)) \cdot \phi(s)^T] \cdot \mathbb{E}[\phi(s) \cdot \phi(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \phi(s)]$$

Weight Updates of TDC Algorithm

$$\begin{aligned}\Delta \mathbf{w} &= -\frac{1}{2}\alpha \cdot \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \\ &= \alpha \cdot \mathbb{E}[(\phi(s) - \gamma \cdot \phi(s')) \cdot \phi(s)^T] \cdot \mathbb{E}[\phi(s) \cdot \phi(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \phi(s)] \\ &= \alpha \cdot (\mathbb{E}[\phi(s) \cdot \phi(s)^T] - \gamma \cdot \mathbb{E}[\phi(s') \cdot \phi(s)^T]) \cdot \mathbb{E}[\phi(s) \cdot \phi(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \phi(s)] \\ &= \alpha \cdot (\mathbb{E}[\delta \cdot \phi(s)] - \gamma \cdot \mathbb{E}[\phi(s') \cdot \phi(s)^T] \cdot \mathbb{E}[\phi(s) \cdot \phi(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \phi(s)]) \\ &= \alpha \cdot (\mathbb{E}[\delta \cdot \phi(s)] - \gamma \cdot \mathbb{E}[\phi(s') \cdot \phi(s)^T] \cdot \boldsymbol{\theta})\end{aligned}$$

$\boldsymbol{\theta} = \mathbb{E}[\phi(s) \cdot \phi(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \phi(s)]$ is the solution to weighted least-squares linear regression of $\mathbf{B}^\pi \cdot \mathbf{V} - \mathbf{V}$ against Φ , with weights as μ_π .

Cascade Learning: Update both \mathbf{w} and $\boldsymbol{\theta}$ ($\boldsymbol{\theta}$ converging faster)

- $\Delta \mathbf{w} = \alpha \cdot \delta \cdot \phi(s) - \alpha \cdot \gamma \cdot \phi(s') \cdot (\boldsymbol{\theta}^T \cdot \phi(s))$
- $\Delta \boldsymbol{\theta} = \beta \cdot (\delta - \boldsymbol{\theta}^T \cdot \phi(s)) \cdot \phi(s)$

Note: $\boldsymbol{\theta}^T \cdot \phi(s)$ operates as estimate of TD error δ for current state s

Key Takeaways from this Chapter

- Batch RL makes efficient use of data
- DQN uses Experience-Replay and fixed Q-learning targets, avoiding the pitfalls of time-correlation and semi-gradient
- LSTD is a direct (gradient-free) solution of Batch TD Prediction
- LSPI is an off-policy, experience-replay Control Algorithm using LSTDQ for Policy Evaluation
- Optimal Exercise of American Options can be tackled with LSPI and Deep Q-Learning algorithms
- Value Function Geometry provides tremendous intuition
- Projected Bellman Error (PBE) is the right loss function to use
- The gradient of PBE loss function yields Gradient TD algorithms