Example: Animal Learning

- RL studied experimentally for more than 60 years in psychology
  - Rewards: food, pain, hunger, drugs, etc.
  - Mechanisms and sophistication debated

- Example: foraging
  - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
  - Bees have a direct neural connection from nectar intake measurement to motor planning area

Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to V(s) using a neural network
- Combined with depth 3 search, one of the top 3 players in the world
- You could imagine training Pacman this way…
- … but it’s tricky!

The Story So Far: MDPs and RL

Things we know how to do:

- We can solve small MDPs exactly, offline

Techniques:

- Value and policy iteration

Bellman Equation:

\[ V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]
Passive Learning

- Simplified task
  - You don’t know the transitions $T(s,a,s')$
  - You don’t know the rewards $R(s,a,s')$
  - You are given a policy $\pi(s)$
  - Goal: learn the state values
  - ... what policy evaluation did

- In this case:
  - Learner “along for the ride”
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We’ll get to the active case soon
  - This is NOT offline planning! You actually take actions in the world and see what happens…

Recap: Model-Based Policy Evaluation

- Simplified Bellman updates to calculate $V$ for a fixed policy:
  - New $V$ is expected one-step-look-ahead using current $V$
  - Unfortunately, need $T$ and $R$

Model-Based Learning

- Idea:
  - Learn the model empirically through experience
  - Solve for values as if the learned model were correct

- Simple empirical model learning
  - Count outcomes for each $s,a$
  - Normalize to give estimate of $T(s,a,s')$
  - Discover $R(s,a,s')$ when we experience $(s,a,s')$

- Solving the MDP with the learned model
  - Iterative policy evaluation, for example

Sample-Based Policy Evaluation?

- Who needs $T$ and $R$? Approximate the expectation with samples (drawn from $T$)

Model-Free Learning

- Want to compute an expectation weighted by $P(x)$:

\[
E[f(x)] = \sum_x P(x) f(x)
\]

- Model-based: estimate $P(x)$ from samples, compute expectation

\[
x_i \sim P(x), \quad \hat{P}(x) = \text{count}(x) / k, \quad E[f(x)] \approx \sum_x \hat{P}(x) f(x)
\]

- Model-free: estimate expectation directly from samples

\[
x_i \sim P(x), \quad E[f(x)] \approx \frac{1}{k} \sum_i f(x_i)
\]

- Why does this work? Because samples appear with the right frequencies!
Example: Direct Estimation

- Episodes:
  - (1,1) up -1
  - (1,2) up -1
  - (1,2) right -1
  - (2,1) up -1
  - (2,2) right -1
  - (3,3) right -1
  - (3,2) up -1
  - (3,3) right -1
  - (4,3) exit +100

\[ V(2,3) = \frac{96 + (-103)}{2} = -3.5 \]
\[ V(3,3) = \frac{99 + 97 + (-102)}{3} = 31.3 \]

\[ \gamma = 1, R = -1 \]

Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update \( V(s) \) each time we experience \((s,a,s',r)\)
  - Likely \( s' \) will contribute updates more often

- Temporal difference learning
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!

\[ \text{Sample of } V(s): \quad \text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s') \]
\[ \text{Update to } V(s): \quad V^\pi(s) = (1 - \alpha) V^\pi(s) + \alpha \text{sample} \]
\[ \text{Same update:} \quad V^\pi(s) = V^\pi(s) + \alpha (\text{sample} - V^\pi(s)) \]

Example: TD Policy Evaluation

\[ V^\pi(s) = (1 - \alpha) V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')] \]

\[ \gamma = 1, \alpha = 0.5 \]

Problems with TD Value Learning

- TD value learning is a model-free way to do policy evaluation
- However, if we want to turn values into a (new) policy, we’re sunk:
  \[ \pi(s) = \arg \max_a Q^\pi(s, a) \]
  \[ Q^\pi(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')] \]
- Idea: learn Q-values directly
- Makes action selection model-free too!

The Story So Far: MDPs and RL

Things we know how to do:
- We can solve small MDPs exactly, offline
- We can estimate values \( V^\pi(s) \) directly for a fixed policy \( \pi \).

Techniques:
- Value and policy iteration
- Temporal difference learning
Active Learning

- Full reinforcement learning
  - You don’t know the transitions $T(s,a,s’)$
  - You don’t know the rewards $R(s,a,s’)$
  - You can choose any actions you like
  - Goal: learn the optimal policy
  - ... what value iteration did!
- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens...

Detour: Q-Value Iteration

- Value iteration: find successive approx optimal values
  - Start with $V_0(s) = 0$, which we know is right (why?)
  - Given $V_i$: calculate the values for all states for depth $i+1$:
    $$V_{i+1}(s) = \max_a \sum_{s’} T(s,a,s’) \left[ R(s,a,s’) + \gamma V_i(s’) \right]$$
- But Q-values are more useful!
  - Start with $Q_0(s,a) = 0$, which we know is right (why?)
  - Given $Q_i$, calculate the q-values for all q-states for depth $i+1$:
    $$Q_{i+1}(s,a) = \sum_{s’} T(s,a,s’) \left[ R(s,a,s’) + \gamma \max_{a’} Q_i(s’,a’) \right]$$

Q-Learning

- Q-Learning: sample-based Q-value iteration
  - Learn $Q^*(s,a)$ values
    - Receive a sample $(s,a,s’,r)$
    - Consider your old estimate: $Q(s,a)$
    - Consider your new sample estimate:
      $$Q^*(s,a) = \max_a \sum_{s’} T(s,a,s’) \left[ R(s,a,s’) + \gamma \max_{a’} Q(s’,a’) \right]$$
      $$\text{sample} = R(s,a,s’) + \gamma \max_{a’} Q(s’,a’)$$
    - Incorporate the new estimate into a running average:
      $$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \text{[sample]}$$

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
    - ... but not decrease it too quickly!
  - Basically doesn’t matter how you select actions (!)
- Neat property: off-policy learning
  - learn optimal policy without following it (some caveats)

Example: Greedy ADP

- Imagine we find the lower path to the good exit first
- Some states will never be visited following this policy from (1,1)
- We’ll keep re-using this policy because following it never collects the regions of the model we need to learn the optimal policy

What Went Wrong?

- Problem with following optimal policy for current model:
  - Never learn about better regions of the space if current policy neglects them
- Fundamental tradeoff: exploration vs. exploitation
  - Exploration: must take actions with suboptimal estimates to discover new rewards and increase eventual utility
  - Exploitation: once the true optimal policy is learned, exploration reduces utility
  - Systems must explore in the beginning and exploit in the limit
Exploration / Exploitation

- Several schemes for forcing exploration
  - Simplest: random actions (ε greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability 1-ε, act according to current policy

- Problems with random actions?
  - You do explore the space, but keep thrashing around once learning is done
  - One solution: lower ε over time
  - Another solution: exploration functions

Exploration Functions

- When to explore
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established

- Exploration function
  - Takes a value estimate and a count, and returns an optimistic utility, e.g.
    \[ f(u, n) = u + k/n \]
    \[ Q_{t+1}(s, a) = r(s, a, s') + \gamma \max_{a'} Q_t(s', a') \]
    \[ Q_{t+1}(s, a) = r(s, a, s') + \gamma \max_{a'} f(Q_t(s', a'), N(s', a')) \]

Q-Learning

- Q-learning produces tables of q-values:

Q-Learning

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we’ll see it over and over again

The Story So Far: MDPs and RL

Things we know how to do:

- We can solve small MDPs exactly, offline
- We can estimate values V(π(s)) directly for a fixed policy π.
- We can estimate Q*(s,a) for the optimal policy while executing an exploration policy

Techniques:

- Value and policy iteration
- Temporal difference learning
- Q-learning
- Exploratory action selection

Example: Pacman

- Let’s say we discover through experience that this state is bad:
- In naïve q learning, we know nothing about this state or its q states:
- Or even this one!
Feature-Based Representations

- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1/(dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - ... etc.
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)

Linear Feature Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:
  \[ V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]
  \[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]
- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but be very different in value!

Function Approximation

- Q-learning with linear q-functions:
  \[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g. if something unexpectedly bad happens, disprefer all states with that state’s features
- Formal justification: online least squares

Example: Q-Pacman

- Q-learning with linear q-functions:
  \[ Q(s, a) = 4.0 f_{DOT}(s, a) - 1.0 f_{GUST}(s, a) \]
  \[ f_{DOT}(s, \text{NORTH}) = 0.5 \]
  \[ f_{GUST}(s, \text{NORTH}) = 1.0 \]
  \[ Q(s, a) = +1 \]
  \[ R(s, a, s') = -500 \]
  \[ \text{error} = -501 \]
  \[ w_{DOT} = 4.0 + \alpha [-501] 0.5 \]
  \[ w_{GUST} = -1.0 + \alpha [-501] 1.0 \]
  \[ Q(s, a) = 3.0 f_{DOT}(s, a) - 3.0 f_{GUST}(s, a) \]

Linear regression

- Given examples \((x_i, y_i)_{i=1}^n\)
  - Predict \(\hat{y}_{n+1}\) given a new point \(x_{n+1}\)

\[ \hat{y}_{n+1} = w_0 + w_1 x_{n+1} \]
Ordinary Least Squares (OLS)

Minimizing Error

\[ E(w) = \frac{1}{2} \sum \left( \sum_k f_k(x_i)w_k - y_i \right)^2 \]

\[ \frac{\partial E}{\partial w_k} = \sum_i \left( \sum_k f_k(x_i)w_k - y_i \right) f_m(x_i) \]

\[ E \leftarrow E + \alpha \sum_i \left( \sum_k f_k(x_i)w_k - y_i \right) f_m(x_i) \]

Value update explained:

\[ w_i \leftarrow w_i + \alpha \text{error} f_i(s,a) \]

Policy Search

Policy Search

• Problem: often the feature-based policies that work well aren’t the ones that approximate V / Q best
  • E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  • We’ll see this distinction between modeling and prediction again later in the course

• Solution: learn the policy that maximizes rewards rather than the value that predicts rewards
  • This is the idea behind policy search, such as what controlled the upside-down helicopter

Policy Search

• Simplest policy search:
  • Start with an initial linear value function or q-function
  • Nudge each feature weight up and down and see if your policy is better than before

• Problems:
  • How do we tell the policy got better?
  • Need to run many sample episodes!
  • If there are a lot of features, this can be impractical