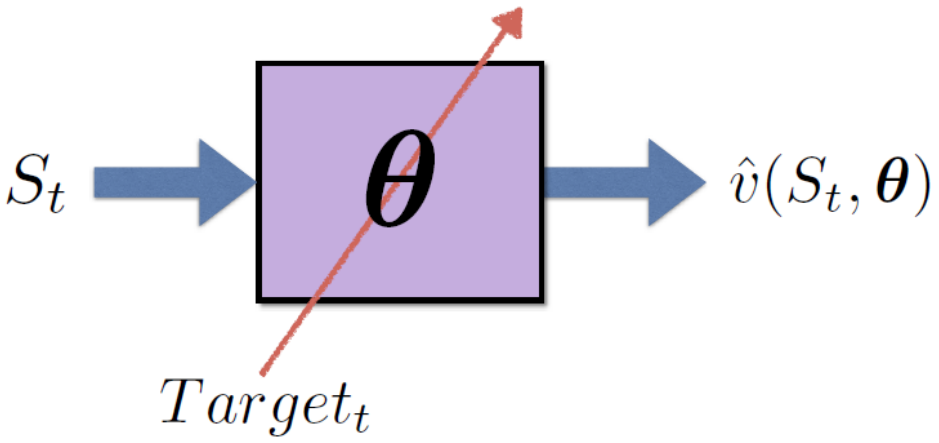
Class 4: On Policy Prediction With Approximation
Chapter 9

Sutton slides/silver slides

Value function approximation (VFA) replaces the table with a general parameterized form



Forms of approximations functions:

- A linear approximation,
- a neural network,
- a decision tree

The Prediction Objective

With approximation we can no longer hope to converge to the exact value for each state.

We must specify a state weighting or distribution $\mu(s) \geq 0, \sum_s \mu(s) = 1$

representing how much we care about the error in each state s.

The objective function is to minimize the Mean Square Value Error, denoted:

$$\overline{\text{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \Big[v_\pi(s) - \hat{v}(s, \mathbf{w}) \Big]^2 .$$

mu(s) is the fraction of time spent in s, which is called “on-policy distribution”

The continuing case and the episodic case are different.

- It is not obvious that the above is a good objective for RL (we want the value function in order to generate a good policy, but this is what we use.
- For a general function form no guarantee to converge to optimal w*

Stochastic-gradient and Semi-gradient Methods

In gradient-descent methods, the weight vector is a column vector with a fixed number of real valued components, $\mathbf{w} \doteq (w_1, w_2, \ldots, w_d)^\top$,[1] and the approximate value function $\hat{v}(s,\mathbf{w})$ is a differentiable function of $\mathbf{w}$ for all $s \in \mathcal{S}$. We will be updating $\mathbf{w}$ at each of a series of discrete time steps, $t = 0, 1, 2, 3, \ldots$, so we will need a notation $\mathbf{w}_t$ for the weight vector at each step. For now, let us assume that, on each step, we observe a new example $S_t \mapsto v_\pi(S_t)$ consisting of a (possibly randomly selected) state S_t and its true value under the policy. These states might be successive states from an interaction

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]^2 \tag{9.4}$$

$$= \mathbf{w}_t + \alpha\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t), \tag{9.5}$$

where α is a positive step-size parameter, and $\nabla f(\mathbf{w})$, for any scalar expression $f(\mathbf{w})$, denotes the vector of partial derivatives with respect to the components of the weight vector:

$$\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)^\top. \tag{9.6}$$

General Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is the idea behind most approximate learning

General SGD: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \, Error_t^2$

For VFA: $\leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right]^2$

Chain rule: $\leftarrow \boldsymbol{\theta} - 2\alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right]$

Semi-gradient: $\leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \hat{v}(S_t, \boldsymbol{\theta})$

Linear case: $\leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \boldsymbol{\phi}(S_t)$

Action-value form: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{q}(S_t, A_t, \boldsymbol{\theta}) \right] \boldsymbol{\phi}(S_t, A_t)$

A natural objective in VFA
is to minimize the Mean Square Value Error

$$\text{MSVE}(\boldsymbol{\theta}) \doteq \sum_{s \in \mathcal{S}} d(s) \Big[v_\pi(s) - \hat{v}(s, \boldsymbol{\theta}) \Big]^2$$

where $d(s)$ is the fraction of time steps spent in state s

True SGD will converge to a local minimum of the error objective

In *linear* VFA, there is only one minimum: local=global

Gradient Monte Carlo Algorithm for estimating v

we cannot perform the exact update (9.5) because
$v(S_t)$ is unknown, but we can approximate it by substituting U_t in place of $v(S_t)$. This yields the following general SGD method for state-value prediction:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[U_t - \hat{v}(S_t, \mathbf{w}_t) \Big] \nabla \hat{v}(S_t, \mathbf{w}_t).$$

Gradient Monte Carlo Algorithm for Approximating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^n \to \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta}$ as appropriate (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat forever:
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  For $t = 0, 1, \ldots, T-1$:
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \big[G_t - \hat{v}(S_t, \boldsymbol{\theta}) \big] \nabla \hat{v}(S_t, \boldsymbol{\theta})$

Th egeneral SGD (aiming at G_t) converges to a local optimum approximation

Semi Gradient Methods

Replacing G_t with a bootstrapping target such as TD(0) or G_{t:t+n} will not guarantee convergence (but for linear functions)

semi-gradient (bootstrapping) methods offer important advantages: they typically enable significantly faster learning, without waiting for the end of an episode. This enables them to be used on continuing problems and provides computational advantages.
A prototypical semi-gradient method is semi-gradient TD(0),

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S' is terminal

State Aggregation

State aggregation is the simplest kind of VFA

- States are partitioned into disjoint subsets (groups)
- One component of $\boldsymbol{\theta}$ is allocated to each group

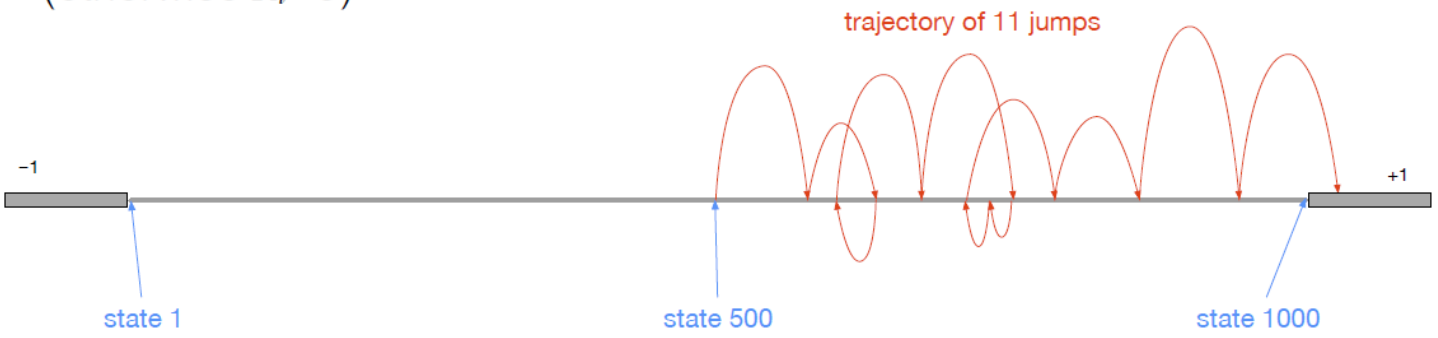
$$\hat{v}(s, \boldsymbol{\theta}) \doteq \theta_{group(s)}$$

$$\nabla_{\boldsymbol{\theta}} \hat{v}(s, \boldsymbol{\theta}) \doteq [0, 0, \ldots, 0, 1, 0, 0, \ldots, 0]$$

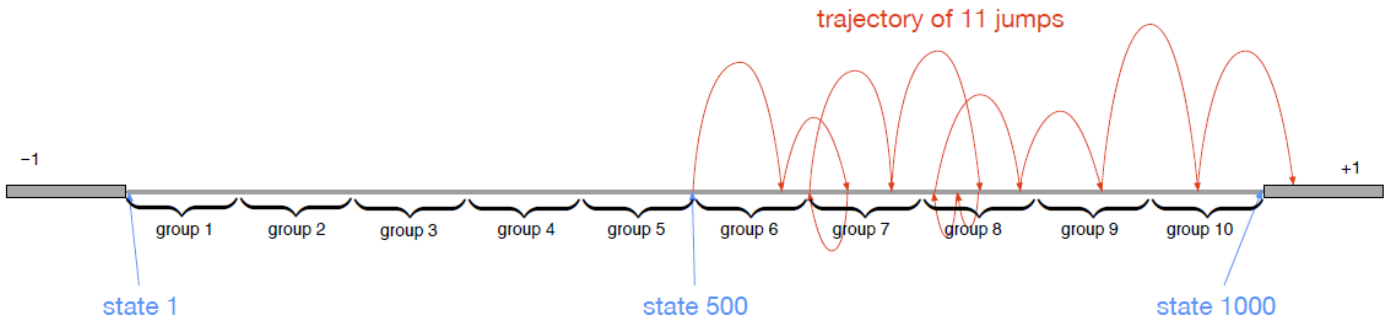
Recall: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \hat{v}(S_t, \boldsymbol{\theta})$

The 1000-state random walk example

- States are numbered 1 to 1000
- Walks start in the near middle, at state 500 $S_0 = 500$
- At each step, *jump* to one of the 100 states to the right, or to one of the 100 states to the left $S_1 \in \{400..499\} \cup \{501..600$
- If the jump goes beyond 1 or 1000, terminates with a reward of −1 or +1 (otherwise R_t=0)



State aggregation into 10 groups of 100

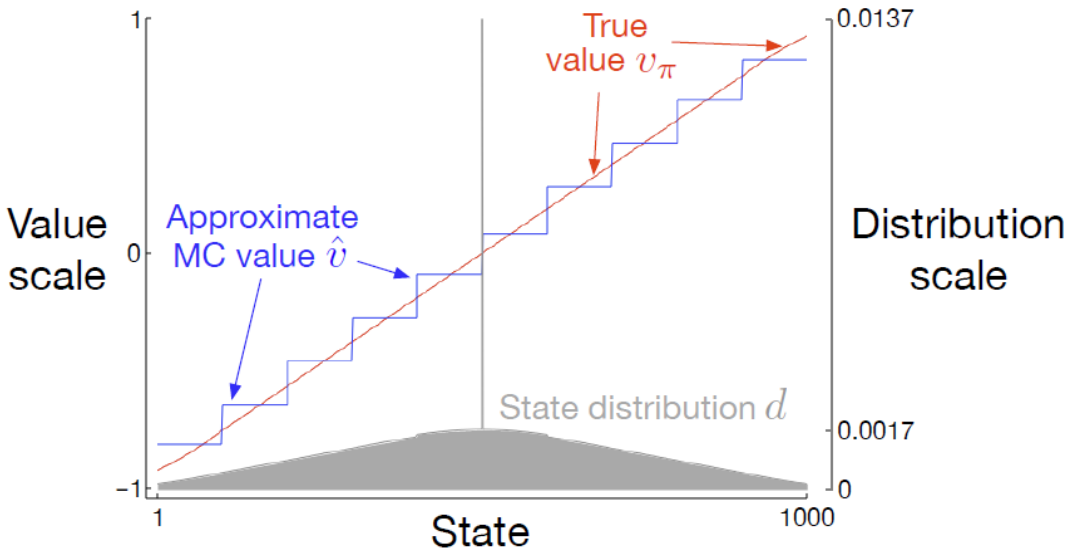


The whole value function over 1000 states will be approximated with 10 numbers!

Gradient MC works well on the 1000-state random walk using *state aggregation*

- 10 groups of 100 states
- after 100,000 episodes
- $\alpha = 2 \times 10^{-5}$

- state distribution affects accuracy



Linear Methods

One of the most important special cases of function approximation is that in which the approximate function, $\hat{v}(\cdot,\mathbf{w})$, is a linear function of the weight vector, $\mathbf{w}$. Corresponding to every state s, there is a real-valued vector $\mathbf{x}(s) \doteq (x_1(s), x_2(s), \ldots, x_d(s))^\top$, with the same number of components as $\mathbf{w}$. Linear methods approximate state-value function by the inner product between $\mathbf{w}$ and $\mathbf{x}(s)$:

$$\hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s). \tag{9.8}$$

X(s) is a feature vector with the same dimensionality as w

$$\nabla \hat{v}(s,\mathbf{w}) = \mathbf{x}(s).$$

Thus, in the linear case the general SGD update (9.7) reduces to a particularly simple form:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[U_t - \hat{v}(S_t,\mathbf{w}_t)\Big]\mathbf{x}(S_t).$$

In the linear case there is only one optimum thus the Semi-SGD is guaranteed to converge to or near a local optimum.
SGD does converges to the global optimum if alpha satisfies the usual conditions
Of reducing over time.

TD(0) Convergence

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha\Big(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t\Big)\mathbf{x}_t \\ &= \mathbf{w}_t + \alpha\Big(R_{t+1}\mathbf{x}_t - \mathbf{x}_t\big(\mathbf{x}_t - \gamma \mathbf{x}_{t+1}\big)^\top \mathbf{w}_t\Big),\end{aligned} \tag{9.9}$$

where here we have used the notational shorthand $\mathbf{x}_t = \mathbf{x}(S_t)$. Once the system has reached steady state, for any given $\mathbf{w}_t$, the expected next weight vector can be written

$$\mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t), \tag{9.10}$$

where

$$\mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d \quad \text{and} \quad \mathbf{A} \doteq \mathbb{E}\Big[\mathbf{x}_t\big(\mathbf{x}_t - \gamma \mathbf{x}_{t+1}\big)^\top\Big] \in \mathbb{R}^d \times \mathbb{R}^d \tag{9.11}$$

From (9.10) it is clear that, if the system converges, it must converge to the weight vector $\mathbf{w}_{\text{TD}}$ at which

$$\begin{aligned} & \mathbf{b} - \mathbf{A}\mathbf{w}_{\text{TD}} = \mathbf{0} \\ \Rightarrow \quad & \mathbf{b} = \mathbf{A}\mathbf{w}_{\text{TD}} \\ \Rightarrow \quad & \mathbf{w}_{\text{TD}} \doteq \mathbf{A}^{-1}\mathbf{b}. \end{aligned} \tag{9.12}$$

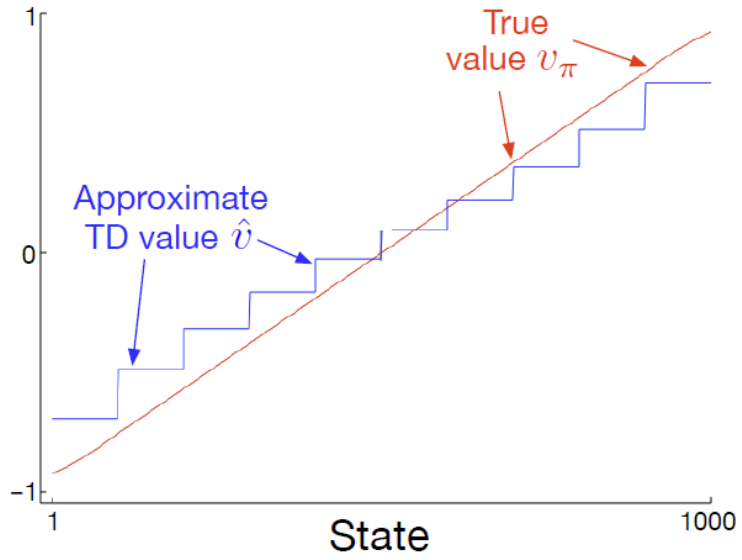
This quantity is called the *TD fixed point*. In fact linear semi-gradient TD(0) converges to this point. Some of the theory proving its convergence, and the existence of the inverse above, is given in the box.

Bootstrapping on the 1000-state random walk

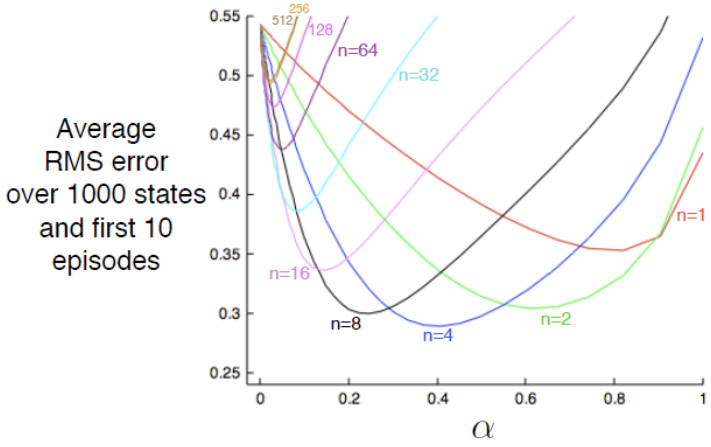
Gradient TD is less accurate than MC
on the 1000-state random walk using state aggregation

- 10 groups of 100 states
- after 100,000 episodes
- α = 2 x 10^{-5}

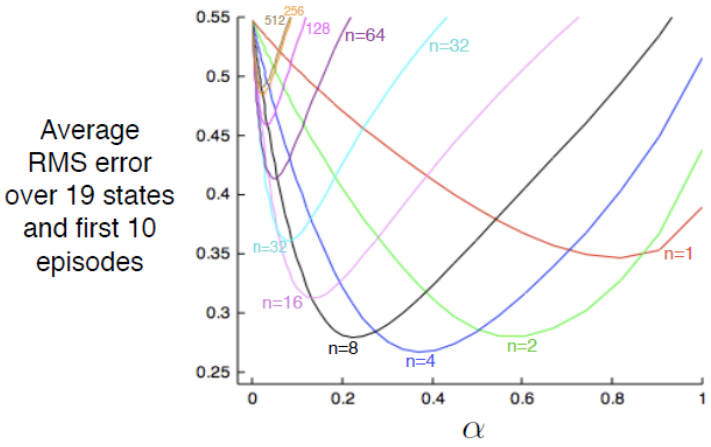
Relative values are still pretty accurate



Bootstrapping still greatly speeds learning very much like the tabular 19-state walk



1000 states aggregated
into 20 groups of 50



19 states tabular
(from Chapter 7)

n-Step Semi-gradient TD for v

The semi-gradient n-step TD algorithm we used in this example is the natural extension of the tabular n-step TD algorithm presented in Chapter 7 to semi-gradient function approximation. The key equation, analogous to (7.2), is

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha \left[G_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1}) \right] \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}), \qquad 0 \leq t < T, \tag{9.15}$$

where the n-step return is generalized from (7.1) to

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}), \quad 0 \leq t \leq T - n. \tag{9.16}$$

n-step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (S_t and R_t) can take their index mod n

Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t + 1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n \hat{v}(S_{\tau+n}, \mathbf{w})$ $(G_{\tau:\tau+n})$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{v}(S_\tau, \mathbf{w}) \right] \nabla \hat{v}(S_\tau, \mathbf{w})$
  Until $\tau = T - 1$