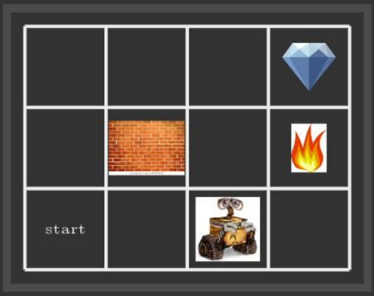
An MDP is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition function $T(s, a, s')$
  - Probability that a from s leads to s', i.e., $P(s'|s, a)$
  - Also called the model or the dynamics
- A reward function $R(s, a, s')$
  - Sometimes just $R(s)$ or $R(s')$
- A start state, s_0
- Maybe a terminal state



STATIONARY PREFERENCES

Assumption: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$

Then: there are only two ways to define utilities

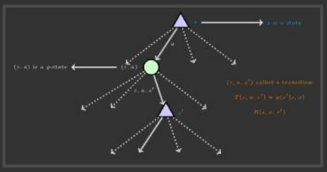
- Additive utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \ldots$$

- Discounted utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$$

The value (utility) of a state s: $V^*(s)$ = expected utility starting in s and acting optimally



The value (utility) of a q-state (s, a): $Q^*(s, a) =$ expected utility starting out having taken action a from state s and (thereafter) acting optimally

The optimal policy: $\pi^*(s)$ = optimal action from state s

VALUES OF STATES

Fundamental operation: compute the (expectimax) value of a state

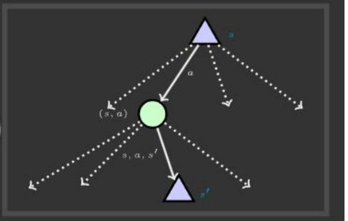
- Expected utility under optimal action
- Average sum of (discounted) rewards
- This is just what expectimax computed

Recursive definition of value:

$$V^*(s) = \max_a Q^*(s,a) \tag{1}$$

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

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \tag{2}$$



Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

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

Repeat until convergence

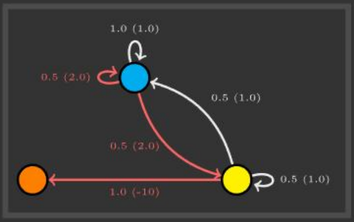
Complexity of each iteration: $\mathcal{O}(S^2 A)$

Theorem: will converge to unique optimal values

- Basic idea: approximations get refined towards optimal values
- Policy may converge long before values do

EXAMPLE VALUE ITERATION

	Cool	Warm	Overheated
V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



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

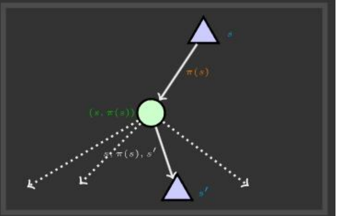
Assume no discount

POLICY EVALUATION

How do we calculate the V's for a fixed policy π?

Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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



Efficiency: $\mathcal{O}(S^2)$ per iteration

Idea 2: Without the maxes, the Bellman equations are just a linear system

- Solve with Matlab (or your favorite linear system solver)

COMPUTING VALUES FROM ACTIONS

Imagine we have the optimal values $V^*(s)$

How should we act?

- It is not obvious

We need to do a mini-expectimax (one step)

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

This is called policy extraction, since it gets the policy implied by the values.

COMPUTING ACTIONS FROM Q-VALUES

Imagine we have the optimal q-values: $Q^*(s,a)$

How should we act?

- Completely trivial to decide !!

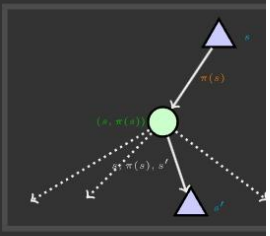
$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s,a)$$

Important lesson: actions are easier to select from q-values than values !!

PROBLEMS WITH VALUE ITERATION

Value iteration repeats the Bellman updates:

$$V_{k+1}^{*} = \max_{a} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k^{*}(s')]$$



Problem 1: It is slow – $\mathcal{O}(S^2A)$ per iteration

Problem 2: The "max" at each state rarely changes

Problem 3: The policy often converges long before the values

Alternative approach for optimal values:

- Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities) until convergence
- Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal) utilities as future values
- Repeat steps until policy converges

This is policy iteration

- It is still optimal !!
- Can converge (much) faster under some conditions

POLICY ITERATION

Evaluation: For fixed current policy π, find values with policy evaluation:

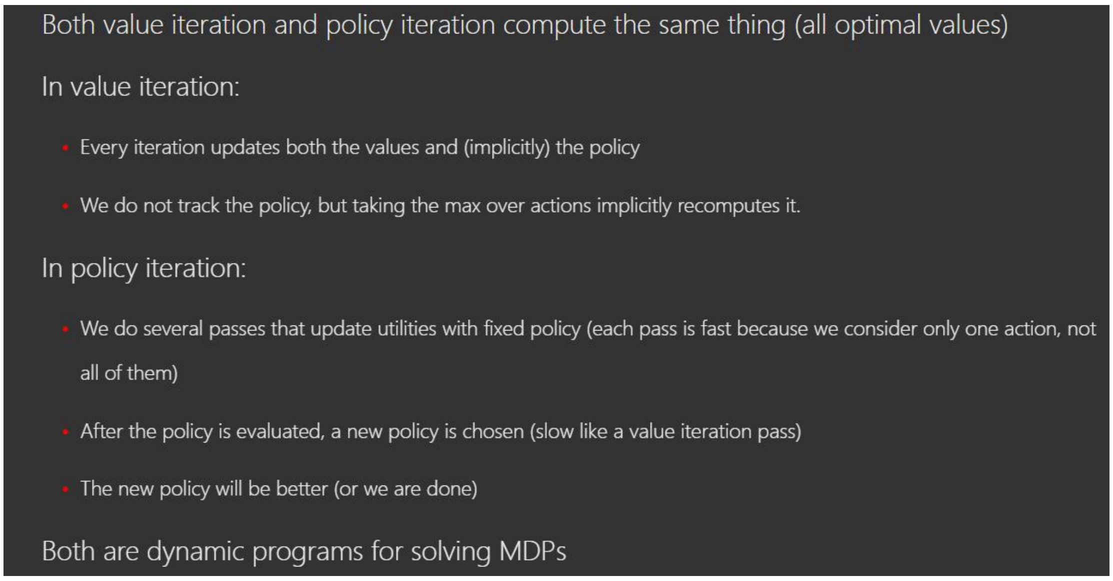
- Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) = \sum_{s'} T(s,\pi_i(s),s')[R(s,\pi_i(s),s') + \gamma V_k^{\pi_i}(s')]$$

Improvement: For fixed values, get a better policy using policy extraction

- One-step look-ahead:

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$$



Both value iteration and policy iteration compute the same thing (all optimal values)

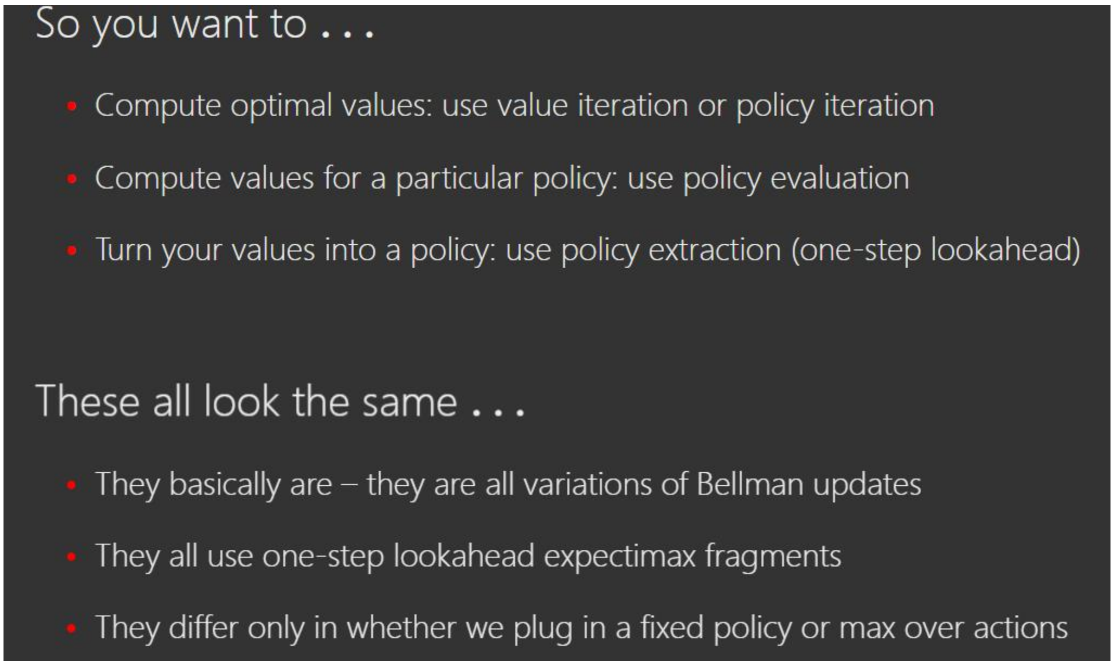
In value iteration:

- Every iteration updates both the values and (implicitly) the policy
- We do not track the policy, but taking the max over actions implicitly recomputes it.

In policy iteration:

- We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
- After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
- The new policy will be better (or we are done)

Both are dynamic programs for solving MDPs

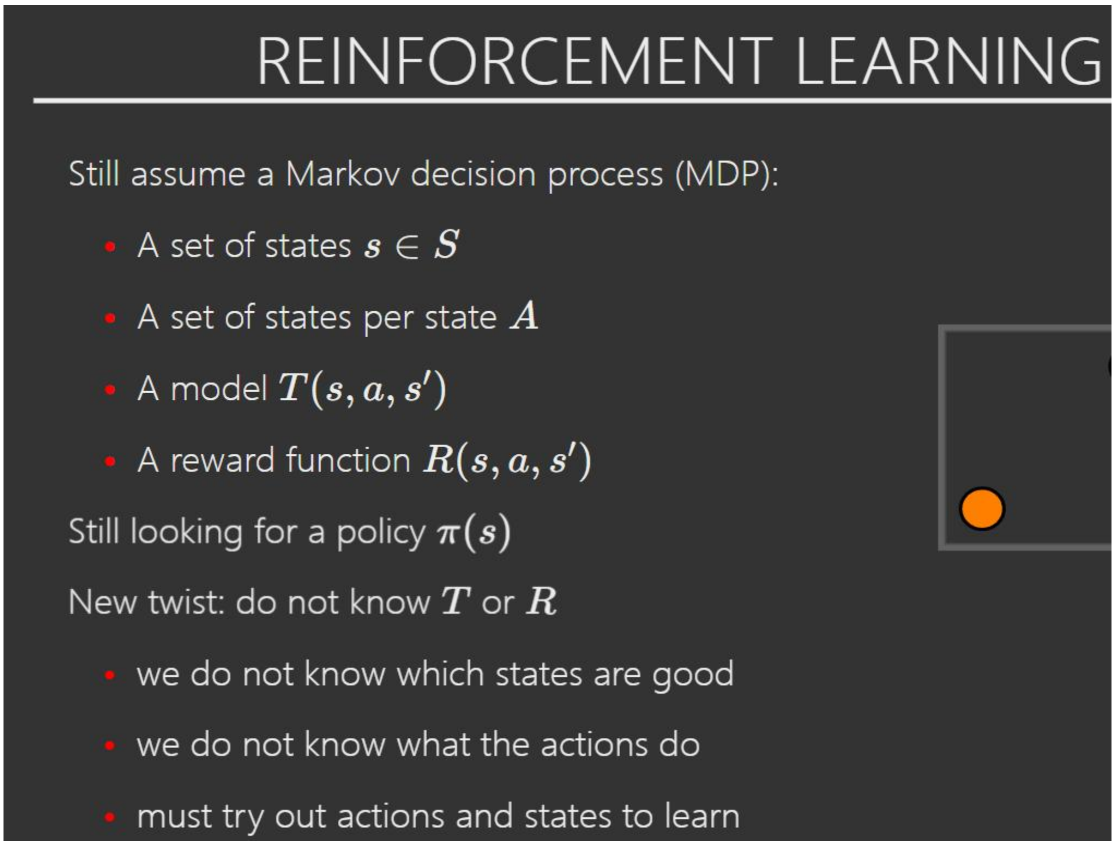


So you want to . . .

- Compute optimal values: use value iteration or policy iteration
- Compute values for a particular policy: use policy evaluation
- Turn your values into a policy: use policy extraction (one-step lookahead)

These all look the same . . .

- They basically are – they are all variations of Bellman updates
- They all use one-step lookahead expectimax fragments
- They differ only in whether we plug in a fixed policy or max over actions



REINFORCEMENT LEARNING

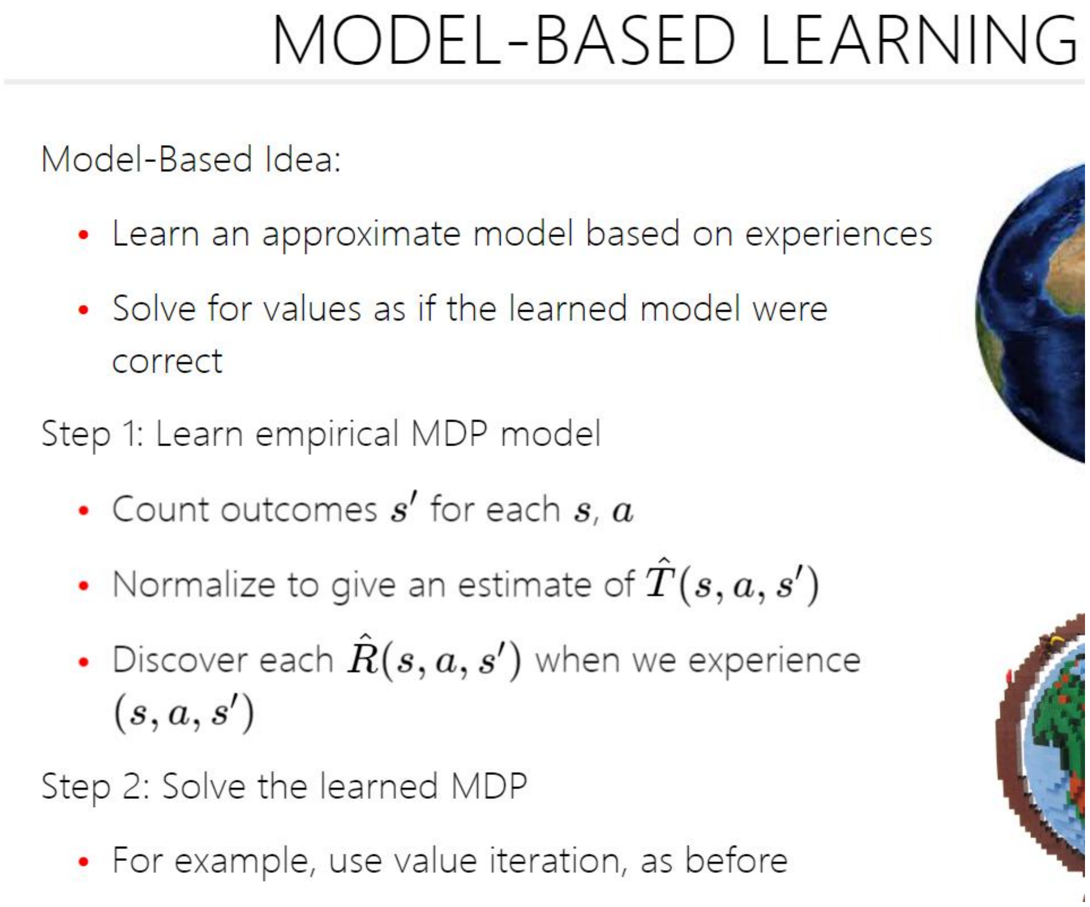
Still assume a Markov decision process (MDP):

- A set of states $s \in S$
- A set of states per state A
- A model $T(s, a, s')$
- A reward function $R(s, a, s')$

Still looking for a policy $\pi(s)$

New twist: do not know T or R

- we do not know which states are good
- we do not know what the actions do
- must try out actions and states to learn



MODEL-BASED LEARNING

Model-Based Idea:

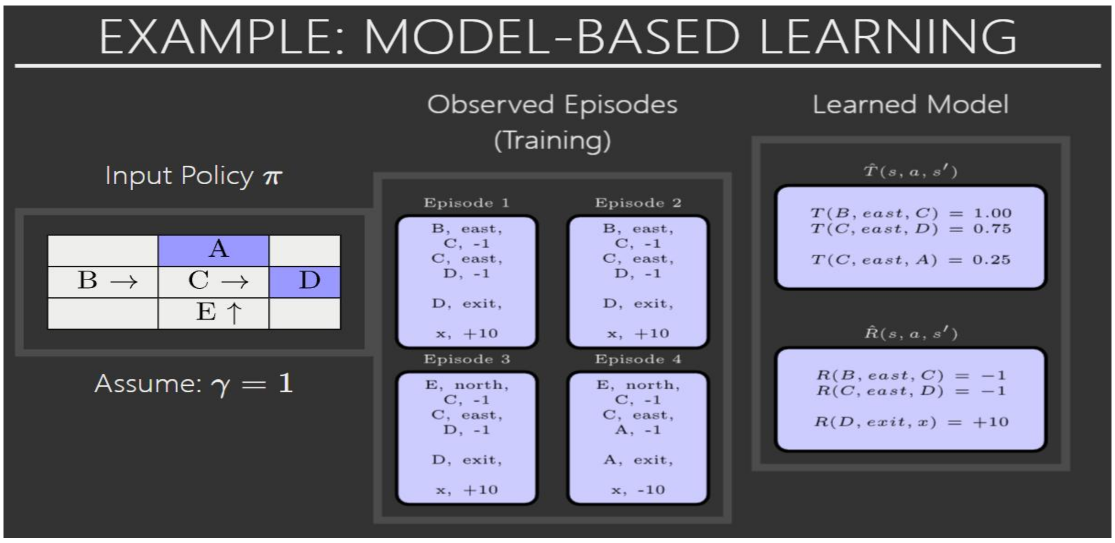
- Learn an approximate model based on experiences
- Solve for values as if the learned model were correct

Step 1: Learn empirical MDP model

- Count outcomes s' for each s, 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, as before



EXAMPLE: MODEL-BASED LEARNING

Input Policy π

	A	
B $\rightarrow$	C $\rightarrow$	D
	E $\uparrow$	

Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1	Episode 2
B, east, C, -1 C, east, D, -1 D, exit, x, +10	B, east, C, -1 C, east, D, -1 D, exit, x, +10

Episode 3	Episode 4
E, north, C, -1 C, east, D, -1 D, exit, x, +10	E, north, C, -1 C, east, A, -1 A, exit, x, -10

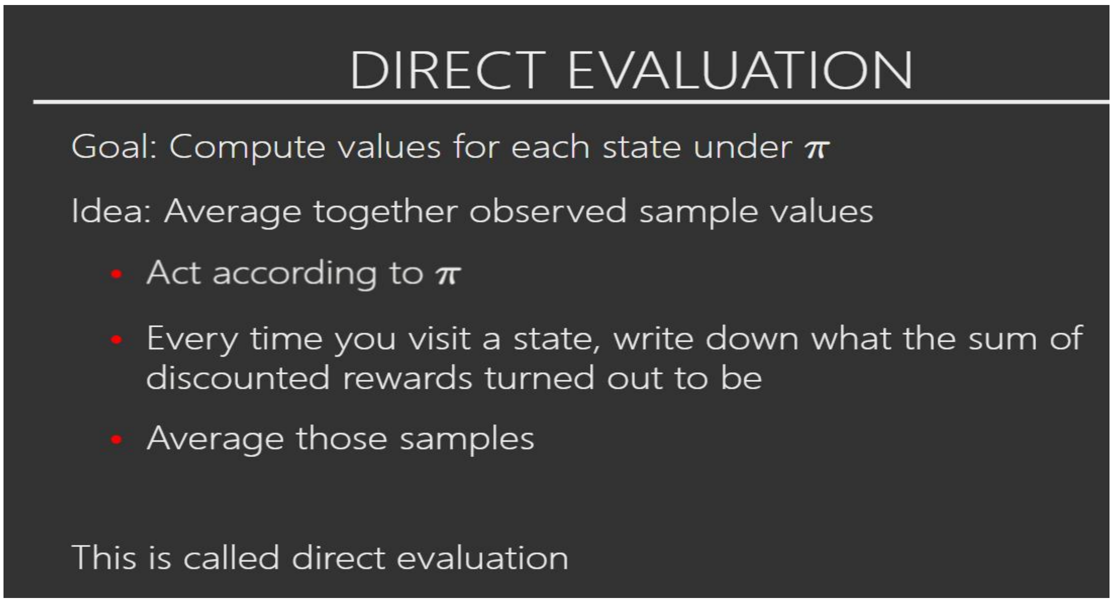
Learned Model

$\hat{T}(s, a, s')$

$T(B, east, C) = 1.00$
$T(C, east, D) = 0.75$
$T(C, east, A) = 0.25$

$\hat{R}(s, a, s')$

$R(B, east, C) = -1$
$R(C, east, D) = -1$
$R(D, exit, x) = +10$



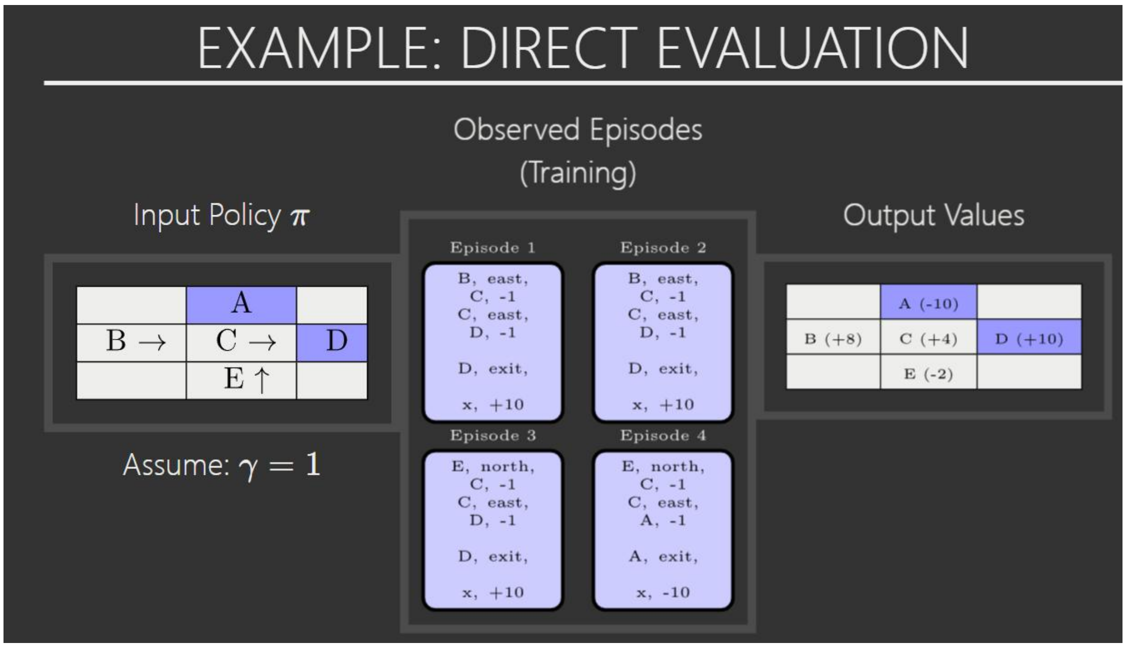
DIRECT EVALUATION

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



EXAMPLE: DIRECT EVALUATION

Input Policy π

	A	
B $\rightarrow$	C $\rightarrow$	D
	E $\uparrow$	

Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1	Episode 2
B, east, C, -1 C, east, D, -1 D, exit, x, +10	B, east, C, -1 C, east, D, -1 D, exit, x, +10

Episode 3	Episode 4
E, north, C, -1 C, east, D, -1 D, exit, x, +10	E, north, C, -1 C, east, A, -1 A, exit, x, -10

Output Values

	A (-10)	
B (+8)	C (+4)	D (+10)
	E (-2)	

PROBLEMS WITH DIRECT EVALUATION

What is good about direct evaluation?

- It is easy to understand
- It does not require any knowledge of T, R
- It eventually computes the correct average values, using just sample transitions

What is bad about it?

- It wastes information about state connections
- Each state must be learned separately
- So, it takes a long time to learn

Output Values

	A (-10)	
B (+8) →	C (+4) →	D (+10)
	E (-2)↑	

If B and E both go to C under this policy, how can their values be different?

SAMPLE-BASED POLICY EVALUATION

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

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

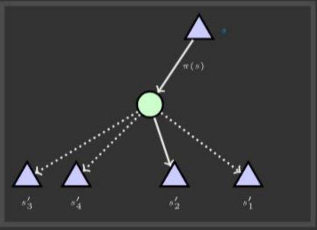
Idea: Take samples of outcomes s' (by doing the action) and average

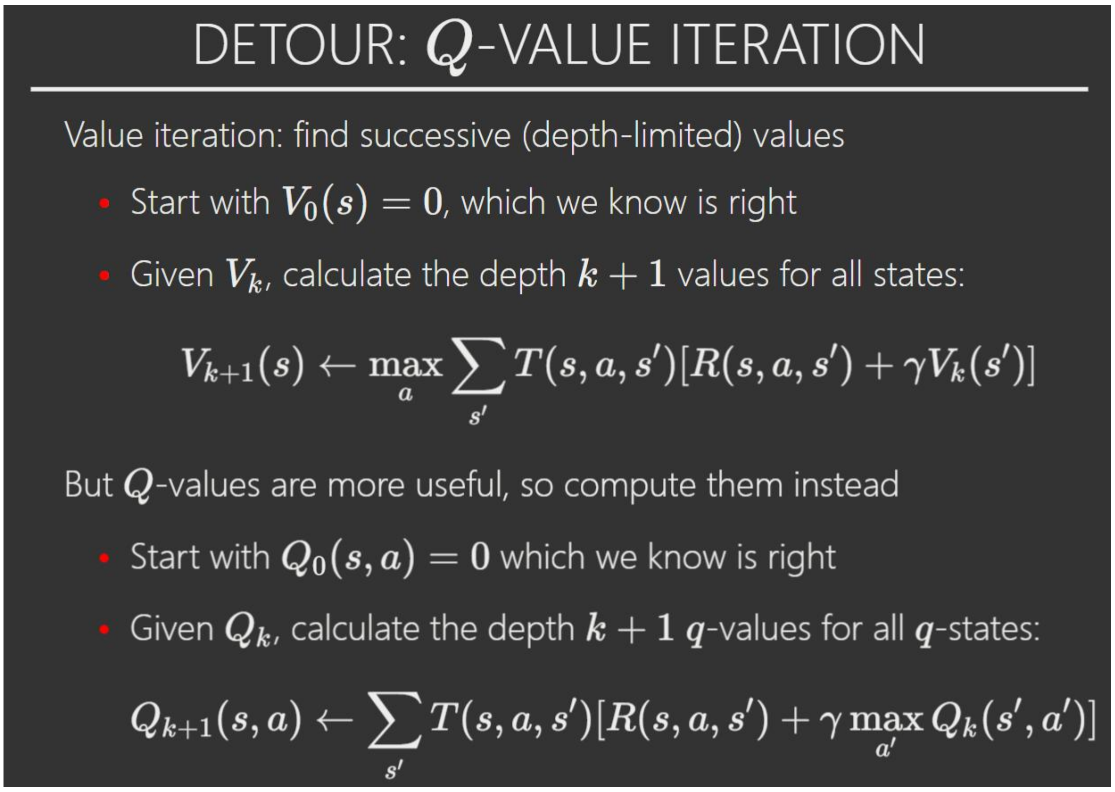
$$sample_1 = R(s,\pi(s),s'_1) + \gamma V^{\pi}_{k}(s'_1) \quad (4)$$

$$sample_2 = R(s,\pi(s),s'_2) + \gamma V^{\pi}_{k}(s'_2) \quad (5)$$

$$\dots$$

$$sample_n = R(s,\pi(s),s'_n) + \gamma V^{\pi}_{k}(s'_n) \quad (6)$$





DETOUR: Q-VALUE ITERATION

Value iteration: find successive (depth-limited) values

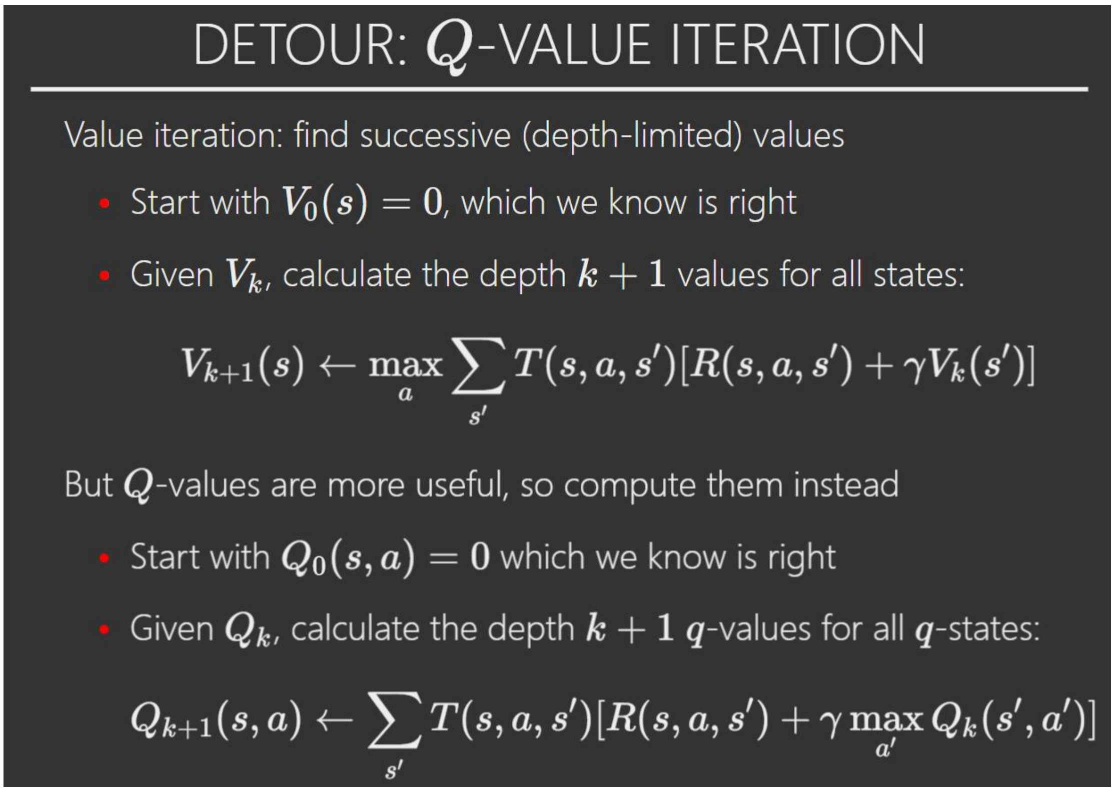
- Start with $V_0(s) = 0$, which we know is right
- Given V_k, calculate the depth $k+1$ values for all states:

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

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:

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



DETOUR: 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:

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

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:

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

Q-LEARNING

Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')] \tag{4}$$

Learn $Q(s,a)$ values as you go

- Receive a sample (s,a,s',r)
- Consider your old estimate: $Q(s,a)$
- Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a') \tag{5}$$

- Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)sample \tag{6}$$

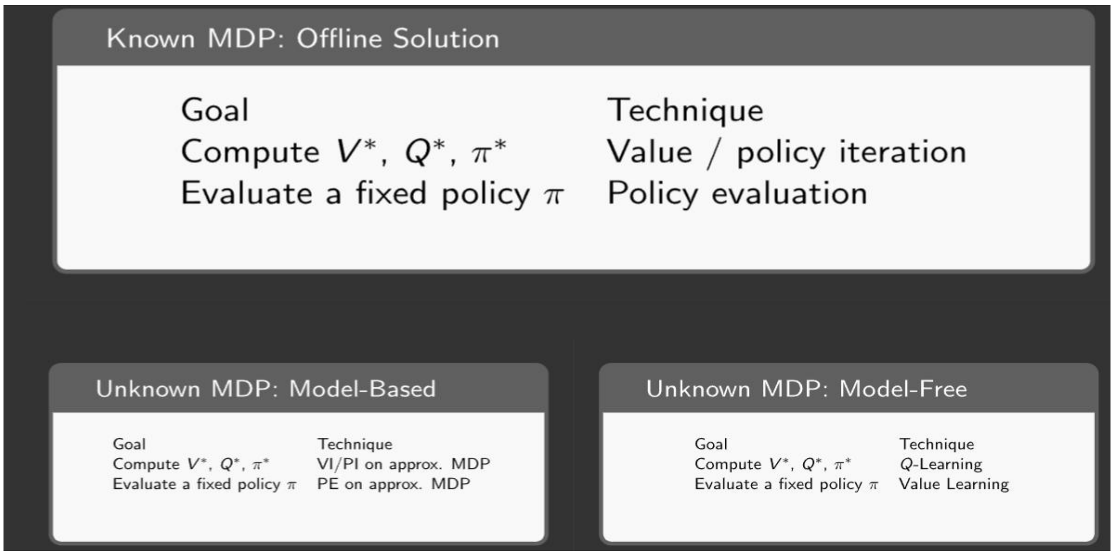
Q-LEARNING PROPERTIES

Amazing result: Q-learning converges to optimal policy -- even if you are acting sub-optimally.

This is called off-policy learning.

Caveats:

- You have 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.



Known MDP: Offline Solution

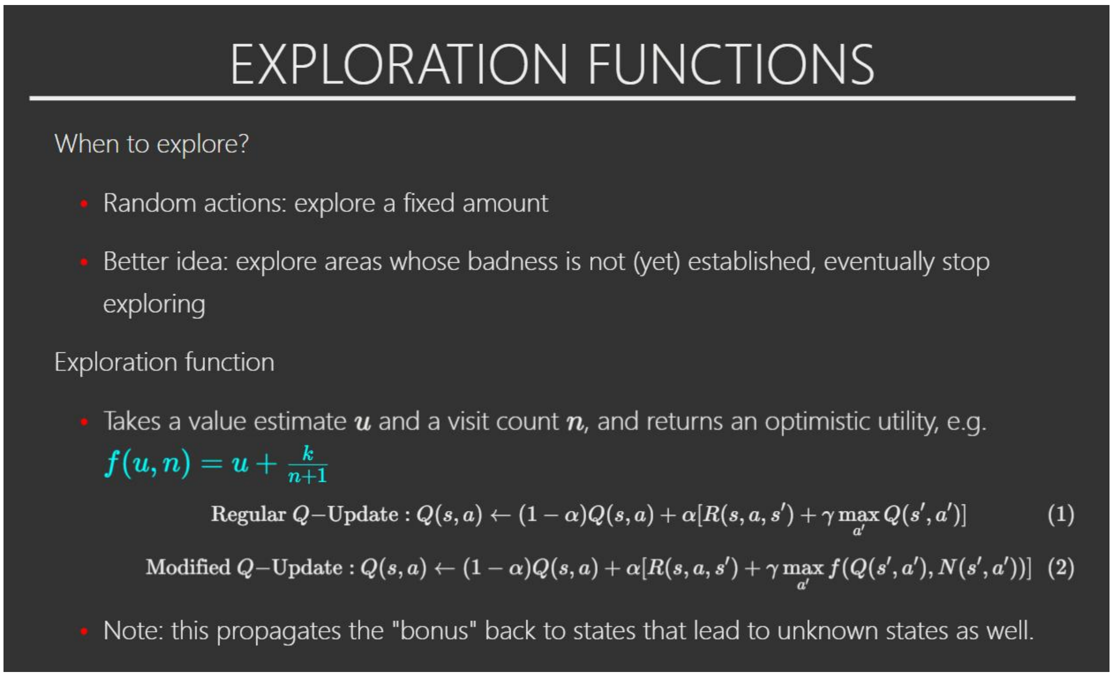
Goal	Technique
Compute V^*, Q^*, π^*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V^*, Q^*, π^*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

Goal	Technique
Compute V^*, Q^*, π^*	Q-Learning
Evaluate a fixed policy π	Value Learning



EXPLORATION FUNCTIONS

When to explore?

- Random actions: explore a fixed amount
- Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

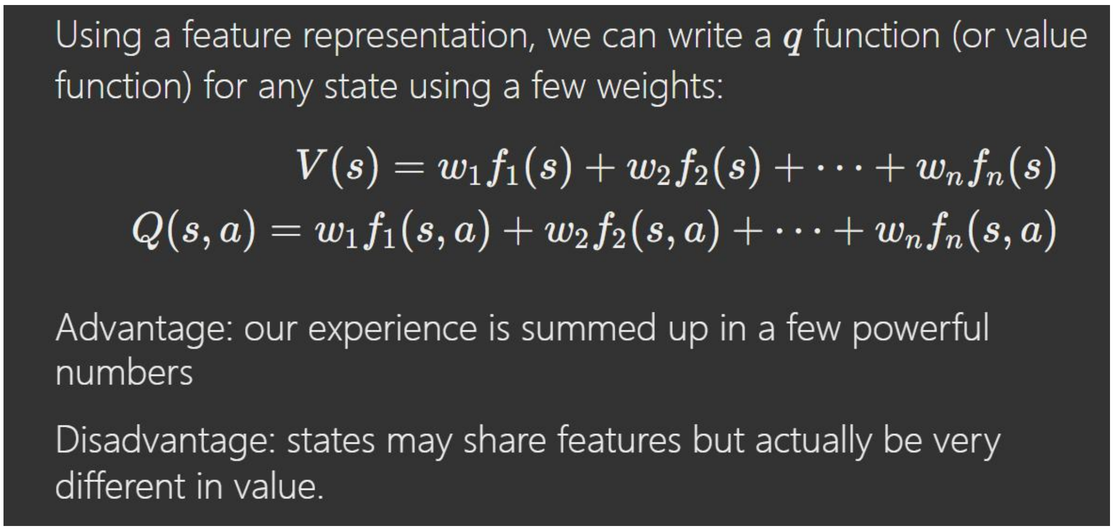
Exploration function

- Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u,n) = u + \frac{k}{n+1}$

$$\textbf{Regular } Q\textbf{−Update} : Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a')] \quad (1)$$

$$\textbf{Modified } Q\textbf{−Update} : Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))] \quad (2)$$

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

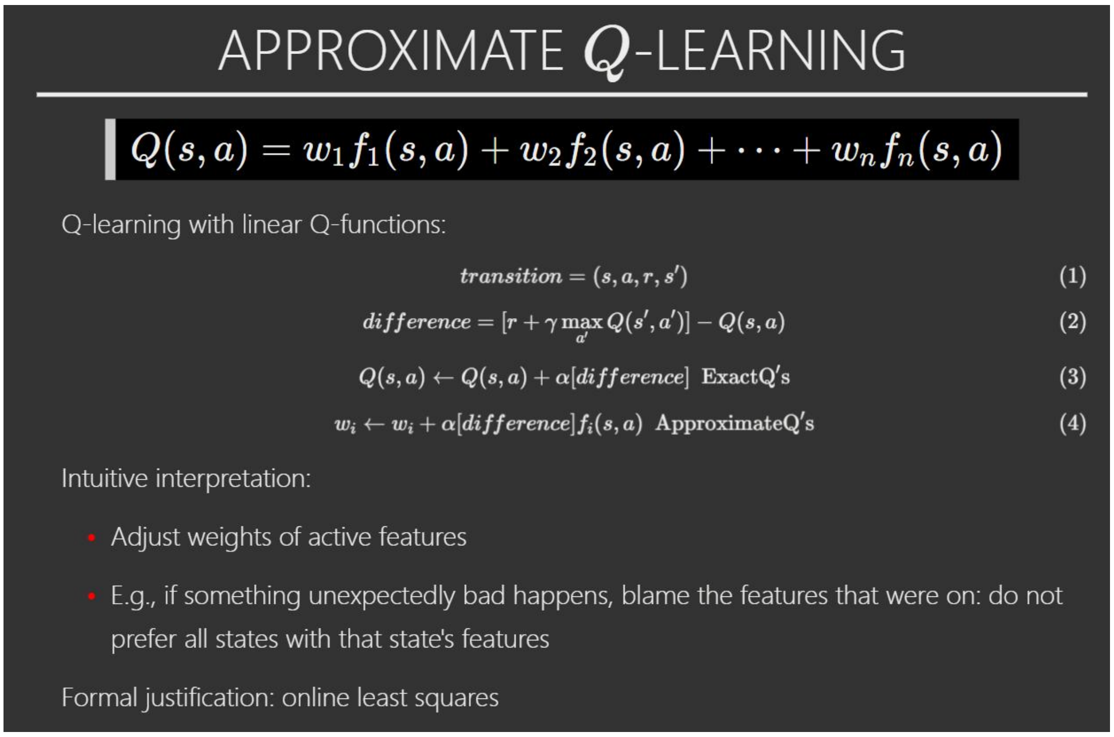


Using a feature representation, we can write a q function (or value function) for any state using a few weights:

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

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

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

Q-learning with linear Q-functions:

$$transition = (s, a, r, s') \tag{1}$$

$$difference = [r + \gamma \max_{a'} Q(s', a')] - Q(s,a) \tag{2}$$

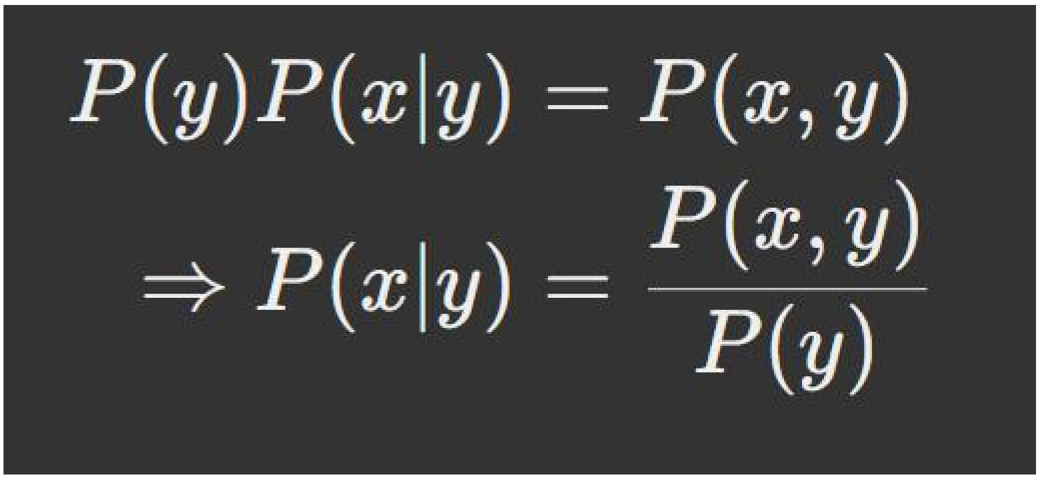
$$Q(s,a) \leftarrow Q(s,a) + \alpha[difference] \ \text{ExactQ's} \tag{3}$$

$$w_i \leftarrow w_i + \alpha[difference] f_i(s,a) \ \text{ApproximateQ's} \tag{4}$$

Intuitive interpretation:

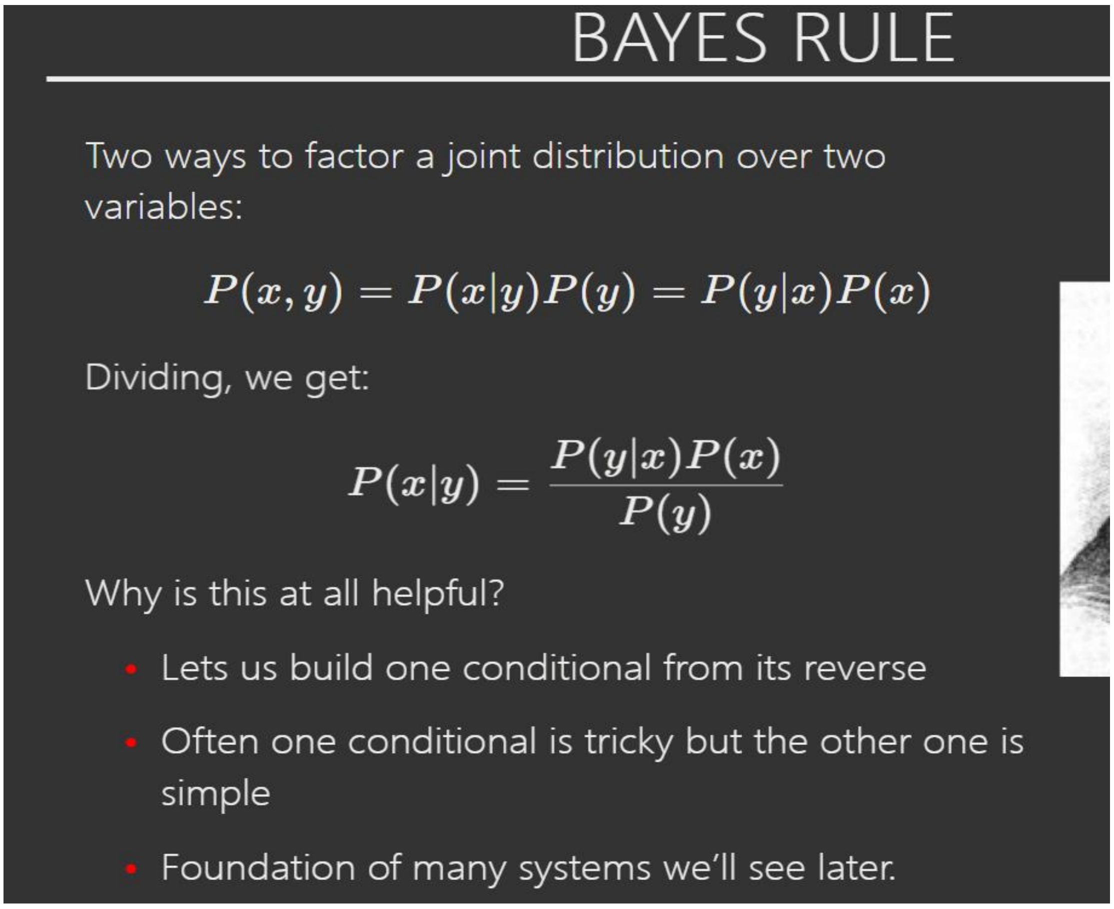
- Adjust weights of active features
- E.g., if something unexpectedly bad happens, blame the features that were on: do not prefer all states with that state's features

Formal justification: online least squares



$$P(y)P(x|y) = P(x,y)$$

$$\Rightarrow P(x|y) = \frac{P(x,y)}{P(y)}$$



BAYES RULE

Two ways to factor a joint distribution over two variables:

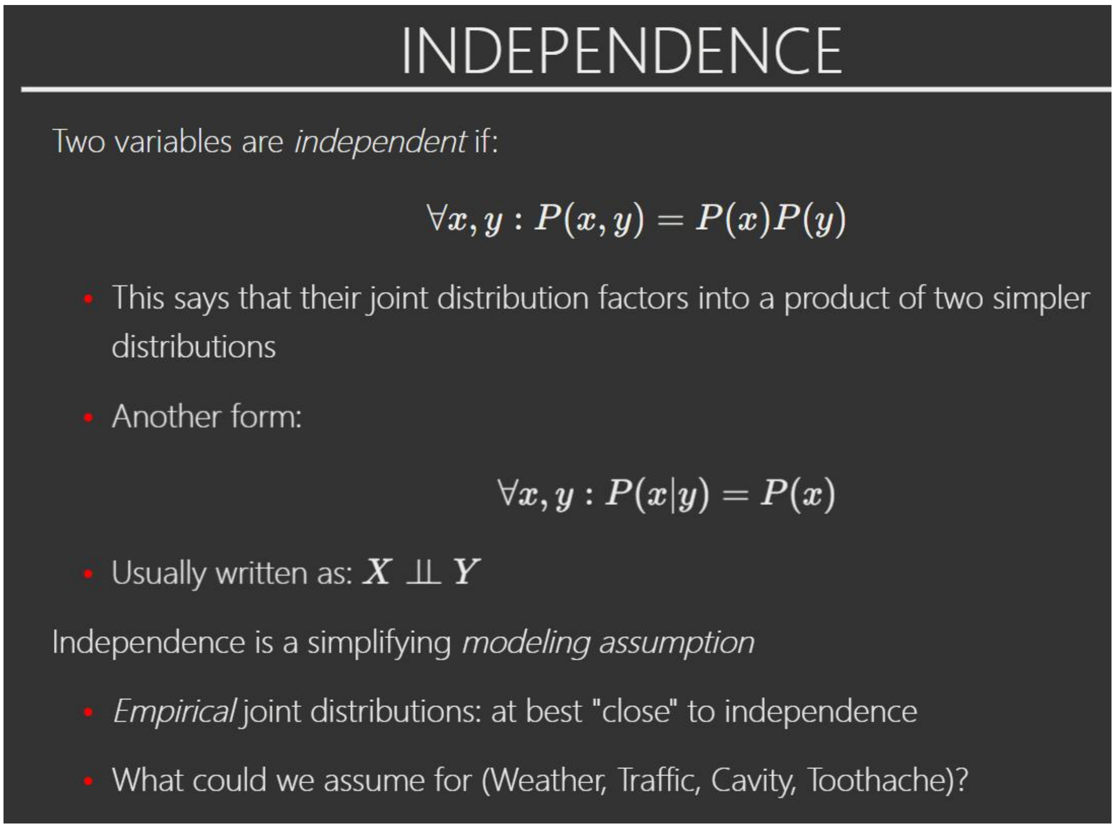
$$P(x,y) = P(x|y)P(y) = P(y|x)P(x)$$

Dividing, we get:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Why is this at all helpful?

- Lets us build one conditional from its reverse
- Often one conditional is tricky but the other one is simple
- Foundation of many systems we'll see later.



INDEPENDENCE

Two variables are *independent* if:

$$\forall x, y : P(x,y) = P(x)P(y)$$

- This says that their joint distribution factors into a product of two simpler distributions
- Another form:

$$\forall x, y : P(x|y) = P(x)$$

- Usually written as: $X \perp\!\!\!\perp Y$

Independence is a simplifying *modeling assumption*

- *Empirical* joint distributions: at best "close" to independence
- What could we assume for (Weather, Traffic, Cavity, Toothache)?

CONDITIONAL INDEPENDENCE

Unconditional (absolute) independence very rare (why?)

Conditional independence is our most basic and robust form of knowledge about uncertain environments.

X is conditionally independent of Y given Z i.e., $X \perp\!\!\!\perp Y | Z$

- iff:

$$\forall x, y, z: \; P(x, y|z) = P(x|z)P(y|z)$$

- or, equivalently, iff

$$\forall x, y, z: \; P(x|y, z) = P(x|z)$$

CONDITIONAL INDEPENDENCE AND CHAIN RULE

Chain rule: $P(X_1, X_2, \ldots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_2, X_1)\ldots$

Trivial decomposition:

$$P(T, R, U) = P(R)P(T|R)P(U|R, T)$$

With assumption of conditional independence:

$$P(T, R, U) = P(R)P(T|R)P(U|R)$$

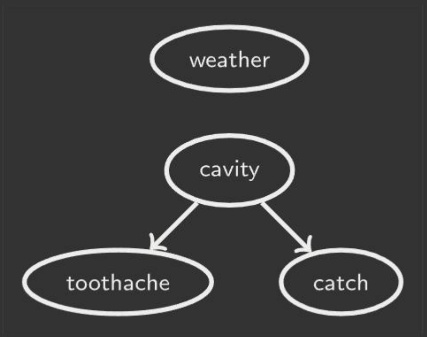
Bayes nets / graphical models help us express conditional independence assumptions

Nodes: variables (with domains)

- Can be assigned (observed) or unassigned (unobserved)

Arcs: interactions

- Similar to CSP constraints
- Indicate "direct influence" between variables
- Formally: encode conditional independence (more later)



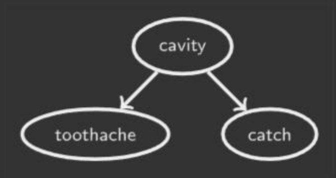
For now: imagine that arrows mean direct causation (in general, they don't.)

Bayes nets implicitly encode joint distributions

- As a product of local conditional distributions
- To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together:

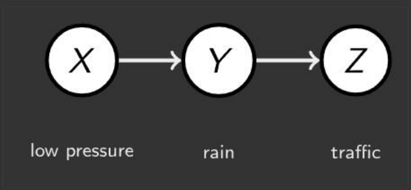
$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | parents(x_i))$$

- Example: $P(+cavity, +catch, -toothache)$



CAUSAL CHAINS

This configuration is a "causal chain"



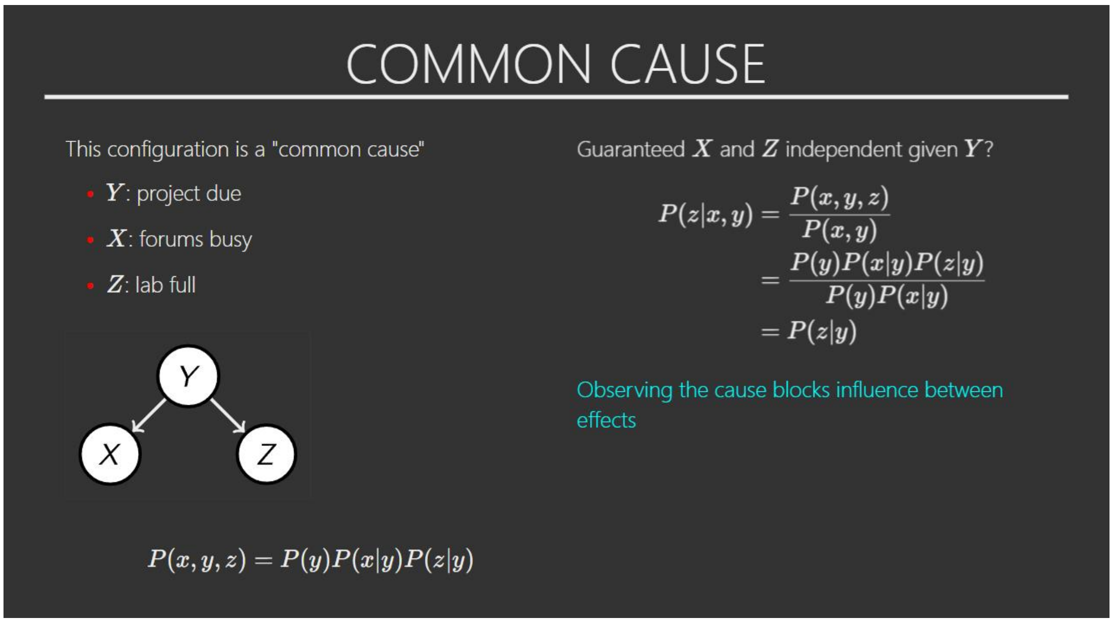
$$P(x, y, z) = P(x)P(y|x)P(z|y)$$

Guaranteed X independent of Z?

- Set CPTs for which X is not independent of Z. Sufficient to show this independence is not guaranteed.
- Example:
  - Low pressure causes rain causes traffic, high pressure causes no rain causes no traffic
  - In numbers:

$$P(+y| + x) = 1, P(-y| - x) = 1$$

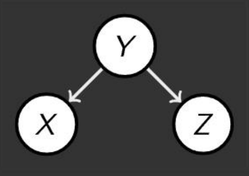
$$P(+z| + y) = 1, P(-z| - y) = 1$$



COMMON CAUSE

This configuration is a "common cause"

- Y: project due
- X: forums busy
- Z: lab full

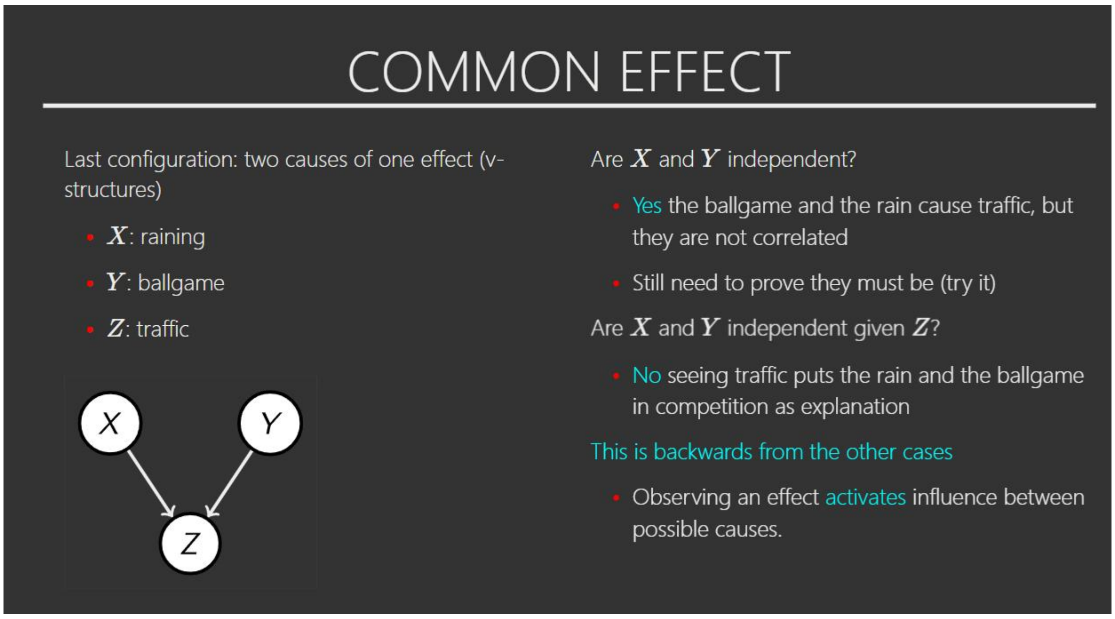


$$P(x,y,z) = P(y)P(x|y)P(z|y)$$

Guaranteed X and Z independent given Y?

$$\begin{aligned} P(z|x,y) &= \frac{P(x,y,z)}{P(x,y)} \\ &= \frac{P(y)P(x|y)P(z|y)}{P(y)P(x|y)} \\ &= P(z|y) \end{aligned}$$

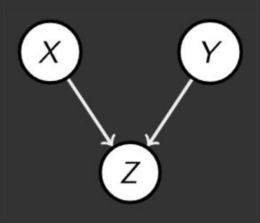
Observing the cause blocks influence between effects



COMMON EFFECT

Last configuration: two causes of one effect (v-structures)

- X: raining
- Y: ballgame
- Z: traffic



Are X and Y independent?

- Yes the ballgame and the rain cause traffic, but they are not correlated
- Still need to prove they must be (try it)

Are X and Y independent given Z?

- No seeing traffic puts the rain and the ballgame in competition as explanation

This is backwards from the other cases

- Observing an effect activates influence between possible causes.

CityU CS5491 Tutorial on MDP and RL

Markov Decision Processes Recap

- A **Markov Decision Process (MDP)** is similar to a state transition system. It has states, actions, a transition function $T(s, a, s')$ specifying the probability an agent ends up in state s' when he takes action a from state s, a distribution over start states, and possibly a set of terminal states. It also has a **reward function** $R(s, a, s')$, which represents the reward that an agent receives for performing action a in state s and ending up in state s'.
- A **q-state** is a (state, action) pair. From a state s, the agent chooses an action a, and then from that q-state (s, a), the (possibly nondeterministic) transition function chooses the resulting state s'.
- The utility of a state is not just the reward associated with that state, it also depends on what is going to happen in the future, so we need to compute utilities for sequences of rewards. We can define the utility of a sequence of rewards $[r_0, r_1, r_2, \dots]$ as $U([r_0, r_1, r_2, \dots]) = \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \dots$ for some $0 \le \gamma \le 1$. If $\gamma \ne 1$, the sum will converge, so the utilities will be finite; this is called temporal **discounting**.
- The **value** of a state s is the total expected future utility given that the agent is currently in state s. Similarly, the **q-value** of a q-state (s, a) is the total expected future utility given that the agent is currently in state s and has just taken action a. Under an optimal policy, the value and q-value are denoted $V^*(s)$ and $Q^*(s, a)$, respectively.
- The **Bellman equations** specify the relationship between $V^*(s)$ and $Q^*(s, a)$:

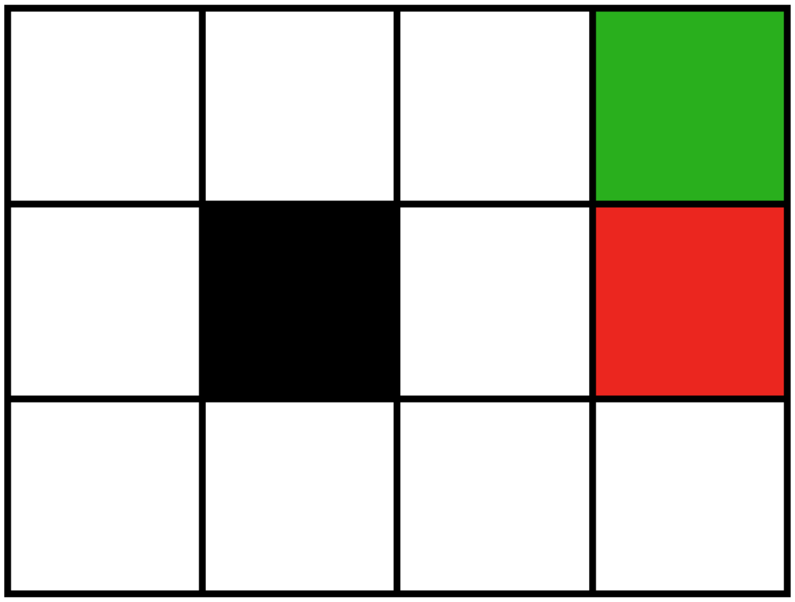
$$V^*(s) = \max_a Q^*(s, a)$$

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

- If we plug the second Bellman equation into the first one, we get a recursive definition of $V^*(s)$. We can approximate $V^*(s)$ by $V_k^*(s)$, the optimal value considering only the next k time steps: as $k \to \infty, V_k^* \to V^*(s)$. So, to compute V^*, we can use **value iteration**: initialize $V_0^*(s)$ to 0 for all s, and then given V_i^*, plug it into the Bellman equations to compute V_{i+1}^*, and repeat until convergence. **Policy iteration** is a similar process but updates the policy instead of the values. Often, the policy will converge before the values do.

MDP Exercise: Value Iteration

For today's class the environment we're exploring is a grid world, pictured below.



With this grid world, we define the following Markov decision process $\langle S, A, T, R, \gamma \rangle$:

- S: our states are represented as grid cells, each grid cell is one state in our world;
- A: the actions the robot can take in the world include moving *up*, *down*, *left*, and *right*; so there are 4 total actions that can be taken in this world;
- T: the robot moves as expected with action a from one state (s) to another (s') with probability 0.8. With probability 0.1, the robot moves to either left or right in perpendicular to action a. Here are some examples:
    - If a specifies that the robot should move to the *right*, with probability 0.8 the robot will move right. With probability 0.1 the robot will move up. And with probability 0.1 the robot will move down.
    - If a specifies that the robot should move to the *down*, with probability 0.8 the robot will move down. With probability 0.1 the robot will move right. And with probability 0.1 the robot will move left.
- R: Reward function only depends on the state of the world, i.e., $R(green) = +1$, $R(red) = -1$, and $R(\text{everywhere else}) = -2$.
- γ: for today's class we'll use a fixed discounting factor γ of value of 0.8.

Task today: With our grid world and the specified reward function shown above, compute the value for each state in the grid world using the Bellman equation and the value iteration approach. Before we start with value iteration, you can assume that the value for each state is 0.

To keep everyone on the same page, we're all going to follow the same set of trajectories:

> Note: To best keep track of each state, we will denote each state by it's row and column where the bottom left is the origin $[0, 0]$. For example, the green state at time k can be represented as $s_k = [2, 3]$

	Trajectory 1	Trajectory 2	Trajectory 3	Trajectory 4	Trajectory 5	Trajectory 6
s_0	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
V						
a_0	UP	UP	RIGHT	RIGHT	UP	RIGHT
s_1	$[1, 0]$	$[1, 0]$	$[0, 1]$	$[0, 1]$	$[1, 0]$	$[0, 1]$
V						
a_1	UP	UP	RIGHT	RIGHT	UP	RIGHT
s_2	$[2, 0]$	$[2, 0]$	$[0, 2]$	$[0, 2]$	$[2, 0]$	$[0, 2]$
V						
a_2	RIGHT	RIGHT	UP	RIGHT	RIGHT	UP
s_3	$[2, 1]$	$[2, 1]$	$[1, 2]$	$[0, 3]$	$[2, 1]$	$[1, 2]$
V						
a_3	RIGHT	RIGHT	RIGHT	UP	RIGHT	UP
s_4	$[2, 2]$	$[2, 2]$	$[1, 3]$	$[1, 3]$	$[2, 2]$	$[2, 2]$
V						
a_4	RIGHT	RIGHT			RIGHT	RIGHT
s_5	$[2, 3]$	$[2, 3]$			$[2, 3]$	$[2, 3]$
V						

Reinforcement Learning

- In **reinforcement learning (RL)** an agent gets feedback in the form of rewards, and he wants to learn a policy to maximize his expected utility. In passive RL, the policy is given, and the agent needs to learn the states' values from observation. In active RL, the agent has to choose actions and create a policy.

- In **temporal difference (TD) learning**, the agent estimates V directly from samples, without estimating T and R. It uses an exponentially-weighted moving average: $V_{new} = \alpha V_{sampled} + (1 - \alpha)V_{old}$ for some weight α, known as the *learning rate*.

- For constructing a policy, it is more useful to know Q than to know V. **Q-learning** is sample-based q-value iteration: each time the agent takes an action, he updates his Q values based on the Bellman equations.

- An agent must trade off between **exploration** (to learn a better policy) and **exploitation** (to get the most benefit out of the existing policy). In the **ϵ-greedy** method, with probability ϵ the agent chooses a random action, otherwise he acts according to his current policy. A better alternative is to have an "optimistic" utility function, which adds more utility to states that the agent knows less about.

- **Approximate Q-Learning**. In many contexts, there are too many states to explore all of them and store their information separately. One solution is to encode states (or q-states) as feature vectors and then estimate a state's value as a linear function of its feature vector; this enables the agent to generalize from examples to new states that it has not seen before.

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

Approximate Q-learning with linear Q-functions:

$$transition = (s, a, r, s')$$

$$\Delta = \left(r + \gamma \underset{a'}{max} Q(s', a')\right) - Q(s,a)$$
$$w_i \leftarrow w_i + \alpha[\Delta] f_i(s,a)$$

RL Exercise: Approximate Q-learning

A self-driving car needs to decide whether to Accelerate (**A**) or Brake (**B**) so as to drive to a location without hitting other cars. It receives a reward of +1 if the car moves and does not hit another car, 0 if it does not move, and −2 if it hits another car. The discount factor $\gamma = 1$.

We want to use Approximate Q-learning to learn a good driving policy for the car. The car has sensors that allow it to observe the distance (**D**) to the nearest object and the current speed (**S**). We decide to create a set of four features: f_{AD}, f_{AS}, f_{BD}, f_{BS}. The first two features are for action A and the second two features are for action B. Q values will be approximated by a linear combination of four features, with four weights (one for each feature): w_{AD}, w_{AS}, w_{BD}, w_{BS}.

Suppose that some learning has already happened such that we have $w_{AD} = 1$, but the other weights are all 0. The learning rate $\alpha = 0.5$. Below is the stream of data the car receives as it drives. Compute the weights after each step.

Observed Data	Weights after seeing data
	$w_{AD} = 1, w_{AS} = w_{BD} = w_{BS} = 0$
Initial Sensors: $D = 0, S = 2$ Action: A Reward: -2 Final Sensors: $D = 1, S = 0$	
Initial Sensors: $D = 1, S = 0$ Action: B Reward: 0 Final Sensors: $D = 1, S = 0$	

Given the learned weights, suppose that the sensors read $D = 1, S = 1$. Which action would be preferred?

CityU CS5491 Tutorial on MDP and RL

Markov Decision Processes Recap

- A **Markov Decision Process (MDP)** is similar to a state transition system. It has states, actions, a transition function $T(s, a, s')$ specifying the probability an agent ends up in state s' when he takes action a from state s, a distribution over start states, and possibly a set of terminal states. It also has a **reward function** $R(s, a, s')$, which represents the reward that an agent receives for performing action a in state s and ending up in state s'.

- A **q-state** is a (state, action) pair. From a state s, the agent chooses an action a, and then from that q-state (s, a), the (possibly nondeterministic) transition function chooses the resulting state s'.

- The utility of a state is not just the reward associated with that state, it also depends on what is going to happen in the future, so we need to compute utilities for sequences of rewards. We can define the utility of a sequence of rewards $[r_0, r_1, r_2, \ldots]$ as $U([r_0, r_1, r_2, \ldots]) = \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \ldots$ for some $0 \leq \gamma \leq 1$. If $\gamma \neq 1$, the sum will converge, so the utilities will be finite; this is called temporal **discounting**.

- The **value** of a state s is the total expected future utility given that the agent is currently in state s. Similarly, the **q-value** of a q-state (s, a) is the total expected future utility given that the agent is currently in state s and has just taken action a. Under an optimal policy, the value and q-value are denoted $V^*(s)$ and $Q^*(s, a)$, respectively.

- The **Bellman equations** specify the relationship between $V^*(s)$ and $Q^*(s, a)$:

$$V^*(s) = \max_a Q^*(s, a)$$

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

- If we plug the second Bellman equation into the first one, we get a recursive definition of $V^*(s)$. We can approximate $V^*(s)$ by $V_k^*(s)$, the optimal value considering only the next k time steps: as $k \to \infty, V_k^* \to V^*(s)$. So, to compute V^*, we can use **value iteration**: initialize $V_0^*(s)$ to 0 for all s, and then given V_i^*, plug it into the Bellman equations to compute V_{i+1}^*, and repeat until convergence. **Policy iteration** is a similar process but updates the policy instead of the values. Often, the policy will converge before the values do.

Reinforcement Learning

- In **reinforcement learning (RL)** an agent gets feedback in the form of rewards, and he wants to learn a policy to maximize his expected utility. In passive RL, the policy is given, and the agent needs to learn the states' values from observation. In active RL, the agent has to choose actions and create a policy.
- In **temporal difference (TD) learning**, the agent estimates V directly from samples, without estimating T and R. It uses an exponentially-weighted moving average: $V_{new} = \alpha V_{sampled} + (1 - \alpha)V_{old}$ for some weight α, known as the *learning rate*.
- For constructing a policy, it is more useful to know Q than to know V. **Q-learning** is sample-based q-value iteration: each time the agent takes an action, he updates his Q values based on the Bellman equations.
- An agent must trade off between **exploration** (to learn a better policy) and **exploitation** (to get the most benefit out of the existing policy). In the **ϵ-greedy** method, with probability ϵ the agent chooses a random action, otherwise he acts according to his current policy. A better alternative is to have an "optimistic" utility function, which adds more utility to states that the agent knows less about.
- **Approximate Q-Learning**. In many contexts, there are too many states to explore all of them and store their information separately. One solution is to encode states (or q-states) as feature vectors and then estimate a state's value as a linear function of its feature vector; this enables the agent to generalize from examples to new states that it has not seen before.

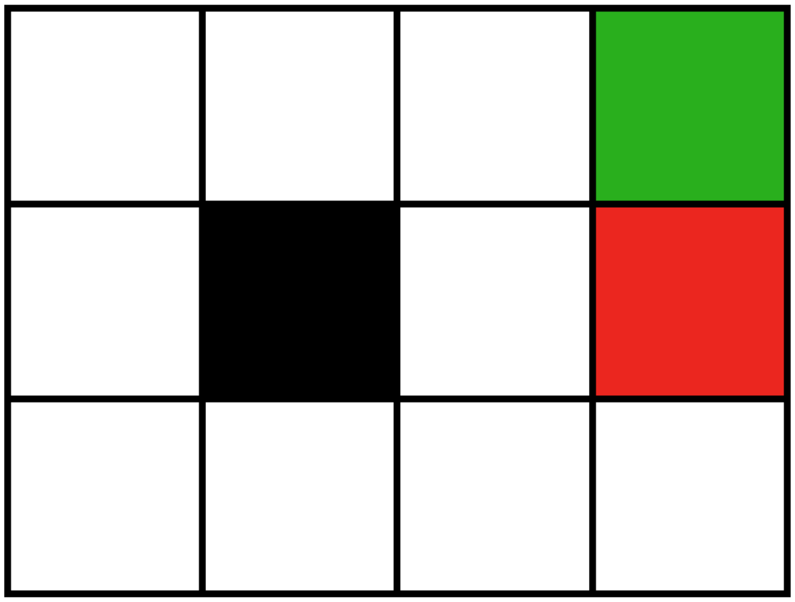
$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)$$

Approximate Q-learning with linear Q-functions:

$$transition = (s, a, r, s')$$
$$\Delta = \left(r + \gamma \underset{a'}{max} Q(s', a')\right) - Q(s,a)$$
$$w_i \leftarrow w_i + \alpha[\Delta] f_i(s,a)$$

1 MDP Exercise: Value Iteration

For today's class the environment we're exploring is a grid world, pictured below.



With this grid world, we define the following Markov decision process $\langle S, A, T, R, \gamma \rangle$:

- S: our states are represented as grid cells, each grid cell is one state in our world;
- A: the actions the robot can take in the world include moving *up*, *down*, *left*, and *right*; so there are 4 total actions that can be taken in this world;
- T: the robot moves as expected with action a from one state (s) to another (s') with probability 0.8. With probability 0.1, the robot moves to either left or right in perpendicular to action a. Here are some examples:
    - If a specifies that the robot should move to the *right*, with probability 0.8 the robot will move right. With probability 0.1 the robot will move up. And with probability 0.1 the robot will move down.
    - If a specifies that the robot should move to the *down*, with probability 0.8 the robot will move down. With probability 0.1 the robot will move right. And with probability 0.1 the robot will move left.
- R: Reward function only depends on the state of the world, i.e., $R(green) = +1$, $R(red) = -1$, and $R(\text{everywhere else}) = -2$.
- γ: for today's class we'll use a fixed discounting factor γ of value of 0.8.

Task today: With our grid world and the specified reward function shown above, compute the value for each state in the grid world using the Bellman equation and the value iteration approach. Before we start with value iteration, you can assume that the value for each state is 0.

To keep everyone on the same page, we're all going to follow the same set of trajectories:

> Note: To best keep track of each state, we will denote each state by it's row and column where the bottom left is the origin $[0,0]$. For example, the green state at time k can be represented as $s_k = [2,3]$

	Trajectory 1	Trajectory 2	Trajectory 3	Trajectory 4	Trajectory 5	Trajectory 6
s_0	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
V	$V([0,0]) = -2$	$V([0,0]) = -2.32$	$V([0,0]) = -2.47$	$V([0,0]) = -3.77$	$V([0,0]) = -4.89$	$V([0,0]) = -5.09$
a_0	UP	UP	RIGHT	RIGHT	UP	RIGHT
s_1	$[1,0]$	$[1,0]$	$[0,1]$	$[0,1]$	$[1,0]$	$[0,1]$
V	$V([1,0]) = -2$	$V([1,0]) = -3.6$	$V([0,1]) = -2$	$V([0,1]) = -3.6$	$V([1,0]) = -4.88$	$V([0,1]) = -4.07$
a_1	UP	UP	RIGHT	RIGHT	UP	RIGHT
s_2	$[2,0]$	$[2,0]$	$[0,2]$	$[0,2]$	$[2,0]$	$[0,2]$
V	$V([2,0]) = -2$	$V([2,0]) = -3.6$	$V([0,2]) = -2$	$V([0,2]) = -2.34$	$V([2,0]) = -4.88$	$V([0,2]) = -3.70$
a_2	RIGHT	RIGHT	UP	RIGHT	RIGHT	UP
s_3	$[2,1]$	$[2,1]$	$[1,2]$	$[0,3]$	$[2,1]$	$[1,2]$
V	$V([2,1]) = -2$	$V([2,1]) = -3.6$	$V([1,2]) = -2.28$	$V([0,3]) = -2.08$	$V([2,1]) = -3.54$	$V([1,2]) = -3.06$
a_3	RIGHT	RIGHT	RIGHT	UP	RIGHT	UP
s_4	$[2,2]$	$[2,2]$	$[1,3]$	$[1,3]$	$[2,2]$	$[2,2]$
V	$V([2,2]) = -2$	$V([2,2]) = -1.52$	$V([1,3]) = -1$	$V([1,3]) = -1$	$V([2,2]) = -1.66$	$V([2,2]) = -1.74$
a_4	RIGHT	RIGHT			RIGHT	RIGHT
s_5	$[2,3]$	$[2,3]$			$[2,3]$	$[2,3]$
V	$V([2,3]) = 1$	$V([2,3]) = 1$			$V([2,3]) = 1$	$V([2,3]) = 1$

- Trajectory 1
  - $V([2,3]) = 1$
    - We can determine this value because $R(s_t, a_t) = +1$ for being in the green state and once we get into the state, the world ends (we cannot transition into any further states), meaning that for all a_t, $\left(\gamma \sum_{s' \in S} T(s,a,s')V(s')\right) = 0$.
  - $V(\text{all other states}) = -2$
    - We can determine this value because $R(s_t, a_t) = -2$ and all states initially start out with a value of 0 for all states until after we go through the first iteration.
- Trajectory 2
  - $V([0,0]) = -2 + \gamma(0.8 \cdot 0 + 0.1 \cdot -2 + 0.1 \cdot -2) = -2 + \gamma(-0.4) = -2 + 0.8 \cdot -0.4 = -2.32$

    * The optimal action in this case would be a move to the RIGHT since the value of state $[0, 1]$ is currently 0 and the value of the state $[1, 0]$ is currently -2.

- $V([1,0]) == -2 + \gamma(0.8 \cdot -2 + 0.1 \cdot -2 + 0.1 \cdot -2) = -2 + \gamma(-2) = -2 + 0.8 \cdot -2 = -3.6$
    * The optimal action in this case would be a move UP since the value of state $[2, 0]$ is currently -2 and the value of the state $[0, 0]$ is now -2.32.
- . . .

RL Exercise: Approximate Q-learning

A self-driving car needs to decide whether to Accelerate (**A**) or Brake (**B**) so as to drive to a location without hitting other cars. It receives a reward of +1 if the car moves and does not hit another car, 0 if it does not move, and −2 if it hits another car. The discount factor $\gamma = 1$.

We want to use Approximate Q-learning to learn a good driving policy for the car. The car has sensors that allow it to observe the distance (**D**) to the nearest object and the current speed (**S**). We decide to create a set of four features: f_{AD}, f_{AS}, f_{BD}, f_{BS}. The first two features are for action A and the second two features are for action B. Q values will be approximated by a linear combination of four features, with four weights (one for each feature): $w_{AD}, w_{AS}, w_{BD}, w_{BS}$.

Suppose that some learning has already happened such that we have $w_{AD} = 1$, but the other weights are all 0. The learning rate $\alpha = 0.5$. Below is the stream of data the car receives as it drives. Compute the weights after each step.

Observed Data	Weights after seeing data
	$w_{AD} = 1, w_{AS} = w_{BD} = w_{BS} = 0$
Initial Sensors: $D = 0, S = 2$ Action: A Reward: -2 Final Sensors: $D = 1, S = 0$	$s[D = 0, S = 2]$, taken action A, $f_{AD} = 0$, $f_{AS} = 2$, $f_{BD} = f_{BS} = 0$ $Q(s, \text{A}) = w_{AD} \cdot f_{AD} + w_{AS} \cdot f_{AS} + w_{BD} \cdot f_{BD} + w_{BS} \cdot f_{BS}$ $= 1 \cdot 0 + 0 \cdot 2 + 0 \cdot 0 + 0 \cdot 0 = 0$ $s'[D = 1, S = 0]$, taken action A, $f_{AD} = 1$, $f_{AS} = 0$, $f_{BD} = f_{BS} = 0$ $Q(s', \text{A}) = 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 1$ $s'[D = 1, S = 0]$, taken action B, $f_{AD} = f_{AS} = 0$, $f_{BD} = 1$, $f_{BS} = 0$ $Q(s', \text{B}) = 1 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 0$ $\Delta = (-2 + 1.0 \cdot max(Q(s', \text{A}), Q(s', \text{B})) - 0 = -2 + 1.0(1) - 0 = -1$ $w_{AD} \leftarrow w_{AD} + 0.5 \cdot \Delta \cdot f_{AD}(s, A) = 1 + 0.5 \cdot (-1) \cdot 0 = \boxed{1}$ $w_{AS} \leftarrow w_{AS} + 0.5 \cdot \Delta \cdot f_{AS}(s, A) = 0 + 0.5 \cdot (-1) \cdot 2 = \boxed{-1}$ $w_{BD} = \boxed{0}, w_{BS} = \boxed{0}$
Initial Sensors: $D = 1, S = 0$ Action: B Reward: 0 Final Sensors: $D = 1, S = 0$	$s[D = 1, S = 0]$, taken action B, $f_{AD} = f_{AS} = 0$, $f_{BD} = 1$, $f_{BS} = 0$ $Q(s, \text{B}) = w_{AD} \cdot f_{AD} + w_{AS} \cdot f_{AS} + w_{BD} \cdot f_{BD} + w_{BS} \cdot f_{BS}$ $= 1 \cdot 0 + (-1) \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 0$ $s'[D = 1, S = 0]$, taken action A, $f_{AD} = 1$, $f_{AS} = 0$, $f_{BD} = f_{BS} = 0$ $Q(s', \text{A}) = 1 \cdot 1 + (-1) \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 1$ $s'[D = 1, S = 0]$, taken action B, $f_{AD} = f_{AS} = 0$, $f_{BD} = 1$, $f_{BS} = 0$ $Q(s', \text{B}) = 1 \cdot 0 + (-1) \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 0$ $\Delta = (0 + 1.0 \cdot max(Q(s', \text{A}), Q(s', \text{B})) - 0 = 0 + 1.0(1) - 0 = 1$ $w_{AD} = \boxed{1}, w_{AS} = \boxed{-1}$ $w_{BD} \leftarrow w_{BD} + 0.5 \cdot \Delta \cdot f_{BD}(s, B) = 0 + 0.5 \cdot 1 \cdot 1 = \boxed{0.5}$ $w_{BS} \leftarrow w_{BS} + 0.5 \cdot \Delta \cdot f_{BS}(s, B) = 0 + 0.5 \cdot 1 \cdot 0 = \boxed{0}$

Given the learned weights, suppose that the sensors read $D = 1$, $S = 1$. Which action would be preferred?

- At state $s[D = 1, S = 1]$, if take action A, $Q(s, \text{A}) = 1 \cdot 1 + (-1) \cdot 1 + 0.5 \cdot 0 + 0 \cdot 0 = 0$
- At state $s[D = 1, S = 1]$, if take action B, $Q(s, \text{B}) = 1 \cdot 0 + (-1) \cdot 0 + 0.5 \cdot 1 + 0 \cdot 1 = \boxed{0.5}$

Hence, action B will be preferred as the corresponding Q-value (i.e., $Q(s, \text{B})$) is greater.

Network Basics

A patient goes to the doctor for a medical condition, and the doctor suspects 3 diseases as the cause of the condition. The 3 diseases are D_1, D_2, and D_3, and they are independent from each other (given no other observations). There are 4 symptoms S_1, S_2, S_3, and S_4, and the doctor wants to check for presence in order to find the most probable cause. S_1 can be caused by D_1, S_2 can be caused by D_1 and D_2, S_3 can be caused by D_1 and D_3, and S_4 can be caused by D_3. Assume all random variables are Bernoulli, i.e. the patient has the disease/symptom or not.

- **Q:** Draw a Bayesian network for this problem with the variable ordering $D_1, D_2, D_3, S_1, S_2, S_3, S_4$.

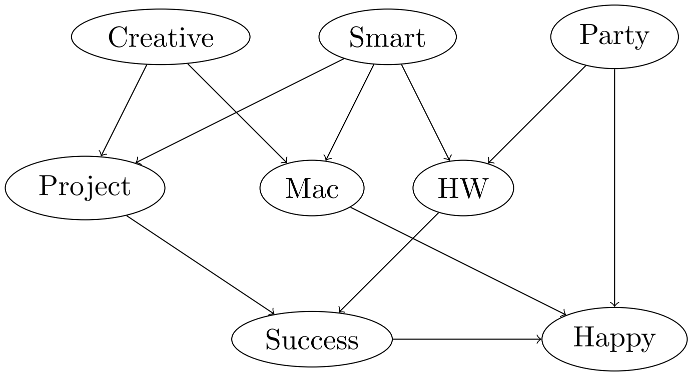
- **Q:** Write down the expression for the joint probability distribution given this network.

D-Separation

As part of a comprehensive study of the role of CS 181 on people's happiness, we have been collecting important data from students. In an entirely optional survey that all students are required to complete, we ask the following highly objective questions:

Do you party frequently [Party: Yes/No]?
Are you smart [Smart: Yes/No]?
Are you creative [Creative: Yes/No]?
Did you do well on all your homework assignments? [HW: Yes/No]
Do you use a Mac? [Mac: Yes/No]
Did your last major project succeed? [Project: Yes/No]
Did you succeed in your most important class? [Success: Yes/No]
Are you currently Happy? [Happy: Yes/No]

After consulting behavioral psychologists we build the following model:



- **Q:** True or False: *Party* is independent of *Success* given *HW*.

- **Q:** True or False: *Creative* is independent of *Happy* given *Mac*.

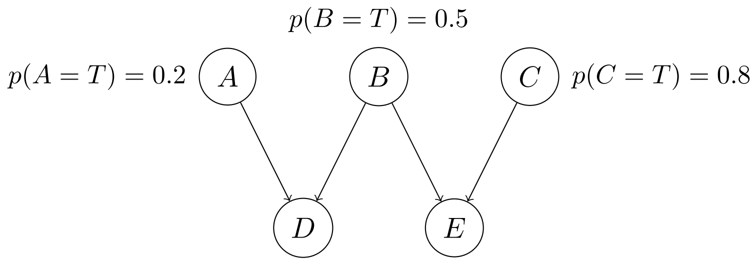
- **Q:** True or False: *Party* is independent of *Smart* given *Success*.

- **Q:** True or False: $Party$ is independent of $Creative$ given $Happy$.

- **Q:** True or False: $Party$ is independent of $Creative$ given $Success$, $Project$ and $Smart$.

Inference

Consider the following Bayesian network, where all variables are Bernoulli.



A	B	$p(D = T \mid A, B)$
F	F	0.9
F	T	0.6
T	F	0.5
T	T	0.1

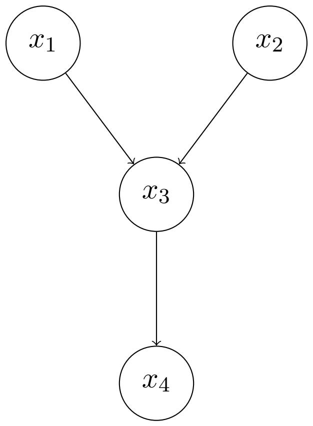
B	C	$p(E = T \mid B, C)$
F	F	0.2
F	T	0.4
T	F	0.8
T	T	0.3

- **Q:** What is the probability that all five variables are simultaneously false (F)?

- **Q:** What is the probability that A is false given that the remaining variables are all known to be true (T)?

Variable Elimination in Bayesian Networks

We apply an inference algorithm called variable elimination to the following Bayesian network:



Assume that all of the random variables are Bernoulli, meaning their domain is $\{0, 1\}$ with domain size $k = 2$. In this network, we can encode the joint distribution as

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)$$

If we wanted to calculate the marginal distribution of x_4 that is, have x_4 be our query without any evidence (conditioned on variables), we could naively marginalize out all other variables:

$$\begin{aligned} p(x_4) &= \sum_{x_1}\sum_{x_2}\sum_{x_3} p(x_1, x_2, x_3, x_4) \\ &= \sum_{x_1}\sum_{x_2}\sum_{x_3} p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3) \end{aligned}$$

To calculate these sums, for each value of x_4, we would need to do a sum-product that involves summing over the $k^3 = 8$ possible combinations of x_1, x_2, and x_3. In general, the number of combinations grows exponentially in the number of variables ($O(k^n)$ if you're familiar with big-O notation).

Note that Bayesian nets encode dependencies between variables, which we can use to calculate the marginal distribution more efficiently. By reordering the sums and eliminating one variable at a time, we derive the variable elimination procedure:

$$\begin{aligned} p(x_4) &= \sum_{x_1}\sum_{x_2}\sum_{x_3} p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3) \\ &= \sum_{x_3} p(x_4|x_3) \sum_{x_2} p(x_2) \sum_{x_1} p(x_3|x_1, x_2)p(x_1) \\ &= \sum_{x_3} p(x_4|x_3) \sum_{x_2} p(x_2)p(x_3|x_2) \\ &= \sum_{x_3} p(x_4|x_3)p(x_3) \\ &= p(x_4) \end{aligned}$$

Here, we eliminate x_1 using a k by k matrix $g_1(x_3, x_2)$, because we have to sum over x_1 for each possible value of x_2 and x_3. Then we eliminate x_2 with a K-dimensional vector $g_2(x_3)$, likewise because we sum over x_2 for each possible value of x_3. Lastly, we eliminate x_3, which results in a final K-dimensional vector of probabilities for x_4. Notice that we have a poly-tree, and we're eliminating leaves first and working towards our query variable, x_4.

In this way, we can perform the same computation in $O(k^3)$ time, because the longest elimination step has to do k^2 sum-product calculations for each element in the matrix $g_1(x_3, x_2)$, and each sum-product calculation takes $O(k)$ time (since we are summing over x_1). Compare this polynomial-time complexity (where we add the time taken for each elimination step, so our total time complexity only depends on the longest step) with the exponential-time complexity (based on the number of variables) of the naive approach.

Alternatively, we could have eliminated variables in a different order:

$$\begin{aligned}
p(x_4) &= \sum_{x_1} \sum_{x_2} \sum_{x_3} p(x_1) p(x_2) p(x_3|x_1, x_2) p(x_4|x_3) \\
&= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2) \sum_{x_3} p(x_3|x_1, x_2) p(x_4|x_3) \\
&= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2) p(x_4|x_1, x_2) \\
&= \sum_{x_1} p(x_1) p(x_4|x_1) \\
&= p(x_4)
\end{aligned}$$

Here, we eliminate x_3, then x_2, then x_1. Notice that the ordering matters: eliminating x_3 first results in a $k \times k \times k$ object $g(x_1, x_2, x_4)$, so our overall algorithm will run in $O(k^4)$ time. (Again, note that we have to account for both the k^3 sum-product calculations and the $O(k)$ time to do each calculation.)

In general, the computational cost of variable elimination depends on the number of variables in these intermediate factors, in particular the largest object computed ('tree-width').

Exercise: Variable Elimination

Consider the Bayesian network described in above, and assume the following Conditional Probability Table (CPT). Let $x_i \in \{0,1\}$ denote the values that variable X_i can take. Our goal is to find $p(x_4)$.

x_1	$p(x_1)$
0	0.3
1	0.7

x_2	$p(x_2)$
0	0.6
1	0.4

x_3	x_1	x_2	$p(x_3 \mid x_1, x_2)$
0	0	0	0.5
0	0	1	0.2
0	1	0	0.9
0	1	1	0.5
1	0	0	0.5
1	0	1	0.8
1	1	0	0.1
1	1	1	0.5

x_4	x_3	$p(x_4 \mid x_3)$
0	0	0.7
0	1	0.1
1	0	0.3
1	1	0.9

1. Eliminate X_1 first. Draw the resulting Bayesian network and compute the CPT.
2. Eliminate X_3 first. Draw the resulting Bayesian network and compute the CPT.
3. How many sum-product calculations do each of these variable elimination orders require? Which one is preferable?

Network Basics

A patient goes to the doctor for a medical condition, and the doctor suspects 3 diseases as the cause of the condition. The 3 diseases are D_1, D_2, and D_3, and they are independent from each other (given no other observations). There are 4 symptoms S_1, S_2, S_3, and S_4, and the doctor wants to check for presence in order to find the most probable cause. S_1 can be caused by D_1, S_2 can be caused by D_1 and D_2, S_3 can be caused by D_1 and D_3, and S_4 can be caused by D_3. Assume all random variables are Bernoulli, i.e. the patient has the disease/symptom or not.

- **Q:** Draw a Bayesian network for this problem with the variable ordering $D_1, D_2, D_3, S_1, S_2, S_3, S$

$$\begin{tikzcd}
D_1 \arrow[d] \arrow[dr] \arrow[drr] & D_2 \arrow[d] & D_3 \arrow[d] \arrow[dr] & \\
S_1 & S_2 & S_3 & S_4
\end{tikzcd}$$

- **Q:** Write down the expression for the joint probability distribution given this network.

$$P(D_1, D_2, D_3, S_1, S_2, S_3, S_4)$$
$$= P(D_1) \cdot P(D_2) \cdot P(D_3) \cdot P(S_1 | D_1) \cdot P(S_2 | D_1, D_2) \cdot P(S_3 | D_1, D_3) \cdot P(S_4 | D_3)$$

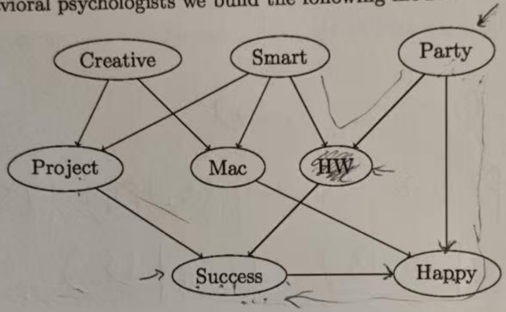
D-Separation

As part of a comprehensive study of the role of CS 181 on people's happiness, we have been collecting important data from students. In an entirely optional survey that all students are required to complete, we ask the following highly objective questions:

Do you party frequently [Party: Yes/No]?
Are you smart [Smart: Yes/No]?
Are you creative [Creative: Yes/No]?
Did you do well on all your homework assignments? [HW: Yes/No]
Do you use a Mac? [Mac: Yes/No]
Did your last major project succeed? [Project: Yes/No]
Did you succeed in your most important class? [Success: Yes/No]
Are you currently Happy? [Happy: Yes/No]

After consulting behavioral psychologists we build the following model:

BN →



Causal chain

- **Q:** True or False: *Party* is independent of *Success* given *HW*.

False, Party → HW ← smart → Project → Success

- **Q:** True or False: *Creative* is independent of *Happy* given *Mac*.

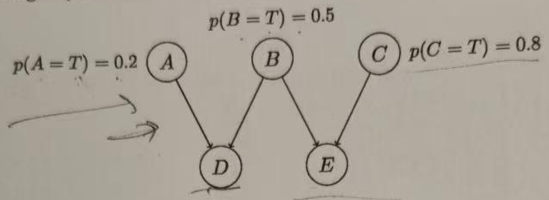
- **Q:** True or False: *Party* is independent of *Smart* given *Success*.

calcutate some useful quntity.

Inference by enumeration

⇒ binary

Consider the following Bayesian network, where all variables are Bernoulli.



A	B	$p(D = T \mid A, B)$
F	F	0.9
F	T	0.6
T	F	0.5
T	T	0.1

B	C	$p(E = T \mid B, C)$
F	F	0.2
F	T	0.4
T	F	0.8
T	T	0.3

- **Q:** What is the probability that all five variables are simultaneously false (F)?

$P(\neg a, \neg b, \neg c, \neg d, \neg e)$.

$= P(\neg a) \cdot P(\neg b) \cdot P(\neg c) \cdot P(\neg d \mid \neg a, \neg b) \cdot P(\neg e \mid \neg b, \neg c)$.

$= 0.8 \times 0.5 \times 0.2 \times 0.1 \times 0.8 = - —$.

- **Q:** What is the probability that A is false given that the remaining variables are all known to be true (T)?

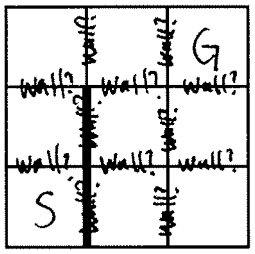
Name: Your Name CSE 473, Spring 2012

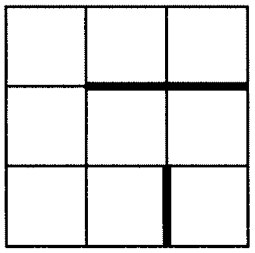
Please answer clearly and succinctly. Show your work clearly for full credit. If an explanation is requested, think carefully before writing. Points will be removed for rambling answers with irrelevant information (and may be removed in cases of messy and hard to read answers). If a question is unclear or ambiguous, feel free to make the additional assumptions necessary to produce the answer. State these assumptions clearly; you will be graded on the basis of the assumption as well as subsequent reasoning.

There are 10+0 problems worth 78 points on 7 pages

Problem 0 (1 point) Write your name on the top of each page.

Problem 1 (11 points) Suppose that an agent is in a 3x3 maze environment similar to the ones shown in the illustration below. The agent knows the size of the maze, that the initial state is (1,1) in the lower left and that the goal is (3,3). There are four actions, **Left**, **Right**, **Up** and **Down**, which have their usual effects, except when blocked by a wall or by the maze boundaries; in this case, the agent does not move. The agent does ***not*** (initially) know if there are any walls present in the maze nor where such walls may be. After each action, the agent receives a single percept, which tells it if it has failed to move; by using this percept, the agent may deduce the presence of walls.





a) (5 points) Describe the problem of finding a path from the initial state to the goal as a search problem in belief state space.

Initial State: location: (1,1) walls: (unknown, unknown, unknown, unknown, ..., unknown)
(all 12 walls are unknown)

Goal State : location: (3,3)

States : location, belief about each of the 12 walls (Yes, no, unknown)

Operators: actions (left, right, up, down) where you update knowledge about walls when you get it.

b) (3 points) How large is the *initial* belief state?

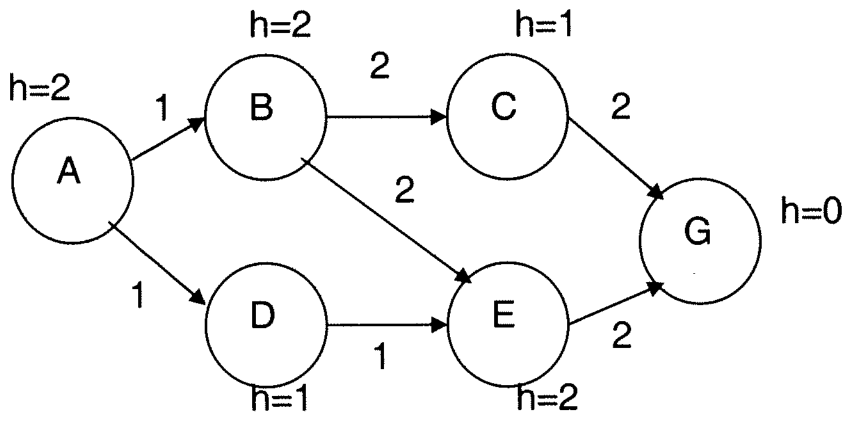
2^{12} each wall has two possible values (Yes, No) and there are 12 independent walls.

c) (3 points) Approximately how many distinct belief states are there? (Some of the possible belief states may not be reachable from the initial state, but include them anyway).

9×2^{12}

↖ # physical positions

Problem 2 (10 points) The figure below shows a problem-space graph, where A is the initial state and G denotes the goal. Edges are labeled with their true cost. We have a heuristic function, f(), written in the standard form: f(n)=g(n)+h(n) where g(n) is the cost to get from A to n and h(n) is an estimate of the remaining distance to G.



a) (2 points) In the graph above, is f() admissible? Why or why not?

Yes. h() is always ≤ true cost to G.

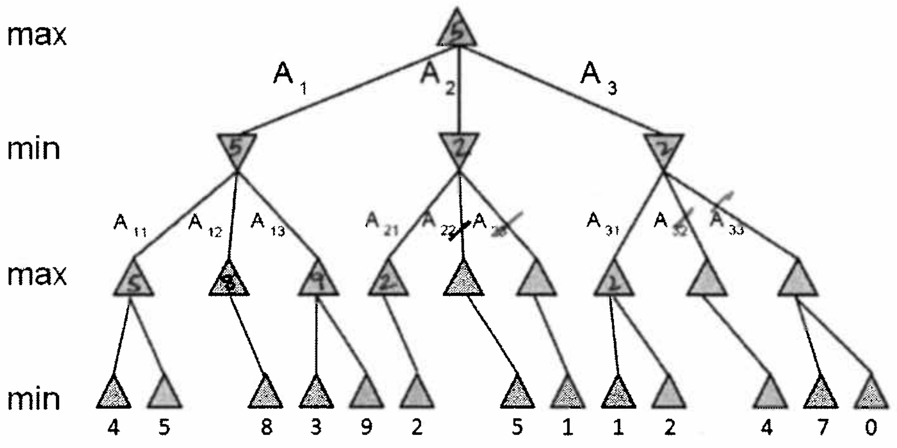
b) (2 points) Is f() monotonic? Why or why not?

Yes. h(n)+g(n) always increases on any path.

c) (6 points) Suppose we use IDA* to search the graph and that states having the same f values are visited in alphabetical order. In what order does the algorithm consider f-limits and visit states? Fill out the table below:

f-limit	States visited (in order)
2	A, D
3	B
4	C, E, G

Problem 3 (8 points) Label the intermediate nodes in the search tree, below, with values and draw a line across edges to denote any pruning done by alpha-beta search. Which action should the agent choose?



choose A_1

Problem 4 (4 points) Encode "All germans speak the same language" in first-order logic using the following notation: **Speaks(p,l)** denotes that person p speaks language l; **Nationality(p, c)** denotes that person p is from country c; the constant **G** denotes Germany.

$\forall p \exists l \ Nationality(p, G) \Rightarrow Speaks(p, l)$

also acceptable to say

$\forall p_1, p_2 \exists l \ Nationality(p_1, G) \wedge Nationality(p_2, G) \Rightarrow Speaks(p_1, l) \wedge Speaks(p_2, l)$

received full credit but first answer is better

Problem 5 (4 points)

Which of the following are semantically and syntactically correct translations of "Everyone's zipcode within a state has the same first digit"?

i. $\forall x, s, z_1\ [State(s) \land LivesIn(x,s) \land Zip(x) = z_1] \Rightarrow$
$[\forall y, z_2\ LivesIn(y,s) \land Zip(y) = z_2 \Rightarrow Digit(1, z_1) = Digit(1, z_2)]$.

ii. $\forall x, s\ [State(s) \land LivesIn(x,s) \land \exists z_1\ Zip(x) = z_1] \Rightarrow$
$[\forall y, z_2\ LivesIn(y,s) \land Zip(y) = z_2 \land Digit(1, z_1) = Digit(1, z_2)]$.

iii. $\forall x, y, s\ State(s) \land LivesIn(x,s) \land LivesIn(y,s) \Rightarrow Digit(1, Zip(x) = Zip(y))$.

iv. $\forall x, y, s\ State(s) \land LivesIn(x,s) \land LivesIn(y,s) \Rightarrow Digit(1, Zip(x)) = Digit(1, Zip(y))$.

Answer			
	i	[✓ yes]	no
	ii	yes	[✓ no]
	iii	yes	[✓ no]
	iv	[✓ yes]	no

Problem 6 (4 points) After your yearly checkup, your doctor has bad news and good news. The bad news is that you tested positive for a serious disease is 99% accurate (ie the probability of testing positive when you do have the disease is 0.99 as is the probability of testing negative when you don't have the disease). The good news is that this is a rare disease, striking only 1 in 10,000 people of your age. Use Bayesian reasoning to calculate the chance that you actually have the disease?

$$Pr(\text{disease} \mid \text{test}) = \frac{Pr(\text{test} \mid \text{disease}) \cdot P(\text{disease}}{P(\text{test} \mid \text{disease}) \cdot P(\text{disease}) + P(\text{test} \mid \neg\text{disease}) \cdot P(\neg\text{dise}}$$

$$= .98\%$$

Problem 7 (4 points) Suppose you are given a sack containing n unbiased coins. You are told that n-1 of the coins are normal with heads on one side and tails on the other. One coin in the sack is fake, having heads on both sides. Suppose that you reach into the sack, pick a coin uniformly at random, flip it twice, and get heads both times. What is the (conditional) probability that you picked the fake coin?

$$P(\text{Fake} \mid T1=H, T2=H) = \frac{P(T1=H, T2=H \mid \text{Fake}) \cdot Pr(\text{Fake})}{P(T1=H, T2=H)}$$

$P(T1=H, T2=H) =$
$P(T1=H, T2=H \mid \text{Fake}) \cdot Pr(\text{Fake})$
$+ P(T1=H, T2=H \mid \text{Real}) \cdot Pr(\text{Real})$
$= 1 \cdot \frac{1}{n} + \left(\frac{1}{2}\right)^2 \left(\frac{n-1}{n}\right)$

$$= \frac{1 \cdot \frac{1}{n}}{\frac{1}{n}{}_4 + \frac{1}{4}\frac{(n-1)}{n}} = \frac{4}{3+n}$$

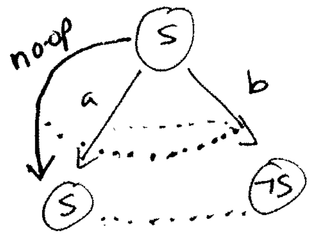
Problem 8 (13 points) Suppose an agent inhabits a world with two states, S and ¬S, can do exactly one of two actions, a and b. Action a does nothing and action b flips from one state to the other. Suppose that S is initially true.

a) (6 points) Draw the complete planning graph (with mutex relations) for the initial propositional level, the first action level and the next propositional level.

Prop$_0$

action$_1$

prop$_1$



$$V(S) = \arg\max_{a} \left\{ \sum_{s'} Q(s',a) \right\}$$

$$Q(s',a) = \sum_{s'} R_a(s') + \gamma V(s')$$

b) (6 points) Consider this world as an MDP (with deterministic actions) and let R(S)=3, R(¬S)=2 and γ=0.5. Complete the columns of the following table. (Note we are assuming that reward is a function of the destination state and is independent of the starting state or action executed.)

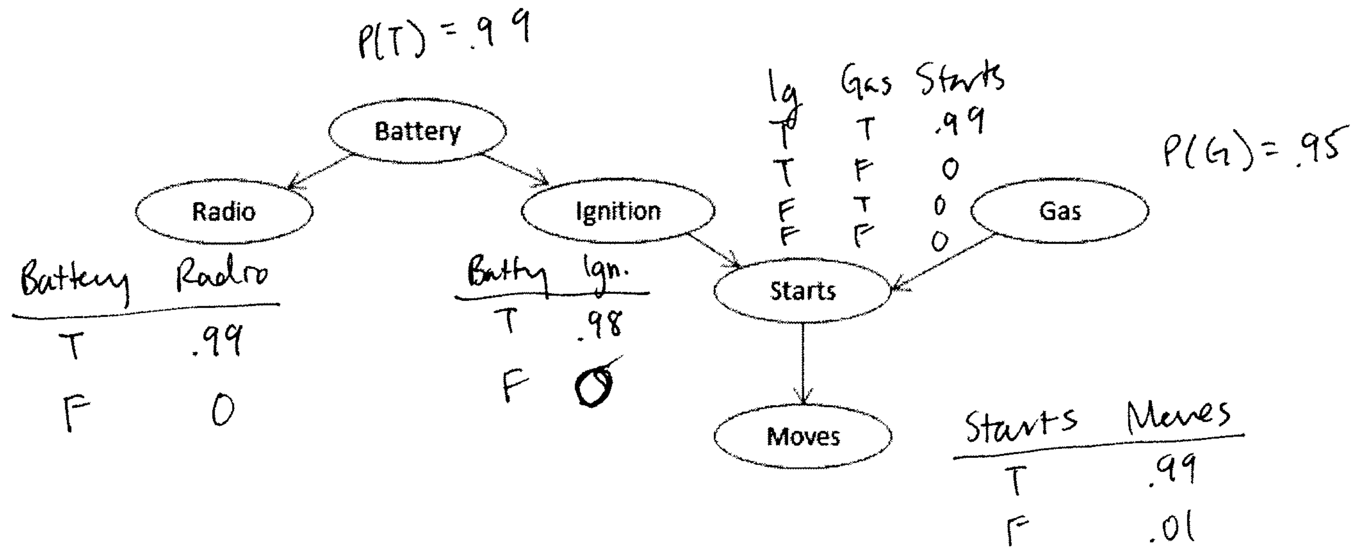
Time	Q(S,a)	Q(S,b)	V(S)	Q(¬S,a)	Q(¬S,b)	V(¬S)
0	■	■	0	■	■	0
1	3	2	3	2	3	3
2	$3+3\gamma$	$3+2\gamma$	$3+3\gamma$	$2+2\gamma$	$2+3\gamma$	$2+3\gamma$
3	$3+\gamma(3+3\gamma)$	$2+\gamma(3+3\gamma)$	$3+\gamma(3+3\gamma)$	$2+\gamma(3+3\gamma$	$2+\gamma(3+3\gamma)$	$2+\gamma(2+3\gamma)$

c) (1 point) Based on the V3 values you have computed, what is the policy an agent should choose?

$\pi^* = a$

Name: ______________________________ CSE 473, Spring 2012

Problem 9 (7 points) Consider the Bayesian network shown below.



P(T) = .99

Ig	Gas	Starts
T	T	.99
T	F	0
F	T	0
F	F	0

P(G) = .95

Battery	Radio
T	.99
F	0

Batty	Ign.
T	.98
F	0

Starts	Moves
T	.99
F	.01

a) (1 points) Is Radio conditionally independent of Gas given Battery?

Yes

b) (1 points) Is Radio conditionally independent of Gas given Starts?

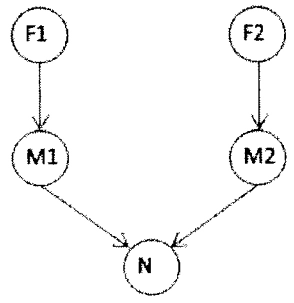
No

c) (1 points) Is radio conditionally independent of Gas given Moves?

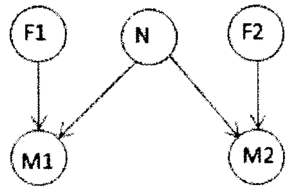
No

d) (4 points) Give reasonable probability tables for all the nodes (draw them by the nodes in the network, above)

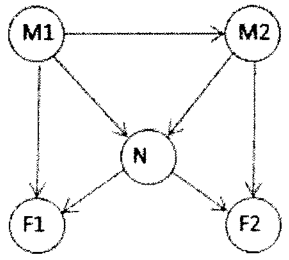
Problem 10 (12 points) Two astronomers in different parts of the world make measurements, M1 and M2, of the number of stars, N, in some small region of the sky, using their telescopes. Normally, there is a small possibility ε of error by up to one star in each direction. Eash telescope can also (with a much smaller probability *f*) be badly out of focus (events F1 and F2), in which case the scientist will undercount by three or more starts (or if N is less than 3, fail to detect any stars at all). Consider the three networks shown below:



(a)



(b)



(c)

a) (6 points) Which of these Bayesian Networks are correct (but not necessarily efficient) representations of the preceding information?

b, c

a is not correct because it has F_1 cond indep of N given M_1

b) (1 points) Which is the best network (explain)?

b. It is the simplest of the valid networks.

c) (5 points) Write out a conditional distribution for P(M1 | N) for the case where N ∈ {1, 2, 3} and M1 ∈ {0, 1, 2, 3, 4}. Each entry in the conditional distribution should be expressed as a function of the parameters ε and/or *f*.

N

M	1	2	3
0	$f + \varepsilon/2$	f	f
1	$1-\varepsilon$	$\varepsilon/2$	0
2	$\varepsilon/2$	$1-\varepsilon$	$\varepsilon/2$
3	0	$\varepsilon/2$	$1-\varepsilon$
4	0	0	$\varepsilon/2$