Introduction to

Artificial Intelligence (AI)

Computer Science cpsc502, Lecture 17

Nov, 8, 2011

Slide credit : C. Conati, S. Thrun, P. Norvig, Wikipedia

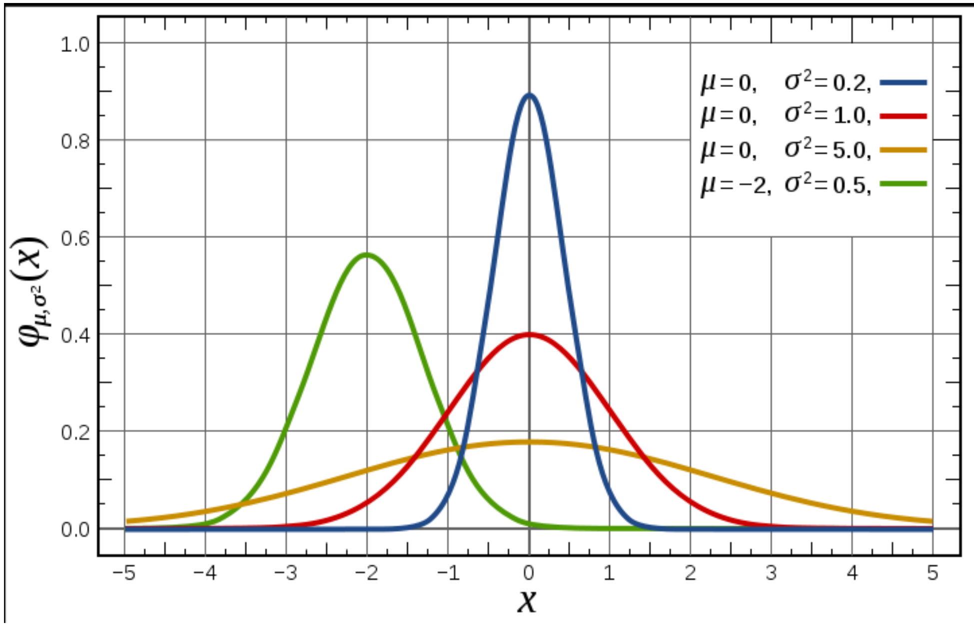
Today Nov 8

- **Unsupervised Machine Learning**
  - K-means
  - Intro to EM

- **Brief Intro to Reinforcement Learning (RL)**
  - Q-learning

Gaussian Distribution

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



- Models a large number of phenomena encountered in practice
- Under mild conditions the sum of a large number of random variables is distributed approximately normally ?

Gaussian Learning: Parameters

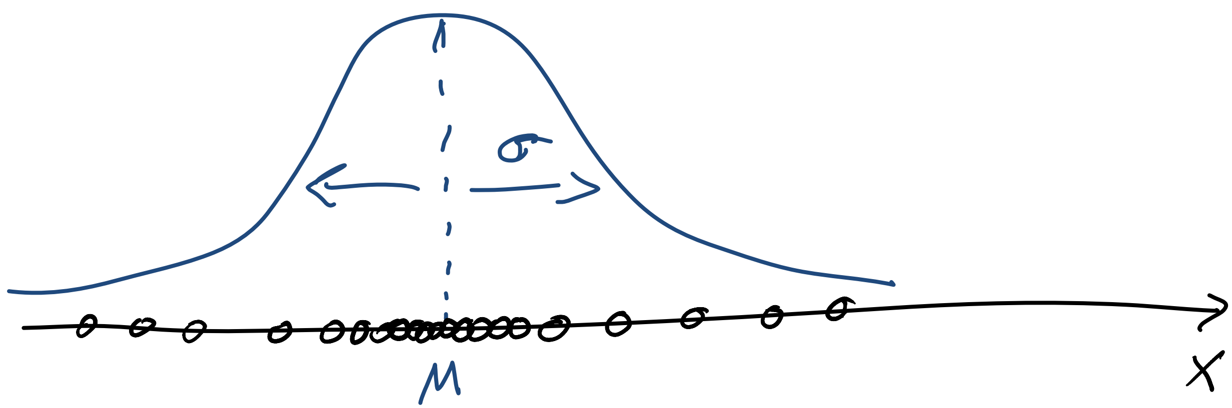
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

average

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i-\mu)^2$$

average deviation



- n data points

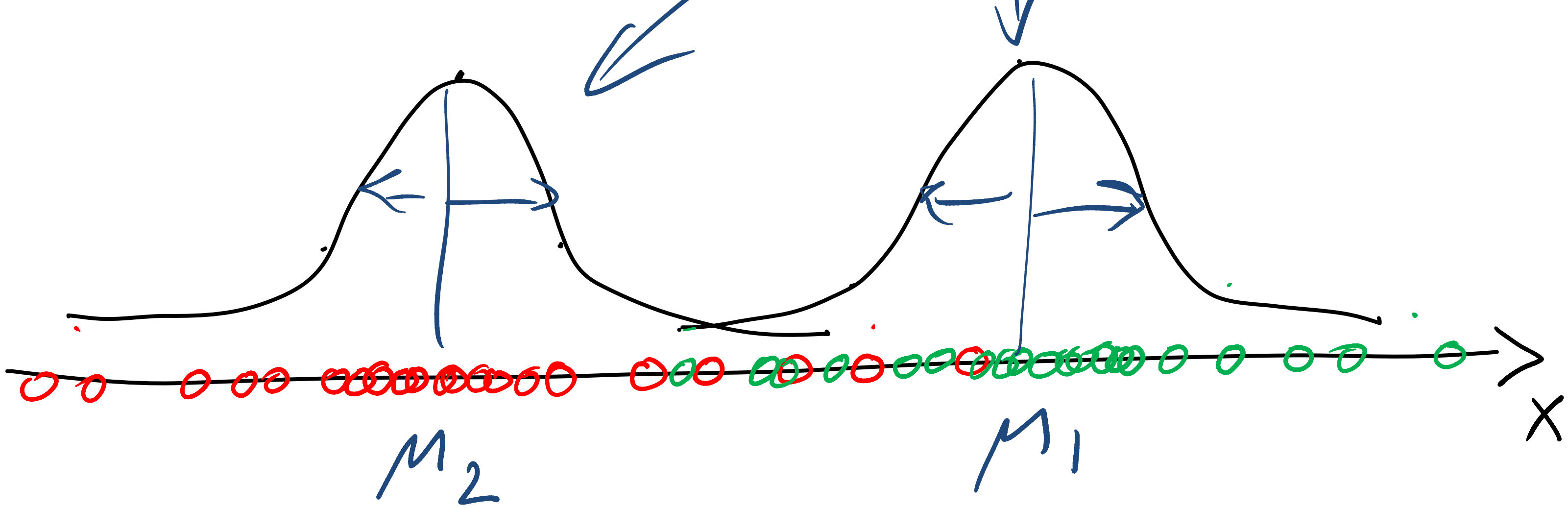
Expectation Maximization for Clustering: Idea

- **Lets assume:** that our Data were generated from several Gaussians (a mixture, technically)
- **For simplicity** – one dimensional data – only two Gaussians (with same variance, but possibly different ………. means)

σ^2

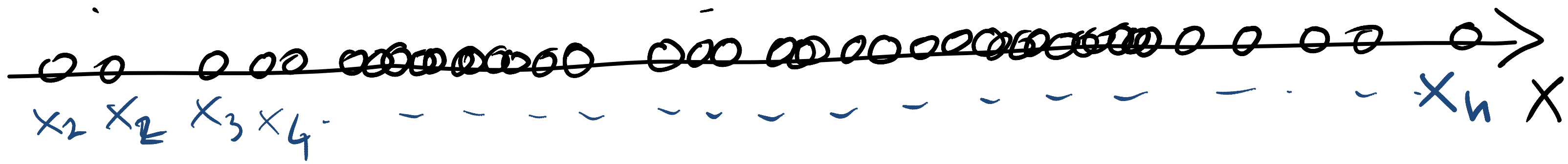
- **Generation Process**
  - Gaussian/Cluster is selected
  - Data point is sampled from that cluster

θ_2 .8 θ_1 .2



But this is what we start from

- n data points without labels! And we have to cluster them into two (soft) clusters.



- “Identify the two Gaussians that best explain the data”
- Since we assume they have the same variance, we “just” need to find their priors and their means

- *In K-means we assume we know the center of the clusters and iterate…..*

Here we assume that we know

- Prior for clusters and the two means

θ_1	θ_2	μ_1	μ_2
.3	.7	10.5	30.7

- We can compute the probability that data point x_i corresponds to the cluster N_j

$$P(N_J \mid x_i) = \frac{P(N_J, x_i)}{P(x_i)}$$

E step

$$z_{ij} = \frac{\theta_j * N(x_i \mid \mu_j, \sigma)}{\boxed{\sum_{m=1}^{2} \theta_m * N(x_i \mid \mu_m, \sigma)}}$$

$$\boxed{N(x_i \mid \mu_j, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}$$

Z

	N_1	N_2
x_1	.9	.1
x_2	.82	.18
x_3	. .	. .
⋮		
x_n	.01	.99

M step

We can now recompute

- **Prior for clusters**

$$\theta_j = \frac{\sum_{i=1}^{n} z_{ij}}{n}$$

- **The means**

$$\mu_j = \frac{\sum_{i=1}^{n} z_{ij} x_i}{\sum_{i=1}^{n} z_{ij}}$$

$j = 1$

$$\boxed{\theta_1 = \frac{\sum_{i=1}^{n} z_{i1}}{n}}$$

$$n = \sum_i \sum_j z_{ij}$$

	N_1	N_2	val
x_1	.9	.1	−10.5
x_2	.82	.18	−7
x_3	.	.	−3.4
⋮			⋮
x_n	.01	.99	123.7

$$\mu_1 = \frac{\sum_{i=1}^{n} \boxed{z_{i1}} x_i}{\sum_{i=1}^{n} z_{i1}}$$

Hard cluster

$$\mu_1 = \frac{\sum \text{values of points in } N_1}{\#\text{ points in } N_1}$$

Expectation Maximization

$\theta_1 \; \theta_2$ $\quad \mu_1 \; \mu_2$

Converges! ☺

$P(D \mid \mu_1, \mu_2)$

Proof [Neal/Hinton, McLachlan/Krishnan]:

- E/M step does not decrease data likelihood

But does not assure optimal solution ☹

Practical EM

Number of Clusters unknown

Algorithm:

- **Guess initial # of clusters**
- **Run EM**
  - ✓**Kill cluster** center that doesn't contribute (two clusters with the same data)
  - ✓**Start new cluster** center if many points "unexplained" (uniform cluster distribution for lots of data points)

EM is a very general method!

- **Baum-Welch Algorithm** (also known as *forward-backward*): Learn HMMs from unlabeled data

- **Inside-Outside Algorithm:** unsupervised induction of probabilistic context-free grammars. NLP

- More generally, learn parameters for hidden variables in any Bnets (see textbook example 11.1.3 to learn parameters of Naïve-Bayes classifier)

Today Nov 8

- **Unsupervised Machine Learning**
  - K-means
  - Intro to EM

- **Brief Intro to Reinforcement Learning (RL)**
  - Q-learning

MDP and RL

- **Markov decision process**
  - Set of **states** S, set of **actions** A
  - **Transition** probabilities to next states P(s'| s, a′)
  - **Reward** functions R(s, s', a)
- **RL is based on MDPs, but**
  - Transition model is **not known**
  - Reward model is **not known**
- While for **MDPs** we can ***compute*** an optimal policy
- **RL** ***learns*** an optimal policy

Search-Based Approaches to RL

- **Policy Search (*evolutionary algorithm*)**
  a) Start with an arbitrary policy
  b) Try it out in the world (evaluate it)
  c) Improve it (stochastic local search)
  d) Repeat from (b) until happy

- **Problems with evolutionary algorithms**
  - **Policy space can be huge**: with n states and m actions there are m^n policies
  - **Policies are evaluated as a whole**: cannot directly take into account locally good/bad behaviors

Q-learning

- Contrary to search-based approaches**, Q-learning learns after every action**
- **Learns components of a policy**, rather than the policy itself
- *Q(a,s)* = expected value of doing action *a* in state *s* and then following the optimal policy

Discounted reward we have seen in MDPs

$$Q(s,a) \;=\; R(s)+\gamma\sum_{s'}P(s'|\,a,s)V^{\pi^*}(s')$$

reward in *s*

states reachable from s by doing a

Probability of getting to *s'* from *s* via *a*

expected value of following optimal policy π in s'

Q values

$$Q(s,a) \;=\; R(s) + \gamma \sum_{s'} P(s' \mid s,a) V^{\pi^*}(s') \qquad (1)$$

- *Q(s,a)* are known as Q-values, and are related to the utility of state *s* as follows

$$V^{\pi^*}(s) = \max_{a} Q(s,a) \qquad (2)$$

- From (1) and (2) we obtain a constraint between the *Q* value in state *s* and the *Q* value of the states reachable from *a*

$$Q(s,a) \;=\; R(s) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')$$

Q values

	s_0	s_1	…	s_k
a_0	**Q[s_0,a_0]**	**Q[s_1,a_0]**	….	**Q[s_k,a_0]**
a_1	**Q[s_0,a_1]**	**Q[s_1,a_1]** max	…	**Q[s_k,a_1]**
…	…	…	….	…
a_n	**Q[s_0,a_n]**	**Q[s_1,a_n]**	….	**Q[s_k,a_n]**

- Once the agent has a **complete Q-function**, it knows how to act in every state
- By learning what to do in each state, rather then the complete policy as in search based methods, learning becomes linear rather than exponential in the number of states
- **But how to learn the Q-values?**

Learning the Q values

- Can we exploit the relation between Q values in "adjacent" states?

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

- No, because we don't know the transition probabilities *P(s'|s,a)*

- We'll use a different approach, that relies on the notion on Temporal Difference (TD)

Average Through Time

- Suppose we have a sequence of values (your sample data):

$$v_1,\ \ v_2, .., v_k$$

- And want a running approximation of their expected value
  - e.g., given sequence of grades, estimate expected value of next grade
- A reasonable **estimate** is the average of the first k values:

$$A_k = \frac{v_1 + v_2 + ... + v_k}{k}$$

Average Through Time

$$A_k = \frac{v_1 + v_2 + ... + v_k}{k}$$

$$kA_k = v_1 + v_2 + ... + v_k$$ and equivalently for k-1:

$$(k-1)A_{k-1} = v_1 + v_2 + ... + v_{k-1}$$ which substituted in the equation above gives

$$kA_k = (k-1)A_{k-1} + v_k$$ Dividing by k we get :

$$A_k = (1 - \frac{1}{k})A_{k-1} + \frac{v_k}{k}$$

and if we set $\alpha_k = 1/k$

$$A_k = (1 - \alpha_k)A_{k-1} + \alpha_k v_k$$

$$= A_{k-1} + \alpha_k (v_k - A_{k-1})$$

Estimate by Temporal Differences

$$A_k = A_{k-1} + \alpha_k (v_k - A_{k-1})$$

NEW ESTIMATE

PREVIOUS ESTIMATE

NEW VALUE

- $(v_k - A_{k-1})$ is called a ***temporal difference error*** or ***TD-error***
  - it specifies how different the new value v_k is from the prediction given by the previous running average A_{k-1}
- The new estimate (average) is obtained by updating the previous average by α_k times the TD error

Q-learning: General Idea

- Learn from the ***history*** of interaction with the environment, *i.e.*, a sequence of state-action-rewards

$$<s_0,\ a_0, r_1, s_1,\ a_1, r_2, s_2,\ a_2, r_3, \ldots.>$$

- History is seen as sequence of ***experiences***, i.e., tuples

$$<s, a, r, s'>$$

  - agent doing action *a* in state *s*,
  - receiving reward *r* and ending up in *s'*

- These experiences are used to estimate the value of $Q(s,a)$ expressed as

$$Q(s,a) = r + \gamma V(s') \quad \text{where } V(s') = \max_{a'} Q[s', a']$$

Q-learning: General Idea

But remember

s a r s'

$$Q(s,a) = r + \gamma \max_{a'} Q[s',a']$$

Is an **approximation**. The real link between Q(s,a) and Q(s',a') is

$$Q(s,a) \;=\; R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

Q-learning: Main steps

Store ***Q[S, A]*,** for every state *S* and action *A* in the world

- Start with **arbitrary estimates** in $Q^{(0)}[S, A]$,
- Update them by using experiences
  - Each **experience** $<s, a, r, s'>$ provides one new data point on the actual value of $Q[s, a]$

$$Q[s,a] = r + \gamma \max_{a'} Q[s',a']$$

current *estimated* value of Q[s',a'], where s' is the state the agent arrives to in the current experience

New value of Q[s,a],

Q-learning: Update step

$$A_k = A_{k-1} + \alpha_k (v_k - A_{k-1})$$

NEW ESTIMATE

PREVIOUS ESTIMATE

NEW VALUE

- *TD* formula applied to Q[s,a]

$$Q^{(i)}[s,a] \leftarrow Q^{(i-1)}[s,a] + \alpha((r + \gamma \max_{a'} Q^{(i-1)}[s',a']) - Q^{(i-1)}[s,a])$$

updated *estimated* value of Q[s,a]

New value for Q[s,a] from <s,a,r,s'>

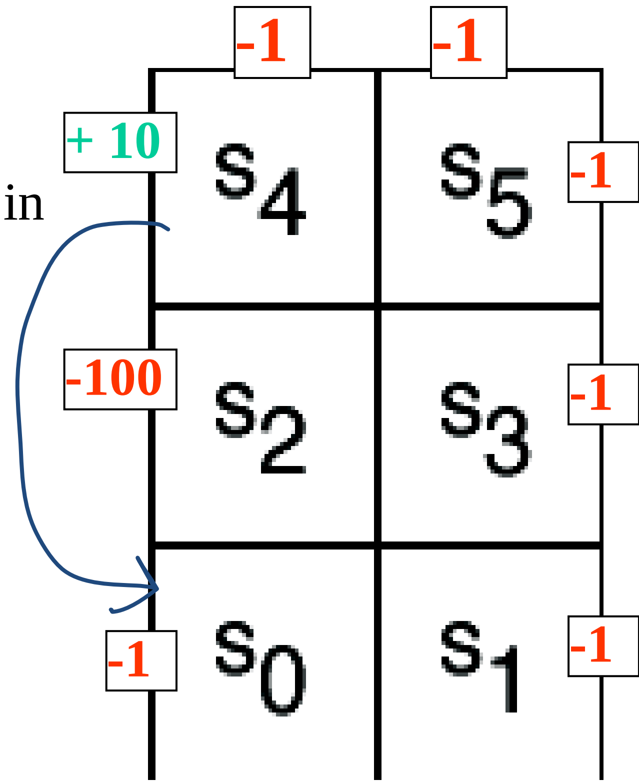
Previous *estimated* value of Q[s,a]

Q-learning: algorithm

controller Q-learning(S,A)
inputs:
  S is a set of states
  A is a set of actions
  γ the discount
  α is the step size
internal state:
  real array $Q[S,A]$
  previous state s
  previous action a
begin
  initialize $Q[S,A]$ arbitrarily
  observe current state s
  repeat forever:
    select and carry out an action a
    observe reward r and state s'
    $Q[s,a] \leftarrow Q[s,a] + \alpha\left(r + \gamma \max_{a'} \boxed{Q[s',a']} - Q[s,a]\right)$
    $s \leftarrow s'$;
  end-repeat
end

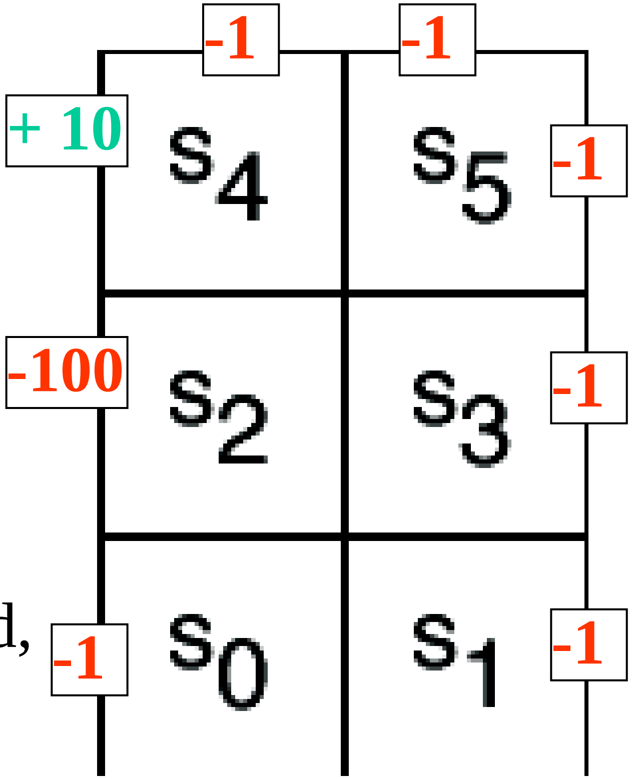
Example

- Six possible states $<s_0,..,s_5>$
- 4 actions:
  - *UpCareful:* moves one tile up unless there is wall, in which case stays in same tile. Always generates a penalty of -1
  - *Left:* moves one tile left unless there is wall, in which case
    - ✓ stays in same tile if in s_0 or s_2
    - ✓ Is sent to s_0 if in s_4
  - *Right:* moves one tile right unless there is wall, in which case stays in same tile
  - *Up:* 0.8 goes up unless there is a wall, 0.1 like *Left*, 0.1 like *Right*



- Reward Model:
  - -1 for doing *UpCareful*
  - Negative reward when hitting a wall, as marked on the picture

Example



- The agent **knows** about the 6 states and 4 actions
- Can perform an action, fully observe its state and the reward it gets
- **Does not know** how the states are configured, nor what the actions do
  - **no transition model, nor reward model**

Example (variable α_k)

- Suppose that in the simple world described earlier, the agent has the following sequence of experiences

  $<s_0$, *right*, 0, s_1, *upCareful*, -1, s_3, *upCareful*, -1, s_5, *left*, 0, s_4, *left*, 10, $s_0>$

- And repeats it *k* times (not a good behavior for a Q-learning agent, but good for didactic purposes)

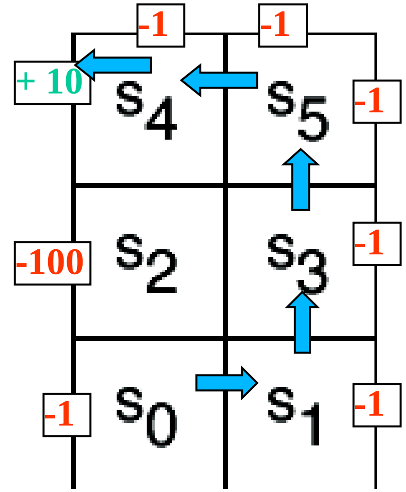
- Table shows the first 3 iterations of Q-learning when
  - *Q[s,a]* is initialized to 0 for every *a* and *s*
  - $\alpha_k = 1/k$, $\gamma = 0.9$

Iteration	$Q[s_0, right]$	$Q[s_1, upCare]$	$Q[s_3, upCare]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	4.5	10
3	0	-1	0.35	6.0	10

- For full demo, see http://www.cs.ubc.ca/~poole/demos/rl/tGame.html

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$

$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$



k=1

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	0	0	0	0	0	0
Left	0	0	0	0	0	0
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k((r + 0.9 \max_{a'} Q[s_1, a']) - Q[s_0, right]);$$

$$Q[s_0, right] \leftarrow$$

$$Q[s_1, upCarfull] \leftarrow Q[s_1, upCarfull] + \alpha_k((r + 0.9 \max_{a'} Q[s_3, a']) - Q[s_1, upCarfull];$$

$$Q[s_1, upCarfull] \leftarrow$$

$$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k((r + 0.9 \max_{a'} Q[s_5, a']) - Q[s_3, upCarfull];$$

$$Q[s_3, upCarfull] \leftarrow$$

$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_4, a']) - Q[s_5, Left];$$

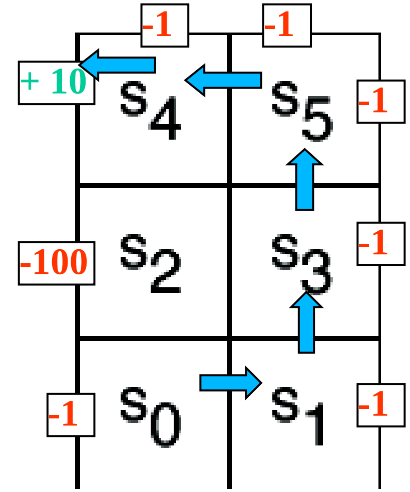
$$Q[s_5, Left] \leftarrow 0 + 1(0 + 0.9 * 0 - 0) = 0$$

Only immediate rewards are included in the update in this first pass

$$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_0, a']) - Q[s_4, Left];$$

$$Q[s_4, Left] \leftarrow 0 + 1(10 + 0.9 * 0 - 0) = 10$$

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$



$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

k=2

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	**0**	**-1**	0	**-1**	0	0
Left	0	0	0	0	**10**	**0**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k((r + 0.9 \max_{a'} Q[s_1, a']) - Q[s_0, right]);$$

$$Q[s_0, right] \leftarrow 0 + 1/2(0 + 0.9 * 0 - 0) = 0$$

$$Q[s_1, upCarfull] \leftarrow Q[s_1, upCarfull] + \alpha_k((r + 0.9 \max_{a'} Q[s_3, a']) - Q[s_1, upCarfull] =$$

$$Q[s_1, upCarfull] \leftarrow -1 + 1/2(-1 + 0.9 * 0 + 1) = -1$$

$$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k((r + 0.9 \max_{a'} Q[s_5, a']) - Q[s_3, upCarfull] =$$

$$Q[s_3, upCarfull] \leftarrow -1 + 1/2(-1 + 0.9 * 0 + 1) = -1$$

1 step backup from previous positive reward in s4

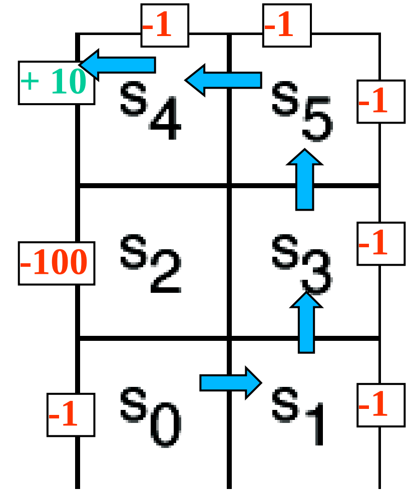
$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_4, a']) - Q[s_5, Left] =$$

$$Q[s_5, Left] \leftarrow$$

$$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_0, a']) - Q[s_4, Left] =$$

$$Q[s_4, Left] \leftarrow 10 + 1(10 + 0.9 * 0 - 10) = 10$$

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$



$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

k=3

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	**0**	**-1**	0	**-1**	0	0
Left	0	0	0	0	**10**	**4.5**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k((r + 0.9 \max_{a'} Q[s_1, a']) - Q[s_0, right]);$$

$$Q[s_0, right] \leftarrow 0 + 1/3(0 + 0.9*0 - 0) = 0$$

$$Q[s_1, upCarfull] \leftarrow Q[s_1, upCarfull] + \alpha_k((r + 0.9 \max_{a'} Q[s_3, a']) - Q[s_1, upCarfull] =$$

$$Q[s_1, upCarfull] \leftarrow -1 + 1/3(-1 + 0.9*0 + 1) = -1$$

$$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k((r + 0.9 \max_{a'} Q[s_5, a']) - Q[s_3, upCarfull] =$$

$$Q[s_3, upCarfull] \leftarrow -1 + 1/3(-1 + 0.9*4.5 + 1) = 0.35$$

The effect of the positive reward in s4 is felt two steps earlier at the 3rd iteration

$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_4, a']) - Q[s_5, Left] =$$

$$Q[s_5, Left] \leftarrow 4.5 + 1/3(0 + 0.9*10 - 4.5) = 6$$

$$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_0, a']) - Q[s_4, Left] =$$

$$Q[s_4, Left] \leftarrow 10 + 1(10 + 0.9*0 - 10) = 10$$

Example (variable α_k)

Iteration	$Q[s_0, right]$	$Q[s_1, upCare]$	$Q[s_3, upCare]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	4.5	10
3	0	-1	0.35	6.0	10
4	0	-0.92	1.36	6.75	10
10	0.03	0.51	4	8.1	10
100	2.54	4.12	6.82	9.5	11.34
1000	4.63	5.93	8.46	11.3	13.4
10000	6.08	7.39	9.97	12.83	14.9
100000	7.27	8.58	11.16	14.02	16.08
1000000	8.21	9.52	12.1	14.96	17.02
10000000	8.96	10.27	12.85	15.71	17.77
∞	11.85	13.16	15.74	18.6	20.66

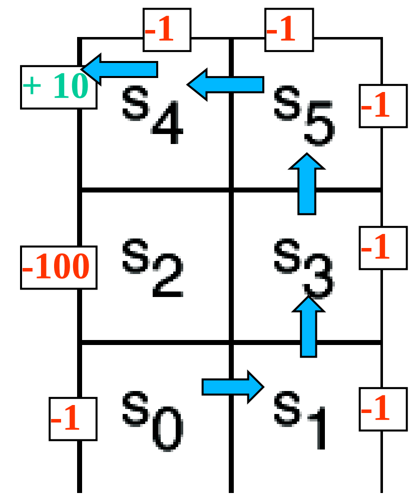
- As the number of iteration increases, the effect of the positive reward achieved by moving left in s_4 trickles further back in the sequence of steps
- Q[s_4,left] starts changing only after the effect of the reward has reached s_0 (i.e. after iteration 10 in the table)

Why 10 and not 6?

Example (Fixed α=1)

- First iteration same as before, let's look at the second

$$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$$



$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

k=2

Q[s,a]	***s*₀**	***s*₁**	***s*₂**	***s*₃**	***s*₄**	***s*₅**
upCareful	**0**	**-1**	0	**-1**	0	0
Left	0	0	0	0	**10**	**0**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow 0 + 1(0 + 0.9 * 0 - 0) = 0$$

$$Q[s_1, upCarfull] \leftarrow -1 + 1(-1 + 0.9 * 0 + 1) = -1$$

$$Q[s_3, upCarfull] \leftarrow -1 + 1(-1 + 0.9 * 0 + 1) = -1$$

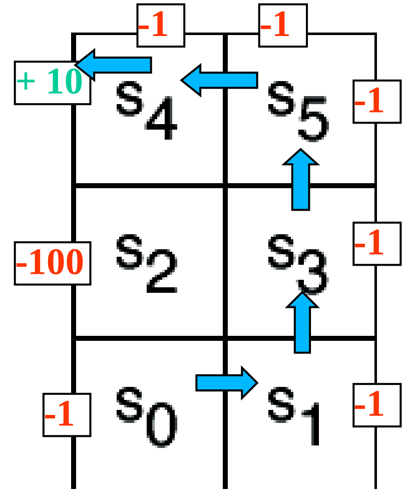
$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_4, a']) - Q[s_5, Left] =$$

$$Q[s_5, Left] \leftarrow 0 + 1(0 + 0.9 * 10 - 0) = 9$$

New evidence is given much more weight than original estimate

$$Q[s_4, Left] \leftarrow 10 + 1(10 + 0.9 * 0 - 10) = 10$$

$$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$$



$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

k=3

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	**0**	**-1**	0	**-1**	0	0
Left	0	0	0	0	**10**	**9**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow 0 + 1(0 + 0.9 * 0 - 0) = 0$$

$$Q[s_1, upCarfull] \leftarrow -1 + 1(-1 + 0.9 * 0 + 1) = -1$$

$$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k((r + 0.9 \max_{a'} Q[s_5, a']) - Q[s_3, upCarfull] =$$

$$Q[s_3, upCarfull] \leftarrow -1 + 1(-1 + 0.9 * 9 + 1) = 7.1$$

Same here

$$Q[s_5, Left] \leftarrow 9 + 1(0 + 0.9 * 10 - 9) = 9$$

$$Q[s_4, Left] \leftarrow 10 + 1(10 + 0.9 * 0 - 10) = 10$$

No change from previous iteration, as all the reward from the step ahead was included there

Comparing fixed α *(top)* and variable α *(bottom)*

Iteration	$Q[s_0, right]$	$Q[s_1, upCare]$	$Q[s_3, upCare]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	9	10
3	0	-1	7.1	9	10
4	0	5.39	7.1	9	10
5	4.85	5.39	7.1	9	14.37
6	4.85	5.39	7.1	12.93	14.37
10	7.72	8.57	10.64	15.25	16.94
20	10.41	12.22	14.69	17.43	19.37
30	11.55	12.83	15.37	18.35	20.39
40	11.74	13.09	15.66	18.51	20.57
∞	11.85	13.16	15.74	18.6	20.66

Iteration	$Q[s_0, right]$	$Q[s_1, upCare]$	$Q[s_3, upCare]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	4.5	10
3	0	-1	0.35	6.0	10
4	0	-0.92	1.36	6.75	10
10	0.03	0.51	4	8.1	10
100	2.54	4.12	6.82	9.5	11.34
1000	4.63	5.93	8.46	11.3	13.4
10000	6.08	7.39	9.97	12.83	14.9
100000	7.27	8.58	11.16	14.02	16.08
1000000	8.21	9.52	12.1	14.96	17.02
10000000	8.96	10.27	12.85	15.71	17.77
∞	11.85	13.16	15.74	18.6	20.66

Fixed α generates faster update:

all states see some effect of the positive reward from <s4, left> by the 5th iteration

Each update is much larger

Gets very close to final numbers by iteration 40, while with variable α still not there by iteration 10^7

However, remember:

Q-learning with fixed α is not guaranteed to converge

Why approximations work…

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

True relation between Q(s.a) and Q(s'a')

$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

Q-learning approximation based on each individual experience <*s, a, s'*>

- Way to get around the **missing transition model and reward model**
- Aren't we in danger of using data coming from unlikely transition to make incorrect adjustments?
- No, as long as Q-learning tries each action an unbounded number of times
  - Frequency of updates reflects transition model, *P(s'|a,s)*

Course summary

R&R + **ML**

Deterministic Environment

(not in this picture)

Stochastic Environment

Query

Belief Nets

Var. Elimination

Approx. Inference

Markov Chains and HMMs

Temporal. Inference

Planning

Decision Nets

Var. Elimination

Markov Decision Processes

POMDPs

Value Iteration

Approx. Inference

Decision Trees

Supervised

learn parameters by counting. Examples

NB Classifiers

Language Models

Unsupervised

EM

K-MEANS clustering

Reinforcement Learning

Q-learning

502: what is next

- **Midterm exam** @5:30-7pm this room DMP 201 Mon

- Readings / Your Presentations will start Nov 17

- We will have a make-up class later