“For even the very wise cannot see all ends.”

J.R.R. Tolkien, The Fellowship of the Ring

Marlos C. Machado

Class 7/12

Plan

- Finish Chapter 8: Planning and Learning with Tabular Methods.
- Chapter 9: On-policy Prediction with Approximation.
- Chapter 10: On-policy Control with Approximation.
    - I will not talk about the Average Reward formulation today.

Reminder I

You **should be enrolled in the private session** we created in Coursera for CMPUT 655.

I **cannot** use marks from the public repository for your course marks.

You **need** to **check**, **every time**, if you are in the private session and if you are submitting quizzes and assignments to the private section.

The deadlines in the public session **do not align** with the deadlines in Coursera.

If you have any questions or concerns, **talk with the TAs** or email us `cmput655@ualberta.ca`.

Reminder II

- On the project
  - The project proposal is due Wednesday 😉.
  - The course project is very different from what we generally ask undergraduate students to do. It is to be done over an extended period of time.
  - Recommended readings:
    - *The Elements of Style* by W. Strunk Jr.
    - *Empirical Design in Reinforcement Learning* by A. Patterson, S. Neumann, M. White, and A. White.
- There is no scheduled Coursera activity for you to do next week

Reminder III

- **I want your feedback!**
  - Mid-term Course and Instruction Feedback online evaluation opened today.
  - It will close today.
  - 21 of 54 students responded so far 😭

Please, interrupt me at any time!

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

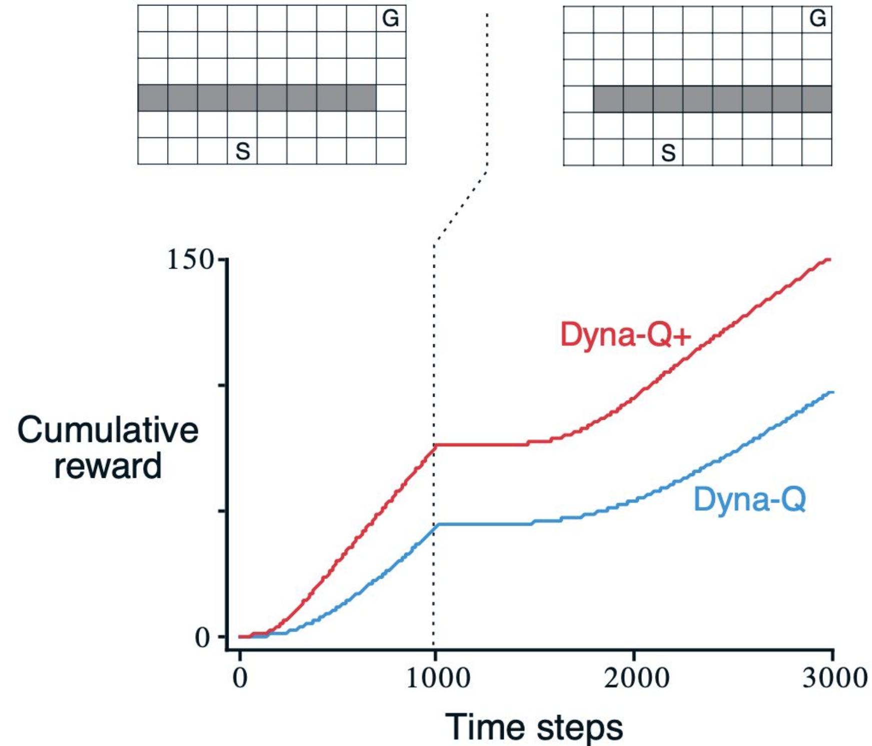
Chapter 8

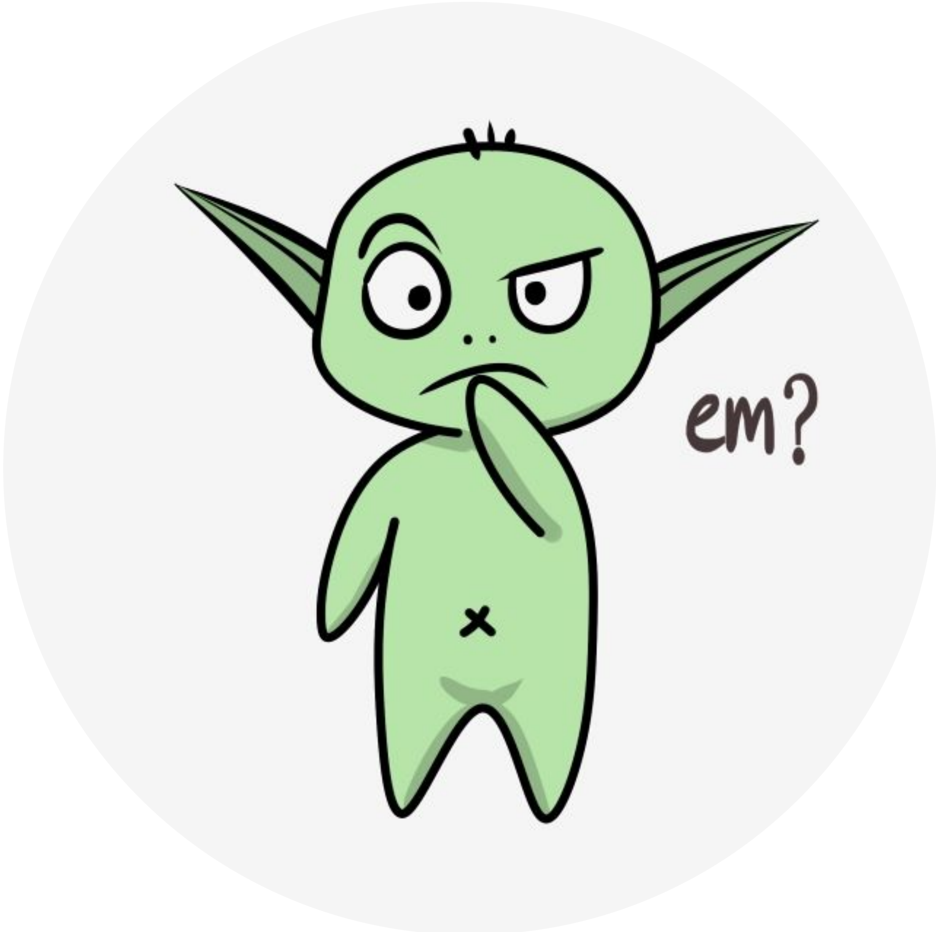
Planning and Learning with Tabular Methods

Last Class: When the Model Is Wrong

- A model can be wrong for all sorts of reasons (e.g., stochastic environment, function approximation, non-stationarity in the environment).
- An incorrect model often leads to suboptimal policies.
- One needs to constantly explore to refine the learned model.
  - Exploration: take actions that improve the model.
  - Exploitation: behaving in the optimal way given the current model.
- Dyna-Q+: Provides "bonus rewards" for long-untried actions.
  Specifically, consider the reward $r + \kappa\sqrt{\tau}$, where τ is the number of time steps since that transition was tried for the last time.

Last Class: Dyna-Q+ Sometimes Works ¯_(ツ)_/¯





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

Prioritized Sweeping

- How should we select transitions to simulate for the planning update?
- Working backward from goal states seem like a good idea, but it is dependent on the idea of a “goal state”.
  - More generally, we want to work back from any state whose value has changed.
- *Prioritized sweeping* is the idea of prioritizing updates according to their urgency.
  - When the effect of a change is greater than a threshold, the state in which that change happened is added to a priority queue.

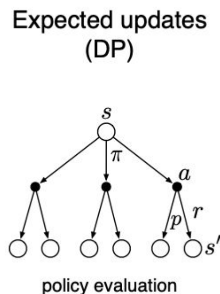
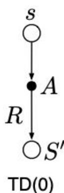
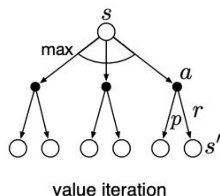
Prioritized Sweeping

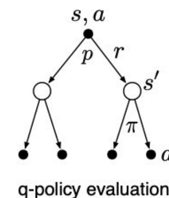
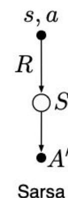
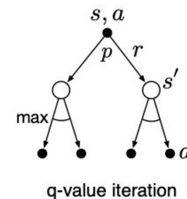
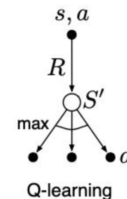
Prioritized sweeping for a deterministic environment

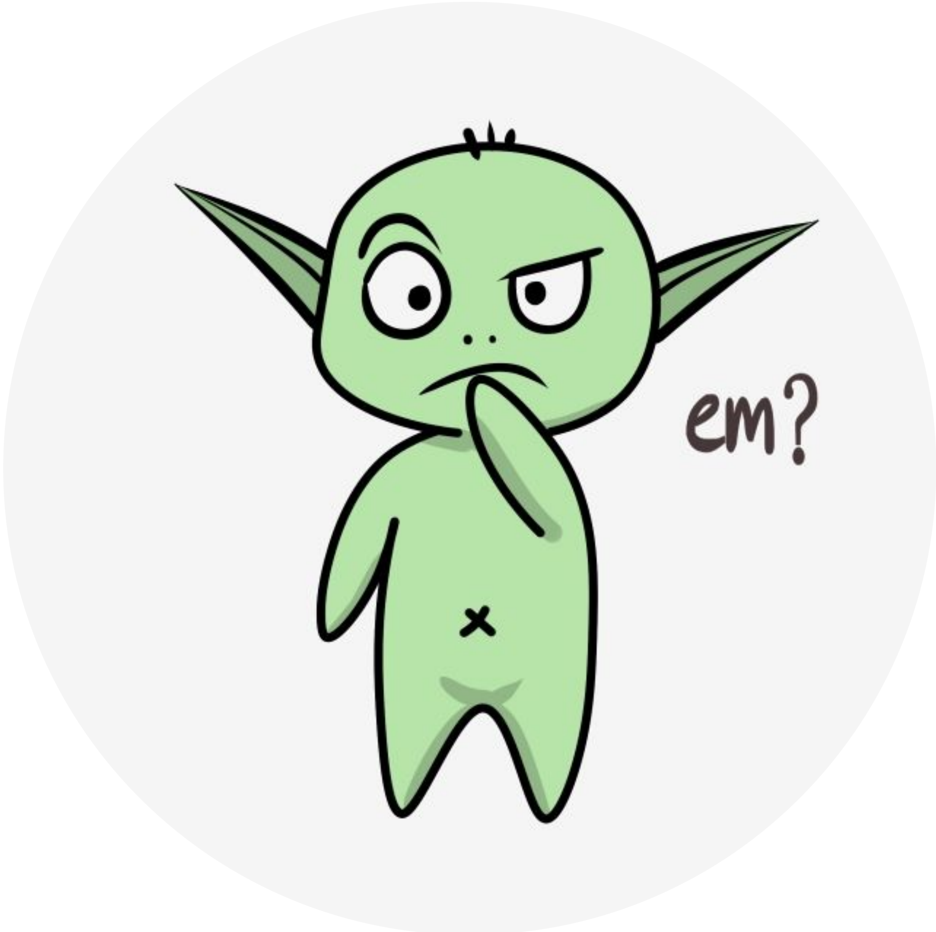
Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Loop forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow policy(S, Q)$
  (c) Take action A; observe resultant reward, R, and state, S'
  (d) $Model(S,A) \leftarrow R, S'$
  (e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
  (f) if $P > \theta$, then insert S,A into $PQueue$ with priority P
  (g) Loop repeat n times, while $PQueue$ is not empty:
    $S, A \leftarrow first(PQueue)$
    $R, S' \leftarrow Model(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    Loop for all $\bar{S}, \bar{A}$ predicted to lead to S:
      $\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
      $P \leftarrow |\bar{R} + \gamma \max_a Q(S,a) - Q(\bar{S},\bar{A})|$.
      if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P

Expected vs. Sample Updates

- There are three dimensions in the updates one can do:
    - Should we use state values or action values?
    - Should we estimate the value for the optimal policy or for an arbitrary given policy?
    - Should we use expected or sample updates?

Value estimated	Expected updates (DP)	Sample updates (one-step TD)
$v_\pi(s)$		
$v_*(s)$		

Value estimated	Expected updates (DP)	Sample updates (one-step TD)
$q_\pi(s, a)$		
$q_*(s, a)$		



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

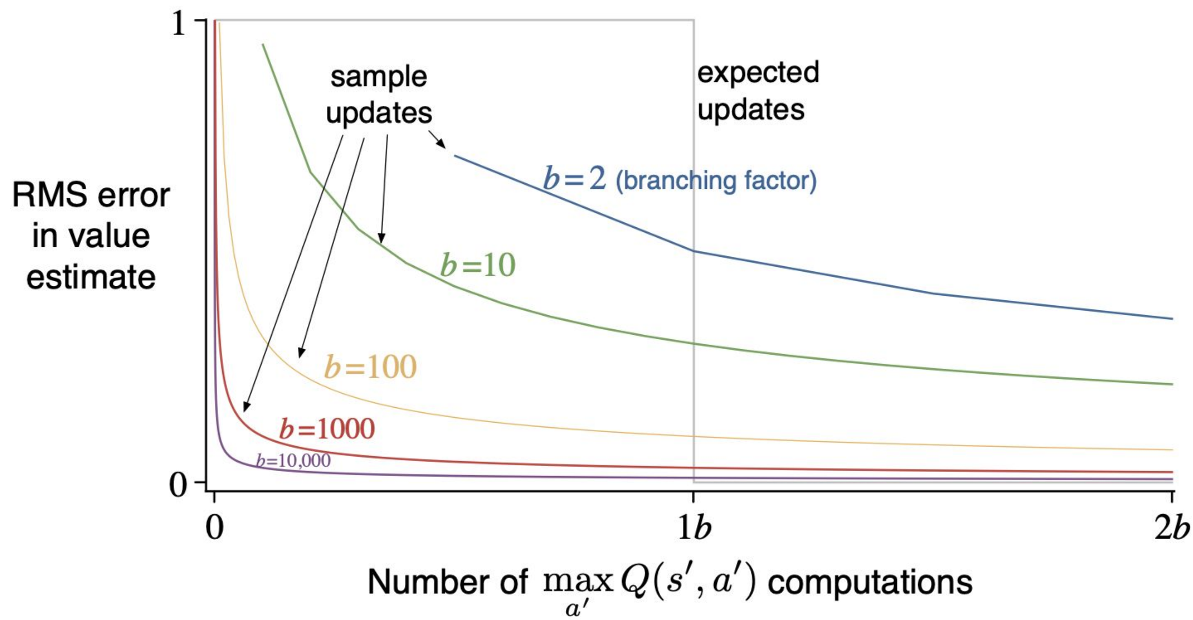
Expected vs. Sample Updates

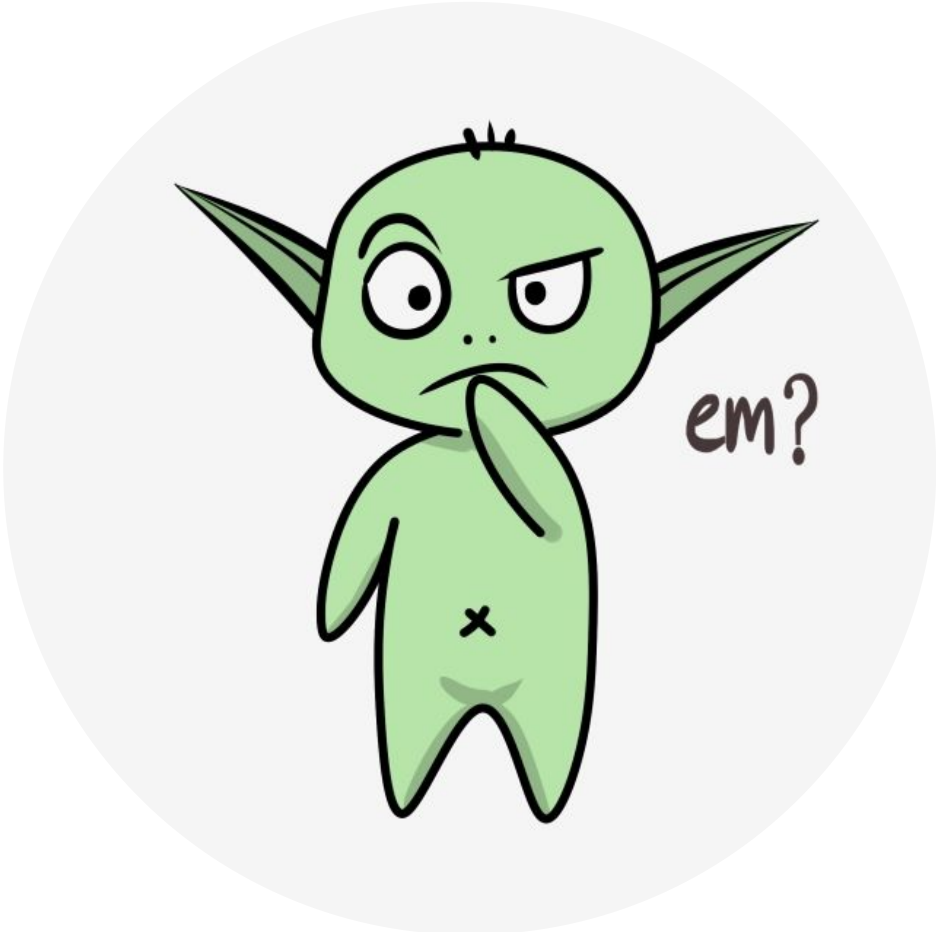
- If possible, are expected updates always preferable?
  - They yield a better estimate because they are uncorrupted by sampling error, but they also require more computation, and computation is often the limiting resource in planning.

$$Q(s,a) \leftarrow \sum_{s',r} \hat{p}(s',r|s,a)\Big[r + \gamma \max_{a'} Q(s',a')\Big] \qquad Q(s,a) \leftarrow Q(s,a) + \alpha\Big[R + \gamma \max_{a'} Q(S',a') - Q(s,a)\Big]$$

- Do we have enough time to do an expected update?
- Is it better to have a few sample updates at many state–action pairs or to have expected updates at a few pairs?

Often, the error falls dramatically with a fraction of b updates



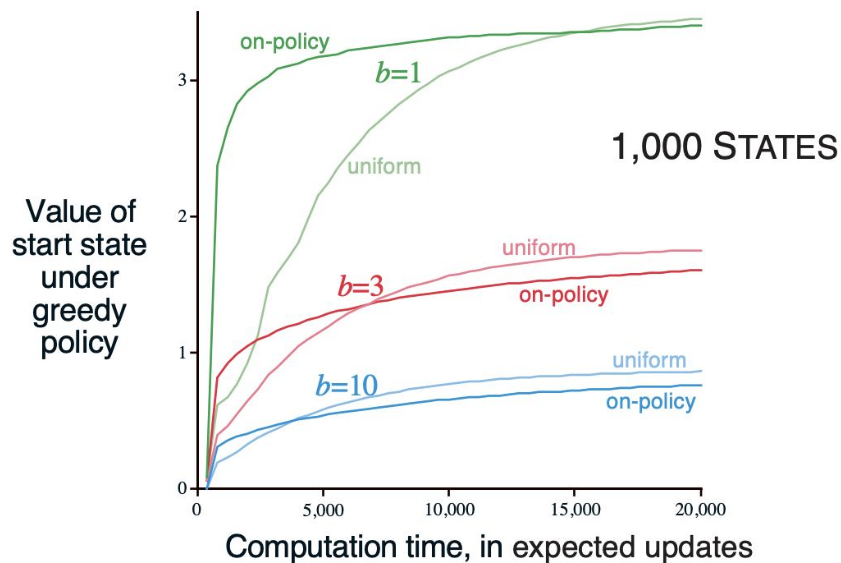


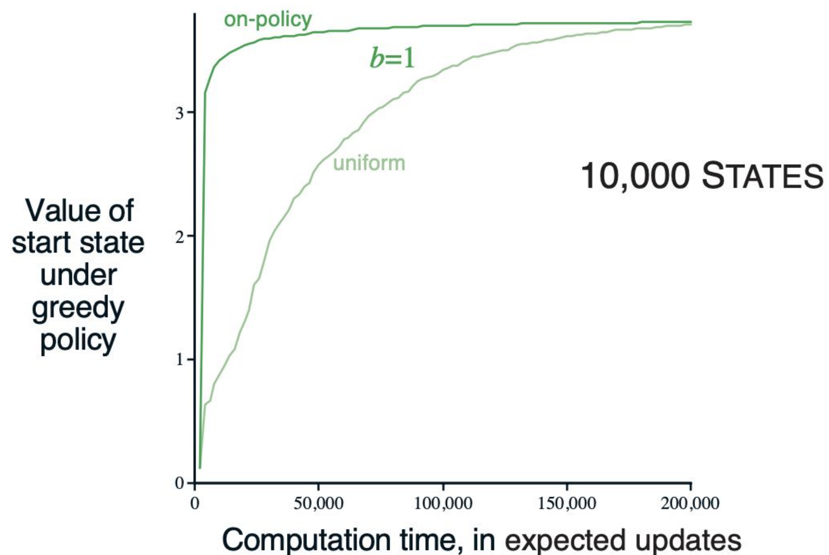
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Trajectory Sampling

- The classical approach, from dynamic programming, is to perform sweeps through the entire state space, updating each state once per sweep.
- However, in many tasks, most states are irrelevant under good policies.
- What if we sampled states from the state or state–action space according to some distribution?
- *Trajectory sampling* is the idea of sampling states from the on-policy distribution.

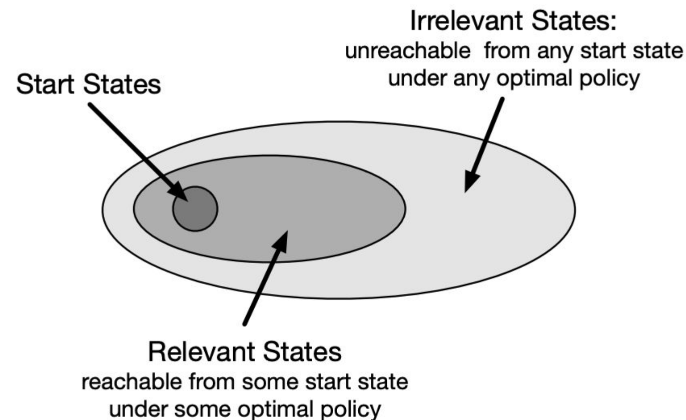
Inconclusive Results

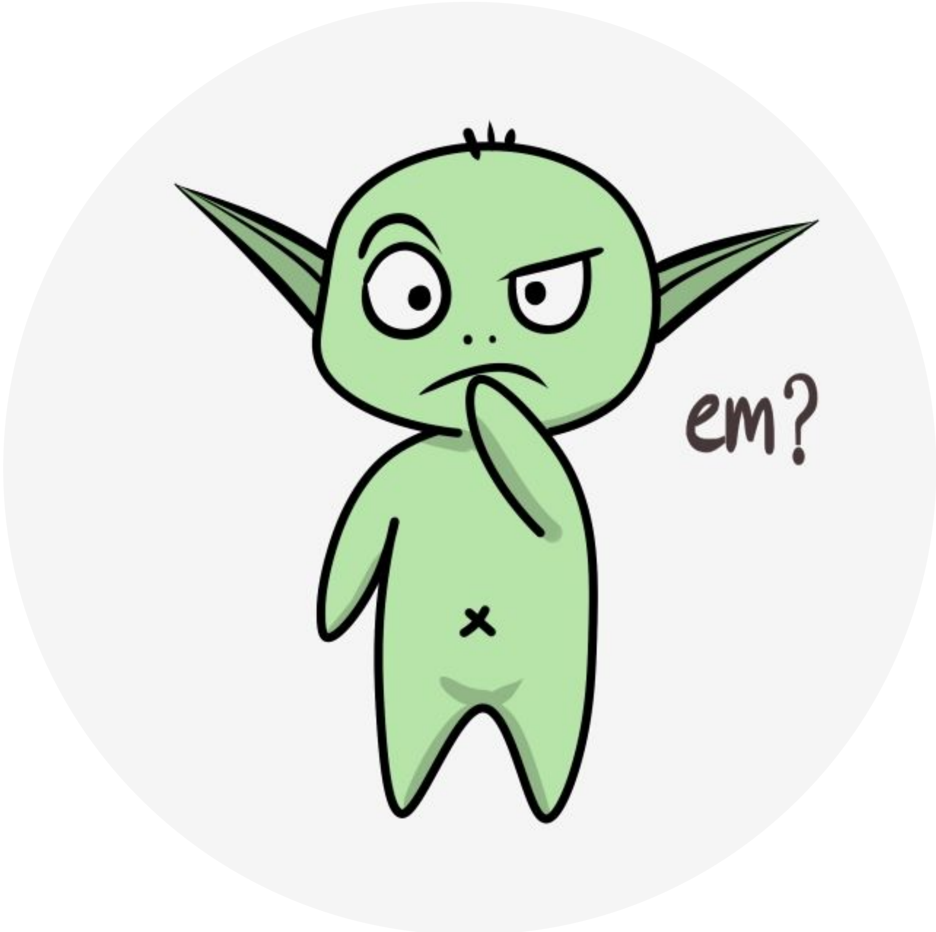




Real-time Dynamic Programming

- Real-time dynamic programming, or RTDP, is an on-policy trajectory-sampling version of the value-iteration algorithm of dynamic programming (DP).
- RTDP updates the values of states visited in actual or simulated trajectories by means of expected tabular value-iteration updates.
- RTDP is an example of an asynchronous DP algorithm. In RTDP, the update order is dictated by the order states are visited in real or simulated trajectories.
- It has some interesting convergence results in stochastic optimal path problems.





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

Decision-time Planning

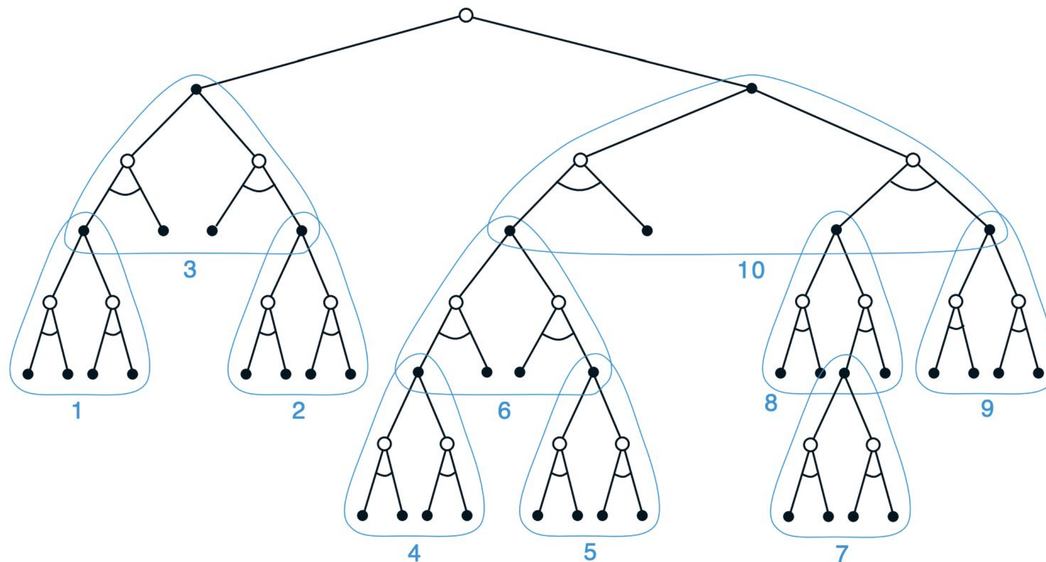
- We've been discussing *background planning:* using planning to gradually improve a policy or value function based on simulated experience obtained from a model.
    - Well before an action is selected for any current state S_t, planning has played a part in improving the table entries needed to select actions for many states, including S_t.
- *Decision-time planning* uses planning to begin and complete it after encountering each new state S_t, as a computation whose output is the selection of an action A_t; on the next step planning begins anew with S_{t+1} to produce A_{t+1}, and so on.
- We can still see decision-time planning as proceeding from simulated experience to updates and values, and ultimately to a policy.
    - Now the values and policy are specific to the current state and the action choices available there.
- The response time really matters in this choice.

Heuristic Search

- The classical state-space planning methods in artificial intelligence are decision-time planning method collectively known as *heuristic search*.
- If one has a perfect model and an imperfect action-value function, then in fact deeper search will usually yield better policies.
- Much of its effectiveness is due to its search tree being focused on the states and actions that might immediately follow the current state.

Heuristic Search

Heuristic search can be implemented as a sequence of one-step updates (shown here outlined in blue) backing up values from the leaf nodes toward the root. The ordering shown below is for a selective depth-first search.



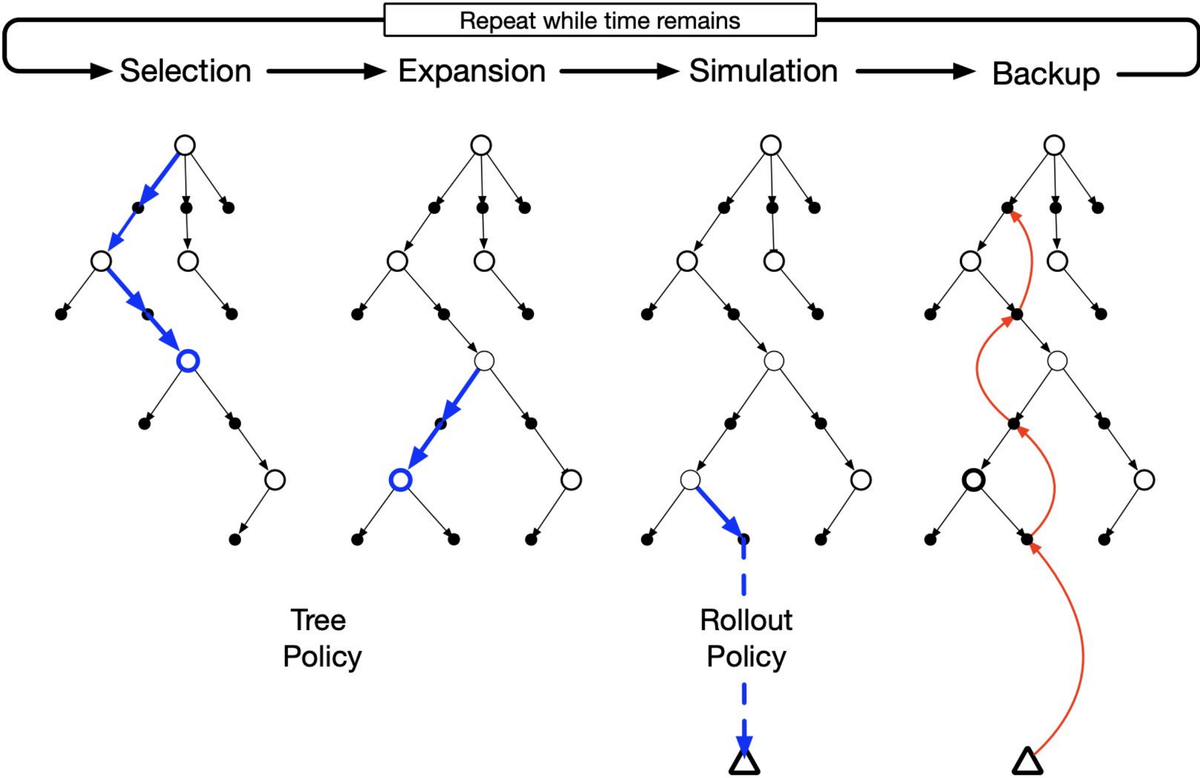
Rollout Algorithms

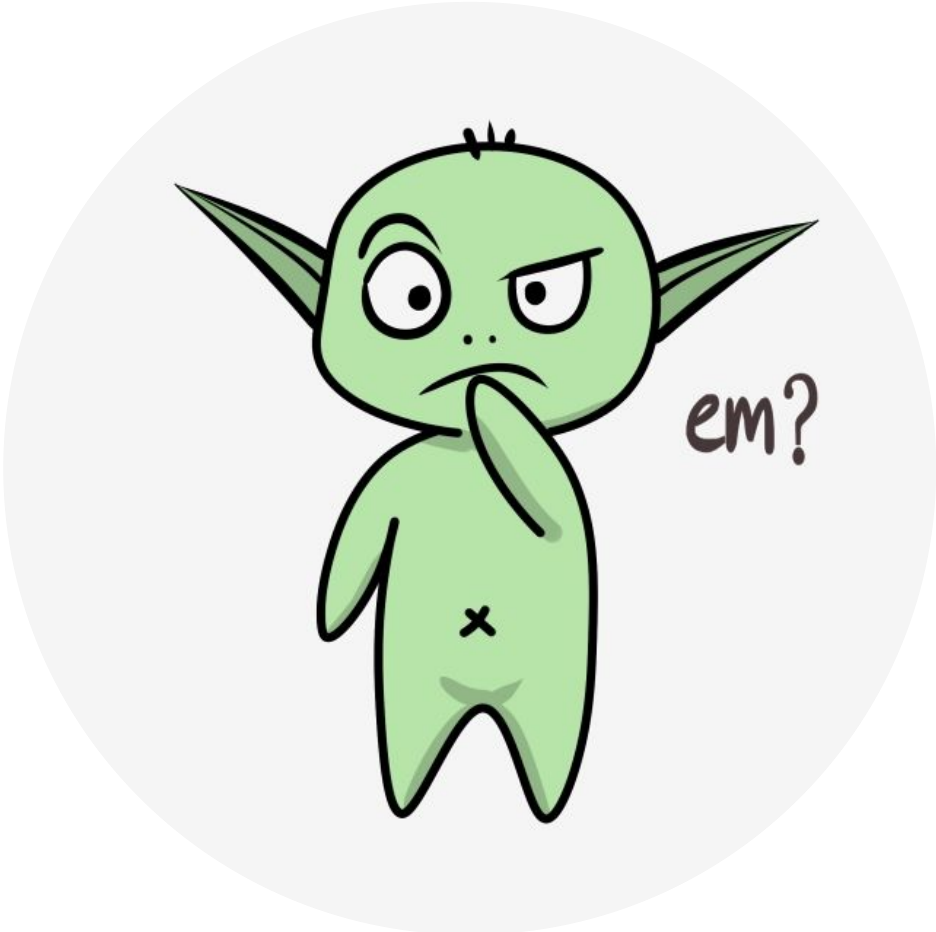
- Rollout algorithms are decision-time planning algorithms based on MC control applied to simulated trajectories that all begin at the current environment state.
  - They estimate action values for a given policy by averaging the returns of many simulated trajectories that start with each possible action and then follow the given policy.
- Unlike the Monte Carlo control algorithms previously described, the goal of a rollout algorithm is not to estimate a complete optimal action-value function, q_*, or a complete action-value function, q_π, for a given policy π.
  - They produce Monte Carlo estimates of action values only for each current state and for a given policy usually called the rollout policy.
- They are not learning algorithms *per se*, but they do leverage the RL toolkit.

Monte Carlo Tree Search (MCTS)

- MCTS is a great example of a rollout, decision-time planning algorithm.
  - But enhanced by the addition of a means for accumulating value estimates obtained from the MC simulations in order to successively direct simulations toward more highly-rewarding trajectories.
- The core idea of MCTS is to successively focus multiple simulations starting at the current state by extending the initial portions of trajectories that have received high evaluations from earlier simulations.
  - Monte Carlo value estimates are maintained only for the subset of state–action pairs that are most likely to be reached in a few steps, which form a tree rooted at the current state.

Monte Carlo Tree Search (MCTS)





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

Upper Confidence Bound 1 Applied to Trees (UCT)

Bandit based Monte-Carlo Planning

Levente Kocsis and Csaba Szepesvári

Computer and Automation Research Institute of the
Hungarian Academy of Sciences, Kende u. 13-17, 1111 Budapest, Hungary
kocsis@sztaki.hu

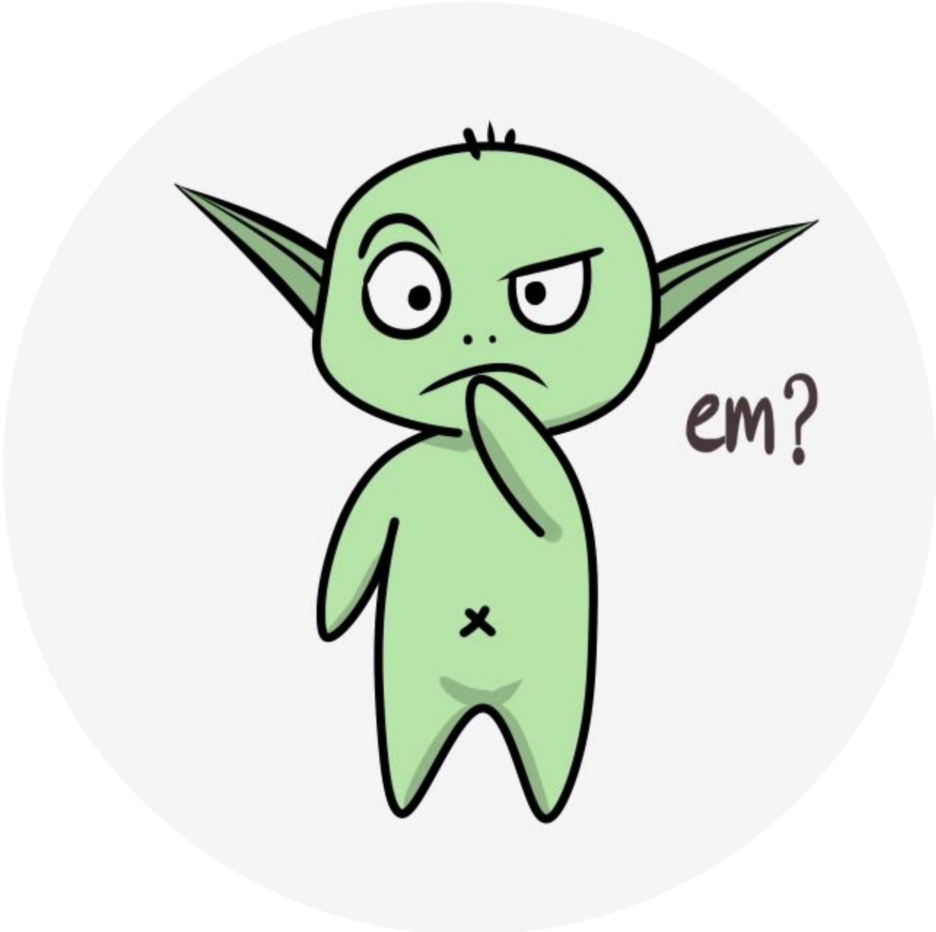
Monte Carlo Tree Search (MCTS)

Choose in each node of the game tree the move as the argmax of

$$\frac{w_i}{n_i} + \kappa\sqrt{\frac{\ln N_i}{n_i}}$$

- w_i: number of wins for the node considered after the i-th move.
- n_i: number of times the child node has been visited after the i-th move.
- N_i: number of times the parent node has been visited after the i-th move.
- κ: scalar parameter for trading-off exploration and exploitation.

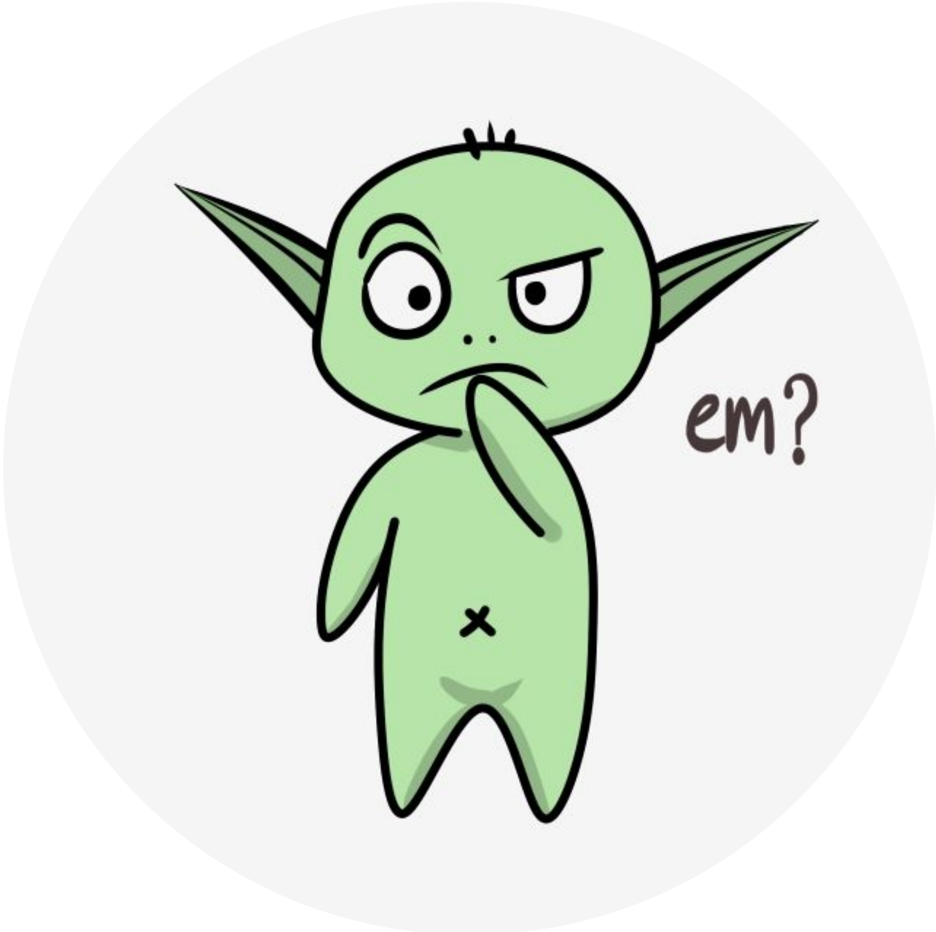
[Kocsis and Szepesvari, 2006]



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

MCTS incorporates several RL principles

- MCTS is a decision-time planning algorithm based on MC control applied to simulations that start from the root state (it is a kind of rollout algorithm).
    - It benefits from online, incremental, sample-based value estimation and policy improvement.
- It saves action-value estimates attached to the tree edges and updates them using reinforcement learning's sample updates.
    - It focuses the Monte Carlo trials on trajectories whose initial segments are common to high-return trajectories previously simulated.
- By incrementally expanding the tree, MCTS effectively grows a lookup table to store a partial action-value function, with memory allocated to the estimated values of state–action pairs visited in the initial segments of high-yielding sample trajectories
    - MCTS avoids the problem of globally approximating an action-value function while it retains the benefit of using past experience to guide exploration.



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

Wrapping Up

- We have finished Part I of the textbook, Tabular Solution Methods.
- Reinforcement learning can be seen as being more than a collection of individual methods, but a coherent set of ideas cutting across methods.
    - They all seek to estimate value functions.
    - They all operate by backing up values along actual or possible state trajectories.
    - They all follow the general strategy of generalized policy iteration (GPI).

Wrapping Up

width
of update

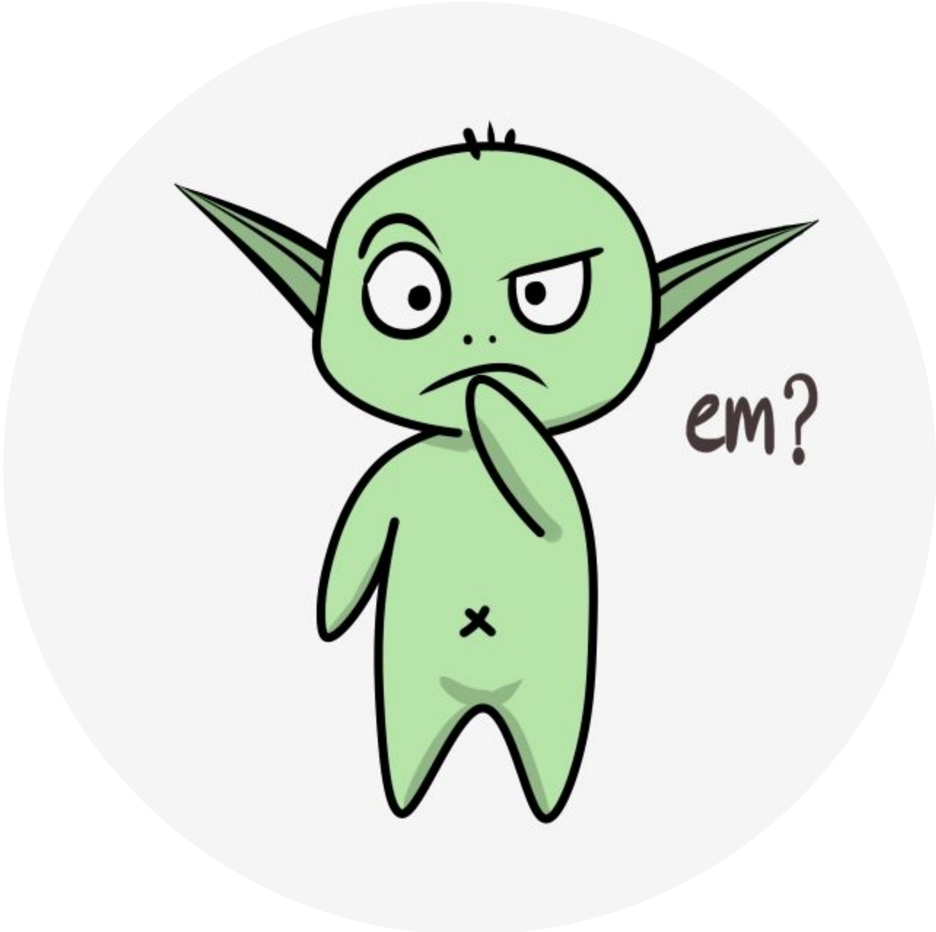
Temporal-
difference
learning

Dynamic
programming

depth
(length)
of update

Monte
Carlo

Exhaustive
search

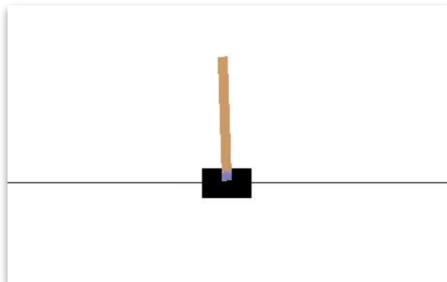


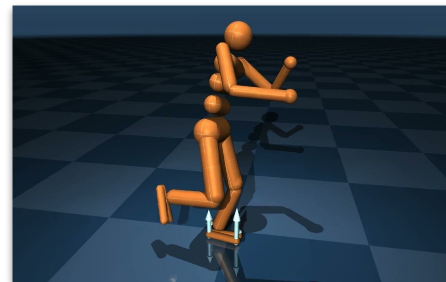
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Part II:
Approximate Solution Methods

What We Have Done So Far: *Tabular RL*

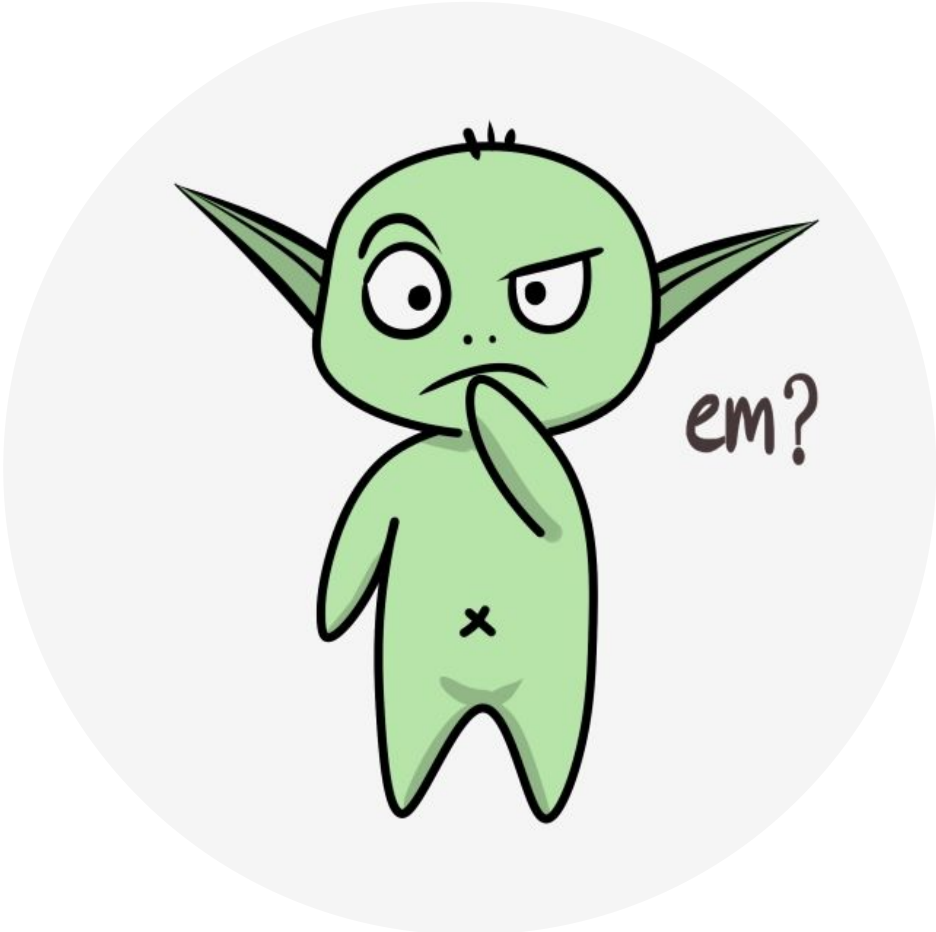
- Absolutely everything we did was in the *tabular* case.
  - There's a huge memory cost in having to fill a table with a ridiculous total number of states.
  - We might never see the same state twice.
  - We cannot expect to find an optimal policy (or value function) even in the limit of infinite time and data.
- What about…





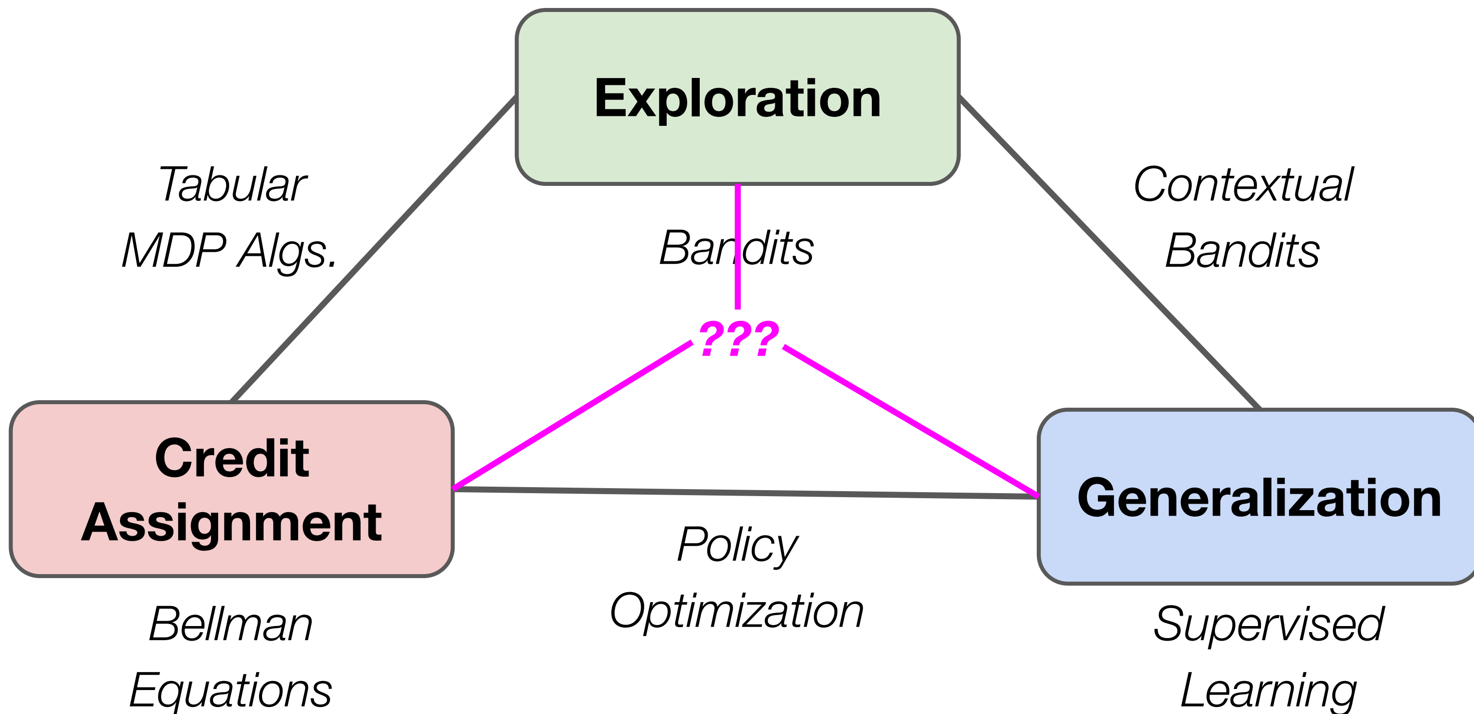
From Now On: *Generalization*

- Instead, we should find a good approximate solution using limited computational resources.
- We need to generalize to from previous encounters with different states that are in some sense similar to the current one.
- We obtain generalization with *function approximation* (often from the supervised learning literature).

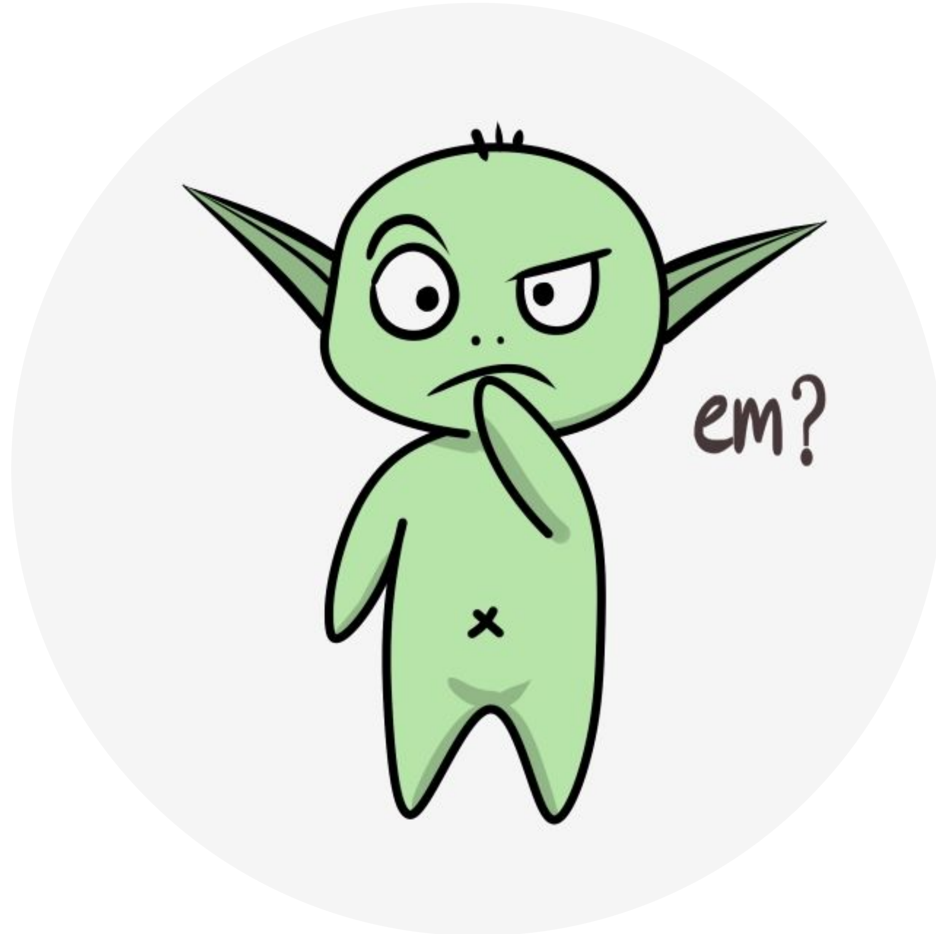


https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Finally! The Three Fundamental Problems of RL

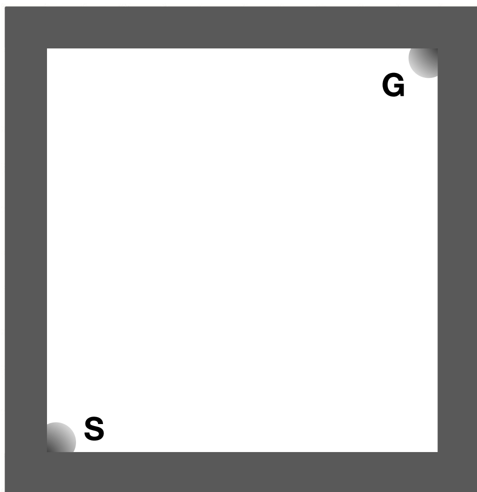


[Inspired by John Langford's slides]



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

Function Approximation – An Example



State space: $\langle x, y \rangle$ coordinates (continuous, no grid) and $\langle \dot{x}, \dot{y} \rangle$ velocity (continuous).

Start state: Somewhere in the bottom left corner, where a suitable $\langle x, y \rangle$ coordinate is selected randomly.

Action space: Adding or subtracting a small force to $\dot{x}$ velocity or $\dot{y}$ velocity, or leaving them unchanged.

Dynamics: Traditional physics, collisions with obstacles are fully elastic and cause the agent to bounce.

Reward function: +1 when you hit the region in G.

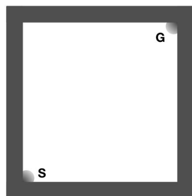
γ: 0.9.

Function Approximation

- In the tabular case, $\mathbf{v}_\pi \in \mathbb{R}^{|\mathcal{S}|}$.
- Instead, we will approximate $\mathbf{v}_\pi$ using a function parameterized by some weights $\mathbf{w} \in \mathbb{R}^d$ where $d \ll |\mathcal{S}|$. We will write $\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$.

Function Approximation

- In the tabular case, $\mathbf{v}_\pi \in \mathbb{R}^{|\mathcal{S}|}$.
- Instead, we will approximate $\mathbf{v}_\pi$ using a function parameterized by some weights $\mathbf{w} \in \mathbb{R}^d$ where $d \ll |\mathcal{S}|$. We will write $\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$.
- An example:



$$\mathbf{s} = \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} \quad \hat{v}(\mathbf{s}, \mathbf{w}) = \mathbf{s}^\top \mathbf{w}$$

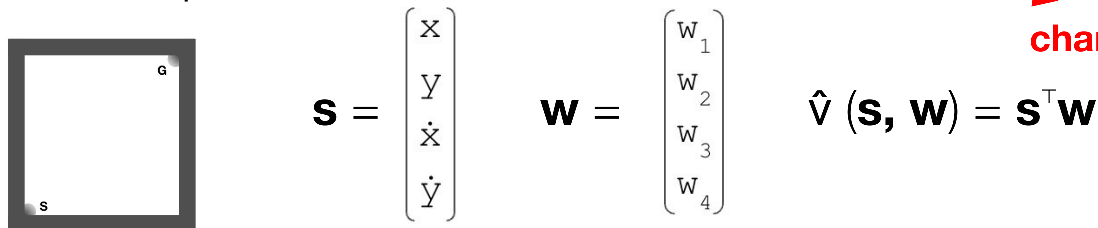
Generalization: When something changes, many states can be affected.

Feature vector

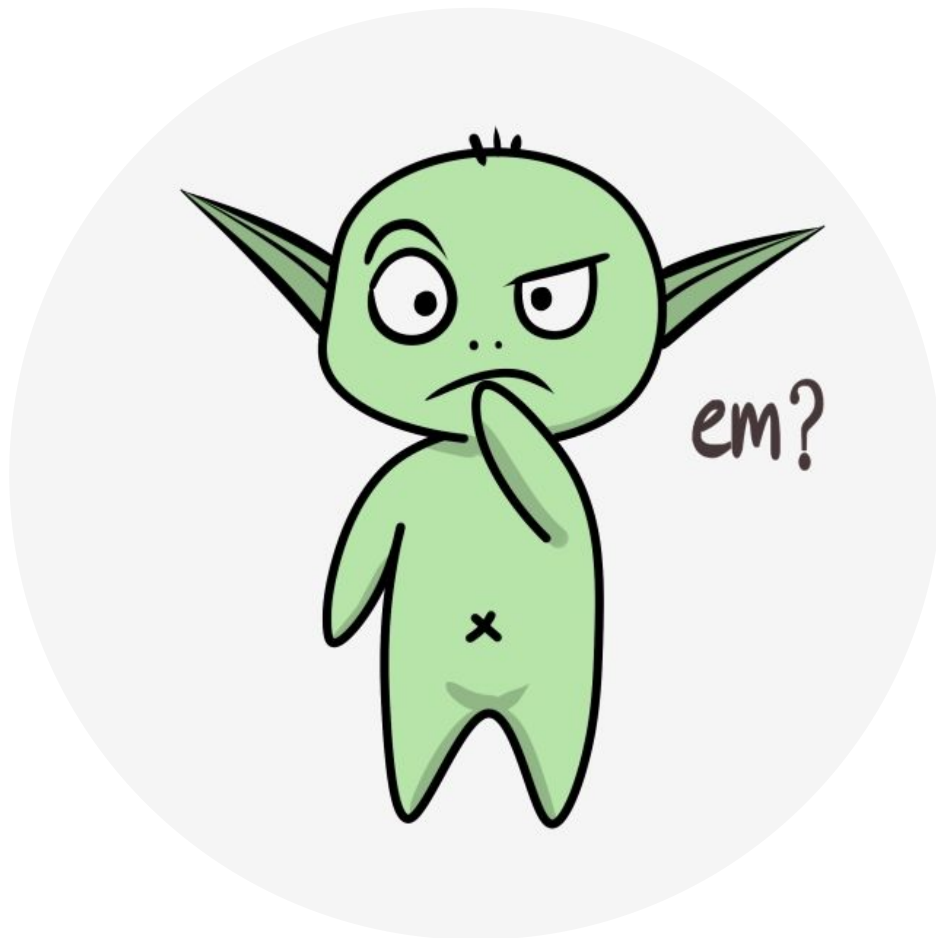
Function Approximation

- In the tabular case, $\mathbf{v}_\pi \in \mathbb{R}^{|\mathcal{S}|}$.
- Instead, we will approximate $\mathbf{v}_\pi$ using a function parameterized by some weights $\mathbf{w} \in \mathbb{R}^d$ where $d \ll |\mathcal{S}|$. We will write $\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$.
- An example:

Generalization: When something changes, many states can be affected.



- Extending RL to function approximation also makes it applicable to partially observable problems, in which the full state is not available to the agent. I'll often use **o** to denote the agent's observation (instead of **s**).



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

Chapter 9

On-policy Prediction with Approximation

Prediction

Value-function Approximation

- We can interpret each update we have seen so far as an example of the desired input-output behavior of the value function.
- Let $s \mapsto u$ denote an individual update, where *s* is the state updated and *u* the update target that *s*'s estimated value is shifted to.
  - $S_t \mapsto G_t$ **Monte Carlo update**
  - $S_t \mapsto R_{t+1} + \gamma\, v_\pi(S_{t+1})$ **TD(0)**
  - $S_t \mapsto G_{t:t+n}$ **n-step TD**
  - $s \mapsto \mathbb{E}_\pi\,[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s]$ **Dynamic Programming**
- Supervised learning methods learn to mimic input-output examples, and when the output are numbers, the process is called *function approximation*.

A note from the textbook

Viewing each update as a conventional training example in this way enables us to use any of a wide range of existing function approximation methods for value prediction. In principle, we can use any method for supervised learning from examples, including artificial neural networks, decision trees, and various kinds of multivariate regression. However, not all function approximation methods are equally well suited for use in reinforcement learning. The most sophisticated artificial neural network and statistical methods all assume a static training set over which multiple passes are made. In reinforcement learning, however, it is important that learning be able to occur online, while the agent interacts with its environment or with a model of its environment. To do this requires methods that are able to learn efficiently from incrementally acquired data. In addition, reinforcement learning generally requires function approximation methods able to handle nonstationary target functions (target functions that change over time). For example, in control methods based on GPI (generalized policy iteration) we often seek to learn q_π while π changes. Even if the policy remains the same, the target values of training examples are nonstationary if they are generated by bootstrapping methods (DP and TD learning). Methods that cannot easily handle such nonstationarity are less suitable for reinforcement learning.

Not really!

The Prediction Objective (A Notion of Accuracy)

- In the tabular case we can have equality, but with FA, not anymore.
  - Making one state's estimate more accurate invariably means making others' less accurate.
- Mean Squared Error:

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

How much do we care about the error in each state s.

Usually, the fraction of time spent in s.
On-policy distribution.

The Prediction Objective (A Notion of Accuracy)

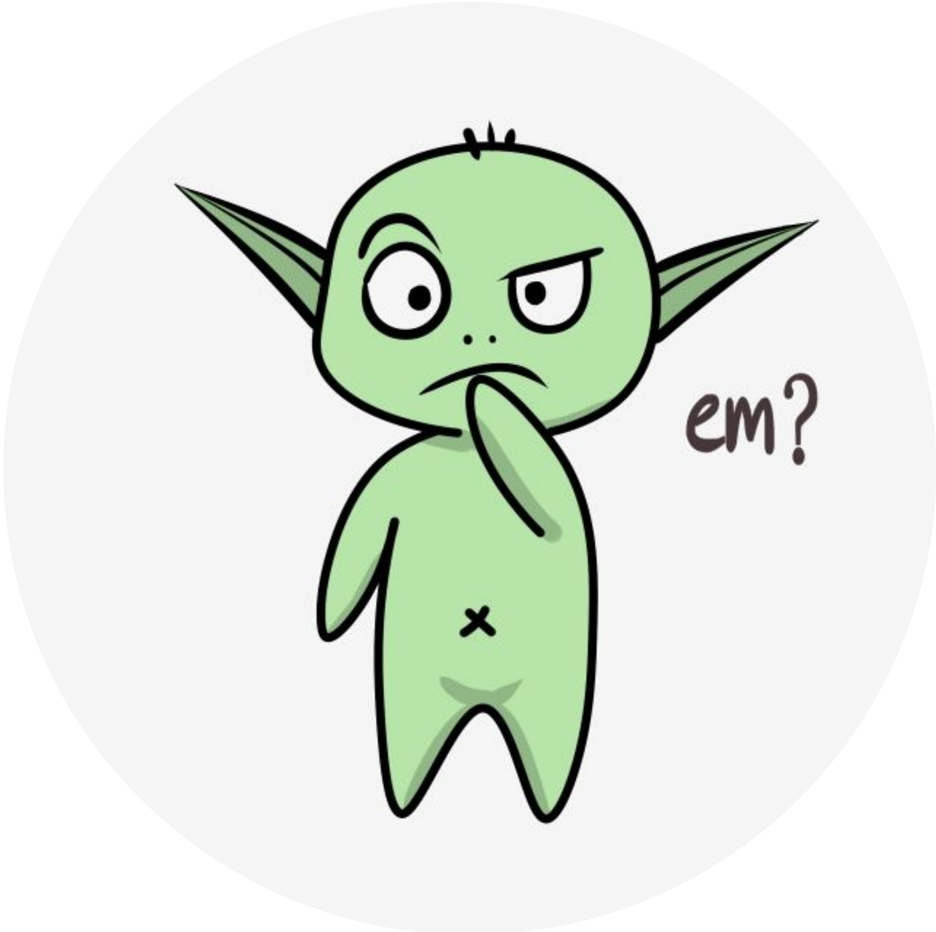
- In the tabular case we can have equality, but with FA, not anymore.
  - Making one state's estimate more accurate invariably means making others' less accurate.
- Mean Squared Error:

How much do we care about the error in each state s.

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

Usually, the fraction of time spent in s.
On-policy distribution.

- When doing nonlinear function approximation, we lose pretty much every guarantee we had (often, even convergence guarantees).



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

Stochastic-gradient and Semi-gradient Methods

- The approximate value function, $\hat{v}(s,\mathbf{w})$, needs to be a differentiable function of **w** for all states.
- For this class, consider that, on each step, we observe a new example $S_t \mapsto v_\pi(S_t)$. Even with the exact target, we need to properly allocate resources.
- *Stochastic gradient-descent (SGD)* is a great strategy:

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

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

Few (one) state at a time

We need to consider the impact of our update. Thus, small updates are often preferred.

A More Realistic Update

- Let U_t denote the *t*-th training example, $S_t \mapsto v_\pi(S_t)$, of some (possibly random), approximation to the true value.

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

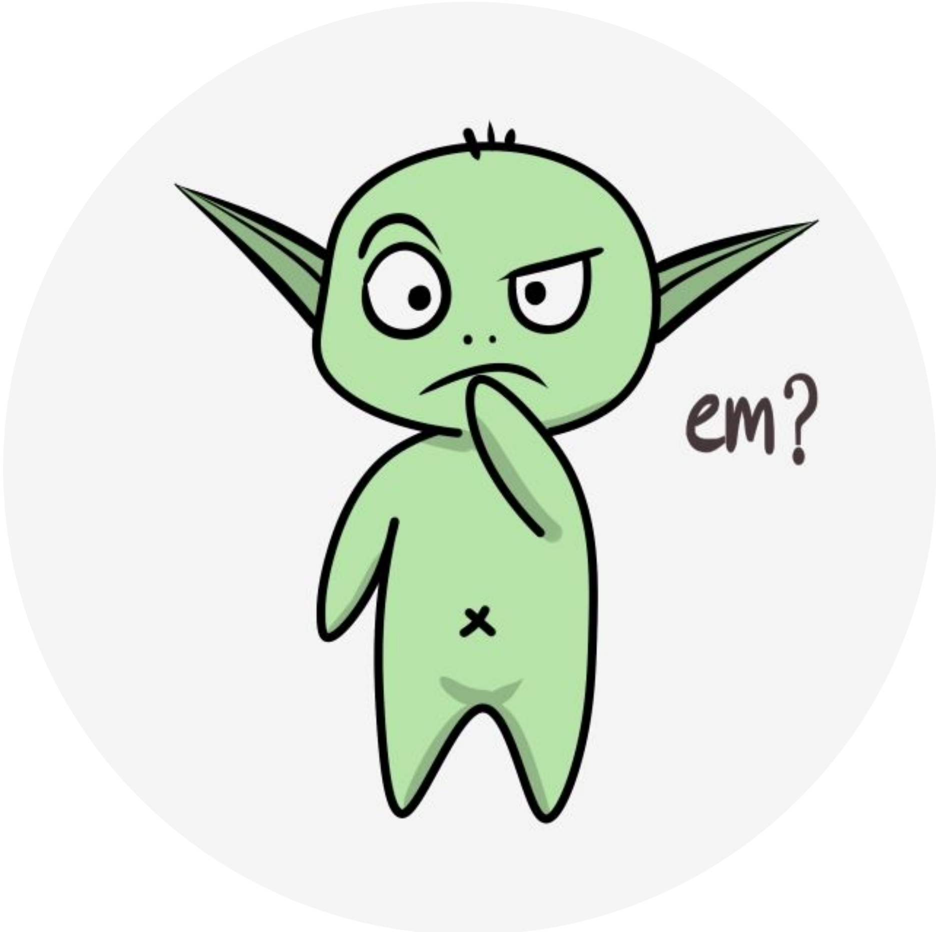
Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

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

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



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

Semi-gradient TD

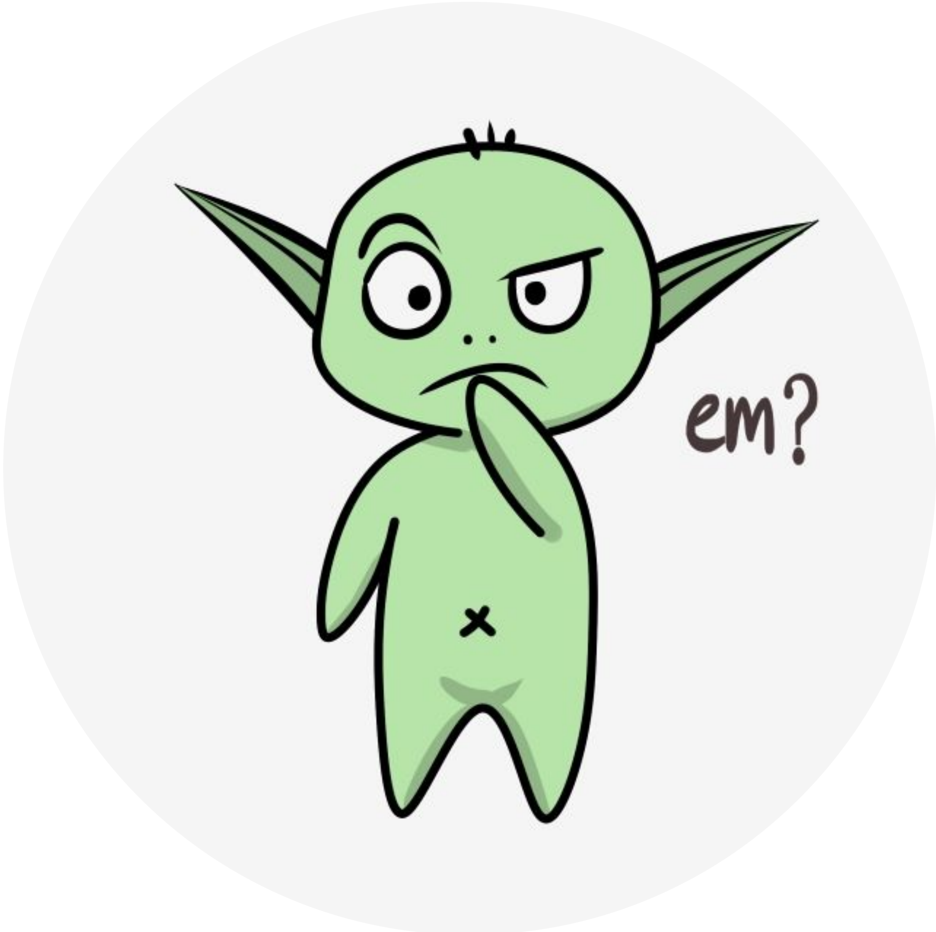
- What if $U_t \doteq R_{t+1} + \gamma \, \hat{\mathbf{v}}(S_{t+1}, \mathbf{w}_t)$?
- We lose several guarantees when we use a bootstrapping estimate as target.
  - The target now also depends on the value of $\mathbf{w}_t$, so the target is not independent of $\mathbf{w}_t$.
- Bootstrapping are not instances of true gradient descent. They take into account the effect of changing the weight vector $\mathbf{w}_t$ on the estimate, but ignore its effect on the target. Thus, they are a *semi-gradient method*.
- Regardless of the theoretical guarantees, we use them all the time ¯_(ツ)_/¯

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

TD Fixed Point with Linear Function Approximation

- We do have convergence results for linear function approximation.

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

TD Fixed Point with Linear Function Approximation

- We do have convergence results for linear function approximation.

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t\Big)\mathbf{x}_t$$
$$= \mathbf{w}_t + \alpha\Big(R_{t+1}\mathbf{x}_t - \mathbf{x}_t\big(\mathbf{x}_t - \gamma \mathbf{x}_{t+1}\big)^\top \mathbf{w}_t\Big)$$

In a steady state, for any given $\mathbf{w}_t$, the expected next weight vector can be written

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

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

TD Fixed Point with Linear Function Approximation

- We do have convergence results for linear function approximation.

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t\Big)\mathbf{x}_t$$
$$= \mathbf{w}_t + \alpha\Big(R_{t+1}\mathbf{x}_t - \mathbf{x}_t\big(\mathbf{x}_t - \gamma \mathbf{x}_{t+1}\big)^\top \mathbf{w}_t\Big)$$

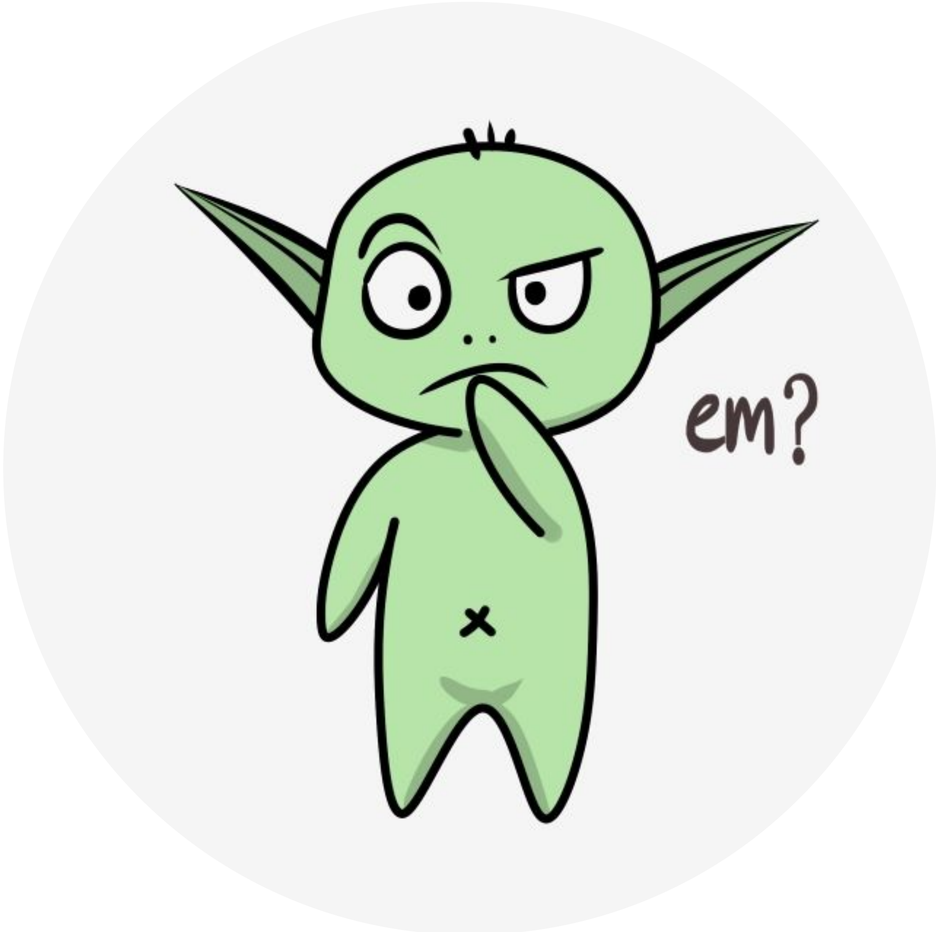
In a steady state, for any given $\mathbf{w}_t$, the expected next weight vector can be written

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

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

It converges to:

$$\mathbf{b} - \mathbf{A}\mathbf{w}_{\text{TD}} = \mathbf{0}$$
$$\Rightarrow \quad \mathbf{b} = \mathbf{A}\mathbf{w}_{\text{TD}}$$
$$\Rightarrow \quad \mathbf{w}_{\text{TD}} \doteq \mathbf{A}^{-1}\mathbf{b}.$$



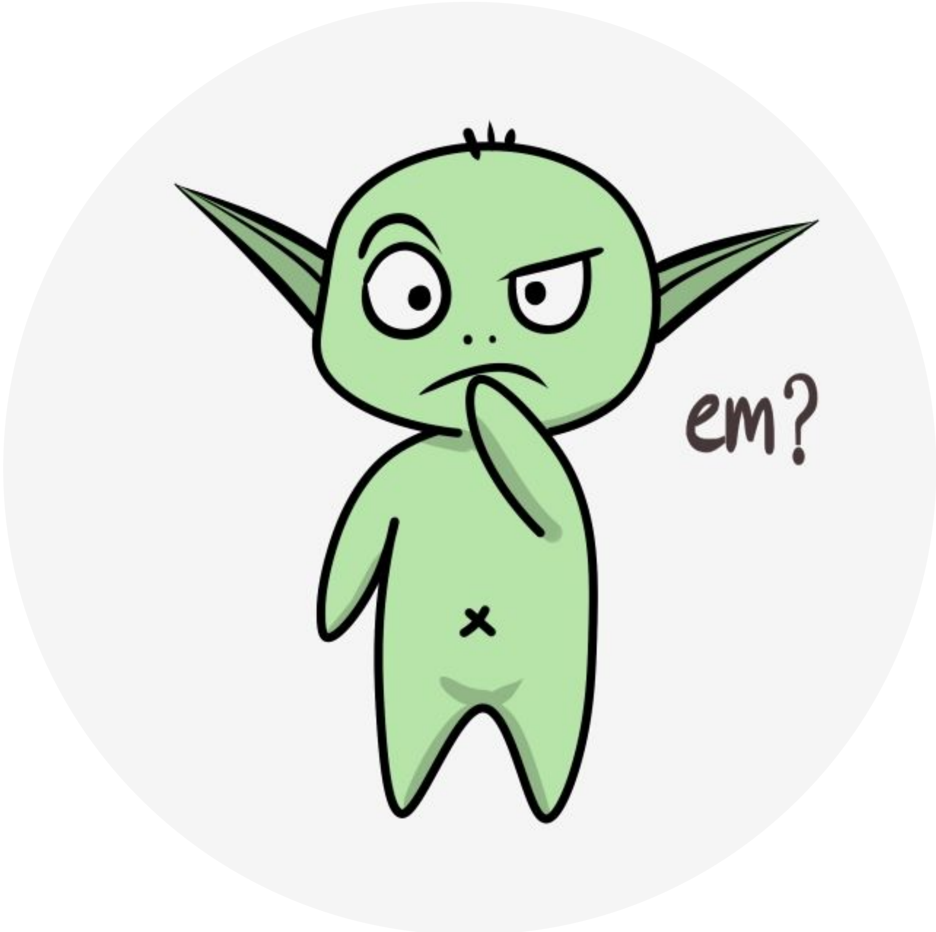
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

n-Step Semi-gradient TD

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, a positive integer n
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (S_t and R_t) can take their index mod $n+1$

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



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

Question

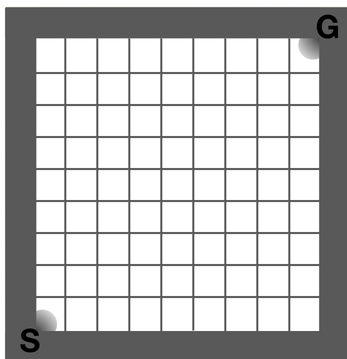
How are tabular methods related to linear function approximation?

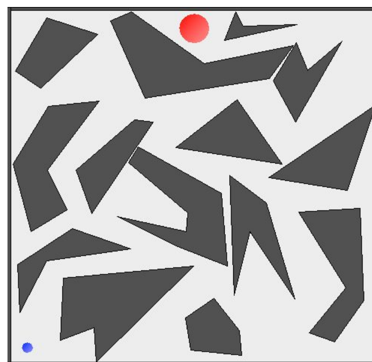
Feature Construction for Linear Methods

- Linear methods can be effective, but they heavily rely on how states are represented in terms of features.
- Feature construction is a way of adding domain knowledge; but at the same time, it went out of fashion because of *deep reinforcement learning*.
- Naïve linear function approximation methods do not take into consideration the interaction between features.

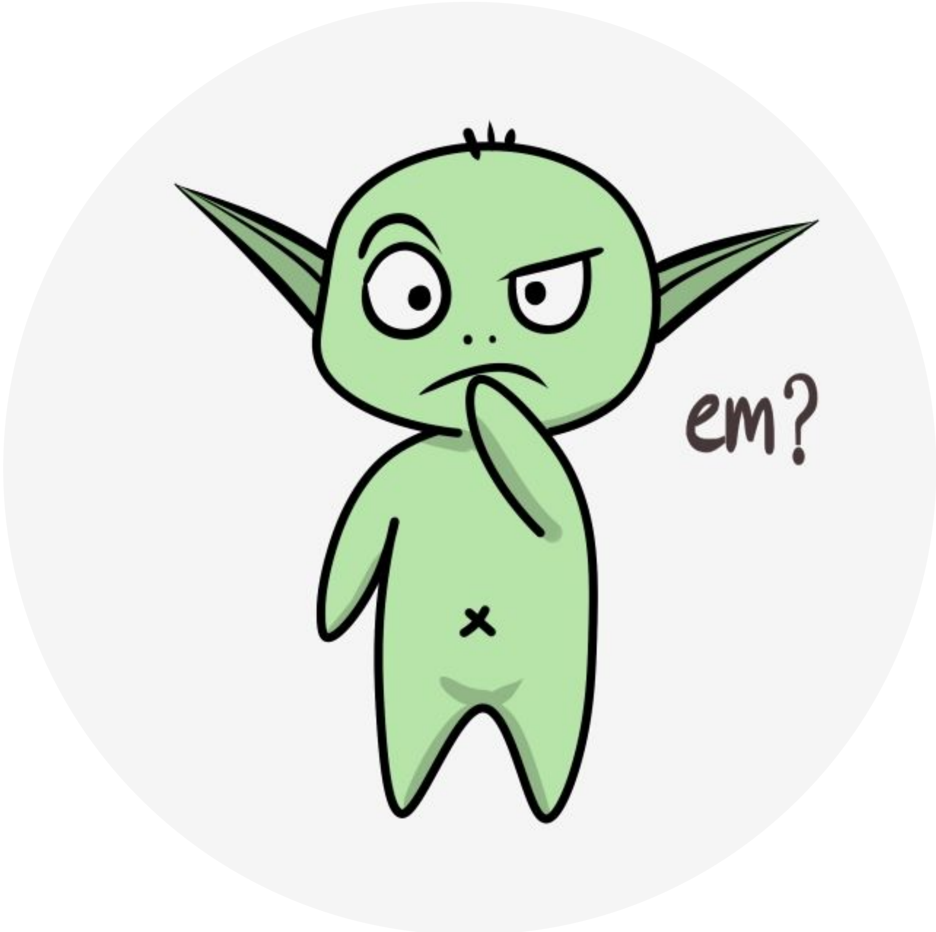
State Aggregation

- Simplest form of representation
- States are grouped together (one component of the vector **w**) for each group.
- State aggregation is a special case of SGD in which the gradient, $\nabla \hat{v}(S_t, \mathbf{w}_t)$, is 1 for S_t's group's component and 0 for the other components.





http://irl.cs.brown.edu/pinball/



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

Polynomials

- Doesn’t work so well, but they are one of the simplest families of features.

Polynomials

- 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.

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

Polynomials

- 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?

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

Polynomials

- 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…

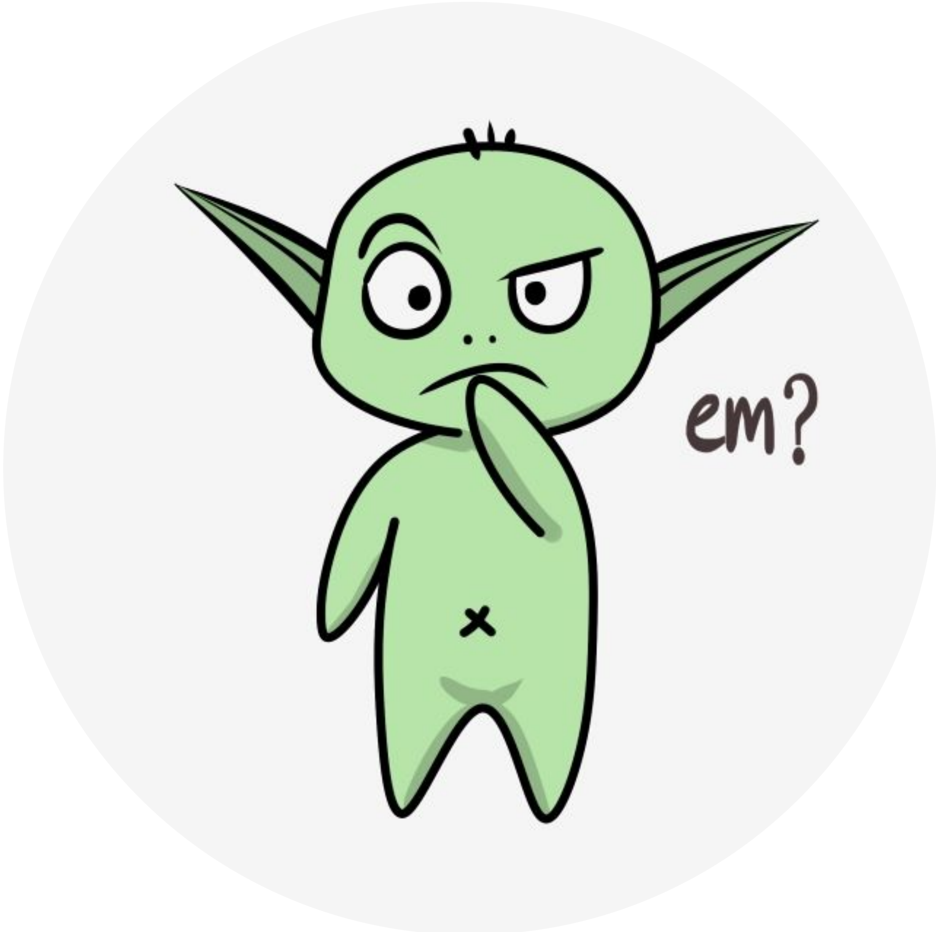
$$\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_2^2)^\top$$

Polynomials

Suppose each state s corresponds to k numbers, s_1, s_2, ..., s_k, with each $s_i \in \mathbb{R}$. For this k-dimensional state space, each order-n polynomial-basis feature x_i can be written as

$$x_i(s) = \Pi_{j=1}^{k} s_j^{c_{i,j}}, \qquad (9.17)$$

where each $c_{i,j}$ is an integer in the set $\{0, 1, \ldots, n\}$ for an integer $n \geq 0$. These features make up the order-n polynomial basis for dimension k, which contains $(n+1)^k$ different features.



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.

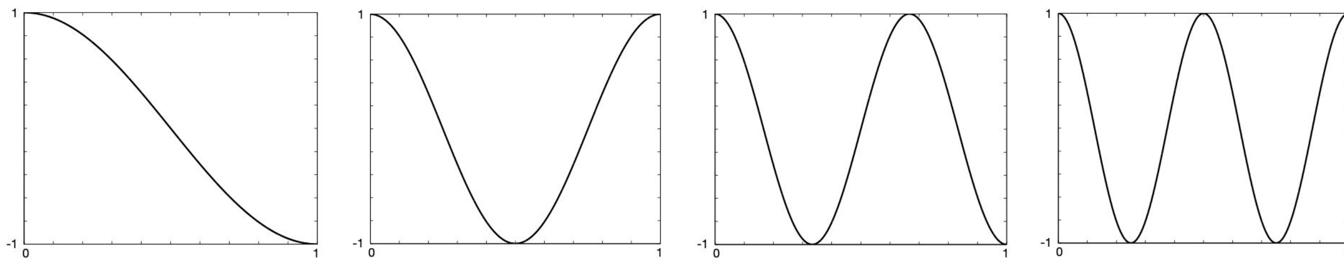


Figure 9.3: One-dimensional Fourier cosine-basis features x_i, $i = 1, 2, 3, 4$, for approximating functions over the interval $[0, 1]$. After Konidaris et al. (2011).

Fourier Basis Beyond One Dimension

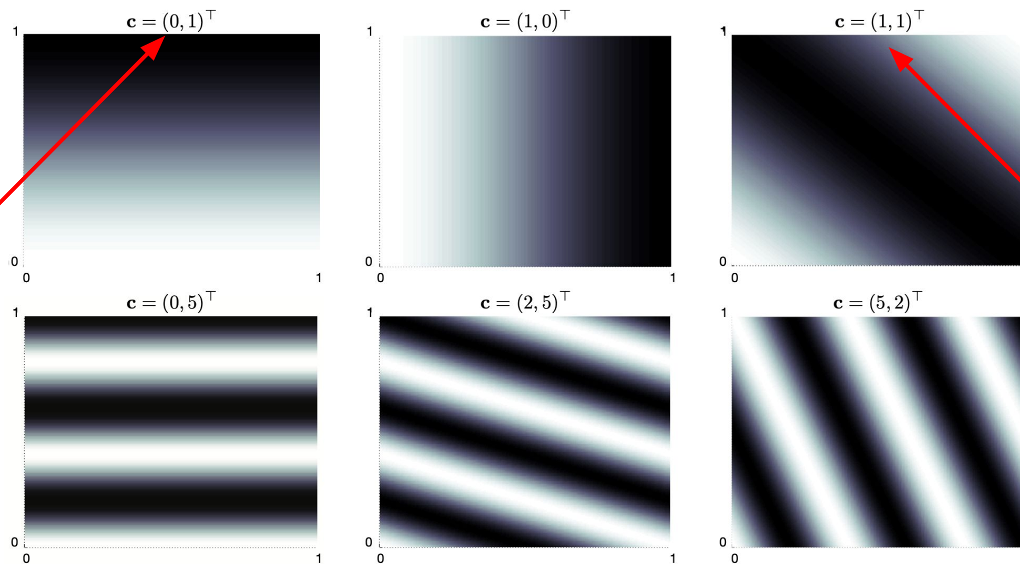
Suppose each state s corresponds to a vector of k numbers, $\mathbf{s} = (s_1, s_2, ..., s_k)^\top$, with each $s_i \in [0, 1]$. The ith feature in the order-n Fourier cosine basis can then be written

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

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

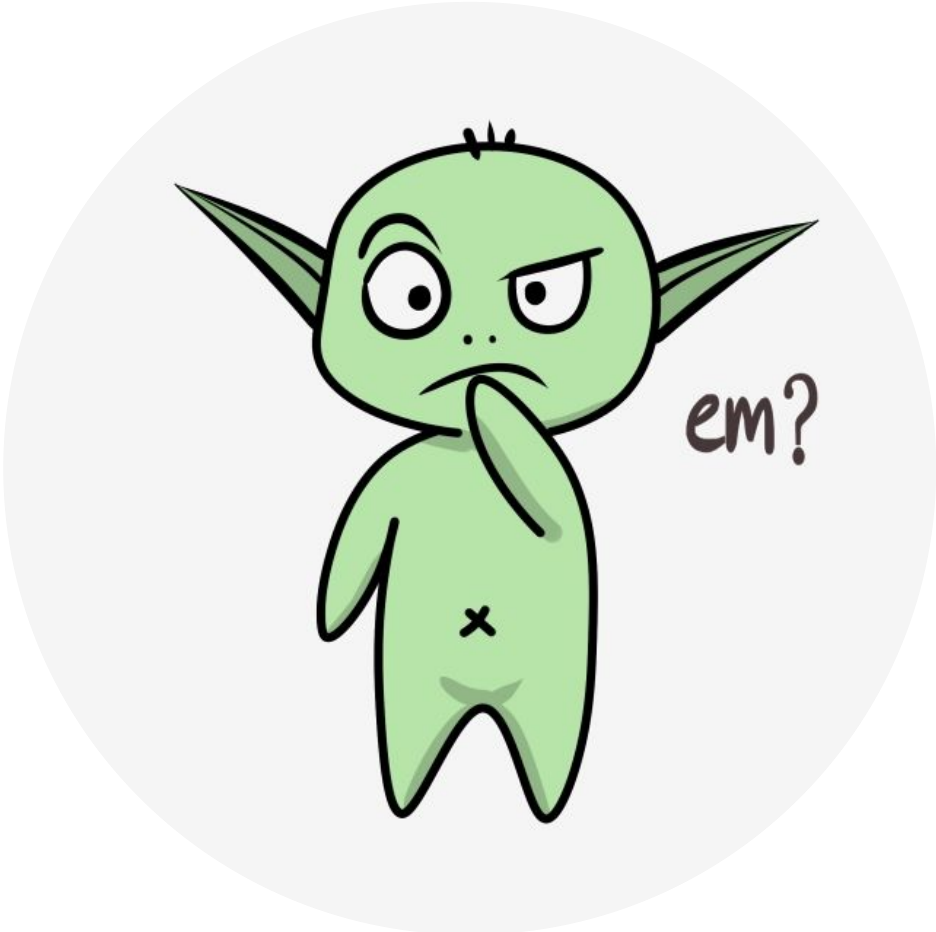
- Consider representing a state as a vector of 2 numbers (k = 2), where each $\mathbf{c}^i = (c^i_1, c^i_2)^\top$.



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

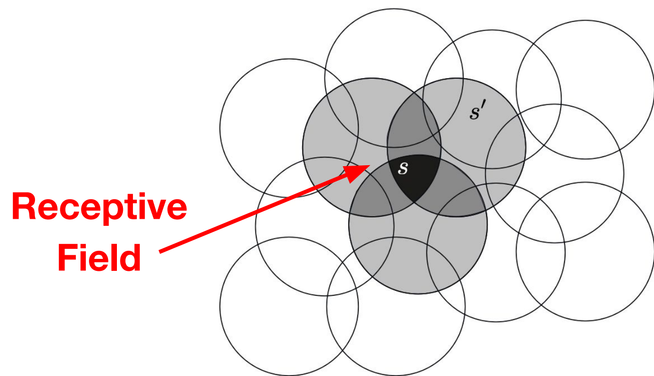
Figure 9.4: A selection of six two-dimensional Fourier cosine features, each labeled by the vector $\mathbf{c}^i$ that defines it (s_1 is the horizontal axis, and $\mathbf{c}^i$ is shown with the index i omitted). After Konidaris et al. (2011).



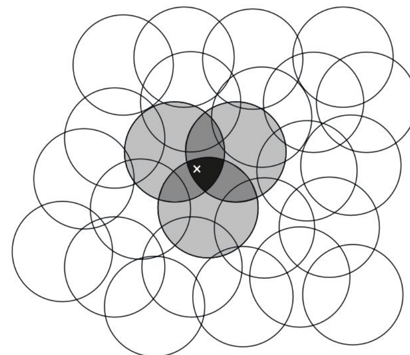
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Coarse Coding

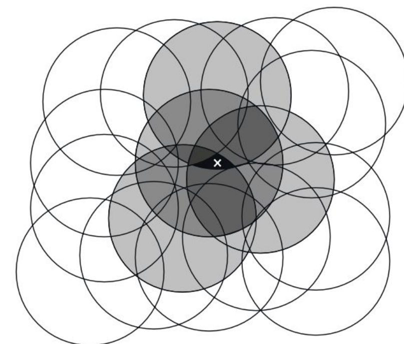
- Consider a task in which the natural representation of the state set is a continuous two- dimensional space.
- We define binary features indicating whether a state is present or not in a specific circle.



The shape defines generalization



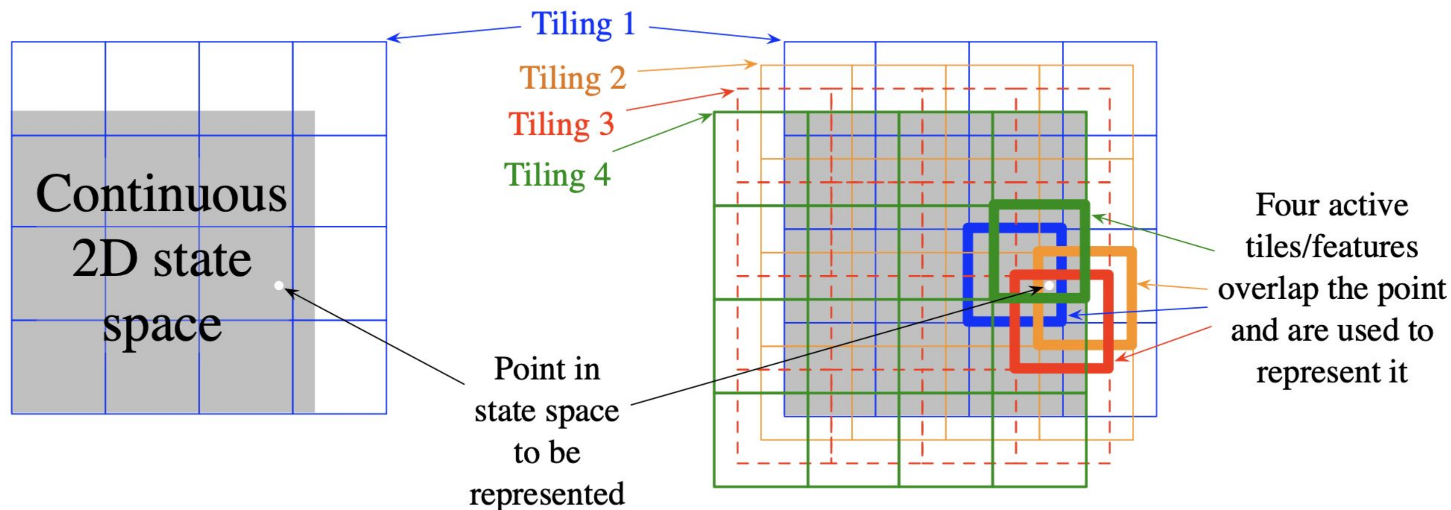
Narrow generalization



Broad generalization

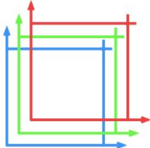
Tile Coding

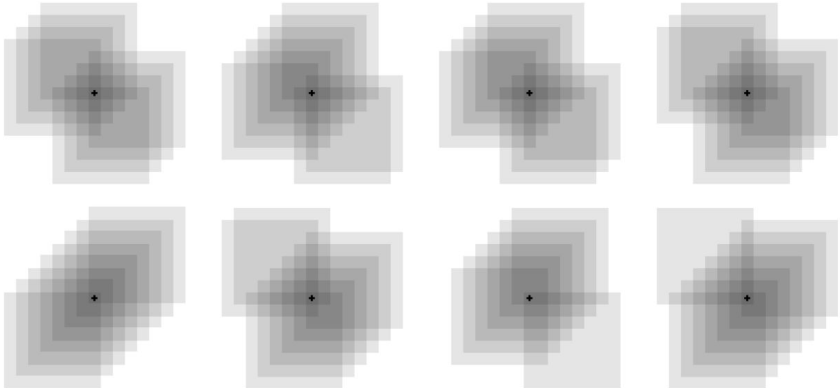
- Tile coding is a form of coarse coding for multi-dimensional continuous spaces (with a fixed number of active features per timestep).



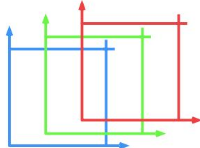
Tile Coding

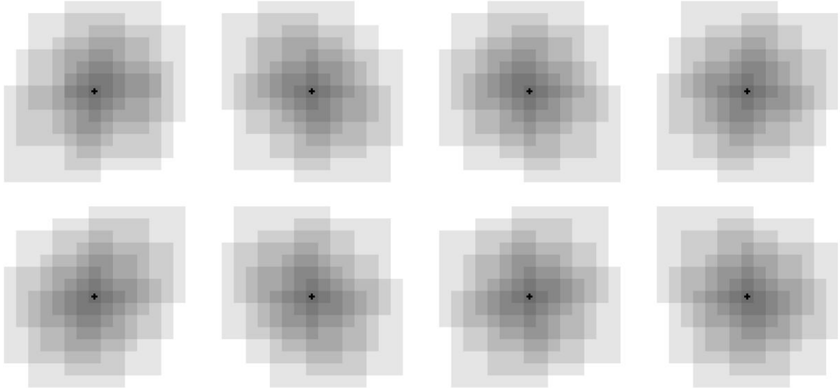
Possible generalizations for uniformly offset tilings

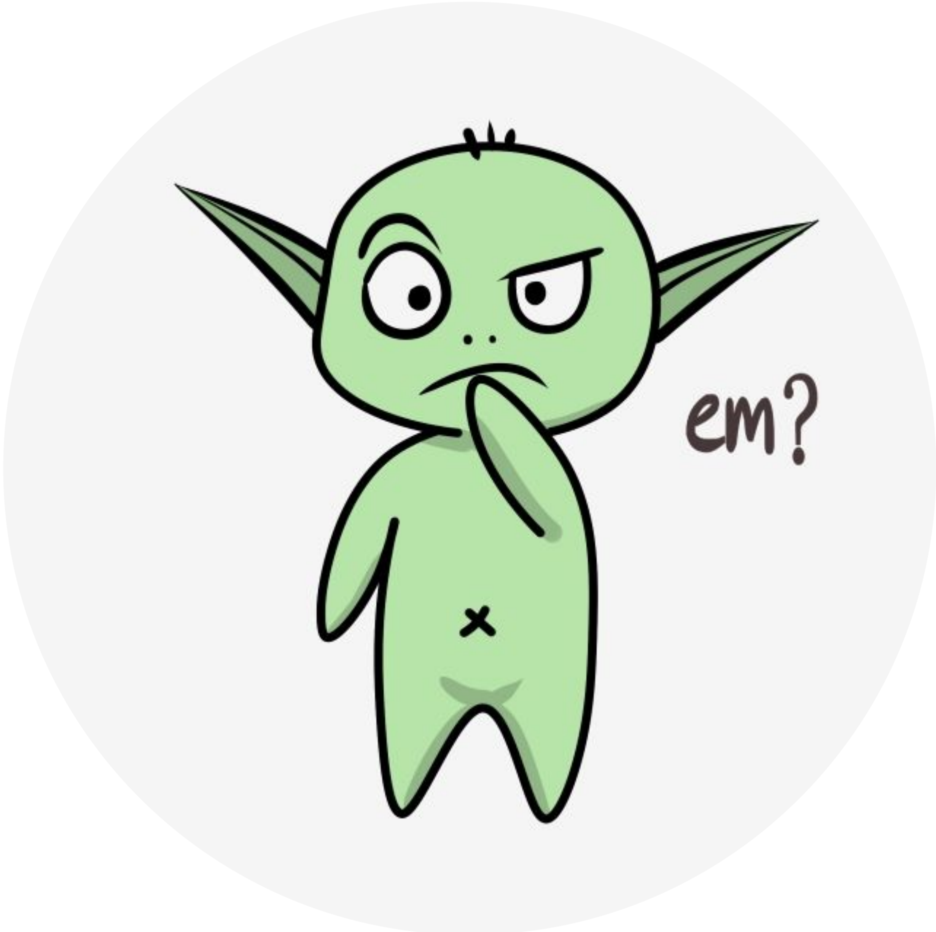




Possible generalizations for asymmetrically offset tilings





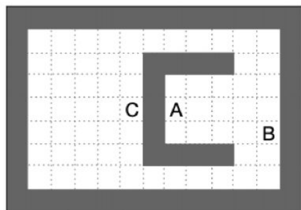


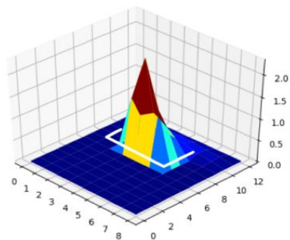
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

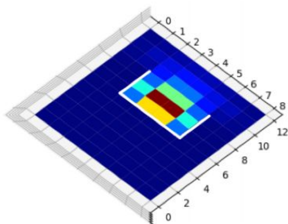
It Isn't that We do Function Approximation Because We Cannot do Tabular Reinforcement Learning

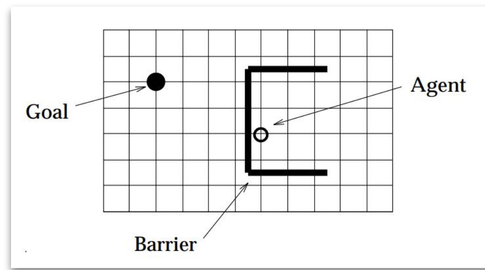
- Successor Representation [Dayan, Neural Computation 1993].

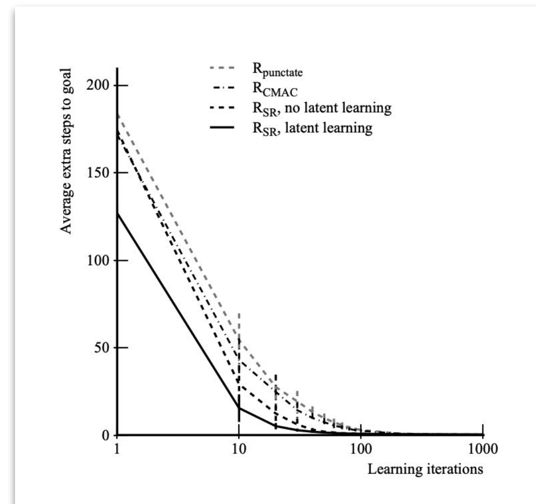
$$\Psi_\pi(s, s') = \mathbb{E}_\pi\left[\sum_t \gamma^t \mathbf{1}_{S_t = s'} | S_0 = s\right]$$

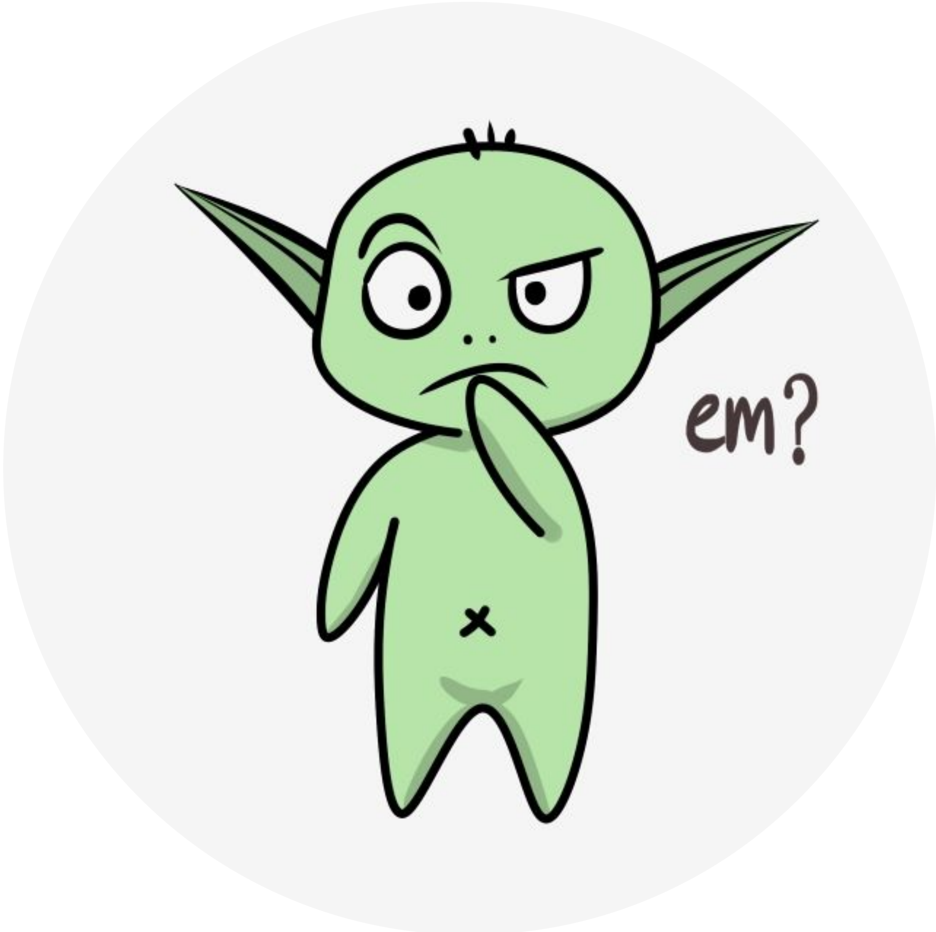










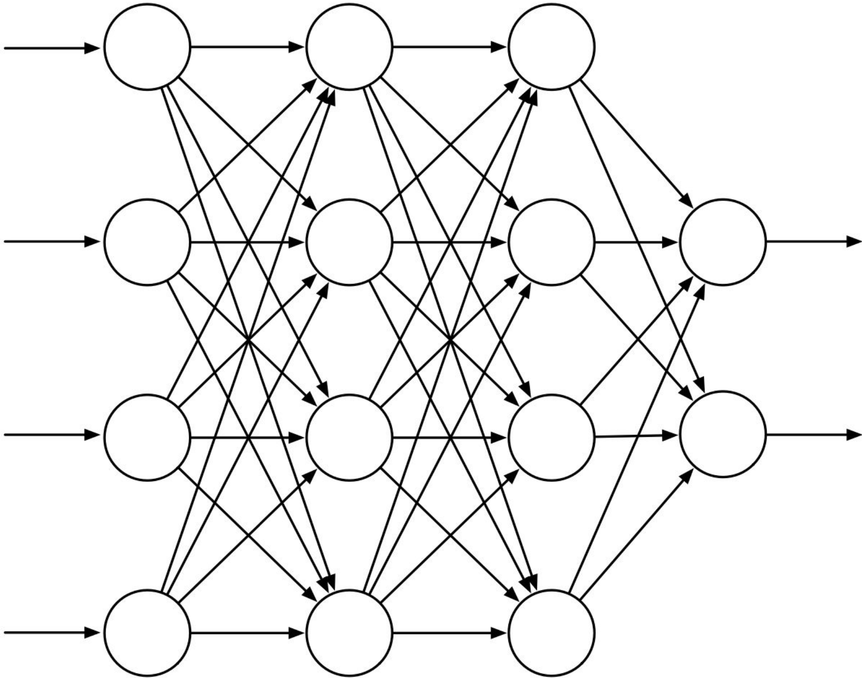


https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

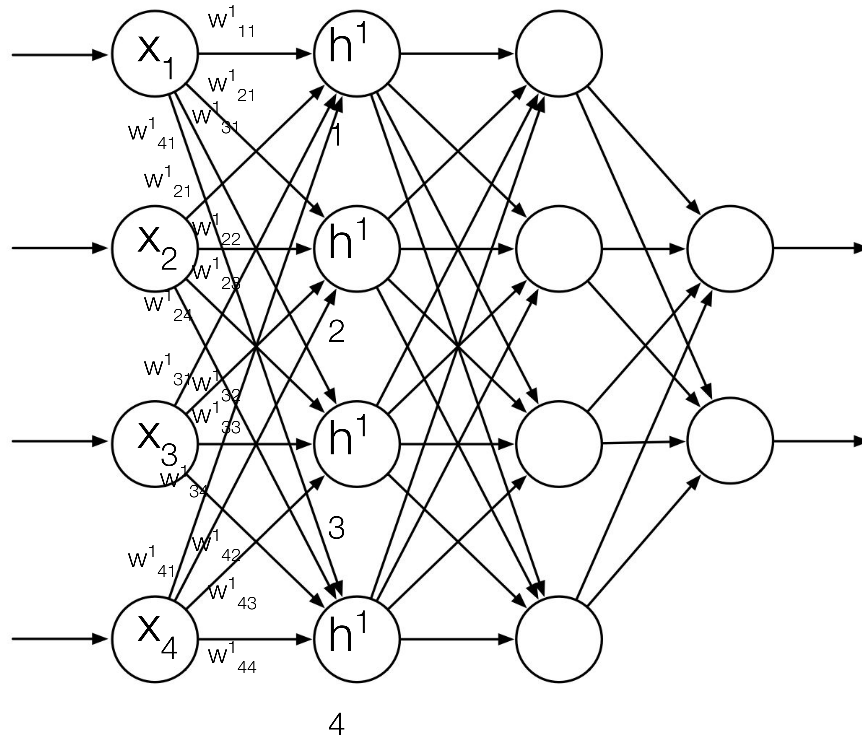
Nonlinear Function Approximation: Artificial Neural Networks

- 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



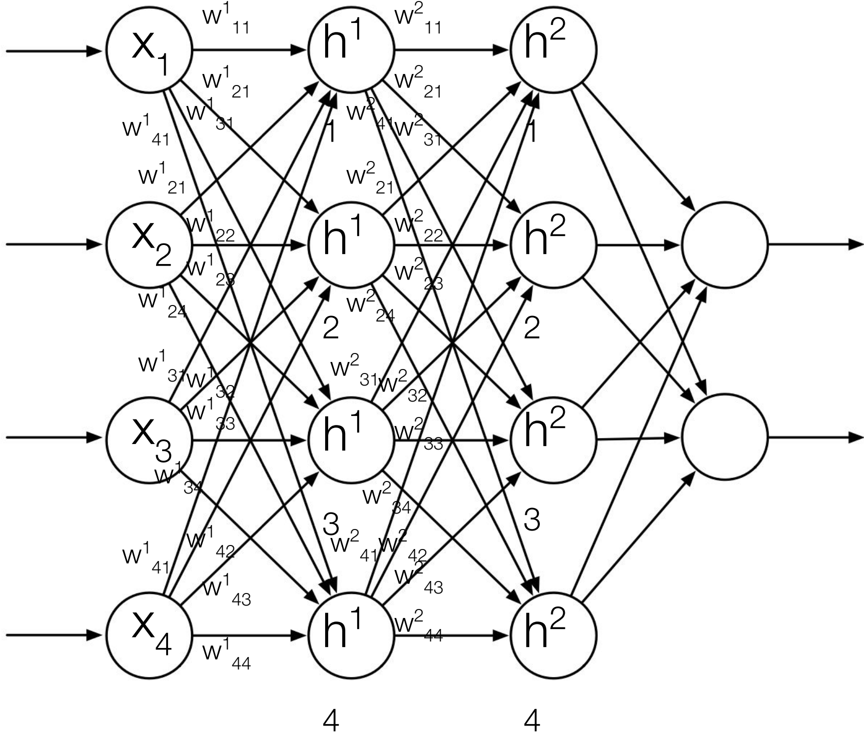
The activation function introduces non-linearity

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

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

s.t. $h^1_1 = x_1w^1_{11} + x_2w^1_{21} + x_3w^1_{31} + x_4w^1_{41} + B^1$

Neural Networks



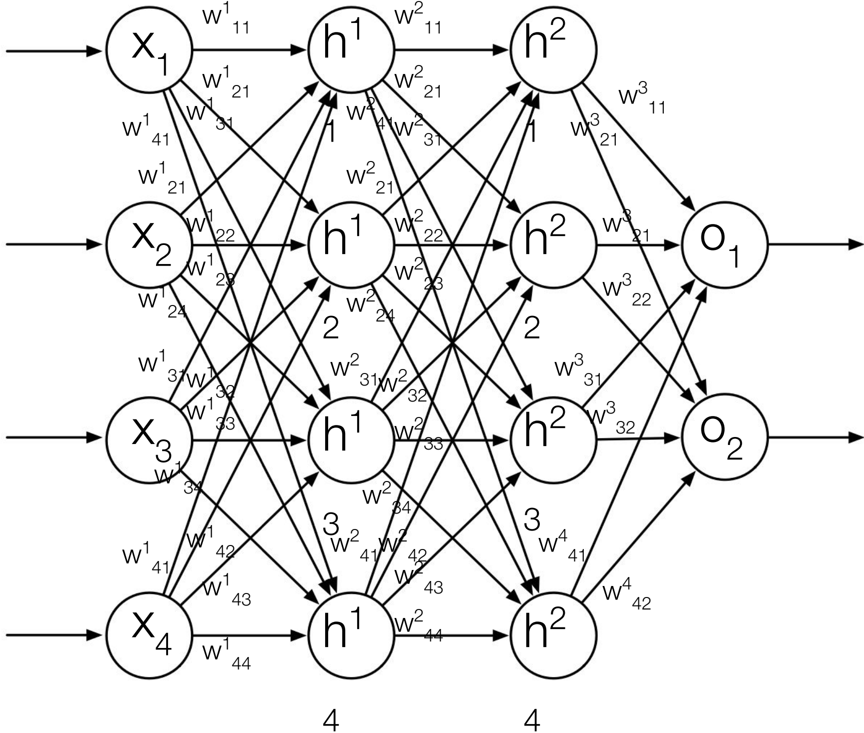
$\mathbf{h^1} = \text{act}(\mathbf{xW^1 + b^1})$

s.t. $h^1{}_1 = x_1w^1{}_{11} + x_2w^1{}_{21} + x_3w^1{}_{31} + x_4w^1{}_{41} + B^1$

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

s.t. $h^2{}_1 = h^1{}_1w^2{}_{11} + h^1{}_2w^2{}_{21} + h^1{}_3w^2{}_{31} + h^1{}_4w^2{}_{41} + B^2$

Neural Networks



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

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

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

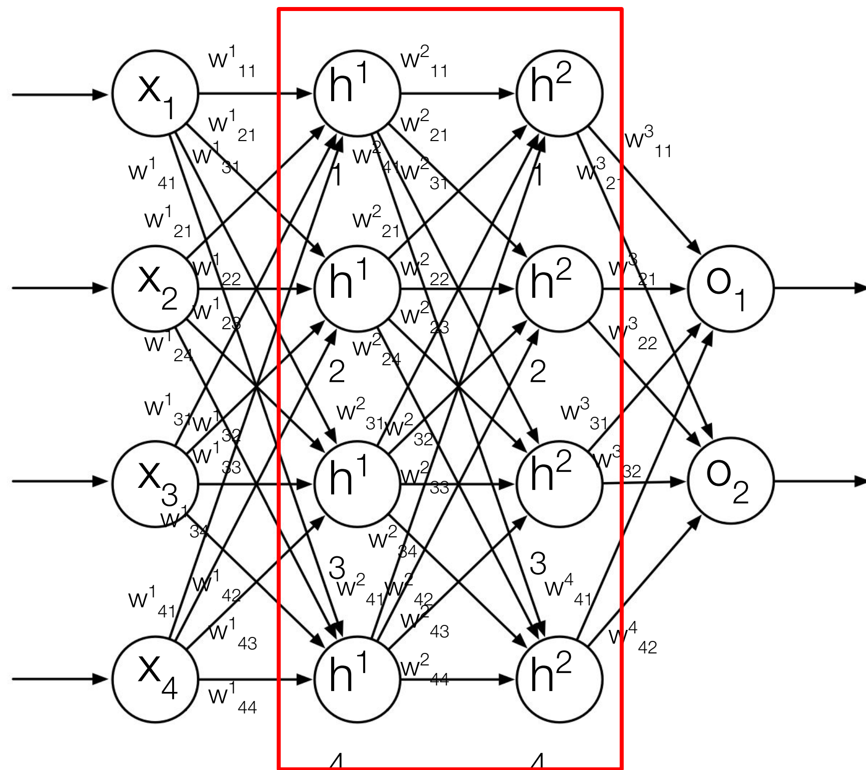
s.t. $h^2_1 = h^1_1 w^2_{11} + h^1_2 w^2_{21} + h^1_3 w^2_{31} + h^1_4 w^2_{41} + B^2$

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

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

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

Neural Networks



Representation (Learned features)

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

s.t. $h^1_1 = x_1w^1_{11} + x_2w^1_{21} + x_3w^1_{31} + x_4w^1_{41} + B^1$

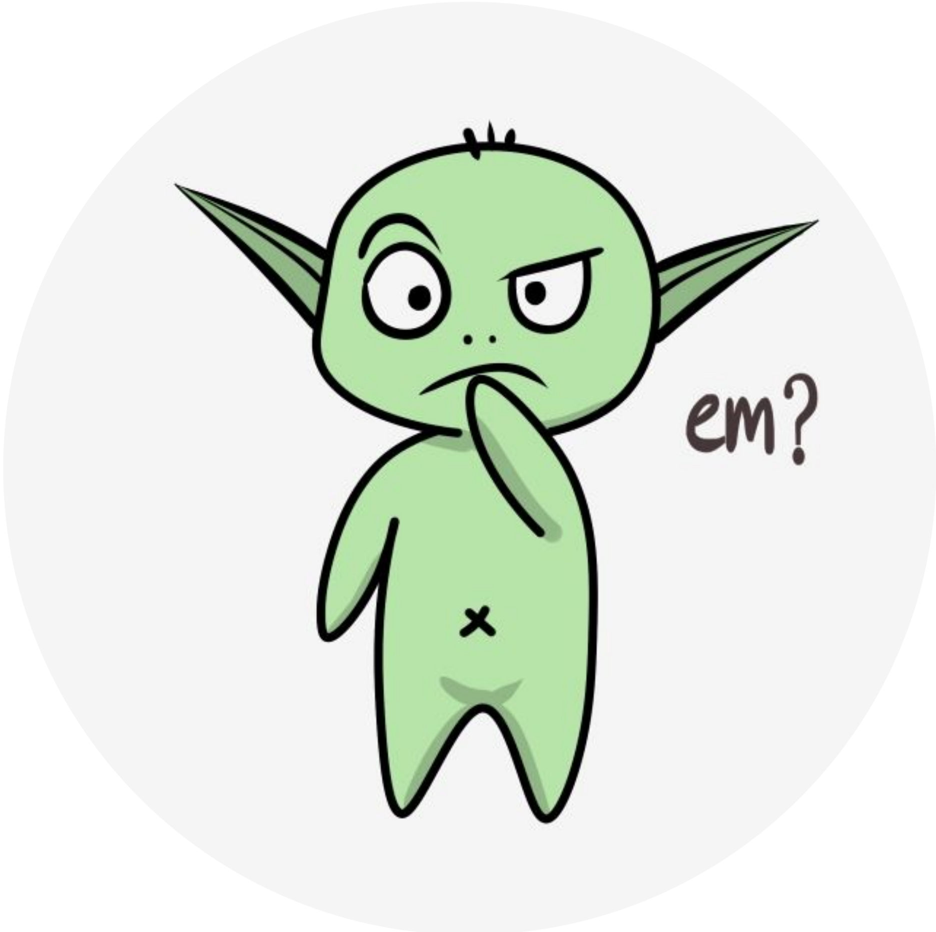
$\mathbf{h^2} = \text{act}(\mathbf{h^1W^2} + \mathbf{b^2})$

s.t. $h^2_1 = h^1_1w^2_{11} + h^1_2w^2_{21} + h^1_3w^2_{31} + h^1_4w^2_{41} + B^2$

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

s.t. $o_1 = h^2_1w^3_{11} + h^2_2w^3_{21} + h^2_3w^3_{31} + h^2_4w^3_{41} + B^2$

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



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

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

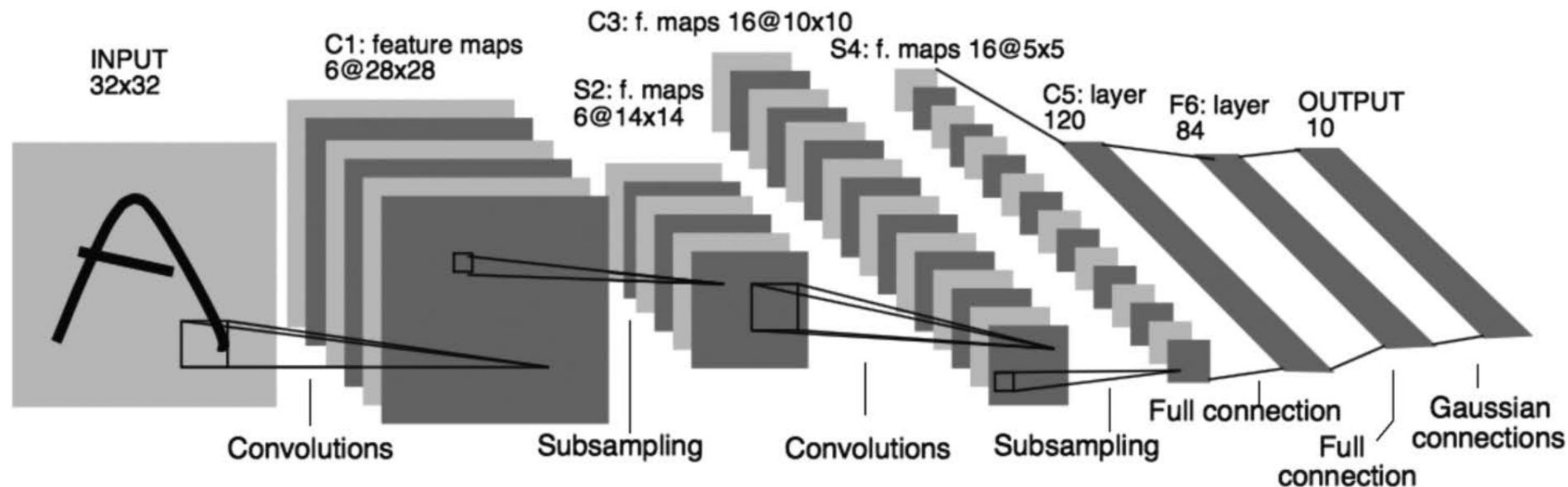
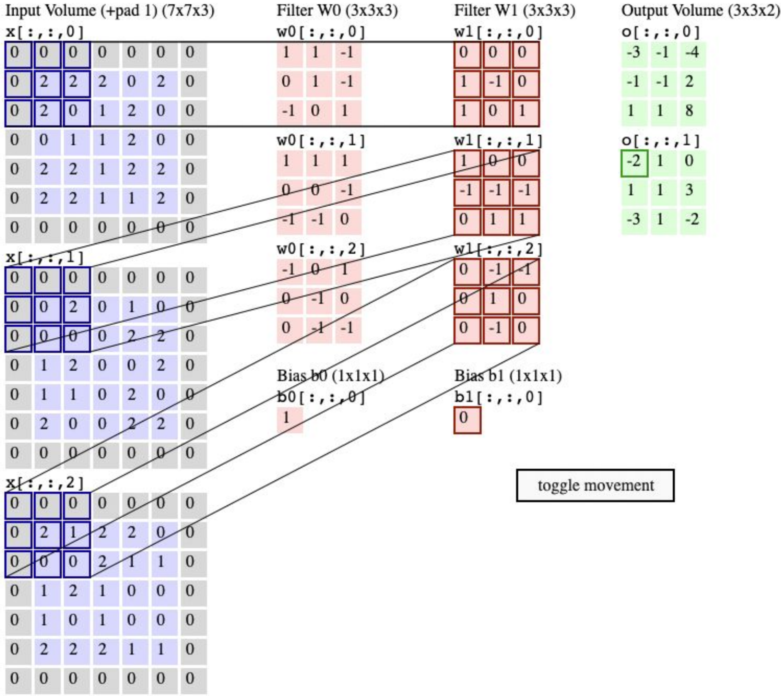


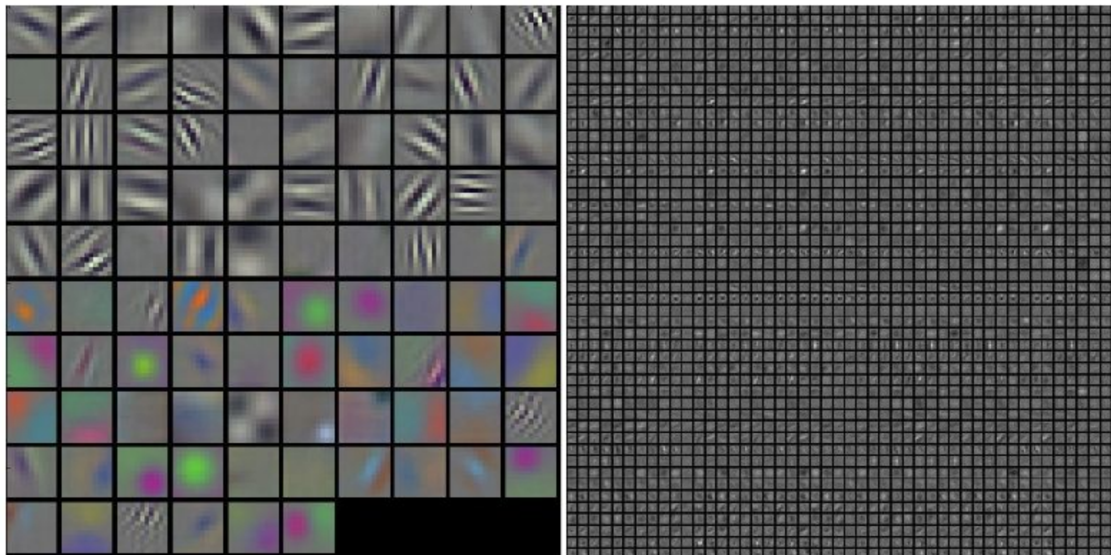
Figure 9.15: Deep Convolutional Network. Republished with permission of Proceedings of the IEEE, from Gradient-based learning applied to document recognition, LeCun, Bottou, Bengio, and Haffner, volume 86, 1998; permission conveyed through Copyright Clearance Center, Inc.

Deep Convolutional Network



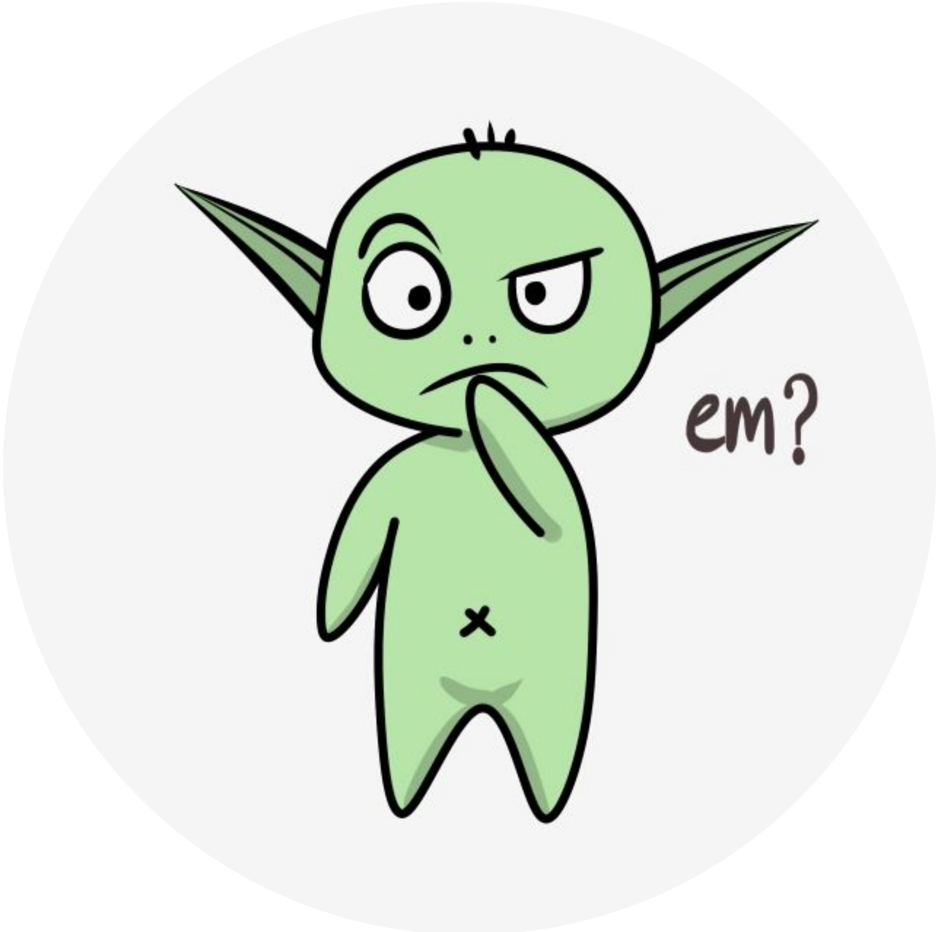
[Figure from demo in https://cs231n.github.io/convolutional-networks/]

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

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

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

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

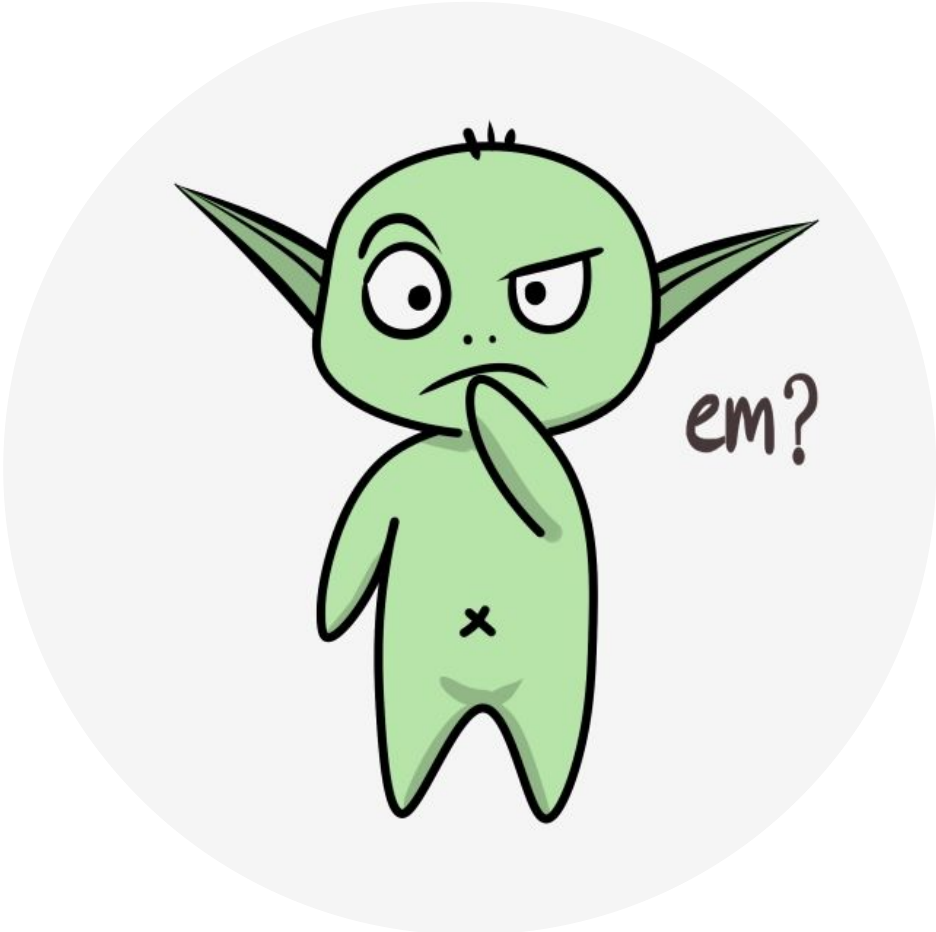
Least-Squares TD (LSTD)

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'; $\mathbf{x}' \leftarrow \mathbf{x}(S')$
    $\mathbf{v} \leftarrow \widehat{\mathbf{A}^{-1}}^\top (\mathbf{x} - \gamma \mathbf{x}')$
    $\widehat{\mathbf{A}^{-1}} \leftarrow \widehat{\mathbf{A}^{-1}} - (\widehat{\mathbf{A}^{-1}}\mathbf{x})\mathbf{v}^\top / (1 + \mathbf{v}^\top \mathbf{x})$ ← **Sherman-Morrison formula**
    $\widehat{\mathbf{b}} \leftarrow \widehat{\mathbf{b}} + R\mathbf{x}$
    $\mathbf{w} \leftarrow \widehat{\mathbf{A}^{-1}}\widehat{\mathbf{b}}$
    $S \leftarrow S'$; $\mathbf{x} \leftarrow \mathbf{x}'$
  until S' is terminal

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



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.

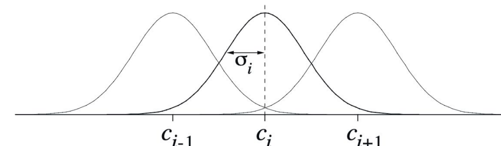
Kernel-based Function Approximation

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

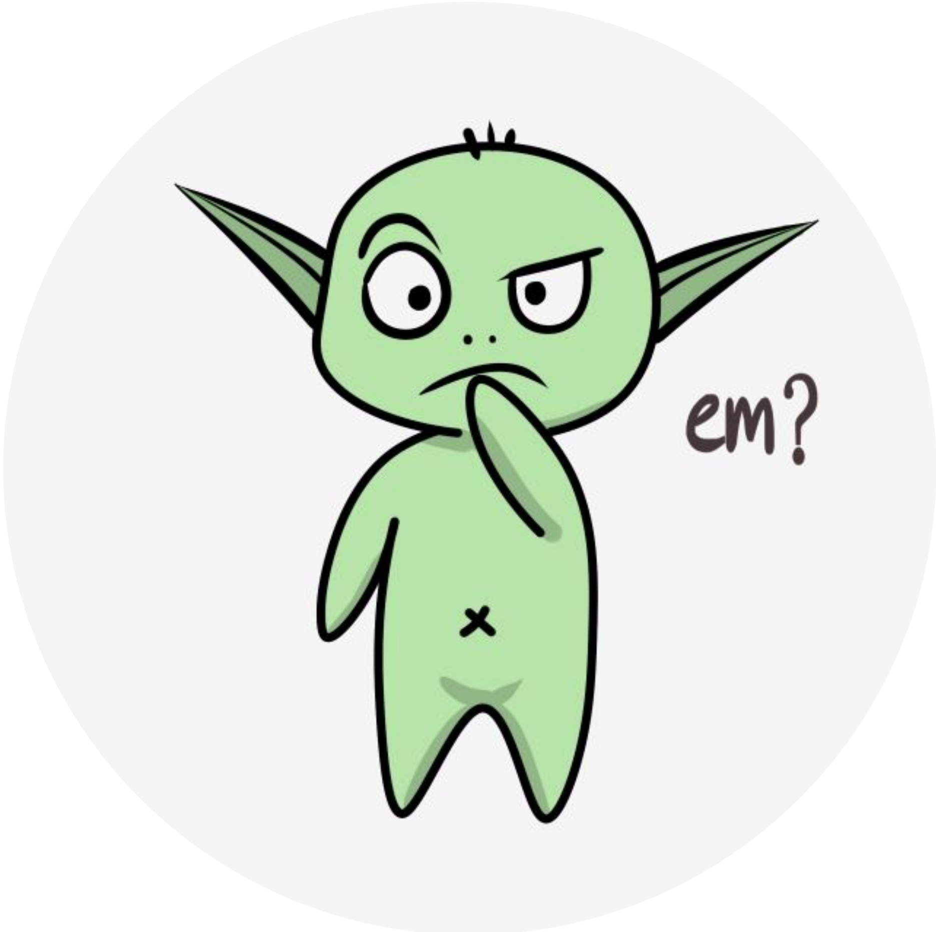
Loop for each episode:
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        If S' is terminal:
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
            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 $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$
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

To be continued...