INTRO TO A.I.

CS188

NOTE 1

→ Agents:

↳ **rational agent**, an entity that has goals or preference and tries to perform a series of actions that yield the best/optimal expected outcome given these goals.

↳ it exists an environment, which is specific to given instantiation

↳ uses sensors to interact with the environment

↳ **Reflex agent**: Doesn't think about the consequences of its action, but selects actions solely based on the current state of the world.

↳ These agents are outperformed by planning agent

↳ **Planning agent**: Maintain a model of the world and use this model to simulate performing various actions. Then it can determine hypothesized consequences of the actions and then select the best one.

→ Define the task environment : **PEAS**

- **Environment** — summarize where the agent acts & what affects the agent
- **Sensors** — methods with which the agent acts on the env and recieves info from it.
- **Performance Measure** — what utility the agent tries to inc
- **Actuators** — methods with which the agent acts on the env and recieves info from it.

→ Types of Environment:-

- **Partially obs env**: Not have full info about the state & agent must have internal estimate of stat possible outcomes with diff pro
- **Stochastic env**: Uncertainty in transition model, i.e., taking an action in specific state may have mul
- **multi-agent env**: Acts in env along with other agents. Agents might need to randomize actions
- **Static env**: If env does not change as the agent acts on it
- **Known phys env**: Transition model is unknown to agent & can use that when path planning

bc - backward cost
fc - forward cost

- Dominance: If heuristic a is dominant over heuristic b then the estimated goal dist for a is > estimated goal dist for b. $h_a(n) \geq h_b(n)$
- If heuristic is consistent ⟶ it is admissible not the other way around.

NOTE 2/3

→ A search problem consists of:-

- State space: Set of all possible states that are possible in ur given world
- Actions: actions available in the state
- Transition model: outputs the next state when a specific action is taken at current state
- Action cost: incurred when moving from one state to another for the action
- Start state: State in which an agent exists initially
- Goal test: Whether it is at goal state or not.

→ World state: Contains all info about a given state

Search state: Contains only the info about the world that's necessary for planning

→ State Space graph: Constructed with states representing nodes, with directed edges existing from a state to its children. Edges represents actions and any associated weights represent cost of action.

each state is represented exactly once no need to represent multiple times.

→ Search Trees: No restriction on the no. of times a state can appear.

→ Uninformed Search:

Stack - (LIFO)
- DFS: deepest node from start for expansion - Not Complete (because if it has cycle special case if all weights equal) | Not optimal | T: $O(b^m)$ | S: $O(bm)$

Queue - (FIFO)
- BFS: shallow node from start for expansion - Complete | Not optimal | T: $O(b^s)$ | S: $O(b^s)$
- UCS: Lowest cost node for expansion - Complete | Optimal if (+ve) | T: $O(b^{C^*/\varepsilon})$ | S: $O(b^{C^*/\varepsilon})$

only bc

→ Informed Search:

Similar to ucs with a PQ
- Greedy Search: Selects lowest heuristic value for exp - Not guaranteed to be complete or optimal

only fc
- A* Search: Selects node with lowest estimated total cost - Complete & optimal

bc + fc → Heuristics:-

- Driving force that allow estimation of dist to goal states
- Want heuristic function to be lower bound on the remaining dist to goal.
- Common heuristic: Manhattan dist: $|x_1 - x_2| + |y_1 - y_2|$

g(n): bc (UCS)
h(n): fc (Greedy)
f(n) = g(n) + h(n)

① Admissibility: $\forall n,\ 0 \leq h(n) \leq h^*(n)$ (true optimal fc to reach goal for a given node n.)

② Consistency: $\forall A, C \quad h(A) - h(C) \leq cost(A, C)$

→ enforce not only that a heuristic underestimates the total dist to goal from any given node but also the cost of each edge in the graph.

NOTE 4

state space are sets of "complete" solutions

→ Local Search: Allow us to find goal states without worrying about the path to get there

Greediness tends to be stuck in local maxima

- **Hill-Climbing Search**: Moves from the current state towards the neighboring state that inc the obj value the most. Incomplete
- **Simulated Annealing Search**: Combine random walk + hill-climbing. Algo chooses a random move at each timestamp, if move leads to higher obj value, it is always accepted. If it leads to smaller objective value, then move is accepted with some prob. Prob determined by temp parameter which high at start and dec gradually.

Another variant of hill-climbing

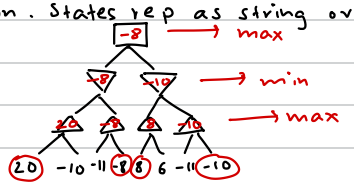
- **Local Beam Search**: Diff is that local beam keep track of K states at each iteration. Algo starts with random initialization of K states and at each iteration it takes on K new states as in hill-climbing. Algo selects the K-best succ states from complete list of succ states from all the threads

susceptible to be stuck in flat region

- **Genetic Algorithm**: Variant of local beam search. Begin with K randomly initialized states called population. States rep as string over a finite alphabet.

takes inspiration from evolution

eg:



NOTE 5

→ **Minimax**:-

∀ agent-controlled states, $V(s) = \max V(s') \quad s' \in successors(s)$

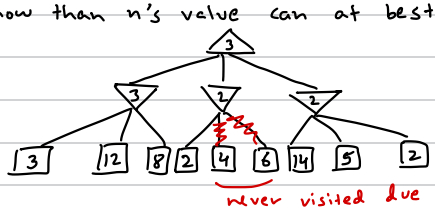
∀ opponent controlled states, $V(s) = \min V(s') \quad s' \in successors(s)$

∀ terminal states, $V(s) = Known$

→ **Alpha-Beta Pruning**:-

If u are trying to determine the value of a node n by looking at its successors, stop looking as soon as you know than n's value can at best equal the optimal value of n's parent.

Eg:



→ Evaluation function: Common design is a linear function

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

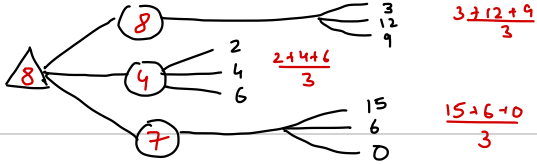
↳ Eg of non-linear is neural network in RL

f_i: feature extracted from input state

w_i: corresponding weight

NOTE 6

eg:



→ Expectimax:-

∀ agent-controlled states $V(s) = \max V(s')$, $s' \in$ successors (s)

∀ chance states $V(s) = \sum P(s'|s)\, V(s')$, $s' \in$ successors (s)

∀ terminal states $V(s) =$ Known

→ Monte Carlo Tree Search:-

- Evaluation by rollouts : From state s play many times using a policy and count win/loss
- Selective search : Explore parts of the tree, without constraints on the horizon, that will improve decision at the root.
- UCB Algorithm : $UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$

  $U(n)$: total no. of wins for Player(parent(n))

  $N(n)$: total no. of rollouts from node n

NOTE 7

→ Language of Logic:-

- NOT ($\neg$) : Opposite of Logic value
- AND ($\wedge$) : If all T then T
- OR ($\vee$) : If atleast one T the T
- Implication ($\rightarrow$) : T $\rightarrow$ F $\equiv$ F else T
- Bi cond ($\leftrightarrow$) : $(A \rightarrow B) \wedge (B \rightarrow A)$

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativity of $\wedge$

$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ commutativity of $\vee$

$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativity of $\wedge$

$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ associativity of $\vee$

$\neg(\neg\alpha) \equiv \alpha$ double-negation elimination

$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ contraposition

$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$ implication elimination

$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ biconditional elimination

$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ De Morgan

$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ De Morgan

$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributivity of $\wedge$ over $\vee$

$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributivity of $\vee$ over $\wedge$

NOTE 8

→ Propositional Logic:-

- Written in sentences composed of proposition symbols, possibly joined by logical connectives. A proposition symbols is generally rep as single uppercase letter.
- A model is an assignment of true or false to all the proposition symbols which we might think of as a "possible world."

→ In general N symbols, there are 2^N possible models

→ · Valid Sentence: If it is T in all of these models

· Satisfiable : If there is atleast one model in which it is T

· Unsatisfiable: If it is not true in any models

→ CNF: Only has $\vee$, $\wedge$, and $\sim$. (only on literals)

- $A \rightarrow B \equiv \sim A \vee B$
- $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A) \equiv (\sim A \vee B) \wedge (\sim B \vee A)$

→ **Propositional Logical Inference**

$A \models B$

- A entails another sentence B if in all models that A is true, B is as well.
  - → models of A are a subset of the models of B.
- To show inference we do $KB \models q$

→ **Thm to show entailment:-**

- $A \models B$ iff $A \rightarrow B$ is valid — direct proof
- $A \models B$ iff $A \wedge \neg B$ is unsatisfied — proof by contraction

→ **Model Checking:-**

One simple algo for checking whether $KB \models q$ is to enumerate all possible models and check if in if in all KB is T, q is also T.

→ **DPLL Algorithm:-** depth-search, backtracking over possible models with 3 tricks to reduce excessive backtracking

Track
- model
- symbols
- clauses

- **Early Termination**: Clause T if any symbol T. Sentence F if any single clause F
- **Pure Symbol Heuristic**: Symbol only in either +ve or -ve form.
- **Unit Clause Heuristic**: Clause with just one literal or disjunction with 1 literal & many Fs.

A:

eg: $(\neg N \vee \neg S) \wedge (M \vee Q \vee N) \wedge (L \vee \neg M) \wedge (L \vee \neg Q) \wedge (\neg L \vee \neg P) \wedge (R \vee P \vee N) \wedge (\neg R \vee \neg L) \wedge (S)$

① M: { }
S: {L, M, N, P, Q, R, S}
C: A

② M: {S:T}
S: {L, M, N, P, Q, R}
C: $(\neg N) \ldots (\neg R \vee \neg L)$

③ M: {S:T, N:F}
S: {L, M, P, Q, R}
C: $(M \vee Q) \wedge \ldots (R \vee P) \wedge (\neg R \vee \neg L)$

④ M: {S:T, N:F, M:T}
S: {L, P, Q, R}
C: $L \wedge \ldots (R \vee P) \wedge (\neg R \vee \neg L)$

⑤ M: {S:T, N:F, M:T, Q:F}
S: {L, P, R}
C:

⑥ M: {S:T, N:F, M:T, Q:F, L:T}
S: {P, R}
C: $\neg P \wedge (R \vee P) \wedge \neg R$

⑦ M: {S:T, N:F, M:F, Q:T, L:T, P:F}
S: {R}
C: $R \wedge \neg R$

→ **Theorem Proving:-**

Alt approach is to apply rules of inference to KB to prove $KB \models q$.

Prove entailment using 3 rules of inference:-

- If our KB contains A & $A \rightarrow B$ we can infer B. (Modus Ponens)
- If our KB contains $A \wedge B$ we can infer A. We can also infer B. (&-Elim^n)
- If our KB contains A & B we can infer $A \wedge B$. (Resolution)

→ **Forward Chaining:** Iterate through every implications statement in which the premise is known to be T, adding the conclusion to list of known facts.

eg: ① $A \rightarrow B$ ② $A \rightarrow C$ ③ $B \wedge C \rightarrow D$ ④ $D \wedge E \rightarrow Q$ ⑤ $A \wedge D \rightarrow Q$ ⑥ A

1. count: [1, 1, 2, 2, 2, 0]
   inferred: [A .... E, Q: F]
   agenda [A]

2. count: [0, 0, 2, 2, 1, 0]
   I: {A: T, B ... E, Q: F}
   A: [B, C]

3. C: [0, 0, 1, 2, 1, 0]
   I: {A, B: T, C ... E, Q: F}
   A: [C]

4. C: [0, 0, 0, 2, 1, 0]
   I: {A, B, C: T, D, E, Q: F}
   A: [D]

5. C: [0, 0, 0, 1, 0, 0]
   I: {A.. D: T, E, Q: F}
   A: {Q}

NOTE 9

- **Term:** Logical expression that refer to an object.
- **Atomic** sentence: Description bet objects, T if relationship holds.
- $\forall$: for all $\exists$: There exists
- $\forall x \neg P \equiv \neg \exists x P$, $\forall x P \equiv \neg \exists x \neg P$
- **Successor - state axion:** Hot^t = the air is hot at time t

NOTE 10 →

- $P(A, B) = P(A|B)P(B) = P(B|A)P(A)$

  $P(A_1, A_2 \ldots A_k) = P(A_1) P(A_2|A_1) \ldots P(A_k | A_1, \ldots A_{k-1})$
- **Marginal distribution:** $P(A) = \sum_b \sum_c P(A, B=b, C=c)$
- Sometime all prob $\neq 1$, so we would **normalize** it, to do so take sum of all entries in dist and divide each entry by sum.
- **Conditional prob:** $P(A|B) = P(A,B)/P(B)$ | $P(A|B) = P(B|A)P(A)/P(B)$
- **Mutually Independent:** $A \perp\!\!\!\perp B \equiv B \perp\!\!\!\perp A \rightarrow P(A, B) = P(A)P(B)$ | $P(A|B) = P(A)$
- **Conditionally Independent:** If we say A & B are conditionally independent given C then $P(A, B|C) = P(A|C)P(B|C)$. Meaning if we have knowledge about the value of C, then B and A does not affect each other.

  $P(A|B, C) = P(A|C)$ and $P(B|A, C) = P(B|C)$

→ **Inference by enumeration:-** $P(Q_1 \dots Q_m \mid e_1 \dots e_n)$

- **Query variables (Q_i):** Unknown and on left side of (|)
- **Evidence variables (e_i):** Known and observed var on the right side of (|)
- **Hidden variables:** Values present in overall joint dist not in the desired dist.

⊙ **Algorithm:-**

- Collect all the rows consistent with obs e_i
- Sum out (marginalize) all the hidden variables
- Normalize the table so that it is a prob distribution.

NOTE 11 → **Bayesian Network Representation:-**

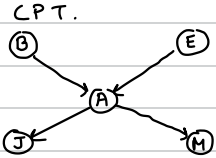
Rather than storing info in giant table, prob distributed across a number of smaller conditional prob table along with a DAG, which captures a relationship bet variables.

Formally it consists of

- DAG of nodes, one per variable X.
- Cond^n dist for each node $P(X \mid A_1 \dots A_n)$ where A_i is ith parent of X stored as CPT.

$$P(x_1, x_2 \dots x_n) = \prod_{i=1}^{n} P(x_i \mid \text{parents}(X_i))$$

eg:



$$P(-b, -e, +a, +j, -m) = P(-b)P(-e)P(+a \mid -b, -e)P(+j \mid +a)P(-m \mid +a)$$

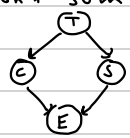
→ **Structure of Bayes Net:-**

→ Each node is conditionally independent of all its ancestor nodes (non-descendants in the graph, given the parents.

→ Each node is conditionally independent of all other variables given its Markov blanket

NOTE 12 → Variable Elimination:-

- Alt approach is to eliminate hidden var one-by-one. To eliminate a var X, we:-
  ① Join (multiply together) all factors involving X.
  ② Sum out X
- Factor: An unnormalized probability.
  └ At all points during var elimination, each factor will be proportional to the prob it corresponds to but the underlying dist for each factor won't sum to 1

eg: 

We need to find: P(T | +e)
└ to do so we eliminate C and S

Approach:-

- Join all factors involving C: $f_1(C, T, +e, S) = P(C|T)\,P(+e|C,S) = P(C, +e|T, S)$
- Summing out C, leaving new factor $f_2(+e, T, S) = P(+e|T, S)$
- Join all factors involving S: $f_3(+e, T, S) = P(S|T) \cdot f_2(+e, T, S) = P(+e, S|T)$
- Sum out S yielding $f_4(+e, T) = P(+e|T)$
- Join remaining factors $f_5(+e, T) = f_4(+e, T) \cdot P(T)$
  ↳ Compute $P(T|+e)$ by normalizing
  ↳ $\alpha P(T) \sum_S P(S|T) \sum_C P(C|T)\,P(+e|C,S)$

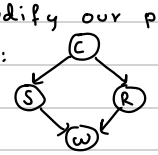
NOTE 13:-

P(Q) → Prior Sampling:-
very simple/ very inefficient

- Explore and understand the behavior of parameters before obs actual data.
- Involves generating values from the prior dist of the parameters

P(Q|e) bit more efficient → Rejection Sampling:-

- Modify our procedure to early reject any sample inconsistent with evidence.

eg: 

want $P(C|r, w)$ ← reject sample with $\neg r$ and $\neg w$

$C, \neg S, r, w$
~~$C, S, \neg r$~~
~~$\neg C, S, r, \neg w$~~
~~$C, \neg S, \neg r$~~
$\neg C, \neg S, r, w$

P(Q|e) evidence only effects downstream RVs →

Likelihood Weighting:-

- ensures that we never generate a bad sample
- We manually set all variables equal to the evidence in our query.
- Eg: If we want $P(C|\neg t)$, we simply declare that t is false
- Algo: ① Input evidence $e_1 \dots e_k$
  ② $w = 1.0$
  ③ for $i = 1 \dots n$
    if X_i is evidence var
      X_i = obs value$_i$ for X_i
      Set $w = w * P(x_i | \text{Parents}(X_i))$
    else
      Sample x_i from $P(X_i | \text{parents}(X_i))$
  return $(x_1 \dots x_n), w$
- Value of down stream variables affected by upstream evidence.

most efficient evidence effects both downstream & upstream →

Gibbs Sampling:-

- First, set all variables to some totally random value. We then repeatedly pick one variable at a time, clear its value, and resample it given the values currently assigned to all other variables.

eg: $\omega_0 \rightarrow \omega_1 \rightarrow \omega_2 \rightarrow \omega_3 \rightarrow \dots$

NOTE 14 →

Markov's Model:-

- Analogous to a chain-like, infinite-length Bayes's net.
- has 2 things, first being initial dist (t=0) and transition model $t=i$
- Memoryless property: independent of the weather at all other timestamps besides $t=i$
  ↳ $P(\omega_0, \omega_1, \omega_2) = P(\omega_0) P(\omega_1|\omega_0) P(\omega_2|\omega_0, \omega_1) = P(\omega_0) P(\omega_1|\omega_0) P(\omega_2|\omega_1)$
  ↳ $P(\omega_0 \dots \omega_n) = P(\omega_0) \prod_{i=0}^{n-1} P(\omega_{i+1}|\omega_i)$
- Final assumption: Transition model is stationary.

→ Mini-Forward Algorithm:- (dist of a weather on a give day t)

$$P(\omega_{i+1}) = \sum_{\omega_i} P(\omega_{i+1}|\omega_i) P(\omega_i)$$

W_0	$P(W_0)$
sun	0.8
rain	0.2

W_{i+1}	W_i	$P(W_{i+1}\|W_i)$
sun	sun	0.6
rain	sun	0.4
sun	rain	0.1
rain	rain	0.9

eg: $P(w_1 = \text{sun}) = P(w_1 = \text{sun}|w_0 = \text{sun}) P(w_0 = \text{sun}) + P(w_1 = \text{sun}|w_0 = \text{rain}) P(w_0 = \text{rain})$
$= 0.6 \times 0.8 + 0.1 \times 0.2 = 0.48 + 0.02 = 0.5$

$P(w_1 = \text{rain}) = P(w_1 = \text{rain} \mid w_0 = \text{rain})\, P(w_0 = \text{rain}) + P(w_1 = \text{rain} \mid w_0 = \text{sun})\, P(w_0 = \text{sun}) = 0.5$

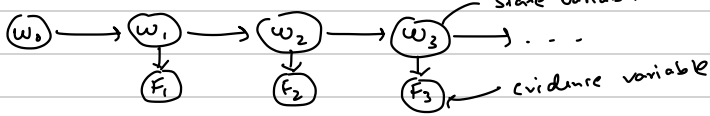
w_1	$P(w_1)$
rain	0.5
sun	0.5

→ **Stationary Distribution:-**

- One that remains the same after the passage of time, i.e., $P(w_{t+1} \mid w_t)$

NOTE 15

eg:



→ **Hidden Markov Model:-**

- Allows us to observe some evidence at each timestamp, which can potentially affect the belief distribution at each of the states.

eg:
- Speech tagging
- Robot tracking
- Machine translation
- Molecular bio

- $F_i \perp w_0 \mid w_i$
- Additional assumption: Sensor Model $P(F_i \mid w_i)$ is stationary as well
- 3 prob table: ① Initial dist $P(x_0)$ ② Transition model $P(x_t \mid x_{t-1})$ ③ Sensor Model $P(E_t \mid x_t)$
- Final point on notation → Belief dist: $B(w_i) = P(w_i \mid f_1, .., f_i)$
- $P(x_0, \ldots x_T) = P(x_0) \prod_{t-1} P(x_t \mid x_{t-1})$
- $P(x_0, E_0, x_1, \ldots x_T, E_T) = P(x_0) \prod_{t-1} P(x_t \mid x_{t-1})\, P(E_t \mid x_t)$
- Random sampling: evidence variables are not independent

- **Inference tasks**
  - Filtering $P(x_t \mid e_{1:t})$
  - Prediction $P(x_{t+k} \mid e_{1:t})$ for $K > 0$
  - Smoothing $P(x_K \mid e_{1:t})$ for $0 \le K < t$
  - Most likely explanation: $\arg\max_{x_{1:t}} P(x_{1:t} \mid e_{1:t})$

→ **Filtering Algorithm:-**

$$P(x_{t+1} \mid e_{1:t+1}) = P(x_{t+1} \mid e_{1:t}, e_{t+1}) \quad \text{(Bayes Thrm)}$$

$$= \alpha\, P(e_{t+1} \mid x_{t+1}, e_{1:t})\, P(x_{t+1} \mid e_{1:t})$$

$$= \alpha\, P(e_{t+1} \mid x_{t+1})\, P(x_{t+1} \mid e_{1:t}) \quad \text{(cond}^n \text{ prob)}$$

$$= \alpha\, P(e_{t+1} \mid x_{t+1}) \sum_{x_t} P(x_t \mid e_{1:t})\, P(x_{t+1} \mid x_t, e_{1:t})$$

$$= \alpha\, P(e_{t+1} \mid x_{t+1}) \sum_{x_t} P(x_t \mid e_{1:t})\, P(x_{t+1} \mid x_t)$$

By HMM — Pre-computed — By HMM

→ **The forward Algorithm :-**

$B'(w_i) = P(w_i \mid f_1 \dots f_{i-1})$

$B(w_i) = P(w_i \mid f_1 \dots f_i)$

$$B'(w_{i+1}) = \sum_{w_i} P(w_{i+1} \mid w_i)\, B(w_i)$$

$$B(w_{i+1}) \propto P(f_{i+1} \mid w_{i+1})\, B'(w_{i+1})$$

$$B(w_{i+1}) \propto P(f_{i+1} \mid w_{i+1}) \sum_{w_i} P(w_{i+1} \mid w_i)\, B(w_i)$$

→ Time elapse update : advance model's state by one timestamp
  ↳ $B'(w_{i+1}) = \sum_{w_i} P(w_{i+1} \mid w_i)\, B(w_i)$

→ Observation update : incorporate new evidence
  ↳ $B(w_{i+1}) \propto P(f_{i+1} \mid w_{i+1})\, B'(w_{i+1})$

eg:

W_0	$B(W_0)$
sun	0.8
rain	0.2

W_{i+1}	W_i	$P(W_{i+1} \mid W_i)$
sun	sun	0.6
rain	sun	0.4
sun	rain	0.1
rain	rain	0.9

F_i	W_i	$P(F_i \mid W_i)$
good	sun	0.8
bad	sun	0.2
good	rain	0.3
bad	rain	0.7

Q) Compute $B(w_1)$

$B'(w_1 = s) = P(w_1 = s \mid w_0 = s)\, B(w_0 = s) + P(w_1 = s \mid w_0 = r)\, B(w_0 = r)$

$= 0.6 \times 0.8 + 0.1 \times 0.2$

$= 0.5$

$B'(w_1 = r) = P(w_1 = r \mid w_0 = s)\, B(w_0 = s) + P(w_1 = r \mid w_0 = r)\, B(w_0 = r)$

$= 0.4 \times 0.8 + 0.9 \times 0.2$

$= 0.5$

w_1	$B'(w_1)$
s	0.5
r	0.5

$B(w_1 = s) \propto P(f_1 = \text{good} \mid w_1 = \text{sun})\, B'(w_1 = \text{sun}) \propto 0.8 \times 0.5 = 0.4$

$B(w_1 = r) \propto P(f_1 = \text{good} \mid w_1 = r)\, B'(w_1 = r) \propto 0.3 \times 0.5 = 0.15$

Last step is to normalize :-

$B(w_1 = s) = 0.4 / 0.55 = 8/11$

$B(w_1 = r) = 0.15 / 0.55 = 3/11$

→ **Viterbi Algorithm:-**

- Solve for $\arg\max_{x_{1:N}} P(x_{1:N} \mid e_{1:N})$ =
- Algo consists of 2 passes:-
  1. Run forward in time & computes the prob of the best path to each (state, time) tuple given the evidence observed so far.
  2. Run backward in time: first it find the terminal state that lies on the path with the highest prob, and then transverse backward through time along the path that leads into this state.

- $m_t[x_t] = \max_{x_{1:t-1}} P(x_{1:t}, e_{1:t})$

  ↳ $= \max_{x_{t-1:t-1}} P(e_t \mid x_t)\, P(x_t \mid x_{t-1})\, P(x_{1:t-1}, e_{1:t-1})$

  $= P(e_t \mid x_t) \max_{x_{t-1}} P(x_t \mid x_{t-1}) \max_{x_{1:t-2}} P(x_{1:t-1}, e_{1:t-1})$

  $= P(e_t \mid x_t) \max_{x_{t-1}} P(x_t \mid x_{t-1})\, m_{t-1}[x_{-1}]$

NOTE 16 → **Particle Filtering:-**

- Instead of storing a full probability table mapping each state to its belief probability, we'll instead store a list of n particles where each particle is in one of the d possible states in the domain of our time-dependent RV.
- eg. we have values in range [10, 20] for n = 10 for

  T = [15, 12, 12, 10, 18, 14, 12, 11, 11, 10]

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_i)$	2/10	2/10	3/10	0/10	1/10	1/10	0/10	0/10	1/10	0/10	0/10

→ **Particle Filtering Simulation:-**

- Begins with particle initialization which can be done quite flexibly - we can sample particles randomly, uniformly, or from some initial distribution.
- **Time Elapse Update:** Update the value of each particle acc^n to the transition model. For a particle in state t_i, sample the updated value from prob dist given by $P(T_{i+1} \mid t_i)$
- **Observation Update:** Use sensor model $P(F_i \mid T_i)$ to weight each particle acc^n to the prob dictated by obs evidence and particle's state.
  1. Calc weight of all particles described above.
  2. Calc the total weight for each state.
  3. If sum of all weights across all state is 0, reinitialize all particles.

④ Else normalize the dist of total weights over states and resample our list of particles from the dist

→ **Utilities:-**

- **Principle of max utility:** They must always select the action that maximizes their expected utility.
- If agent prefers recieving aprize A to B then $A \succ B$
- If agent is indiff bet recieving A or B then $A \sim B$
- A lottery is situation with diff prizes resulting with diff probs. eg: $L = [p, A; (1-p), B]$
- **5 Axioms of Rationality:-**

① **Orderability:** $(A \succ B) \vee (B \succ A) \vee (A \sim B)$

② **Transitivity:** $(A \succ B) \wedge (B \succ C) \rightarrow (A \succ C)$

③ **Continuity:** $A \succ B \succ C \rightarrow \exists p\ [p, A; (1-p), C] \sim B$

④ **Substitutability:** $A \sim B \rightarrow [p, A; (1-p), C] \sim [p, B; (1-p), C]$

⑤ **Monotonicity:** $A \succ B \rightarrow (p \geq q \leftrightarrow [p, A; (1-p), B] \succeq [q, A; (1-q), B])$

If all satisfied then max of expected utility

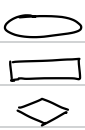
eg: $L = [0.5, \$0; 0.5, \$1000]$, 3 agents with utility func $U_1(x) = x$, $U_2(x) = \sqrt{x}$, $U_3(x) = x^2$

$U_1(L) = 0.5 \times 1000 + 0.5 \times 0 = 500$

$U_2(L) = 0.5 \times \sqrt{1000} + 0.5 \times \sqrt{0} = 15.81$

$U_3(L) = 0.5 (1000)^2 + 0.5 \times 0^2 = 500000$

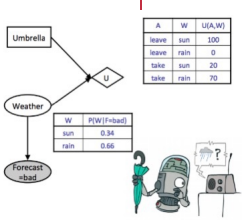
→ **Decision Network:-** on uncertainty Bayes



- Chance node: Each outcome has an associated prob, determined by running inference
- Action node: Complete control over the node. Rep choice bet any no. of actions choose fo
- Utility node: Child of chance and Action. Ans based on Chance and Action

- **Expected utility (EU):** of taking action a given evidence e with n chance nodes.
  - ↳ $EU(a|e) = \sum_{x_1, \ldots, x_n} P(x_1, \ldots, x_n | e)\, U(a, x_1, \ldots x_n)$
- **Maximize expected utility (MEU):** Action that maximizes EU
  - ↳ $MEU(E=e) = \max_a EU(A=a | E=e)$

eg:-



A	W	U(A,W)
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

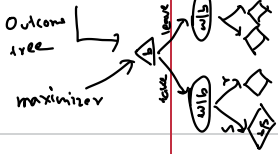
W	P(W|F=bad)
sun	0.34
rain	0.66

Find $P(W | F = bad)$:-

$EU(W = leave | F = bad) = \sum_W P(W | bad)\, U(leave, W) = 0.34 \times 100 + 0.66 \times 0 = 34$

$EU(W = take | F = bad) = \sum_W P(W | bad)\, U(take, W) = 0.34 \times 20 + 0.66 \times 70 = 53$

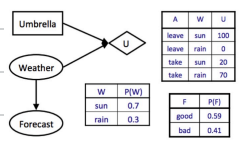
$MEU(F = bad) = \max_a EU(a | bad) = 53$



→ **Value of Perfect Information (VPI):—**

- Mathematically quantifies the amount an agent's MEU is expected to inc if it obs some new evidence.
- Currently the max utility with evidence e is $MEU(e) = \max_a \sum_s P(s|e)U(s,a)$
- If obs new evidence e', $MEU(e,e') = \max_a \sum_s P(s|e,e')U(s,a)$
- But we don't know what e' is → replace with random E'. Represent new MEU with $MEU(e,E') = \sum_{e'} P(e'|e)MEU(e,e')$
- Therefore, $VPI(E'|e) = MEU(e,E') - MEU(e)$

eg:



A	W	U
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

W	P(W)
sun	0.7
rain	0.3

F	P(F)
good	0.59
bad	0.41

$MEU(\varnothing) = \max_a MEU(a) = \max_a \sum_w P(w)U(a,w) = \max(0.7\times100 + 0.3\times0,\ 0.7\times20 + 0.3\times70)$

$= 70$

$MEU(e,f') = MEU(F) = \sum_{e'} P(e'|e)MEU(e,e') = \sum_f P(F=f)MEU(F=f)$

$= P(F=g)MEU(F=g) + P(F=b)MEU(F=b)$

$= 77.78$

$VPI(F) = 77.78 - 70 = 7.78$

→ **Properties of VPI**

① **Non negativity**: $\forall E', e\ VPI(E',e) \geq 0$

- Obsing new evidence can only help u (or be irrelevant, i.e 0 value)

② **Non additivity**: $VPI(E_j, E_k|e) \neq VPI(E_j|e) + VPI(E_k|e)$ in general

- Obsing evidence E might affect how much we care about E_k can't just add their VPI together

③ **Order-independence**: $VPI(E_j, E_k|e) = VPI(E_j|e) + VPI(E_k|e,E_j) = VPI(E_k|e) + VPI(E_j|e,E_k)$

- Order of obsing evidence doesn't matter since we only take an action after obsing new evidence

NOTE 17

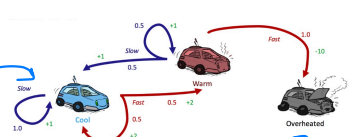
→ **Markov Decision Processes (MDP):-**

actions: {fast, slow}

① Properties:-

- Set of states S
- Set of actions A
- Start state
- One or more terminal states
- Discount factor (γ)
- transition function T(s, a, s')
- Reward function R(s, a, s')

eg:-



- **Transition Function**: $T(s,a,s')$
  - $T(cool, slow, cool) = 1$
  - $T(warm, slow, cool) = 0.5$
  - $T(warm, slow, warm) = 0.5$
  - $T(cool, fast, cool) = 0.5$
  - $T(cool, fast, warm) = 0.5$
  - $T(warm, fast, overheated) = 1$
- **Reward Function**: $R(s,a,s')$
  - $R(cool, slow, cool) = 1$
  - $R(warm, slow, cool) = 1$
  - $R(warm, slow, warm) = 1$
  - $R(cool, fast, cool) = 2$
  - $R(cool, fast, warm) = 2$
  - $R(warm, fast, overheated) = -10$

- Agent's goal is to maximize its reward across all time stamps
  $U([s_0, a_0 \ldots]) = R(s_0, a_0, s_1) + \ldots$
- Q states represented as tuple (s, a)

→ **Finite Horizons & Discounting:-**

- Time constraint on the number of timestamps for which a agent can take action and collect rewards.
- **Finite horizons:** Defines a "lifetime" for agents which gives them some n timestamps to collect reward before terminated.
- **Discounts:** Introduced to model an exponential decay in the value of rewards over time.
  - Reward $= \gamma^t R(s_t, a_t, s_{t+1})$
  - discounted utility $= U([s_0, a_0 \ldots]) = R(s_0, a_0, s_1) + \gamma^1 R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \ldots$
    - If $|\gamma| < 1$ we use sum of gp formula
      $= \sum_0^\infty \gamma^t R(s_t, a_t, s_{t+1}) = \boxed{\frac{R_{max}}{1-\gamma}}$

→ **Markovianess:-**

- Satisfy the markov or memoryless property.
- $P(S_{t+1} = s_{t+1} \mid S_t = s_t, A_t = a_t \ldots S_0 = s_0) = P(S_{t+1} = s_{t+1} \mid S_t = s_t, A_t = a_t)$
- $T(s, a, s') = P(s' \mid s, a)$

→ **Solving MDP:-**

- Solving MDP means finding at optimal policy $\pi^* : S \to A$, a func mapping each state $s \in S$ to an action $a \in A$

MEU over all possible states

→ **Bellman Equation (To solve complex MDP):-**

- $U^*(s)$ (Optimal value of a state): Expected value of the utility an optimally behaving agent that starts in s will recieve over the rest of agent's lifetime.

$$\boxed{U^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma U^*(s') \right]}$$

- $Q^*(s, a)$ (Optimal value of Q-state (s, a)): Expected value of utility agent recieves after starting in s, taking a and acting optimally.

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

$$\boxed{U^*(s) = \max_a Q^*(s, a)}$$

NOTE 18

→ **Value Iteration:-**

- DP algo that uses an iteratively longer time limit to compute time-limited values until convergence (i.e, until the U values are same for each state as they were in past iteration)
- **Algorithm:-**

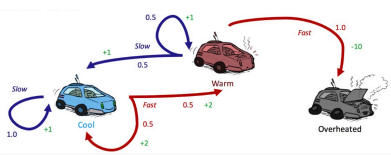
① $\forall s \in S$, initialize $U_0(s) = 0$. (means no action can be taken before termination)

② Repeat until converges:-

$$\forall s \in S,\ U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma U_k(s')\right]$$

- Convergence is when $\forall s \in S,\ U_k(s) = U_{k+1}(s) = U^*(s)$

eg:



① Initialize all $U(s) = 0$

	cool	warm	overheated
U_0	0	0	0
U_1	2	1	0
U_2	2.75	1.75	0

② $U_1(\text{cool}) = \max\{1[1+0.5\times0],\ 0.5[+2+0.5\times0]+0.5[2+0.5\times0]\} = \max\{1, 2\} = 2$

$U_1(\text{warm}) = \max\{0.5[1+0.5\times0]+0.5[1+0.5\times0],\ 1[-10+0.5\times0]\} = \max\{1, -10\} = 1$

$U_1(\text{over}) = \max\{\} = 0$

③ $U_2(\text{cool}) = \max\{1[1+0.5\times2],\ 0.5[+2+0.5\times2]+0.5[2+0.5\times1]\} = \max\{2, 2.75\} = 2.75$

$U_2(\text{warm}) = \max\{0.5[1+0.5\times2]+0.5[1+0.5\times1],\ 1[-10+0.5\times0]\} = \max\{1.75, -10\} = 1.75$

$U_2(\text{over}) = \max\{\} = 0$

→ **Policy extraction:** If u are in a state s you should take action a which yields max EU. $\pi^*(s) = \arg\max_a Q^*(s,a) = \arg\max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma U^*(s')\right]$

→ **Q-Value Iteration:-**

- DP algo that computes time-limited Q-values

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

→ **Policy Iteration:-**

- Value Iteration is too slow ← takes $O(|S|^2|A|)$ runtime
- Algo that maintains the optimality of value iteration while providing significant performance gains.

① Define an initial policy

② Repeat until convergence:-

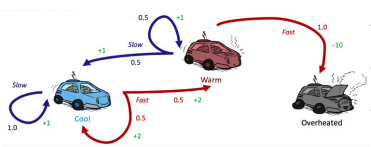
$$U^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')\left[R(s, \pi(s), s') + \gamma U^{\pi}(s')\right]$$

Policy Improvement →

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

↳ If $\pi_{i+1} = \pi_i$, the algo converged, we conclude $\pi_{i+1} = \pi_i = \pi^*$

eg:



① Initialize:

	cool	warm	overheated
π_0	S	S	—

② $U^{\pi}(cool) = 1\left[1 + 0.5\, U^{\pi_0}(cool)\right]$

$U^{\pi}(warm) = 0.5\left[1 + 0.5\, U^{\pi_0}(cool)\right] + 0.5 \cdot \left[1 + 0.5 \cdot U^{\pi_0}(warm)\right]$

Solving Linear eqn → $U^{\pi_0}(c) = 2$, $U^{\pi_0}(w) = 2$, $U^{\pi_0}(o) = 0$

③ Policy extraction:-

$\pi_1(cool) = \text{argmax}\{\text{slow}: 1[1 + 0.5 \times 2],\ \text{fast}: 0.5[2 + 0.5 \times 2] + 0.5 \times [2 + 0.5 \times 2]\}$

$= \text{argmax}\{\text{slow}: 2,\ \text{fast}: 3\} = \text{fast}$

$\pi_1(warm) = \text{argmax}\{\text{slow}: 0.5[1 + 0.5 \times 2] + 0.5[1 + 0.5 \times 2],\ \text{fast}: 1[-10 + 0.5 \times 0]\}$

$= \text{argmax}\{\text{slow}: 3,\ \text{fast}: -10\} = \text{slow}$

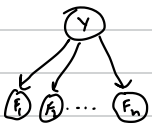
After running 2nd iteration it is still fast and slow so it has converged.

NOTE 19 → **MACHINE LEARNING:-**

- **Types of learning:-**
  - ① **Supervised**: training data has labels, eg: classification
  - ② **Unsupervised**: training data has no labels, eg: clustering
- **Data split into 3:-**
  - ① **Training set**: Used to fit the model [model mapping input to output]
  - ② **Validation set**: Used to tune hyperparameters (lr, model structure etc)
  - ③ **Test set**: Used to test the entire model

- Types of ML problem :-
  1. Classification : try to classify data into discrete classes
  2. Regression : try to estimate some numerical val from data
  3. Clustering : try to group similar data into clusters.

→ Naive Bayes :- ← with 2^{n+1} entries



- Model that can predict a label Y given features, where we assume all features are independently affected by the label.
- eg: Spam filter
  - Y is in {Spam, ham}
  - F_i in {0,1} where i is in email or not
    - $P(Y = ham \mid F = f_1, \dots F_n = f_n)$ $P(Y = spam \mid F_i = f_1 \dots F_n = f_n)$
- Generalized :-

$$\text{prediction}(f_1 \dots f_n) = \arg\max_y P(Y = y \mid F_1 = f_1 \dots F_N = f_n)$$
$$= \arg\max_y P(Y = y, F_1 = f_1 \dots F_N = f_n)$$
$$= \arg\max_y P(Y = y) \prod_{i=1}^{n} P(F_i = f_i \mid Y = y)$$

→ Parameter Estimation :-

Assume u have set of N sample points or obs $x_1 \dots x_N$ and believe that this data was drawn from a dist parametrized by unknown θ.

o Maximum Likelihood Estimation (MLE) makes following estimation:-

- Each sample is drawn from the same dist (each x_i is identically distributed)
- Each sample x_i is cond^n independent, given the parameter for our dist
- All possible values of θ are equally likely before we've seen any data
- $\mathcal{L}(\theta) = P_\theta(x_1, \dots x_n)$
  - using i.i.d → $\mathcal{L}(\theta) = \prod_{i=1}^{N} P_\theta(x_i)$
- Lastly we check $\frac{\partial}{\partial \theta}\mathcal{L}(\theta) = 0$ to get value of θ

eg: A bag with red and blue balls, we yield R, R, B, what is θ.

$$P_\theta(x_i) = \begin{cases} \theta & R \\ 1-\theta & B \end{cases}$$

$$\mathcal{L}(\theta) = \theta \cdot \theta \cdot (1-\theta) = \theta^2 - \theta^3$$

$$\frac{\partial}{\partial \theta}\mathcal{L}(\theta) = \frac{\partial}{\partial \theta}(\theta^2 - \theta^3) = 2\theta - 3\theta^2 = 0$$

$$\theta = \frac{2}{3} \quad \text{or} \quad \theta = 0$$

→ **Smoothing:-**

- Technique we use to adjust the prob used in model so that our model can perform more accurately and even handle words absent in training data.

Overfitting with NB's classifier can be mitigated by ↗

- Laplace Smoothing
  - ↳ $P_{MLE}(x) = \frac{count(x)}{N}$
  - with strength k → $P_{LAP,k}(x) = \frac{count(x) + k}{N + k|x|}$

$$P_{LAP,k}(x|y) = \frac{count(x,y) + k}{count(y) + k|x|}$$

- When $k = 0$, $P_{LAP,0}(x) = P_{MLE}(x)$
  $k = \infty$, $P_{LAP,\infty}(x) = 1/|x|$

NOTE 20

→ **Linear Classifiers:-**

- Classification using a linear combⁿ of the features - value called activation.

$$activation_w(x) = h_w(x) = \sum w_i f_i(x) = w^T f(x) = w \cdot f(x)$$

$$classify(x) = \begin{cases} + & \text{if } h_w(x) > 0 \\ - & \text{if } h_w(x) < 0 \end{cases}$$

↳ $||w||\,||f(x)|| \cos\theta$ ↳ since magnitude is always non-neg

$$classify(x) = \begin{cases} + & \cos(\theta) > 0 \\ - & \cos(\theta) < 0 \end{cases}$$

$$classify(x) = \begin{cases} + & \theta < \pi/2 \\ - & \theta > \pi/2 \end{cases}$$

- If $h_w(x) = 0$ or $\theta = \pi/2$ (orthogonal) we draw a blue line called decision boundary. ← seperates regions of + and −

→ **Binary Perceptron:-**

- Goal is to find the decision boundary that perfectly seperates training data
- Algorithm:-

Example

Let's see an example of running the perceptron algorithm step by step.

Let's run one pass through the data with the perceptron algorithm, taking each data point in order. We'll start with the weight vector $[w_0, w_1, w_2] = [-1, 0, 0]$ (where w_0 is the weight for our bias feature, which remember is always 1).

Training Set

#	f_1	f_2	y^*
1	1	1	-
2	3	2	+
3	2	4	+
4	3	4	+
5	2	3	-

Single Perceptron Update Pass

step	Weights	Score	Correct?	Update
1	[-1, 0, 0]	$-1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 = -1$	yes	none
2	[-1, 0, 0]	$-1 \cdot 1 + 0 \cdot 3 + 0 \cdot 2 = -1$	no	+[1, 3, 2]
3	[0, 3, 2]	$0 \cdot 1 + 3 \cdot 2 + 2 \cdot 4 = 14$	yes	none
4	[0, 3, 2]	$0 \cdot 1 + 3 \cdot 3 + 2 \cdot 4 = 17$	yes	none
5	[0, 3, 2]	$0 \cdot 1 + 3 \cdot 2 + 2 \cdot 3 = 12$	no	-[1, 2, 3]
6	[-1, 1, -1]			

1. Initialize all weights to 0: $\mathbf{w} = \mathbf{0}$
2. For each training sample, with features $\mathbf{f}(\mathbf{x})$ and true class label $y^* \in \{-1, +1\}$, do:
   (a) Classify the sample using the current weights, let y be the class predicted by your current $\mathbf{w}$:
   $$y = \text{classify}(x) = \begin{cases} +1 & \text{if } h_w(\mathbf{x}) = \mathbf{w}^T\mathbf{f}(\mathbf{x}) > 0 \\ -1 & \text{if } h_w(\mathbf{x}) = \mathbf{w}^T\mathbf{f}(\mathbf{x}) < 0 \end{cases}$$
   (b) Compare the predicted label y to the true label y^*:
   - If $y = y^*$, do nothing ← classifier is right
   - Otherwise, if $y \neq y^*$, then update your weights: $\mathbf{w} \leftarrow \mathbf{w} + y^*\mathbf{f}(\mathbf{x})$ ← classifier is wrong
3. If you went through **every** training sample without having to update your weights (all samples predicted correctly), then terminate. Else, repeat step 2

→ Multi class Perceptron:-

• Every similar to binary, instead have many weight vectors. So we keep it in a weight matrix.

eg: $w_0 = [-2 \quad 2 \quad 1]$
$w_1 = [0 \quad 3 \quad 4]$
$w_2 = [1 \quad 4 \quad -2]$
$\longrightarrow w = \begin{bmatrix} -2 & 2 & 1 \\ 0 & 3 & 4 \\ 1 & 4 & -2 \end{bmatrix}$ and $f(x) = \begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix}$

$$\text{label} = \arg\max(w \cdot x) = \arg\max \begin{bmatrix} 11 \\ 13 \\ 8 \end{bmatrix} = 1$$

NOTE 21

→ Linear Regression:- (Least squares)

• Regression model are form of ML prob in which the output is continuous.

↳ use the following linear model to predict: $h_w(x) = w_0 + w_1 x^1 + \dots w_n x^n$

w_0 is intercept of the model ← w_i : what we want to estimate

↳ $\text{Loss}(h_w) = \frac{1}{2} \|y - Xw\|^2 \quad \left(\left(\sum_i h_w(x^i) w - y^i \right)^2 / 2 \right)$

x^i is ith data point.

$$y = \begin{bmatrix} y^1 \\ \vdots \\ y^n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_1^1 \dots x_1^n \\ 1 & x_2^1 \dots x_2^n \\ \vdots & \\ 1 & x_n^1 \dots x_n^n \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix}$$

• $\hat{w} = (X^T X)^{-1} X^T y$

↳ $h_{\hat{w}}(x) = \hat{w}^T x$

→ Optimization:-

• Gradient based method: idea behind this is that the gradient points towards the direction of steepest inc of the objective.

↳ Maximize a func by moving towards the steepest ascent direction

↳ Minimize a func by moving towards the steepest descent direction.

• Gradient ascent: Used if the obj is a func which we try to maximize

Randomly initialize w
while w not converged do
  $w \leftarrow w + \alpha \nabla_w f(w)$
end

lr: need it to large enough to reach goal fast enough but small enough so that it not diverges

- **Gradient descent**: Used if obj is a loss func that we try to minimize

on large n points too computational

Randomly initialize w

while w not converged do

$w \leftarrow w - \alpha \nabla_w f(w)$

end

- **Least Square gradient descent**

Randomly initialize w

while w not converged do

$w \leftarrow w - \alpha(-X^T y + X^T X w)$

end

Used to intro non-linearity in neural networks

→ **Logistic Regression:-**

- Allows us to turn a linear combination of our input features into prob using logistic func:

$$h_w(x) = \frac{1}{1 + e^{-w^T x}}$$

↳ Used to solve classification prob, not regression prob.

- Logistic func $g(z) = \frac{1}{1+e^{-z}}$ is freq used to model binary outputs.

↳ Output always between 0 and 1

- For example, after we trained logistic regression, we have some data points.

If value of output is > 0.5 classify with label 1 else with label 0.

↳ $P(y = +1 \mid f(x); w) = \frac{1}{1+e^{-w^T f(x)}}$ | $P(y = -1 \mid f(x); w) = 1 - \frac{1}{1+e^{-w^T f(x)}}$

- Property note : $g'(z) = g(z)(1 - g(z))$

→ **Multi class Logistic Regression:-**

↳ Classify points into K distinct categories rather than just 2.

↳ How to make scores into probs?

$z_1, z_2, z_3 \longrightarrow \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}$

z_1, z_2, z_3: original activation

$\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}$: Softmax activation

eg: $w_1 = [-3, 4, 2]$, $w_2 = [2, 2, 7]$, $w_3 = [0, -1, 0]$, $x = [1, 2, 0]$

$w_1 x = 5$

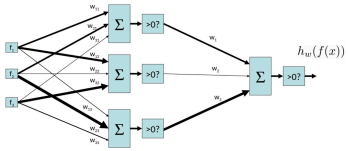
$w_2 x = 6$ ← highest so chooses this

$w_3 x = -2$

Prob class 1 $= \frac{e^5}{e^5 + e^6 + e^{-2}} = 0.2$ . . . .

NOTE 22

Perceptron that takes as input the outputs of another perceptron.



→ **Multilayer Perceptron:-**

- Universal Func Approx: A 2-layer NN with a sufficient no. of neurons can approximate any continuous func to any desired accuracy.

→ **Measuring Accuracy:-**

- accuracy of binary perceptron after making n preds :-

$$l^{acc}(w) = \frac{1}{n}\sum_{i=1}^{n}(\text{sgn}(w \cdot f(x_i)) == y_i)$$

y_i — actual class label of x_i; x_i — data point i

sgn — indicator func: $\begin{cases} -1 & x < 0 \\ 1 & x > 0 \end{cases}$

- likelihood of a particular set of weights explaining the obs labels and data pts. $l(w) = \prod_{i=1}^{n} P(y_i \mid f(x_i); w)$

$$\log l(w) = \log \prod_{i=1}^{n} P(y_i \mid f(x_i); w) = \sum_{i=1}^{n} \log P(y_i \mid f(x_i), w)$$

→ **Multilayer Feed forward NN:-**

- $f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & \text{otherwise} \end{cases}$
  - ↳ Sigmoid : $\sigma(x) = \frac{1}{1+e^{-x}}$
  - ↳ ReLU : $f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$

→ **Loss Functions & Multivariate Optimization:-**

- To maximize our log-likelihood func, we differentiate it to obtain a gradient vector consisting of its partial derivatives for each parameter

↳ $\nabla_w l(w) = \left[\frac{\partial l l(w)}{\partial w_1}, \ldots \frac{\partial l l(w)}{\partial w_n} \right]$

→ **Neural Network : Back propagation :-**

- Used to calculate the gradients for each parameter in a nn.

① Algorithm :-

- Set weights to some initial values
- Input training data, run forward pass to generate values at all nodes, calc loss func on final output.
- Run backward pass, calc the gradient of loss func w.r.t. each weight.
- Use gradient descent to update all the weights
- Repeat with more data.

Randomly initialize w
while w not converged do
  $w \leftarrow w - \alpha \nabla_w f(w)$
end

MDP was offline planning ← full knowledge of reward & transition states

NOTE 24

→ **Reinforcement Learning :-**

- **Online Planning :**
  - ↳ no prior knowledge of rewards or transition in the world
  - ↳ must try exploration which performs actions and recieves feedback in form of successor states it arrives in and corresponding rewards it reaps.

- **Sample as a tuple (s, a, s', r)**
  - ↳ collection of samples is called an **episode**
    - ↳ Agent goes to many episodes during exploration to collect suff data

→ **Passive RL** : learn values from experience using a given policy then use that to inform better policies.

① Model Based Learning :

- agent generates an approx of the transition func $\hat{T}(s, a, s')$ by keeping count of no. of times it arrives in each state s' after entering each Q-state (s, a). The agent can then generate the approx $\hat{T}$ upon real by normalising the counts it has collected dividing the count for each obs tuple.

eg:

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

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

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

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

s	a	s'	count
A	exit	x	1
B	east	C	2
C	east	A	1
C	east	D	3
D	exit	x	3
E	north	C	2

Recalling that $T(s,a,s') = P(s'|a,s)$, we can estimate the transition function with these counts by dividing the counts for each tuple (s,a,s') by the total number of times we were in Q-state (s,a) and the reward function directly from the rewards we reaped during exploration:

- **Transition Function**: $\hat{T}(s,a,s')$
  - $\hat{T}(A,exit,x) = \frac{\#(A,exit,x)}{\#(A,exit)} = \frac{1}{1} = 1$
  - $\hat{T}(B,east,C) = \frac{\#(B,east,C)}{\#(B,east)} = \frac{2}{2} = 1$
  - $\hat{T}(C,east,A) = \frac{\#(C,east,A)}{\#(C,east)} = \frac{1}{4} = 0.25$
  - $\hat{T}(C,east,D) = \frac{\#(C,east,D)}{\#(C,east)} = \frac{3}{4} = 0.75$
  - $\hat{T}(D,exit,x) = \frac{\#(D,exit,x)}{\#(D,exit)} = \frac{3}{3} = 1$
  - $\hat{T}(E,north,C) = \frac{\#(E,north,C)}{\#(E,north)} = \frac{2}{2} = 1$
- **Reward Function**: $\hat{R}(s,a,s')$
  - $\hat{R}(A,exit,x) = -10$
  - $\hat{R}(B,east,C) = -1$
  - $\hat{R}(C,east,A) = -1$
  - $\hat{R}(C,east,D) = -1$
  - $\hat{R}(D,exit,x) = +10$
  - $\hat{R}(E,north,C) = -1$

T(S,a,S'): No. of times u end up in S' after being in state S and taking action a, divided by total no. of times took a from s.

R(S,a,S'): Avg reward u got from string S, taking action a, ending up in S'.

② Model-free Learning:-

- **Direct Evaluation:-**
  - Fix some policy π and have the agent experience several episodes while following π.
  - As agent collects sample through these episodes it maintains counts of the total utility obtained from each state and number of times it visited the state.
  - At any point compute estimated value of any state s by dividing the total utility obtained from s by number of times s was visited.

eg:

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

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

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

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

S	Total Reward	Times Visited	$V^{\pi}(s)$
A	-10	1	-10
B	8+8	2	8
C	8+8	4	4
D	10+10+10	3	10
E	-4	2	-2

- **Temporal Difference Learning (TD-Learning):-**
  - Uses the idea of learning from every experience, rather than simply keeping track of total rewards and no. of times states are visited and learning at the end as direct evaluation does.
  - **Algorithm:-**
    1. $\forall s,\ V^{\pi}(s) = 0$
    2. Take action $\pi(s)$ and recieve reward $R(s, \pi(s), s')$
    3. Obtain sample, $\text{sample} = R(s, \pi(s), s') + \gamma V^{\pi}(s')$
    4. Update value estimate with exponential moving avg

$$V^{\pi}(s) \longleftarrow (1-\alpha)V^{\pi}(s) + \alpha \cdot \text{sample}$$

where α = lr, $0 \le \alpha \le 1$

- Learn at every time stamp (using iterative update
- give exponentially less weight to older, potentially less accurate sample
- Converge to learning true state values much faster with fewer episodes.

→ **Active RL**: Learn policy from our experience directly, can generate and test better policies while training

① **Q-Learnig**:-

- Learn $Q(s,a)$ values - which gives us the optimal policy (maximize Q)
- Entirely model free
- **Algorithm**:-
  1. Initialize $Q(s,a)$ estimates to 0
  2. For each timestamp:-
     * Take some action, any action! It doesn't need to be optimal
     * Obtain sample, $\text{sample} = R(s,a,s') + \gamma \max_{a'} Q(s',a')$
     * Update Q-value estimate with expo moving avg

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

  3. If we explore enough and dec α appropriately, learn optimal Q-value

② **Approximate Q-Learning**:-

- Key to generalizing learning experience is feature-based rep of states which rep each state as a vector known as feature vector.
  - ↳ eg: feature vector of pac man may include:-
    - dist to closest goal
    - dist to the closest food pallet
    - no. of ghost
- $V(s) = w_1 \cdot f_1(s) + \ldots + w_n f_n(s) = \vec{w} \cdot \vec{f}(s)$

$Q(s,a) = w_1 \cdot f_1(s,a) + \ldots + w_n f_n(s,a) = \vec{w} \cdot \vec{f}(s,a)$

↳ where $\vec{f}(s) = [f_1(s)\ f_2(s)\ \ldots\ f_n(s)]^T$

$\vec{f}(s,a) = [f_1(s,a) \ldots f_n(s,a)]^T$

$\vec{w} = [w_1 \ldots w_n]$

↳ $\text{diff} = [R(s,a,s') + \gamma \max_{a'} Q(s',a')] - Q(s,a)$

↳ update = $w_i \leftarrow w_i + \alpha \cdot \text{difference} \cdot f_i(s,a)$

- Final Note, reexpress update rule for exact Q-learning using diff as

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \text{difference}$$

NOTE 25

→ **Multiple-Armed Bandit Problems:-**

Can choose one action at each step, get reward

- Set of available action A
- Prob dist of reward given actions: $R^a(r) = P[r|a]$
- At each timestamp t, agent picks action $a_t \in A$ and gets random reward $r_t \sim R^{a_t}$ based on action
- Goal to max cumulative reward.

→ **Regret:-**

- Action value $Q(a)$ is mean reward for action a, $Q(a) = E[r|a]$
- Optimal value V^* is action-value for best action, $V^* = Q(a^*) = \max_{a \in A} Q(a)$
- Regret I_t is expected opportunity loss for one step, considering possibly random action a_t. $I_t = E[V^* - Q(a_t)]$
- Total Regret L_t : Total regret till timestamp t, $L_t = E\left[\sum_{\tau=1}^{t} V^* - Q(a_\tau)\right]$
- Gap is diff bet optimal value and value of a $\Delta_a = V^* - Q(a)$