06 02 06
13 03 07
08 04 22 16
09 05 23
~~10~~ ~~14~~
~~11~~
~~12~~ 15
13

~~18~~
~~19~~
~~20~~
~~21~~

MDP

- Finite-stage v.s. ongoing problems
  - infinite horizon: the process may go on forever.
  - indefinite horizon: the agent will eventually stop, but it does not know when it will stop.
- Utility at the end v.s. a sequence of rewards

An MDP needs:

S: set of states

A: Set of actions

transition probabilities: $P(s'|s,a)$

reward function: $R(s,a,s')$

Choice of Reward Function

$R(s)$: the reward of entering state s.

- Total reward $R(s_0)+R(s_1)+R(s_2)+\cdots$
  - If the sum is infinite, we cannot compare two policies of an MDP.
- Average reward $\lim_{n\to\infty}\frac{1}{n}(R(s_0)+R(s_1)+R(s_2)+\cdots)$
  - If the total reward is finite, the average reward is zero.
- Discounted reward $R(s_0)+\gamma R(s_1)+\gamma^2 R(s_2)+\cdots$ where $0\le\gamma<1$ is the discount factor.
  - We prefer getting the same reward sooner rather than later.
  - Everyday, there is a chance that tomorrow won't come.
  - The total discounted reward is finite.

Make Sure the policy is Finite

Type of MDP

- A fully-observable MDP
  The agent knows what state it is in.
- A partially observable MDP (POMDP)
  combines a MDP and a hidden Markov model.
  The agent does not know what state it is in, but it can get some noisy signal of the state.

Grid World

Eg.

- There are four actions: up, down, left, and right.
  Every action is possible in every state.
- The transition model $P(s'|s,a)$:
  An action achieves its intended effect with probability 0.8.
  An action leads to a 90-degree left turn with probability 0.1.
  An action leads to a 90-degree right turn with probability 0.1.
  If the robot bumps into a wall, it stays in the same square.
- The reward function $R(s)$ is the reward of entering state s.
  $R(s_{24})=-1$.
  $R(s_{34})=1$.
  Otherwise, $R(s)=-0.04$.

CQ: The robot is in s_{14} and tries to move to our right, what is the probability that the robot stays in s_{14}?

(A) 0.1
(B) 0.2
(C) 0.8
(D) 0.9
(E) 1.0

	1	2	3	4
1	Start			
2		X		-1
3				+1

0.1, 0.8, 0.1

Tips: If environment is deterministic, don't worry about Agent would not land unexpected (100%)

A fixed Sequence of Action?

CQ: A fixed sequence of actions

The robot could reach S_{11} w/ prob 0.1^5.
OR
The robot could reach S_{14} w/ prob $0.1*0.1*0.8*0.8*0.8$.
unexpected

CQ: Consider the action sequence "down, down, right, right, and right". This action sequence could take the robot to more than one square with positive probability.

(A) True
(B) False
(C) I don't know

	1	2	3	4
1	Start			
2		X		-1
3				+1

So, fixed Sequence not fitting to MDP.

Policies

A policy specifies what the agent should do as a function of the current state.

A policy is
- non-stationary if it is a function of the state and the time.
- stationary if it is a function of the state.

The optimal policies — R(s)!

The optimal policy of the grid world changes based on $R(s)$ for any non-goal state s. It shows a careful balancing of risk and reward.

	1	2	3	4
1	Start			
2		X	Risk	-1
3				+1

Exit

1. If R(s) too big negative, It would go either exit (-1) or (+1)
2. If R(s) is normal negative, It would go (+1), if possible, (two way acceptable)
3. If R(s) is small negative, It would go (+1) try better than ②, Conservative ↑, risk itself ↓ (eg. go wall, avoid (2,3), one best way)
4. If R(s) is positive, It would avoid Both goal state (Exit)

When $R(s)>0$, what does the optimal policy look like?

	1	2	3	4
1	Start	↔	↔	↑
2	↔	X	←	-1
3	↔	↔	←	+1

- avoids both goal states.

Optimal policy — Given V^*

$R(s)\to -0.04$, $\gamma=1$

The Expected Utility of a Policy

$R(s)$: one-time reward of entering state s.

$V^\pi(s)$: expected utility of entering state s and following the policy π thereafter.
the expected value of total discounted reward.
long-term

$V(s)$: expected utility of entering state s and following the optimal policy π^* thereafter.

π → Given policy
$*$ → Optimal policy

→ close to (+1), higher expected utilities

	1	2	3	4
1	0.705	0.655	0.611	0.388
2	0.762	X	0.660	-1
3	0.812	0.868	0.918	+1

Figure: $V^*(s)$ for $\gamma=1$ and $R(s)=-0.04, \forall s\neq s_{24}, s\neq s_{34}$.

→ Quantity Q^*:
Expected Utilities of taking actions in Current State

$$Q^*(s,a)=\sum_{s'}P(s'|s,a)\,V^*(s')$$

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

Eg.

$Q(s_{13},\text{down}) = 0.8*0.660 + 0.1*0.388 + 0.1*0.655 = 0.6323$

$Q(s_{13},\text{left}) = 0.8*0.655 + 0.1*0.660 + 0.1*0.388 = 0.6511$

$Q(s_{13},\text{right}) = 0.8*0.388 + 0.1*0.611 + 0.1*0.660 = 0.4375$

$Q(s_{13},\text{up}) = 0.8*0.611 + 0.1*0.655 + 0.1*0.388 = 0.5931$

$\pi(s_{13}) = \text{left}$.

	1	2	3	4
1	0.705	0.655	0.611	0.388
2	0.762	X	0.660	-1
3	0.812	0.868	0.918	+1

Solving $V^*(s)$ using Value Iteration.

V: EU of Agent → get in S

, then follow policy after get immediate reward (long time Total discount reward)

Q: long time total discount reward, but we are already in a state what happen if we take particular action.

V & Q relation (Bellman)

V and Q are defined recursively in terms of each other.

$$V^*(s) = R(s) + \gamma \max_a Q^*(s,a) \quad (3)$$

$$Q^*(s,a) = \sum_{s'} P(s'|s,a) V^*(s'). \quad (4)$$

Combining equations 3 and 4, we get the Bellman equations:

Bellmen:

$$V^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s'). \quad (5)$$

$V^*(s)$ are the unique solution to the Bellman equations.

Eg.

Write down $V^*(s_{11})$

	1	2	3	4
1	0.705	0.655	0.611	0.388
2	0.762	X	0.660	-1
3	0.812	0.868	0.918	+1

Write down the Bellman equation for $V^*(s_{11})$.

$$V(s_{11}) = -0.04 + \gamma \max \begin{cases} 0.8 * V(s_{12}) + 0.1 * V(s_{21}) + 0.1 * V(s_{11}), & \text{(right)} \\ 0.9 * V(s_{11}) + 0.1 * V(s_{12}), & \text{(up)} \\ 0.9 * V(s_{11}) + 0.1 * V(s_{21}), & \text{(left)} \\ 0.8 * V(s_{21}) + 0.1 * V(s_{12}) + 0.1 * V(s_{11}). & \text{(down)} \end{cases}$$

Value Iteration Algo

Tips: Solve system of Bellman equation efficiently? No, [max]

So, Iteration needed

Solving for $V^*(s)$ iteratively

The Bellman equations:

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

Let $V_i(s)$ be the values for the i-th iteration.

1. Start with arbitrary initial values for $V_0(s)$.
2. At the i-th iteration, compute $V_{i+1}(s)$ as follows.

$$V_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s').$$

3. Terminate when $\max_s |V_i(s) - V_{i+1}(s)|$ is small enough.

If we apply the Bellman update infinitely often, the V_i's are guaranteed to converge to the optimal values.

Eg.

CQ: Calculating $V_1(s_{23})$

$V_1(s_{23}) = -0.04 + 1 * \max\{0.8*0+0.1*(-1)+0.1*0, \text{up}; 0.8*(-1)+0.1*0+0.1*0, \text{right}; 0.8*0+0.1*(-1)+0.1*0, \text{down}; 0.8*0+0.1*0+0.1*0, \text{left}\}$

$= -0.04 + 0 = -0.04$

0 step

CQ: What is $V_1(s_{23})$? $= -0.04$

(A) $(-\infty, 0)$ (B) $[0, 0.25)$ (C) $[0.25, 0.5)$
(D) $[0.5, 0.75)$ (E) $[0.75, 1]$

$V_0(s)$:

	1	2	3	4
1	0	0	0	0
2	0	X	0	-1
3	0	0	0	+1

CQ: Calculating $V_1(s_{33})$ $V_1(s_{33}) = -0.04 + 1 * \max\{0.8*1+0.1*0+0.1*0, \text{right}; 0.8*0+0.1*1+0.1*0, \text{up}; 0.8*0+0.1*0+0.1*0, \text{left}; 0.8*0+0.1*1+0.1*0, \text{down}\}$

$= -0.04 + 0.8 = 0.76$

1 Step

CQ: What is $V_1(s_{33})$?

(A) 0.26 (B) 0.36 (C) 0.46
(D) 0.56 (E) 0.76

$V_0(s)$:

	1	2	3	4
1	0	0	0	0
2	0	X	0	-1
3	0	0	0	+1

$V_1(s)$:

	1	2	3	4
1	−0.04	−0.04	−0.04	−0.04
2	−0.04	X	−0.04	-1
3	−0.04	−0.04	0.76	+1

$V_2(s)$:

	1	2	3	4
1	−0.08	−0.08	−0.08	−0.08
2	−0.08	X	0.464	-1
3	−0.08	0.56	0.832	+1

Thus, Each state accumulate negative rewards until Algo finds a path to (+1)

synchronous: □□ — update → □□ same time

asynchronous: Real time update, Converges faster

Policy Iteration Algo

(Reinforcement needed)

[inspired]: to derive the optimal policy doesn't require accurate estimate of utility function $V^*(s)$

∴ Policy Iteration we got

- Policy iteration alternates between two steps.
  - Policy evaluation: Given a policy π_i, calculate $V^{\pi_i}(s)$, which is the utility of each state if π_i were to be executed.
  - Policy improvement: Calculate a new policy π_{i+1} using V^{π_i}.

Terminates when there is no change in the policy.

policy iteration: $\pi_0 \to V^{\pi_0}(s) \to \pi_1(s) \to V^{\pi_1}(s) \to \cdots \to \pi^*(s) \to V^*(s) \to \pi^*(s)$

(Policy □→△→□→△) (Value □□→△)

- Policy improvement: $V^{\pi_i}(s) \to \pi_{i+1}(s)$

$$\pi_{i+1}(s) = \arg\max_a \underbrace{Q(s,a)}_{\sum_{s'} P(s'|s,a) V^{\pi_i}(s')}.$$

- Policy evaluation: $\pi_i(s) \to V^{\pi_i}(s)$

$$V_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) V_i(s').$$

Simplified Bellmen without max

Policy evaluation: $V(S_{11}) = -0.04 + \gamma(0.8\,V(S_{12}) + 0.1\,V(S_{21}) + 0.1\,V(S_{11}))$

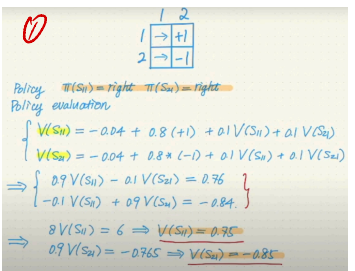
linear $V(s) = R(s) + \gamma \sum_{s'} P(s'|s,\pi(s)) V(s')$.

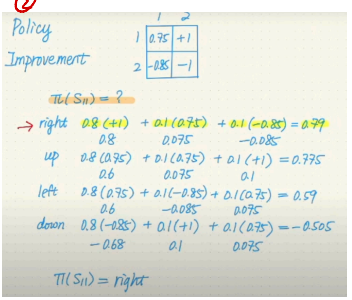
$V(s_{11}) = -0.04 + \gamma \max\{0.8*V(s_{12}) + 0.1*V(s_{21}) + 0.1*V(s_{11}),$
$0.9*V(s_{11}) + 0.1*V(s_{12}),$
$0.9*V(s_{11}) + 0.1*V(s_{21}),$
$0.8*V(s_{21}) + 0.1*V(s_{12}) + 0.1*V(s_{11})\}$

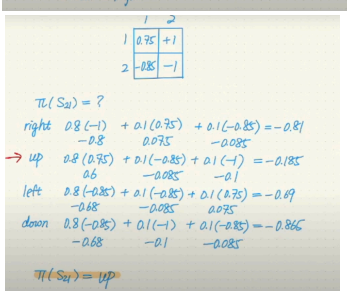
Bellman equations:

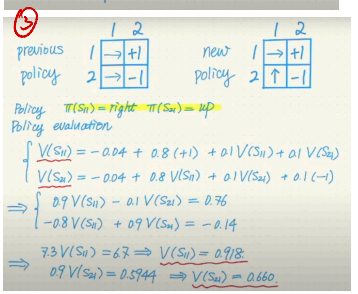
nonlinear $V(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V(s')$.

Eg.

① 

② 



③ 

Terminate until no change

Reinforcement Learning

Eg.

An Reinforcement Learning Agent

Let's consider fully observable, single-agent reinforcement learning.
We will formalize this problem as a Markov decision process.

- Given the possible states and the set of actions.
- Observes the state and the rewards received.
- Carries out an action.
- Goal is to maximize its discounted reward.

Why is reinforcement learning challenging?

- Which action was responsible for this reward/punishment?
- How will this action impact my utility?
- Should I explore or exploit?

Model-based

Include $P(s'|s,a)$ and $R(s)$

Passive Learning Agent

- Fix a policy π.
- Goal is to learn $V^\pi(s)$ (the expected value of policy π for state s).
- Similar to policy evaluation.
- Does not know the transition model $P(s'|s,a)$ nor the reward function $R(s)$.
- Solution: Adaptive Dynamic Programming
  - Learn $P(s'|s,a)$ and $R(s)$ using the observed transitions and rewards.
  - Learn $V^\pi(s)$ by solving Bellman equations (exactly or iteratively).

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s,\pi(s)) V^\pi(s').$$

Algo Step:

Passive ADP Algorithm

1. Repeat steps 2 to 5.
2. Follow policy π and generate an experience (s, a, s', r').
3. Update reward function: $R(s') \leftarrow r'$
4. Update the transition probability.

$$N(s,a) = N(s,a) + 1$$
$$N(s,a,s') = N(s,a,s') + 1$$
$$P(s'|s,a) = N(s,a,s')/N(s,a)$$

5. Derive $V^\pi(s)$ by using the Bellman equations.

$$V(s) = R(s) + \gamma \sum_{s'} P(s'|s,\pi(s)) V(s')$$

Eg.

s_{11}	$+1$
s_{21}	-1

- $\pi(s_{11}) = down, \pi(s_{21}) = right$
- $\gamma = 0.9$
- $R(s_{11}) = -0.01, R(s_{21}) = -0.01, R(s_{12}) = 1, R(s_{22}) = -1$
- $N(s,a) = 5, \forall s, a.$
- $N(s,a,s') = 3$ for the intended direction.
- $N(s,a,s') = 1$ for a direction to the left or right of the intended direction.
- The current state is s_{11}.

experience

s_{11}	$+1$
s_{21}	-1

$\langle s_{11}, down, s_{21}, -0.04 \rangle$

1. No need to update the reward function.
2. Update the counts.
   $N(s_{11}, down) = 6$ and $N(s_{11}, down, s_{21}) = 4$.
3. Solve the Bellman equations. $P(s_{21}|s_{11}, down) = \frac{4}{6} = 0.667$.

$$V(s_{11}) = -0.04 + 0.9(0.667V(s_{21}) + 0.167(1) + 0.167V(s_{11}))$$
$$V(s_{21}) = -0.04 + 0.9(0.6(-1) + 0.2V(s_{11}) + 0.2V(s_{21}))$$

The solutions are:

$$V(s_{11}) = -0.4378, V(s_{21}) = -0.8034$$

→ Active ADP

short term: Reward from action

long term: get data learn model

The passive ADP agent learns the expected value of a fixed policy.

What action should the agent take at each step?
Two things are useful for the agent to do:

1. exploit: take an action that maximizes $V(s)$.
2. explore: take an action that is different from the optimal one.

The greedy agent seldom converges to the optimal policy and sometimes converges to horrible policies because the learned model is not the same as the true environment

There is a trade-off between exploitation and exploration.

Trade off Exploitation and Exploration

1. ϵ-greedy exploration strategy:
   - select random action with probability ϵ, and
   - select the best action with probability $1 - \epsilon$.
   - may decrease ϵ over time.
2. Softmax selection using Gibbs/Boltzmann distribution.
   - Choose action a with probability $\frac{Q(s,a)/T}{\sum_a Q(s,a)/T}$.
   - $T > 0$ is the temperature. When T is high, the distribution is close to uniform. When T is low, the higher-valued actions have higher probabilities.
3. Initialize the values optimistically to encourage exploration.

Optimistic Utility Values to Encourage Exploration

We will learn $V^+(s)$ (the optimistic estimates of $V(s)$).

$$V^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} P(s'|s,a) V^+(s'), N(s,a)\right)$$

$$f(u,n) = \begin{cases} R^+, & \text{if } n < N_e \\ u, & \text{otherwise} \end{cases}$$

$f(u,n)$ trade-offs exploitation and exploration.

- R^+ is the optimistic estimate of the best possible reward obtainable in any state.
- If we haven't visited (s,a) at least N_e times, assume its expected value is R^+.
- Otherwise, use the current $V^+(s)$ value.

Active ADP Algorithm

1. Initialize $R(s), V^+(s), N(s,a), N(s,a,s')$.
2. Repeat steps 3 to 7 until we have visited each (s,a) at least N_e times and the $V^+(s)$ values converged. (check convergence)
3. Determine the best action a for current state s using $V^+(s)$.
4. Take action a and generate an experience $\langle s, a, s', r' \rangle$
5. Update reward function: $R(s') \leftarrow r'$
6. Update the transition probability.

$$N(s,a) = N(s,a) + 1, N(s,a,s') = N(s,a,s') + 1$$
$$P(s'|s,a) = N(s,a,s')/N(s,a)$$

7. Updated $V^+(s)$ using the Bellman updates. (check convergence)

$$V^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} P(s'|s,a) V^+(s'), N(s,a)\right)$$

ADP: more efficient experience, fast learn

Q: low cost, computation

Model-free.

- Temporal Difference

Introduce it:

Bellman Equations for $Q(s,a)$

$Q(s,a)$ is the expected value of performing action a in state s.
We can define Bellman equations for both $V(s)$ and $Q(s,a)$.

Bellman equations for $V(s)$:

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

Bellman equations for $Q(s,a)$:

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

Learning $V(s)$ and $Q(s,a)$ are equivalent!
What is the advantage of learning $Q(s,a)$?

Temporal Difference Error

Assume that we observed (s_1, r_1, a, s_2).
Based on this transition, what should $Q(s_1,a)$ satisfy?

Start with the Bellman equations for $Q(s_1,a)$.

$$Q(s_1,a) = R(s_1) + \gamma \sum_{s'} P(s'|s_1,a) \max_{a'} Q(s',a')$$

$Q(s_1,a)$ should be computed by the RHS of the above equation.
Assume that this transition always occurs ($P(s_2|s_1,a) = 1$).
Thus, $Q(s_1,a)$ should be

$$R(s_1) + \gamma \max_{a'} Q(s_2,a')$$

(prediction of $Q(s_1,a)$)

Temporal difference (TD) error:

$$(R(s_1) + \gamma \max_{a'} Q(s_2,a')) - Q(s_1,a)$$

1. 你从起点开始，移动到下一个格子。
2. 一旦你移动到一个新的格子，你获得一个即时奖励，并且观察到新格子的价值。
3. 你更新你离开的那个格子的价值。基于即时奖励和新格子的价值。

价值更新规则是：

$$V(s) \leftarrow V(s) + \alpha [r + \gamma V(s') - V(s)]$$

Q learning — from TD

Updates

Q-Learning Updates

TD

Given an experience (s, r, a, s'), update $Q(s,a)$ as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(\underbrace{R(s) + \gamma \max_{a'} Q(s',a')}_{\text{predicted value}} - \underbrace{Q(s,a)}_{\text{current value}} \right)$$

An alternative version:

$$Q(s,a) \leftarrow \underbrace{(1-\alpha) Q(s,a)}_{\text{current value}} + \alpha \underbrace{\left(R(s) + \gamma \max_{a'} Q(s',a') \right)}_{\text{predicted value}}$$

Passive Q-Learning Algorithm

1. Repeat steps 2 to 4.
2. Follow policy π and generate an experience $\langle s, r, a, s' \rangle$.
3. Update reward function: $R(s) \leftarrow r$
4. Update $Q(s,a)$ by using the temporal difference update rules:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

The learning rate α:
α controls the size of each update. If α decreases as $N(s,a)$ increases, Q values will converge to the optimal values.
For example, $\alpha(N(s,a)) = \frac{10}{9 + N(s,a)}$.

# of a.d. optimal ↑, slow↑

Active Q-Learning Algorithm

like ADP, no policy but

1. Initialize $R(s), Q(s,a), N(s,a), N(s,a,s')$.
2. Repeat steps 3 to 6 until we have visited each (s,a) at least N_e times and the $Q(s,a)$ values converged.
3. Determine the best action a for current state s using $V^+(s)$

$$a = \arg\max_a f\left(Q(s,a), N(s,a)\right), f(u,n) = \begin{cases} R^+, & \text{if } n < N_e \\ u, & \text{otherwise} \end{cases}$$

4. Take action a and generate an experience $\langle s, r, a, s' \rangle$
5. Update reward function: $R(s) \leftarrow r$
6. Update $Q(s,a)$ using the temporal difference update rules.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

Probability

A **random variable**

- Has a **domain** of possible values
- Has an associated **probability distribution**, which is a function from the domain of the random variable to $[0, 1]$.

0 ~ 1

The inclusion-exclusion principle:

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

Prior / posterior

$P(X)$:

- **prior** or **unconditional** probability
- Likelihood of X in the absence of any other information
- Based on the background information

$P(X|Y)$

- **posterior** or **conditional** probability
- Likelihood of X given Y.
- Based on Y as evidence

Joint Distribution

↳ the P over a subset of variables ?
↳ conditional P ?

Sum Rule

Given a joint probability distribution,
we can compute the probability over a subset of the variables
using the sum rule.

We can sum out every variable that we do not care about.

Given a joint distribution over A, B, and C,
we can calculate $P(A \wedge B)$ by summing out C:
$P(A \wedge B) = P(A \wedge B \wedge C) + P(A \wedge B \wedge \neg C)$
we can calculate $P(A)$ by summing out B and C:
$P(A) = P(A \wedge B \wedge C) + P(A \wedge B \wedge \neg C) + P(A \wedge \neg B \wedge C) + P(A \wedge \neg B \wedge \neg C)$

Eg.

(A) 0.05
(B) 0.06
(C) 0.07
(D) 0.08
(E) 0.09

	A: G	A: ¬G	¬A: G	¬A: ¬G
W	0.032	0.048	0.036	0.324
¬W	0.008	0.012	0.054	0.486

$P(A \wedge \neg G) = 0.06.$

Conditional P — Product Rule

Given a joint probability distribution,
how do we compute the probability one variable A
conditioned on knowing the value of another variable B?

We can use the product rule.

For example, how do we calculate $P(A|B)$
given a joint distribution over A, B, C?

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} \qquad P(A \wedge B) = P(A|B)\,P(B).$$

$P(A \wedge B) = P(A)P(B)$ iff independent

Eg.

Mrs. Gibbon is NOT calling given that the alarm is going?

(A) 0.2
(B) 0.4
(C) 0.6
(D) 0.8
(E) 1.0

	A: G	A: ¬G	¬A: G	¬A: ¬G
W	0.032	0.048	0.036	0.324
¬W	0.008	0.012	0.054	0.486

$P(\neg A \wedge W) = 0.36,$
$P(A \wedge \neg G) = 0.06,$
$P(\neg A) = 0.9.$

$P(\neg G|A) = 0.6$

Addition: $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

Inference

Inference Using the Prior and Conditional Probabilities

How do we

- calculate a probability over a subset of the variables?
- calculate a conditional probability?

The prior probabilities

$P(A) = 0.1$

The conditional probabilities $P(x|y)$

$P(W|A) = 0.9$ $\quad P(G|A) = 0.3$
$P(W|\neg A) = 0.4$ $\quad P(G|\neg A) = 0.1$

$P(W|A \wedge G) = 0.9$ $\quad P(G|A \wedge W) = 0.3$
$P(W|A \wedge \neg G) = 0.9$ $\quad P(G|A \wedge \neg W) = 0.3$
$P(W|\neg A \wedge G) = 0.4$ $\quad P(G|\neg A \wedge W) = 0.1$
$P(W|\neg A \wedge \neg G) = 0.4$ $\quad P(G|\neg A \wedge \neg W) = 0.1$

Chain Rule: for joint ~ A∩B

Calculate a Joint Probability Using the Chain Rule

For two variables (a.k.a. the product rule):

$$P(A \wedge B) = P(A|B) * P(B) = P(B|A) * P(A)$$

For three variables:

$$P(A \wedge B \wedge C) = P(A|B \wedge C) * P(B|C) * P(C)$$

For any number of variables:

$$P(X_n \wedge X_{n-1} \wedge \cdots \wedge X_2 \wedge X_1)$$
$$= \prod_{i=1}^{n} P(X_i | X_{i-1} \wedge \cdots \wedge X_1)$$
$$= P(X_n | X_{n-1} \wedge \cdots \wedge X_2 \wedge X_1) * \ldots * P(X_2|X_1) * P(X_1)$$

Flipping a conditional P:

Flipping a Conditional Probability

leading to bayes rule

Often you have causal knowledge:

- $P(\text{symptom} \mid \text{disease})$
- $P(\text{alarm} \mid \text{fire})$

See evidence

...and you want to do evidential reasoning:

- $P(\text{disease} \mid \text{symptom})$
- $P(\text{fire} \mid \text{alarm})$.

Bayes Rule

Flipping a Conditional Probability using the Bayes' Rule

$$P(X \wedge Y) = P(X|Y) * P(Y) = P(Y|X) * P(X)$$

$$P(X|Y) = \frac{P(Y|X)\,P(X)}{P(Y \wedge X) + P(Y \wedge \neg X)} = \frac{P(Y|X)\,P(X)}{P(Y|X)\,P(X) + P(Y|\neg X)\,P(\neg X)}$$

Definition (Bayes' rule)

$$P(X|Y) = \frac{P(Y|X) * P(X)}{P(Y)}$$ a normalization constant.

$$P(\neg X|Y) = \frac{P(Y|\neg X)\,P(\neg X)}{P(Y|X)\,P(X) + P(Y|\neg X)\,P(\neg X)}$$ Same

① calculate $P(Y|X)P(X)$ and $P(Y|\neg X)P(\neg X)$
② normalize the two values so that they sum to 1.

know these 3, then we know $P(X|Y)$, $P(\neg X|Y)$

Eg. (Tips: we can reorder/derive new P based on Given P())

CQ: Calculate a joint probability

$P(A \wedge W \wedge \neg G) = P(A)\,P(W|A)\,P(\neg G|A \wedge W)$
$= 0.1 * 0.9 * (1 - 0.3) = 0.063$

order: $A \to W \to \neg G$

CQ: What is probability that **the alarm is going,**
Dr. Watson is calling and Mrs. Gibbon is NOT calling?

(A) 0.060
(B) 0.061
(C) 0.062
(D) 0.063
(E) 0.064

$P(A) = 0.1$
$P(W|A) = 0.9$
$P(W|A \wedge \neg G) = 0.9$
$P(G|A) = 0.3$
$P(G|A \wedge W) = 0.3$

$P(G|A \wedge W) + P(\neg G|A \wedge W) = 1$

$P(A \wedge W \wedge \neg G) = P(A)\,P(\neg G|A)\,P(W|A \wedge \neg G)$
$= 0.1 * (1 - 0.3) * 0.9 = 0.063$

order: $A \to \neg G \to W$

Eg.

CQ: Flipping a conditional probability

$P(\neg A|W) = \frac{P(W|\neg A)\,P(\neg A)}{P(W)} = \frac{0.4 * (1 - 0.1)}{0.45} = 0.8$

$P(W) = P(W|A)\,P(A) + P(W|\neg A)\,P(\neg A) = 0.9 * 0.1 + 0.4 * 0.9 = 0.09 + 0.36 = 0.45$

CQ: What is the probability that **the alarm is NOT going given that Dr. Watson is calling**?

(A) 0.6
(B) 0.7
(C) 0.8
(D) 0.9
(E) 1.0

$P(A) = 0.1$
$P(W|A) = 0.9$
$P(W|\neg A) = 0.4$

$P(W|A)\,P(A) = 0.9 * 0.1 = 0.09$
$P(W|\neg A)\,P(\neg A) = 0.4 * 0.9 = 0.36$
$0.09 + 0.36 = 0.45$

$P(A|W) = \frac{0.09}{0.45} = 0.2$ $\quad P(\neg A|W) = \frac{0.36}{0.45} = 0.8$

Universal approach

Step 1: Conditional Probability

To calculate a conditional probability, we are looking to find out how likely an event is to happen given that another event has already happened. Mathematically, this is expressed as $P(A|B)$, which reads as "the probability of A given B". Using the product rule in reverse, we can say:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

where:

- $P(A \cap B)$ is the joint probability of both events A and B occurring.
- $P(B)$ is the probability of event B occurring.

Step 2: Joint Probability (Not Involving All Variables)

Sometimes we want to calculate the probability of two events occurring together but we don't have direct information about their joint occurrence. We can use the sum rule in reverse to calculate it. For instance, if we want to calculate $P(A \cap B)$, and we know how A relates to other variables, we can sum over all the possible values of these other variables (let's call one of them C):

$$P(A \cap B) = \sum_C P(A \cap B \cap C)$$

Step 3: Every Joint Probability (Involving All Variables)

For a complete joint probability distribution involving all variables in our model, we use the chain rule. The chain rule is a way to decompose the joint probability of a set of events into conditional probabilities. For events A, B, and C, the chain rule tells us:

$$P(A \cap B \cap C) = P(A) \cdot P(B|A) \cdot P(C|A \cap B)$$

Eg.

Calculate $P(A|E)$
Given $P(A)$ $P(B|A), P(B|\neg A)$
$P(E|A \wedge B), P(E|A \wedge \neg B), P(E|\neg A \wedge B), P(E|\neg A \wedge \neg B)$

Step 1: $P(A|E) = \frac{P(A \wedge E)}{P(E)}$

Step 2: $P(A \wedge E) = P(A \wedge E \wedge B) + P(A \wedge E \wedge \neg B)$
$P(E) = P(E \wedge A \wedge B) + P(E \wedge A \wedge \neg B)$
$+ P(E \wedge \neg A \wedge B) + P(E \wedge \neg A \wedge \neg B)$

Step 3: $P(E \wedge \neg A \wedge B) = P(\neg A \wedge B \wedge E)$
$= P(\neg A) * P(B|\neg A) * P(E|\neg A \wedge B).$

∵ In Bay, we assume attributes are independent given label Y

∴ $P(A \wedge B \wedge Y) = P(Y) \cdot P(A|Y) \cdot P(B|Y)$

OR $P(Y|A,B)$

Laplace Smooth:

$P_{LAP,k}(x|y) = \frac{c(x,y) + k}{c(y) + k|X|}$ ← add

◦ Independence

— Unconditional Independence

Definition ((unconditional) independence)

X and Y are (unconditionally) independent iff

$$P(X|Y) = P(X)$$

$$P(Y|X) = P(Y)$$

$$P(X \wedge Y) = P(X)P(Y)$$

Learning Y does NOT influence your belief about X.

— Conditional Independence

Definition (conditional independence)

X and Y are conditionally independent given Z if

$$P(X|Y \wedge Z) = P(X|Z).$$

$$P(Y|X \wedge Z) = P(Y|Z).$$

$$P(Y \wedge X|Z) = P(Y|Z)P(X|Z).$$

Learning Y does NOT influence your belief about X if you already know Z.

Eg.

CQ: Deriving a compact representation

n random variables, $2^n - 1$ probabilities.

Boolean.

CQ: Consider a model with three random variables, A, B, C.

1. What is the minimum number of probabilities required to specify the joint distribution? 7 probabilities.

$P(A \wedge B \wedge C) = P(A) * P(B|A) * P(C|A \wedge B)$ $2^3 = 8$ probabilities.

2. Assume that A, B, and C are independent. What is the minimum number of probabilities required to specify the joint distribution? 3 probabilities.

$P(A \wedge B \wedge C) = P(A) * P(B) * P(C)$

1+1+1=3

1+2+4 = 7

3. Assume that A and B are conditionally independent given C. What is the minimum number of probabilities required to specify the joint distribution? 5 probabilities

$P(A \wedge B \wedge C) = P(C) * P(A|C) * P(B|C \wedge A)$

$= P(C) * P(A|C) * P(B|C)$

1 + 2 + 2 = 5

◦ Bayesian Network

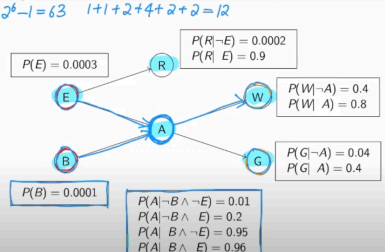
A Bayesian Network

- is a compact version of the joint distribution, and
- takes advantage of the unconditional and conditional independence among the variables.

Eg. no circle, →

A Bayesian Network for the Holmes Scenario

$2^6 - 1 = 63$ 1+1+2+4+2+2 = 12



— Representing joint distribution

Can Compute each joint P use formula:

$$P(X_n \wedge \dots \wedge X_1) = \prod_{i=1}^{n} P(X_i | \text{Parents}(X_i))$$

Eg.

Example: What is the probability that

- The alarm has sounded,
- Neither a burglary nor an earthquake has occurred,
- Both Watson and Gibbon call and say they hear the alarm, and
- There is no radio report of an earthquake?

$P(\neg B \wedge \neg E \wedge A \wedge \neg R \wedge W \wedge G)$

$= P(\neg B) P(\neg E) P(A|\neg B \wedge \neg E) P(\neg R|\neg E) P(W|A) P(G|A)$

$= (1 - 0.0001)(1 - 0.0003)(0.01)(1 - 0.0002)(0.4)(0.8)$

$= 0.0032$

— Un/Conditional Independence relationship in Bayes

Eg. Causal chain

1. Burglary → Alarm → Watson

Burglary, Watson independent?

No

B, W Conditionally inde?

Yes

2. Alarm happen → Watson W call / Gibbon G call; Alarm → Gibbon; call SB

W, G (uncon) independent?

No

W, G Con inde?

Yes

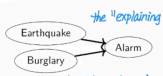
3. Earthquake → Alarm ← Burglary

E, B independent?

Yes

CQ: Are Earthquake and Burglary conditionally independent given Alarm?

the "explaining away" effect



(A) Yes

(B) No

Suppose that the Alarm is going off.

If an earthquake is happening, then it is less likely that the Alarm is caused by a Burglary.

If an earthquake is not happening, then it is more likely that a Burglary is happening and causing the alarm to go off.

— Bayes Net: D-Seperation (Independent or Not)

Definition (D-Separation)

E d-separates X and Y iff E blocks every un-directed path between X and Y.

If E d-separates X and Y, then X and Y are independent given E.

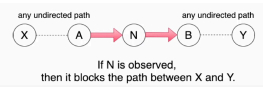
path we consider

① un-direct

② multiple paths

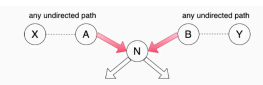
③ multiple node

X - - [Scenario] - - Y



If N is observed, then it blocks the path between X and Y.

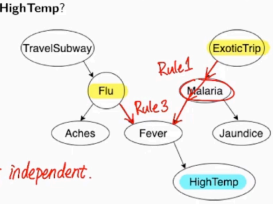
If N is observed, then it blocks the path between X and Y.



If N and N's descendants are NOT observed, then they block the path between X and Y.

Eg.

CQ 3b: Are **Flu** and **ExoticTrip** conditionally independent given **HighTemp**?



Not independent.

Construction Bayes Net

For a joint probability distribution, there are multiple correct Bayesian networks.

A Bayesian network is correct if every independence relationship the network represents is correct. ⇝ exists in the joint distribution.

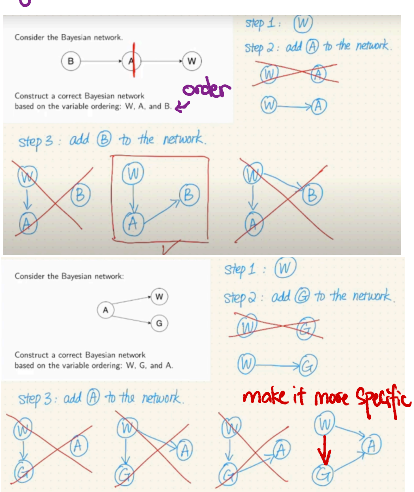
We prefer one Bayesian network over another one if the former requires fewer probabilities.

Joint P dis { BN1, BN2, BN3 } , BN# with fewer P is good,

Inde correct → Network correct

1. Determine the set of variables for the domain.
2. Order the variables $\{X_1, \ldots, X_n\}$.
3. For each variable X_i in the ordering,
   - 3.1 **Choose the node's parents:** Choose the smallest set of parents from $\{X_1, \ldots, X_{i-1}\}$ such that given $Parents(X_i)$ X_i is independent of all the nodes in $\{X_1, \ldots, X_{i-1}\} - Parents(X_i)$. Formally,
     $$P(X_i | Parents(X_i)) = P(X_i | X_{i-1} \wedge \cdots \wedge X_1).$$
   - 3.2 Create a link from each parent of X_i to the node X_i.
   - 3.3 Write down the conditional probability table $P(X_i | Parents(X_i))$.

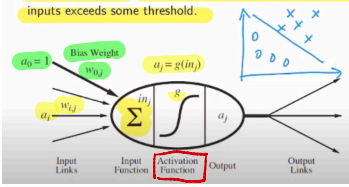
Eg.



Neural Network

– mathematical model

- A linear classfier — it "fires" when a linear combination of its inputs exceeds some threshold.



- Neuron j computes a weighted sum of its input signals. $in_j = \sum_{i=0}^{n} w_{ij} a_i$.
- Neuron j applies an activation function g to the weighted sum to derive the output. $a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$.

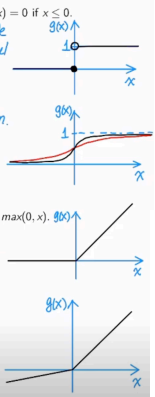
About Activation Function:

What are some desirable properties of the activation function?

- It should be non-linear.
  combining linear functions will not produce a non-linear function.
  complex relationships are often non-linear.
- It should mimic the behaviour of real neurons.
  If the weighted sum of input signals is large enough, the neuron fires. Otherwise, it does not fire.
- It should be differentiable almost everywhere.
  learn a neural network using gradient descent, which requires the activation function to be differentiable.

Common Activation Function:

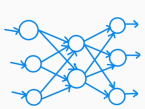
- Step function: $g(x) = 1$ if $x > 0$. $g(x) = 0$ if $x \leq 0$.
  simple to use, not differentiable
  not used in practice, but useful to explain concepts.
- Sigmoid function: $g(x) = \frac{1}{1 + e^{-kx}}$.
  can approximate the step function.
  clear, bounded prediction.
  differentiable.
  "vanishing gradient" problem.
- Rectified linear unit (ReLu): $g(x) = max(0, x)$.
  computationally efficient.
  non-linear and differentiable.
  the dying ReLu problem.
- Leaky ReLU: $g(x) = (0.1 * x, x)$.
  enables learning for negative input values.



Network

Connecting the neurons together into a network

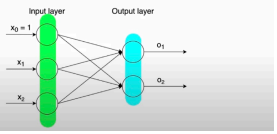
- Feed-forward network
  directed acyclic graph
  no loops.
  a function of its inputs.



- Recurrent network
  has loops.
  has memory.
  better model of human brain,
  but more difficult to interpret and train.

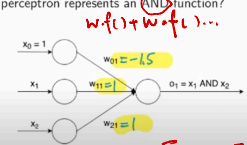
Perceptrons

- Single-layer feed-forward neural network
- The inputs are connected directly to the outputs.
- Can represent logical functions, e.g. AND, OR, and NOT.

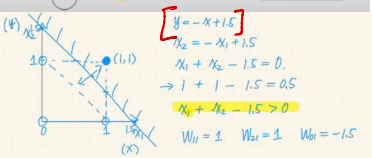


Eg. logical GATE

CQ: Consider the perceptron below where the activation function is the step function ($g(x) = 1$ if $x > 0$. $g(x) = 0$ if $x \leq 0$.). What should the weights w_{01}, w_{11} and w_{21} be such that the perceptron represents an AND function?



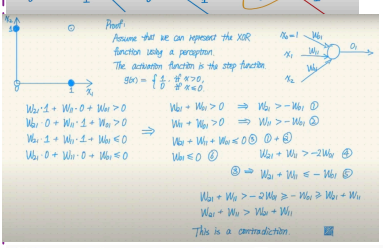
x_1	x_2	o_1
0	0	0
0	1	0
1	0	0
1	1	1



Eg.

CQ: Why can't a perceptron represent XOR?

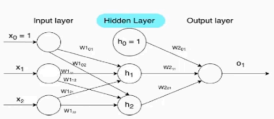
a perceptron is a linear classifier.
XOR is not linearly separable.

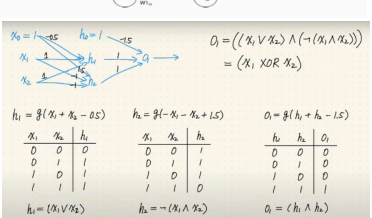


XOR as a 3-Layer Neural Network

$O_1 = ((X_1 \vee X_2) \wedge (\neg (X_1 \wedge X_2)))$

Can you come up with the weights such that the following network represents the XOR function?





Gradient Descent

Gradient Descent

expected outputs based on training data
actual outputs based on training data
network.

"Walking downhill and always taking a step in the direction that goes down the most."

error

- A local search algorithm to find the minimum of a function.
- Steps of the algorithm:
  - Initialize weights randomly.
  - Change each weight in proportion to the negative of the partial derivative of the error with respect to the weight.

$$w := w - \eta \frac{\partial error}{\partial w}$$

sum over all training examples.

  - η is the learning rate.
  - Terminate after some number of steps when the error is small or when the changes get small.

Weight

How do we update the weights based on the data points?

cheaper steps, weights become more accurate more quickly.
not guaranteed to converge.

- Gradient descent updates the weights after sweeping through all the examples.
- To speed up learning, update weights after each example.
  - Incremental gradient descent
    update weights after each example.
    - Stochastic gradient descent
      same as incremental gradient descent except that each example is chosen randomly.
- Trade off learning speed and convergence.
  - Batched gradient descent
    update weights after a batch of examples.
    start w/ small batches to learn quickly.
    then, increase batch size so it converges.

All

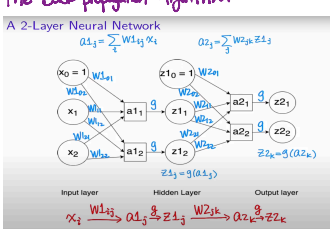
Tips: How to derive Gradient Descent

Why update the weight proportional to the negative of the partial derivative?

- Suppose that we want to find the minimum of $y = x^2$.
- Start with $x = x_0$.
- In what direction should we change the value of x?
  Change x in the direction of the negative of the partial derivative.
- By what amount should we change the value of x?
  What is the step size?
  Take a step in proportion to the magnitude of the partial derivative.

The Back propagation Algorithm

A 2-Layer Neural Network



The Back-propagation Algorithm

- An efficient method of calculating the gradients in a multi-layer neural network.
- There are some training examples $(\vec{x}_n, \vec{y}_n)$ and an error/loss function $E(z2, y)$. Perform 2 passes.
  - Forward pass: compute the error E given the inputs and the weights.
  - Backward pass: compute the gradients $\frac{\partial E}{\partial W2_{jk}}$ and $\frac{\partial E}{\partial W1_{ij}}$.
- Update each weight by the sum of the partial derivatives for all the training examples.

Forward Pass

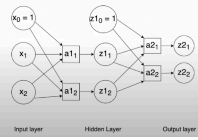
Forward Pass for a 2-layer Network

Calculate the values of $z1_j$ and $z2_k$ and E.

$$a1_j = \sum_i x_i W1_{ij} \qquad z1_j = g(a1_j) \tag{1}$$

$$a2_k = \sum_j z1_j W2_{jk} \qquad z2_k = g(a2_k) \tag{2}$$

$$E(z2, y) \tag{3}$$



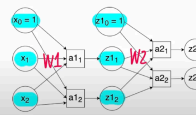
Backward

Backward Pass for a 2-layer Network

Calculate the gradients for $W1_{ij}$ and $W2_{jk}$.

$$\frac{\partial E}{\partial W2_{jk}} = \frac{\partial E}{\partial a2_k} z1_j = \delta 2_k z1_j \qquad \delta 2_k = \frac{\partial E}{\partial z_k} g'(a2_k) \tag{4}$$

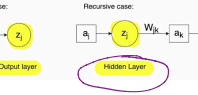
$$\frac{\partial E}{\partial W1_{ij}} = \frac{\partial E}{\partial a1_j} x_i = \delta 1_j x_i \qquad \delta 1_j = \left(\sum_k \delta 2_k W2_{jk}\right) g'(a1_j) \tag{5}$$



Recursive relationship

For unit j, $\delta_j = \frac{\partial E}{\partial a_j}$.

$$\delta_j = \begin{cases} \frac{\partial E}{\partial z_j} \times g'(a_j), & \text{base case, } j \text{ is an output unit} \\ \left(\sum_k \delta_k W_{jk}\right) \times g'(a_j), & \text{recursive case, } j \text{ is a hidden unit} \end{cases} \tag{6}$$



Deriving the expressions for $\partial E / \partial W2_{jk}$

$x_i \xrightarrow{W1_{ij}} a1_j \xrightarrow{g} z1_j \xrightarrow{W2_{jk}} a2_k \xrightarrow{g} z2_k$

$$\frac{\partial E}{\partial W2_{jk}} = \frac{\partial E}{\partial z2_k}\frac{\partial z2_k}{\partial a2_k}\frac{\partial a2_k}{\partial W2_{jk}} = \frac{\partial E}{\partial z2_k} g'(a2_k) z1_j$$

an example w/ $j=1$ and $k=2$.

$$\frac{\partial E}{\partial W2_{12}} = \frac{\partial E}{\partial z2_2}\frac{\partial z2_2}{\partial a2_2}\frac{\partial a2_2}{\partial W2_{12}} = \frac{\partial E}{\partial z2_2} g'(a2_2) z1_1$$

chain rule $\frac{d}{dx} g(f(x)) = \frac{dg}{df}\frac{df}{dx}$

$$E = \frac{1}{2}\sum_k (z2_k - y_k)^2 \Rightarrow \frac{\partial E}{\partial z2_k} = (z2_k - y_k) \qquad z2_k = g(a2_k) \Rightarrow \frac{\partial z2_k}{\partial a2_k} = g'(a2_k)$$

$$g(x) = \frac{1}{1+e^{-x}} \Rightarrow g'(x) = g(x)(1-g(x)) \qquad a2_k = \sum_j W2_{jk} z1_j \Rightarrow \frac{\partial a2_k}{\partial W2_{jk}} = z1_j$$

Deriving the expressions for $\partial E / \partial W1_{ij}$

$x_i \xrightarrow{W1_{ij}} a1_j \xrightarrow{g} z1_j \xrightarrow{W2_{jk}} a2_k \xrightarrow{g} z2_k$

$$\frac{\partial E}{\partial W1_{ij}} = \left(\sum_k \frac{\partial E}{\partial z2_k}\frac{\partial z2_k}{\partial a2_k}\frac{\partial a2_k}{\partial z1_j}\right)\frac{\partial z1_j}{\partial a1_j}\frac{\partial a1_j}{\partial W1_{ij}}$$

$$= \left(\sum_k \frac{\partial E}{\partial z2_k} g'(a2_k) W2_{jk}\right) g'(a1_j) x_i$$

an example w/ $i=1$ and $j=2$.

$$\frac{\partial E}{\partial W1_{12}} = \left(\sum_k \frac{\partial E}{\partial z2_k} g'(a2_k) W2_{2k}\right) g'(a1_2) x_1$$

Defining the delta values

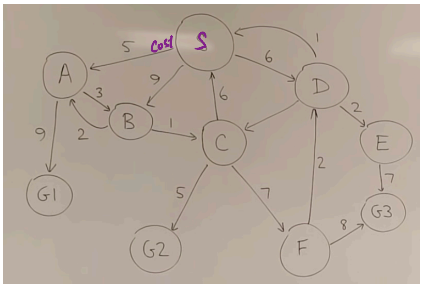
① $\frac{\partial E}{\partial W2_{jk}} = \underbrace{\frac{\partial E}{\partial z2_k} g'(a2_k)}_{\delta 2_k} z1_j = \delta 2_k z1_j$

② $\frac{\partial E}{\partial W1_{ij}} = \underbrace{\left(\sum_k \frac{\partial E}{\partial z2_k} g'(a2_k) W2_{jk}\right) g'(a1_j)}_{\delta 1_j} x_i = \delta 1_j x_j$

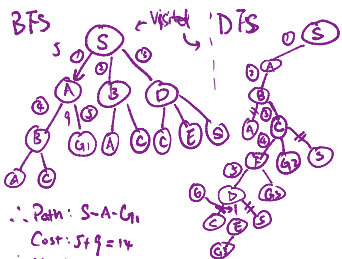
① $\delta 2_k = \frac{\partial E}{\partial z2_k} g'(a2_k)$

② $\delta 1_j = \left(\sum_k \delta 2_k W2_{jk}\right) g'(a1_j)$

Search



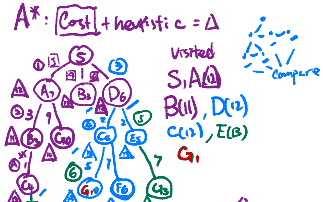
BFS ← Visited → DFS



∴ Path: S-A-G1
Cost: 5+9 = 14
∴ Its shortest, but not less cost

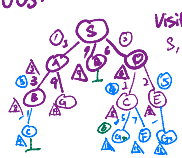
∴ Path: S-A-B-C-D-E-G3

A*: Cost + heuristic = Δ

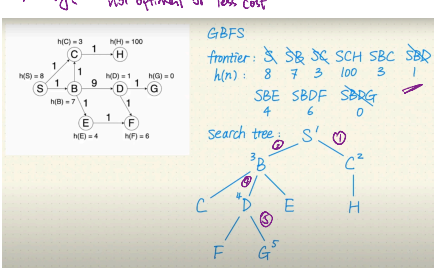


Path: S-D-C-G1
B cost: 5 + 3 + 3

UCS:



Greedy: not optimal or less cost



CONSTRAINT SATISFACTION

- A set of variables, $X_1, X_2, X_3 \dots X_n$
- Each variable X_i has a non-empty domain of possible values, e.g. $X_1 = \{1, 2, \dots\}$
- A set of constraints, e.g. $X_1 \neq 3$ (unary), $X_3 > X_4$ (binary)
- Solution: assign a value to each variable such that none of the constraints are broken
- Simplest algorithm: backtracking

$A = \{1, 2, 3\}$
$B = \{1, 2, 3\}$
$C = \{1, 2, 3\}$
$A > B$
$B \neq C$
$A \neq C$

$A=1, B=1$ ×
$A=1, B=2$ ×
$A=1, B=3$ ×
$A=2, B=1$ ×
$A=2, B=1, C=2$ ×
$A=2, B=1, C=3$ ✓

THE AC-3 ALGORITHM

1. Turn each binary constraint into two arcs e.g. $A \neq B$ becomes $A \neq B$ and $B \neq A$
2. Add all the arcs to an agenda
3. Repeat until agenda is empty
   - take an arc (X_i, X_j) off the agenda and check it
   - for every value of X_i, there must be some value of X_j
   - remove any inconsistent values from X_i
   - if X_i has changed, add all arcs of the form (X_k, X_i) to the agenda

Eg.

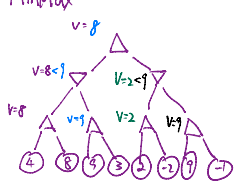
$A = \{1, 2, 3\}$
$B = \{1, 2, 3\}$
$C = \{1, 2, 3\}$
$A > B$
$B = C$

↓ Start Agenda:
~~A>B~~
~~B<A~~ ← B<A changing value
~~B=C~~
~~C=B~~
~~A>B~~
~~B=C~~
nothing change

Arcs
$A > B$
$B < A$
$B = A$
$C = B$

Arc Consistency ⇒ ∴ A{2,3}, B{1,2}, C{1,2}

MinMax

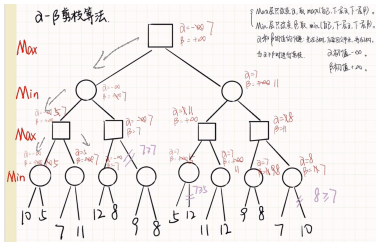


MinMax - α-β Pruning

MINIMAX WITH α-β PRUNING

```
Max-Value (s, α, β):
  if terminal(s) return U(s)
  v = - ∞
  for c in next-states(s):
    v' = Min-value (c, α, β)
    if v' > v, v = v'
    if v' ≥ β, return v
    if v' > α, α = v'
  return v

Min-Value (s, α, β):
  if terminal(s) return U(s)
  v = + ∞
  for c in next-states(s):
    v' = Max-value (c, α, β)
    if v' < v, v = v'
    if v' ≤ α, return v
    if v' < β, β = v'
  return v
```

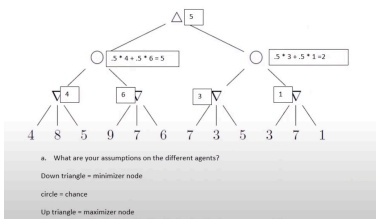


- Evaluation Function
  - ↳ Produce estimates of value of non-terminal utilities

$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$

Eg. Eval(s) = 2 · agent_kings(s) + agent_Pawns(s) − 2 · opponent_kings(s) − opponent_Pawns(s)

- Expectmax



Multiagent