Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Function Approximation for Prediction and Control

CMU 10-703

Katerina Fragkiadaki

Used Materials

- **Disclaimer**: Much of the material and slides for this lecture were borrowed from Russ Salakhutdinov, Rich Sutton's class and David Silver's class on Reinforcement Learning.

Large-Scale Reinforcement Learning

- Reinforcement learning has been used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter: continuous state space

- Tabular methods clearly do not work

Value Function Approximation (VFA)

- So far we have represented value function by a lookup table
  - Every state s has an entry V(s), or
  - Every state-action pair (s,a) has an entry Q(s,a)
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation

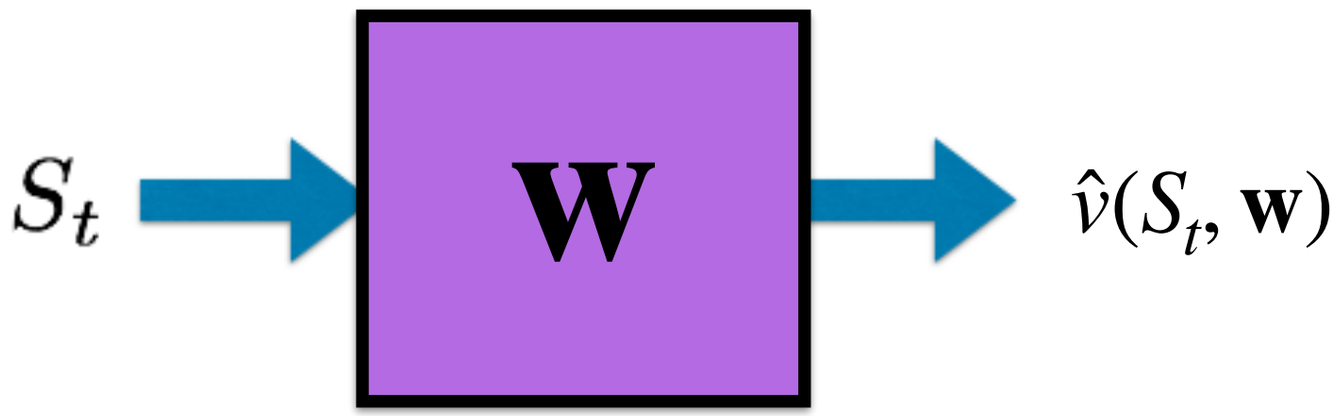
$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

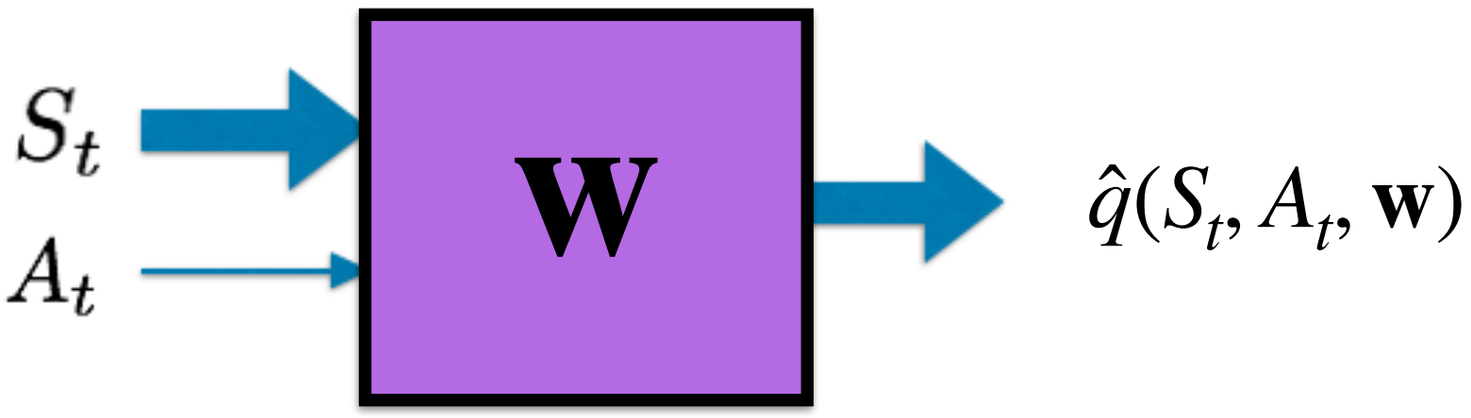
$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - Generalize from seen states to unseen states

Value Function Approximation (VFA)

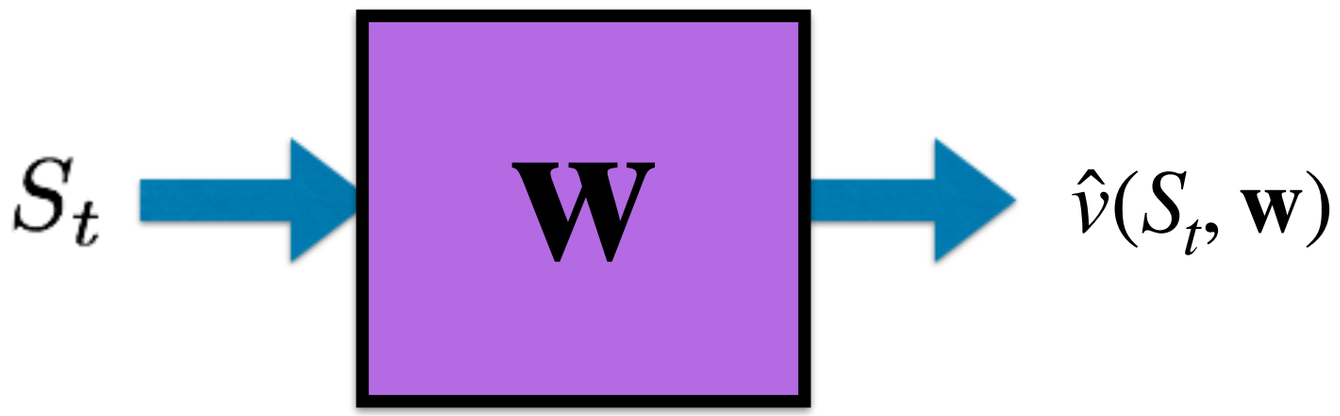
- Value function approximation (VFA) replaces the table with a general parameterized form:



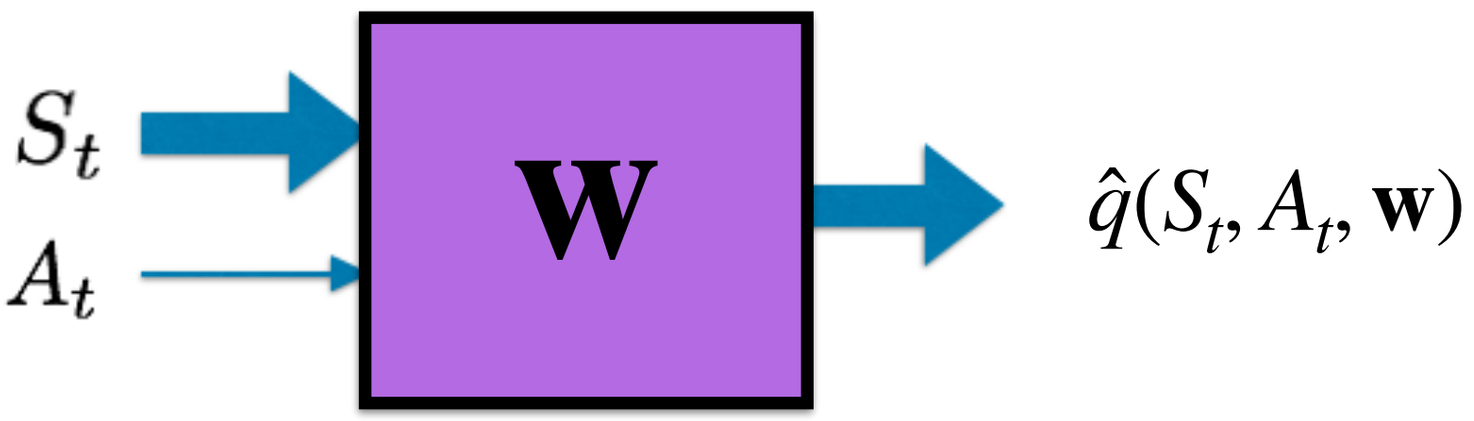


Value Function Approximation (VFA)

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



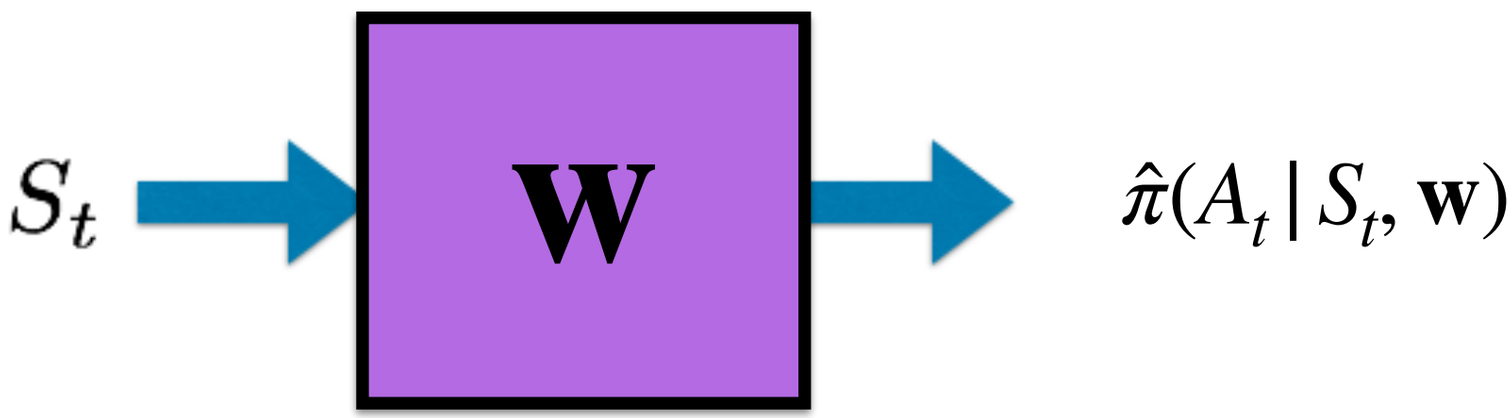
$$|\mathbf{w}| << |\mathcal{S}|$$



When we update the parameters $\mathbf{w}$, the values of many states change simultaneously!

Policy Approximation

- Policy approximation replaces the table with a general parameterized form:



Which Function Approximation?

- There are many function approximators, e.g.
  - Linear combinations of features
  - Neural networks
  - Decision tree
  - Nearest neighbour
  - Fourier / wavelet bases
  - …

Which Function Approximation?

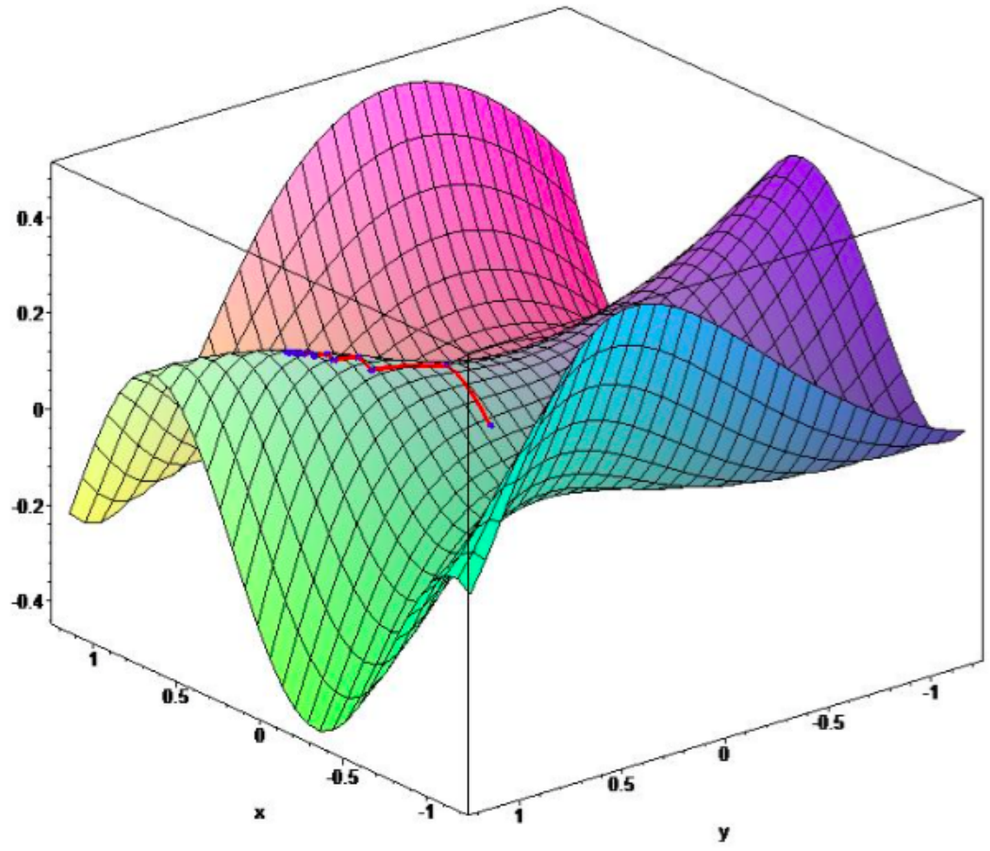
- There are many function approximators, e.g.
  - Linear combinations of features
  - Neural networks
  - Decision tree
  - Nearest neighbour
  - Fourier / wavelet bases
  - …

- differentiable function approximators

Gradient Descent

- Let J(w) be a differentiable function of parameter vector w
- Define the gradient of J(w) to be:

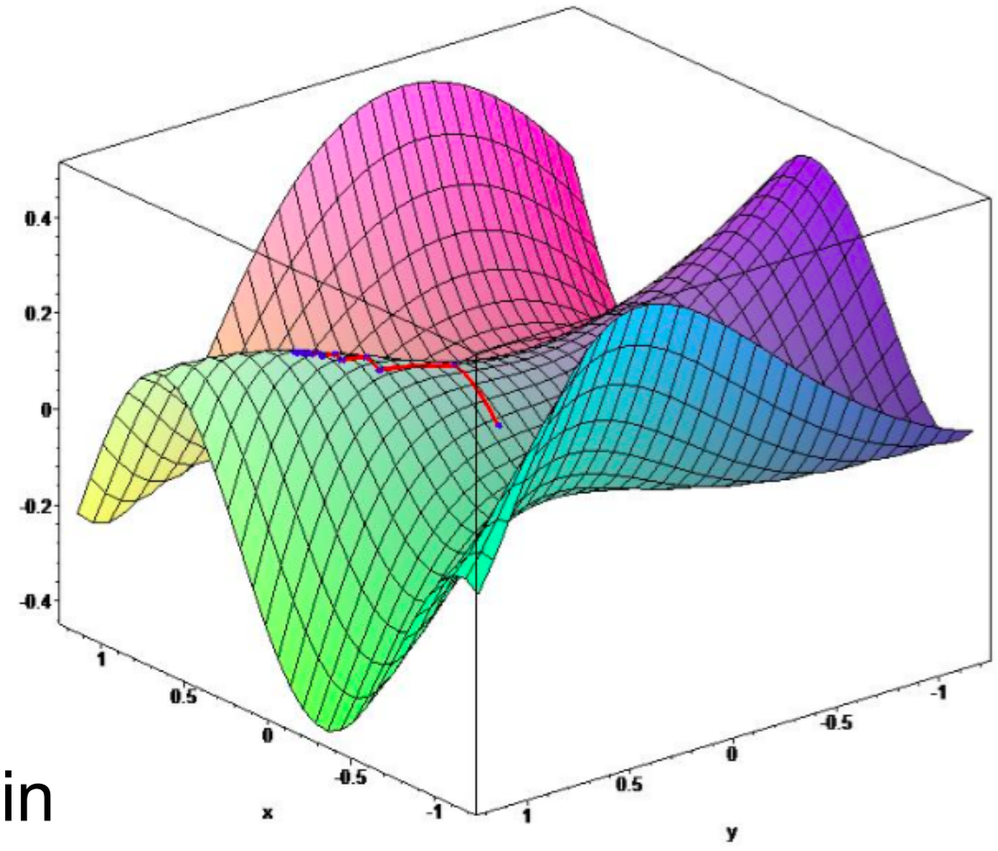
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$



Gradient Descent

- Let J(w) be a differentiable function of parameter vector w
- Define the gradient of J(w) to be:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$



- To find a local minimum of J(w), adjust w in direction of the negative gradient:

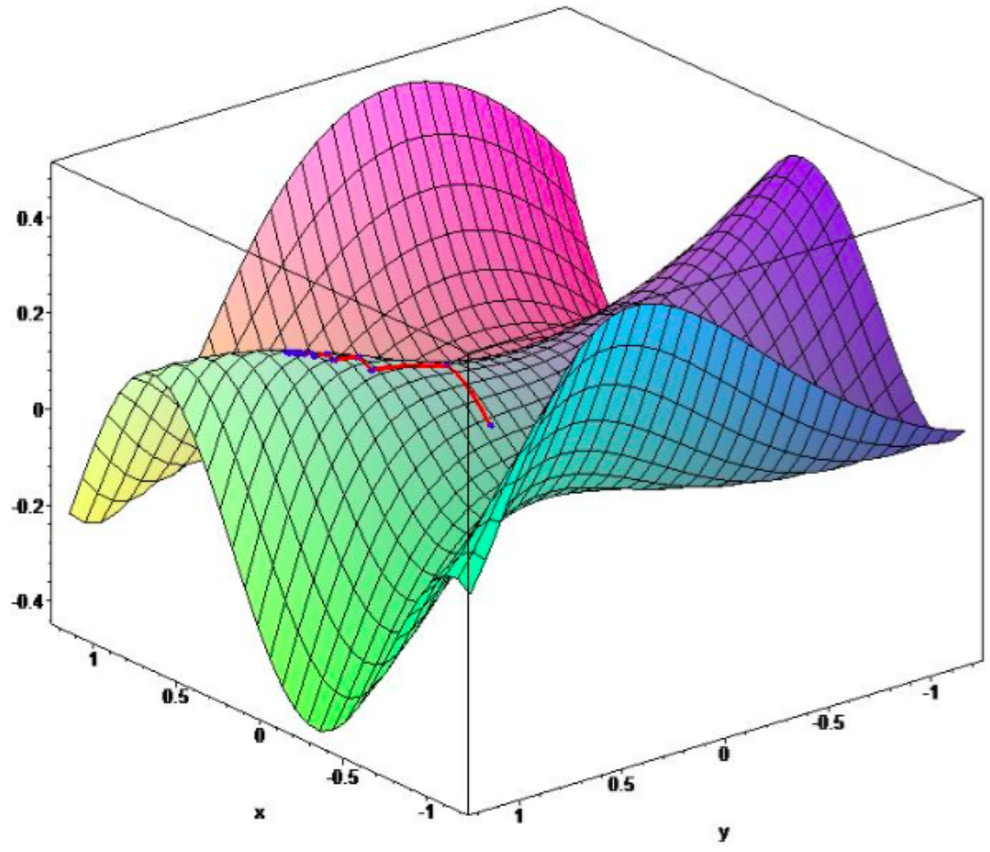
$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

Step-size

Gradient Descent

- Let J(w) be a differentiable function of parameter vector w
- Define the gradient of J(w) to be:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$



- Starting from a guess $\mathbf{w}_0$
- We consider the sequence $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \ldots$ s.t. :

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}_n)$$

- We then have $J(\mathbf{w}_0) \geq J(\mathbf{w}_1) \geq J(\mathbf{w}_2) \geq \ldots$

Our objective

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$

Our objective

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

Our objective

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

Let $\mu(S)$ denote how much time we spend in each state s under policy π, then:

$$J(w) = \sum_{n=1}^{|\mathcal{S}|} \mu(S) \left[v_\pi(S) - \hat{v}(S, \mathbf{w}) \right]^2 \qquad \sum_{s \in \mathcal{S}} \mu(S) = 1$$

Very important choice: it is OK if we cannot learn the value of states we visit very few times, there are too many states, I should focus on the ones that matter: the RL solution to curse of dimensionality.

Our objective

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

Let $\mu(S)$ denote how much time we spend in each state s under policy π , then:

$$J(w) = \sum_{n=1}^{|\mathcal{S}|} \mu(S) \left[v_\pi(S) - \hat{v}(S, \mathbf{w}) \right]^2 \qquad \sum_{s \in \mathcal{S}} \mu(S) = 1$$

In contrast to:

$$J_2(w) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \left[v_\pi(S) - \hat{v}(S, \mathbf{w}) \right]^2$$

On-policy state distribution

Let $h(s)$ be the initial state distribution, i.e, the probability that an episode starts at state s.

Then the un-normalized on-policy state probability $\eta(s)$ satisfies the following recursions:

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_{a} \pi(a \mid \bar{s}) p(s \mid \bar{s}, a), \ \forall s \in \mathcal{S}$$

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}, \quad \forall s \in \mathcal{S}$$

Gradient Descent

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

Gradient Descent

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Starting from a guess w_0

Gradient Descent

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Starting from a guess w_0
- We consider the sequence $w_0, w_1, w_2, \ldots$ s.t. :

$$w_{n+1} = w_n - \frac{1}{2} \alpha \nabla_w J(w_n)$$

- We then have $J(w_0) \geq J(w_1) \geq J(w_2) \geq \ldots$

Gradient Descent

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

- Gradient descent finds a local minimum:

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

Stochastic Gradient Descent

- Goal: find parameter vector w minimizing mean-squared error between the true value function $v_\pi(S)$ and its approximation $\hat{v}(S, \mathbf{w})$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

- Gradient descent finds a local minimum:

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Stochastic gradient descent (SGD) samples the gradient:

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

Least Squares Prediction

- Given value function approximation: $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And experience D consisting of ⟨state,value⟩ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Find parameters w that give the best fitting value function v(s,w)?
- Least squares algorithms find parameter vector w minimizing sum-squared error between $v(S_t,w)$ and target values v_t^π:

$$\begin{aligned} LS(\mathbf{w}) &= \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2 \\ &= \mathbb{E}_{\mathcal{D}} \left[(v^\pi - \hat{v}(s, \mathbf{w}))^2 \right] \end{aligned}$$

SGD with Experience Replay

- Given experience consisting of ⟨state, value⟩ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Repeat
  - Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

  - Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha(v^\pi - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

- Converges to least squares solution

Feature Vectors

- Represent state by a feature vector

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- For example
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Linear Value Function Approximation (VFA)

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters w

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\mathbf{x}(S)$$

- Update = step-size × prediction error × feature value
- Later, we will look at the neural networks as function approximators.

Incremental Prediction Algorithms

- We have assumed the true value function $v_\pi(s)$ is given by a supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a target for $v_\pi(s)$
- For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

- For TD(0), the target is the TD target: $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

Remember $\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$

Monte Carlo with VFA

- Return G_t is an unbiased, noisy sample of true value $v_\pi(S_t)$
- Can therefore apply supervised learning to "training data":

$$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, \ldots, \langle S_T, G_T \rangle$$

- For example, using linear Monte-Carlo policy evaluation

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)\end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum

Monte Carlo with VFA

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 $t = 0, 1, \ldots, T-1$:
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[G_t - \hat{v}(S_t,\boldsymbol{\theta})\big]\nabla\hat{v}(S_t,\boldsymbol{\theta})$

TD Learning with VFA

- The TD-target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ a biased sample of true value $v_\pi(S_t)$

- Can still apply supervised learning to "training data":

$$\langle S_1, R_2 + \gamma \hat{v}(S_2, \mathbf{w}) \rangle, \langle S_2, R_3 + \gamma \hat{v}(S_3, \mathbf{w}) \rangle, ..., \langle S_{T-1}, R_T \rangle$$

- For example, using linear TD(0):

$$\begin{aligned} \Delta \mathbf{w} &= \alpha(R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \\ &= \alpha \delta \mathbf{x}(S) \end{aligned}$$

We ignore the dependence of the target on w!

We call it semi-gradient methods

TD Learning with VFA

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

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

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

Control with VFA

- Policy evaluation Approximate policy evaluation: $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$
- Policy improvement ε-greedy policy improvement

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimize mean-squared error between the true action-value function $q_\pi(S,A)$ and the approximate action-value function:

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2} \nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

Linear Action-Value Function Approximation

- Represent state and action by a feature vector

$$\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$$

- Represent action-value function by linear combination of features

$$\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A)\mathbf{w}_j$$

- Stochastic gradient descent update

$$\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$$

$$\Delta\mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\mathbf{x}(S, A)$$

Incremental Control Algorithms

- Like prediction, we must substitute a target for $q_\pi(S,A)$
- For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- For TD(0), the target is the TD target: $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

Incremental Control Algorithms

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta} \in \mathbb{R}^n$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        If S' is terminal:
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \boldsymbol{\theta})$ (e.g., ε-greedy)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{q}(S', A', \boldsymbol{\theta}) - \hat{q}(S, A, \boldsymbol{\theta})\big]\nabla\hat{q}(S, A, \boldsymbol{\theta})$
        $S \leftarrow S'$
        $A \leftarrow A'$

Incremental Control Algorithms

- Like prediction, we must substitute a target for $q_\pi(S,A)$
- For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

- For TD(0), the target is the TD target: $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

Can we guess the deep Q learning update rule?

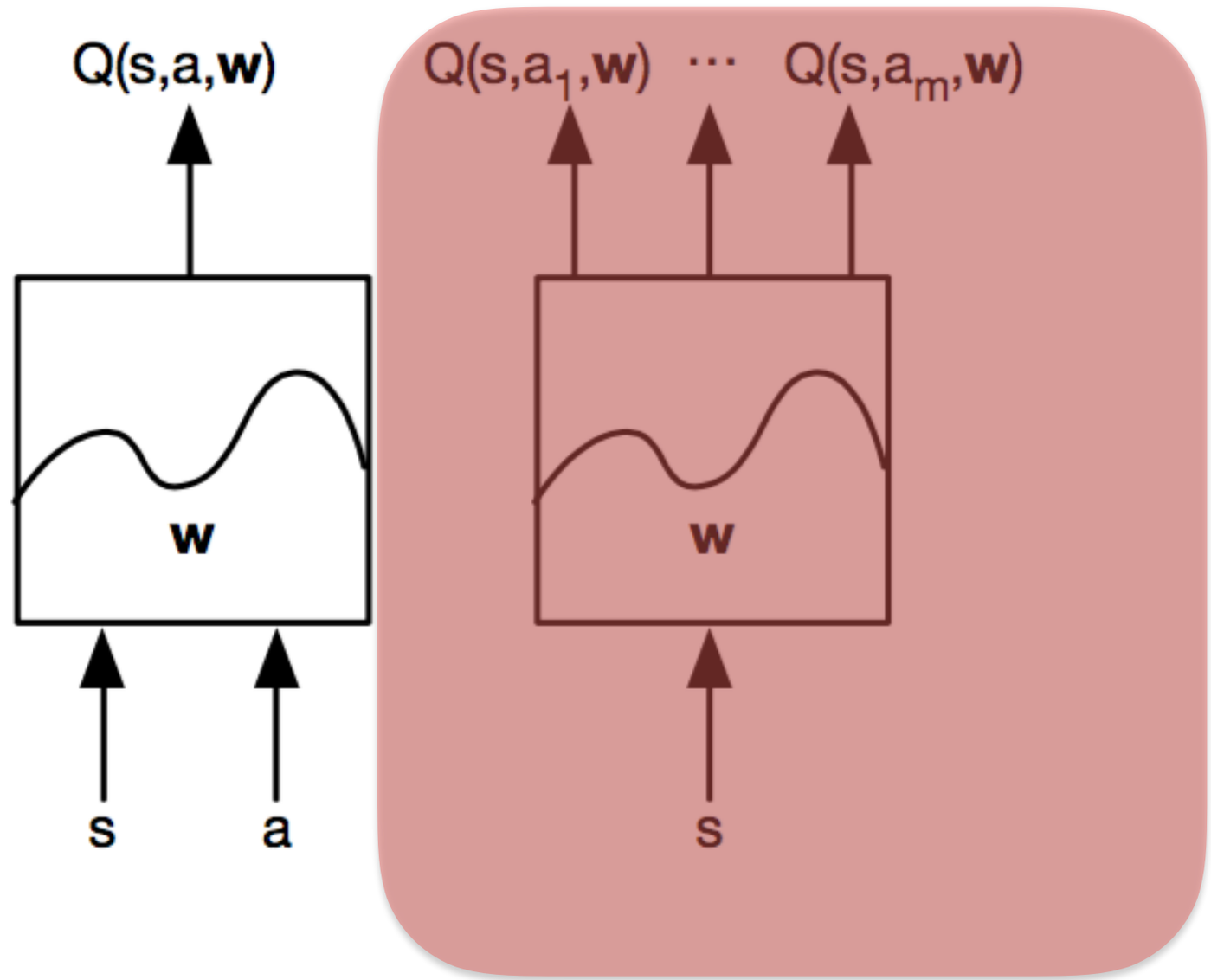
$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \max_{A_{t+1}} \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

Deep Q-Networks (DQNs)

- Represent action-state value function by Q-network with weights w

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$

When would this be preferred?



Q-Learning with FA

- Minimize MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \right)^2$$

- Converges to Q_* using table lookup representation

- But diverges using neural networks due to:
  1. Correlations between samples
  2. Non-stationary targets

Q-Learning

- Minimize MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \right)^2$$

- Converges to Q_* using table lookup representation

- But diverges using neural networks due to:
    1. Correlations between samples
    2. Non-stationary targets

Solutions to both problems in:

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou
Daan Wierstra Martin Riedmiller
DeepMind Technologies

DQN

- To remove correlations, build data-set from agent's own experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

  - Sample experiences from data-set and apply update

$$I = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- To deal with non-stationarity, target parameters w− are held fixed

Experience Replay

- Given experience consisting of ⟨state, value⟩, or <state, action/value> pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Repeat
  - Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

  - Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha(v^\pi - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

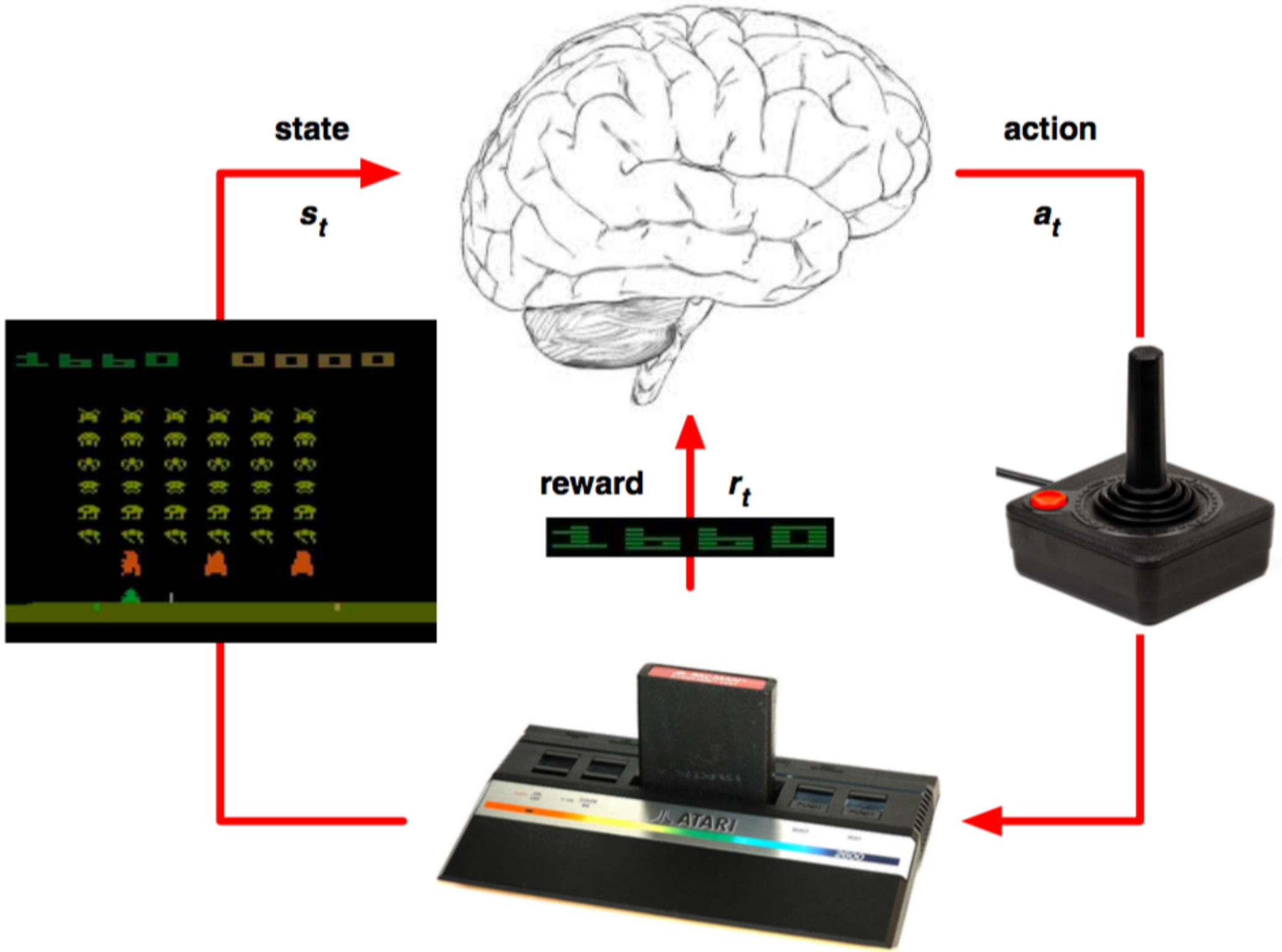
DQNs: Experience Replay

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t,a_t,r_{t+1},s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s,a,r,s′) from D
- Compute Q-learning targets w.r.t. old, fixed parameters w−
- Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}}\right)^2\right]$$

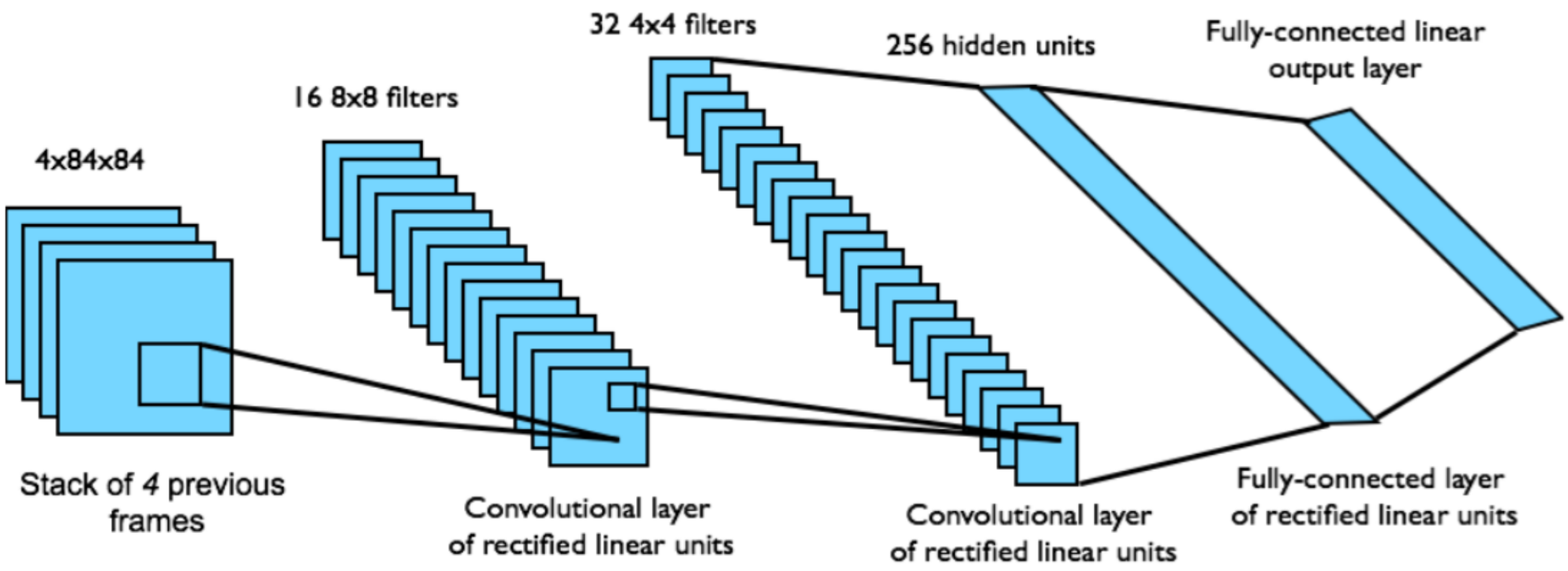
- Use stochastic gradient descent

DQNs in Atari



DQNs in Atari

- End-to-end learning of values Q(s,a) from pixels
- Input observation is stack of raw pixels from last 4 frames
- Output is Q(s,a) for 18 joystick/button positions
- Reward is change in score for that step

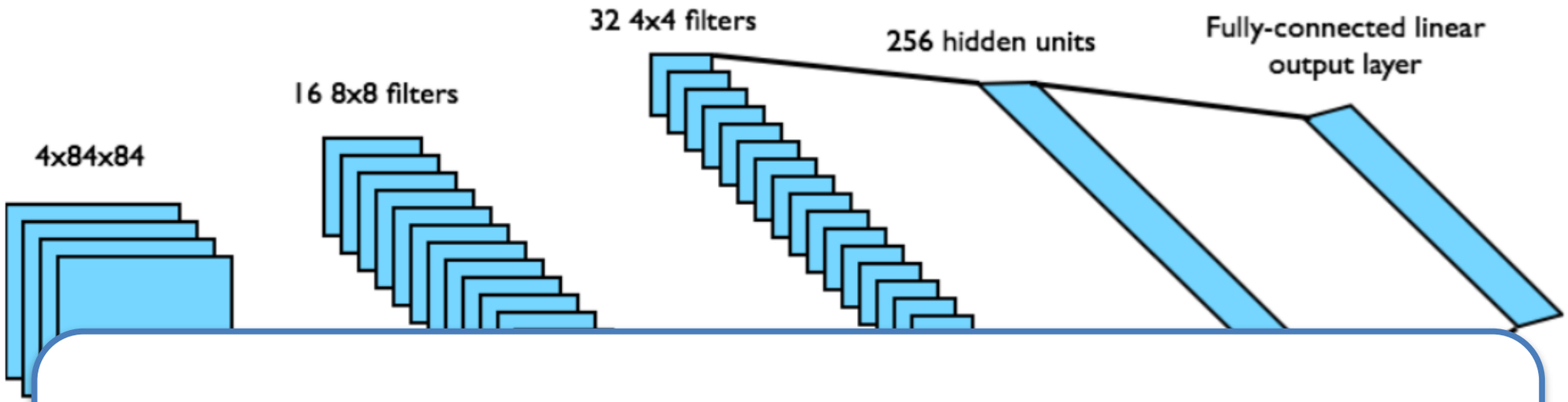


- Network architecture and hyperparameters fixed across all games

Mnih et.al., Nature, 2014

DQNs in Atari

- End-to-end learning of values Q(s,a) from pixels s
- Input observation is stack of raw pixels from last 4 frames
- Output is Q(s,a) for 18 joystick/button positions
- Reward is change in score for that step



DQN source code: sites.google.com/a/deepmind.com/dqn/

- Network architecture and hyperparameters fixed across all games

Mnih et.al., Nature, 2014

Extensions

- Double Q-learning for fighting maximization bias
- Prioritized experience replay
- Dueling Q networks
- Multistep returns
- Value distribution
- Stochastic nets for explorations instead of \epsilon-greedy

Double Tabular Q-Learning

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state},\cdot) = Q_2(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

    else:

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

  $S \leftarrow S'$;
until S is terminal

Hado van Hasselt 2010

- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}}\right)^2\right]$$

Double Deep Q-Learning

- Current Q-network w is used to select actions
- Older Q-network w^- is used to evaluate actions

Action evaluation: w^-

$$l = \left(r + \gamma Q(s', \underbrace{\underset{a'}{\text{argmax}}\, Q(s', a', \mathbf{w})}_{\text{Action selection: w}}, \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

van Hasselt, Guez, Silver, 2015

Prioritized Replay

- Weight experience according to ``surprise” (or error)
- Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

- Stochastic Prioritization

p_i is proportional to DQN error

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

- α determines how much prioritization is used, with α = 0 corresponding to the uniform case.

Schaul, Quan, Antonoglou, Silver, ICLR 2016

Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Multi-step Bootstrapping

CMU 10-403

Katerina Fragkiadaki

Slides from Rich Sutton

Multistep Returns

- Truncated n-step return from a state s_t: $$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}$$

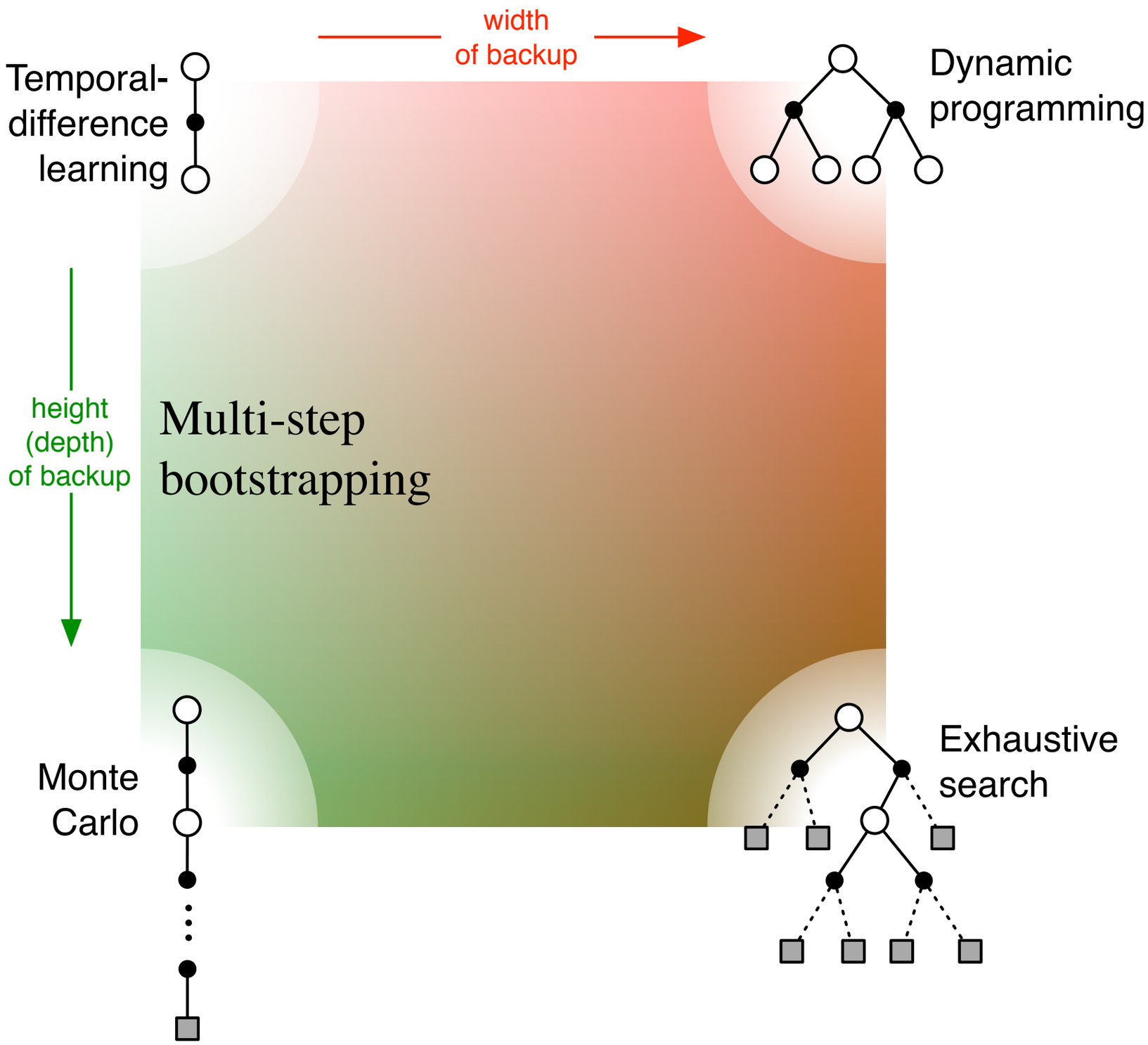
- Multistep Q-learning update rule:

$$I = \left(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} Q(S_{t+n}, a', \mathbf{w}) - Q(s, a, \mathbf{w})\right)^2$$

- Singlestep Q-learning update rule:

$$I = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})\right)^2$$

Unified View



n-step TD Returns/Targets

- Monte Carlo: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

n-step TD Returns/Targets

- Monte Carlo: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

- TD: $G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$
  - Use V_t to estimate remaining return

n-step TD Returns/Targets

- Monte Carlo: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

- TD: $G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$
  - Use V_t to estimate remaining return

- *n*-step TD:
  - 2 step return: $G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_t(S_{t+2})$

n-step TD Returns/Targets

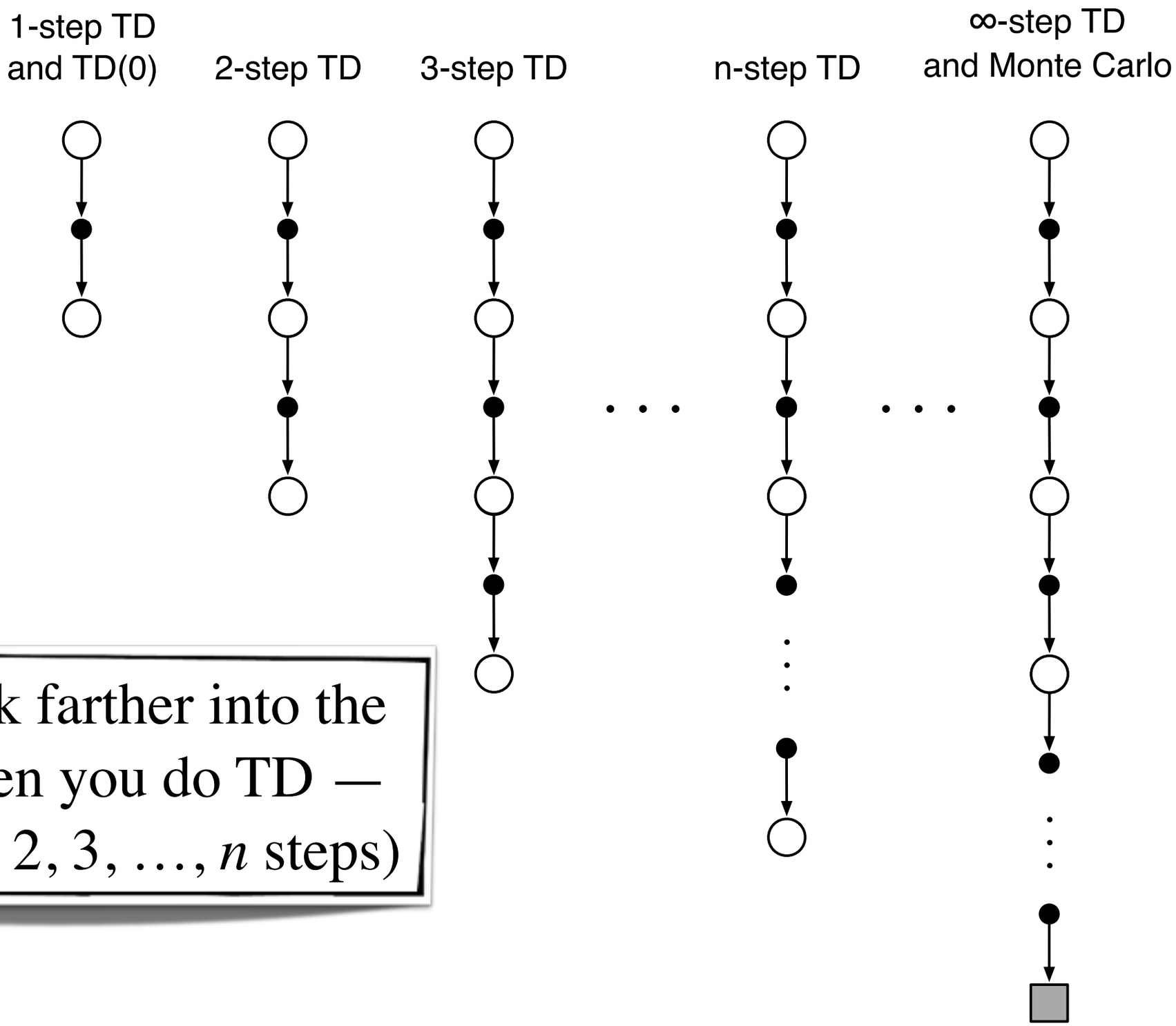
- Monte Carlo: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

- TD: $G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$
  - Use V_t to estimate remaining return

- *n*-step TD:
  - 2 step return: $G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_t(S_{t+2})$
  - *n*-step return: $G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_t(S_{t+n})$

    with $G_t^{(n)} \doteq G_t$ if $t + n \geq T$

n-step TD Prediction



Idea: Look farther into the future when you do TD — backup (1, 2, 3, …, *n* steps)

n-step TD

- Recall the *n*-step return:

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \quad n \geq 1, 0 \leq t < T-n$$

- Of course, this is not available until time *t+n*

- The natural algorithm is thus to wait until then:

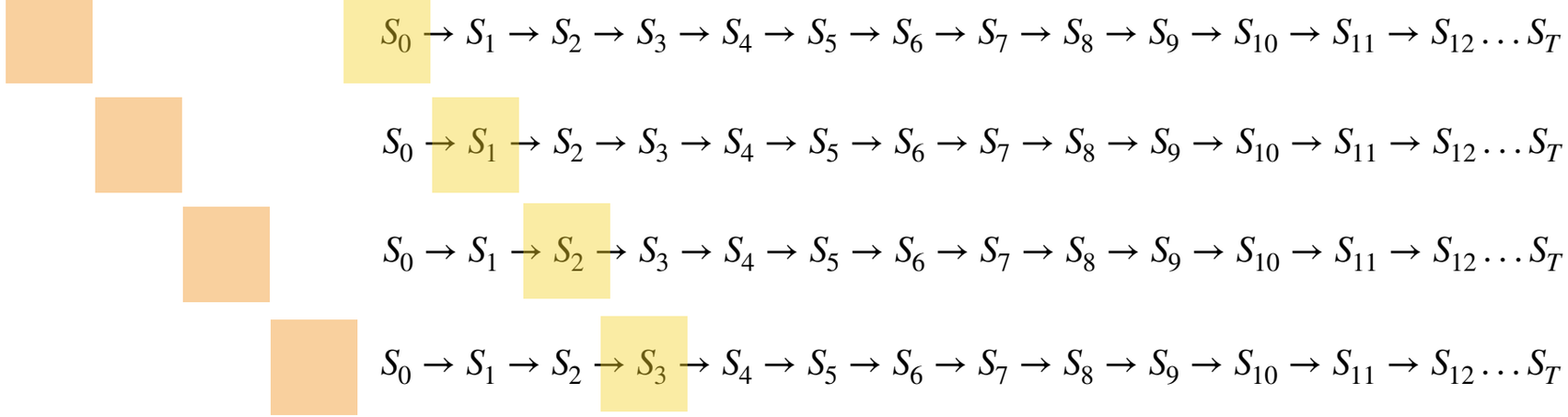
$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_t^{(n)} - V_{t+n-1}(S_t) \right], \qquad 0 \leq t < T$$

- This is called *n*-step TD

n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

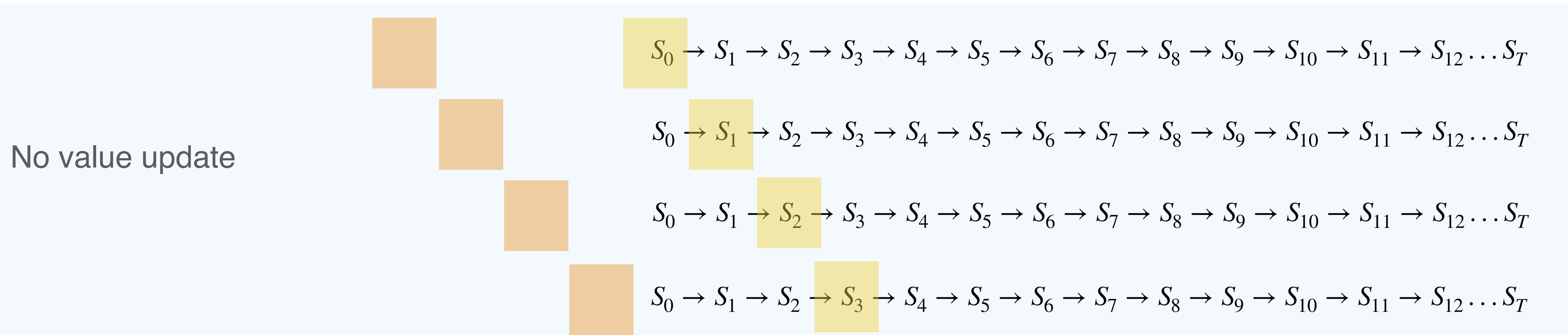
Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | | Take an action according to $\pi(\cdot|S_t)$
  | | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | | If S_{t+1} is terminal, 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_\tau^{(n)})$
  | | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$



n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_\tau^{(n)})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$



n-step TD for estimating $V \approx v_\pi$

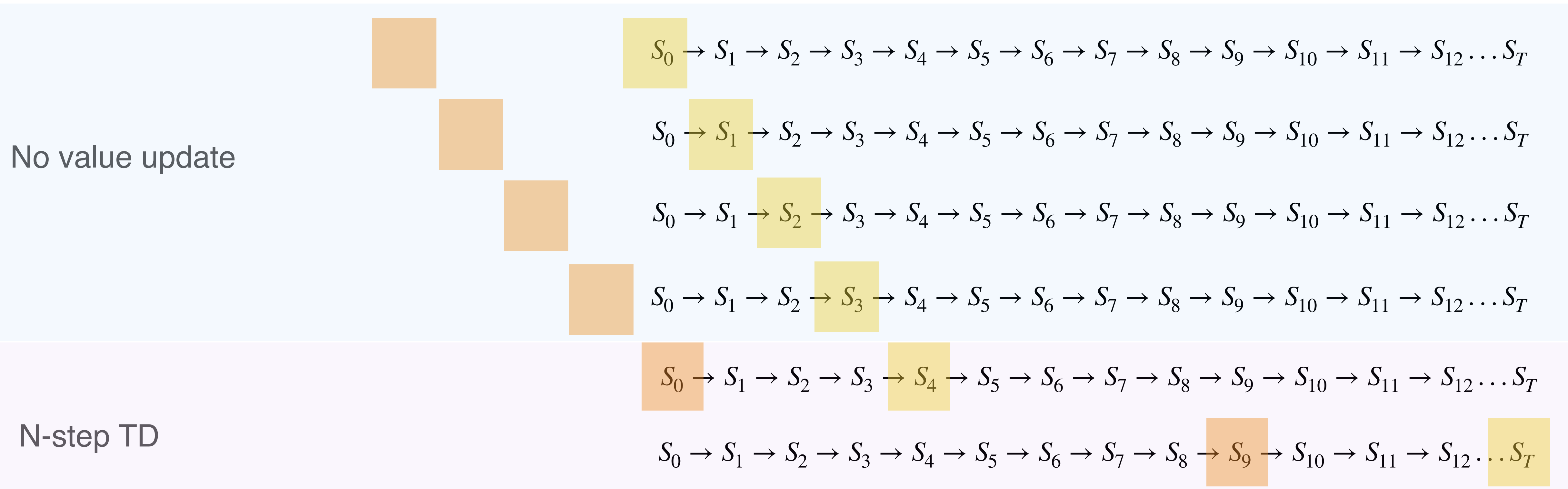
Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \dots$:
  | 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_\tau^{(n)})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
  Until $\tau = T - 1$

n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

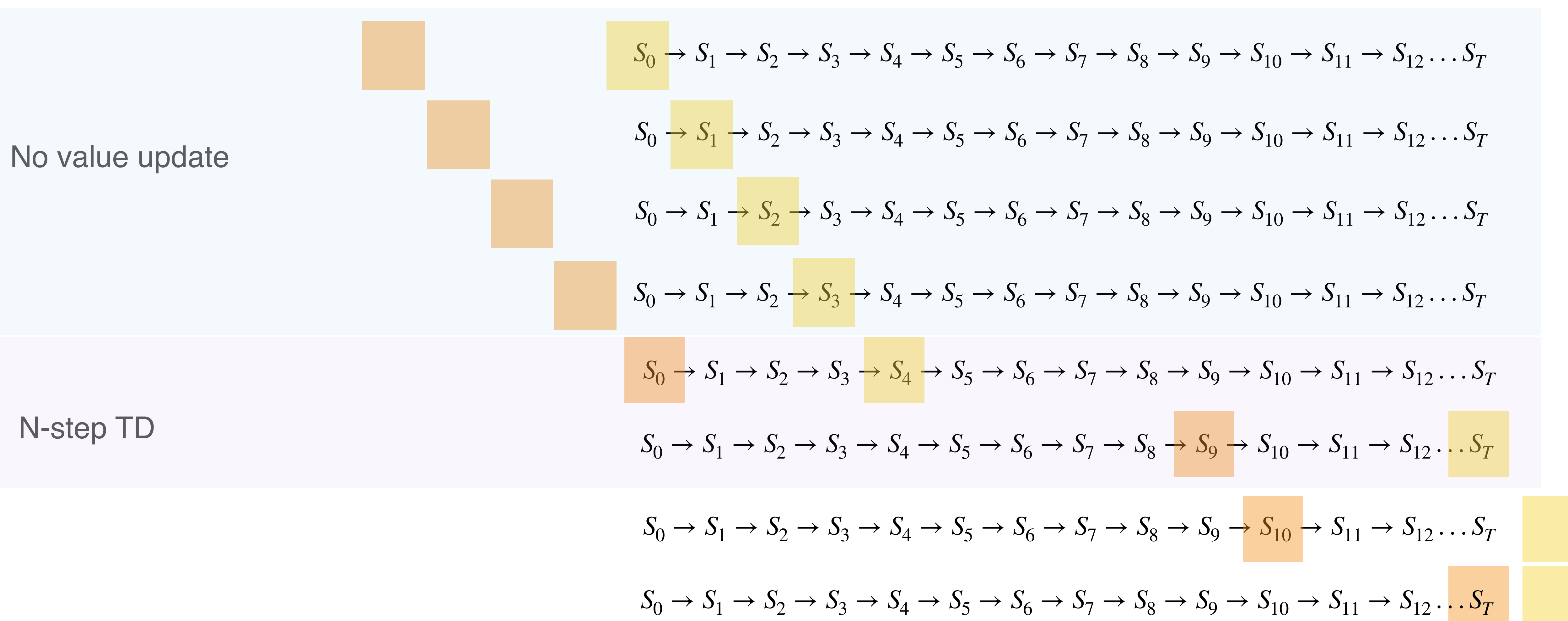
Repeat (for each episode):
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    For $t = 0, 1, 2, \dots$:
    | If $t < T$, then:
    | Take an action according to $\pi(\cdot|S_t)$
    | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | If S_{t+1} 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_\tau^{(n)})$
    | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \, [G - V(S_\tau)]$
    Until $\tau = T - 1$



n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

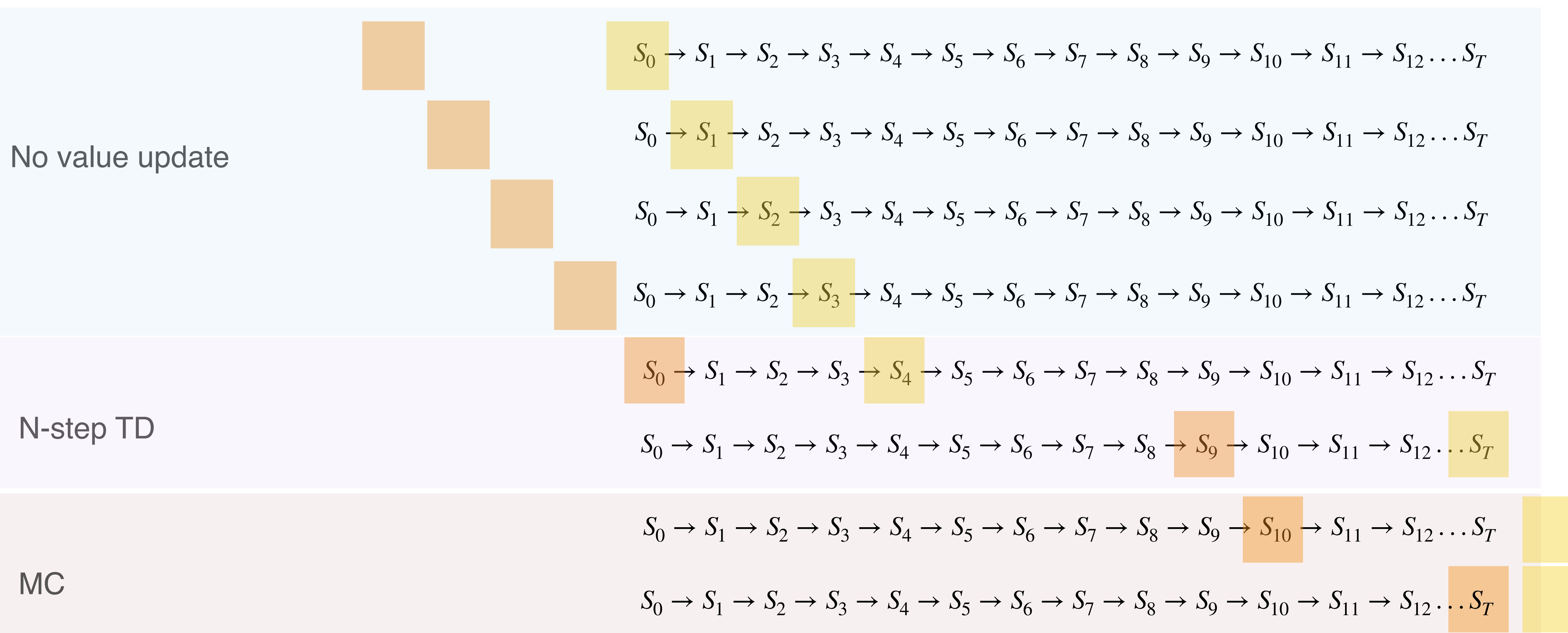
Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_\tau^{(n)})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$



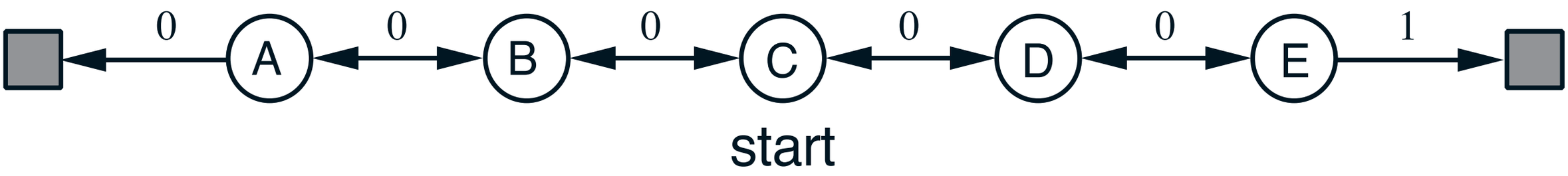
n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_\tau^{(n)})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

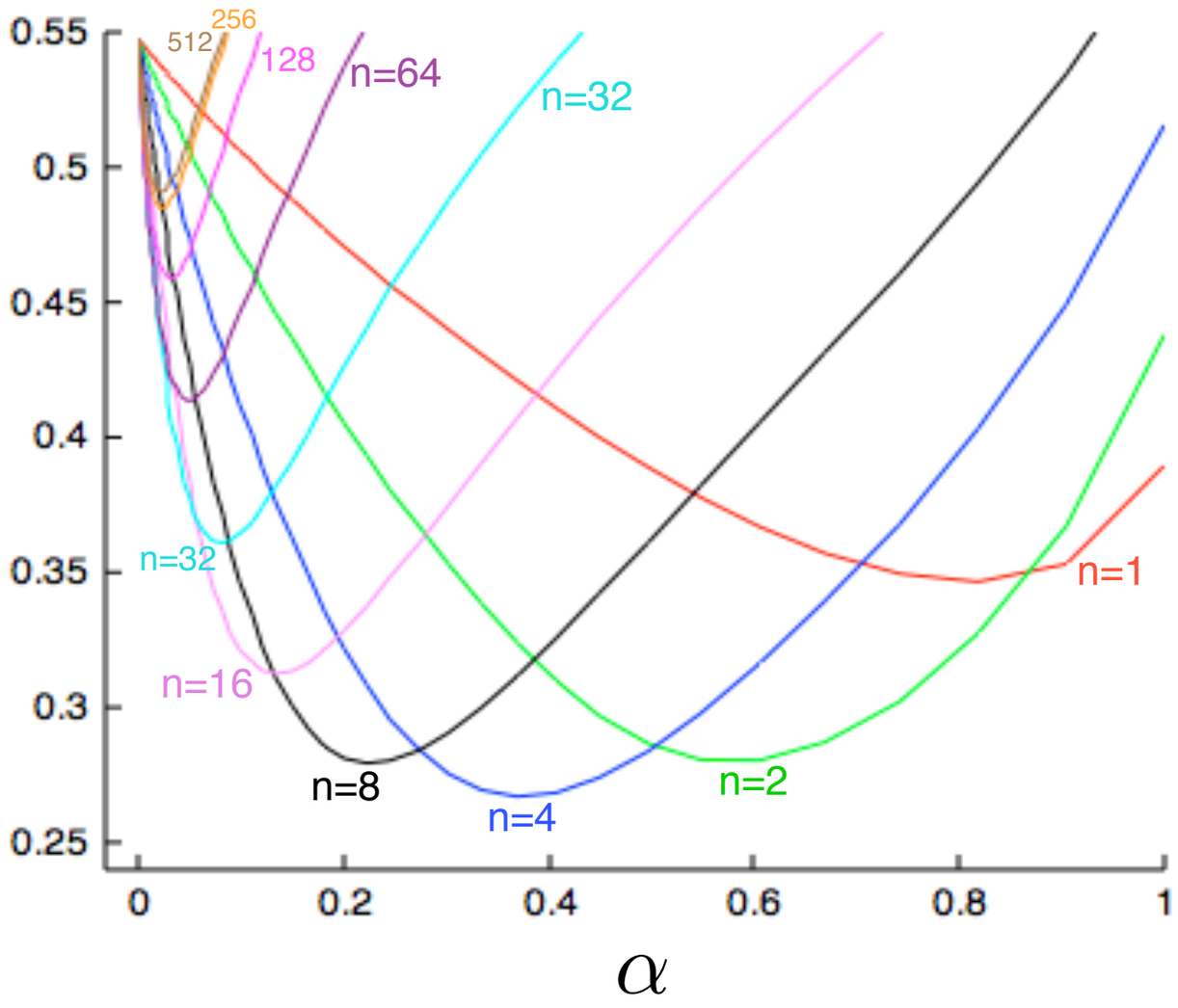


Random Walk Examples



A Larger Example – 19-state Random Walk

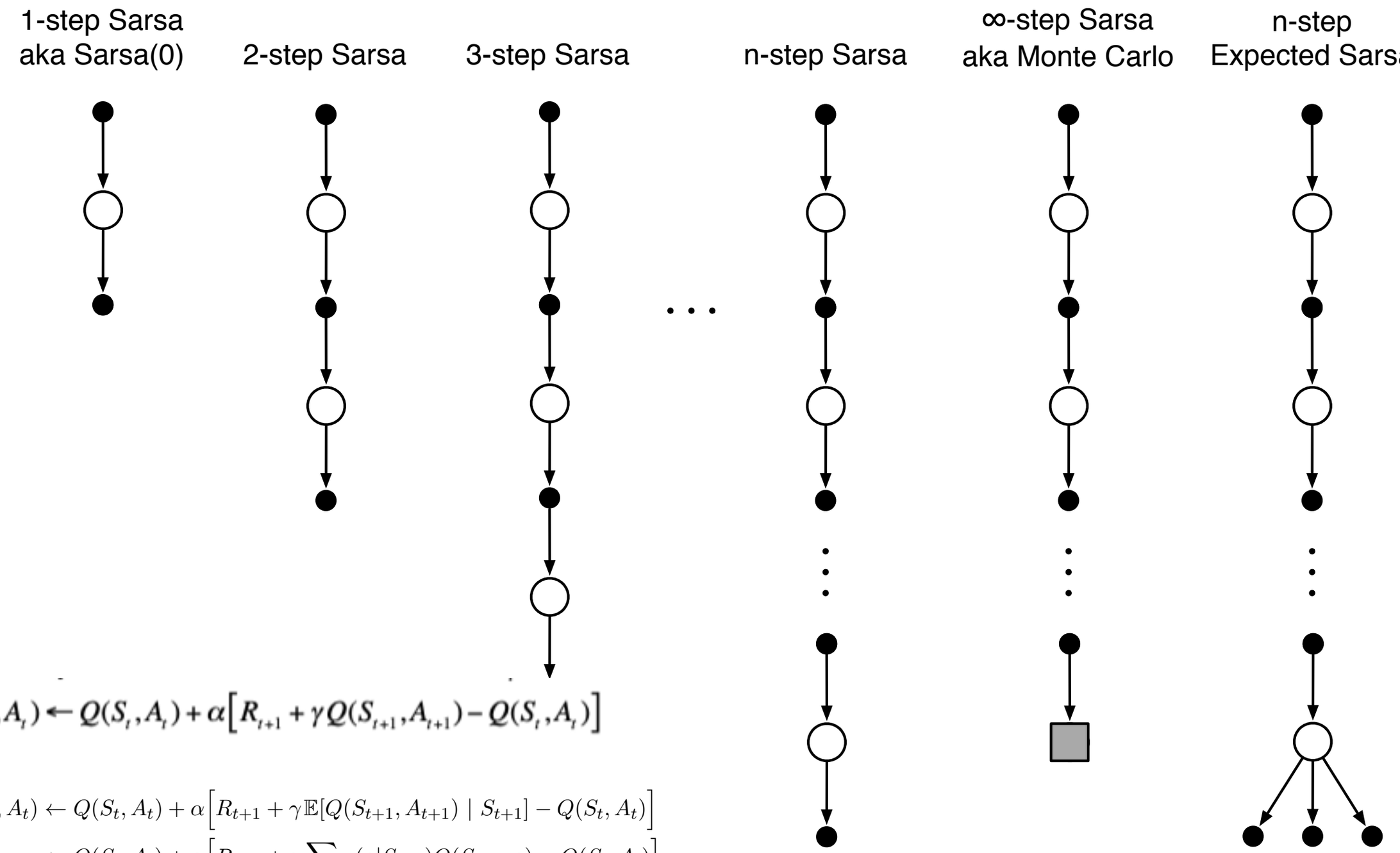
Average RMS error over 19 states and first 10 episodes



n-step TD results

- An intermediate α is best
- An intermediate n is best

It's much the same for action values



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big] \\ &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \end{aligned}$$

On-policy *n*-step Action-value Methods

- Action-value form of *n*-step return

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

- *n*-step Sarsa:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_t^{(n)} - Q_{t+n-1}(S_t, A_t) \right]$$

- *n*-step Expected Sarsa is the same update with a slightly different *n*-step return:

$$G_t^{(n)} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|S_{t+n}) Q_{t+n-1}(S_{t+n}, a)$$

Off-policy *n*-step Methods by Importance Sampling

- Recall the *importance-sampling ratio*:

$$\rho_t^{t+n} \doteq \prod_{k=t}^{\min(t+n-1,T-1)} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}$$

- We get off-policy methods by weighting updates by this ratio

- Off-policy *n*-step TD:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\rho_t^{t+n}\left[G_t^{(n)} - V_{t+n-1}(S_t)\right]$$

- Off-policy *n*-step Sarsa:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1}^{t+n}\left[G_t^{(n)} - Q_{t+n-1}(S_t, A_t)\right]$$

- Off-policy *n*-step Expected Sarsa:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\,\rho_{t+1}^{t+n-1}\left[G_t^{(n)} - Q_{t+n-1}(S_t, A_t)\right]$$

Conclusions Regarding *n*-step Methods

- Generalize Temporal-Difference and Monte Carlo learning methods, sliding from one to the other as n increases
  - $n = 1$ is TD as in Chapter 6
  - $n = \infty$ is MC as in Chapter 5
  - an intermediate n is often much better than either extreme
    - applicable to both continuing and episodic problems
- There is some cost in computation
  - need to remember the last n states
  - learning is delayed by n steps
  - per-step computation is small and uniform, like TD