MDPs and Reinforcement Learning

15-381 lecture notes, 18 April 2000

David Cohn <www.cs.cmu.edu/~cohn>

- By the end of this lecture, you should be very comfortable with the following terms
  - "certainty equivalence"
  - "model free" learning
  - "TD(λ)"
  - "Q-learning"

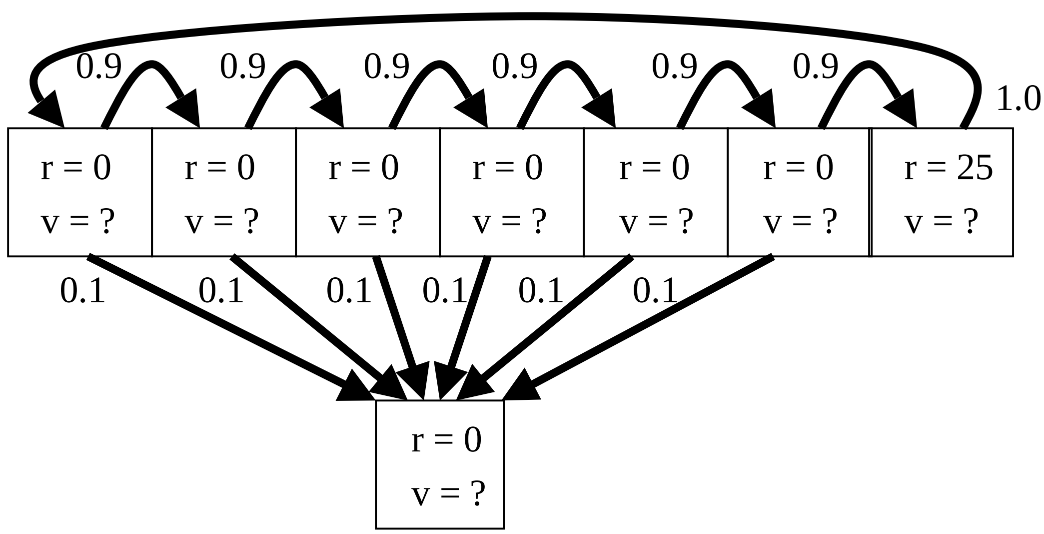
What makes a process “Markov”?

- Important definition: a system has the ***Markov*** property if its transition probabilities and rewards depend only on its current state
  - history, prior states, number of steps taken, etc. don’t matter
  - current state (and action) completely defines probabilities of what will happen next
- Example: game of chess
  - if state is chessboard layout and whose move it is?
- How about an inverted pendulum
  - if state is position and angle of pole?
  - why not?

Markov processes

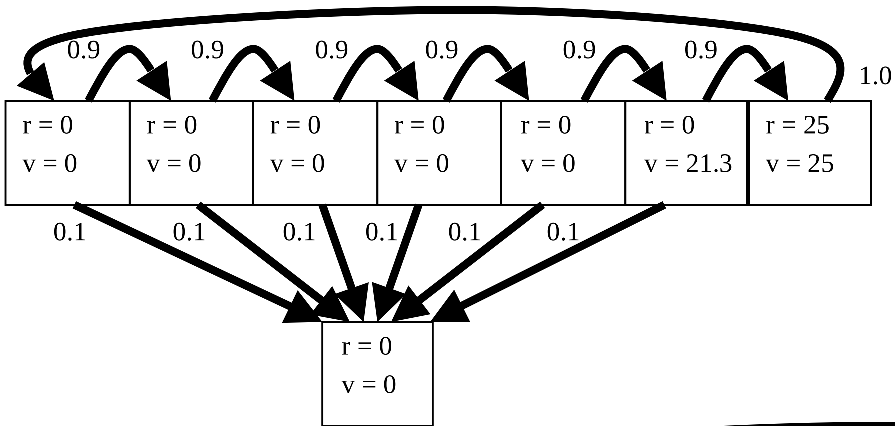
- You've been abducted by aliens
  - each day they run experiments on you
  - at the end of each day, they decide whether to send you back or keep you for another day of experiments
  - at the end of 7 days of successful experiments, they direct deposit \$0.25 into your student bank account, and start a new series of tests
- Assuming a discount factor of $\gamma = 0.95$, what's your expected future sum of discounted rewards on each day?
- Dynamic programming with Bellman updates

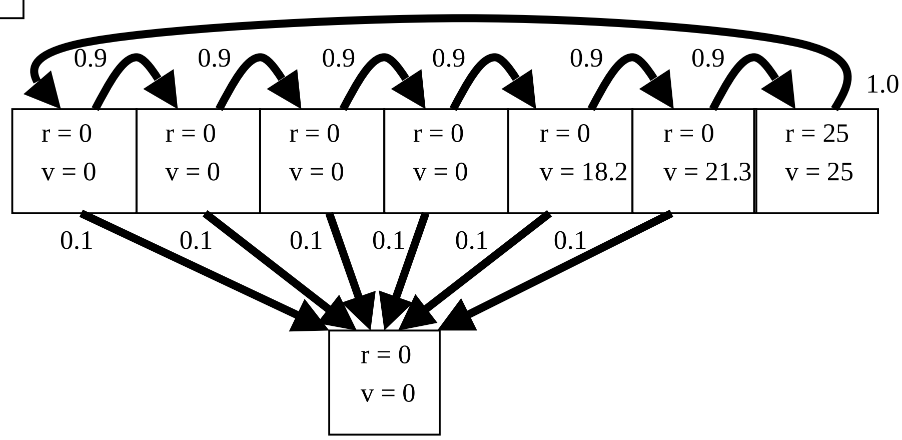
$$J^*(s_i) = r_i + \gamma \Sigma_j\, p_{ij}\, J^*(s_j)$$



Markov processes

- $J^*(s_i) = r_i + \gamma \Sigma_j \, p_{ij} \, J^*(s_j)$

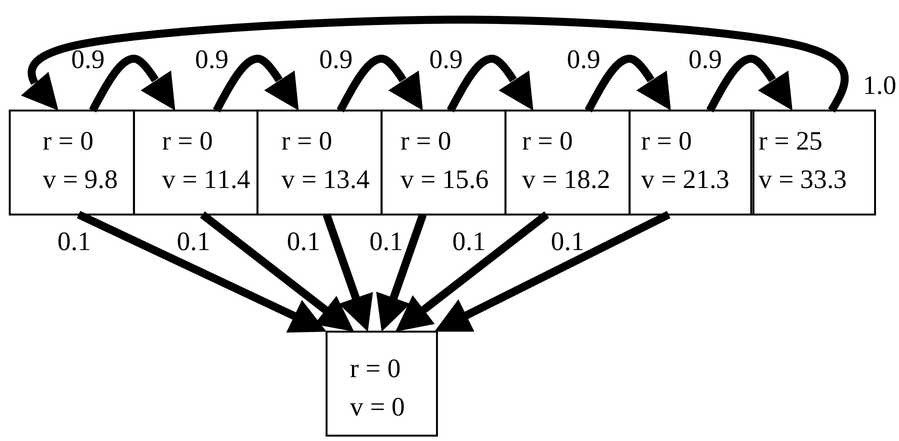




…

…

…



Markov decision processes

- You’ve been abducted by aliens (again!?!?)
  - each day they put you somewhere in a maze
  - you wander until you find a piece of cheese (oddly, worth exactly $0.25)
  - you’re then picked up and placed at another random spot in the maze
  - you’re wearing rollerblades (you were trying to learn how to use them when you were abducted). As a result, your actions are nondeterministic. when you try to go in a particular direction,
    - you have an 80% chance of actually getting there
    - you have a 15% chance of going in another direction
    - you have a 5% chance of staying right where you are
- Assuming $\gamma = 0.95$, what’s your expected future sum of discounted rewards at each step?
- Dynamic programming with Bellman updates:

  $$J^*(s_i) = \max_a [\, r_i + \gamma \textstyle\sum_j p^a_{ij} J^*(s_j) \,]$$

0	0	0	0	0	0
0				0	0
0	0	0	0	0	25

Markov decision processes

- *Modeling the alien cheese maze...*

0	0	0	0	0	0
0				0	0
0	0	0	0	0	25

0	0	0	0	0	0
0				0	19
0	0	0	0	19	25

0	0	0	0	0	14.44
0				15.3	19.9
0	0	0	14.44	19.9	27.58

Dynamic programming backups

- Backing up state G
  - immediate reward (r_i) is 25
  - p_{ij} to other states is uniform:

$$J^*(s_i) = \max_a \left[r_i + \gamma \cdot p_{ij}^a J^*(s_j) \right]$$
$$= r_i + \gamma \cdot p_{ij}^{do\_nothing} J^*(s_j)$$
$$= 25 + 0.95 \cdot \left(\frac{2}{14} 19.9 + \frac{2}{14} 14.44 + \frac{1}{14} 15.3 + \frac{9}{14} 0 \right)$$
$$= 30.70$$

0	0	0	0	0	14.44
0				15.3	19.9
0	0	0	14.44	**X** 19.9	**G** 27.58

Dynamic programming backups

- Backing up state X

$$J^*(x) = \max_a \left[r_x + \gamma \cdot \sum_j p_{xj}^a J^*(s_j) \right]$$

  – compute value given each possible action

$$r_i + \gamma \cdot p_{ij}^{go\_right} J^*(s_j) = 0 + 0.95 \cdot (0.8 \cdot 30.70 + 0.1 \cdot 19.9 + 0.05 \cdot 15.3 + 0.05 \cdot 14.4)$$
$$= 26.63$$

$$r_i + \gamma \cdot p_{ij}^{go\_up} J^*(s_j) = 0 + 0.95 \cdot (0.05 \cdot 30.70 + 0.1 \cdot 19.9 + 0.8 \cdot 15.3 + 0.05 \cdot 14.4)$$
$$= 15.63$$

$$r_i + \gamma \cdot p_{ij}^{go\_left} J^*(s_j) = 0 + 0.95 \cdot (0.05 \cdot 30.70 + 0.1 \cdot 19.9 + 0.05 \cdot 15.3 + 0.8 \cdot 14.4)$$
$$= 15.01$$

$$r_i + \gamma \cdot p_{ij}^{go\_down} J^*(s_j) = 0 + 0.95 \cdot (0.05 \cdot 30.70 + 0.85 \cdot 19.9 + 0.05 \cdot 15.3 + 0.05 \cdot 14.4)$$
$$= 18.94$$

  – value of state is max of value given each possible action

0	0	0	0	0	14.44
0				15.3	19.9
0	0	0	14.44	**X** 19.9	**G** 30.70

result of *go_right* action

Reinforcement learning

- So far, we've assumed we have a perfect model of the Markov decision process
- What if we don't?
  - observe that we're in state s_{27}
  - take action a_3, find ourselves in state s_{32}, and receive a dollar
  - take action a_{16}, find ourselves in state s_{44}, and receive nothing
  - take action a_8, find ourselves in state s_7, and receive nothing...
- How can we estimate the expected sum of future discounted rewards (i.e. *value*) of being in state s?
- How can we learn the optimal action to take in each state (i.e., an *optimal policy*)?

- **This is *reinforcement learning* - the problem of learning from delayed rewards**

Reinforcement learning in a discounted Markov system

- [We'll ignore free will for the moment, and work with a Markov system without decisions]
- You wander through life, observing the following transitions
  $s_1(r=0) \Rightarrow s_2(r=0) \Rightarrow s_3(r=4) \Rightarrow s_2(r=0) \Rightarrow s_4(r=0) \Rightarrow s_5(r=0)$

s_1 $\hat{r} = 0$	s_2 $\hat{r} = 0$	s_3 $\hat{r} = 4$
s_4 $\hat{r} = 0$	s_5 $\hat{r} = 0$	

- One approach: use statistics of trajectory to estimate transition probabilities
- This is called *certainty equivalence* learning

Certainty equivalence learning

- Observation: in the limit, observed rewards and transition probabilities converge to true values
- So given a long enough trajectory, certainty equivalence will converge to the correct estimates
- How to compute certainty equivalence?
  - full backups:
    - keep full set of statistics for each state - # times in state, total reward for state, # times transitioned to each next state
    - whenever get a new observation, solve for new statistics to compute new $\hat{J}(s_i)$ for all states
  - advantages:
    - estimates will always make full use of data
  - disadvantages:
    - use space $O(|S|^2)$
    - might use time up to $O(|S|^3)$ on each step!

Certainty equivalence learning

- Another alternative: single backups
  - keep full set of statistics for each state - # times in state, total reward for state, # times transitioned to each next state
  - whenever get new observation $s_i(r) \Rightarrow s_j$, just perform one backup, computing
    $$\hat{J}(s_i) = \hat{r}_i + \gamma \cdot \sum_j \hat{p}_{ij} \hat{J}(s_j)$$
  - advantages:
    - computationally cheap: O(1)
  - disadvantages:
    - still uses space $O(|S|^2)$
    - wastes a lot of data - estimates won't be as accurate for a given number of steps
- Yet another alternative: *prioritized sweeping* [Moore & Atkeson, '93]
  - on each transition, back up a number of "most deserving" states
  - kept on priority queue of which states have the biggest potential change in their values of $\hat{J}(s_i)$
  - nearly as data efficient as full backups, not much more expensive than single backups

Beyond certainty equivalence

- So far, learning value of state *s* by building a model of Markov process it's a part of
  - certainty equivalence guarantees that we'll converge to right answer, but we're using an awful lot of memory:
    - we want to estimate $|S|$ numbers - why should we have to maintain a table of size $O(|S|^2)$?

- *Temporal difference* learning [Sutton '88]
  - only maintain table of $\hat{J}(s_i)$
  - when observe a transition $s_i(r) \Rightarrow s_j$, update $\hat{J}(s_i)$ by nudging it towards expected future rewards:

$$\begin{aligned}\hat{J}(s_i) &= (1-\alpha)\hat{J}(s_i) + \alpha[\text{future expected rewards}] \\ &= (1-\alpha)\hat{J}(s_i) + \alpha\left[r + \gamma \cdot \hat{J}(s_j)\right]\end{aligned}$$

  - α is a learning rate, just like with backprop

- This is ***model-free*** reinforcement learning
  - we're not building a model of the environment or transition probabilities

Temporal difference (TD) learning

- The great bit about TD:
  analysis shows that, as $t \to \infty$, $E\left[\hat{J}(s_i)\right] \to J^*(s_i)$
- The not so great thing about TD:
  as $t \to \infty$, $E\left[\left(\hat{J}(s_i) - J^*(s_i)\right)^2\right] \to \dfrac{\alpha\sigma^2}{(2-\alpha)}$
  where $\sigma^2 =$ variance of reward $= E\left[\left(r_i - J^*(s_i)\right)^2\right]$
- Why does this make us sad?
- What can we do about it?
  – pick a tiny learning rate (boo - will learn too slowly!)
  – start with a large learning rate and let it decay to zero (will this work?!?)

Decaying learning rates for TD

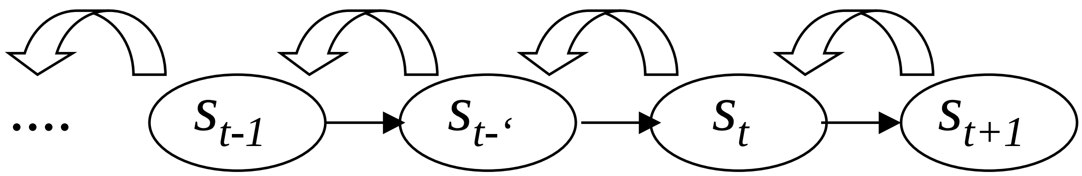
- Peter Dayan showed that decaying α will converge to J^* **if**
  - each state visited infinitely often
  - sum over time $\Sigma_t \, \alpha_t = \infty$
  - sum over time $\Sigma_t \, \alpha_t^2 < \infty$

- What sorts of learning rates work?
  - works: $\alpha_t = 1/t$, $\alpha_t = \beta/(\beta + t)$ (e.g. β=1000)
  - doesn't: $\alpha_t = \alpha_0$, $\alpha_t = \beta \, \alpha_{t-1}$ (e.g. β=0.9)

TD(λ) - getting more mileage per transition

- Recall the data efficiency argument for certainty equivalence: single backups were data inefficient

$$\hat{J}(s_t) = (1-\alpha)\hat{J}(s_t) + \alpha\left(r + \gamma \cdot \hat{J}(s_{t+1})\right)$$



  - what if we kept a list of recent transitions?
  - whenever we did a backup of some state, we could backup the previous state, and the previous state...
- How much to back up?
  - further back in time, less effect
  - control with parameter $0 \leq \lambda \leq 1$

$$\hat{J}(s_{t-u}) = \left(1-(\gamma\lambda)^u \alpha_t\right)\hat{J}(s_{t-u}) + (\gamma\lambda)^u \alpha_t\left(r + \gamma \cdot \hat{J}(s_{t+1})\right)$$

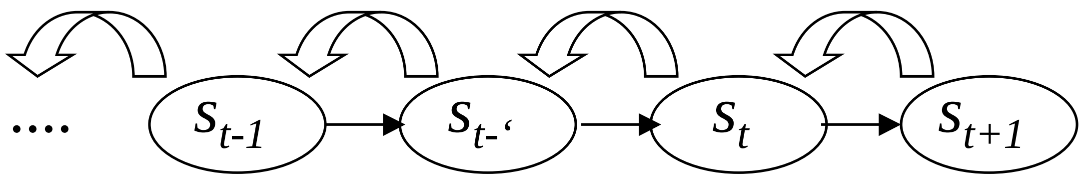
for all u in which $(\gamma\lambda)^u > \varepsilon$

TD(λ) - some typical values for λ

- What sort of behavior would you expect from λ=0?

$$\hat{J}(s_{t-u}) = \left(1-(\gamma\lambda)^u \alpha_t\right)\hat{J}(s_{t-u}) + (\gamma\lambda)^u \alpha_t \left(r + \gamma \cdot \hat{J}(s_{t+1})\right)$$

- How about λ=0?
- Empirically, TD(λ=0.7) is often best empirically



Learning policies for MDPs

- Instead of being passive observers, need to act
- Actions affect rewards, future states
  - “You are in state 52, with three actions available..”
  - (take action 1)
  - “Reward is +5; you are now in state 37, with two actions available...”
  - (take action 2)
  - “Reward is -6; you are now in state 52 (again) with three actions available...”
- Markov property means that once you’ve selected an action, the PDF of your next state is the same as the last time you tried that action in that state

- The goal: learn an optimal policy
- The difficult thing about learning a policy: the *Credit Assignment Problem*
  - after a long game of chess, receive reward (we won!)
  - but which moves actually contributed to winning?

Learning policies for MDPs

- Can use same certainty equivalence to build model, just as we did for Markov systems without actions
  - full backup
  - single backups
  - prioritized sweeping
- Compute state-action-transition and state-reward statistics, perform backups according to Bellman:

$$\hat{J}(s_i) = \max_a \left[\hat{r}_i + \gamma \cdot \sum_j \hat{p}_{ij}^a \hat{J}(s_j) \right]$$

- We're in state s_i, we can estimate r_i and p^a_{ij} and $J(s_j)$
  - so how should we choose actions?
- Idea #1: $a = \arg\max_{a'} \left[\hat{r}_i + \gamma \cdot \sum_j \hat{p}_{ij}^{a'} \hat{J}(s_j) \right]$
- Idea #2: $a = random$
- What's the problem with each of these?
- How can we do better?
  - very difficult problem, subject of much research
  - recent algorithm by Singh and Kearns '99 finds "optimal" tradeoff

Model-free learning in MDPs

- Just like with Markov systems, may not want to construct ponderous statistical model of entire world just to know how to act
- Idea #1: use TD learning
  - take action *a* in state s_i, observe transition to s_j
  - update $\hat{J}(s_i) = (1-\alpha)\hat{J}(s_i) + \alpha\left[r_i + \gamma \cdot \hat{J}(s_j)\right]$

- What's the problem with this?

- Idea #2: estimate value of state-action pair
  - called Q-learning [Watkins '88]
  - $Q^*(s_i,a)$ = expected sum of future discounted rewards
    - if start in state s_i
    - take action *a*
    - action optimally from then on

- Mental gymnastics:
  - define Q^* in terms of J^*
  - define J^* in terms of Q^*

Q-learning - the update rule

- True value of *Q** is

$$Q^*(s_i, a) = r_i + \gamma \sum_j p_{ij}^a J^*(s_j)$$
$$= r_i + \gamma \sum_j p_{ij}^a \max_{a'} Q^*(s_i, a)$$

- Q-learning update rule is just online version of this:

$$\hat{Q}(s_i, a) = (1-\alpha)\hat{Q}(s_i, a) + \alpha\left[r_i + \gamma \max_{a'} \hat{Q}(s_j, a')\right]$$

- Especially clever bit: Q-value estimates are not biased by exploration policy - avoids credit assignment problem

Q-learning - choosing actions

- Want to explore, so don't always choose *argmax*$_a$*Q(*s_i*,a)*
  - (will this converge? to what?)
- Don't want to wander aimlessly, so don't always choose *a* randomly
  - (will this converge? to what?)

- One possibility: Boltzmann exploration [Watkins]

$$p(a) \propto \exp\left(\frac{-\hat{Q}(s,a)}{k_t}\right)$$

- Another: optimism in the face of uncertainty [Sutton '90, Kaelbling '90]
  - initialize Q-values to unrealistically high values to encourage exploration
  - or prefer actions that have not been tried much

More resources

- Gerald Tesauro, Temporal Difference Learning and TD-Gammon, http://www.research.ibm.com/massive/tdl.html
- P. Dayan (1992) “The Convergence of TD-lambda for General lambda,” Machine Learning, vol. 8, no. 3/4, pp. 341-362.
- R. S. Sutton (1998) “Learning to Predict by the Methods of Temporal Differences", Machine Learning, vol. 3, pp. 9-44.