

Value Function Geometry and Gradient TD

Ashwin Rao

Stanford University, ICME

October 27, 2018

Overview

- 1 Motivation and Notation
- 2 Geometric/Vector Space Representation of Value Functions
- 3 Solutions with Linear System Formulations
- 4 Residual Gradient TD
- 5 Gradient TD

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

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

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

VF Vector Space and VF Linear Approximations

- n -dimensional space, with each dim corresponding to a state in \mathcal{S}
- A vector in this space is a specific VF (typically denoted \mathbf{v}): $\mathcal{S} \rightarrow \mathbb{R}$
- Each dimension's coordinate is the VF for that dimension's state
- Coordinates of vector \mathbf{v}_π for policy π are: $[\mathbf{v}_\pi(s_1), \mathbf{v}_\pi(s_2), \dots, \mathbf{v}_\pi(s_n)]$
- Consider m vectors where j^{th} vector is: $[\phi_j(s_1), \phi_j(s_2), \dots, \phi_j(s_n)]$
- These m vectors are the m columns of $n \times m$ matrix $\Phi = [\phi_j(s_i)]$
- Their span represents an m -dim subspace within this n -dim space
- Spanned by the set of all $\mathbf{w} = [w_1, w_2, \dots, w_m] \in \mathbb{R}^m$
- Vector $\mathbf{v}_\mathbf{w} = \Phi \cdot \mathbf{w}$ in this subspace has coordinates $[\mathbf{v}_\mathbf{w}(s_1), \mathbf{v}_\mathbf{w}(s_2), \dots, \mathbf{v}_\mathbf{w}(s_n)]$
- Vector $\mathbf{v}_\mathbf{w}$ is fully specified by \mathbf{w} (so we often say \mathbf{w} to mean $\mathbf{v}_\mathbf{w}$)

Some more notation

- Denote $r(s, a)$ as the Expected Reward upon action a in state s
- Denote $p(s, s', a)$ as the probability of transition $s \rightarrow s'$ upon action a
- Define

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

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

- Denote \mathbf{R} as the vector $[\mathbf{R}(s_1), \mathbf{R}(s_2), \dots, \mathbf{R}(s_n)]$
- Denote \mathbf{P} as the matrix $[\mathbf{P}(s_i, s_{i'})], 1 \leq i, i' \leq n$
- Denote γ as the MDP discount factor

Bellman operator \mathbf{B}_π

- Bellman operator \mathbf{B}_π for policy π operating on VF vector \mathbf{v} defined as:

$$\mathbf{B}_\pi \mathbf{v} = \mathbf{R} + \gamma \mathbf{P} \cdot \mathbf{v}$$

- Note that \mathbf{v}_π is the fixed point of operator \mathbf{B}_π (meaning $\mathbf{B}_\pi \mathbf{v}_\pi = \mathbf{v}_\pi$)
- If we start with an arbitrary VF vector \mathbf{v} and repeatedly apply \mathbf{B}_π , by Contraction Mapping Theorem, we will reach the fixed point \mathbf{v}_π
- This is the Dynamic Programming Policy Evaluation algorithm
- Monte Carlo without func approx also converges to \mathbf{v}_π (albeit slowly)

Projection operator Π_{Φ}

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

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

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

- Projection operator for subspace spanned by Φ is denoted as Π_{Φ}
- Π_{Φ} performs an orthogonal projection of VF vector \mathbf{v} on subspace Φ
- So we need to minimize $d(\mathbf{v}, \Pi_{\Phi} \mathbf{v})$
- This is a weighted least squares regression with solution:

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

- So, the Projection operator Π_{Φ} can be written as:

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

4 VF vectors of interest in the Φ subspace

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

- ① Projection of \mathbf{v}_π : $\mathbf{w}_\pi = \arg \min_{\mathbf{w}} d(\mathbf{v}_\pi, \mathbf{v}_{\mathbf{w}})$
 - This is the VF we seek when doing linear function approximation
 - Because it is the VF vector “closest” to \mathbf{v}_π in the Φ subspace
 - Monte-Carlo with linear func approx will (slowly) converge to \mathbf{w}_π
- ② Bellman Error (BE)-minimizing: $\mathbf{w}_{BE} = \arg \min_{\mathbf{w}} d(\mathbf{B}_\pi \mathbf{v}_{\mathbf{w}}, \mathbf{v}_{\mathbf{w}})$
 - This is the solution to a linear system (covered later)
 - In model-free setting, Residual Gradient TD Algorithm (covered later)
 - Cannot learn if we can only access features, and not underlying states

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

③ Projected Bellman Error (PBE)-minimizing:

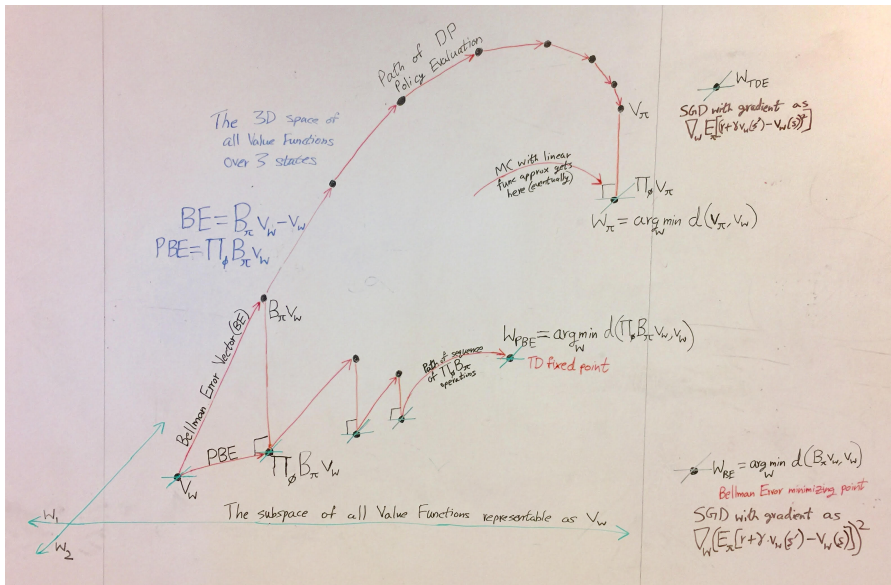
$$\mathbf{w}_{PBE} = \arg \min_{\mathbf{w}} d((\Pi_{\Phi} \cdot \mathbf{B}_{\pi})\mathbf{v}_{\mathbf{w}}, \mathbf{v}_{\mathbf{w}})$$

- The minimum is 0, i.e., $\Phi \cdot \mathbf{w}_{PBE}$ is the fixed point of operator $\Pi_{\Phi} \cdot \mathbf{B}_{\pi}$
- This fixed point is the solution to a linear system (covered later)
- Alternatively, if we start with an arbitrary $\mathbf{v}_{\mathbf{w}}$ and repeatedly apply $\Pi_{\Phi} \cdot \mathbf{B}_{\pi}$, we will converge to $\Phi \cdot \mathbf{w}_{PBE}$
- This is a DP-like process with approximation - repeatedly thrown out of the Φ subspace (applying Bellman operator \mathbf{B}_{π}), followed by landing back in the Φ subspace (applying Projection operator Π_{Φ})
- In model-free setting, Gradient TD Algorithms (covered later)

④ Temporal Difference Error (TDE)-minimizing:

$$\mathbf{w}_{TDE} = \arg \min_{\mathbf{w}} \mathbb{E}_{\pi}[\delta^2]$$

- δ is the TD error
- Minimizes the expected square of the TD error when following policy π
- Naive Residual Gradient TD Algorithm (covered later)



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

$$\begin{aligned}\mathbf{w}_{BE} &= \arg \min_{\mathbf{w}} d(\mathbf{v}_{\mathbf{w}}, \mathbf{R} + \gamma \mathbf{P} \cdot \mathbf{v}_{\mathbf{w}}) \\ &= \arg \min_{\mathbf{w}} d(\Phi \cdot \mathbf{w}, \mathbf{R} + \gamma \mathbf{P} \cdot \Phi \cdot \mathbf{w}) \\ &= \arg \min_{\mathbf{w}} d(\Phi \cdot \mathbf{w} - \gamma \mathbf{P} \cdot \Phi \cdot \mathbf{w}, \mathbf{R}) \\ &= \arg \min_{\mathbf{w}} d((\Phi - \gamma \mathbf{P} \cdot \Phi) \cdot \mathbf{w}, \mathbf{R})\end{aligned}$$

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

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

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

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

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

$$\mathbf{B}_\pi \mathbf{v} = \mathbf{R} + \gamma \mathbf{P} \cdot \mathbf{v}$$

Therefore,

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

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

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

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

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

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

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

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

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

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

- $r + \gamma \cdot w^T \cdot \phi(s') - w^T \cdot \phi(s)$ is the TD error, denoted δ

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

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

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

- 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:

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

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

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

- We defined \mathbf{w}_{BE} as the vector in the Φ subspace that minimizes BE
- But BE for a state is the expected TD error in that state
- So we want to do SGD with gradient of square of expected TD error

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

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

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

- For on-policy linear func approx, semi-gradient TD works
- For non-linear func approx or off-policy, we need Gradient TD
 - GTD: The original Gradient TD algorithm
 - GTD-2: Second-generation GTD
 - TDC: TD with Gradient correction
- We need to set up the loss function whose gradient will drive SGD
- $\mathbf{w}_{PBE} = \arg \min_{\mathbf{w}} d(\Pi_{\Phi} \mathbf{B}_{\pi} \mathbf{v}_{\mathbf{w}}, \mathbf{v}_{\mathbf{w}}) = \arg \min_{\mathbf{w}} d(\Pi_{\Phi} \mathbf{B}_{\pi} \mathbf{v}_{\mathbf{w}}, \Pi_{\Phi} \mathbf{v}_{\mathbf{w}})$
- So we define the loss function (denoting $\mathbf{B}_{\pi} \mathbf{v}_{\mathbf{w}} - \mathbf{v}_{\mathbf{w}}$ as $\delta_{\mathbf{w}}$) as:

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

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

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

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

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

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

Substituting, we get:

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

Weight Updates of TDC Algorithm

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

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

Cascade Learning: Update both w and θ (θ converging faster)

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

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