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 $V(s; \boldsymbol{w})$ using $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ for each data pair:

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

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

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

- n updates are performed in sequence for $i = 1, 2, \ldots, n$
- 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 $V(s; \boldsymbol{w}^*)$ such that

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{2} \cdot (V(S_i; \boldsymbol{w}) - G_i)^2$$

$$= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(S,G)\sim\mathcal{D}}[\frac{1}{2} \cdot (V(S; \boldsymbol{w}) - G)^2]$$

- 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 $(S, G) \sim \mathcal{D}$

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

- This will ultimately converge to desired value function $V(s; \boldsymbol{w}^*)$
- 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 $(S, R, S') \sim \mathcal{D}$

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

- 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

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

- 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:

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

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

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 $Q(S_t, a; \boldsymbol{w})$
- 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 $\boldsymbol{w}$ using Q-learning targets based on "frozen" parameters $\boldsymbol{w}^-$ of *target network*

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

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

Parameters $\boldsymbol{w}^-$ of target network infrequently updated to values of Q-network parameters $\boldsymbol{w}$ (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 $\phi_j : \mathcal{S} \to \mathbb{R}, j = 1, 2, \ldots, m$
- Parameters w is a weights vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_m) \in \mathbb{R}^m$
- Value Function is approximated as:

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

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

Least-Squares Monte-Carlo (LSMC)

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

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

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

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

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

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

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

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

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

Least-Squares Temporal-Difference (LSTD)

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

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

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

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

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

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

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

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

- 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 $\boldsymbol{E}_i$
- Note: $\boldsymbol{E}_i$ accumulates $\nabla_{\boldsymbol{w}} V(s; \boldsymbol{w}) = \phi(s)$ in each trace experience
- When accumulating, previous step's eligibility traces discounted by $\lambda\gamma$

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

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

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

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

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{E_i} \cdot R_i$$

- 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):

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

- Direct solve for $\boldsymbol{w}^*$ 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 $\boldsymbol{w}^*$ to minimize:

$$\mathcal{L}(\boldsymbol{w}) = \sum_i (Q(s_i, a_i; \boldsymbol{w}) - (r_i + \gamma \cdot Q(s_i', \pi_D(s_i'); \boldsymbol{w})))^2$$
$$= \sum_i (\phi(s_i, a_i)^T \cdot \boldsymbol{w} - (r_i + \gamma \cdot \phi(s_i', \pi_D(s_i'))^T \cdot \boldsymbol{w}))^2$$

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

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

Solving for weights $\boldsymbol{w}^*$ with LSTDQ

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

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

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

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

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

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

- Sherman-Morrison incremental inverse can be done in $O(m^2)$
- This least-squares solution of $\boldsymbol{w}^*$ (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., $Q(s, a) = g(s)$

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

  for feature funcs $\phi(\cdot) = [\phi_i(\cdot) | i = 1, \ldots, m]$ 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 $Q(s, a; \boldsymbol{w}^*)$
- 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:

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

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

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

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

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

LSPI customized for American Options Pricing

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

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

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

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

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

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

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

- Sherman-Morrison incremental inverse of $\boldsymbol{A}$ 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)

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

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

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

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

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

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

- When s_i' is a terminal state, the update is:

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

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**

Notation

- Assume finite state space $\mathcal{S} = \mathcal{N} = \{s_1, s_2, \ldots, s_n\}$
- 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 $\boldsymbol{V}^{\pi} : \mathcal{S} \to \mathbb{R}$
- m feature functions $\phi_1, \phi_2, \ldots, \phi_m : \mathcal{S} \to \mathbb{R}$
- Feature vector for a state $s \in \mathcal{S}$ denoted as $\boldsymbol{\phi}(s) \in \mathbb{R}^m$
- For linear function approximation of VF with weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_m)$, VF $\boldsymbol{V_w} : \mathcal{S} \to \mathbb{R}$ is defined as:

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

- $\boldsymbol{\mu_\pi} : \mathcal{S} \to [0, 1]$ 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 $\boldsymbol{V}$): $\mathcal{S} \to \mathbb{R}$ as a vector in this space
- Each dimension's coordinate is the VF for that dimension's state
- Coordinates of vector $\boldsymbol{V}^\pi$ for policy π are: $[\boldsymbol{V}^\pi(s_1), \ldots, \boldsymbol{V}^\pi(s_n)]$
- Consider m independent vectors with j^{th} vector: $[\phi_j(s_1), \ldots, \phi_j(s_n)]$
- These m vectors are the m columns of $n \times m$ matrix $\boldsymbol{\Phi} = [\phi_j(s_i)]$
- Their span represents m-dim subspace within this n-dim space
- Spanned by the set of all $\boldsymbol{w} = [w_1, w_2, \ldots, w_m] \in \mathbb{R}^m$
- Vector $\boldsymbol{V_w} = \boldsymbol{\Phi} \cdot \boldsymbol{w}$ in this subspace has coordinates $[\boldsymbol{V_w}(s_1), \ldots, \boldsymbol{V_w}(s_n)]$
- Vector $\boldsymbol{V_w}$ is fully specified by $\boldsymbol{w}$ (so we often say $\boldsymbol{w}$ to mean $\boldsymbol{V_w}$)

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 $s \to s'$ upon action a
- Define

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

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

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

Bellman operator B^π

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

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

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

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

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

Projection operator $\mathbf{\Pi_\Phi}$

- First we define "distance" $d(\boldsymbol{V_1}, \boldsymbol{V_2})$ between VF vectors $\boldsymbol{V_1}, \boldsymbol{V_2}$
- Weighted by $\boldsymbol{\mu_\pi}$ across the n dimensions of $\boldsymbol{V_1}, \boldsymbol{V_2}$

$$d(\boldsymbol{V_1}, \boldsymbol{V_2}) = \sum_{i=1}^{n} \boldsymbol{\mu_\pi}(s_i) \cdot (\boldsymbol{V_1}(s_i) - \boldsymbol{V_2}(s_i))^2 = (\boldsymbol{V_1} - \boldsymbol{V_2})^T \cdot \boldsymbol{D} \cdot (\boldsymbol{V_1} - \boldsymbol{V_2})$$

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

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

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

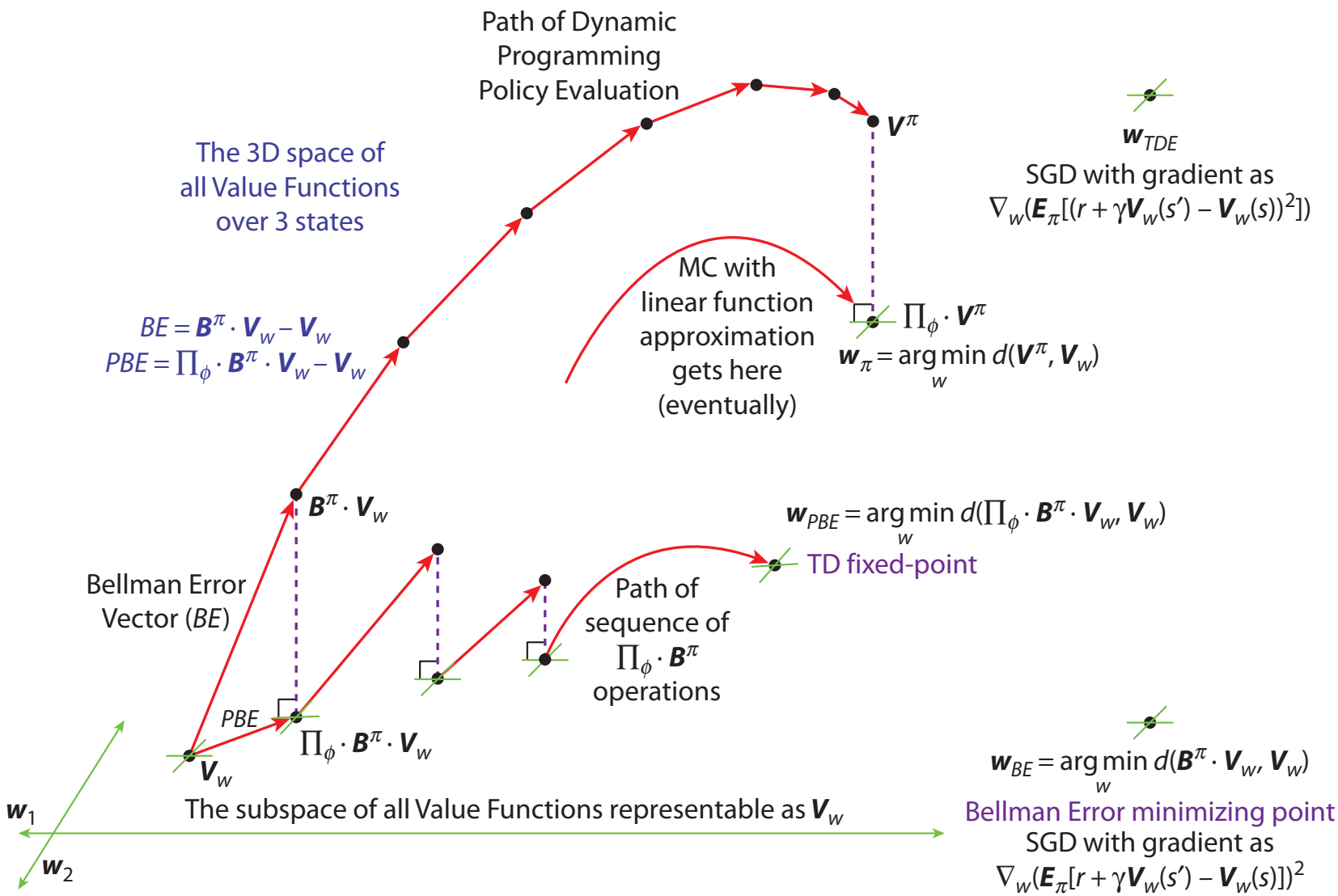
- So, we denote and treat Projection operator $\mathbf{\Pi_\Phi}$ as a $n \times n$ matrix:

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

4 VF vectors of interest in the Φ subspace

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

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


Path of Dynamic Programming Policy Evaluation
V^π
w_{TDE}
SGD with gradient as
$\nabla_w(E_\pi[(r+\gamma V_w(s') - V_w(s))^2])$
The 3D space of all Value Functions over 3 states
MC with linear function approximation gets here (eventually)
$\prod_\phi \cdot V^\pi$
$w_\pi = \arg\min_w d(V^\pi, V_w)$
$BE = B^\pi \cdot V_w - V_w$
$PBE = \prod_\phi \cdot B^\pi \cdot V_w - V_w$
$B^\pi \cdot V_w$
$w_{PBE} = \arg\min_w d(\prod_\phi \cdot B^\pi \cdot V_w, V_w)$
TD fixed-point
Bellman Error Vector (BE)
Path of sequence of $\prod_\phi \cdot B^\pi$ operations
PBE
$\prod_\phi \cdot B^\pi \cdot V_w$
V_w
w_1
w_2
The subspace of all Value Functions representable as V_w
$w_{BE} = \arg\min_w d(B^\pi \cdot V_w, V_w)$
Bellman Error minimizing point
SGD with gradient as
$\nabla_w(E_\pi[r+\gamma V_w(s') - V_w(s)])^2$

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

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

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

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

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

Model-Free Learning of $\boldsymbol{w}_{BE}$

- Let us refer to $(\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{P}}^{\pi} \cdot \boldsymbol{\Phi})^T \cdot \boldsymbol{D} \cdot (\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{P}}^{\pi} \cdot \boldsymbol{\Phi})$ as $\boldsymbol{A}$
- Let us refer to $(\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{P}}^{\pi} \cdot \boldsymbol{\Phi})^T \cdot \boldsymbol{D} \cdot \boldsymbol{\mathcal{R}}^{\pi}$ as $\boldsymbol{b}$
- So that $\boldsymbol{w}_{BE} = \boldsymbol{A}^{-1} \cdot \boldsymbol{b}$
- Following policy π, each time we perform a model-free transition from s to s' getting reward r, we get a sample estimate of $\boldsymbol{A}$ and $\boldsymbol{b}$
- Estimate of $\boldsymbol{A}$ is the outer-product of vector $\phi(s) - \gamma \cdot \phi(s')$ with itself
- Estimate of $\boldsymbol{b}$ 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 $\boldsymbol{w}_{BE}$

- $\boldsymbol{w}_{BE}$ is the vector in the $\boldsymbol{\Phi}$ 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

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

- This is called the *Residual Gradient* algorithm
- Requires two independent samples of s' transitioning from s
- In that case, converges to $\boldsymbol{w}_{BE}$ 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 $\boldsymbol{w}_{TDE}$

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

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

- 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:

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

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

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

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

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

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

$$\boldsymbol{\Pi}_{\boldsymbol{\Phi}} = \boldsymbol{\Phi} \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi})^{-1} \cdot \boldsymbol{\Phi}^T \cdot \boldsymbol{D}$$

$$\boldsymbol{B}^{\pi} \cdot \boldsymbol{V} = \boldsymbol{\mathcal{R}}^{\pi} + \gamma \boldsymbol{\mathcal{P}}^{\pi} \cdot \boldsymbol{V}$$

Therefore,

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

Since columns of $\boldsymbol{\Phi}$ are assumed to be independent (full rank),

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

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

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

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

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

Model-Free Learning of $\boldsymbol{w}_{PBE}$

- How do we construct matrix $\boldsymbol{A} = \boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot (\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{\Phi})$ and vector $\boldsymbol{b} = \boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\mathcal{R}}^\pi$ 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 $\boldsymbol{A}$ and $\boldsymbol{b}$
- Estimate of $\boldsymbol{A}$ is outer-product of vectors $\boldsymbol{\phi}(s)$ and $\boldsymbol{\phi}(s) - \gamma \cdot \boldsymbol{\phi}(s')$
- Estimate of $\boldsymbol{b}$ is scalar r times vector $\boldsymbol{\phi}(s)$
- 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:

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

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

$$\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot (\boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{\Phi} \cdot \boldsymbol{w} - \boldsymbol{\Phi} \cdot \boldsymbol{w}) = 0$$

$$\Rightarrow \boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot (\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{\Phi}) \cdot \boldsymbol{w} = \boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\mathcal{R}}^\pi$$

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

- 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

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

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

$$\mathcal{L}(\mathbf{w}) = (\boldsymbol{\Pi}_{\boldsymbol{\Phi}} \cdot \boldsymbol{\delta}_{\boldsymbol{w}})^T \cdot \boldsymbol{D} \cdot (\boldsymbol{\Pi}_{\boldsymbol{\Phi}} \cdot \boldsymbol{\delta}_{\boldsymbol{w}}) = \boldsymbol{\delta}_{\boldsymbol{w}}^T \cdot \boldsymbol{\Pi}_{\boldsymbol{\Phi}}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Pi}_{\boldsymbol{\Phi}} \cdot \boldsymbol{\delta}_{\boldsymbol{w}}$$

$$= \boldsymbol{\delta}_{\boldsymbol{w}}^T \cdot (\boldsymbol{\Phi} \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi})^{-1} \cdot \boldsymbol{\Phi}^T \cdot \boldsymbol{D})^T \cdot \boldsymbol{D} \cdot (\boldsymbol{\Phi} \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi})^{-1} \cdot \boldsymbol{\Phi}^T \cdot \boldsymbol{D}) \cdot \boldsymbol{\delta}_{\boldsymbol{w}}$$

$$= \boldsymbol{\delta}_{\boldsymbol{w}}^T \cdot (\boldsymbol{D} \cdot \boldsymbol{\Phi} \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi})^{-1} \cdot \boldsymbol{\Phi}^T) \cdot \boldsymbol{D} \cdot (\boldsymbol{\Phi} \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi})^{-1} \cdot \boldsymbol{\Phi}^T \cdot \boldsymbol{D}) \cdot \boldsymbol{\delta}_{\boldsymbol{w}}$$

$$= (\boldsymbol{\delta}_{\boldsymbol{w}}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi}) \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi})^{-1} \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi}) \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi})^{-1} \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\delta}_{\boldsymbol{w}})$$

$$= (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\delta}_{\boldsymbol{w}})^T \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi})^{-1} \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\delta}_{\boldsymbol{w}})$$

TDC Algorithm to solve for w_{PBE}

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

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = 2 \cdot (\nabla_{\boldsymbol{w}} (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\delta}_{\boldsymbol{w}})^T) \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Phi})^{-1} \cdot (\boldsymbol{\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\delta}_{\boldsymbol{w}})$$

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

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

Substituting, we get:

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

Weight Updates of TDC Algorithm

$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha \cdot \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$$

$$= \alpha \cdot \mathbb{E}[(\boldsymbol{\phi}(s) - \gamma \cdot \boldsymbol{\phi}(s')) \cdot \boldsymbol{\phi}(s)^T] \cdot \mathbb{E}[\boldsymbol{\phi}(s) \cdot \boldsymbol{\phi}(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \boldsymbol{\phi}(s)]$$

$$= \alpha \cdot (\mathbb{E}[\boldsymbol{\phi}(s) \cdot \boldsymbol{\phi}(s)^T] - \gamma \cdot \mathbb{E}[\boldsymbol{\phi}(s') \cdot \boldsymbol{\phi}(s)^T]) \cdot \mathbb{E}[\boldsymbol{\phi}(s) \cdot \boldsymbol{\phi}(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \boldsymbol{\phi}(s)]$$

$$= \alpha \cdot (\mathbb{E}[\delta \cdot \boldsymbol{\phi}(s)] - \gamma \cdot \mathbb{E}[\boldsymbol{\phi}(s') \cdot \boldsymbol{\phi}(s)^T] \cdot \mathbb{E}[\boldsymbol{\phi}(s) \cdot \boldsymbol{\phi}(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \boldsymbol{\phi}(s)])$$

$$= \alpha \cdot (\mathbb{E}[\delta \cdot \boldsymbol{\phi}(s)] - \gamma \cdot \mathbb{E}[\boldsymbol{\phi}(s') \cdot \boldsymbol{\phi}(s)^T] \cdot \boldsymbol{\theta})$$

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

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

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

Note: $\boldsymbol{\theta}^T \cdot \boldsymbol{\phi}(s)$ 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