Tufts

Class #23: Solving MDPs & Reinforcement Learning

Machine Learning (COMP 135): M. Allen, 25 Nov. 19

1

Review: Almost-Greedy Policies

- One simple way to add exploration is to use a policy that is ***mostly*** greedy, but ***not always***
- An "epsilon-greedy" (ε-greedy) policy sets some probability threshold, ε, and chooses actions by:
  1. Picking a random number $R \in [0, 1]$
  2. If $R \leq \varepsilon$, choosing the action ***at random***
  3. If $R > \varepsilon$, acting in a ***greedy*** fashion (as before)

2

Review: TD-Learning

function TD-LEARNING(mdp) **returns a policy**
  inputs: mdp, an MDP

  $\forall s \in S, U(s) = 0$
  repeat for each episode E:
    set start-state $s \leftarrow s_0$
    repeat for each time-step t of episode E, until s is terminal:
      choose action a, using ϵ-greedy policy based on $U(s)$
      observe next state s', one-step reward r
      $U(s) \leftarrow U(s) + \alpha[r + \gamma U(s') - U(s)]$
      $s \leftarrow s'$

  return policy π, set greedily for every state $s \in S$, based upon $U(s)$

- Algorithm is the same, but ***explores*** using sometimes-greedy and sometimes-probabilistic action-choices instead of fixed policy π
  - We reduce learning parameter α just as before to converge

3

Randomness and Weighting in Learning

- Our algorithm uses two parameters, α and ε (plus the usual discount factor γ), to control its overall behavior
- Each can be adapted over time to control algorithm

1. ε: the amount of randomness in the policy
   - When we don't know much, set it to a ***high value***, so that we start off with lots of random exploration
   - We ***reduce*** this value over time until ε = 0, and we are being purely greedy, and just exploiting what he have learned
2. α: the weight on each learning-update step
   - Reduce this over time, as well: when α = 0, U-values don't change anymore, and we can converge on final policy values

4

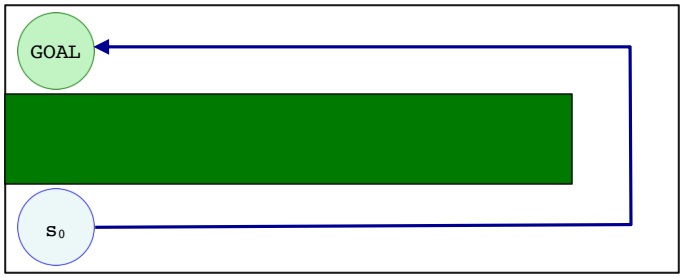
Randomness and Weighting in Learning

- The control parameters α and ε give us simple ways to control complex learning behavior
- We ***don't always*** want to reduce each over time
- In a purely ***stationary environment***, where system dynamics don't ever change, and all probabilities stay the same, we can simply slowly reduce each until we converge upon a stable learned behavior
- In a ***non-stationary*** environment, where things may change at some point, learned solutions may quit working

5

Non-Stationary Environments

- Suppose environment starts off in one configuration:



- Over time, we can learn a policy for shortest path to goal
- By letting ε and α go to 0, the policy becomes stable

6

Non-Stationary Environments

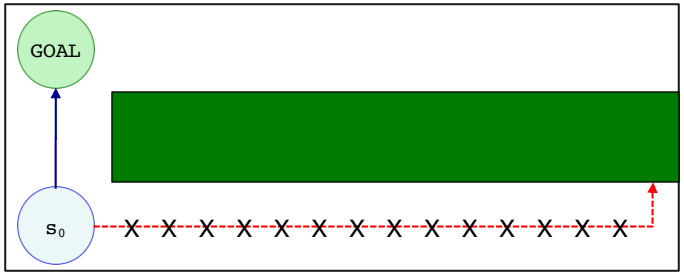
- The environment may change, however:

- If ε and α ***stay at*** 0, policy is ***sub-optimal*** from now on

7

Non-Stationary Environments

- We may be able to tell that environment changes, however



- If value drops off over a long time, we can ***increase*** ε and α again, to resume learning and find new optimal policy

8

Bellman Equations for *Q*-values

- Instead of the value of a ***state*** $U(s)$, we can calculate the value of a ***state-action pair*** $Q(s, a)$
- The value of taking action a in state s, and then following the policy π after that:

$$Q^{\pi}(s,a) = \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma\, Q^{\pi}(s', \pi(s'))\right]$$

- Similarly, we calculate optimal values $Q^*(s, a)$ of taking a in state s, then following ***best*** possible policy after that:

$$Q^{\star}(s,a) = \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^{\star}(s', a')\right]$$

- We can do learning for Q-values, too…

TD (*SARSA*) Learning for Q-values

function SARSA-LEARNING(mdp) **returns a policy**
  inputs: mdp, an MDP

  $\forall s \in S, \forall a \in A, Q(s,a) = 0$
  repeat for each episode E:
    set start-state $s \leftarrow s_0$
    set action a, chosen ϵ-greedily based on $Q(s,a)$
    repeat for each time-step t of episode E, until s is terminal:
      take action a
      observe next state s', one-step reward r
      set next action a', chosen ϵ-greedily based on $Q(s',a')$
      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma\, Q(s',a') - Q(s,a)]$
      $s \leftarrow s'$; $a \leftarrow a'$

  return policy π, set greedily for every state $s \in S$, based upon $Q(s,a)$

- Same basic RL method, converging to optimal Q^*
- Called ***SARSA***, due to information used (s, a, r, s', a')

On-Policy Updates

function SARSA-LEARNING(mdp) **returns a policy**
  inputs: mdp, an MDP

  $\forall s \in S, \forall a \in A, Q(s,a) = 0$
  repeat for each episode E:
    set start-state $s \leftarrow s_0$
    set action a, chosen ϵ-greedily based on $Q(s,a)$
    repeat for each time-step t of episode E, until s is terminal:
      take action a
      observe next state s', one-step reward r
      set next action a', chosen ϵ-greedily based on $Q(s',a')$
      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma\, Q(s',a') - Q(s,a)]$
      $s \leftarrow s'$; $a \leftarrow a'$

  return policy π, set greedily for every state $s \in S$, based upon $Q(s,a)$

- Both basic TD and SARSA are on-policy learning/update methods
- We choose our initial action (a) based on current ε-greedy policy

On-Policy Updates

function SARSA-LEARNING(mdp) **returns a policy**
  inputs: mdp, an MDP

  $\forall s \in S, \forall a \in A, Q(s,a) = 0$
  repeat for each episode E:
    set start-state $s \leftarrow s_0$
    set action a, chosen ϵ-greedily based on $Q(s,a)$
    repeat for each time-step t of episode E, until s is terminal:
      take action a
      observe next state s', one-step reward r
      set next action a', chosen ϵ-greedily based on $Q(s',a')$
      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma\, Q(s',a') - Q(s,a)]$
      $s \leftarrow s'$; $a \leftarrow a'$

  return policy π, set greedily for every state $s \in S$, based upon $Q(s,a)$

- When we do the value update, we ***also*** choose the next action (a') based on the same current ε-greedy policy

The Effect of On-Policy Updates

- When we do this sort of updating, we are ***not*** basing our value calculation on the ***best possible*** policy
- Instead, we are basing it on our ***learning policy***, which means the values that we base our updates and choices on will combine the values that we get from:
  1. greedy action selection for exploitation
  2. random actions in some states for exploration
- Values we learn can reflect what would happen in a state if we sometimes acted in a non-optimal way
  - For example, on the edge of a cliff, we will sometimes randomly explore jumping off the cliff when learning
  - Edge-states are thus risky, and get lower value than they would really have under the optimal policy (where we only do the best thing, and never jump)

13

Off-Policy Methods

- One possible solution is to update the values we learn based on the ***best actions only***
- That is, we ***ignore*** rewards and outcomes that come from any of the possible bad actions we take when exploring
- The policy being updated is then ***not*** the current learning version, but the optimal one
  - This is the policy that we wanted to learn in the end, anyway!
- How can we do this?

14

Q-Learning: Off-Policy Updates

function Q-LEARNING(mdp) **returns a policy**
inputs: mdp, an MDP

$\forall s \in S, \forall a \in A, Q(s,a) = 0$
repeat for each episode E:
  set start-state $s \leftarrow s_0$
  repeat for each time-step t of episode E, until s is terminal:
    set action a, chosen ϵ-greedily based on $Q(s,a)$
    take action a
    observe next state s', one-step reward r
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$

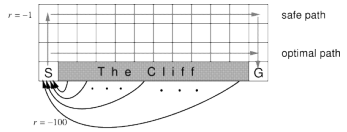
return policy π, set greedily for every state $s \in S$, based upon $Q(s,a)$

- We still choose actions (a) in an ε-greedy way (so we ***are*** sometimes random)
- However, we update values based upon whatever action would actually be ***best***

15

Comparing the Methods: Cliff Problem

- Shortest path to the goal goes along edge of a cliff
  - SARSA learns safer path, since edge-states get lower values due to random falls
  - Q-Learning learns best path, since it ***ignores*** random jumps off edge
  - Why does QL do worse in the end? How can we fix this over a period of time?



Example from: Sutton & Barto, 1998

16

Unifying the Methods

- Both SARSA and Q-Learning can be made to converge to the same optimal policy over time
- By reducing the epsilon-value in our ε-greedy policy, we eventually reduce the randomness
- Thus, the SARSA agent will eventually learn better values even for risky states, and come to use the optimal policy, too (e.g. walking along the cliff's edge)
- So what's the difference?
  - In many cases, Q-Learning can converge on values somewhat ***faster***
  - Doesn't have to spend time "fixing" the values of states where it has ***over-estimated*** negative risk
  - Thus we can reduce the ε-value more rapidly, and learn optimal state-values more quickly
  - What are the potential risks of doing this?

17

Extending the Value Update Procedure

- Basic reinforcement learning algorithms update the value of a ***single state*** (or single state-action pair) at a time
  - Repeated occurrences of the same state sequences eventually cause observed utility value to "spread out" over time, so that states that tend to lead to particularly good (or bad) states down the road also get higher (or lower) values
- Sometimes, we can speed up learning by ***directly*** implementing the process of spreading values over time

18

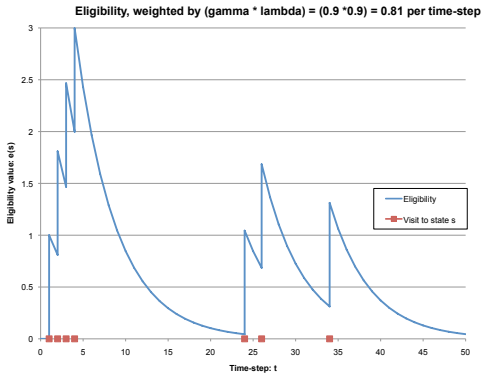
Eligibility Traces

- For each state, s, we set aside some extra memory, $e(s)$, to keep track of ***how recently*** we visited it
  - Each time we visit a state, we ***increase*** this value
  - We can choose a ***lambda*** value, $0 \leq \lambda \leq 1$, to control how quickly the "eligibility" variable $e(s)$ decreases over time
- At each time step t, we update the eligibility for all states, including the one we are currently visiting, s_t:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

19

Eligibility Traces

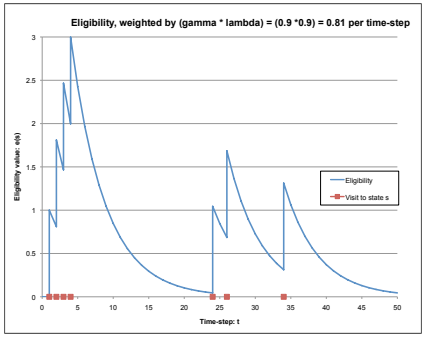


- The eligibility trace grows each time a state is visited, and then "decays" towards 0 over time, until that state is visited again

20

Eligibility Traces

- We then take the TD error—the difference between values of the current state and the last state, plus current reward—and ***spread this out*** over all the past states, ***in proportion*** to their eligibility values:



$$\delta_t = r_{t+1} + \gamma\, Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

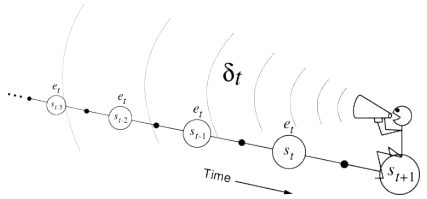
21

SARSA-λ: Learning with Eligibility Traces

function SARSA-λ (mdp) **returns a policy**
  inputs: mdp, an MDP, and π, a policy to be evaluated
  $\forall s \in S, \forall a \in A, Q(s,a) = 0$ **and** $e(s,a) = 0$
  repeat for each episode E:
    set start-state and action $s \leftarrow s_0$, $a \leftarrow \max_a Q(s_0, a)$
    repeat for each time-step t of episode E, until s is terminal:
      take action a
      observe next state s', one-step reward r
      choose next action a', chosen ϵ-greedily based on $Q(s', a')$
      $\delta \leftarrow r + \gamma\, Q(s', a') - Q(s, a)$
      $e(s,a) \leftarrow e(s,a) + 1$
      $\forall s \in S, \forall a \in A$:
        $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
        $e(s,a) \leftarrow \gamma\lambda e(s,a)$
      $s \leftarrow s', a \leftarrow a'$
  return policy π, set greedily for every state and action based upon Q-values

22

Using Eligibility Traces

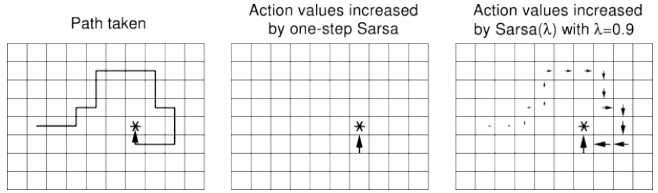


- The algorithm sends the TD-error back over prior time-steps, with a "distance" that is affected by the choice of λ
- If we choose λ = 0, the algorithm is simply doing a ***single-state backup*** as before
- As λ nears (but never equals) 1, the algorithm pushes the updates further and further back in time

Diagram taken from: Sutton & Barto, 1998

23

Possible Advantages of Using Traces



- In many cases, using Eligibility Traces can speed the learning process, since whole chains of states can be updated at once
- This works especially well for path-planning problems, and things similar to it, since the value updates can affect multiple locations along the path to a goal at the same time

Example from: Sutton & Barto, 1998

24

After Break

- **HW 06**: due Monday, 09 December, 9:00 AM
- **Project 03**: due Monday, 16 December, 9:00 AM
- **Final Exam**: Thursday, 12 December, 7:00 PM – 9:00 PM
  - Usual classroom
  - Practice exam to be posted by 05 December
- **Office Hours**: 237 Halligan, Tuesday, 11:00 AM – 1:00 PM
  - Hours over exam period will be announced later