SMU Lyle School of Engineering

Reinforcement Learning

Prediction and Control with Approximation

Sutton/Barto* Chapter 9-11

Michael Hahsler, SMU

With figures from Sutton/Barto*

*Sutton and Barto, Reinforcement Learning: An Introduction, 2nd edition, MIT Press, Cambridge, MA, 2018

Topics of this Course

- Introduction to reinforcement learning
- Markov decision processes
- Part I: Tabular Methods
  - Dynamic programming
  - Monte Carlo methods
  - Temporal-difference learning
  - Multi-step bootstrapping
  - Planning and learning with tabular methods
- **Part II: Approximate Solution Methods**
  - **Prediction and Control using Approximation**
  - Eligibility Traces
  - Policy Gradient Methods
- Part III: Modern RL Methods
  - Deep Reinforcement Learning
  - Current Applications

Summary of Notation

General

X	capital letters: random variables
x, p	lower-case letters: realizations of random variables or scalar functions
$\boldsymbol{w}$	Bold lower-case letters: real-valued vectors (even if random variables)
$\boldsymbol{W}$	bold capitals: matrices
α	Greek letters: parameters (vectors if in bolt)
$\Pr\{X = x\}$	probability that a random variable X takes on the value x
$X \sim p$	random variable X selected from distribution $p(x) = \Pr\{X = x\}$
$\mathbb{E}[X]$	expectation of a random variable X, i.e., $\mathbb{E}[X] = \sum_x p(x)x$
$\mathrm{argmax}_a f(a)$	a value of action a at which $f(a)$ takes its maximal value

Value Function

G_t	return (cumulative reward) following time t
$G_{t:h}$	return from t to h (discounted and corrected)
$v_\pi(s)$	value of state s under policy π (expected return)
$v_*(s)$	value of state s under the optimal policy
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under the optimal policy
V, V_t	array estimates of state-value function v_π or v_*
Q, Q_t	array estimates of action-value function q_π or v_*

MDP

s, s'	states
a	an action
r	a reward
$\mathcal{S}$	set of all (nonterminal) states, $\mathcal{S}^+$ are all states
$\mathcal{A}(s)$	set of all actions available in state s
γ	discount-rate parameter
t	discrete time step
T	final time step of an episode (a.k.a. horizon)
A_t	random variable for the action at time t
S_t	random variable for the state at time t
R_t	random variable for the reward at time t
$p(s', r \mid s, a)$	probability of transition to state s' and receiving reward r, from state s taking action a.
$p(s' \mid s, a)$	probability of transition to state s' fom state s taking action a.
$r(s, a)$	expected immediate reward from state s after action a.
$r(s, a, s')$	expected immediate reward from state s to s' with action a.
$\pi(a \mid s)$	probability of taking action a in state s under stochastic policy π
$\pi(s)$	action taken in state s under deterministic policy π

Approximate Solution Methods

Motivation

- **Issue**: Tabular methods do not scale for large state spaces.
  a. Memory needed for states.
  b. Many states will never be encountered during learning.

- **Idea**: try to find an approximate value function for policy π.

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

a. The function needs to have a manageable number of parameters (weight vector $\boldsymbol{w}$) and can be calculated from **features** of state s.
b. The function needs to generalize, so states with similar features get a similar value (called **generalization**). Since all states share the weight vector, this function will do this automatically.

Value Function Approximation (Prediction)

- Use **function approximation** (supervised learning).
- We need **training data** in the form of input $\longmapsto$ output pairs.
- We create training pairs $s \longmapsto u$, where the target u is a "backed-up value" representing the desired value. The learning algorithm will update the model to reduce the error, i.e., make the predicted value for s a little more similar to u. Note this change will also affect the prediction for other states!

- Some options for $\mathrm{S_t} \longmapsto u$:
  - MC: $\mathrm{S_t} \longmapsto G_t$
  - TD(0): $\mathrm{S_t} \longmapsto R_{t+1} + \gamma \hat{v}(S_{\mathrm{t+1}}, \mathbf{w}_\mathrm{t})$
  - DP: $\mathrm{S_t} \longmapsto \mathbb{E}_\pi[R_{t+1} + \gamma \hat{v}(S_{\mathrm{t+1}}, \mathbf{w}_\mathrm{t})]$

Prediction Objective ($\overline{VE}$)

- Prediction error for state s:

$$|v_\pi - \hat{v}(s, \boldsymbol{w})|$$

- To calculate the error over all states we need to define how much we care about each state. We use a state distribution function

$$\mu(s) \geq 0; \sum_s \mu(s) = 1$$

- **Mean square value error**:

$$\overline{VE} \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} \mu(s)[v_\pi - \hat{v}(s, \boldsymbol{w})]^2$$

- **Note**: $\mu(s)$ is often the fraction of time spent in s when following π. For continuous tasks, this is a stationary distribution called the on-policy distribution. For episodic tasks the distribution is slightly more complicated to calculate.

Learning Goal

- Find the global optimum $\boldsymbol{w}^*$ with
$$\overline{VE}(\boldsymbol{w}^*) \leq \overline{VE}(\boldsymbol{w})\ \forall\ \boldsymbol{w}$$

- Often we will find a local optimum where the equation above only holds for $\boldsymbol{w}$ in a small neighborhood around $\boldsymbol{w}^*$

- **Note**: finding the optimal value function may not be necessary if we are looking for the (near) optimal policy.

Stochastic-gradient and Semi-gradient Methods

Learning Method

Setup

- We use $\boldsymbol{w} \stackrel{\text{def}}{=} (w_1, w_2, \dots, w_d)^\top$
- Choose $\hat{v}(s, \boldsymbol{w}) \approx v_\pi$ as a **differentiable function** of $\boldsymbol{w}$ for all $s = \mathcal{S}$

Updates

- Following π, we observe at each discrete time step $t = 0, 1, 2, 3, \dots$ a new examples $S_t \mapsto v_\pi(S_t)$.
- We assume examples appear following the distribution μ.
- Update $\boldsymbol{w}$ (we use $\boldsymbol{w}_t$ for the weights at time t) to minimize $\overline{VE}$

Stochastic-gradient Descend (SGD)

Minimize: $\overline{VE} \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} \mu(s)[v_\pi - \hat{v}(s, \boldsymbol{w})]^2$

Gradient of the squared error for the example $S_t \mapsto v_\pi(S_t)$.

Update to change $\boldsymbol{w}$ to reduce $\overline{VE}$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{1}{2}\alpha \nabla[v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)]^2$$

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

Gradient of f with respect to $\boldsymbol{w}$

$$\nabla f(\boldsymbol{w}) = \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \dots, \frac{\partial f(w)}{\partial w_d}\right)^\top$$

- **Stochastic**: because update is on a single example that has been sampled stochastically.
- **Step size $\boldsymbol{\alpha}$**: Since updates of $\boldsymbol{w}$ for one state will change the value for other states as well, we use many small updates (small value of α).

Step Size for SGD

- SGD converges to a local optimum if the **stochastic approximation criterion** is met:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(a) \leq \infty$$

- $\alpha_n(a)$ … step size used after the n-th selection of action a
- $\alpha_n(a) = \frac{1}{n}$ meets the convergence condition and leads to sample averaging.
- $\alpha_n(a) = \alpha$ does not!

Using a Target Estimate for SGD

- Issue: We do not know the value of $v_\pi(S_t)$ for
  $$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha[v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)] \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$

- We can use any target estimate U_t instead
  $$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha[U_t - \hat{v}(S_t, \boldsymbol{w}_t)] \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$

- SGD converges to a local optimum as long as U_t is an **unbiased estimate**. I.e., $\mathbb{E}(U_t|S_t = s) = v\_\pi(s)$.

- Example: MC creates $U_t = G_t$ which an unbiased estimate. SDG with MC is called gradient MC.

Gradient Monte Carlo Algorithm

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

Remember from MC:

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$$

Semi-gradient Methods

Issue:

- **Bootstrapping** methods (e.g., DP and TD) do not create unbiased estimates for U_t because they depend on the current value of $\boldsymbol{w}$.
- Convergence of SDG is not guaranteed because it does not use a true gradient.

Semi-Gradient Methods:

- If we still use the approach, then it is called a semi-gradient method.
- Advantages:
  - are usually faster learners
  - often converge (especially for linear approximators)
  - TD can learn online and use R directly for the update.

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 $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize S
    Loop 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

U_t is a bootstrapped estimate and biased because it uses $\boldsymbol{w}$

Linear Approximation Methods

Linear Regression + Feature Construction

Linear Methods

- Linear approximation of the state value by the inner product

$$\hat{v}(s, \boldsymbol{w}) \stackrel{\text{def}}{=} \boldsymbol{w}^\top \boldsymbol{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

- $\boldsymbol{x}(s)$… feature vector with $x_i \colon \mathcal{S} \rightarrow \mathbb{R}$ (called basis function)\

- Gradient: $\nabla \hat{v}(s, \boldsymbol{w}) = \boldsymbol{x}(s)$
- SGD update: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha[U_t - \hat{v}(S_t, \boldsymbol{w}_t)]\boldsymbol{x}(S_t)$

- Advantages:
  - Simple math
  - Has only one optimum and is **guaranteed to converge** to the global minimum of $\overline{VE}$ if α is reduced following the stochastic approximation condition. This also holds for bootstrapping methods like Semi-gradient TD(0).

Feature Construction

- Linear methods of value estimation =

  interpolation/linear regression task

- Features need to describe the state.
- Linear regression cannot take interactions between features into account: We need to create “interaction features”

- **Idea**: construct features based on the state description.
- We will focus on the most popular feature construction methods that work well for RL:
  - Fourier Basis
  - Tile Coding (coarse coding using tiles)

- Other methods
  - Polynomials
  - Coarse Coding using spherical reception fields
  - Radial Basis Functions

Example: Fourier Basis

- A Fourier series (Fourier transform) represents a periodic function as a weighted sum of sine and cosine basis functions of increasing frequencies.
- State features are not periodic, so we consider a single period
- We choose the period, so we only need to use cosine basis functions.

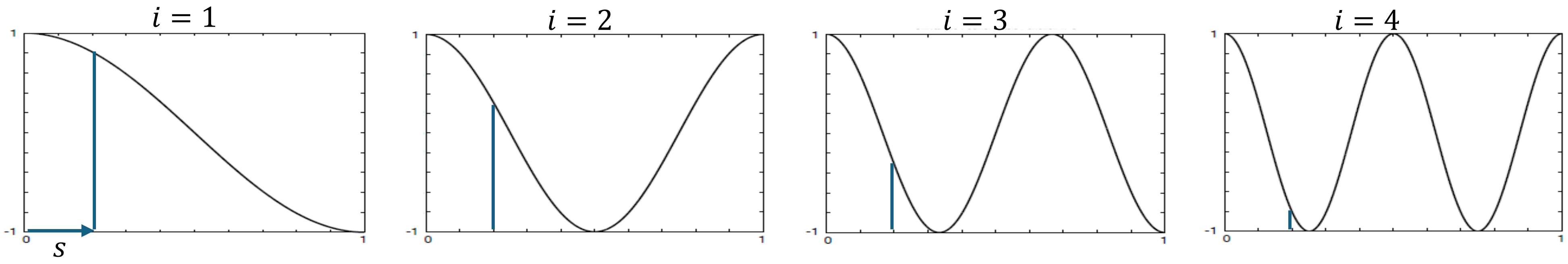
On dimensional case: state is represented by a single number $s \in [0,1]$

$$x_i(s) = \cos(i\pi s)$$

with $i = 0, \ldots, n$ where i is the frequency and n is the order of the Fourier basis function

Note: π is 180° in radians

Example: order-4 Fourier cosine basis creates 4 features plus the intercept for $i = 0$ for linear approximation.



Example: Multi-dimensional Fourier Basis

- State is represented by k numbers: $\boldsymbol{s} = (s_1, s_2, \dots, s_k)$

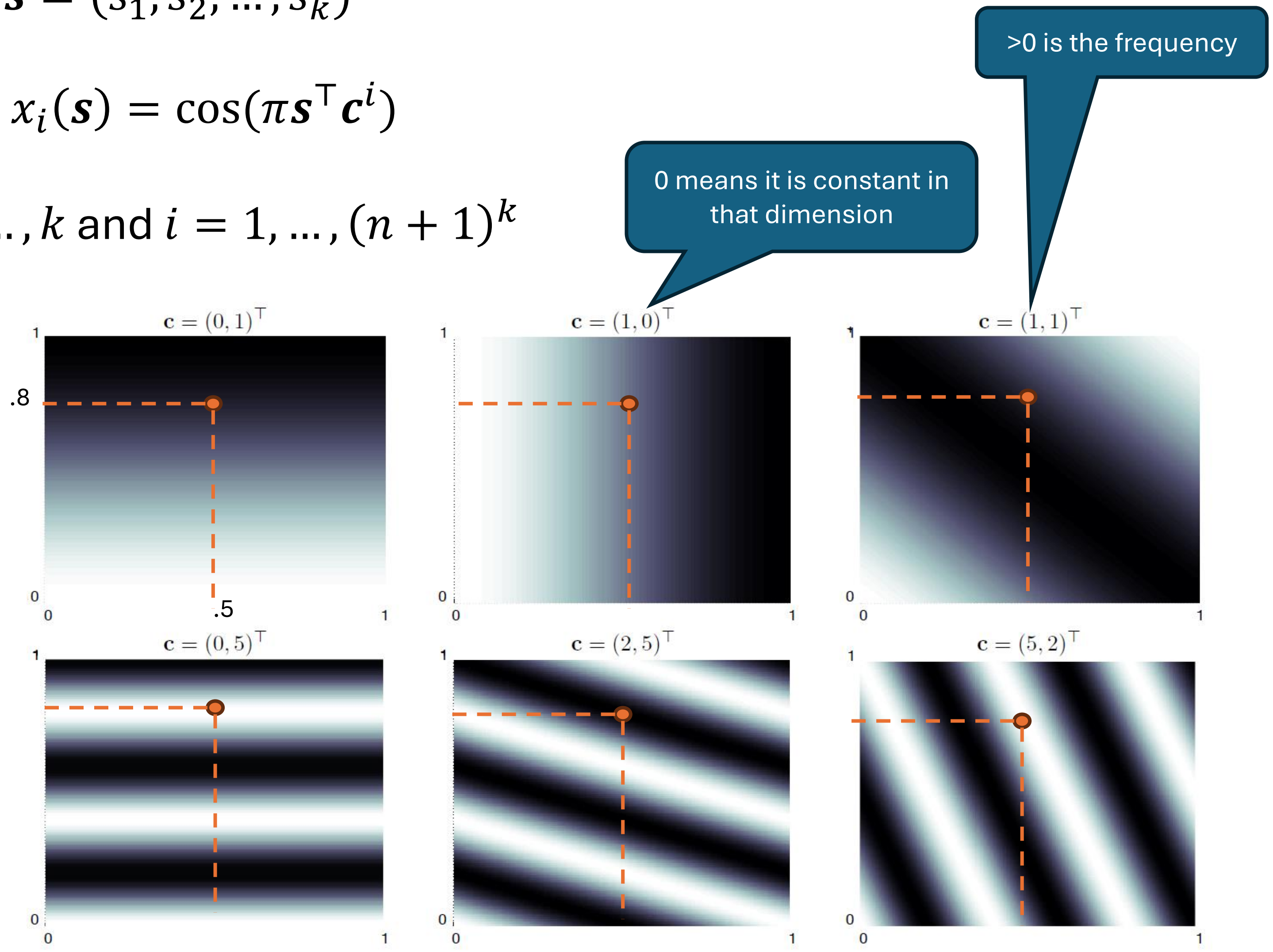
$$x_i(\boldsymbol{s}) = \cos(\pi \boldsymbol{s}^\top \boldsymbol{c}^i)$$

where $\boldsymbol{c}^i = (c_1^i, \dots, c_k^i)$
with $c_j^i \in \{0, \dots, n\}$ for $j = 1, \dots, k$ and $i = 1, \dots, (n+1)^k$

Example: 2-dimensional Fourier cosine features for 6 different $\boldsymbol{c}$.

$$\boldsymbol{s} = (.5, .8)$$

$x_1(\boldsymbol{s}) = \cos(\pi\,(.5, .8)^\top (0,1)) = -.8$
$x_2(\boldsymbol{s}) = \cos(\pi\,(.5, .8)^\top (1,0)) = 0$
$x_3(\boldsymbol{s}) = \cos(\pi\,(.5, .8)^\top (1,1)) = -.6$
$x_4(\boldsymbol{s}) = \cos(\pi\,(.5, .8)^\top (0,5)) = 1$
$x_5(\boldsymbol{s}) = \cos(\pi\,(.5, .8)^\top (2,5)) = -1$
$x_6(\boldsymbol{s}) = \cos(\pi\,(.5, .8)^\top (5,2)) = 1$



1. Features are a combination of both state components, representing interactions.
2. Select only the best features.
3. Use a different α for each feature.

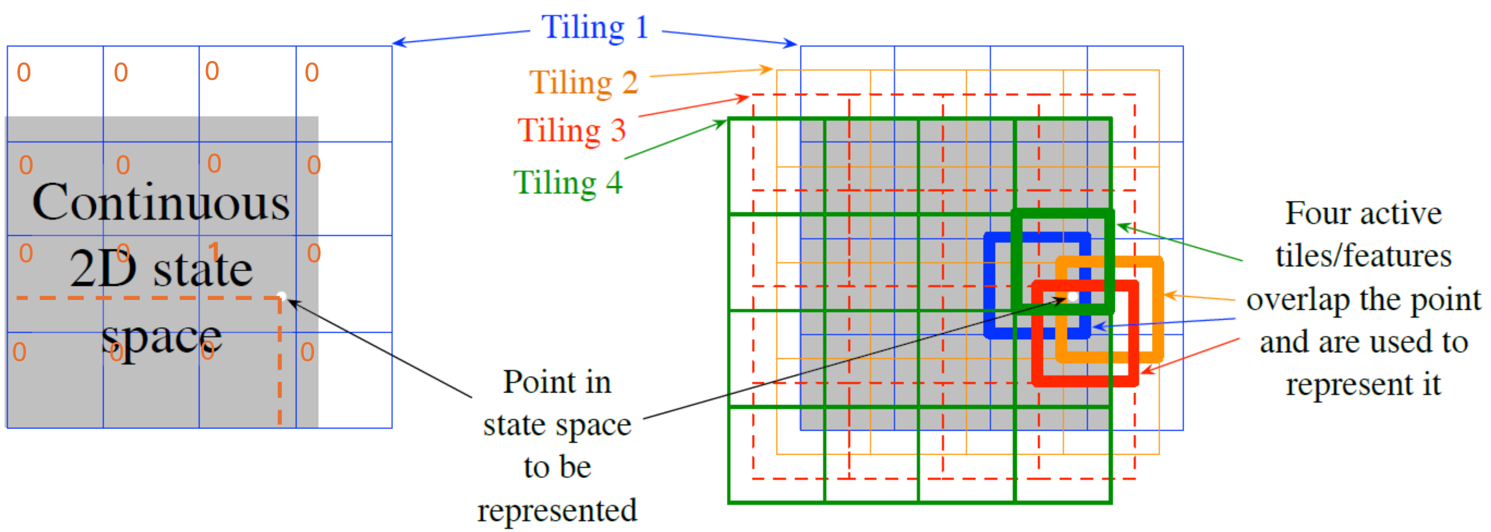
Example: Maze

- TODO

Example: Tile Coding

- Partition the space using several overlapping grids.
- Represent the state by the “active” tiles in the grids.
- Similar states will share active tiles.
- α is typically chosen to be small since a change will also affect neighboring states.
  Popular: $\alpha = \frac{1}{10\ \#\text{tilings}}$

Example:
- State has two components.
- 4 4×4 overlapping tilings give 64 squares.
- State features: 60 0s and only the four active tiles have a 1



Step Size Selection

- For convergence, SGD typically uses a decreasing α, satisfying the **stochastic approximation criterion**

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(a) \leq \infty$$

- E.g., Tabular MC can use: $\alpha_t = \frac{1}{t}$ (sample averaging)

- **Issues**:
  - very slow learning.
  - Not appropriate for TD
  - Does not work for non-stationary problems
  - Not appropriate for function approximation

- Idea: fixed $\alpha = 1/\tau$ means a tabular estimate will approach to the mean target after τ experiences.
- Rule of thumb for linear SDG methods:

$$\alpha \stackrel{\text{def}}{=} \frac{1}{\tau \mathbb{E}[\boldsymbol{x}^\top \boldsymbol{x}]}$$

Average of the squared feature vector length

where $\boldsymbol{x}$ is a random feature vector chosen from the input vector distribution.

Non-Linear Function Approximation

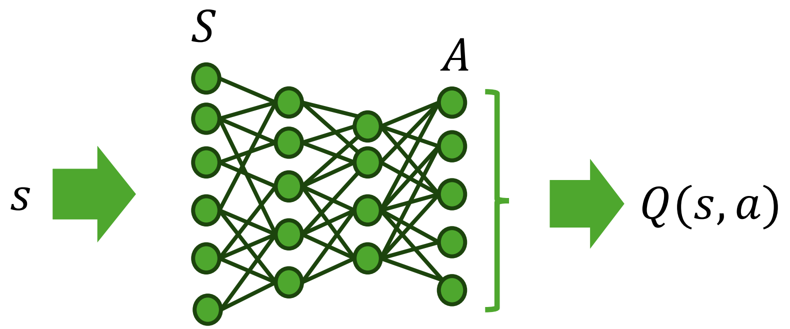
Artificial Neural Networks

Approximation with ANNs

- Uses simple networks with a single hidden layer or deep architectures to approximate the value function.
- Methods:
  - Recurrent networks
  - Convolution networks
  - Transformers
  - Regularization, dropout, weight sharing
  - Deep residual learning...
- Networks are trained like linear models using SGD. For deep networks this uses backpropagation.
- ANNs are non-linear universal approximators so they can learn any value function.

- Issues due to nonlinearity:
  - Instability
  - Convergence is not guaranteed.

- More in Deep Reinforcement Learning (DRL).



Q-network

On-Policy Control with Approximation

Control Problem

- Estimate a parametric approximation for value-action pairs.

$$\hat{q}(s, a, \boldsymbol{w}) \approx q_*(s, a)$$

- Training data is now: $S_t, A_t \mapsto U_t$
- Update: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha[U_t - \hat{q}(S_t, A_t, \boldsymbol{w}_t)] \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$
- U_t can be any approximation:
  - MC's $U_t = G_t$
  - 1-step SARSA's $U_t = R_{t+1} + \gamma \hat{q}(S_t, A_t, \boldsymbol{w}_t)$
  - n-step SARSA
- Policy improvement: update policy with greedy action

$$A^*_{t+1} = \underset{a}{\operatorname{argmax}}\, \hat{q}(S_{t+1}, a, \boldsymbol{w}_t)$$

- For on-policy control, use a soft approximation to the greedy policy (e.g., ϵ-greedy)

Episodic Semi-gradient Sarsa

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

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        If S' is terminal:
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$
        $A \leftarrow A'$

Considerations for Continuing Tasks

- Discounting is used for tabular methods because returns for each state are separately identified and averaged.
- With approximation, returns are shared between states and it is better to focus on average rewards instead of discounting.
- It is often helpful to use differences between the reward and the average returns known as differential returns $R - r(\pi)$.

Summary

Summary

- Approximate the value function using parametrized functions:

$$\hat{v}(s, \boldsymbol{w}) \text{or } \hat{q}(s, a, \boldsymbol{w})$$

- We need to create features that take the interaction of state components into account. Popular are:
  - Fourier basis features
  - Tile coding.

- Use semi-gradient descent to learn $\boldsymbol{w}$.
- Issues are the choice of α.