
“For even the very wise cannot see all ends.”
J.R.R. Tolkien, The Fellowship of the Ring
CMPUT 655
Introduction to RL
Marlos C. Machado
Class 8/12

Plan

- Chapter 9: On-policy Prediction with Approximation.
- Chapter 10: On-policy Control with Approximation.
  - I might not talk about the Average Reward formulation.
- 15:30 to 16:50: Guest Lecture by Andrew Patterson: *Empirical Practices in Reinforcement Learning*.

Reminder

- On the project
  - The project proposal was due on Wednesday. I'll try to mark everything by next week.
  - Three people (a group?) have **not submitted** the project proposal yet.
- There is no scheduled Coursera activity for you to do next week.

Please, interrupt me at any time!

https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Last Class: Linear Function Approximation

- Let $\hat{v}(\mathbf{x}, \mathbf{w}) = \mathbf{x}^\top \mathbf{w}$. We have $\nabla_{\mathbf{w}} \hat{v}(\mathbf{x}, \mathbf{w}) = \mathbf{x}(s)$.
- Thus, $\mathbf{w_{t+1}} \doteq \mathbf{w_t} + \alpha [U_t - \hat{v}(\mathbf{x}, \mathbf{w})] \nabla_{\mathbf{w}} \hat{v}(\mathbf{x}, \mathbf{w})$ becomes:

  $\mathbf{w_{t+1}} \doteq \mathbf{w_t} + \alpha [U_t - \hat{v}(\mathbf{x}, \mathbf{w})]\mathbf{x}$.

Last Class: Polynomials Features

- Doesn't work so well, but they are one of the simplest families of features.
- Suppose an RL problem has states with two numerical dimensions.

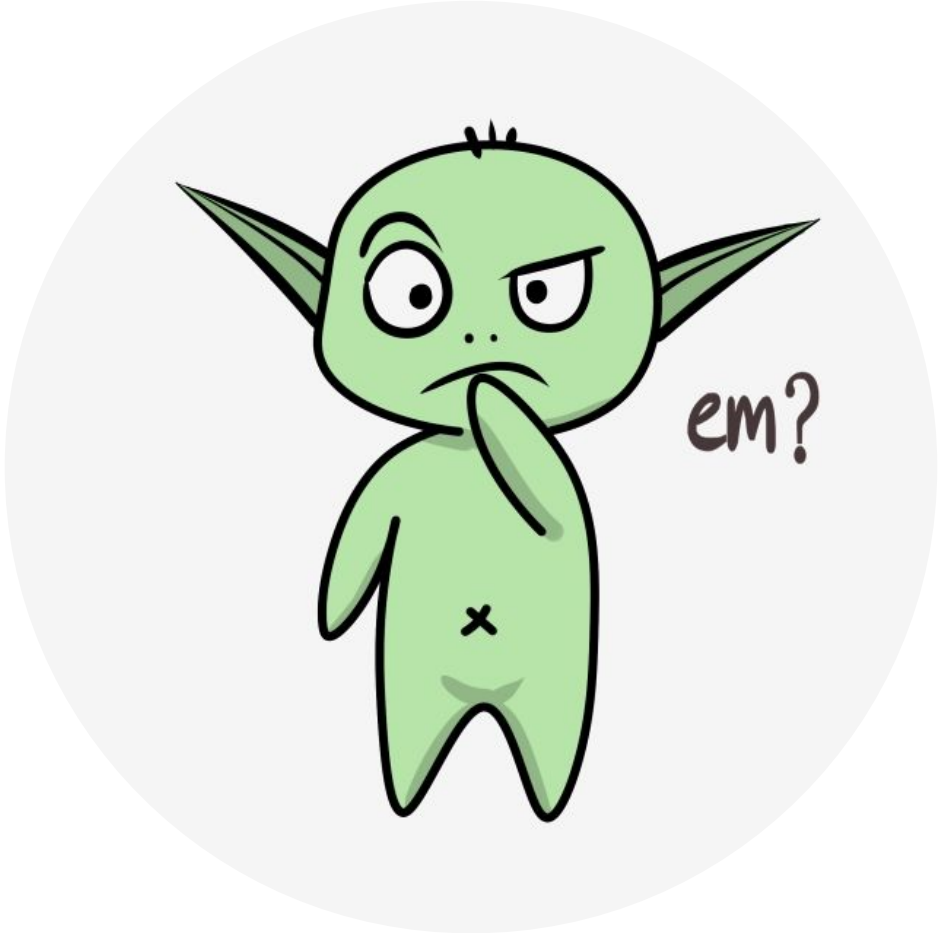
$$\mathbf{x}(s) = (s_1, s_2)^\top$$

But what about interactions? What if both features were zero?

$$\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2)^\top$$

And we can keep going…

$$\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_2^2)^\top$$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Fourier Basis

- Fourier series expresses periodic functions as weighted sums of sine and cosine basis functions (features) of different frequencies.
- Fourier features are easy to use and can perform well in several RL problems.
- When using the Fourier series and the more general Fourier transform, with enough basis functions essentially any function can be approximated as accurately as desired.

Fourier Basis

- Consider the one-dimensional case. The cosine basis consists of the *n + 1* features

$$x_i(s) = \cos(i\pi s), \quad s \in [0, 1],$$

for i = 0, …, n. The figure below shows one-dimensional Fourier cosine features x_i, for i = 1, 2, 3, 4; x_0 is a constant function.

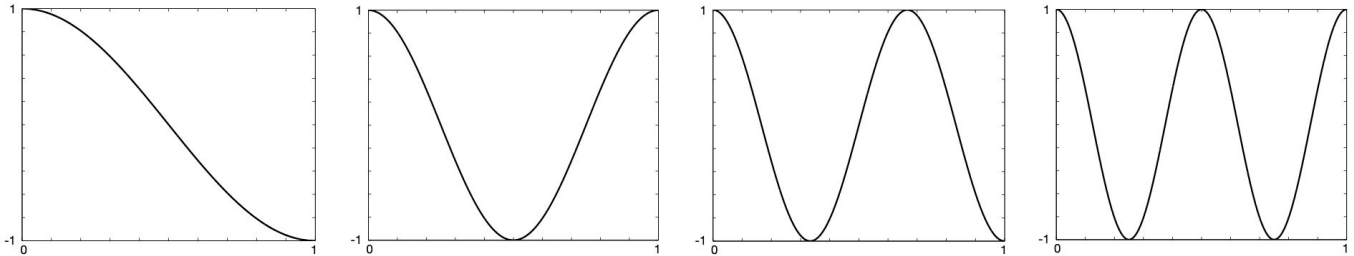


Figure 9.3: One-dimensional Fourier cosine-basis features x_i, $i = 1, 2, 3, 4$, for approximating functions over the interval $[0, 1]$. After Konidaris et al. (2011).

Fourier Basis Beyond One Dimension

Suppose each state s corresponds to a vector of k numbers, $\mathbf{s} = (s_1, s_2, ..., s_k)^\top$, with each $s_i \in [0, 1]$. The ith feature in the order-n Fourier cosine basis can then be written

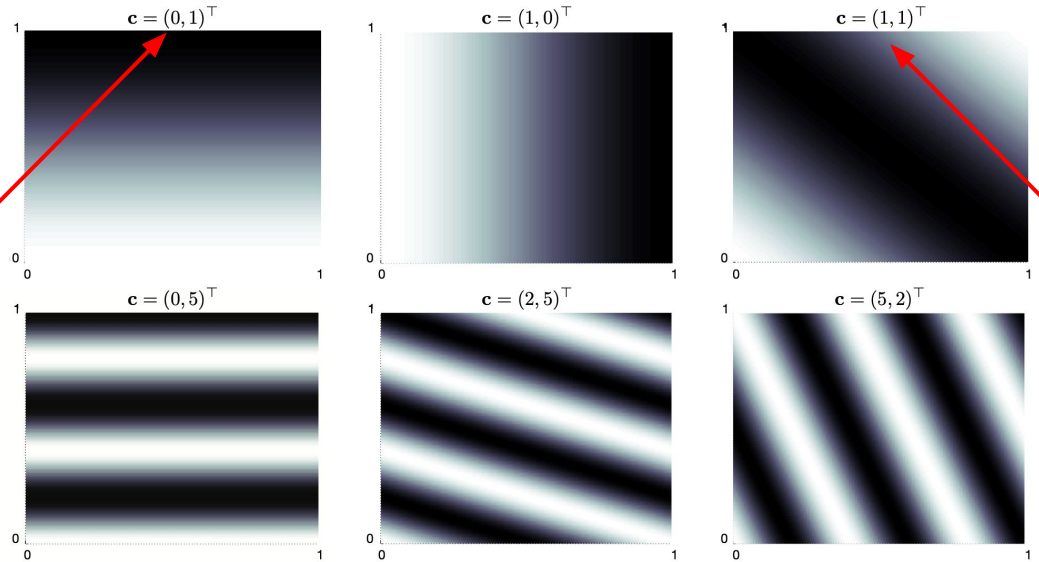
$$x_i(s) = \cos\left(\pi \mathbf{s}^\top \mathbf{c}^i\right), \tag{9.18}$$

where $\mathbf{c}^i = (c_1^i, \ldots, c_k^i)^\top$, with $c_j^i \in \{0, \ldots, n\}$ for $j = 1, \ldots, k$ and $i = 1, \ldots, (n+1)^k$. This defines a feature for each of the $(n+1)^k$ possible integer vectors $\mathbf{c}^i$. The inner product $\mathbf{s}^\top \mathbf{c}^i$ has the effect of assigning an integer in $\{0, \ldots, n\}$ to each dimension of $\mathbf{s}$. As in the one-dimensional case, this integer determines the feature's frequency along that dimension. The features can of course be shifted and scaled to suit the bounded state space of a particular application.

Fourier Basis – Example

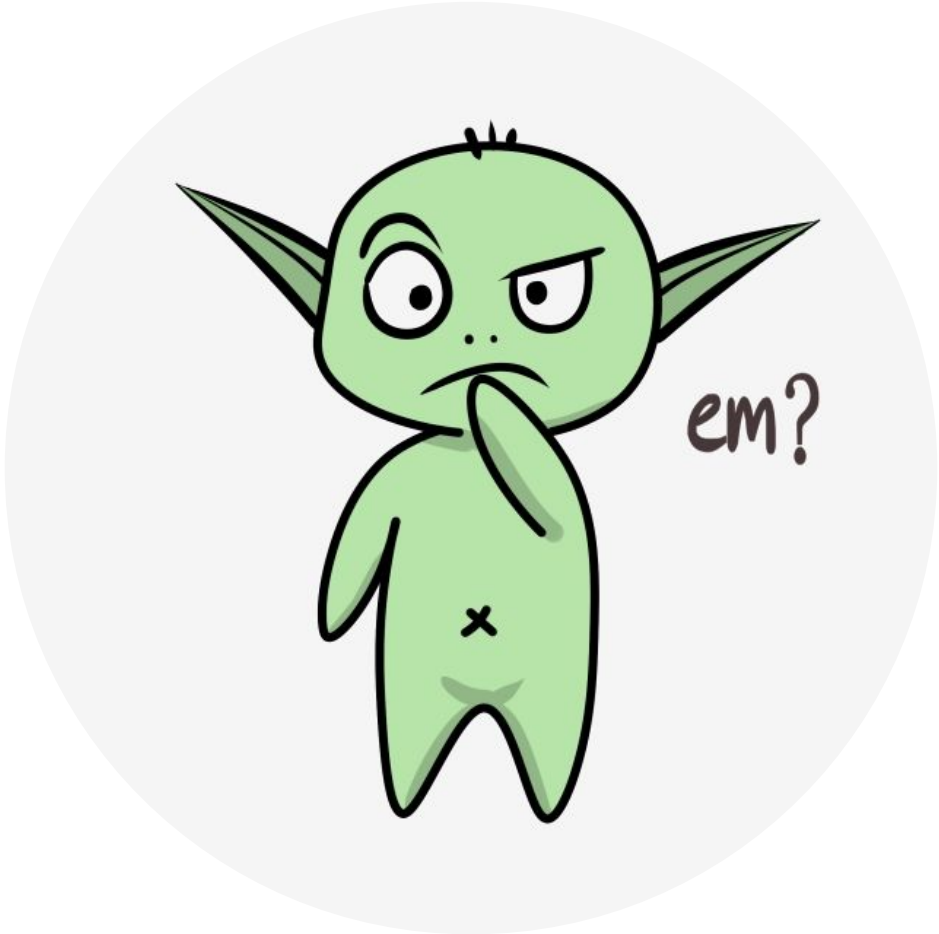
- Consider representing a state as a vector of 2 numbers (k = 2), where each $\mathbf{c}^i = (c^i_1, c^i_2)^\top$.

The feature is constant over the first dimension and varies over the second dimension depending on c_2.



The feature varies along both dimensions and represents an interaction between the two state variables

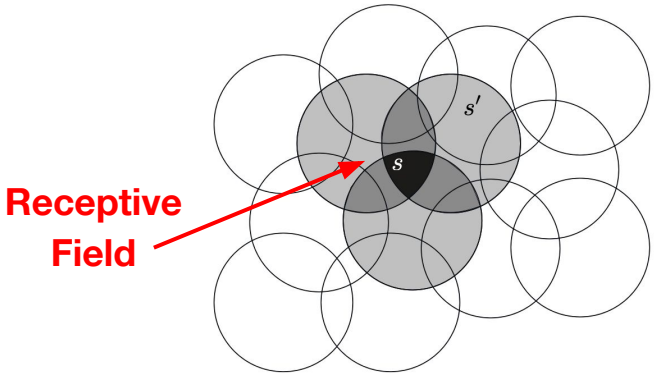
Figure 9.4: A selection of six two-dimensional Fourier cosine features, each labeled by the vector $\mathbf{c}^i$ that defines it (s_1 is the horizontal axis, and $\mathbf{c}^i$ is shown with the index i omitted). After Konidaris et al. (2011).



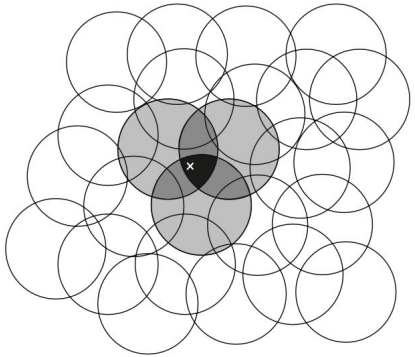
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Coarse Coding

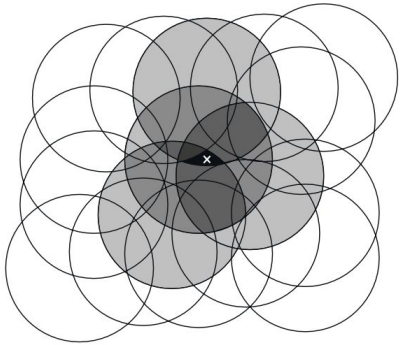
- Consider a task in which the natural representation of the state set is a continuous two- dimensional space.
- We define binary features indicating whether a state is present or not in a specific circle.



The shape defines generalization



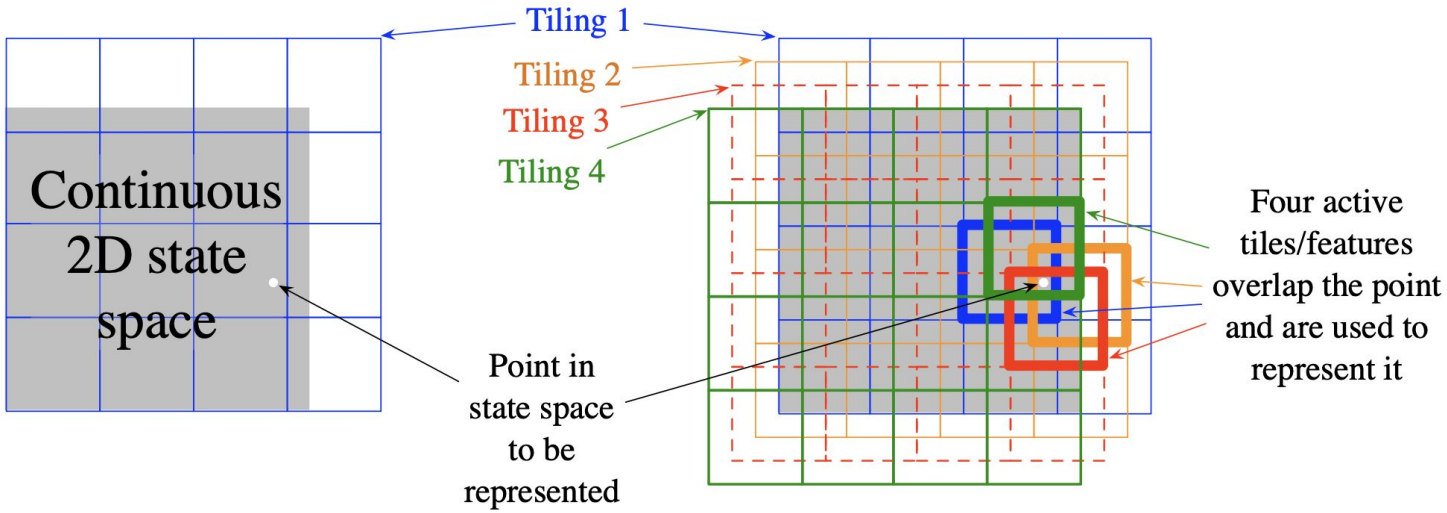
Narrow generalization



Broad generalization

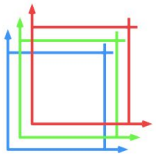
Tile Coding

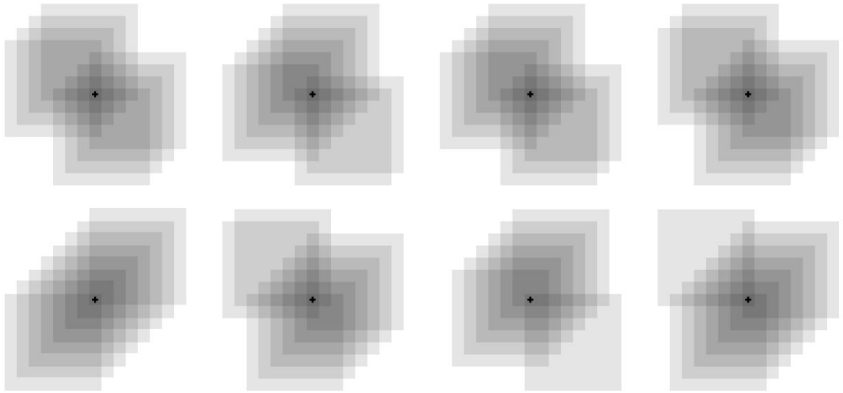
- Tile coding is a form of coarse coding for multi-dimensional continuous spaces (with a fixed number of active features per timestep).



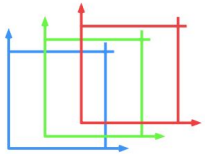
Tile Coding

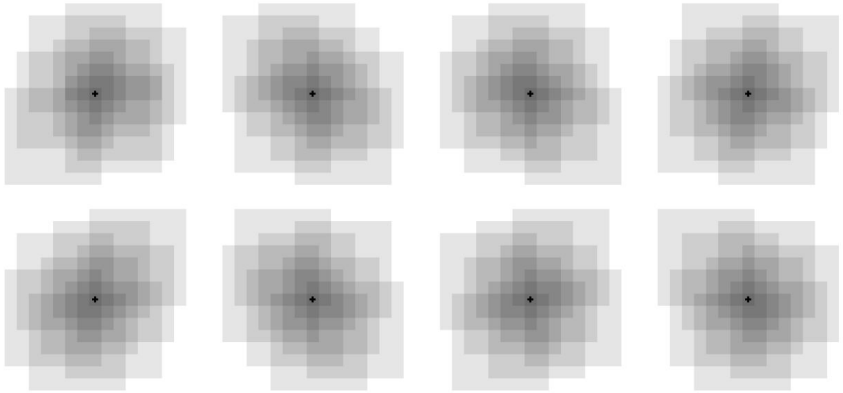
Possible generalizations for uniformly offset tilings

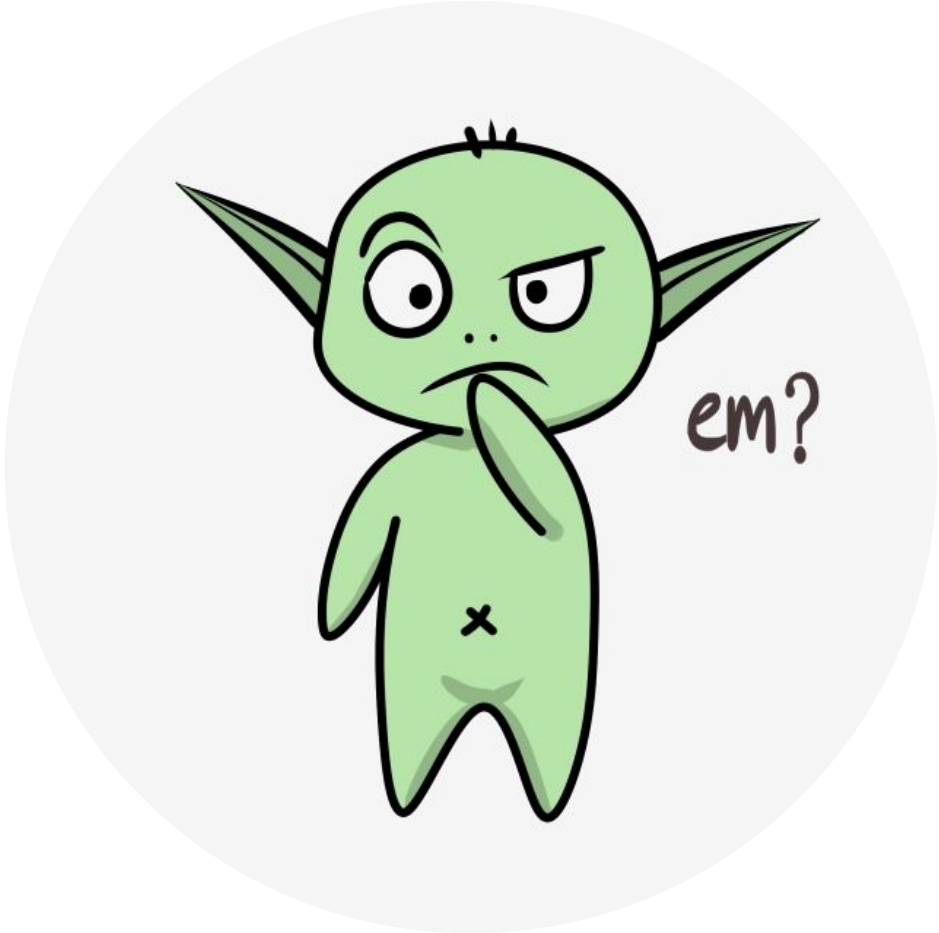




Possible generalizations for asymmetrically offset tilings





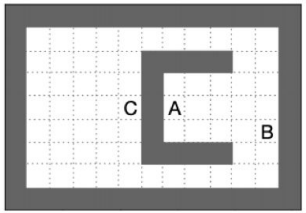


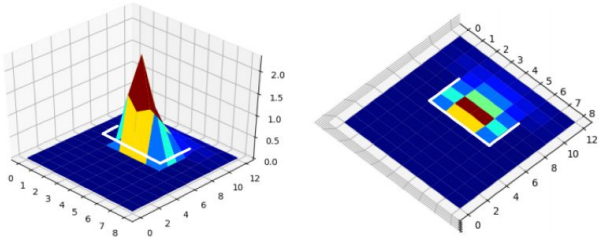
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

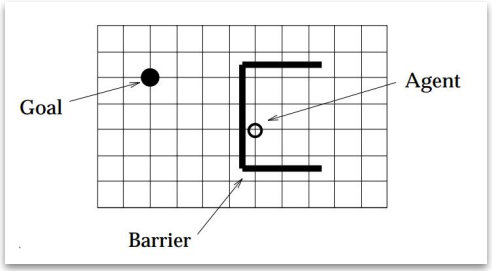
It Isn't that We do Function Approximation Because We Cannot do Tabular Reinforcement Learning

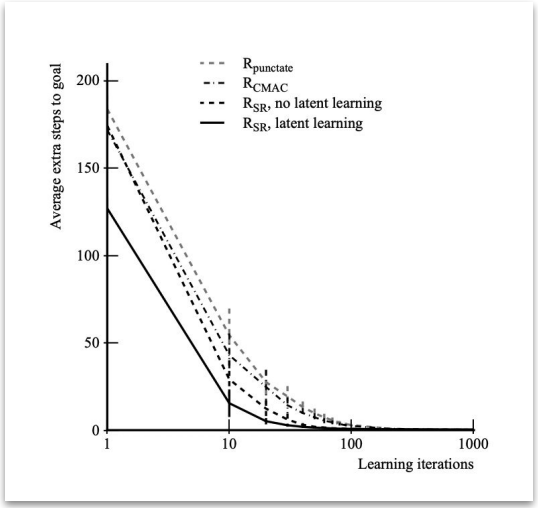
- Successor Representation [Dayan, Neural Computation 1993].

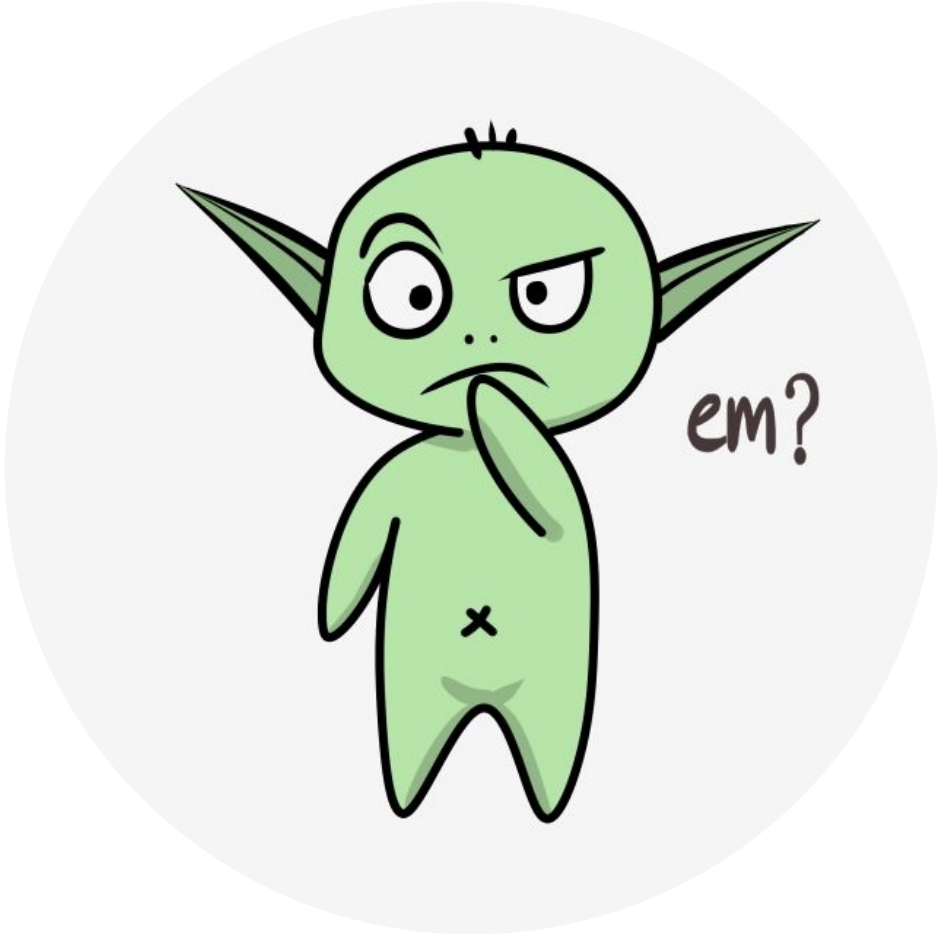
$$\Psi_\pi(s, s') = \mathbb{E}_\pi \left[\sum_t \gamma^t \mathbf{1}_{S_t = s'} | S_0 = s \right]$$









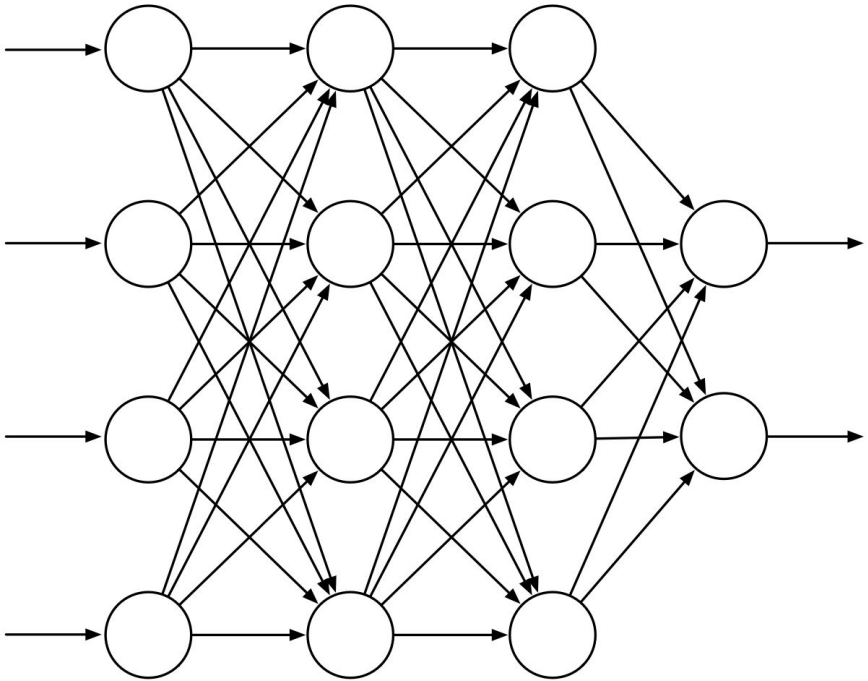


https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

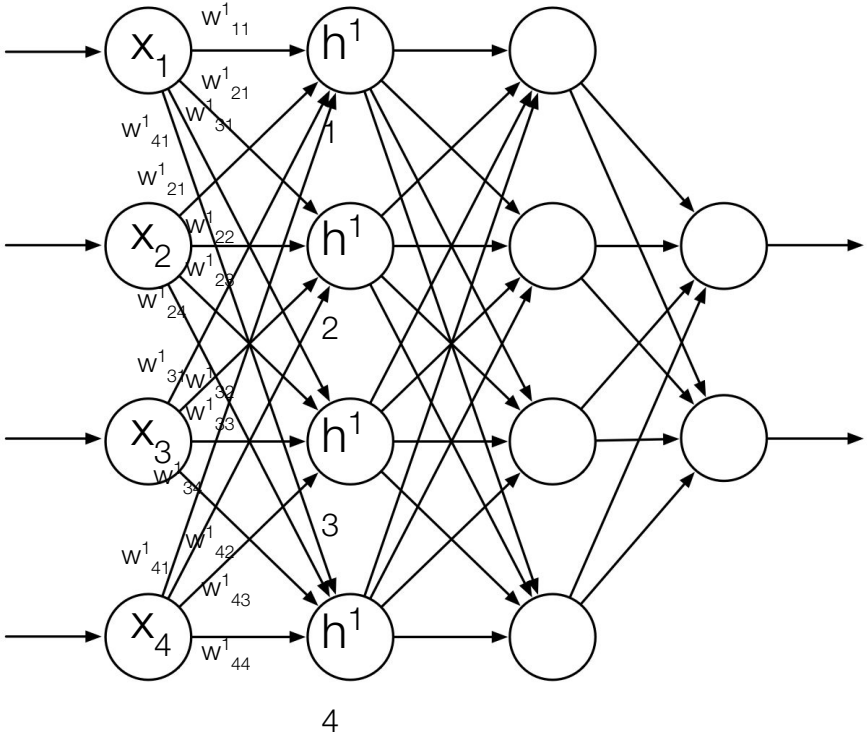
Nonlinear Function Approximation: Artificial Neural Networks

- The basics of deep reinforcement learning.
- Idea: Instead of using linear features, we feed the “raw” input to a neural network and ask it to predict the state (or state-action) value function.

Neural Networks



Neural Networks



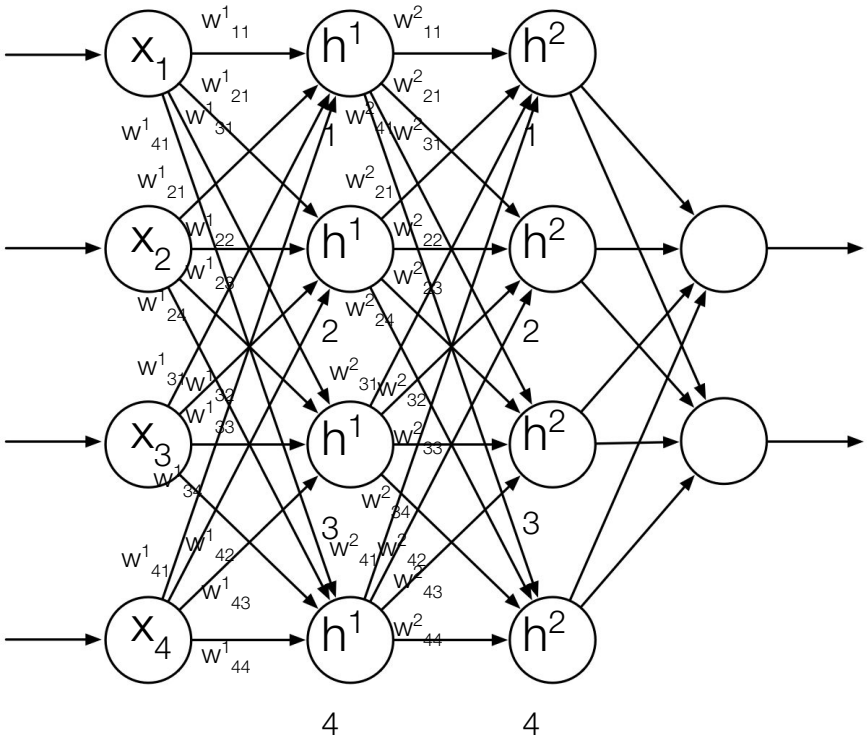
The activation function introduces non-linearity

E.g.: *f(x) = max(0, x)*

$\mathbf{h^1} = \text{act}(\mathbf{xW^1} + \mathbf{b^1})$

s.t. $h^1_1 = x_1 w^1_{11} + x_2 w^1_{21} + x_3 w^1_{31} + x_4 w^1_{41} + B^1$

Neural Networks



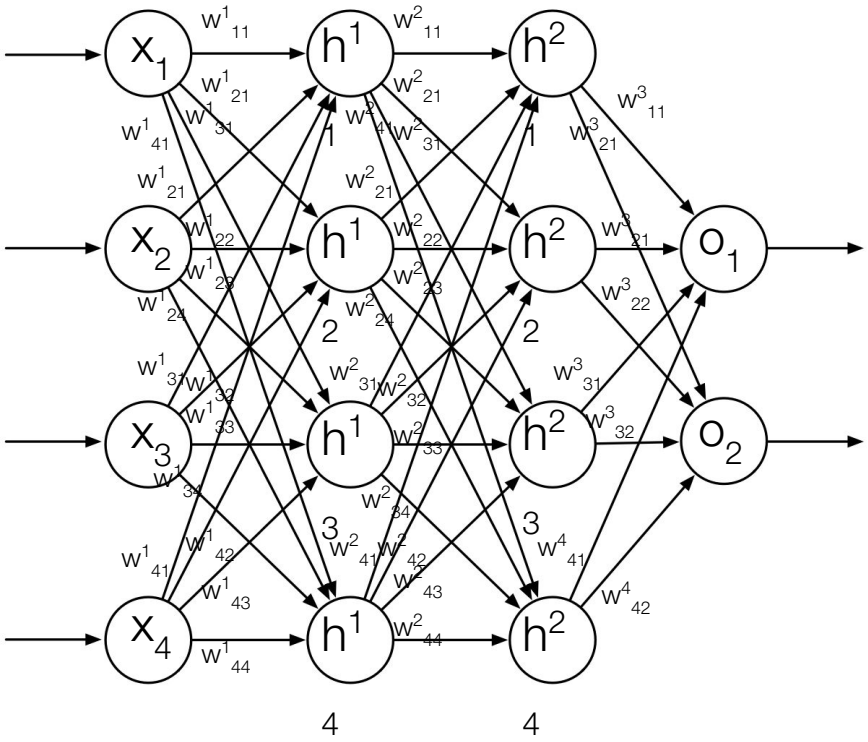
$\mathbf{h^1} = \text{act}(\mathbf{xW^1} + \mathbf{b^1})$

s.t. $h^1_1 = x_1 w^1_{11} + x_2 w^1_{21} + x_3 w^1_{31} + x_4 w^1_{41} + B^1$

$\mathbf{h^2} = \text{act}(\mathbf{h^1 W^2} + \mathbf{b^2})$

s.t. $h^2_1 = h^1_1 w^2_{11} + h^1_2 w^2_{21} + h^1_3 w^2_{31} + h^1_4 w^2_{41} + B^2$

Neural Networks



$\mathbf{h^1} = act(\mathbf{xW^1} + \mathbf{b^1})$

s.t. $h^1_1 = x_1 w^1_{11} + x_2 w^1_{21} + x_3 w^1_{31} + x_4 w^1_{41} + B^1$

$\mathbf{h^2} = act(\mathbf{h^1 W^2} + \mathbf{b^2})$

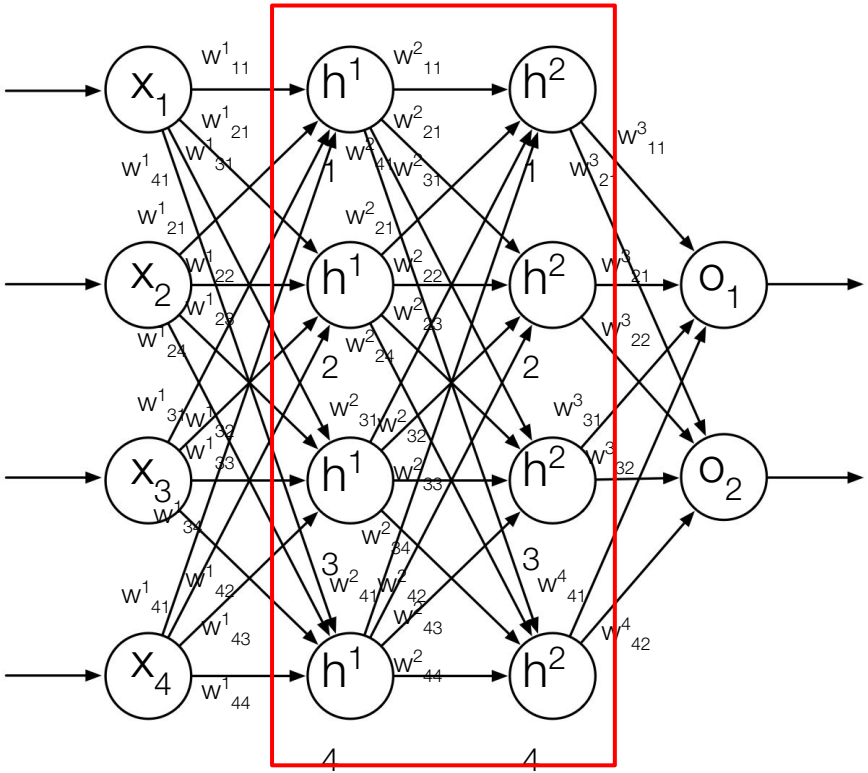
s.t. $h^2_1 = h^1_1 w^2_{11} + h^1_2 w^2_{21} + h^1_3 w^2_{31} + h^1_4 w^2_{41} + B^2$

$\mathbf{o} = act(\mathbf{h^2 W^3} + \mathbf{b^3})$

s.t. $o_1 = h^2_1 w^3_{11} + h^2_2 w^3_{21} + h^2_3 w^3_{31} + h^2_4 w^3_{41} + B^2$

$\mathbf{o} = act(act(act(\mathbf{xW^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2})\mathbf{W^3} + \mathbf{b^3})$

Neural Networks



$\mathbf{h^1} = act(\mathbf{xW^1} + \mathbf{b^1})$

s.t. $h^1_1 = x_1 w^1_{11} + x_2 w^1_{21} + x_3 w^1_{31} + x_4 w^1_{41} + B^1$

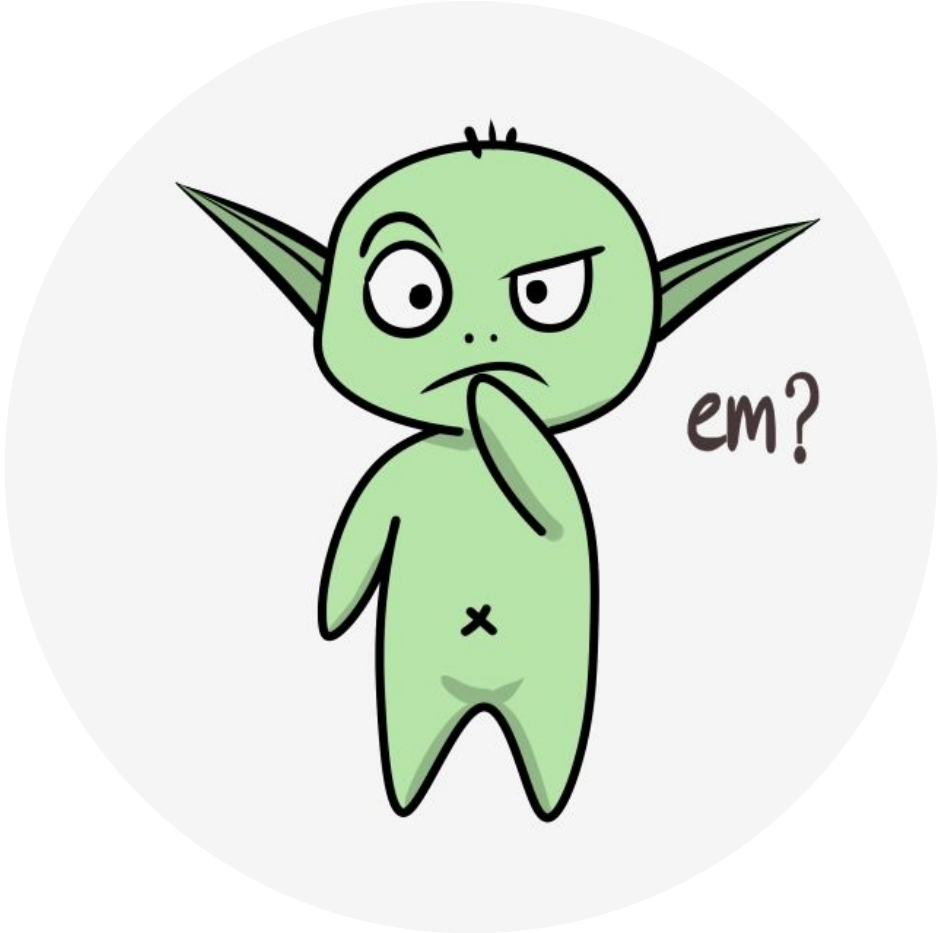
$\mathbf{h^2} = act(\mathbf{h^1 W^2} + \mathbf{b^2})$

s.t. $h^2_1 = h^1_1 w^2_{11} + h^1_2 w^2_{21} + h^1_3 w^2_{31} + h^1_4 w^2_{41} + B^2$

$\mathbf{o} = act(\mathbf{h^2 W^3} + \mathbf{b^3})$

s.t. $o_1 = h^2_1 w^3_{11} + h^2_2 w^3_{21} + h^2_3 w^3_{31} + h^2_4 w^3_{41} + B^2$

$\mathbf{o} = act(act(act(\mathbf{xW^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2})\mathbf{W^3} + \mathbf{b^3})$



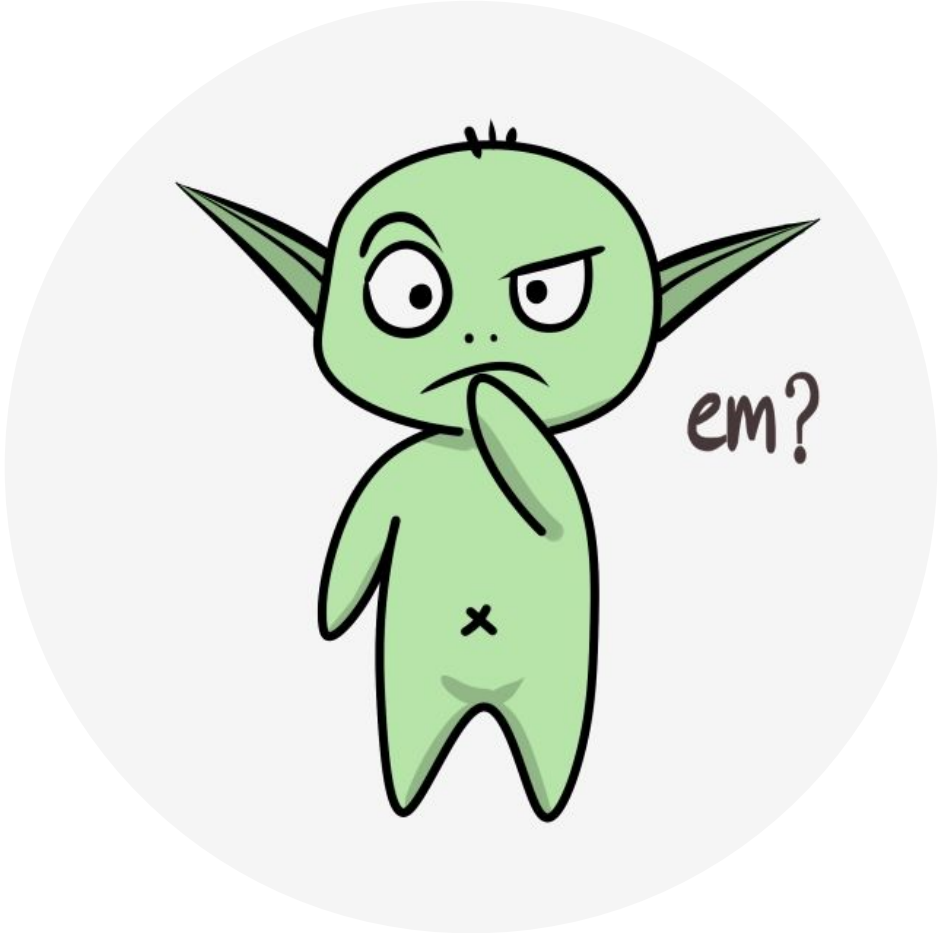
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Remember: 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



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

A Note from the Textbook

The backpropagation algorithm can produce good results for shallow networks having 1 or 2 hidden layers, but it may not work well for deeper ANNs. In fact, training a network with $k+1$ hidden layers can actually result in poorer performance than training a network with k hidden layers, even though the deeper network can represent all the functions that the shallower network can (Bengio, 2009). Explaining results like these is not easy, but several factors are important. First, the large number of weights in a typical deep ANN makes it difficult to avoid the problem of overfitting, that is, the problem of failing to generalize correctly to cases on which the network has not been trained. Second, backpropagation does not work well for deep ANNs because the partial derivatives computed by its backward passes either decay rapidly toward the input side of the network, making learning by deep layers extremely slow, or the partial derivatives grow rapidly toward the input side of the network, making learning unstable. Methods

Things change quickly...

Deep Convolutional Network

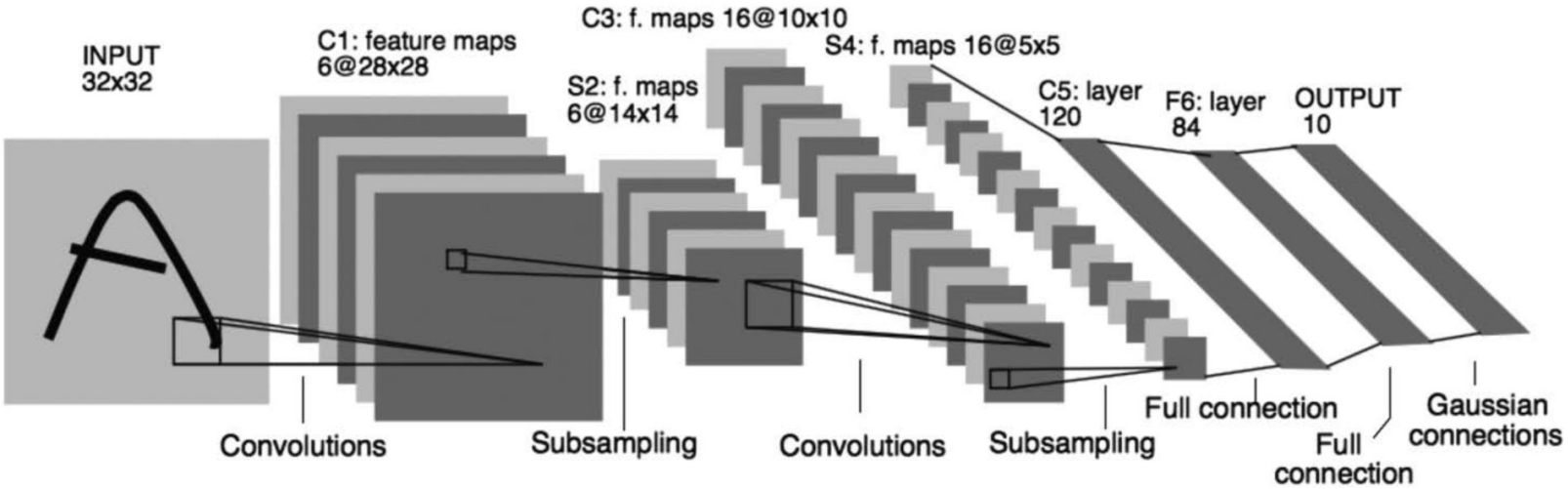
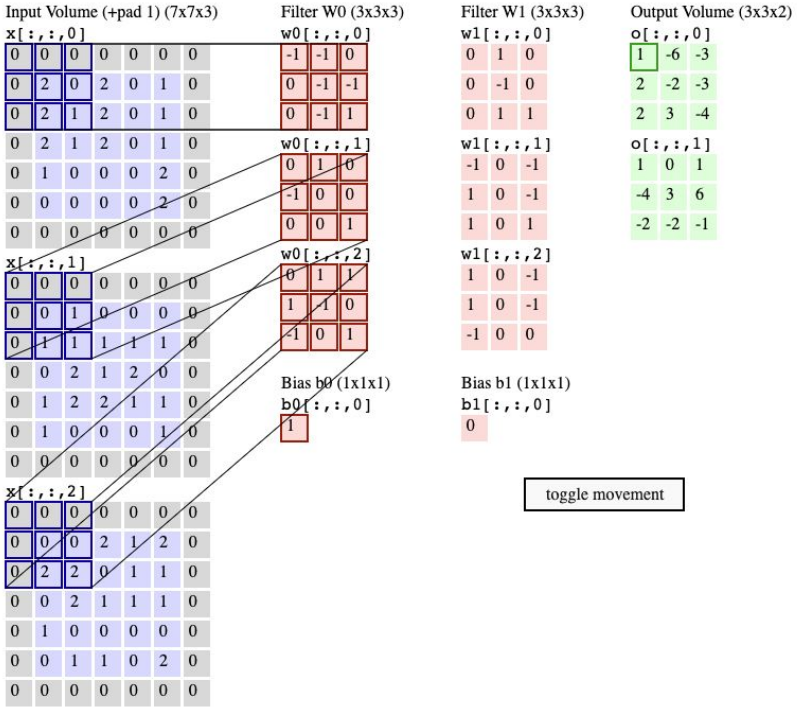


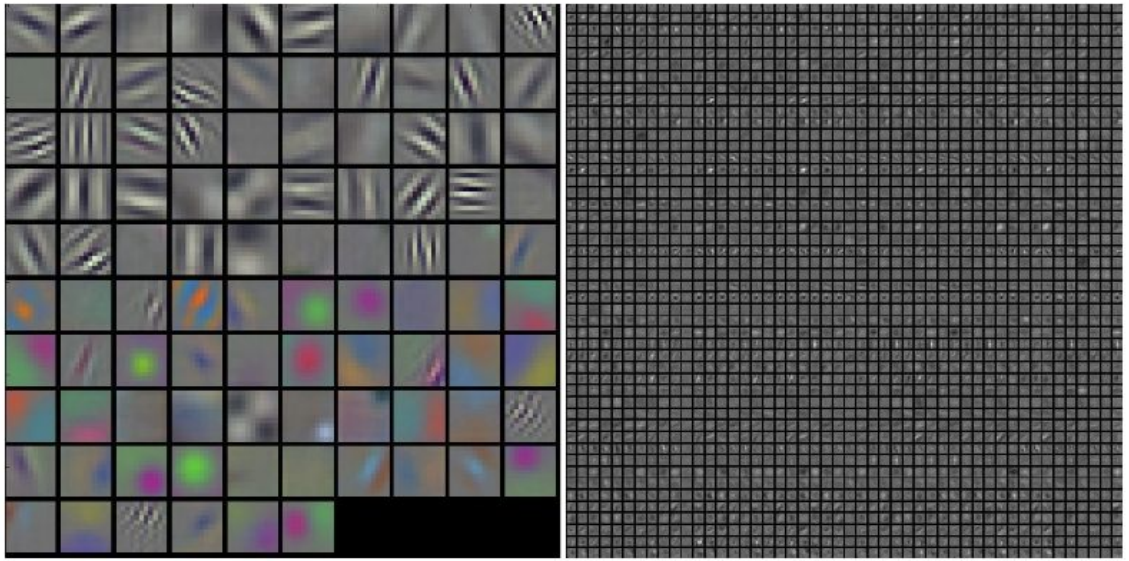
Figure 9.15: Deep Convolutional Network. Republished with permission of Proceedings of the IEEE, from Gradient-based learning applied to document recognition, LeCun, Bottou, Bengio, and Haffner, volume 86, 1998; permission conveyed through Copyright Clearance Center, Inc.

Deep Convolutional Network



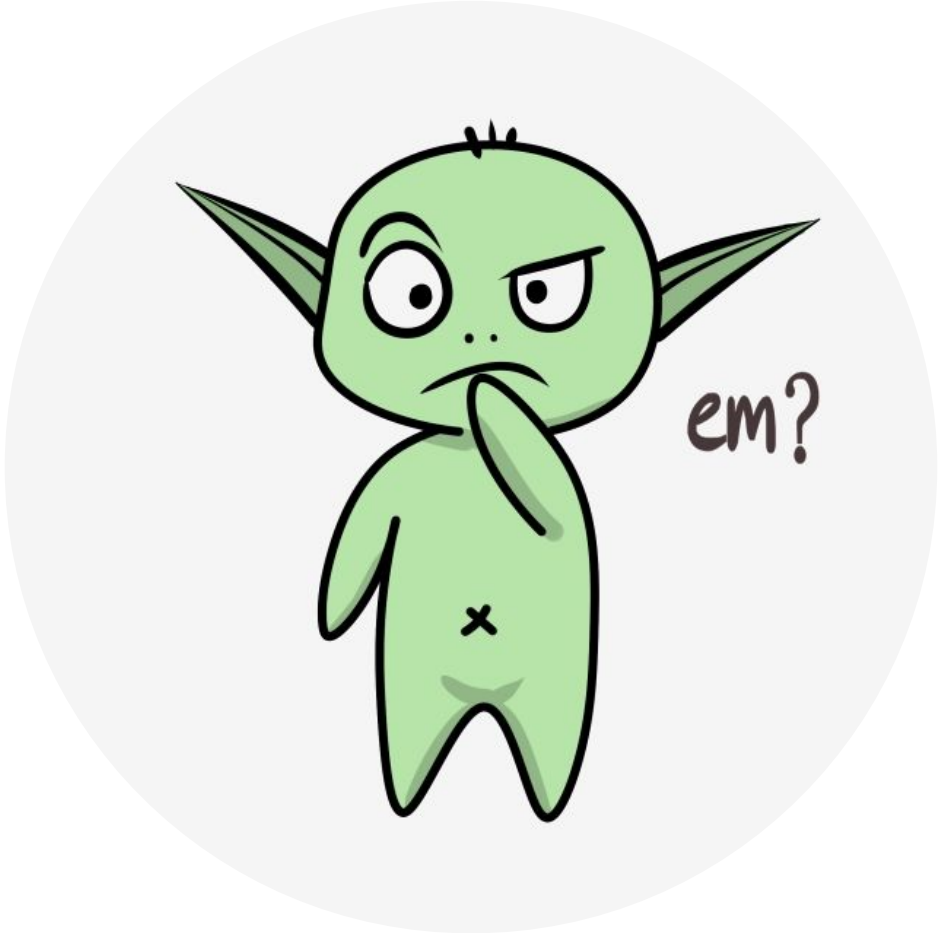
[Figure from demo in https://cs231n.github.io/convolutional-networks/]

Learned Representations



Typical-looking filters on the first CONV layer (left), and the 2nd CONV layer (right) of a trained AlexNet. Notice that the first-layer weights are very nice and smooth, indicating nicely converged network. The color/grayscale features are clustered because the AlexNet contains two separate streams of processing, and an apparent consequence of this architecture is that one stream develops high-frequency grayscale features and the other low-frequency color features. The 2nd CONV layer weights are not as interpretable, but it is apparent that they are still smooth, well-formed, and absent of noisy patterns.

[Figure from https://cs231n.github.io/understanding-cnn/]



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Least-Squares TD (LSTD)

- With more computation per time step we can do better.
- Why not compute the TD fixed point exactly?

$$\mathbf{w}_{\text{TD}} = \mathbf{A}^{-1}\mathbf{b},$$

where

$$\mathbf{A} \doteq \mathbb{E}\left[\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top\right] \quad \text{and} \quad \mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t]\,.$$

- Why not use the data to estimate **A** and **b**?

$$\widehat{\mathbf{A}}_t \doteq \sum_{k=0}^{t-1} \mathbf{x}_k(\mathbf{x}_k - \gamma\mathbf{x}_{k+1})^\top + \varepsilon\mathbf{I} \quad \text{and} \quad \widehat{\mathbf{b}}_t \doteq \sum_{k=0}^{t-1} R_{k+1}\mathbf{x}_k$$

Ensures it is always invertible

Least-Squares TD (LSTD)

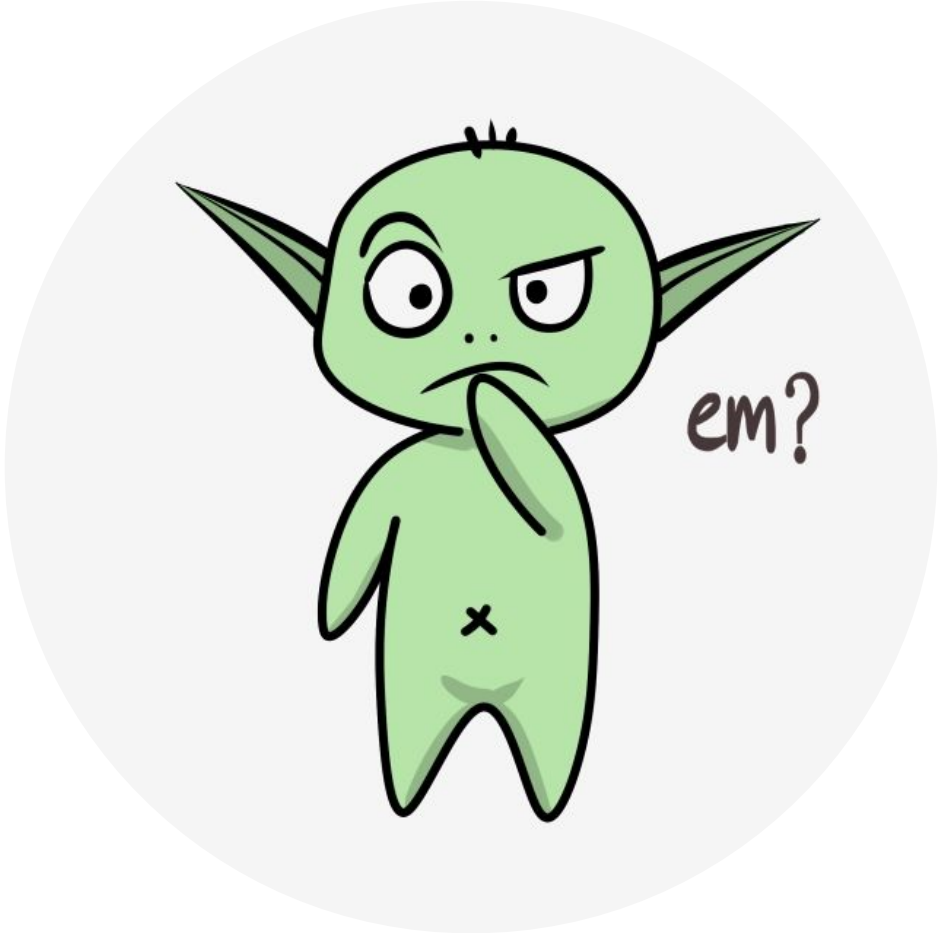
LSTD for estimating $\hat{v} = \mathbf{w}^\top \mathbf{x}(\cdot) \approx v_\pi$ ($O(d^2)$ version)

Input: feature representation $\mathbf{x} : \mathcal{S}^+ \to \mathbb{R}^d$ such that $\mathbf{x}(terminal) = \mathbf{0}$
Algorithm parameter: small $\varepsilon > 0$

$\widehat{\mathbf{A}^{-1}} \leftarrow \varepsilon^{-1}\mathbf{I}$ — A $d \times d$ matrix
$\widehat{\mathbf{b}} \leftarrow \mathbf{0}$ — A d-dimensional vector
Loop for each episode:
  Initialize S; $\mathbf{x} \leftarrow \mathbf{x}(S)$
  Loop for each step of episode:
    Choose and take action $A \sim \pi(\cdot|S)$, observe R, S'; $\mathbf{x}' \leftarrow \mathbf{x}(S')$
    $\mathbf{v} \leftarrow \widehat{\mathbf{A}^{-1}}^\top (\mathbf{x} - \gamma\mathbf{x}')$
    $\widehat{\mathbf{A}^{-1}} \leftarrow \widehat{\mathbf{A}^{-1}} - \left(\widehat{\mathbf{A}^{-1}}\mathbf{x}\right)\mathbf{v}^\top / \left(1 + \mathbf{v}^\top \mathbf{x}\right)$
    $\widehat{\mathbf{b}} \leftarrow \widehat{\mathbf{b}} + R\mathbf{x}$
    $\mathbf{w} \leftarrow \widehat{\mathbf{A}^{-1}}\widehat{\mathbf{b}}$
    $S \leftarrow S'$; $\mathbf{x} \leftarrow \mathbf{x}'$
  until S' is terminal

Sherman-Morrison formula

$$\widehat{\mathbf{A}}_t^{-1} = \left(\widehat{\mathbf{A}}_{t-1} + \mathbf{x}_{t-1}(\mathbf{x}_{t-1} - \gamma\mathbf{x}_t)^\top\right)^{-1}$$
$$= \widehat{\mathbf{A}}_{t-1}^{-1} - \frac{\widehat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_{t-1}(\mathbf{x}_{t-1} - \gamma\mathbf{x}_t)^\top \widehat{\mathbf{A}}_{t-1}^{-1}}{1 + (\mathbf{x}_{t-1} - \gamma\mathbf{x}_t)^\top \widehat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_{t-1}}$$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Memory-based Function Approximation

- Instead of updating some parameters and discarding the training example, we save (a subset of) training examples in memory as they arrive.
- When we want to query a state's value estimate, we retrieve examples from memory and use them to compute such an estimate. That's *lazy learning*.
- These are *nonparametric* methods.
- N*earest neighbor* is the simplest example, and *weighted average* a slightly more complicated one.
  - It finds the example in memory whose state is closest to the query state and returns that example's value as the approximate value of the query state.
- Naturally they inherit the benefits and trade-offs of nonparametric methods.

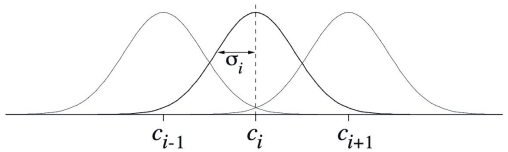
Kernel-based Function Approximation

- The function that assigns the weights in the weighted average is called a *kernel function*, or simply *kernel*, *k(s, s')*.
- *k(s, s')* is a measure of the strength of generalization from s' to s. How relevant is the knowledge about state s to state s'.
- Kernel regression, where *g(s')* denotes the target for state s'.

$$\hat{v}(s,\mathcal{D}) = \sum_{s' \in \mathcal{D}} k(s,s')g(s')$$

Example of a kernel function: Radial Basis Functions (RBFs)

$$x_i(s) \doteq \exp\left(-\frac{||s - c_i||^2}{2\sigma_i^2}\right)$$



Chapter 10

On-policy Control with Approximation

Control

Overview

- More of the same, but now

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

and

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

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., $\mathbf{w} = \mathbf{0}$)

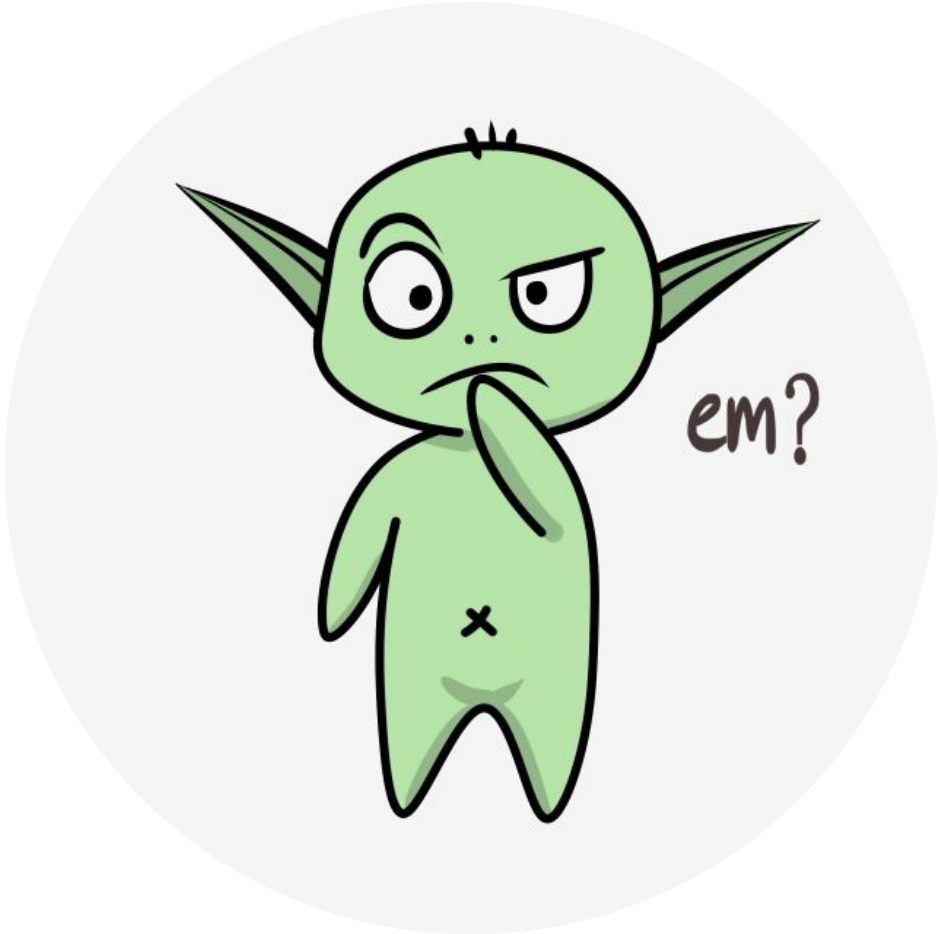
Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$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

Episodic Semi-gradient n-step Sarsa

Episodic semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_*$ or q_π

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$, a positive integer n
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$ or ε-greedy wrt $\hat{q}(S_0, \cdot, \mathbf{w})$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
    If $t < T$, then:
      Take action A_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:
        $T \leftarrow t + 1$
      else:
        Select and store $A_{t+1} \sim \pi(\cdot|S_{t+1})$ or ε-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
    $\tau \leftarrow t - n + 1$ (τ is the time whose 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 $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$ $(G_{\tau:\tau+n})$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{q}(S_\tau, A_\tau, \mathbf{w}) \right] \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})$
  Until $\tau = T - 1$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Average Reward: A New Problem Setting for Continuing Tasks

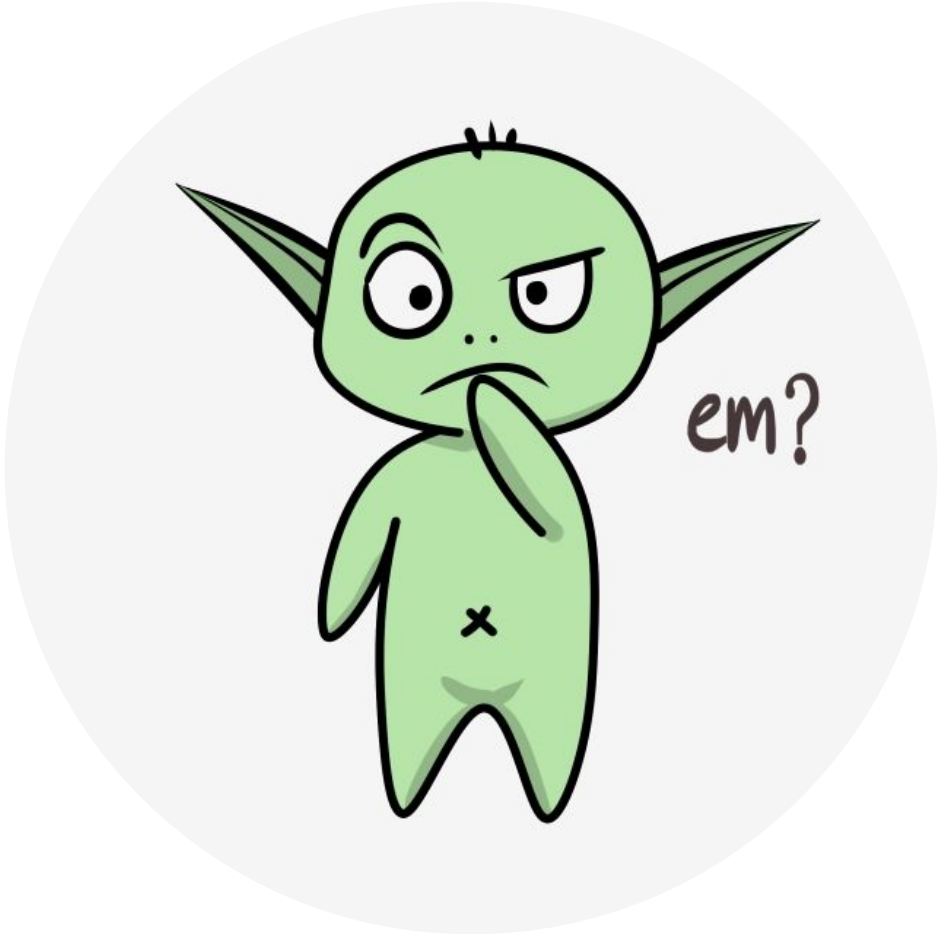
- It is applicable to *continuing* problems.
  - Different from the discounted setting, every time step is equally important.
- In this problem formulation, the quality of a policy π is defined by the *avg. reward*:

$$
\begin{aligned}
r(\pi) &\doteq \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi] \\
&= \lim_{t\to\infty} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi], \\
&= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r
\end{aligned}
$$

The MDP needs to be *ergodic:* in the long run the expectation of being in a state depends only on the policy and the MDP transition probabilities

The steady state distribution μ_π is the special distribution under which, if you select actions according to π, you remain in the same distribution.

$$\sum_s \mu_\pi(s) \sum_a \pi(a|s) p(s'|s, a) = \mu_\pi(s').$$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

What's a Good Policy in the Average Reward Formulation

- We can order policies by their average reward per time step.
- Returns are defined in terms of difference between rewards and the avg. reward:

Differential return

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots.$$

What's a Good Policy in the Average Reward Formulation

- We can order policies by their average reward per time step.
- Returns are defined in terms of difference between rewards and the avg. reward:

Differential return

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots.$$

- Correspondingly, we have differential value functions:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s', r \,|\, s, a) \Big[r - r(\pi) + v_\pi(s') \Big],$$

$$q_\pi(s,a) = \sum_{r,s'} p(s', r \,|\, s, a) \Big[r - r(\pi) + \sum_{a'} \pi(a'|s') q_\pi(s', a') \Big],$$

$$v_*(s) = \max_a \sum_{r,s'} p(s', r \,|\, s, a) \Big[r - \max_\pi r(\pi) + v_*(s') \Big], \text{ and}$$

$$q_*(s,a) = \sum_{r,s'} p(s', r \,|\, s, a) \Big[r - \max_\pi r(\pi) + \max_{a'} q_*(s', a') \Big]$$

There's also the Differential form of TD errors

$$\delta_t \doteq R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t),$$

and

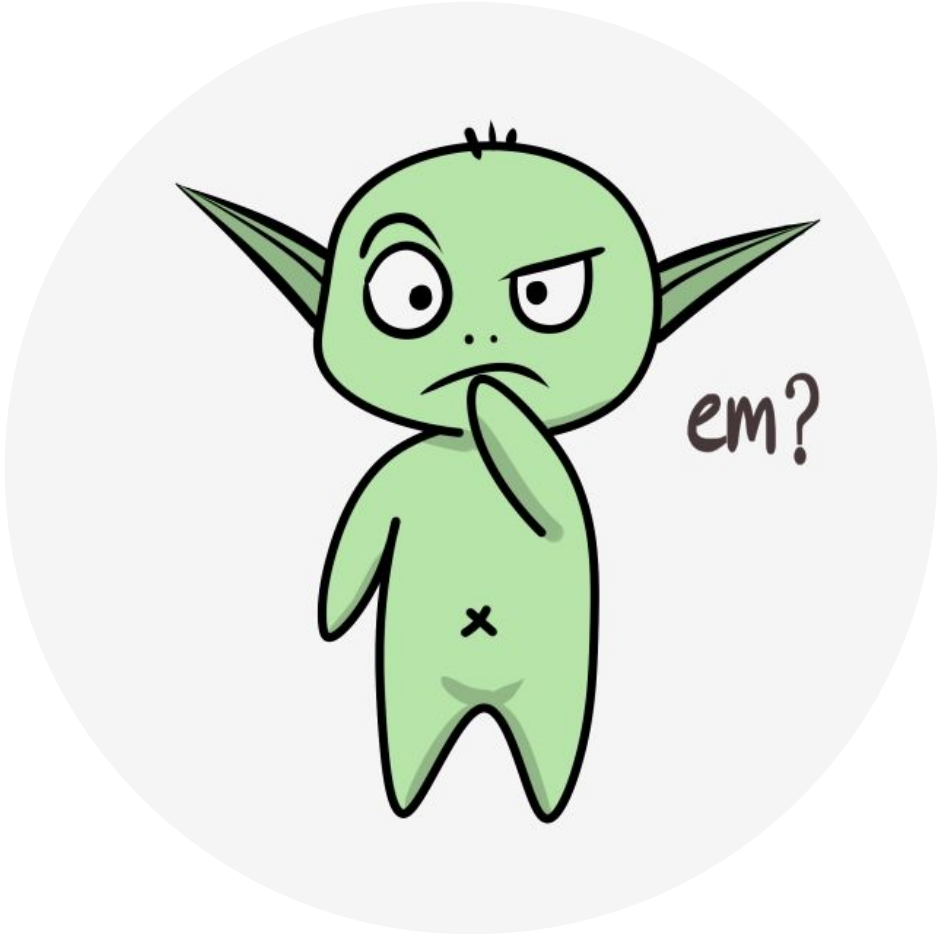
$$\delta_t \doteq R_{t+1} - \bar{R}_t + \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Things Stay Pretty Much the Same Then

Differential 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 sizes $\alpha, \beta > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)

Initialize state S, and action A
Loop for each step:
  Take action A, observe R, S'
  Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
  $\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
  $\bar{R} \leftarrow \bar{R} + \beta\delta$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S, A, \mathbf{w})$
  $S \leftarrow S'$
  $A \leftarrow A'$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

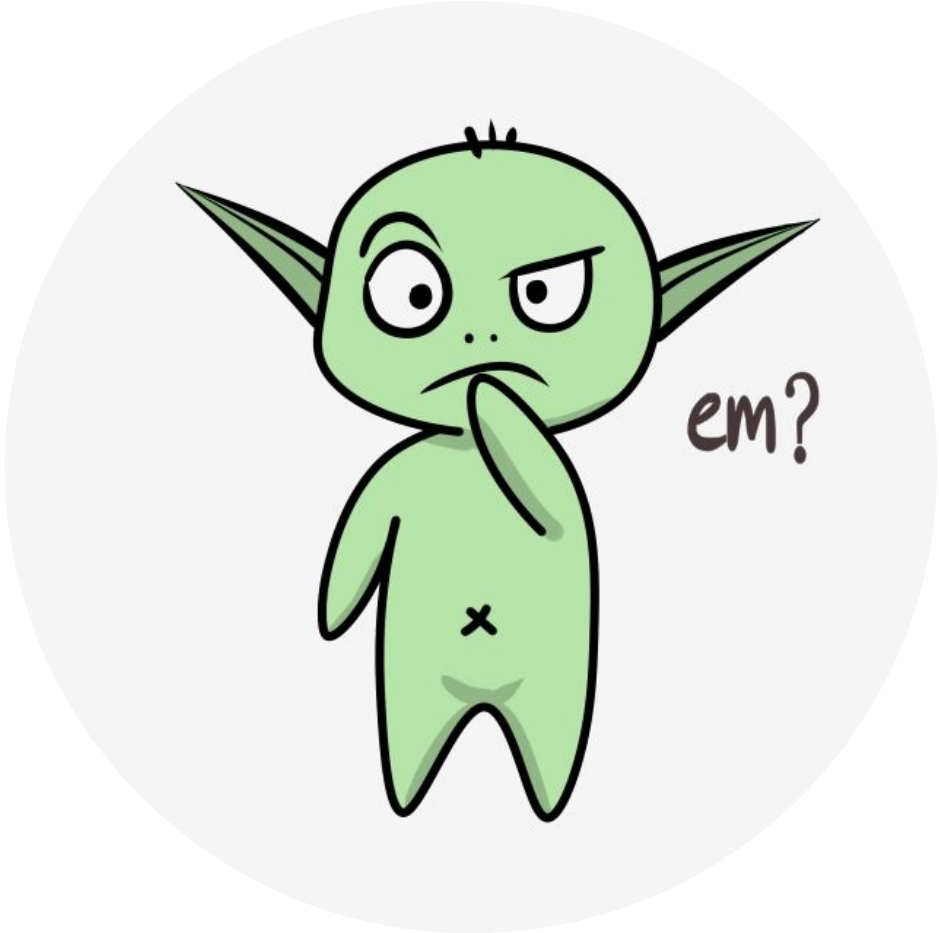
The "Futility" of Discounting in Continuing Problems

Perhaps discounting can be saved by choosing an objective that sums discounted values over the distribution with which states occur under the policy:

$$
\begin{aligned}
J(\pi) &= \sum_s \mu_\pi(s) v_\pi^\gamma(s) && \text{(where } v_\pi^\gamma \text{ is the discounted value function)} \\
&= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \,|\, s, a) \left[r + \gamma v_\pi^\gamma(s') \right] && \text{(Bellman Eq.)} \\
&= r(\pi) + \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \,|\, s, a) \gamma v_\pi^\gamma(s') && \text{(from (10.7))} \\
&= r(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s') \sum_s \mu_\pi(s) \sum_a \pi(a|s) p(s' \,|\, s, a) && \text{(from (3.4))} \\
&= r(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s') \mu_\pi(s') && \text{(from (10.8))} \\
&= r(\pi) + \gamma J(\pi) \\
&= r(\pi) + \gamma r(\pi) + \gamma^2 J(\pi) \\
&= r(\pi) + \gamma r(\pi) + \gamma^2 r(\pi) + \gamma^3 r(\pi) + \cdots \\
&= \frac{1}{1-\gamma} r(\pi).
\end{aligned}
$$

The proposed discounted objective orders policies identically to the undiscounted (average reward) objective. The discount rate γ does not influence the ordering!

The discounting factor still is a very useful hyperparameter for the algorithm itself.



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Differential n-step return and n-step TD error

$$G_{t:t+n} \doteq R_{t+1} - \bar{R}_{t+n-1} + \cdots + R_{t+n} - \bar{R}_{t+n-1} + \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1})$$

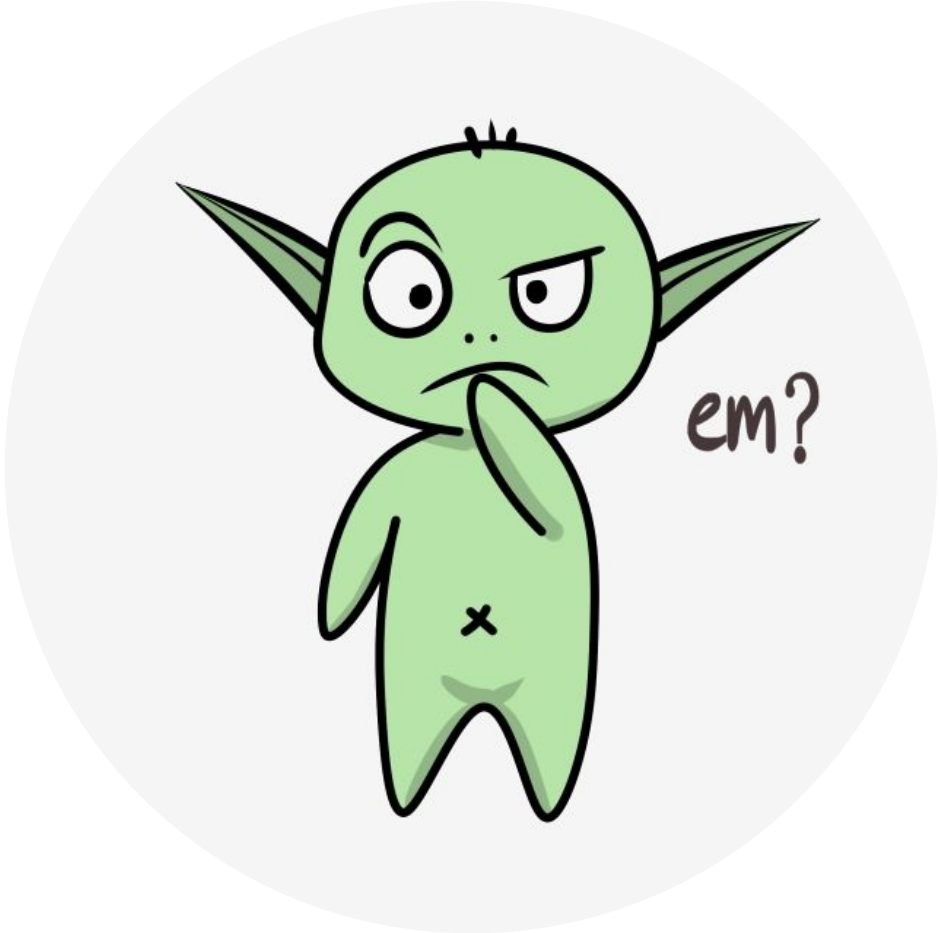
$$\delta_t \doteq G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w})$$

Things Stay Pretty Much the Same Then

Differential semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_\pi$ or q_*

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$, a policy π
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average-reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)
Algorithm parameters: step sizes $\alpha, \beta > 0$, small $\varepsilon > 0$, a positive integer n
All store and access operations (S_t, A_t, and R_t) can take their index mod $n+1$

Initialize and store S_0 and A_0
Loop for each step, $t = 0, 1, 2, \ldots$:
  Take action A_t
  Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$, or ε-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
  $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  If $\tau \geq 0$:
    $\delta \leftarrow \sum_{i=\tau+1}^{\tau+n} (R_i - \bar{R}) + \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w}) - \hat{q}(S_\tau, A_\tau, \mathbf{w})$
    $\bar{R} \leftarrow \bar{R} + \beta\delta$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S_\tau, A_\tau, \mathbf{w})$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht