Artificial Intelligence:
Representation and Problem Solving

15-381

May 8, 2007

Review Session for Final

Essential things to know (from 2nd half)

- Probability and Uncertainty
- Bayes nets
- Decision Trees
- Probabilistic Learning
- Cross Validation
- Clustering and k-Nearest Neighbors
- Neural Nets
- MDPs
- HMMs
- Reinforcement Learning

Probability and Uncertainty

- sum rule (marginalization)

$$p(X = x_i) = \sum_{j=1}^{N} p(X = x_i, Y = y_j)$$

$$p(x) = \sum_{y} p(x, y) \quad \text{short hand notation}$$

$$p(x) = \int dy\, p(x, y)$$

- product rule $p(x, y) = p(x|y)p(y)$

- Bayes rule $p(y|x) = \frac{p(x|y)p(y)}{p(x)} = \frac{p(x|y)p(y)}{\sum_y p(x|y)p(y)}$

- independence $p(x, y) = p(x)p(y)$ iff x and y are **independent**

$$p(x_1, x_2, \ldots, x_N) = \prod_{n=1}^{N} p(x_n)$$

Bayes Nets

- Bayes Net = directed graph of variables with the property that a variable is conditionally independent of its non-descendants given its parents

  P(X | Parents(X),Y,Z,W) = P(X|Parents(X))

- Associated with each variable is the Conditional Probability Table (CPT) of P(X|Parents(X))

- Any entry of the joint probability distribution can be computed from the Bayes Net by looking only at the CPT:

$$P(x_1, \ldots, x_n) \equiv P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n)$$
$$= \prod_{i=1}^{n} P(x_i|\text{parents}(X_i))$$

Bayes Nets

- Inference: Given observed values for a subset of variables (the "evidence" E2), compute the probability distribution of another set of variables E1 (the "query"), symbolically denoted by: P(E1 | E2)
- In principle, any inference can be computed from a Bayes Nets by summing up the appropriate entries of the joint distribution:

$$P(E_1 \mid E_2) = \frac{P(E_1, E_2)}{P(E_2)} = \frac{\sum_{\text{All joint entries } X \text{ that contain } E_1 \wedge E_2} P(X)}{\sum_{\text{All joint entries } Y \text{ that contain } E_2} P(Y)}$$

Bayes Nets

- Inference requires in general exponential time in the number of variables, because of the number of terms in the sums
- The sums can be simplified by cleverly grouping the terms that depend only on one (or a small number of) variable

$$P(D=d) = \sum_{a,b,c} P(D=d \mid C=c)P(C=c \mid B=b)P(B=b \mid A=a)P(A=a)$$

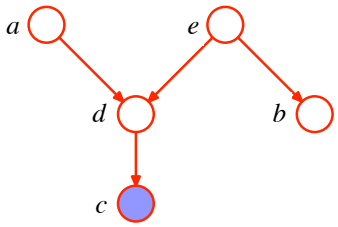
$$\Downarrow$$

$$P(D=d) = \sum_{c} P(D=d \mid C=c) \sum_{b} P(C=c \mid B=b) \sum_{a} P(B=b \mid A=a)P(A=a)$$

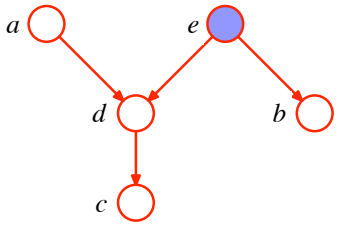
- This can implemented in polynomial time if the graph is a "polytree" (i.e., the undirected version of the graph is a tree)
- Expected:
  - Understand how probabilities can be expressed as sums of products
  - How to group terms advantageously and do so on simple examples

Statistical dependences in belief nets

- Is an independence statement, e.g. is a ⊥ b | c implied by a graph?
- If two nodes are *conditionally independent* given another variable, e.g. a ⊥ b | c then p(a|b,c) = p(a|c).
- Note: There is a technique called "d-separation" for determining whether two noes (or sets of nodes) are independent. It was covered in previous years and is in previous exams, but the book does not cover this and we did not cover it in lecture, so you are not responsible for this material.
- What you should understand: How information can propagate in belief networks to affect the beliefs about certain nodes.

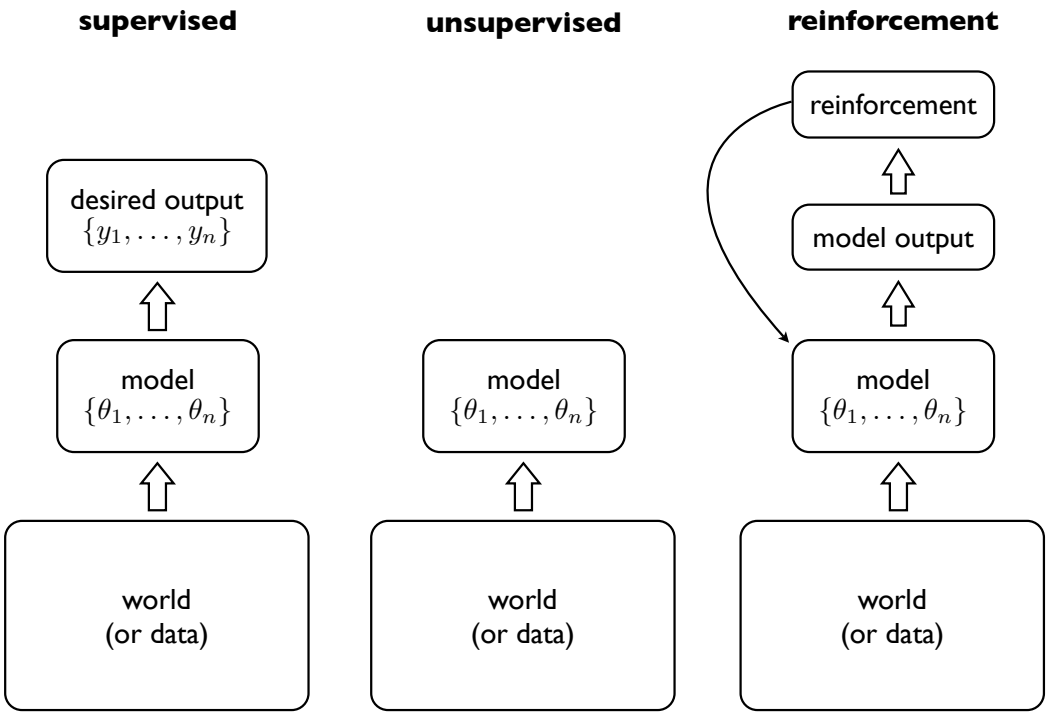


Is a independent of b given c?
No. The inferred value of f depends on b. e is also factor in determining d. d in turn influences a. So the value of b can influence the value of a, so they are not independent.



Is a is independent of b given e ?
Yes. e fixed completely determines the probability of b via $p(b|e)$.

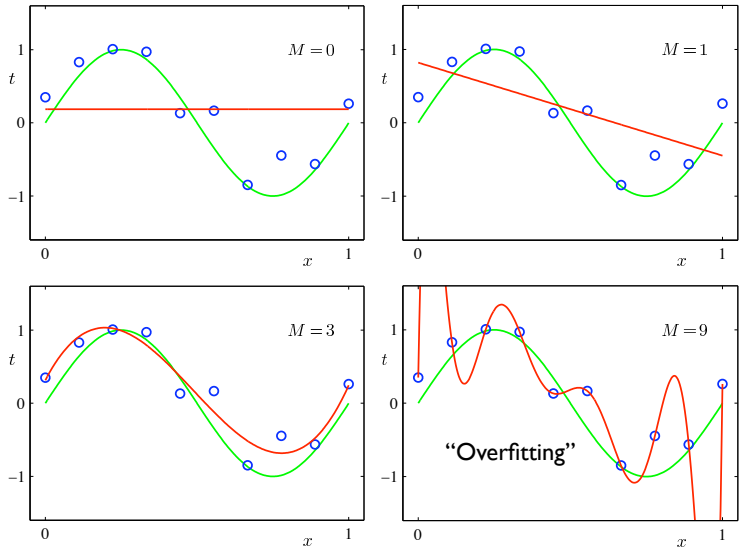
Types of learning



Polynomial curve fitting

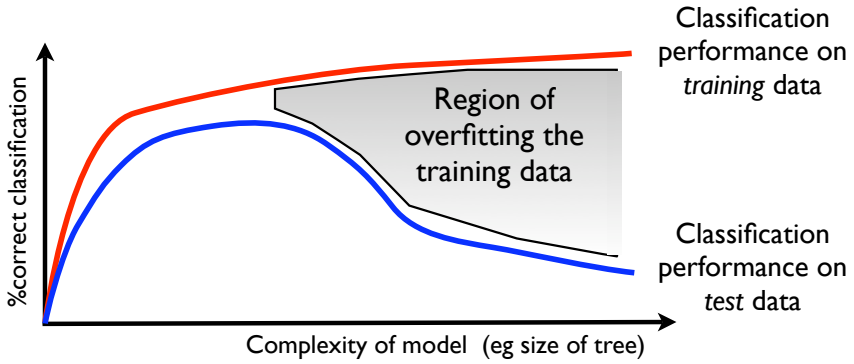
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} [y(x_n, \mathbf{w}) - t_n]^2$$



example from Bishop (2006), *Pattern Recognition and Machine Learning*

General principle

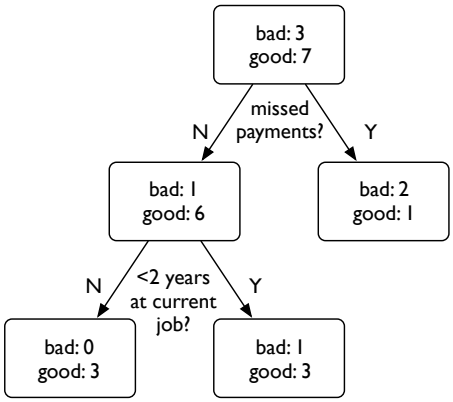


- As its complexity increases, the model is able to better classify the training data
- Performance on the test data initially increases, but then falls as the model *overfits*, or becomes specialized for classifying the noise training
- The complexity in decision trees is the number of free parameters, ie the number of nodes

Decision trees: classifying from a set of attributes

Predicting credit risk

<2 years at current job?	missed payments?	defaulted?
N	N	N
Y	N	Y
N	N	N
N	N	N
N	Y	Y
Y	N	N
N	Y	N
N	Y	Y
Y	N	N
Y	N	N



- each level splits the data according to different attributes
- **goal**: achieve perfect classification with minimal number of decisions
  - not always possible due to noise or inconsistencies in the data

Entropy

- How many bits does it take to specify the attribute of 'defaulted?'
  - P(defaulted = Y) = 3/10
  - P(defaulted = N) = 7/10

$$\begin{aligned} H(Y) &= -\sum_{i=\mathrm{Y,N}} P(Y=y_i)\log_2 P(Y=y_i) \\ &= -0.3\log_2 0.3 - 0.7\log_2 0.7 \\ &= 0.8813 \end{aligned}$$

- How much can we *reduce* the entropy (or uncertainty) of 'defaulted' by knowing the other attributes?
- Ideally, we could reduce it to zero, in which case we classify perfectly.

Predicting credit risk

<2 years at current job?	missed payments?	defaulted?
N	N	N
Y	N	Y
N	N	N
N	N	N
N	Y	Y
Y	N	N
N	Y	N
N	Y	Y
Y	N	N
Y	N	N

Conditional entropy

- H(Y|X) is the remaining entropy of Y given X

  or

  The expected (or average) entropy of P(y|x)

$$\begin{aligned} H(Y|X) &\equiv -\sum_x P(x) \sum_y P(y|x) \log_2 P(y|x) \\ &= -\sum_x P(x) \sum_y P(Y=y|X=x) \log_2 P(Y=y|X=x) \\ &= -\sum_x P(x) \sum_y H(Y|X=x) \end{aligned}$$

- H(Y|X=x) is the *specific conditional entropy*, i.e. the entropy of Y knowing the value of a specific attribute x.

Back to the credit risk example

Predicting credit risk

<2 yrs	missed	def?
N	N	N
Y	N	Y
N	N	N
N	N	N
N	Y	Y
Y	N	N
N	Y	N
N	Y	Y
Y	N	N
Y	N	N

$$\begin{aligned} H(Y|X) &\equiv -\sum_x P(x) \sum_y P(y|x) \log_2 P(y|x) \\ &= -\sum_x P(x) \sum_y P(Y=y|X=x) \log_2 P(Y=y|X=x) \\ &= -\sum_x P(x) \sum_y H(Y|X=x) \end{aligned}$$

$$\begin{aligned} H(\text{defaulted}|<2\text{years}=\text{N}) &= -\frac{5}{5+2}\log_2\frac{5}{5+2} - \frac{2}{7}\log_2\frac{2}{7} = 0.8631 \\ H(\text{defaulted}|<2\text{years}=\text{Y}) &= -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.8133 \\ H(\text{defaulted}|\text{missed}) &= \frac{6}{10}0.8631 + \frac{4}{10}0.8133 = 0.8432 \end{aligned}$$

$$\begin{aligned} H(\text{defaulted}|\text{missed}=\text{N}) &= -\frac{6}{7}\log_2\frac{6}{7} - \frac{1}{7}\log_2\frac{1}{7} = 0.5917 \\ H(\text{defaulted}|\text{missed}=\text{Y}) &= -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.9183 \\ H(\text{defaulted}|\text{missed}) &= \frac{7}{10}0.5917 + \frac{3}{10}0.9183 = 0.6897 \end{aligned}$$

Mutual information

- We now have the entropy - the minimal number of bits required to specify the target attribute:

$$H(Y) = \sum_y P(y) \log_2 P(y)$$

- The conditional entropy - the remaining entropy of Y knowing X

$$H(Y|X) = -\sum_x P(x) \sum_y P(y|x) \log_2 P(y|x)$$

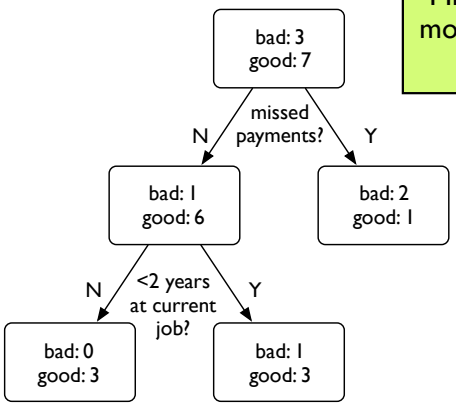
- So we can now define the reduction of the entropy after learning Y.
- This is known as the *mutual information* between Y and X

$$I(Y;X) = H(Y) - H(Y|X)$$

Information gain

$$\begin{aligned} H(\text{defaulted}) &- H(\text{defaulted}|< 2 \text{ years}) \\ 0.8813 &- 0.8432 = 0.0381 \\ H(\text{defaulted}) &- H(\text{defaulted}|\text{missed}) \\ 0.8813 &- 0.6897 = 0.1916 \end{aligned}$$

Missed payments are the most informative attribute about defaulting.



The scaling problem in joint probabilities

$$p(\mathbf{x} = \mathbf{v}|C_k) = \frac{\text{Count}(\mathbf{x} = \mathbf{v} \wedge C_k = k)}{\text{Count}(C_k = k)}$$

$$p(x_1 = v_1, \ldots, x_N = v_N|C_k = k) = \frac{\text{Count}(x_1 = v_1, \ldots \wedge x_N = v_N, \wedge C_k = k)}{\text{Count}(C_k = k)}$$

- The likelihood, the table (or joint probability) is huge!

Simplifying with "Naïve" Bayes

- What if we assume the features are independent?

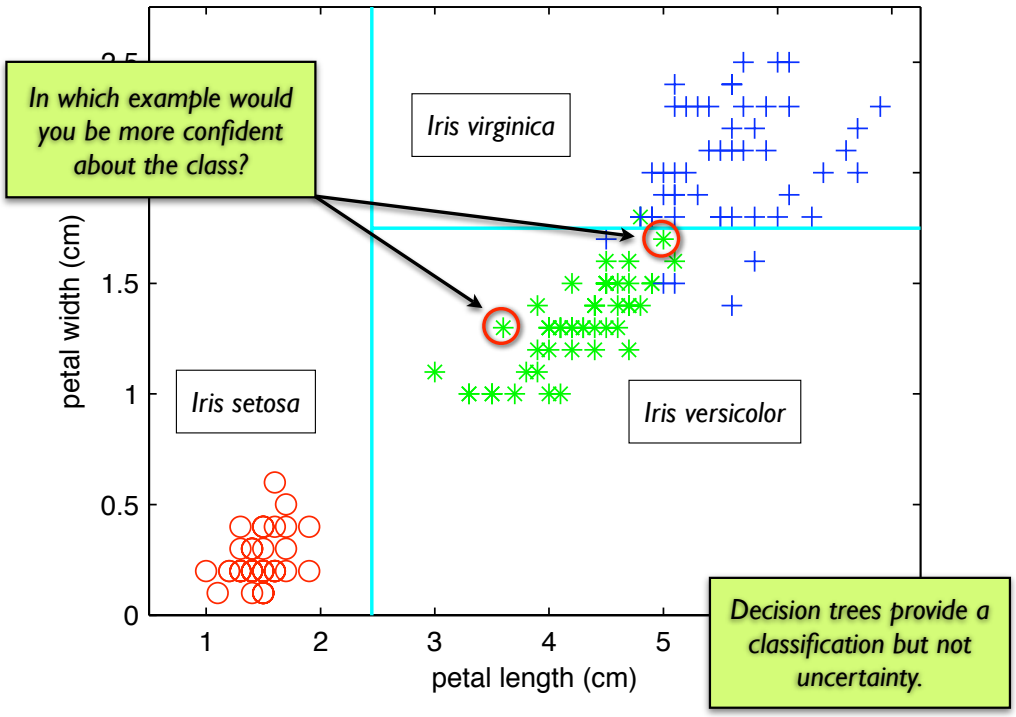
$$\begin{aligned} p(\mathbf{x}|C_k) &= p(x_1, \ldots, x_N|C_k) \\ &= \prod_{n=1}^{N} p(x_n|C_k) \end{aligned}$$

- We know that's not precisely true, but it might make a good approximation.
- Now we only need to specify N different likelihoods:

$$p(x_i = v_i|C_k = k) = \frac{\text{Count}(x_i = v_i \wedge C_k = k)}{\text{Count}(C_k = k)}$$

- Huge savings in number of of parameters

Decision trees with continuous variables



Probabilistic classification

- Let's apply Bayes' rule to infer the most probable class given the observation:

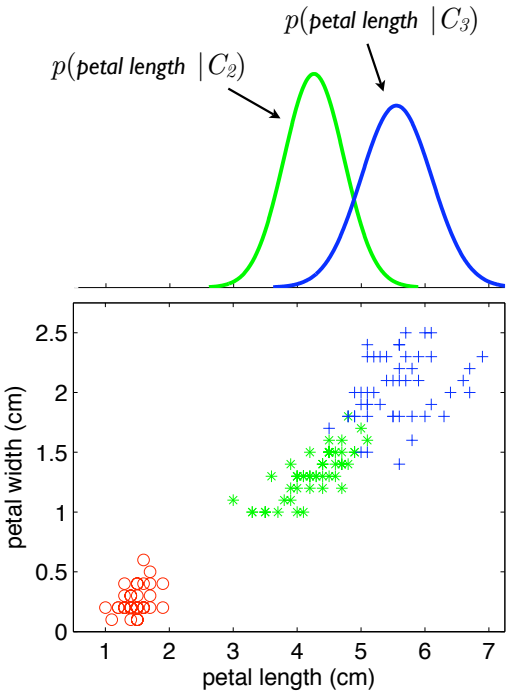
$$
\begin{aligned}
p(C_k|\mathbf{x}) &= \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} \\
&= \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_k p(\mathbf{x}|C_k)p(C_k)}
\end{aligned}
$$

- This is the answer, but what does it mean?
- How do we specify the terms?
  - $p(C_k)$ is the prior probability on the different classes
  - $p(\mathbf{x}|C_k)$ is the data likelihood, ie probability of $\mathbf{x}$ given class C_k
- How should we define this?

What classifier would give "optimal" performance?

- Consider the iris data again.
- How would we minimize the number of *future* mis-classifications?
- We would need to know the true *distribution* of the classes.
- Assume they follow a Gaussian distribution.
- The number of samples in each class is the same (50), so (assume) $p(C_k)$ is equal for all classes.
- Because $p(\mathbf{x})$ is the same for all classes we have:

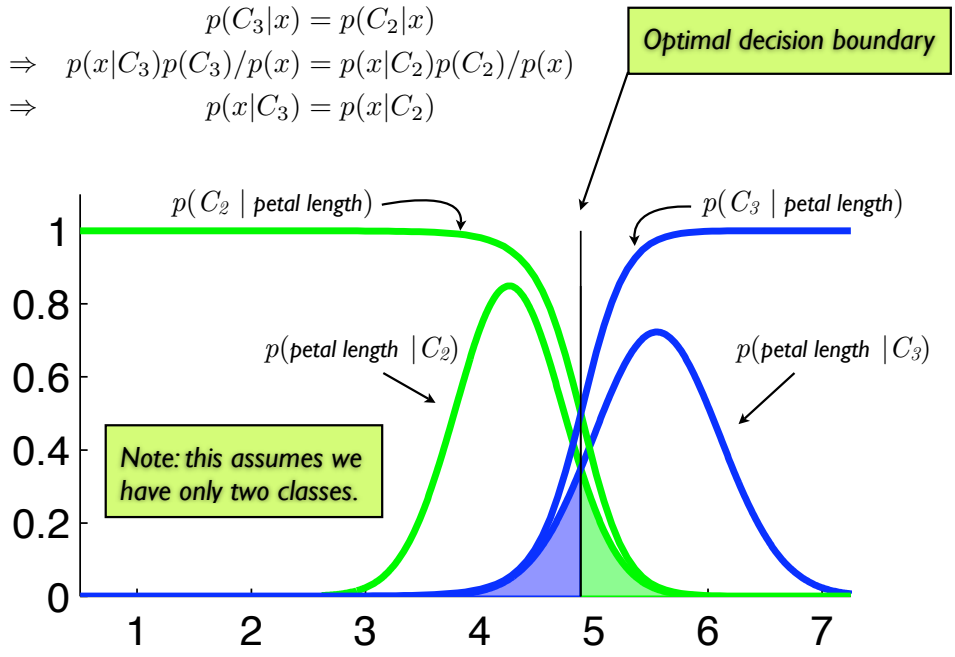
$$\begin{aligned} p(C_k|\mathbf{x}) &= \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} \\ &\propto p(\mathbf{x}|C_k)p(C_k) \end{aligned}$$



Optimal decision boundaries

- The minimal misclassification error at the point where

$$\begin{aligned} & p(C_3|x) = p(C_2|x) \\ \Rightarrow \quad & p(x|C_3)p(C_3)/p(x) = p(x|C_2)p(C_2)/p(x) \\ \Rightarrow \quad & p(x|C_3) = p(x|C_2) \end{aligned}$$



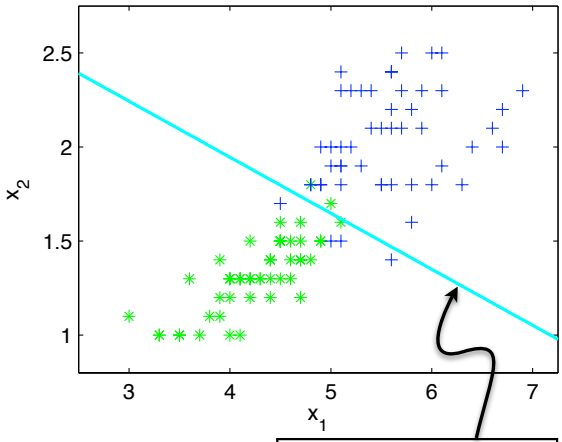
Defining a decision boundary

- consider just two classes
- want points on one side of line in class 1, otherwise class 2.
- 2D *linear discriminant function:*

$$\begin{aligned} y &= \mathbf{m}^T\mathbf{x} + b \\ &= m_1 x_1 + m_2 x_2 + b \\ &= \sum_i m_i x_i + b \end{aligned}$$

- This defines a 2D plane which leads to the decision:

$$\mathbf{x} \in \begin{cases} \text{class 1} & \text{if } y \geq 0, \\ \text{class 2} & \text{if } y < 0. \end{cases}$$



The decision *boundary*:
$y = \mathbf{m}^T\mathbf{x} + b = 0$

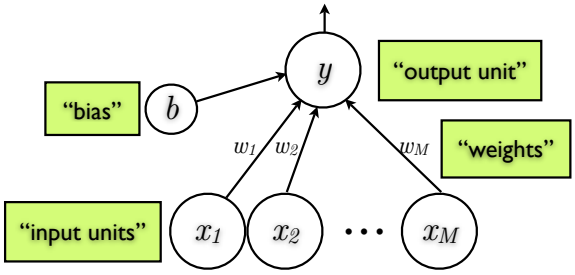
Or in terms of scalars:

$$\begin{aligned} m_1 x_1 + m_2 x_2 &= -b \\ \Rightarrow x_2 &= -\frac{m_1 x_1 + b}{m_2} \end{aligned}$$

Diagraming the classifier as a "neural" network

- The feedforward neural network is specified by weights w_i and bias b:

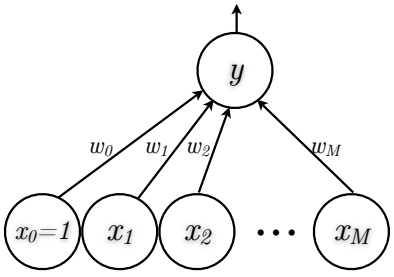
$$\begin{aligned} y &= \mathbf{w}^T\mathbf{x} + b \\ &= \sum_{i=1}^{M} w_i x_i + b \end{aligned}$$



- It can written equivalently as

$$y = \mathbf{w}^T\mathbf{x} = \sum_{i=0}^{M} w_i x_i$$

- where $w_0 = b$ is the bias and a "dummy" input x_0 that is always 1.



Non-linear neural networks

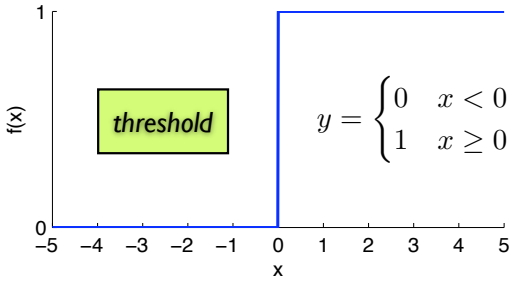
- Idea introduce a non-linearity:

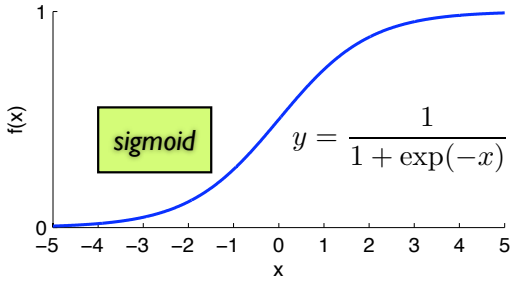
$$y_j = f(\sum_i w_{i,j} x_i)$$

- Now, multiple layers are *not* equivalent

$$\begin{aligned} z_j &= f(\sum_k v_{j,k}\, y_j) \\ &= f(\sum_k v_{j,k}\, f(\sum_i w_{i,j} x_i)) \end{aligned}$$

- Common nonlinearities:
  - threshold
  - sigmoid

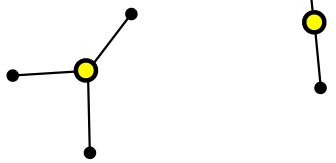




k-means clustering

- Idea: try to estimate k cluster centers by minimizing "distortion"
- Define distortion as:

$$\begin{aligned} D &= \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} \parallel \mathbf{x}_n - \boldsymbol{\mu}_k \parallel^2 \\ r_{nk} &= 1 \text{ if } \mathbf{x}_n \in \text{cluster } k, \text{ 0 otherwise.} \end{aligned}$$



- r_{nk} is 1 for the closest cluster mean to $\mathbf{x}_n$.
- Each point $\mathbf{x}_n$ is the minimum distance from its closet center.
- How do we learn the cluster means?
- Use EM = Estimate Maximize

Using EM to estimate the cluster means

- Our objective function is:

$$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \parallel \mathbf{x}_n - \boldsymbol{\mu}_k \parallel^2$$

- Differential wrt to the mean (the parameter we want to estimate):

$$\frac{\partial D}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k)$$

- We know the optimum is when

$$\frac{\partial D}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k) = 0$$

- Solving for the mean we have:

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}}$$

- This is simply a weighted mean for each cluster.

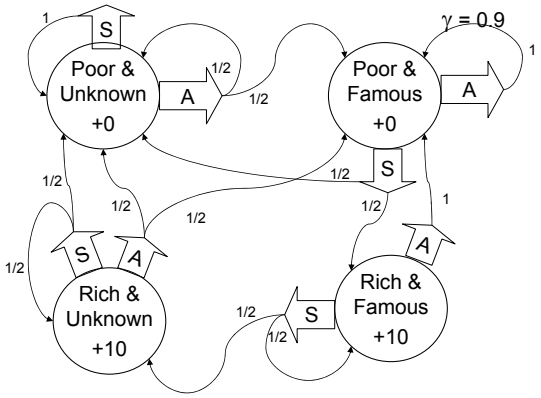
- Thus we have a simple estimation algorithm (*k-means clustering*)
  1. select k points at random
  2. estimate (update) means
  3. repeat until converged

- convergence (to a local minimum) is guaranteed

MDPs

- **MDP** defined by: set of states + set of actions + probabilities of moving from one state (s) to another (s') after executing an action (a) + reward at each state R(s)
- A **policy** is a mapping from states to actions: a=π(s) is the action taken at state s
- The **utility** (or value) of a state U(s) is the expected sum of the rewards that we will receive in the future starting at s.
- We might be interested in a finite amount of time in the future ("finite horizon") or in the rewards over an unlimited amount of time ("infinite horizon").
- If using an infinite horizon, we need to decrease the rewards by a **discount factor** $0 < \gamma < 1$ at every time step.

MDPs



- An MDP is defined by:

states:	$\{s_1, \ldots, s_N\}$
initial state:	S_0
actions:	$\{a_1, \ldots, a_M\}$
rewards:	$R(s) = \{r_1, \ldots, r_N\}$
transitions:	$T(s, a, s') = P(s_{t+1} = s' \mid a_t = a, s_t = s)$
discount:	γ

MDPs

- Assuming a particular policy , U(s) is the sum of the reward at s and the expected reward after executing action a, discounted by γ:

$$V(s) = R(s) + \gamma \sum_{s'} T(s, a, s') V(s')$$

- The optimal policy is the one that yields the largest value of utility for the MDP. The resulting utility satisfies:

$$V^*(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') V^*(s')$$

- The optimal policy is:

$$\pi^*(s) = \arg\max_a \left[\sum_{s'} T(s, a, s') V^*(s') \right]$$

- This cannot be solve directly, since we need to know V*(s).

MDPs

- Iterative techniques to solve for optimal policies
- **Value iteration**

$$V_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') V_i(s')$$
$$V^*(s) = \lim_{i \to \infty} V_i(s)$$

- Then after convergence the optimal policy is:

$$\pi^*(s) = \arg\max_a \left[\sum_{s'} T(s,a,s') V^*(s') \right]$$

MDPs

- Iterative techniques to solve for optimal policies
- **Policy iteration**

$$\pi_{k+1}(s) = \arg\max_a \left[\sum_{s'} T(s,a,s') V_k(s') \right]$$
$$V_{k+1} = R(s) + \gamma \sum_{s'} T(s,a,s') V_k(s'), \quad \text{with } a = \pi_{k+1}(s)$$

- Both value and policy iteration are guaranteed to converge to the optimal value and policy function.

Reinforcement Learning

- Basic problem: Same as before, except we don't know T(.,.,.) or R(.). We need to discover the MDP through exploration.
- Model-Based (**Certainty Equivalent** learning): We estimate T(.,.,.) by recording the states we move through by applying actions during the exploration
- Given estimates of T(.,.,.) (and R(.)), we can compute the optimal V and π by using the previous algorithms (value or policy iteration)
- The problem is that running value/policy iteration is expensive and we'll have to run it many times as we explore. A cheaper way is to update V by using our current estimate of T(.,.,.) at the current visited state (this amounts to doing one step of value iteration at that state only).

Reinforcement Learning

- The model-based techniques requires building estimates of T(.,.,.), which we would like to avoid. One solution: incrementally update V at the state currently visited. This is **temporal difference** or TD learning

$$
\begin{aligned}
V(s) &\leftarrow V(s) + \alpha(R(s) + \gamma V(s') - V(s)) \\
&\text{or} \\
V^{\text{new}}(s) &= V^{\text{old}}(s) + \alpha(R(s) + \gamma V^{\text{old}}(s') - V^{\text{old}}(s))
\end{aligned}
$$

- α is the learning rate and it controls how fast the system converges (low α = slower convergence). This incremental update (Temporal Differencing) is guaranteed to converge, provided that α decreases as the iterations progress.
- Note: The precise conditions on α are stated in the notes, but you do not need to remember these exact conditions. Just the fact that TD can be made to converge.

Reinforcement Learning

- The limitation of TD-like algorithms is that they converge to the optimal V*, but they don't say anything about the optimal policy. An extension is to keep track of the utility of the (state,action) pairs instead of the utility of the states. This is stored in a Q table Q(s,a). This is **Q-learning:**

$$Q^{\text{new}}(s,a) = Q^{\text{old}}(s,a) + \alpha \left[R(s) + \gamma \max_{a'} Q^{\text{old}}(s',a') - Q^{\text{old}}(s,a) \right]$$

- Note on notation: $Q \leftarrow f(Q)$ is the same as $Q^{new} = f(Q^{old})$.
- The optimal policy is computed from our current Q table as (action with the largest utility for each state):

$$\pi(s) = \arg\max_a Q(s,a)$$

- Also: you are not expected to know how these equations are derived.

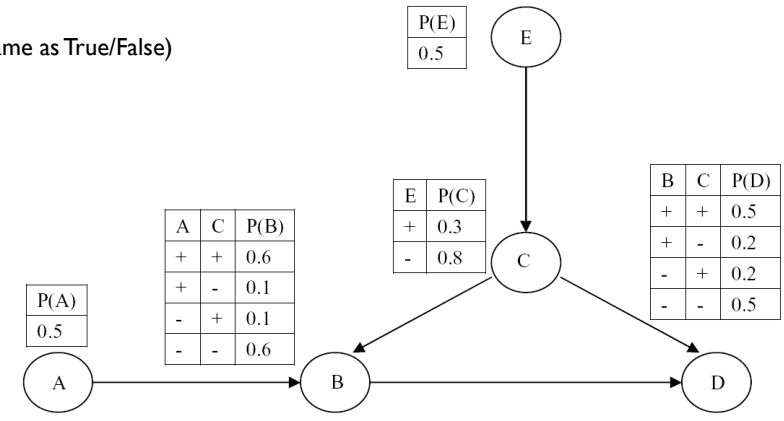
Reinforcement Learning

- **Exploration / exploitation dilemma**.
- In Q-learning care must be taken in how we choose the policy since a poor choice could lead to a sub-optimal policy.
- **Exploration strategies**:
  - *random*: ie no learning estimate best policy at end
  - *greedy*: always choose best estimated action π(s)
  - ϵ-*greedy:* choose best estimated action with probability 1-ϵ
  - *softmax (or Boltzmann):* choose best estimated action with probability, e.g.

$$p \sim \frac{e^{Q(s_i,a)/T}}{\sum_j e^{Q(s_j,a)/T}}$$

some more examples

(+/- is same as True/False)



$$P(C) = P(C,E) + P(C,\neg E) =$$
$$P(C \mid E)P(E) + P(C \mid \neg E)P(\neg E) =$$
$$0.3 \times 0.5 + 0.8 \times 0.5 = 0.55$$

- In the previous slide, the decomposition

$$P(C) = P(C, E) + P(C, \neg E)$$

- is kind of obvious, given that we have a CPT of C conditioned on E, so we know that we'll need to use E. A more mechanical reasoning using factorization works as follows (not that it is not necessary to work through this entire derivation for this kind of question; this only added FYI):

$$P(C = c) = \sum_{a,b,d,e} P(A = a, B = b, C = c, D = d, E = e) =$$

$$\sum_{a,b,d,e} P(A = a)P(B = b \mid A = a, C = c)P(C = c \mid E = e)P(D = d \mid B = b, C = c)P(E = e)$$

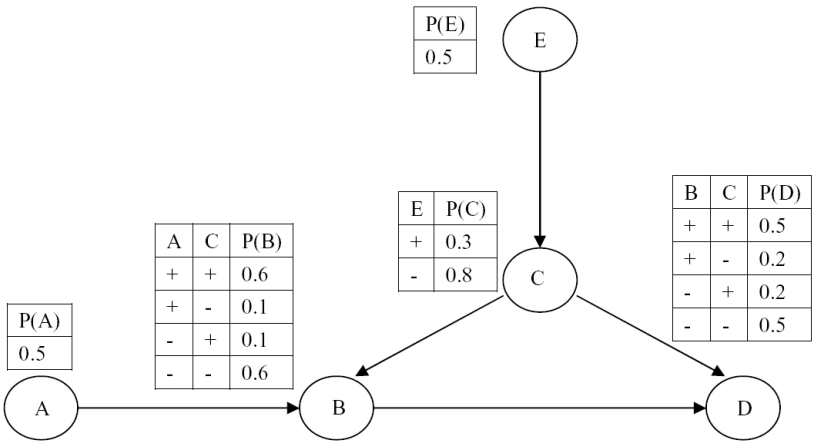
- (The values a,b,c,.. are '+' or '-')
- We can order the sums by moving them as far in as we can:

$$P(C = c) = \sum_{a,b,d,e} P(A = a, B = b, C = c, D = d, E = e) =$$

$$\sum_{e} P(C = c \mid E = e)P(E = e) \sum_{a} P(A = a) \sum_{b} P(B = b \mid A = a, C = c) \sum_{d} P(D = d \mid B = b, C = c)$$

- Three sums disappear because

$$\sum_{d} P(D = d \mid B = b, C = c) = 1 \quad \sum_{b} P(B = b \mid A = a, C = c) = 1 \quad \sum_{a} P(A = a) = 1$$

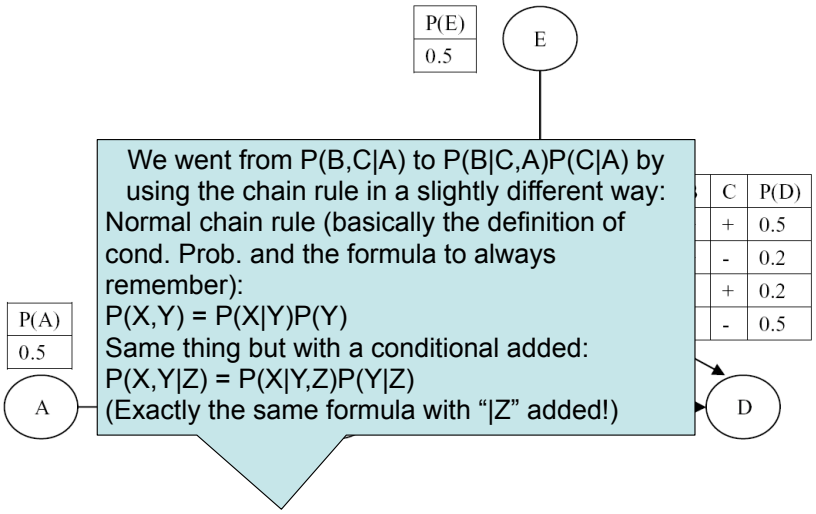


$$P(B \mid A) = P(B, C \mid A) + P(B, \neg C \mid A) =$$

$$P(B \mid A, C)P(C \mid A) + P(B \mid A, \neg C)P(\neg C \mid A) =$$

$$P(B \mid A, C)P(C) + P(B \mid A, \neg C)P(\neg C) =$$

$$0.6 \times 0.55 + 0.1 \times (1 - 0.55) = 0.375$$



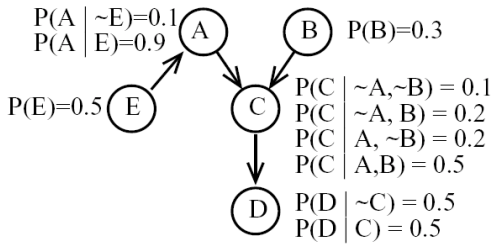
$$P(B \mid A) = P(B, C \mid A) + P(B, \neg C \mid A) =$$

$$P(B \mid A, C)P(C \mid A) + P(B$$

$$P(B \mid A, C)P(C) + P(B$$

$$0.6 \times 0.55 + 0.1 \times (1 - 0.55) = 0.375$$

We can replace P(C|A) by P(C) because A and C are independent

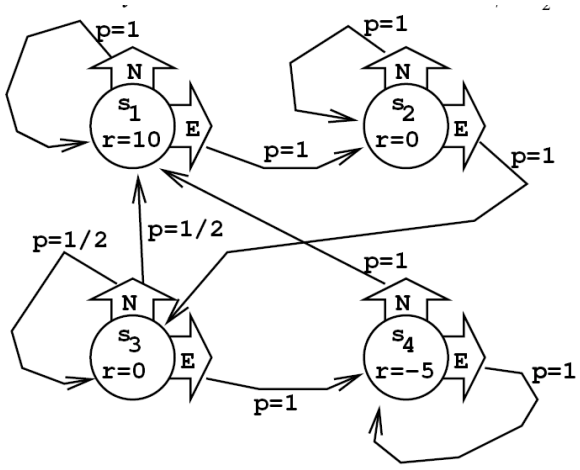


P(E|A,C)/P(E|A) =

P(C|A) =

P(B|D) =

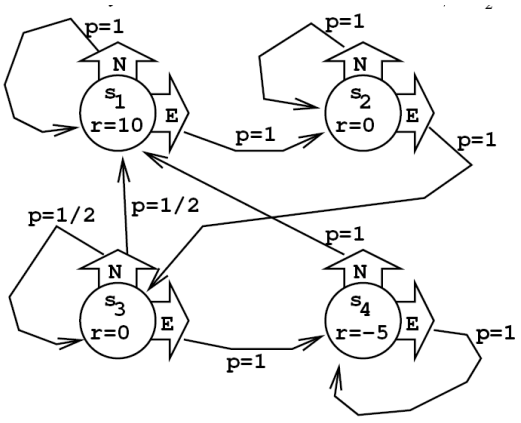
AI 02



- Q: Find the value function V (also commonly called the utility function U) if the policy $\pi_o(s)$ = N for all the states is used.
- Use $\gamma = 0.5$

- This is a direct application of

$$U(s) = R(s) + \gamma \sum_{s'} T(s,a,s')\, U(s')$$



- $U(s_1) = 10 + 0.5 \times U(s_1)$ therefore $U(s_1) = 20$
- $U(s_2) = 0 + 0.5 \times U(s_2)$ therefore $U(s_2) = 0$
- $U(s_3) = 0 + 0.5 \times (0.5 \times U(s_3) + 0.5 \times U(s_1))$ therefore $U(s_3) = U(s_1)/3 = 20/3$
- $U(s_4) = -5 + 0.5 \times U(s_1) = 5$

Q: Run one step of policy iteration after π_o

- This is a direct application of the formula:

  $\pi_{k+1}(s) = \text{argmax}_a \, (\Sigma_{s'} T(s,a,s') \, U_k(s'))$

Expected rewards when *a* = N

Expected rewards when *a* = E

- $\pi_1(s_1) = \text{argmax}(U_o(s_1),U_o(s_2)) = \text{argmax}(20,0) = N$
- $\pi_1(s_2) = \text{argmax}(U_o(s_2),U_o(s_3)) = \text{argmax}(0,20/3) = E$
- $\pi_1(s_3) = \text{argmax}(0.5xU_o(s_1)+0.5xU_o(s_3),U_o(s_4)) = \text{argmax}(10+10/3,5) = N$
- $\pi_1(s_4) = \text{argmax}(U_o(s_1),U_o(s_4)) = \text{argmax}(20,5) = N$

Q: What is the optimal policy after convergence?

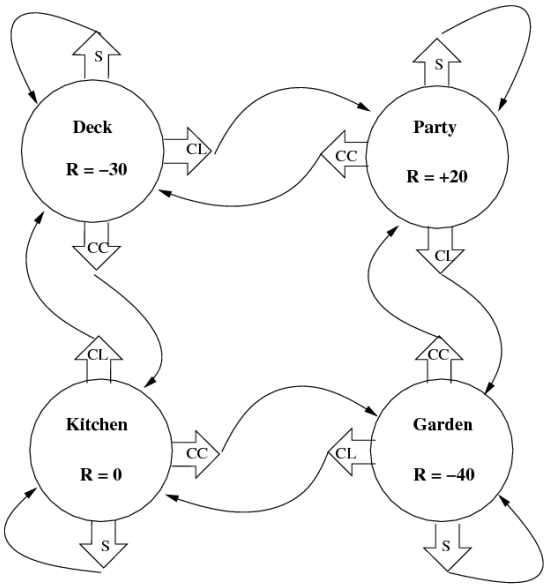
- We can explicitly run another round of iteration to see what happens to π.
- $U_1(s_1) = 10 + 0.5 \times U_1(s_1)$ therefore $U_1(s_1) = 20$
- $U_1(s_3) = 0 + 0.5 \times (0.5 \times U_1(s_3) + 0.5 \times U_1(s_1))$ therefore $U_1(s_3) = U_1(s_1)/3 = 20/3$
- $U_1(s_4) = -5 + 0.5 \times U_1(s_1) = 5$
- $U_1(s_2) = 0 + 0.5 \times U_1(s_3)$ therefore $U_1(s_2) = U_1(s_3)/2 = 10/3$

- $\pi_2(s_1) = \text{argmax}(U_1(s_1),U_1(s_2)) = \text{argmax}(20,0) = N$
- $\pi_2(s_2) = \text{argmax}(U_1(s_2),U_1(s_3)) = \text{argmax}(10/3,20/3) = E$
- $\pi_2(s_3) = \text{argmax}(0.5xU_1(s_1)+0.5xU_1(s_3),U_1(s_4)) = \text{argmax}(10+10/3,5) = N$
- $\pi_2(s_4) = \text{argmax}(U_1(s_1),U_1(s_4)) = \text{argmax}(20,5) = N$
- The policy has not changed from iteration 1 to iteration 2, therefore the optimal policy is $\pi^* = \pi_1$:

  $\pi^*(s_1) = \pi^*(s_3) = \pi^*(s_4) = N \qquad \pi^*(s_2) = E$

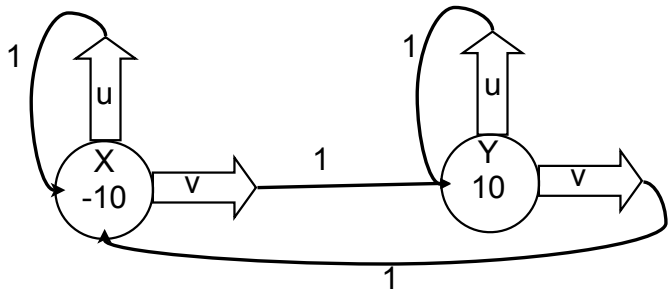
A faster way to do this

- We could have reached the same result without all of these painful calculations. Observe that π_1 is the same as π_o for the states 1,3,4 (action 'N'). Observe also that after applying an action from this policy from any one of these 3 states, it remains in one of the same three states 1,3,4. Therefore, the utility is unchanged and U_1 is the same as U_o for 1,3,4.
- π_1 is different from π_o for state 2 and it is easy to compute its utility U_2. Given that, it is easy to verify (one line in the previous page) that 'E' is still the optimal policy for state 2.
- The conclusion is that π_2 is the same as π_1 and we have converged. The main point is that we have save ourselves the pain of computing U_1 explicitly for 1,3,4.
- Important: In this kind of problem, it often useful to look carefully at the structure of the graph and the transition probabilities (especially when they are mostly 1!). Many of these problems can be solved at least partially "by inspection", thus saving painful computations (or, as a minimum, enabling a sanity check on the computations).



AI 04

Q-learning



- We run Q-learning on this MDP. We use the following strategy for exploration: We start at X and we alternate between choosing action u and action v, starting with action u.
- What are the values in the Q table after 5 iterations?
- Optimal policy from Q table? Optimal policy from the underlying MDP above?
- Call the learning rate α, and set the discount to $\gamma = 1$

Iteration 1: Reward = -10, action u, move from X to X

	X	Y
u	-10α	0
v	0	0

Iteration 2: Reward = -10, action v, move from X to Y

	X	Y
u	-10α	0
v	-10α	0

Direct application of the formula:
$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$

Iteration 3: Reward = 10, action u, move from Y to Y

	X	Y
u	-10α	10α
v	-10α	0

Iteration 4: Reward = 10, action v, move from Y to X

	X	Y
u	-10α	10α
v	-10α	$\alpha(10-10\alpha)$

Note that -10α = max (Q(X,u),Q(X,v))

Direct application of the formula:
$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$

Iteration 5: Reward = -10, action u, move from X to X

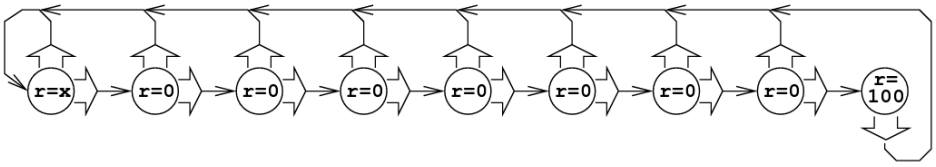
The complete expression for this entry is:
$-10\alpha + \alpha(-10 + 1x(-10\alpha) - (-10\alpha))$

	X	Y
u	-20α	10α
v	-10α	$\alpha(10-10\alpha)$

The optimal policy from the Q table is formed by choosing for each state the action that maximizes Q:
$\pi(X) = v$
$\pi(Y) = u$ (recall that we always choose α between 0 and 1)

This is also the optimal policy for the actual MDP (that fact should be obvious by inspection of the diagram)

Q-learning: Policy exploration



ES1 = random
ES2 = optimal
ES3 = in between

x	Using ES1	Using ES2	Using ES3
0	FIND-OPTIMAL-QUICK FIND-OPTIMAL-SLOW NEVER-FIND-OPTIMAL	FIND-OPTIMAL-QUICK FIND-OPTIMAL-SLOW NEVER-FIND-OPTIMAL	FIND-OPTIMAL-QUICK FIND-OPTIMAL-SLOW NEVER-FIND-OPTIMAL
-1	FIND-OPTIMAL-QUICK FIND-OPTIMAL-SLOW NEVER-FIND-OPTIMAL	FIND-OPTIMAL-QUICK FIND-OPTIMAL-SLOW NEVER-FIND-OPTIMAL	FIND-OPTIMAL-QUICK FIND-OPTIMAL-SLOW NEVER-FIND-OPTIMAL
+1	FIND-OPTIMAL-QUICK FIND-OPTIMAL-SLOW NEVER-FIND-OPTIMAL	FIND-OPTIMAL-QUICK FIND-OPTIMAL-SLOW NEVER-FIND-OPTIMAL	FIND-OPTIMAL-QUICK FIND-OPTIMAL-SLOW NEVER-FIND-OPTIMAL

AI 03