Value Function Geometry and Gradient TD

Ashwin Rao

ICME, Stanford University

March 3, 2021

Overview

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 $\boldsymbol{V}^\pi : \mathcal{S} \rightarrow \mathbb{R}$
- m feature functions $\phi_1, \phi_2, \ldots, \phi_m : \mathcal{S} \rightarrow \mathbb{R}$
- 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 $\boldsymbol{V_w} : \mathcal{S} \rightarrow \mathbb{R}$ is defined as:

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

- $\boldsymbol{\mu_\pi} : \mathcal{S} \rightarrow [0, 1]$ 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 $\boldsymbol{V}$): $\mathcal{S} \rightarrow \mathbb{R}$ as a vector in this space
- Each dimension's coordinate is the VF for that dimension's state
- Coordinates of vector $\boldsymbol{V}^\pi$ 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 $s \rightarrow s'$ 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 $\boldsymbol{B}^\pi$

- 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 a linear operator in vector space $\mathbb{R}^n$
- So we denote and treat $\boldsymbol{B}^\pi$ as a $n \times n$ matrix
- 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 $\boldsymbol{\Pi_\Phi}$

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

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

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

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

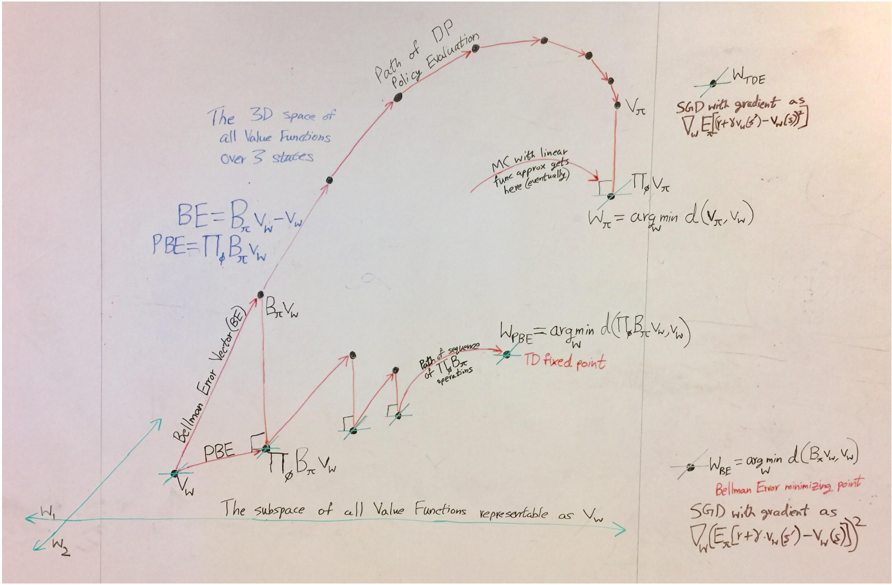
4 VF vectors of interest in the Φ subspace

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: $\boldsymbol{w}_{BE} = \arg\min_{\boldsymbol{w}} d(\boldsymbol{B}^\pi \cdot \boldsymbol{V_w}, \boldsymbol{V_w})$
   - This can be expressed as the solution of a linear system $\boldsymbol{A} \cdot \boldsymbol{w} = \boldsymbol{b}$
   - Matrix $\boldsymbol{A}$ and Vector $\boldsymbol{b}$ comprises of $\mathcal{R}^\pi, \mathcal{P}^\pi, \Phi, \boldsymbol{\mu_\pi}$
   - In model-free setting, $\boldsymbol{A}$ and $\boldsymbol{b}$ can be estimated with batch data
   - For non-linear approx or off-policy, Residual Gradient TD Algorithm
   - Based on observation: $\boldsymbol{w}_{BE} = \arg\min_{\boldsymbol{w}} (\mathbb{E}_\pi[\delta])^2$, where δ is TD Error
   - Cannot learn if we can only access features, and not underlying states
3. Temporal Difference Error (TDE)-minimizing:
   $\boldsymbol{w}_{TDE} = \arg\min_{\boldsymbol{w}} \mathbb{E}_\pi[\delta^2]$
   - Naive Residual Gradient TD Algorithm

4 VF vectors of interest in the Φ subspace (continued)

4. Projected Bellman Error (PBE)-minimizing:
   $\boldsymbol{w}_{PBE} = \arg\min_{\boldsymbol{w}} d((\boldsymbol{\Pi_\Phi} \cdot \boldsymbol{B}^\pi) \cdot \boldsymbol{V_w}, \boldsymbol{V_w})$
   - The minimum is 0, i.e., $\boldsymbol{\Phi} \cdot \boldsymbol{w}_{PBE}$ is the fixed point of operator $\boldsymbol{\Pi_\Phi} \cdot \boldsymbol{B}^\pi$
   - Starting with an arbitrary VF vector $\boldsymbol{V}$ and repeatedly applying $\boldsymbol{B}^\pi$ (potentially taking it out of the subspace) followed by $\boldsymbol{\Pi_\Phi}$ (projecting it back to the subspace), we will reach the fixed point $\boldsymbol{\Phi} \cdot \boldsymbol{w}_{PBE}$
   - $\boldsymbol{w}_{PBE}$ can be expressed as the solution of a linear system $\boldsymbol{A} \cdot \boldsymbol{w} = \boldsymbol{b}$
   - In model-free setting, $\boldsymbol{A}$ and $\boldsymbol{b}$ can be estimated with batch data
   - This yields the *Least Squares Temporal Difference (LSTD)* algorithm
   - For non-linear approx or off-policy, Gradient TD Algorithms



Path of DP Policy Evaluation
The 3D space of all Value Functions over 3 states
V_π
W_{TDE}
SGD with gradient as $\nabla_w E_\pi[(r+\gamma V_w(s')-V_w(s))^2]$
MC with linear func approx gets here (eventually)
$\Pi_\phi V_\pi$
$W_\pi = \arg\min_w d(V_\pi, V_w)$
$BE = B_\pi V_w - V_w$
$PBE = \Pi_\phi B_\pi V_w$
$B_\pi V_w$
$W_{PBE} = \arg\min_w d(\Pi_\phi B_\pi V_w, V_w)$
TD fixed point
Path of sequence of $\Pi_\phi B_\pi$ operations
Bellman Error Vector (BE)
PBE
$\Pi_\phi B_\pi V_w$
V_w
$W_{BE} = \arg\min_w d(B_\pi V_w, V_w)$
Bellman Error minimizing point
SGD with gradient as $\nabla_w (E_\pi[r+\gamma \cdot V_w(s')-V_w(s)])^2$
W_1
W_2
The subspace of all Value Functions representable as V_w

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

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

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

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

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 $\boldsymbol{w}_{BE}$

- We defined $\boldsymbol{w}_{BE}$ as the vector in the $\mathbf{\Phi}$ subspace that minimizes BE
- 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

Naive Residual Gradient Algorithm to solve for $\boldsymbol{w}_{TDE}$

- We defined $\boldsymbol{w}_{TDE}$ as the vector in the $\boldsymbol{\Phi}$ subspace that minimizes the expected square of the TD error δ when following policy π

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

- 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:

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

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

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

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

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

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

Therefore,

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

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

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

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

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

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 $\phi(s)$ and $\phi(s) - \gamma \cdot \phi(s')$
- Estimate of $\boldsymbol{b}$ 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:

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

- 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

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

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

$$\mathcal{L}(\mathbf{w}) = (\boldsymbol{\Pi_\Phi} \cdot \boldsymbol{\delta_w})^T \cdot \boldsymbol{D} \cdot (\boldsymbol{\Pi_\Phi} \cdot \boldsymbol{\delta_w}) = \boldsymbol{\delta_w}^T \cdot \boldsymbol{\Pi_\Phi}^T \cdot \boldsymbol{D} \cdot \boldsymbol{\Pi_\Phi} \cdot \boldsymbol{\delta_w}$$

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

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

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

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

TDC Algorithm to solve for $\boldsymbol{w}_{PBE}$

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 $r + \gamma \cdot \phi(s')^T \cdot \boldsymbol{w} - \phi(s)^T \cdot \boldsymbol{w}$ as δ

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

Substituting, we get:

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

Weight Updates of TDC Algorithm

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

$\boldsymbol{\theta} = \mathbb{E}[\boldsymbol{\phi}(s) \cdot \boldsymbol{\phi}(s)^T]^{-1} \cdot \mathbb{E}[\delta \cdot \boldsymbol{\phi}(s)]$ 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