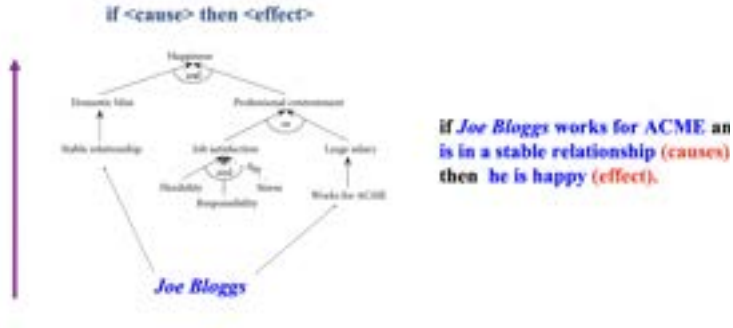
# 1 Lecture 1

A production rule has the form:
if <condition> then <conclusion>

- Deduction(su suy dien): Rules that make up inference network can be used to link cause and effect:
  if <cause> then <effect>
  Example: if Joe Bloggs works for ACME and is in a stable relationship (the causes), then he is happy (the effect).




- Abduction (giai thich suy luan ):
  - Many problems, such as diagnosis, involve reasoning in reverse, i.e, find a cause, given an effect.
  - Given observation Joe Bloggs is happy, infer by abduction Joe Bloggs enjoys domestic bliss and professional contentment.
- Induction (Quy nap):
  - If we have many examples of cause and effect, infer the rule that links them.
  - For instance: if every employee of ACME earns a large salary, infer rule r1_1:
    If the employer of Person is acme THEN the salary of Person becomes large.

conclusion:

- deduction: cause + rule => effect
- abduction: effect + rule => cause
- induction: cause + effect => rule

SEMANTIC NETWORKS

- Facts, Objects, Attributes and Relationships
- Attributes and relationships can be represented as a network, known as an associative network or semantic network
- We can build a model of the subject area of interest

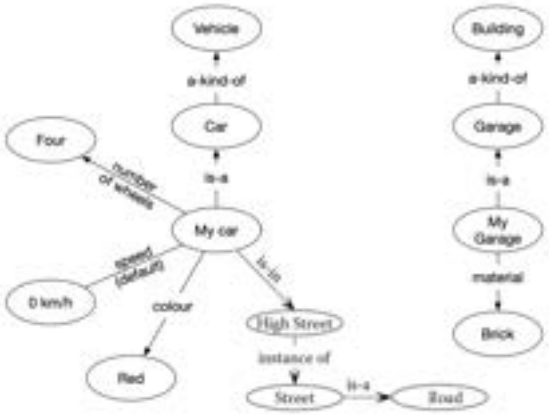
EXAMPLE

- My car is a car
- A car is a vehicle
- A car has four wheels
- A car's speed is 0 mph
- My car is red
- My car is in my garage
- My garage is a garage
- A garage is a building
- My garage is made from brick
- My car is in High Street
- High Street is a street
- A street is a road

Underline = object (instance)

- My car is a **car** (static relationship)
- A car is a **vehicle** (static relationship)
- A car has four wheels (static attribute)
- A car's speed is 0 mph (default attribute)
- My car is red (static attribute)
- My car is in my garage (default relationship)
- My garage is a **garage** (static relationship)
- A garage is a **building** (static relationship)
- My garage is made from brick (static attribute)
- My car is in High Street (transient relationship)
- High Street is a **street** (static relationship)
- A street is a **road** (static relationship)

## A semantic network (with a default)



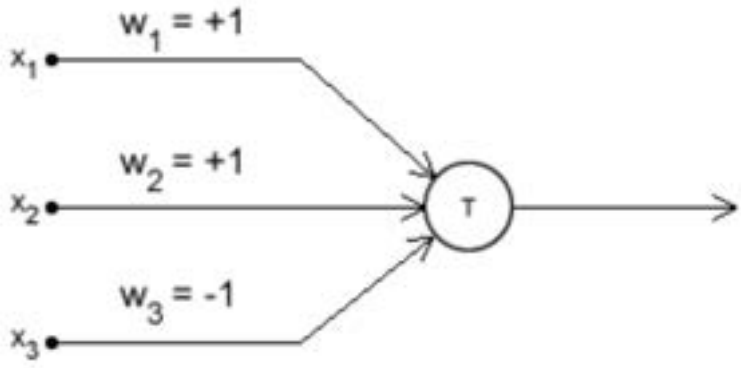

**Example**

Other properties:

- Implication: $p \rightarrow q \equiv \neg p \vee q$
- Double negation: $\neg\neg p \equiv p$
- Contrapositive: $(p \rightarrow q) \equiv (\neg q \rightarrow \neg p)$
- De Morgan's: $\neg(p \vee q) \equiv \neg p \wedge \neg q$

# 2 Lecture 2 - Neural network

- McCulloch-Pitts model
  - inputs 0 or 1
  - outputs 0 or 1
  - Input can be either excitatory or inhibitory.
- Summing inputs
  - If input is 1, and is excitatory, add 1 to sum
  - If input is 1, and is inhibitory, subtract 1 from sum
- Threshold,
  - if sum ¡ threshold Theta, output 0.
  - Otherwise output 1.




$$sum = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + \ldots$$

**if** *sum* $< \theta$ **then** *output* is 0
**else** *output* is 1.

## Single layer perceptron

If we have a function ($G_i$) for each class ($S_i$). The classification rule is:

$$u \in s_i \iff g_i(u) > g_j(u); \quad j \neq i$$

For 2-classes problems, it can be reduced to one function as:

$$g(u) = g_1(u) - g_2(u), \text{ then } u \in s_1, g(u) > 0 \text{ and } u \in s_2 \text{ otherwise}$$

$$\hat{y}(x) = g(X \cdot W^t) \rightarrow \text{perceptron output}$$
$$y(x) \rightarrow \text{target output} \in \{-1, 1\}$$

$$\Delta W_k = \alpha(y(x_k) - \hat{y}(x_k)) \cdot x_k$$
$$W_{k+1} = W_k + \Delta W_k$$

$$\Delta W_k = 0 \rightarrow \text{if the perceptron output is}$$
$$\Delta W_k = +2\alpha x_k \rightarrow \text{if } y(x_k) = 1 \text{ and } \hat{y}(x_k$$
$$\Delta W_k = -2\alpha x_k \rightarrow \text{if } y(x_k) = -1 \text{ and } \hat{y}($$

$$\alpha > 0; \{x_1, ..., x_k\} \text{ input sequence}$$

Learning rule:

Example:

| b | w1 | w2 | x1 | x2 | class | predicted | sgn | |
|---|---|---|---|---|---|---|---|---|
| | 1,5 | 1 | 0 | 0 | 0 | 1 | 1,5 | 1 |
| | 1,5 | 1 | 0 | 1 | 2 | -1 | 2,5 | 1 |
| | -0,5 | -1 | -4 | 2 | -1 | -1 | 1,5 | 1 |
| | -2,5 | -5 | -2 | -2 | 1 | 1 | 5,5 | 1 |
| | -2,5 | -5 | -2 | 0 | 0 | 1 | -2,5 | -1 |
| | -0,5 | -5 | -2 | 1 | 2 | -1 | -9,5 | -1 |
| | -0,5 | -5 | -2 | 2 | -1 | -1 | -8,5 | -1 |
| | -0,5 | -5 | -2 | -2 | 1 | 1 | 7,5 | 1 |
| | -0,5 | -5 | -2 | 0 | 0 | 1 | -0,5 | -1 |
| | 1,5 | -5 | -2 | 1 | 2 | -1 | -7,5 | -1 |

## Network design

Normalization of variables: It is necessary when variables with different units and, therefore, potentially different magnitudes are involved. Sometimes, the magnitudes can differ by several orders of magnitude.

$$x_n = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, x \in [0, 1] \quad (1)$$

$$x_n = 2 \cdot \frac{x - x_{\min}}{x_{\max} - x_{\min}} - 1, x \in [-1, 1] \quad (2)$$

$$N_w < (\text{Number of samples})/10 \quad (3)$$

$$N_w = (N_i + 1) \cdot N_h + (N_h + 1) \cdot N_o \quad (4)$$

Example An MLP with 3 inputs, 4 units in its hidden layer, and 2 outputs, has a number of parameters:
$N_w = (3+1)*(4) + (4+1)*2 = 26$

## Multi layer perceptron

Feedforward:

$$h(1) = w_1 \cdot x_1 + w_2 \cdot x_2 + b_1 \quad (5)$$

$$a(1) = f(h(1)) \quad \text{with } f = \text{sigmoid or tanh or } \ldots \quad (6)$$

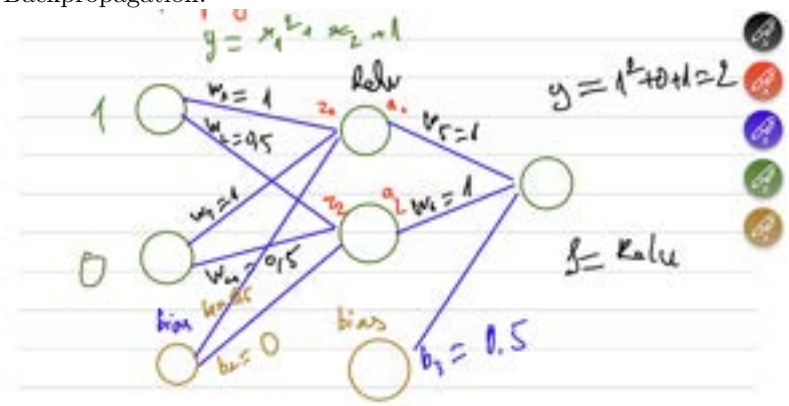
Tanh function:

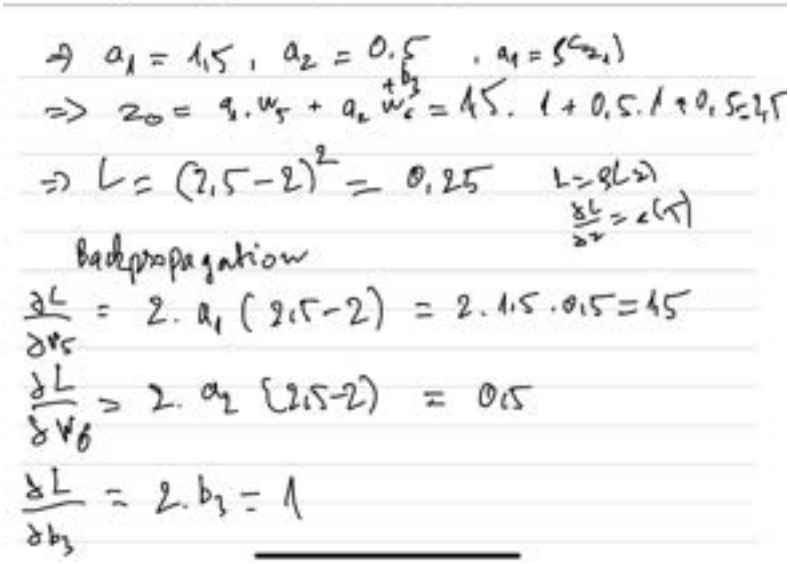
$$tanh(x) = \frac{e^x - e(-x)}{e^x + e^(-x)} = 2 \cdot \frac{1}{1 + e^{-2 \cdot x}} - 1(-1 \text{ to } 1)$$

Sigmoid:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Backpropagation:





$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} = (2 \cdot L \cdot w_5) \cdot (a_1'(z_1) \cdot (x_1)) \quad (7)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} = (2 \cdot L \cdot w_6) \cdot (a_2'(z_2) \cdot (x_1)) \quad (8)$$

# 3 Lecture 3 - Search

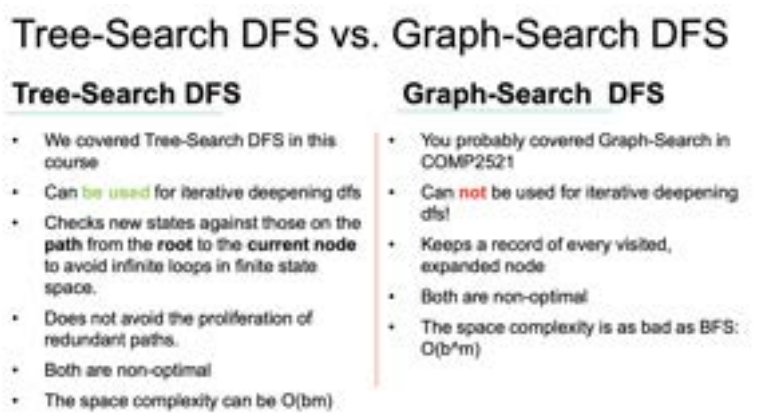
## BFS

- Complete? Yes (if breadth, , is finite, the shallowest goal is at a fixed depth, , and will be found before any deeper nodes are generated)
- Time?

$$1 + b^1 + b^2 + ... + b^d = \frac{b^{d+1} - 1}{b - 1} = O(b^d)$$

- Space? O($b^d$) keeps every node in memory; generate all nodes up to level d. It is the big problem for BFS. It grows exponentially with depth When a node is inserted in the list, it is ***GENERATED***.
  When a node is removed from the list, it is in the process of being ***EXPANDED***.

## DFS




## Iterative Deepening Depth-First Search (IDDFS)

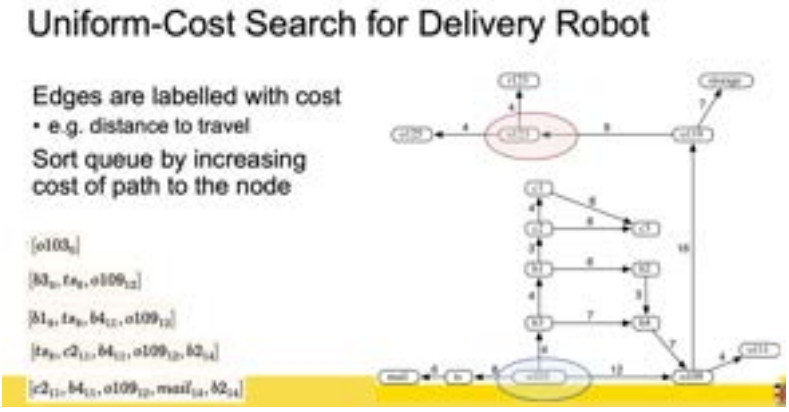
- Complete? Yes
- Time? O($b^d$)
- Space? O(bd)

## Bidirectional Search

- Search both forward from the initial state and backward from the goal. Stop when the 2 searches meet in the middle
- Need efficient way to check if a new node appears in the other half of the search.
- Assume branching factor = b in both directions and that there is a solution at depth = d: Then bidirectional search finds a solution in O($2b^{d/2}$) = O($b^{d/2}$) time steps.

## Uniform-Cost Search, Lowest-cost-first Search

Cost of a path is the sum of the costs of its arcs: An optimal solution has minimum cost




- Complete? Yes, if b is finite and if transition >epsilon with epsilon >0
- Time? Worst case, O($b^{[C^*/epsilon]}$) where C* = cost of the optimal solution every transition costs at least eps, cost per step is C*/eps
- Space? O($b^{[C^*/epsilon]}$), $b^{[C^*/epsilon]}$ = $b^d$ if all step costs are equal

conclusion:

| Criterion | Breadth-First | Uniform-Cost | Depth-First | Depth-Limited | Iterative Deepening |
|---|---|---|---|---|---|
| Time | $O(b^d)$ | $O(b^{\lceil C^*/\epsilon \rceil})$ | $O(b^m)$ | $O(b^k)$ | $O(b^d)$ |
| Space | $O(b^d)$ | $O(b^{\lceil C^*/\epsilon \rceil})$ | $O(bm)$ | $O(bk)$ | $O(bd)$ |
| Complete? | Yes[1] | Yes[2] | No | No[4] | Yes[1] |
| Optimal ? | Yes[3] | Yes | No | No | Yes[3] |

$b$ = branching factor, $d$ = depth of the shallowest solution,
$m$ = maximum depth of the search tree, $k$ = depth limit.

1 = complete if b is finite
2 = complete if b is finite and step costs >or = eps with eps >¿ 0
3 = optimal if actions all have the same costs.
4 = incomplete if the goal is not within the depth bound.

## Greedy Best-First Search

Always select node closest to goal according to heuristic function h(n) is estimated cost to goal. h(n) = 0 if n is a goal state "Greedy" algorithm takes "best" node first.
f(n) = h(n)

- Complete? No. Can get stuck in loops.
- Time? $O(b^m)$ where m is the maximum depth in search space.
- Space? $O(b^m)$, (retains all nodes in memory)
- Optimal? no

## A* Search

f(n)=cost of the cheapest path from s to t via n: f(n) = g(n) + h(n).
g(n) is the cost the path from s to n.
h(n) is an estimate of the cost of an optimal path from n to t.

# 4 Lecture 4 - RL

## Action-value estimation methods

- We denote the real action value as q*(a)
- We denote the estimated value at time-step t as $Q_t(a)$.
- Simple Estimation: to average received rewards when action a has been selected Ka times.

$$Q_t(a) = \frac{R_1 + R_2 + ... + R_{K_a}}{K_a}$$

If Ka = 0, $Q_t(a)$ is defined with an arbitrary value, e.g 0.
As Ka →inf, $Q_t(a)$ converges to q*(a).
Greedy method:

- The simplest way to choose an action: the action with the highest estimated value.
- $A_t^*$ where $Q_t(A_t^*) = \max(Q_t(a))$.

eps-greedy method:

- A simple alternative: to choose the best action most of the time, and sometimes (with a small probability eps) a random one.
- $Q_t(a)$ converges to q*(a) with probability 1-eps.

Softmax method:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{i=1}^{n} e^{Q_t(i)/\tau}}$$

## Non-stationary problems

- Methods of average are appropriate for stationary problems.
- In non-stationary problems, make sense to give more weight to more recent rewards than the past ones.
- Using a constant parameter *step-size*, 0 < α ≤ 1:

$$Q_{k+1} = Q_k + \alpha\left[R_k - Q_k\right]$$

## Returns

- Tasks intended to be performed continuously without limit are referred to as continuous tasks (or non-episodic).
- The return could be infinite, given that T = ∞.
- In this case, the agent maximises the discounted rewards, choosing actions to maximise the discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- Discount rate 0 ≤ γ < 1. Determine the present value of future rewards. If γ = 0, the agent is myopic. If γ → 1 the agent is foresighted.

## Returns

| γ | Reward sequence | Return |
|---|---|---|
| 0.5 | 1 0 0 0 ..... | 1 |
| 0.5 | 0 0 2 0 0 0 ..... | 0.5 |
| 0.9 | 0 0 2 0 0 0 ..... | 1.62 |
| 0.5 | -1 2 6 3 2 0 0 0 ..... | 2 |

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

## Value function

### Value function

- The value of a terminal state, if any, is zero.
- The value of taking action a in state s under policy π, is denoted $q_\pi(s,a)$ or $Q^\pi(S,A)$:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

- Value functions $v_\pi(s)$ and $q_\pi(s,a)$ can be estimated from experience.
- If there are many states, it's impractical to keep values for each state.
- In this case, parameterized function approximators are used to keep $v_\pi(s)$ and $q_\pi(s,a)$.

### Value function

- Try to maximise expected future reward:

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

- $V^\pi(s_t)$ is the value of state $s_t$ under policy $\pi$
- $\gamma$ is a discount factor [0..1]

### Value function

- $V^\pi(s)$ is the expected value of following policy $\pi$ in state $s$
- $V^*(s)$ be the maximum discounted reward obtainable from $s$.
  - i.e. the value of following the optimal policy
- We make the simplification that actions are deterministic, but we don't know which action to take.
  - Other RL algorithms relax this assumption

## $Q$-values

- How to choose an action in a state?

$$Q(s,a) = r(s,a) + \gamma V^*(s')$$

- The $Q$-value for an action, $a$, in a state, $s$, is the immediate reward for the action plus the discounted value of following the optimal policy after that action
- $V^*$ is value obtained by following the optimal policy
- $s' = \delta(s,a)$ is the succeeding state, assuming the optimal policy

## Temporal-difference (TD) prediction

- TD is one central and novel idea in RL.
- Monte Carlo-like methods must wait until the end of the episode to update $V(S_t)$ – (only at that point $G_t$ is known):

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

- The simplest TD method is called TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

- Approximations can be on-policy or off-policy.
- TD control learns an action-value function instead of a state-value function.
- We estimate $q_\pi(s,a)$ for the current policy π.
- Therefore, we consider transitions from state-action pair to state-action pair.

## Sarsa

- Updates after each transition from a non-terminal $S_t$.
- If $S_{t+1}$ is terminal, $Q(S_{t+1}, A_{t+1})$ is defined as zero.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

## Q-learning

## Q-Learning: Off-Policy TD Control

- A simple but important breakthrough is an off-policy TD algorithm.
- The simplest way is *one-step Q-learning*:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

## Exercise

## Question 1: Value functions

Consider a world with two states $S = \{S_1, S_2\}$ and two actions $A = \{a_1, a_2\}$, where the transitions $\delta$ and reward $r$ for each state and action are as follows:

$$\begin{aligned}
\delta(S_1, a_1) &= S_1 \quad r(S_1, a_1) = 0 \\
\delta(S_1, a_2) &= S_2 \quad r(S_1, a_2) = -1 \\
\delta(S_2, a_1) &= S_2 \quad r(S_2, a_1) = +1 \\
\delta(S_2, a_2) &= S_1 \quad r(S_2, a_2) = +5
\end{aligned}$$

i. Draw a picture of this world, using circles for the states and arrows forthe transitions.

ii. Assuming a discount factor of $\gamma = 0.9$, determine:

(a) the optimal policy $\pi^* : S \to A$

(b) the state-value function $V^* : S \to R$

(c) the action-value function $Q^* : S \times A \to R$

iii. Write the Q-values in a table (a.k.a. Q-table) as follows:

| Q | $a_1$ | $a_2$ |
|---|---|---|
| $S_1$ | | |
| $S_2$ | | |

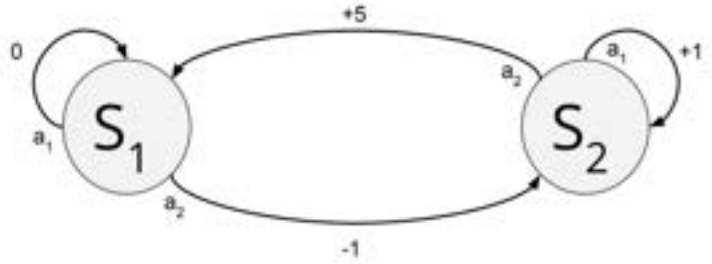



Figure 1: Draw of the world transitions.

ii. Assuming a discount factor of $\gamma = 0.9$, determine:

(a) the optimal policy $\pi^* : S \to A$

**Answer:** The optimal policy is:

$\pi^*(S_1) = a_2$

$\pi^*(S_2) = a_2$

$\delta(S_1, a_1) = S_1$ and $\delta(S_1, a_2) = S_2$

$\delta(S_2, a_1) = S_2$ and $\delta(S_2, a_2) = S_1$

And the rewards:

$r(S_2, a_1) = +1$

$r(S_2, a_2) = +5$

Choosing $a_2$ in $S_1$ takes us to $S_2$, where the long-term reward is higher because $r(S_2, a_2) = +5$.

To break it down:

$a_1$ in $S_1$ gives an immediate reward of 0.

$a_2$ in $S_1$ gives an immediate reward of -1 but takes us to $S_2$.

In $S_2$:

$a_1$ keeps us in $S_2$ with a reward of +1.

$a_2$ moves us back to $S_1$ with a reward of +5.

With the discount factor and long-term rewards in mind, $\pi^*(S_1) = a_2$ makes sense because it ever the immediate -1.

(b) the state-value function $V^* : S \to R$

Remember $Q(s,a) = r(s,a) + \gamma V^*(s')$, if we follow the optimal policy then previous equation is also equal to the optimal state-value $V^*$

**Answer:** The optimal state-value function $V^*$ is calculated as follows:

$V^*(S_1) = -1 + \gamma V^*(S_2)$
$V^*(S_2) = 5 + \gamma V^*(S_1)$
So $V^*(S_1) = -1 + 5\gamma + \gamma^2 V^*(S_1)$
$V^*(S_1) - \gamma^2 V^*(S_1) = -1 + 5\gamma$
$(1-\gamma^2)V^*(S_1) = -1 + 5\gamma$
i.e. $V^*(S_1) = (-1+5\gamma)/(1-\gamma^2) = 3.5/0.19 = 18.42$
And $V^*(S_2) = 5 + \gamma V^*(S_1)$
i.e. $V^*(S_2) = 5 + 0.9 * 3.5/0.19 = 21.58$

(c) the action-value function $Q^* : S \times A \to R$

**Answer:** As in the previous question $Q(s,a) = r(s,a) + \gamma V^*(s')$. So we only need to complete it for the other state-action pairs. The action-value function for the optimal policy is calculated as follows:

$Q(S_1, a_1) = 0 + \gamma V^*(S_1) = 16.58$
$Q(S_1, a_2) = V^*(S_1) = 18.42$
$Q(S_2, a_1) = 1 + \gamma V^*(S_2) = 20.42$
$Q(S_2, a_2) = V^*(S_2) = 21.58$

## Question 2: Temporal-difference learning

Consider the same world as the previous question. Assume the use of temporal-difference learning with the following parameters: learning rate $\alpha = 0.3$, discount factor $\gamma = 0.9$, and $\epsilon$-greedy action selection method with $\epsilon = 0.1$. After a few steps of iterating, the learning agent performs action $a_1$ from state $S_1$ with the Q-table containing the following values:

| Q | $a_1$ | $a_2$ |
|---|---|---|
| $S_1$ | 0.15 | 3.55 |
| $S_2$ | 5.72 | 9.18 |

i. How would look the Q-table after one iteration of the off-policy method Q-learning?

ii. How would look the Q-table after one iteration of the on-policy method Sarsa? Assume a random value $rnd = 0.01$.

iii. Explain how differently would the on-policy Sarsa method converge to the optimal value function in comparison to off-policy Q-learning.

**Answer:** We also need to update $Q(S_1, a_1)$, however, as Sarsa update is as $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$, the update will depend on the random selected action by $\epsilon$-greedy. Therefore, as the random number $rnd < \epsilon$, there are two possibilities.

**First option**, action $a_1$ is randomly selected:
$Q(S_1, a_1) = 0.15 + \alpha(0 + \gamma Q(S_1, a_1) - 0.15$
$Q(S_1, a_1) = 0.15 + 0.3 * (0.9 * 0.15 - 0.15) = 0.1455$

The updated table looks like this:

| Q | $a_1$ | $a_2$ |
|---|---|---|
| $S_1$ | 0.1455 | 3.55 |
| $S_2$ | 5.72 | 9.18 |

**Second option**, if action $a_2$ is randomly selected, the update is just the same as Q-learning update.
$Q(S_1, a_1) = 0.15 + \alpha(0 + \gamma Q(S_1, a_2) - 0.15$
$Q(S_1, a_1) = 0.15 + 0.3 * (0.9 * 3.55 - 0.15) = 1.0635$

The updated table looks like this:

| Q | $a_1$ | $a_2$ |
|---|---|---|
| $S_1$ | 1.0635 | 3.55 |
| $S_2$ | 5.72 | 9.18 |

## Question 3: Returns

Consider a robot learning a task with a discount factor $\gamma = 0.5$ and receiving the following reward sequence: $R_1 = -1$, $R_2 = 2$, $R_3 = 6$, $R_4 = 3$, and $R_5 = 2$, and then 0 all the time. What are $G_0, G1, ..., G_5$? Hint: Work backwards.

**Answer:** We need to consider the discount return equation:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1)$$

Then,
$G_5 = 0$ (and all the following ones)
$G_4 = 2 + 0 = 2$
$G_3 = 3 + 2 \times 0.5 = 4$
$G_2 = 6 + 3 \times 0.5 + 2 \times 0.5^2 = 6 + 1.5 + 0.5 = 8$
$G_1 = 2 + 6 \times 0.5 + 3 \times 0.5^2 + 2 \times 0.5^3 = 2 + 3 + 0.75 + 0.25 = 6$
$G_0 = -1 + 2 \times 0.5 + 6 \times 0.5^2 + 3 \times 0.5^3 + 2 \times 0.5^4 = -1 + 1 + 1.5 + 0.375 + 0.125 = 2$

Alternatively, as we worked backwards, we can also compute each return as $G_t = R_{t+1} + \gamma G_{t+1}$:
$G_5 = 0$

P(U) = 0.401054937, P(D) = 0.351109598, P(L) = 0.175346688, P(R) = 0.072488777

Accumulated P = (0.401054937, 0.752164535, 0.927511223, 1)

# 5 Lecture 5 - Optimisation

- Most common times for algorithms:
- O(1) < O(log n) < O(n) < O(n log n) < $O(n^2)$ < $O(n^3)$ < $O(2^n)$

## Simulated annealing

```python
def simulatedAnnealing(initialT, alpha, nIter, finalT):
    temp = initialT
    currentSolution = np.random.randn() * 10
    bestSolution = currentSolution
    while temp >= finalT:
        for i in range(nIter):
            neighbourSolution = currentSolution + np.random.randn()
            delta = fitness(neighbourSolution) - fitness(currentSolution)
            metropolis = np.exp(-delta / temp)
            if np.random.rand() < metropolis or delta < 0:
                currentSolution = neighbourSolution
                if fitness(currentSolution) < fitness(bestSolution):
                    bestSolution = currentSolution

        temp = alpha * temp
    return bestSolution
```

**Algorithm 1** Simulated annealing optimisation method.

**Require:** Input$(T_0, \alpha, N, T_f)$

$T \leftarrow T_0$
$S_{act} \leftarrow$ generate initial solution
**while** $T \geq T_f$ **do**
  **for** cont $\leftarrow 1$ TO $N(T)$ **do**
    $S_{cond} \leftarrow$ Neighbour solution [from $(S_{act})$]
    $\delta \leftarrow f(S_{cond}) - f(S_{act})$
    **if** rand(0,1) $< e^{-\delta/T}$ **or** $\delta < 0$ **then**
      $S_{act} \leftarrow S_{cond}$
    **end if**
  **end for**
  $T \leftarrow \alpha(T)$
**end while**
**return** Best $S_{act}$ visited

## Tabu search

**Algorithm 2** Tabu search optimisation method.

$s_0 \leftarrow$ generate initial solution
$s_{best} \leftarrow s_0$
tabuList $\leftarrow \{s_0\}$
**repeat**
  $\{s_1, s_2, ..., s_n\} \leftarrow$ generate neighbourhood from $(s_0)$
  $s_{candidate} \leftarrow s_1$
  **for** i $\leftarrow 2$ TO $n$ **do**
    $\delta \leftarrow f(s_i) - f(s_{candidate})$
    **if** $s_i$ is not in tabuList **and** $\delta < 0$ **then**
      $s_{candidate} \leftarrow s_i$
    **end if**
  **end for**
  $s_{best} \leftarrow s_{candidate}$
  Add $s_{candidate}$ to tabuList
**until** a termination criterion is satisfied
**return** $s_{best}$

```python
def tabuSearch(neighbourhoodSize, criteria):
    initialSolution = np.random.randn() * 10
    bestSolution = initialSolution
    tabuList = np.array([])
    tabuList = np.append(tabuList, initialSolution)
    while True:
        neighbours = np.zeros(neighbourhoodSize)
        for j in range(neighbourhoodSize):
            neighbours[j] = bestSolution + np.random.randn()
        candidateSolution = neighbours[0]
        for i in range(1, neighbourhoodSize):
            delta = fitness(neighbours[i]) - fitness(candidateSolution)
            if (neighbours[i] not in tabuList) and (delta < 0):
                candidateSolution = neighbours[i]
        bestSolution = candidateSolution
        tabuList = np.append(tabuList, candidateSolution)
        if fitness(bestSolution) < criteria:
            break
    return bestSolution
```

## search space

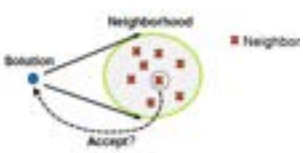
- **Feasible solution:** is in the feasible region of the problem.
- For instance, in linear programming, an optimisation problem can be represented as:

max f(x,y) = 3X + 8Y

subject to $2X + 4Y \leq 1600$ (C1)
$6X + 2Y \leq 1700$ (C2)
$Y \leq 350$ (C3)
$X \geq 0$
$Y \geq 0$

- Optimal solution in c = (100,350) → f(c) = 3100

- Neighbourhood search procedures:
  - Transformations or **movements** from the current solution.
    - Generate an **initial solution**.
    - **Iteratively** modify it until a stopping criterion.
    - Solutions are **evaluated** while traversing.
  - Possible movements create a **neighbourhood**.
  - **Feasible movements** are those that provide a feasible solution.



Constraints:

- Can be **strong** (must be satisfied) or **weak** (recommended to be satisfied).
- **Example:** In course scheduling, a strong constraint is that classes should not overlap, while a weak constraint is that there should be no classes after 4pm.

Restricted exploration of the feasible region within the search space:

- Advantages: Infeasible solutions are not evaluated. The algorithms ensure obtaining a feasible solution.
- Disadvantages: The search can be inefficient if restricted only to the feasible region. Optimal solutions may be located near the boundary and difficult to reach.

Complete exploration of the solution space:

- Advantages: The exploration of the search space is more effective.
- Disadvantages: Time is spent evaluating infeasible solutions. There is a possibility of returning an infeasible solution as the final output of the algorithm.

Three strategies for **restricted exploration** of the feasible region within the search space:

- Rejection strategies: Any infeasible solution generated during the search is directly ignored.
- Repair strategies: A repair operator is applied to each infeasible solution generated to transform it into a feasible solution. This strategy is often based on heuristics.
- Preservation strategies: Both the representation scheme and the operators are specifically designed for the problem in a way that ensures the feasibility of generated solutions. It requires more design effort and are problem-specific.

Complete exploration of the solution space:

The most common scheme for complete exploration of the solution space is penalty-based strategies:

- A **penalty function** is added to the original unconstrained objective function:

$$Min\ f'(x) = f(x) + w\,P(x)$$

- where P(x) is a penalty function and w is a weighting coefficient (intensify/diversify).
- P(x) takes a value of 0 when the solution x is feasible. Otherwise, the greater the degree of constraint violation, the larger the value of P.

## GA search

### Genetic Algorithms

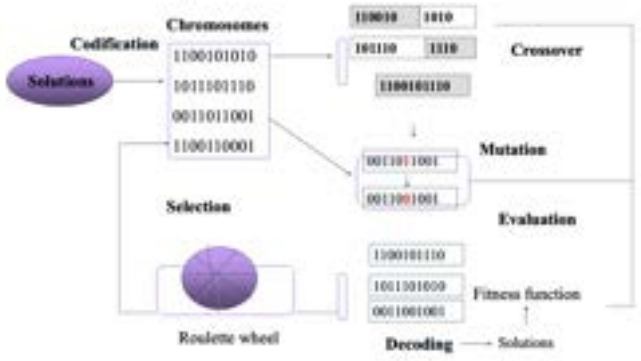
- General structure:
- Chromosome commonly represented as strings of bits or binary representation.
- Parameters include population size and probability of applying the genetic operators

**Algorithm 3** Genetic algorithm optimization method.

1: $t \leftarrow 0$
2: Initialize $P(t)$ ▶ initial population
3: Evaluate $P(t)$
4: **repeat**
5: Generate offspring $C(t)$ from $P(t)$ ▶ using crossover and mutation
6: Evaluate $C(t)$
7: Select $P(t+1)$ from $P(t) \cup C(t)$
8: $t \leftarrow t+1$
9: **until** a termination criterion is satisfied
10: **return** Best individual found from $P$

### Genetic Algorithms

- General structure example:



# 6 Lecture 7 - Computer vision

## Introduction

The original image is “thresholded”, i.e:
new[x,y] = (old[x,y] ¿ threshold)

- Every pixel brighter than a certain threshold is given a value of 1 otherwise it is zero.
- Easy to process and powerful enough to use in some industrial applications, e.g., picking parts from an assembly line.

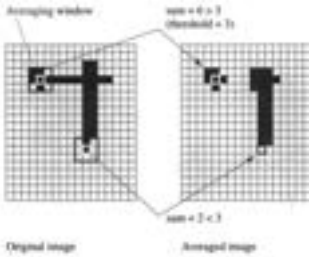
## Image processing: Averaging

It can use a threshold.

Larger rectangles achieve more smoothing.

Broad lines are thickened and thin lines eliminated

In the example, ε = 3, i.e., 0 if sum ≤ 3, 1 otherwise.



- Replace middle value in each window by average of all the values in the window.

| | | | |
|---|---|---|---|
| 9 | 9 | 9 | 3 |
| 9 | 9 | 3 | 3 |
| 9 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 |

| | | | |
|---|---|---|---|
| 9 | 9 | 9 | 3 |
| 9 | 7 | 5 | 3 |
| 9 | 5 | 4 | 3 |
| 3 | 3 | 3 | 3 |

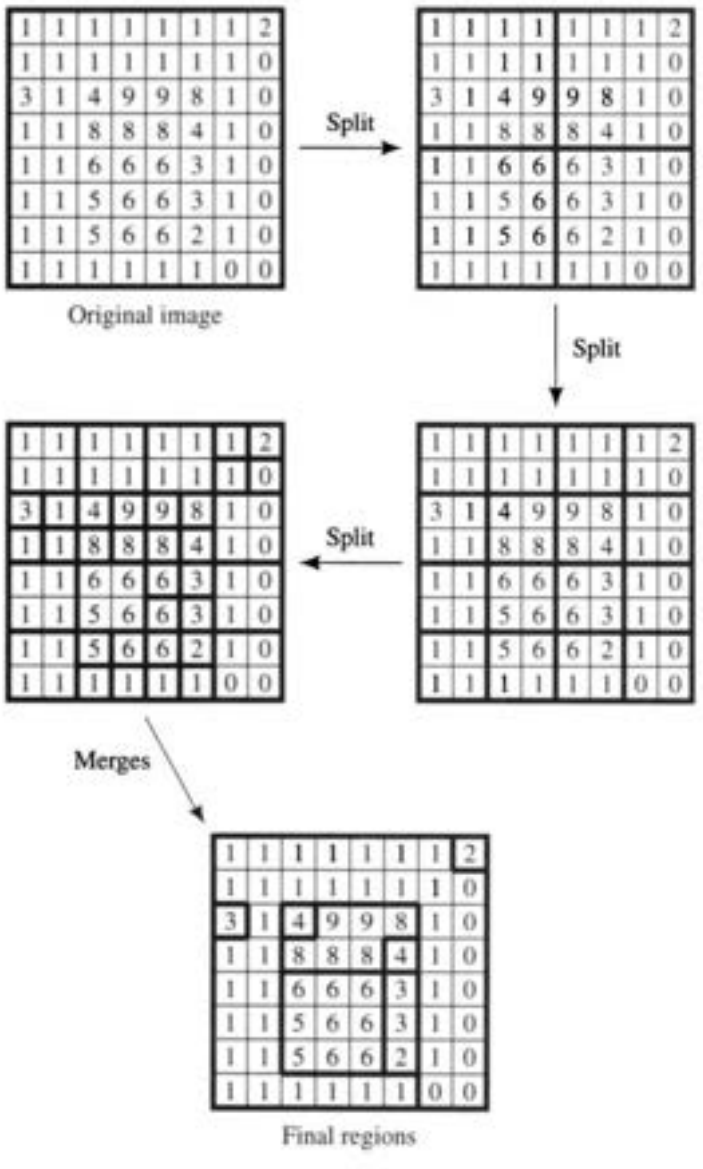
## Image processing: Region finding

To find regions in which a property does not change abruptly.

A region is homogeneous. Intensity difference no more than some ε threshold

Split-and-merge method. $2^n$ x $2^n$ array of pixels.

- Each no homogeneous region is split in four.
- Splits continues until no more splits need to be made.
- Adjacent regions are merged if homogeneous.

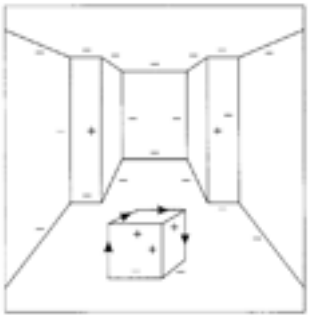


## Interpreting lines and curves in the image

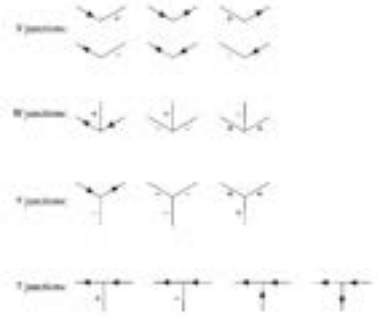
Scene with bounding walls, floor, ceiling, a cube on the floor.

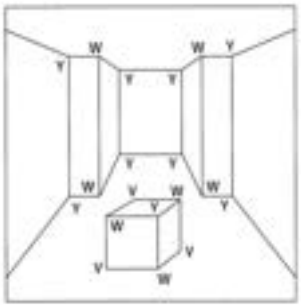
Only three possible intersections:

- Occlude: 2 planes, one occluding (→).
- Blade: both visible forming a convex edge (+).
- Fold: both visible forming a concave edge (-).



Labelling types of junctions: V, W, Y, T assigning +, -, →






Note: vat ben phai mui ten chan vat ben trai mui ten.

## exercise

1. Consider the binary image with dimension 7x16 shown in Figure 1. Use the averaging method with a threshold $\epsilon = 3$ and a 3x3 sliding windows. Show the resulting image.
Answer: White boxes represent a value 0, black boxes represent a value We need to slide a window through the image and replace the middle pixel with a 0 or a 1 depending on if the number of pixels with value 1 are greater or not than the threshold $\epsilon = 3$. Dem cac pixel co gia tri = 1 trong slide window, neu sum ¡= 3 thi center →0, otherwise 1.

# 7 Lecture 8 - NLP

## BPE

### BPE token learner

| corpus | vocabulary |
|---|---|
| 5 l o w _ | _, d, e, i, l, n, o, r, s, t, w |
| 2 l o w e s t _ | |
| 6 n e w e r _ | |
| 3 w i d e r _ | |
| 2 n e w _ | |

Merge e r to er

| corpus | vocabulary |
|---|---|
| 5 l o w _ | _, d, e, i, l, n, o, r, s, t, w, er |
| 2 l o w e s t _ | |
| 6 n e w er _ | |
| 3 w i d er _ | |
| 2 n e w _ | |

| corpus | vocabulary |
|---|---|
| 5 l o w _ | _, d, e, i, l, n, o, r, s, t, w, er |
| 2 l o w e s t _ | |
| 6 n e w er _ | |
| 3 w i d er _ | |
| 2 n e w _ | |

Merge er _ to er_

| corpus | vocabulary |
|---|---|
| 5 l o w _ | _, d, e, i, l, n, o, r, s, t, w, er, er_ |
| 2 l o w e s t _ | |
| 6 n e w er_ | |
| 3 w i d er_ | |
| 2 n e w _ | |

The next merges are:

| Merge | Current Vocabulary |
|---|---|
| (ne, w) | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new |
| (l, o) | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo |
| (lo, w) | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low |
| (new, er_) | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_ |
| (low, _) | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_ |

## Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| n | 9 | ↓8 | ↙←↓9 | ↙←↓10 | ↙←↓11 | ↙←↓12 | ↓11 | ↓10 | ↓9 | ↙8 |
| o | 8 | ↓7 | ↙←↓8 | ↙←↓9 | ↙←↓10 | ↙←↓11 | ↓10 | ↓9 | ↙8 | ←9 |
| i | 7 | ↓6 | ↙←↓7 | ↙←↓8 | ↙←↓9 | ↙←↓10 | ↓9 | ↙8 | ←9 | ←10 |
| t | 6 | ↓5 | ↙←↓6 | ↙←↓7 | ↙←↓8 | ↙←↓9 | ↙8 | ←9 | ←10 | ←↓11 |
| n | 5 | ↓4 | ↙←↓5 | ↙←↓6 | ↙←↓7 | ↙←8 | ↙←↓9 | ↙←↓10 | ↙←↓11 | ↙↓10 |
| e | 4 | ↙3 | ←4 | ↙←5 | ←6 | ←7 | ←↓8 | ↙←↓9 | ↙←↓10 | ↓9 |
| t | 3 | ↙←↓4 | ↙←5 | ↙←↓6 | ↙←↓7 | ↙←↓8 | ↙7 | ←↓8 | ↙←↓9 | ↓8 |
| n | 2 | ↙←3 | ↙←↓4 | ↙←↓5 | ↙←↓6 | ↙←↓7 | ↙←↓8 | ↓7 | ↙←↓8 | ↙7 |
| i | 1 | ↙←↓2 | ↙←↓3 | ↙←↓4 | ↙←↓5 | ↙←↓6 | ↙←↓7 | ↙6 | ←7 | ←8 |
| # | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | # | e | x | e | c | u | t | i | o | n |

## Estimating bigram probabilities: An example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\text{I} \mid \text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam} \mid \text{<s>}) = \frac{1}{3} = .33$ $P(\text{am} \mid \text{I}) = \frac{2}{3} = .67$

$P(\text{</s>} \mid \text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam} \mid \text{am}) = \frac{1}{2} = .5$ $P(\text{do} \mid \text{I}) = \frac{1}{3} = .33$

# 8 Lecture 9 - Bayes

## Conditional probability

### Conditional Probability

- Probability of one (or more) event given the occurrence of another event, denoted P(A | B).
- Conditional probabilities are key for reasoning as they accumulate evidence.
- P(A | B) = 1 is equivalent to B => A in propositional logic. Thus, P(A | B) = 0.9 is B => A with 90% certainty.
- The conditional probability is defined as:

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

- Joint probability is symmetrical: P(A, B) = P(B, A).
  - However, conditional probability is not: P(A | B) != P(B | A)
- Chain Rule: P(A,B,C,D) = P(A|B,C,D)P(B|C,D)P(C|D)P(D)

### Bayes' Rule

- Bayes theorem:

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

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

- P(A|B) is referred to as the posterior probability
- P(A) is referred to as the prior probability.
- P(B|A) is referred to as the likelihood.
- P(B) is referred to as the evidence.

### Bayes' Rule

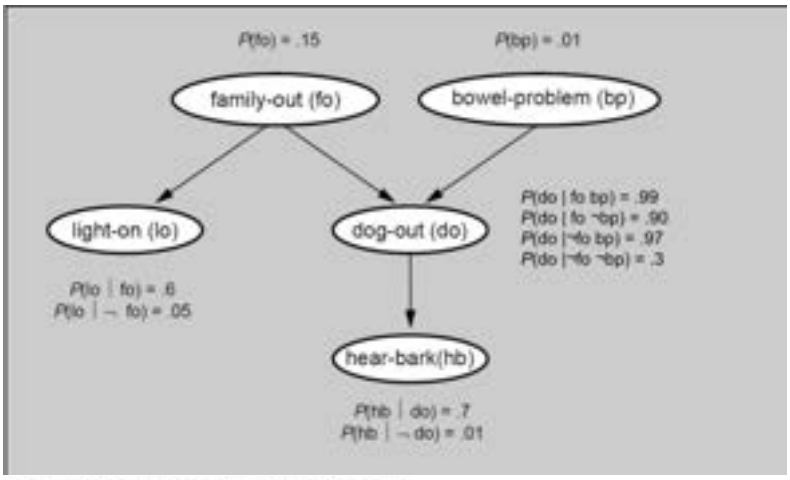
Base rates of women having breast cancer and having no breast cancer are 0.02% and 99.98% respectively. The true positive rate or sensitivity P(positive mammography | breast cancer) = 85% and the true negative or specificity P(negative mammography | ~breast cancer) = 95%. Compute P(C | M).

P(C | M) = P(M | C) * P(C) / P(M)
P(C | M) = 0.85 * 0.0002 / P(M)

From Bayes' rule: P(B) = P(B|A) * P(A) + P(B|~A) * P(~A)
P(M) = P(M|C) * P(C) + P(M|~C) * P(~C)
P(M) = 0.85 * 0.0002 + 0.05 * 0.9998 = 0.05016

P(C | M) = 0.85 * 0.0002 / 0.05016 = 0.00339 or 0.34%

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




P(BP,~FO,DO,~LO,HB) = P(HB,~LO,DO,~FO,BP)
= P(HB | ~LO,DO,~FO,BP) * P(~LO,DO,~FO,BP) by Product Rule
= P(HB|DO) * P(~LO,DO,~FO,BP) by Conditional Independence of HB and LO,FO, and BP given DO
= P(HB|DO) P(~LO | DO,~FO,BP) P(DO,~FO,BP) by Product Rule
= P(HB|DO) P(~LO|~FO) P(DO,~FO,BP) by Conditional Independence of LO and DO, and LO and BP, given FO
= P(HB|DO) P(~LO|~FO) P(DO | ~FO,BP) P(~FO,BP) by Product Rule
= P(HB|DO) P(~LO|~FO) P(DO|~FO,BP) P(~FO | BP) P(BP) by Product Rule
= P(HB|DO) P(~LO|~FO) P(DO|~FO,BP) P(~FO) P(BP) by Independence of FO and BP
= (.7)(1 - .05)(.97)(1 - .15)(.01) = 0.005483

## Fuzzy

Example in tutorial:
T=10 →low, S=50 →low or med
Apply rule, S low + T low →Inject high

$$\frac{50-25}{60-25} = \frac{f_S(x)-1}{0-1} => f_S(x) = 0.2857 f_T(x) = 1 f_{AandB}(x) = min($$