A Guided Tour of Chapter 11:
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:

\[ D = [(S_i, G_i)|1 \leq i \leq n] \]

- Incremental MC estimates \( V(s; w) \) using \( \nabla_w L(w) \) for each data pair:

\[
L(S_i, G_i)(w) = \frac{1}{2} \cdot (V(S_i; w) - G_i)^2
\]

\[
\nabla_w L(S_i, G_i)(w) = (V(S_i; w) - G_i) \cdot \nabla_w V(S_i; w)
\]

\[
\Delta w = \alpha \cdot (G_i - V(S_i; w)) \cdot \nabla_w V(S_i; w)
\]

- \( n \) 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 \( D \)
- 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 $V(s; w^*)$ such that

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

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

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

$$\Delta w = \alpha \cdot (G - V(S; w)) \cdot \nabla_w V(S; w)$$

This will ultimately converge to desired value function $V(s; w^*)$

Repeated use of available data known as Experience-Replay

This makes more efficient use of available training data $\mathcal{D}$
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 $(S, R, S') \sim \mathcal{D}$

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

This is TD Prediction with Experience-Replay on Finite Experiences $\mathcal{D}$.
Batch TD(\(\lambda\)) Prediction

- In Batch TD(\(\lambda\)) 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 \(t\):
  \[
  E_t = \gamma \lambda \cdot E_{t-1} + \nabla_w V(S_{i,t}; w)
  \]
  \[
  \Delta w = \alpha \cdot (R_{i,t+1} + \gamma \cdot V(S_{i,t+1}; w) - V(S_{i,t}; w)) \cdot E_t
  \]
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 $\epsilon$-greedy policy extracted from Q-network values $Q(S_t, a; w)$
- 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 $w$ using Q-learning targets based on “frozen” parameters $w^-$ of target network

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

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

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

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

where $\phi(s) \in \mathbb{R}^m$ is the feature vector for state $s$
Loss function for Batch MC Prediction with data \([(S_i, G_i)|1 \leq i \leq n]\):
\[
\mathcal{L}(w) = \frac{1}{2n} \cdot \sum_{i=1}^{n} \left( \sum_{j=1}^{m} \phi_j(S_i) \cdot w_j - G_i \right)^2 = \frac{1}{2n} \cdot \sum_{i=1}^{n} (\phi(S_i)^T \cdot w - G_i)^2
\]

The gradient of this Loss function is set to 0 to solve for \(w^*\)
\[
\sum_{i=1}^{n} \phi(S_i) \cdot (\phi(S_i)^T \cdot w^* - G_i) = 0
\]

\(w^*\) is solved as \(A^{-1} \cdot b\)

\(m \times m\) Matrix \(A\) is accumulated at each data pair \((S_i, G_i)\) as:
\[
A \leftarrow A + \phi(S_i) \cdot \phi(S_i)^T \quad \text{(i.e., outer-product of \(\phi(S_i)\) with itself)}
\]

\(m\)-Vector \(b\) is accumulated at each data pair \((S_i, G_i)\) as:
\[
b \leftarrow b + \phi(S_i) \cdot G_i
\]

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 \((S_i, R_i, S'_i)\):

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

- \(m\)-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 \(O(m^2)\)
Likewise, we can do LSTD($\lambda$) using Eligibility Traces

Denote the Eligibility Traces of atomic experience $i$ as $E_i$

Note: $E_i$ accumulates $\nabla_w V(s; w) = \phi(s)$ in each trace experience

When accumulating, previous step’s eligibility traces discounted by $\lambda \gamma$

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

$w^*$ is solved as $A^{-1} \cdot b$

$m \times m$ Matrix $A$ is accumulated at each atomic experience $(S_i, R_i, S'_i)$:

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

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

$$b \leftarrow b + E_i \cdot R_i$$

Sherman-Morrison incremental inverse can be done in $O(m^2)$
### Convergence of Least Squares Prediction Algorithms

<table>
<thead>
<tr>
<th>On/Off Policy</th>
<th>Algorithm</th>
<th>Tabular</th>
<th>Linear</th>
<th>Non-Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Policy</td>
<td>MC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>LSMC</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TD</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>LSTD</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Off-Policy</td>
<td>MC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>LSMC</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>TD</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>LSTD</td>
<td>✓</td>
<td>✗</td>
<td></td>
</tr>
</tbody>
</table>
To perform Least Squares RL Control, we do GPI with:
- Policy Evaluation as Least-Squares Q-Value Prediction
- Greedy (or $\epsilon$-Greedy) Policy Improvement

For On-Policy MC/TD Control, Q-Value Prediction (for policy $\pi$):

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

Direct solve for $w^*$ using experiences data generated using policy $\pi$
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 $\pi_D$
- $\pi_D$ is made available from the previous iteration of GPI
- Goal of the iteration is to solve for weights $w^*$ to minimize:

$$\mathcal{L}(w) = \sum_i (Q(s_i, a_i; w) - (r_i + \gamma \cdot Q(s'_i, \pi_D(s'_i); w)))^2$$

$$= \sum_i (\phi(s_i, a_i)^T \cdot w - (r_i + \gamma \cdot \phi(s'_i, \pi_D(s'_i))^T \cdot w))^2$$

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

$$\pi_D(s) = \arg \max_a Q(s, a; w^*)$$
Solving for weights $w^*$ with LSTDQ

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

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

- $w^*$ is solved as $A^{-1} \cdot b$

- $m \times m$ Matrix $A$ is accumulated at each experience $(s_i, a_i, r_i, s'_i)$:

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

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

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

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Tabular</th>
<th>Linear</th>
<th>Non-Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC Control</td>
<td>✓</td>
<td>(✓)</td>
<td>✗</td>
</tr>
<tr>
<td>SARSA</td>
<td>✓</td>
<td>(✓)</td>
<td>✗</td>
</tr>
<tr>
<td>Q-Learning</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>LSPI</td>
<td>✓</td>
<td>(✓)</td>
<td>-</td>
</tr>
</tbody>
</table>

(✓) means it chatters around near-optimal Value Function
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., \( Q(s, a) = g(s) \)

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

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 \( \pi_D \)
- \( \pi_D \) is greedy policy from previous iteration’s solved \( Q(s, a; w^*) \)
- Since we learn Q-Value function for only \( a = c \), behavior policy \( \mu \) 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 \( (s, s') \)
This reduces LSPI Semi-Gradient Equation (1) to:

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

We need to consider two cases for the term \(\hat{Q}(s'_i, \pi_D(s'_i); w^*)\)

- **C1:** If \(s'_i\) is non-terminal and \(\pi_D(s'_i) = c\) (i.e., \(\phi(s'_i)^T \cdot w \geq g(s'_i)\)):
  Substitute \(\phi(s'_i)^T \cdot w^*\) for \(\hat{Q}(s'_i, \pi_D(s'_i); w^*)\) in Equation (2)

- **C2:** If \(s'_i\) is a terminal state or \(\pi_D(s'_i) = e\) (i.e., \(g(s'_i) > \phi(s'_i)^T \cdot w\)):
  Substitute \(g(s'_i)\) for \(\hat{Q}(s'_i, \pi_D(s'_i); w^*)\) in Equation (2)

So rewrite Equation (2) using indicator notation for cases C1, C2 as:

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

Factoring out \(w^*\), we get:

$$
(\sum_i \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{C1} \cdot \gamma \cdot \phi(s'_i)))^T \cdot w^* = \gamma \cdot \sum_i \mathbb{I}_{C2} \cdot \phi(s_i) \cdot g(s'_i)
$$
LSPI customized for American Options Pricing

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

\[
\mathbf{A} = \sum_i \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{C_1} \cdot \gamma \cdot \phi(s_i'))^T
\]

\[
\mathbf{b} = \gamma \cdot \sum_i \mathbb{I}_{C_2} \cdot \phi(s_i) \cdot g(s_i')
\]

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

\[
\mathbf{A} \leftarrow \mathbf{A} + \phi(s_i) \cdot (\phi(s_i) - \mathbb{I}_{C_1} \cdot \gamma \cdot \phi(s_i'))^T
\]

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

\[
\mathbf{b} \leftarrow \mathbf{b} + \gamma \cdot \mathbb{I}_{C_2} \cdot \phi(s_i) \cdot g(s_i')
\]

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

$\phi_0(t)(t) = \sin\left(\frac{\pi(T-t)}{2T}\right), \phi_1(t)(t) = \log(T - t), \phi_2(t)(t) = \left(\frac{t}{T}\right)^2$
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)

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

where \( f(s; w) \) is the deep neural network function approximation

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

\[ \Delta w = \alpha \cdot (\gamma \cdot \hat{Q}(s'_i, \pi(s'_i); w) - f(s_i; w)) \cdot \nabla_w f(s_i; w) \]

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

\[ \Delta w = \alpha \cdot (\gamma \cdot \max(g(s'_i), f(s'_i; w)) - f(s_i; w)) \cdot \nabla_w f(s_i; w) \]

When \( s'_i \) is a terminal state, the update is:

\[ \Delta w = \alpha \cdot (\gamma \cdot g(s'_i) - f(s_i; w)) \cdot \nabla_w f(s_i; w) \]
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 $S = \mathcal{N} = \{s_1, s_2, \ldots, s_n\}$

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 $\pi$ is denoted as $V^\pi : S \rightarrow \mathbb{R}$

$m$ feature functions $\phi_1, \phi_2, \ldots, \phi_m : S \rightarrow \mathbb{R}$

Feature vector for a state $s \in S$ denoted as $\phi(s) \in \mathbb{R}^m$

For linear function approximation of VF with weights $\mathbf{w} = (w_1, w_2, \ldots, w_m)$, VF $V_\mathbf{w} : S \rightarrow \mathbb{R}$ is defined as:

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

$\mu_\pi : S \rightarrow [0, 1]$ denotes the states’ probability distribution under $\pi$
VF Geometry and VF Linear Approximations

- Consider \( n \)-dim space \( \mathbb{R}^n \), with each dim corresponding to a state in \( S \)
- Think of a VF (typically denoted \( \mathbf{V} \)): \( S \rightarrow \mathbb{R} \) as a vector in this space
- Each dimension’s coordinate is the VF for that dimension’s state
- Coordinates of vector \( \mathbf{V}^\pi \) for policy \( \pi \) are: \( [\mathbf{V}^\pi(s_1), \ldots, \mathbf{V}^\pi(s_n)] \)
- 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 \( \Phi = [\phi_j(s_i)] \)
- Their span represents \( m \)-dim subspace within this \( n \)-dim space
- Spanned by the set of all \( \mathbf{w} = [w_1, w_2, \ldots, w_m] \in \mathbb{R}^m \)
- Vector \( \mathbf{V}_\mathbf{w} = \Phi \cdot \mathbf{w} \) in this subspace has coordinates \( [\mathbf{V}_\mathbf{w}(s_1), \ldots, \mathbf{V}_\mathbf{w}(s_n)] \)
- 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 $R(s, a)$ as the Expected Reward upon action $a$ in state $s$
- Denote $P(s, a, s')$ as the probability of transition $s \rightarrow s'$ upon action $a$
- Define
  \[
  R^\pi(s) = \sum_{a \in A} \pi(s, a) \cdot R(s, a)
  \]
  \[
  P^\pi(s, s') = \sum_{a \in A} \pi(s, a) \cdot P(s, a, s')
  \]
- Notation $R^\pi$ refers to vector $[R^\pi(s_1), R^\pi(s_2), \ldots, R^\pi(s_n)]$
- Notation $P^\pi$ refers to matrix $[P^\pi(s_i, s_{i'})], 1 \leq i, i' \leq n$
- Denote $\gamma < 1$ as the MDP discount factor
Bellman operator $B^\pi$

- Bellman Policy Operator $B^\pi$ for policy $\pi$ operating on VF vector $V$:
  \[
  B^\pi(V) = R^\pi + \gamma P^\pi \cdot V
  \]

- $B^\pi$ is a linear operator in vector space $\mathbb{R}^n$
- So we denote and treat $B^\pi$ as a $n \times n$ matrix
- Note that $V^\pi$ is the fixed point of $B^\pi$, i.e.,
  \[
  B^\pi \cdot V^\pi = V^\pi
  \]

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

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

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

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

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

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

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

$$\Pi_\Phi = \Phi \cdot (\Phi^T \cdot D \cdot \Phi)^{-1} \cdot \Phi^T \cdot D$$
4 VF vectors of interest in the $\Phi$ subspace

Note: We will refer to the $\Phi$-subspace VF vectors by their weights $w$

1. Projection $\Pi_\Phi \cdot V^\pi$ yields $w_\pi = \arg\min_w d(V^\pi, V_w)$
   - This is the VF we seek when doing linear function approximation
   - Because it is the VF vector “closest” to $V^\pi$ in the $\Phi$ subspace
   - Monte-Carlo with linear func approx will (slowly) converge to $w_\pi$

2. Bellman Error (BE)-minimizing: $w_{BE} = \arg\min_w d(B^\pi \cdot V_w, V_w)$

3. Temporal Difference Error (TDE)-minimizing:
   $$w_{TDE} = \arg\min_w \mathbb{E}_\pi[\delta^2]$$

4. Projected Bellman Error (PBE)-minimizing:
   $$w_{PBE} = \arg\min_w d((\Pi_\Phi \cdot B^\pi) \cdot V_w, V_w)$$
The 3D space of all Value Functions over 3 states

\[ \text{BE} = B_\pi V_w - V_w \]

\[ \text{PBE} = \Pi\phi B_\pi V_w \]

Path of DP Policy Evaluation

MC with linear function approx. gets here eventually

\[ W_\pi = \arg\min_{w} d(V_\pi, V_w) \]

\[ W_{\text{PBE}} = \arg\min_{w} d(\Pi\phi B_\pi V_w, V_w) \]

TD fixed point

The subspace of all Value Functions representable as \( V_w \)

\[ W_{\text{BE}} = \arg\min_{w} d(B_\pi V_w, V_w) \]

Bellman Error minimizing point

\[ \text{SGD with gradient as} \]

\[ \nabla_w (E(x, y) - V_w(x))^2 \]
Bellman Error (BE)-minimizing $w_{BE}$

$w_{BE}$ is the vector in the $\Phi$ subspace for which the Bellman Error $B^\pi \cdot V_w - V_w$ is minimized

$$w_{BE} = \arg \min_w d(B^\pi \cdot V_w, V_w)$$

$$= \arg \min_w d(V_w, R^\pi + \gamma P^\pi \cdot V_w)$$

$$= \arg \min_w d(\Phi \cdot w, R^\pi + \gamma P^\pi \cdot \Phi \cdot w)$$

$$= \arg \min_w d(\Phi \cdot w - \gamma P^\pi \cdot \Phi \cdot w, R^\pi)$$

$$= \arg \min_w d((\Phi - \gamma P^\pi \cdot \Phi) \cdot w, R^\pi)$$

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

$$w_{BE} = ((\Phi - \gamma P^\pi \cdot \Phi)^T \cdot D \cdot (\Phi - \gamma P^\pi \cdot \Phi))^{-1} \cdot (\Phi - \gamma P^\pi \cdot \Phi)^T \cdot D \cdot R^\pi$$
Model-Free Learning of $w_{BE}$

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

- $w_{BE}$ is the vector in the $\Phi$ subspace for which BE is minimized
- But BE for a state is the expected TD error $\delta$ in that state when following policy $\pi$
- So we want to do SGD with gradient of square of expected TD error

$$\Delta w = -\frac{1}{2} \alpha \cdot \nabla_w (\mathbb{E}_\pi [\delta])^2$$

$$= -\alpha \cdot \mathbb{E}_\pi [r + \gamma \cdot \phi(s')^T \cdot w - \phi(s)^T \cdot w] \cdot \nabla_w \mathbb{E}_\pi [\delta]$$

$$= \alpha \cdot (\mathbb{E}_\pi [r + \gamma \cdot \phi(s')^T \cdot w] - \phi(s)^T \cdot w) \cdot (\phi(s) - \gamma \cdot \mathbb{E}_\pi [\phi(s')])$$

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

- \( w_{TDE} \) is the vector in the \( \Phi \) subspace for which the expected square of the TD error \( \delta \) (when following policy \( \pi \)) is minimized

\[
w_{TDE} = \arg \min_w \sum_{s \in S} \mu_\pi(s) \sum_{r,s'} \mathbb{P}_\pi(r, s'|s) \cdot (r + \gamma \cdot \phi(s'))^T \cdot w - \phi(s)^T \cdot w)^2
\]

- 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:

\[
\Delta w = -\frac{1}{2} \alpha \cdot \nabla_w (r + \gamma \cdot \phi(s'))^T \cdot w - \phi(s)^T \cdot w)^2
\]

\[
= \alpha \cdot (r + \gamma \cdot \phi(s'))^T \cdot w - \phi(s)^T \cdot w) \cdot (\phi(s) - \gamma \cdot \phi(s'))
\]

- This algorithm (named Naive Residual Gradient) converges robustly, but not to a desirable place
Consider the composition of Projection Operator $\Pi_\Phi$ and Bellman Policy Operator $B^\pi$, i.e., $\Pi_\Phi \cdot B^\pi$

We call $\Pi_\Phi \cdot B^\pi$ the Projected Bellman Operator

Applying $B^\pi$ on a VF vector $V_w$ in the $\Phi$ subspace typically throws it out of the $\Phi$ subspace

Then further applying $\Pi_\Phi$ brings it back into the $\Phi$ subspace

Call this resultant VF vector in the $\Phi$ subspace as $V_w'$

Define $w_{BE}$ as the $w$ for which $d(V_{w'}, V_w)$ is minimized

Projected Bellman Error (PBE)-minimizing:

$$w_{PBE} = \arg \min_w d((\Pi_\Phi \cdot B^\pi) \cdot V_w, V_w)$$

The minimum is 0, i.e., $\Phi \cdot w_{PBE}$ is the fixed point of $\Pi_\Phi \cdot B^\pi$

Starting with an arbitrary VF vector $V$ and repeatedly applying $B^\pi$ (potentially taking it out of the subspace) followed by $\Pi_\Phi$ (projecting it back to the subspace), we will reach the fixed point $\Phi \cdot w_{PBE}$
Solution of $w_{PBE}$ with a Linear System Formulation

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

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

$$B^\pi(V) = R^\pi + \gamma P^\pi \cdot V$$

Therefore,

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

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

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

$$\Phi^T \cdot D \cdot (R^\pi + \gamma P^\pi \cdot \Phi \cdot w_{PBE}) = \Phi^T \cdot D \cdot \Phi \cdot w_{PBE}$$

$$\Phi^T \cdot D \cdot (\Phi - \gamma P^\pi \cdot \Phi) \cdot w_{PBE} = \Phi^T \cdot D \cdot R^\pi$$

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

$$w_{PBE} = A^{-1} \cdot b = (\Phi^T \cdot D \cdot (\Phi - \gamma P^\pi \cdot \Phi))^{-1} \cdot \Phi^T \cdot D \cdot R^\pi$$
Model-Free Learning of $w_{PBE}$

- How do we construct matrix $A = \Phi^T \cdot D \cdot (\Phi - \gamma P^\pi \cdot \Phi)$ and vector $b = \Phi^T \cdot D \cdot R^\pi$ without a model?
- Following policy $\pi$, each time we perform a model-free transition from $s$ to $s'$ getting reward $r$, we get a sample estimate of $A$ and $b$
- Estimate of $A$ is outer-product of vectors $\phi(s)$ and $\phi(s) - \gamma \cdot \phi(s')$
- Estimate of $b$ 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:
  \[
  \Delta w = \alpha \cdot (r + \gamma \cdot \phi(s')^T \cdot w - \phi(s)^T \cdot w) \cdot \phi(s)
  \]
- This converges to $w_{PBE}$ because $\mathbb{E}_\pi[\Delta w] = 0$ yields
  \[
  \Phi^T \cdot D \cdot (R^\pi + \gamma P^\pi \cdot \Phi \cdot w - \Phi \cdot w) = 0
  \]
  \[
  \Rightarrow \Phi^T \cdot D \cdot (\Phi - \gamma P^\pi \cdot \Phi) \cdot w = \Phi^T \cdot D \cdot R^\pi
  \]
Gradient TD Algorithms to solve for $w_{PBE}$

- 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

$$w_{PBE} = \arg \min_w d(\Pi_\Phi \cdot B^\pi \cdot V_w, V_w) = \arg \min_w d(\Pi_\Phi \cdot B^\pi \cdot V_w, \Pi_\Phi \cdot V_w)$$

- So we define the loss function (denoting $B^\pi \cdot V_w - V_w$ as $\delta_w$) as:

$$L(w) = (\Pi_\Phi \cdot \delta_w)^T \cdot D \cdot (\Pi_\Phi \cdot \delta_w) = \delta_w^T \cdot \Pi_\Phi^T \cdot D \cdot \Pi_\Phi \cdot \delta_w$$

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

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

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

$$= (\Phi^T \cdot D \cdot \delta_w)^T \cdot (\Phi^T \cdot D \cdot \Phi)^{-1} \cdot (\Phi^T \cdot D \cdot \delta_w)$$
We derive the TDC Algorithm based on $\nabla_w \mathcal{L}(w)$

$$\nabla_w \mathcal{L}(w) = 2 \cdot (\nabla_w (\Phi^T \cdot D \cdot \delta_w)^T) \cdot (\Phi^T \cdot D \cdot \Phi)^{-1} \cdot (\Phi^T \cdot D \cdot \delta_w)$$

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

- $\Phi^T \cdot D \cdot \delta_w = \mathbb{E}[\delta \cdot \phi(s)]$
- $\nabla_w (\Phi^T \cdot D \cdot \delta_w)^T = \mathbb{E}[(\nabla_w \delta) \cdot \phi(s)^T] = \mathbb{E}[(\gamma \cdot \phi(s') - \phi(s)) \cdot \phi(s)^T]$
- $\Phi^T \cdot D \cdot \Phi = \mathbb{E}[\phi(s) \cdot \phi(s)^T]$

Substituting, we get:

$$\nabla_w \mathcal{L}(w) = 2 \cdot \mathbb{E}[(\gamma \cdot \phi(s') - \phi(s)) \cdot \phi(s)^T] \cdot \mathbb{E}[\phi(s) \cdot \phi(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \phi(s)]$$
Weight Updates of TDC Algorithm

\[ \Delta w = -\frac{1}{2} \alpha \cdot \nabla_w \mathcal{L}(w) \]

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

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

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

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

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

**Cascade Learning: Update both \( w \) and \( \theta \) (\( \theta \) converging faster)**

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

Note: \( \theta^T \cdot \phi(s) \) operates as estimate of TD error \( \delta \) 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