Reinforcement Learning Training 2025

Functional Approximation

Motivation

- Up till now, we only deal with problems with **discrete** state space.
    - v and q are represented in a table.
    - *Tabular* case
- Real problem has too many states.
    - The tabular approach requires too much memory and computation.

Formulation

- Instead of representing values in a table, they are now being represented by

$$\hat{v}(s;w) \approx v_\pi(s)$$

$$\hat{q}(s,a;w) \approx q_\pi(s,a)$$

- where the parameter w is the parameter of the function that defines the policy $\pi(a|s)$ that the agent follows.

What is w?

$$\hat{v}(s;w) \approx v_\pi(s)$$

$$\hat{q}(s, a;w) \approx q_\pi(s,a)$$

- Tunable parameters of the function that defines the policy.
- Weights of a deep learning neural network.

Changing w

- When you update the weight vector w based on some update equation for a specific state s or state action pair (s, a)
  - it not only updates the v or q for that specific s or (s, a)
  - but it also updates other *nearby* v and q.
- This is different from the tabular case where you can update each v or q independently.

Functional approach

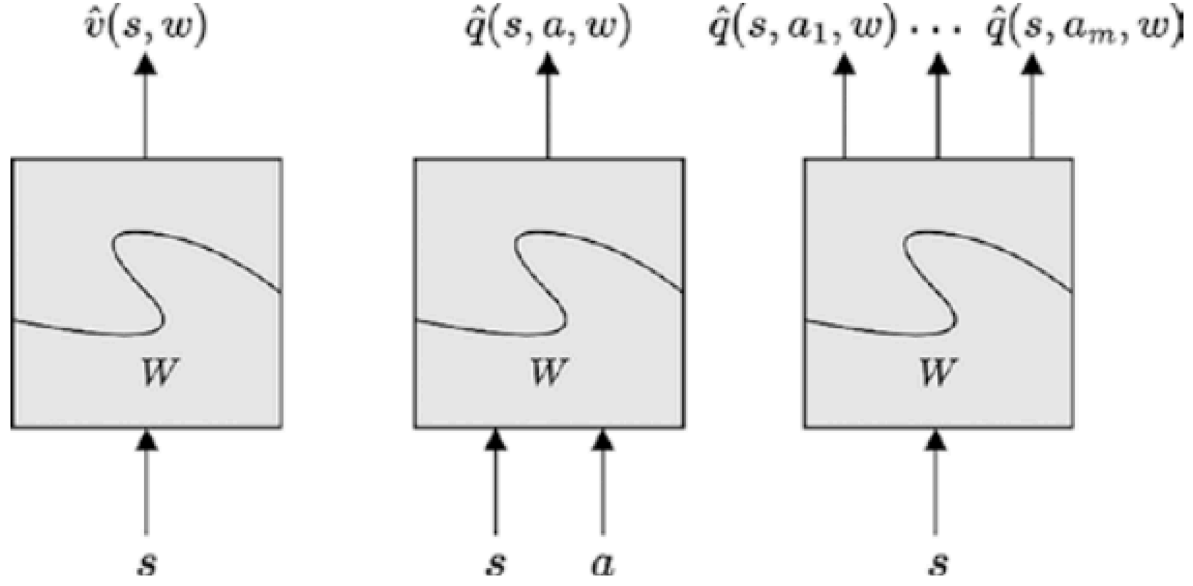


Figure 5-1. *Ways to represent $\hat{v}(s;w)$ or $\hat{q}(s,a;w)$ using the function approximation approach. The first and last ones are the most common ones used in this chapter*

How do we find update equation for w?

> We used technique from supervised learning.

- 1 Define a loss function

$$J(w)=\left[y(t)-\hat{y}(t;w)\right]^2;\text{where } \hat{y}(t;w)=Model_w\left[x(t)\right]$$

- 2 Update w using gradient descent.

$$w_{t+1}=w_t-\alpha\nabla_w J(w)$$

Monte Carlo

- Using the update equation (from before)

$$V_{t+1}(s) = V_t(s) + \alpha\left[G_t(s) - V_t(s)\right]$$

- Gives

$$w_{t+1} = w_t + \alpha\,.\left[G_t(s) - V_t(s;w)\right].\nabla_w V_t(s;w)$$

Temporal Difference (TD(0))

- Using the update equation (from before)

$$V_{t+1}(s)=V_t(s)+\alpha\left[R_{t+1}+\gamma\cdot V_t(s')-V_t(s)\right]$$

- Gives

$$w_{t+1}=w_t+\alpha\,.\left[R_{t+1}+\gamma\cdot V_t(s';w)-V_t(s;w)\right].\nabla_w V_t(s;w)$$

Linear Model

- State value function.

$$\hat{v}(s;w) = x(s)^T \cdot w = \sum_i x_i(s) \cdot w_i$$

- Gradient

$$\nabla_w V_t(s;w) = x(s)$$

Linear Model

- Update equation

$$w_{t+1} = w_t + \alpha . \left[V_\pi(s) - V_t(s;w) \right] . x(s)$$

- In MC, $V_\pi = g_t$
- In TD(0), $V_\pi = R_{t+1} + \gamma \cdot v_t(s')$

Linear Model

- Update equations

MC update:

$$w_{t+1} = w_t + \alpha \,.\left[G_t(s) - V_t(s;w)\right].\, x(s)$$

TD(0) update:

$$w_{t+1} = w_t + \alpha \,.\left[R_{t+1} + \gamma \,.V_t(s';w) - V_t(s;w)\right].\, x(s)$$

Challenge

- Recalled the loss function.

$$J(w) = \left[\boxed{y(t)} - \hat{y}(t;w) \right]^2$$

- In supervised learning, $y(t)$ are labels that do not change. *(No problem)*
- In RL, $y(t)$ are v or q which keep changing (because we are finding them)
  - *Nonstationarity* problem
- Also, in TD(0), we use bootstrapping (target is not a true value).
  - This is even worse.

Table 5-1. Convergence of Prediction/Estimation Algorithms

Policy Type	Algorithm	Table Lookup	Linear	Nonlinear
On-policy	MC	Y	Y	Y
	TD(0)	Y	Y	N
	TD(λ)	Y	Y	N
Off-policy	MC	Y	Y	Y
	TD(0)	Y	N	N
	TD(λ)	Y	N	N

Control

- Similar to how you find v, here we use q

$$\hat{q}(s,a;w) \approx q_{\pi}(s,a)$$

- Cost function

$$J(w) = E_{\pi}\left[\left(q_{\pi}(s,a) - \hat{q}(s,a;w)\right)^{2}\right]$$

- Update equation

$$w_{t+1} = w_{t} - \alpha \,.\, \nabla_{w} J(w)$$

Control

- Gives

MC update:

$$w_{t+1} = w_t + \alpha \cdot \left[G_t(s) - q_t(s,a;w) \right] \cdot \nabla_w \hat{q}(\text{s,a;w})$$

TD(0) update:

$$w_{t+1} = w_t + \alpha \cdot \left[R_{t+1} + \gamma \cdot q_t(s',a';w) - q_t(s,a;w) \right] \cdot \nabla_w \hat{q}(\text{s,a;w})$$

Control

- We use GPI (same as before).

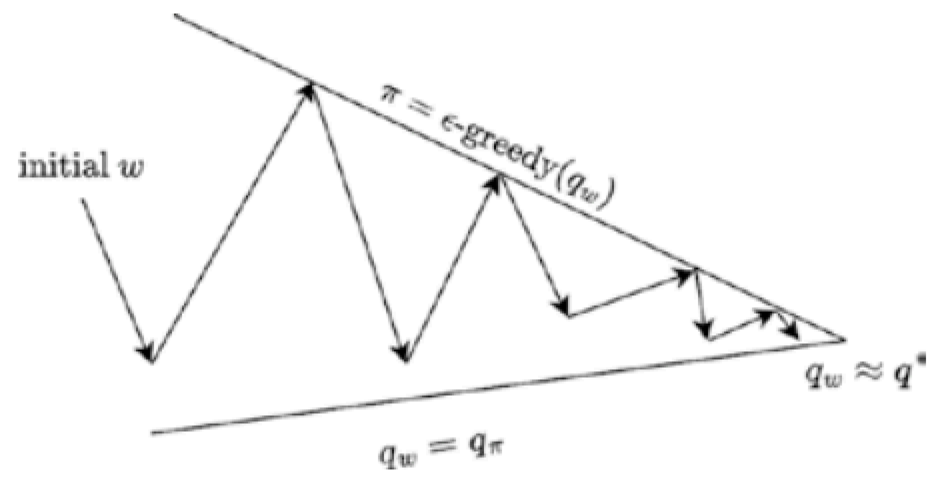
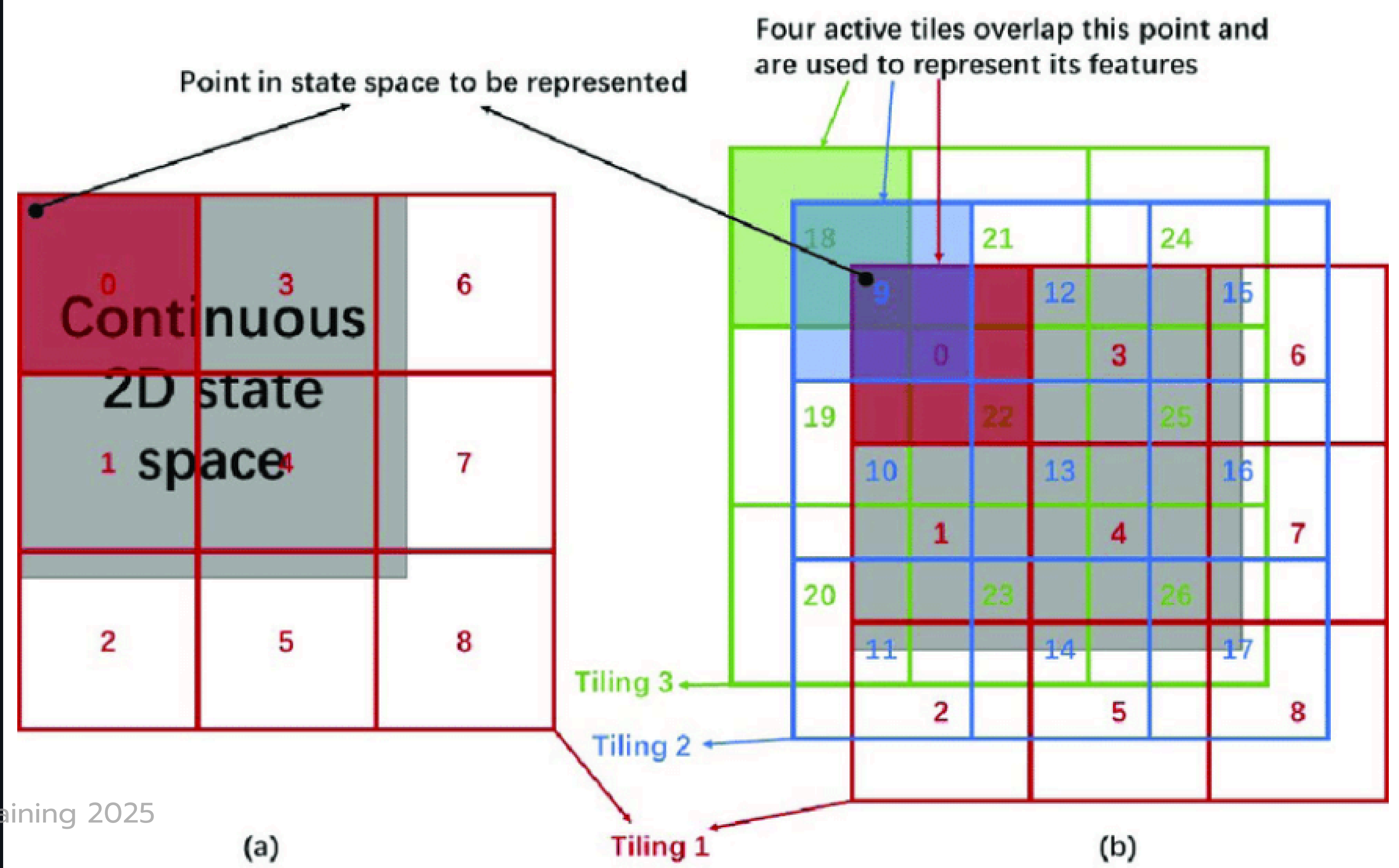


Figure 5-5. *Generalized policy iteration with function approximation*

Semi Gradient n-Step SARSA

- On policy
  - ϵ-greedy policy for evaluation and control.

Tile Encoding



Tile Encoding

- Binary features

$$\hat{q}(S, A; w) = x(S, A)^T . w$$

$$\nabla \hat{q}(S_\tau, A_\tau; w) = x(S, A)$$

Semi Gradient n-Step SARSA Control (Episodic)

Input:

A differentiable function $\hat{q}(s, a; w) : |S| \times |A| \times \mathbb{R}^d \to \mathbb{R}$

Other paramters: steps size α, exploration prob. ϵ, number of steps: n

Initialize:

Initialize weight vector $\mathbf{w} : \mathbb{R}^d$ abitrarily like $\mathbf{w} = \mathbf{0}$

Three arrays to store (S_t, A_t, R_t) which access using "$\mathrm{mod}\ n + 1$"

Loop for each episode:

Start episode with S_0 (non-termial)

Select and store A_0, ϵ-greedy action using $\hat{q}(S_0, \bullet; w)$

$T \leftarrow \infty$

Loop for $t = 1, 2, 3, \ldots$

| if $t < T$, then:

| Take action A_t

| Observe and store R_{t+1} and S_{t+1}

| if S_{t+1} is termial then:

| $T \leftarrow t + 1$

| else:

| select and store A_{t+1} using ϵ-greedy $\hat{q}(S_t, \bullet; w)$

| $\tau \leftarrow t - n + 1$ (τ is inex whose estimate is being updated)

| If $\tau \geq 0$:

| $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} \cdot R_i$

| if $t + n < T$, then $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}; w)$

| $w \leftarrow w + \alpha \cdot [G - \hat{q}(S_\tau, A_\tau; w)] \cdot \nabla_w \hat{q}(S_\tau, A_\tau; w)$

Until $\tau = T - 1$

Semi-gradient SARSA(λ)

- *Skip for now*

Instability

> 3 ways that can cause instability

Function approximation: A way to generalize for a very large state space using a model with the number of parameters smaller than the total number of possible states.

Bootstrapping: Forming target values using estimates of state values, for example, in TD(0) the target being the estimate $R_{t+1} + \gamma \cdot V_t(s'; w)$

Off-policy learning: Training the agent using a behavior policy but learning a different optimal policy.

Table 5-3. Convergence of Control Algorithms

Algorithm	Table Lookup	Linear	Nonlinear
MC control	Y	(Y)	N
On Policy TD (SARSA)	Y	(Y)	N
Off policy Q-Learning	Y	N	N
Gradient Q-Learning	Y	Y	N