Reinforcement Learning

CSC 548, Artificial Intelligence II

Reinforcement Learning

- Basic idea:
  - Receive feedback in the form of rewards
  - The agent's utility is defined by the reward function
  - The agent must learn to act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes

Reinforcement Learning

- Assume a Markov decision process (MDP)
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model $T(s, a, s')$
  - A reward function $R(s, a, s')$
- Goal is to find a policy $\pi(s)$
- The twist: we do not know T or R
  - That is, we do not know which states are good or what the actions do
  - Need to actually try actions to learn

Model-Based Learning

- Model-based idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model is correct
- Step 1: Learn empirical MDP model
  - Count outcomes s' for each s and a
  - Normalize to give an estimate of $\hat{T}(s, a, s')$
  - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')
- Step 2: Solve the learned MDP
  - For example, use value iteration

Example: Expected Age

- Goal compute the expected age of
- With know $P(A)$, $E[A] = \sum_a P(a) \cdot a$
- Without $P(A)$, we collect samples $[a_1, a_2, \ldots, a_N]$
  - Unknown $P(A)$: "model based"

$$\hat{P}(a) = \frac{num(a)}{N}$$
$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

  - Unknown $P(A)$: "model free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - You do not know the transitions $T(s, a, s')$
  - You do not know the rewards $R(s, a, s')$
  - Goal: learn the state values
- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning – you actually take actions in the world

Direct Evaluation

- Goal: compute values for each state under π
- Idea: average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples
- This is called direct evaluation

Problems with Direct Evaluation

- What is good about direct evaluation?
  - It is easy to understand
  - It does not require any knowledge of T or R
  - It eventually computes the correct average values using sample transitions
- What is bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - It takes a long time to learn

Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate V for a fixed policy:
  - Each iteration, replace V with a one-step lookahead layer over V

$$V_0^\pi(s) = 0$$
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V_k^\pi(s')\right]$$

  - This approach fully exploits connections between states
  - Unfortunately, we need T and R to do it
- Key question: how can we do this update to V without knowing T and R?
  - That is, how do we take a weighted average without knowing the weights

Sample-Based Policy Evaluation?

- We want to improve our estimate of V by computing these averages:

$$V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^{\pi}_{k}(s')\right]$$

- Idea: take samples of outcomes s' (by performing the action) and average

$$\begin{aligned}
sample_1 &= R(s, \pi(s), s'_1) + \gamma V^{\pi}_{k}(s'_1) \\
sample_2 &= R(s, \pi(s), s'_2) + \gamma V^{\pi}_{k}(s'_2) \\
&\dots \\
sample_n &= R(s, \pi(s), s'_n) + \gamma V^{\pi}_{k}(s'_n) \\
V^{\pi}_{k+1}(s) &\leftarrow \frac{1}{n} \sum_{i} sample_i
\end{aligned}$$

Temporal Difference Learning

- Big idea: learn from every experience
  - Update $V(s)$ each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often
- Temporal difference learning of values
  - Policy is fixed, still doing evaluation
  - Move values toward value of whatever successor occurs: running average

$$\begin{aligned} \mathit{sample} &= R(s, \pi(s), s') + \gamma V^\pi(s') \\ V^\pi &\leftarrow (1-\alpha) V^\pi(s) + (\alpha)\ \mathit{sample} \\ &= V^\pi(s) + \alpha(\mathit{sampe} - V^\pi(s)) \end{aligned}$$

Exponential Moving Average

- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$
  - Makes recent samples more important

$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2) + \dots}$$

  - Forgets about the past (distant past values were wrong anyway)
- Decreasing the learning rate (alpha) can give converging averages

Problems with TD Value Learning

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- Problem: what if we want to turn values into a (new) policy?

$$\pi(s) = \arg\max_a Q(s,a)$$
$$Q(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]$$

- Idea: learn Q-values, not values
- Makes action selection model-free too

Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You do not know the transitions $T(s, a, s')$
  - You do not know the rewards $R(s, a, s')$
  - You choose the actions
  - Goal: learn the optimal policy / values
- In this case:
  - Learner make choices
  - Fundamental tradeoff: exploration versus exploitation
  - This is NOT offline planning – you take actions in the world and find out what happens

Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$ which we know is right
  - Given V_k calculate the depth $k+1$ values for all states

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- But, Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given Q_k calculate the depth $k+1$ q-values for all q-states

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

Q-Learning

- Q-learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Learn $Q(s,a)$ values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider you new sample estimate:

$$\mathit{sample} = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into the running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[\mathit{sample}]$$

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy – even if you are acting suboptimally
- This is called off-policy learning
- Caveats:
  - You need to explore enough
  - You have to eventually make the learning rate small enough
  - But, not decrease it too quickly
  - Basically, in the limit, it does not matter how you select actions

How to Explore?

- Several schemes for forcing exploration
  - Simplest: random actions (ϵ-greedy)
    - Every time step, flip a coin
    - With (small) probability ϵ, act randomly
    - With (large) probability $1 - \epsilon$, act on current policy
  - Problems with random actions
    - You do eventually 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, eventually stop exploring
- Exploration function
  - Takes a value estimate u and a visit count n and returns an optimistic utility, for example, $f(u, n) = u + \frac{k}{n}$
    - Regular Q-update:

$$Q(s, a) \leftarrow_{\alpha} R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

    - Modified Q-update:

$$Q(s, a) \leftarrow_{\alpha} R(s, a, s') + \gamma \max_{a'} f\left(Q(s', a'), N(s', a')\right)$$

  - Note: this propagates the "bonus" back to states that lead to unknown states as well

Regret

- Even if you learn the optimal policy, you still make mistakes along the way
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret

Generalizing Across States

- Basic Q-learning keeps a table of all q-values
- 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 the experience to new, similar situations
  - This is a fundamental idea in machine learning, and we will see it over and over again

Feature-Based Representations

- Idea: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers that capture important properties of the state
  - We can also describe a q-state (s, a) with features

Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

  $$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) Q(s, a) \quad = w_1 f_1(s, a) + w_2$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value

Approximate Q-Learning

- Q-learning with linear Q-functions:
  - transition $= (s, a, s', r)$
  - difference $= [r + \gamma \max_{a'} Q(s', a')] - Q(s, a)$
  - update

$$Q(s, a) \leftarrow Q(s, a) + \alpha \, [difference]$$

$$w_i leftarrow w_i + \alpha \, [difference] \, f_i(s, a)$$

- Intuitive interpretation:
  - Adjust weights of active features
  - For example, if something unexpectedly bad happens, blame the features that were on – disprefer all states with that state's features
- Formal justification: online least squares

Policy Search

- Problem: often the feature-based policies that work well (win games / maximize utilities) are not the ones that approximate V / Q best
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values correct (prediction)
  - We will see this distinction between modeling and prediction again later in the course
- Solution: learn policies that maximize rewards, not the values that predict them
- Policy search: start with an decent solution then fine-tune it by hill climbing on feature weights

Policy Search

- Simplest policy search:
  - Start with an initial linear value function of 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
  - With a lot of features, this can be impractical
- Better methods exploit lookahead structure, sample wisely, change multiple parameters . . .