CHEAT SHEET

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

$$ReLu(x) = max(0, x)$$

Min-max scale x within range [a,b]

$$x' = (b-a)\frac{x - x_{min}}{x_{max} - x_{min}} + a$$

Simple Action-Value Estimation takes average reward of action a

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

Incremental Estimation updates estimate when a is taken at time step t

$$Q_{t+1}(a) = Q_t(a) + \frac{1}{K_a}(R_{t+1} - Q_t(a))$$

Perceptron Weight Update Rule

$$w_i = w_i + \alpha(target - predicted) * input_i$$

Perceptron Bias Update Rule

$$b = b + \alpha(target - predicted)$$

Note that Perceptron Weight and Bias Update Rule, predicted value is rounded to nearest class label if binary classification task

Rule NN Size: num_params < num_training_samples / 10

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

Softmax
Calculate probability of each action using above formula, then choose the one for which the cumulative probability exceeds the random probability (do in order of action definition)

Discounted return starting time step t

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

If we are given a reward sequence, we can take all rewards after the last one to be 0

Note K_a is num times a was taken

TD(0) Prediction method

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

[...] is called the TD error
R{t+1} is reward for going from S{t} to S{t+1}

SARSA

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

Q-Learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a') - Q(S_t, A_t)]$$

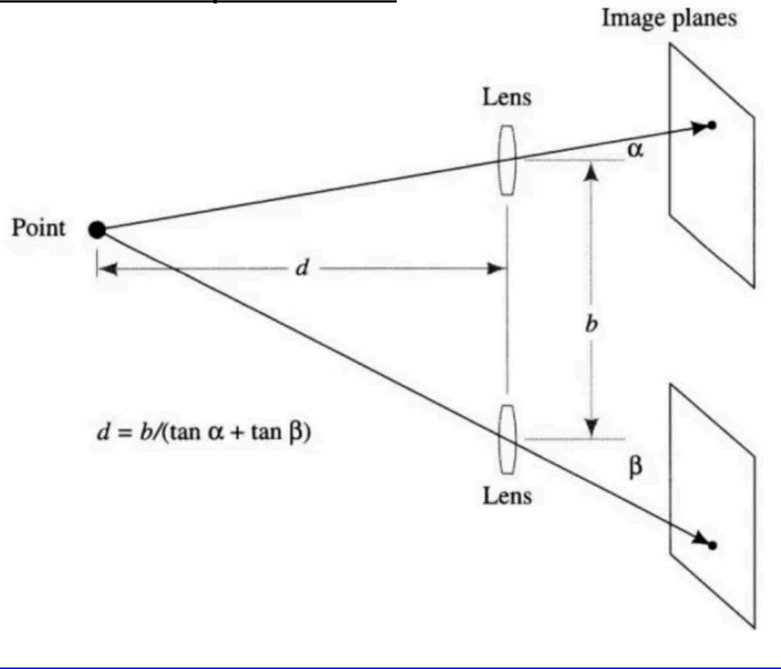
Simulated Annealing

```
// T_init and T_fin -> the initial and final temperature
// alpha -> the cooling rate in (0, 1)
// N the number of iterations to go for
sim_annealing_opt(T_init, alpha, N, T_fin) -> Soln {
    let T = T_init
    // generate an initial solution
    let S_curr = gen_init_soln()
    // while current T greater than final T
    while T >= T_fin {
        // generate neighbour of current soln
        for _ in 0..N {
            let S_new = gen_neighbour_from(S_curr);
            let delta = f(S_new) - f(S_curr)
            // if random exploration chance, or if S_new is better
            if rand(0, 1) < e ^ (-delta/T) || delta < 0 {
                S_curr = S_new
            }
        }
        // cool the temperature
        T = alpha * T
    }

    return S_curr
}
```

Tabu Search

```
tabu_search(N, tenure) -> Soln {
    let tabu_list = []
    let S_curr = gen_init_soln()
    tabu_list.push(S_curr)

    let S_best = S_curr

    // while stopping criterion is not met
    while !stopping_criterion() {
        let best_neighbour = None
        // generate and iterate over neighbours
        for nbour in gen_neighbours(S_curr) {
            // if neighbour is tabu and does not meet the
            // aspiration criteria, skip it
            if tabu_list.contains(nbour) && !asp_crit(nbour) {
                continue
            // if neighbour is better than the best known neighbour
            } else if f(neighbour) < f(best_neighbour) {
                best_neighbour = neighbour
            }
        }
        // move to the best non-tabu neighbour, even if it is worse
        // than the current soln
        S_curr = best_neighbour
        tabu_list.push(S_curr)

        if f(S_curr) < f(S_best) {
            S_best = S_curr
        }
        if len(tabu_list) >= tenure { // optional
            tabu_list.pop()
        }
    }

    return S_best
}
```

Stereo Vision Depth Estimation



Unigram Probability

$$P(W) = \prod_{i=1}^{n} P(w_i)$$

NN Num Parameter Calculation
(L_i + 1) * L_j for number of Params between L_i, L_j

Bigram Probability

$$P(W) = P(w_1) \times \prod_{i=2}^{n} P(w_i | w_{i-1})$$

Bigram Probability

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

Bayes Theorem

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

Gradient Descent Weight Update Formula

$$w_{ij} = w_{ij} - \alpha \frac{\partial Loss}{\partial w_{ij}}$$

Chain: $P(A,B,C,D) = P(A|B,C,D) \times P(B|C,D) \times P(C|D) \times P(D)$

Product: $P(A,B) = P(A) \times P(B|A) = P(B) \times P(A|B)$

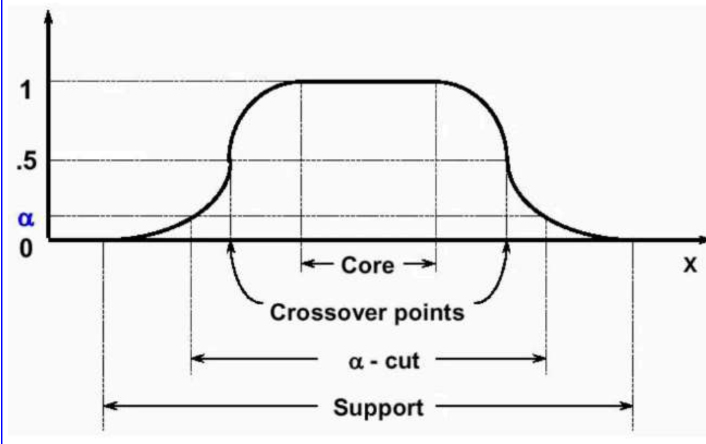
Chain Rule can also be applied like so

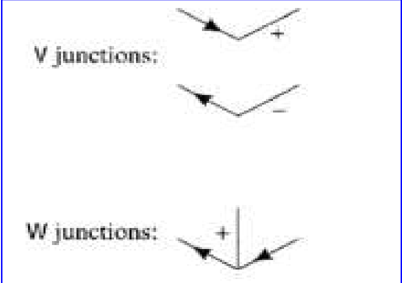
$$P(A|B,C) = P(A|D,B) \times P(D|B,C)$$

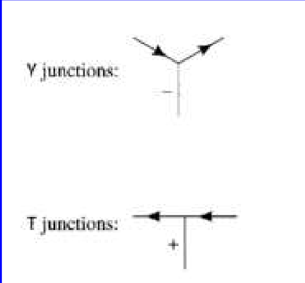
Regex is a formal language for specifying text strings

- `[]` matches any chars inside the brackets → `[nN]uke` matches `nuke` and `Nuke`
- `[A-Z], [a-z], [0-9]` matches any char within the range in the brack → can have `[a-zA-Z]` to match any alphabetic char → `[0-9] nukes` matches `2 nukes`, `3 nukes`, etc.
- `^` means negation if it is the first char in `[]` → e.g. `[^a-zA-Z]` means not alphabetic, but `[a-zA-Z^0-9]` matches a letter or '^' or a digit, and `a^b` matches "a^b" only
- `|` for disjunction → `don|key` matches `don` and `key`
- `?` optionally matches the previous char
- `*` matches 0 or more of the previous char
- `+` matches 1 or more of the previous char
- `.` matches any 1 char
- `\d` for digits, `\D` for no digits
- `\s` for whitespace `\S` for no whitespace
- `\w` for word (`[a-zA-Z0-9_]`), `\W` for non-word characters
- `{x}` means x of the preceding char → e.g. `a{4}` matches `aaaa` and `\d{4}` matches `1234`, `5678`, `8888`, etc.
- Outside of `[]`, `^` indicates the start and `$` indicates the end of a pattern to match
  - `.* ^abc$ .*` matches anything "abc" anything
  - `^[^a-zA-Z]` would match "123" of "123ABCabc"
  - `.$` would match "!" from "Holy moly!"
- `\` is the escape char → e.g. `\(brackets\)` matches "(brackets)"

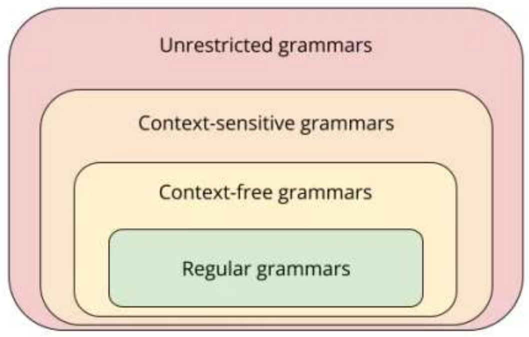
Fuzzy Set Parts







Chomsky's Hierarchy



$$crossover\_rate = \frac{n_{offspring}}{population\_size}$$

Binary Vision Thresholding

$$bin[x,y] = \begin{cases} 1 & \text{if } og[x,y] > threshold, \\ 0 & \text{otherwise} \end{cases}$$

Simple Q-Value Update

$$Q(s_k, a) = r(s_k, a) + \gamma * \max(Q(s_{k+1}, \_)$$

Optimal State-Value Function

$$V^*(s_k) = R + \gamma V^*(s_{k+1})$$

R is reward and s{k+1} is next state when following optimal policy

Action-Value Function

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

Note: s' is the next state if we take action a in state s

Simple Action-Value Estimation

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

K_a is the number of times action a has been taken

if $K_a = 0$, $Q_t(a)$ is defined with an arbitrary (initial) value (e.g. 0)

if $K_a \to \infty$, $Q_t(a)$ converges to $q_*(a)$

MSE

$$E = \frac{1}{N} \sum (target - predicted)^2$$

Conditional Probability

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

Byte-Pair-Encoding

Let vocabulary be the set of all individual characters

= {A, B, C, D,..., a, b, c, d....}

Repeat:

- Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
- Add a new merged symbol 'AB' to the vocabulary
- Replace every adjacent 'A' 'B' in the corpus with 'AB'.

Until *k* merges have been done.

We typically append '_' to words in the training data to avoid subword matches

Iter	Samp	w1	w2	w3	w4	b	ws	targ	delta	Update
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										
21										
22										
23										
24										

SUMMARY

AGENTS

- Reactive, Model-Based, Planning, Utility-Based, Game-Playing and Learning Agents

KNOWLEDGE REPRESENTATION

- Rule-Based Systems
  - Deduction: cause + rule ⇒ effect
  - Abduction: effect + rule ⇒ cause
  - Induction: cause + effect ⇒ rule
- Ontological Engineering defines a hierarchy of categories that represents knowledge (concepts and relationships) in a specific domain
- Parents are more abstract, child are specialisations
-

SEARCH

- BFS, DFS, Depth Limited Search, Iterative Deepening Depth First Search
- Bidirectional starts a search from start and goal node → if BFS is used, unweighted soln is shortest path
- Uniform Cost Search BFS using PQ sorted on costs of paths → reduces to BFS if all weights are the same
- Greedy Best First Search → DFS but pick neighbours in increasing order of h(n) → no opt path guarantee, and can get stuck in loops
- A* Search → UCS using f(n)=g(n)+h(n), where g(n) is actual cost from start to curr → optimal soln if heuristic never overestimates
- Heuristic is admissible if it always underestimates
- h1 dominates h2 if h2 >= h1 always → (h2 is more accurate) → true >= h2 >= h1 → note that h1, h2 are both admissible

NEURAL NETWORKS

- Feed-Forward NNs receive input upstream, and deliver output downstream → one direction, no loops
- Recurrent NNs feed outputs back to its own inputs
- McCulloch Pitts model → 1 if weighted sum > theta, else 0

REINFORCEMENT LEARNING

- Rewards communicate what to achieve, not how to achieve it
- Specifically concerned w learning good policy (action selection) through trial and error
- Softmax avoids cases where the worst actions are very bad → chance of selecting them is low
- For softmax, large τ makes actions equally likely, small τ makes actions with higher values more likely
- Markov Property → state should retain compact info abt the past to make optimal decisions
- Temporal Difference Prediction methods predict the agent's future based on current state and action
- SARSA: on-policy TD method (uses results of its actual policy) → learns safest, longest path
- Q-learning is off-policy TD method (uses results of 'optimal' policy, not own) → learns optimal path (risk)

OPTIMISATION METHODS

- Gradient Descent → adjust weights in opposite direction of computed gradient
- SDG → GD using small (>= 1) subset of random samples to calculate gradient in each iteration
- ADAM adapts individual learning rates for each weight → uses momentum (adds some of the previous update to the current one)
- Recall P != NP, NP-C subset of NP
- NP-C approximation through heuristics or genetic algorithms

OPTIMISATION METHODS – SOLN SPACE STUFF

- Feasible solns are solns within the search space (all possible solns) that satisfy all problem constraints
- Constraints are rules/conditions → can be strong (mandatory) or weak (recommendation)
- Objective function evaluates how good a soln is
- Neighbourhood Search Procedures → search neighbourhood for better solns until an exit condition is met, at which point return the best one found
- Neighbourhood of a soln is all feasible solns the curr one can be transformed into in one transformation
- Can explore only feasible region, or entire soln space
- Rejection strat → ignore infeasible solns
- Repair strat → transform infeasible solns into feasible ones
- Prevention strat → soln representation and transformation ops ensure infeasible solns aren't generated
- Simulated Annealing is a memoryless metaheuristic used to find near-optimal solns in scenarios with many good and few optimal solns
- Tabu Search is a memory-based metaheuristic using a tabu list to track recently visited solns → aspiration criteria untabus solns so algo doesn't skip good ones

OPTIMISTION METHODS - POPULATION BASED

- Can simultaneously explore multiple regions of search space → avoid premature convergence to suboptimal solns, but may require careful parameter tuning, and can be computationally expensive
- Chromosomes (solns) represented as bits
- Crossover op involves swapping bits of two chromosomes at a crossover point
- Mutation involves manipulating bits → e.g. bit flip
- Selection involves choosing chromosomes to continue onto the next generation through a fitness function that evaluates chromosomes
- Selective pressure → high favours fitter chromosomes, low is more random (exploration)
- Expanded Selection Method chooses everything
- Stochastic Sampling introduces randomness to prevent super fit chromosomes from dominating → prevents premature convergence
- Deterministic Sampling takes fittest chromosomes
- Note population size doesn't change for Deterministic and Stochastic

SUMMARY

COMPUTER VISION

- Challenges include variable lighting, complex and hard-to-describe or overlapping objects, shadows
- Perspective projection → map 3D scene to 2D image plane → many points along a viewing direction are projected onto a single point on the 2D plane
- Many-to-one transformation means we lose spatial and depth info unless we use multiple images from different angles
- Binary Vision simplifies images to binary images for algorithms to process
- Averaging technique smoothes pixels by using the avg value of a neighbourhood/window → reduces noise, but may blur the image and lose sharp details
- Cropping skips pixels where the window is not within the bounds of the image matrix
- Padding uses default values for out-of-bounds window pixels
- Edge Enhancement makes edges more visible → apply filters to detect edges (severe intensity transitions), and emphasise pixels along edges
- Region Finding groups pixels into meaningful regions based on specific criteria → find connected regions within images that represent objects/areas of interest
- Regions are homogeneous if “max - min <= ε”
- Split-and-merge splits non-homogenous regions in 4, recursing until all regions are homogeneous then merging adjacent ones if the result is also homogeneous

COMPUTER VISION – SCENE ANALYSIS

- Scenes are visual environments containing objects, regions, activities
- General info is stuff like camera location and orientation, light sources, indoor/outdoor setting
- Specific info is stuff like objects detected, their classes, estimated depth, and recognised activities
- Iconic models are structured models of key elements representing a scene → maps/structural diagrams simplifying objects into their basic form whilst preserving spatial arrangements and relationships
- We fit straight lines to straight lines, and curved ones to neat conic sections (ellipses, parabolas, etc.)
- Fitting algos can try to infer and fill in missing edges
- Occlusion (→): intersection of two planes where one blocks the other
- Blade (+): intersection of planes at convex (pop-out) edge
- Fold (-): intersection of planes at concave (cave-in) edge
- V, W and T junctions look like the letter
- Stereo Vision perceives depth by analysing differences between multiple images from different viewpoints → kinda like triangulation

COMPUTER VISION – COGNITIVE VISION

- Extracting features from visual data for real-time control
- Predict-act-observe-adapt feedback loop
- Synonymy → understand synonyms
- Hypernymy → recognising is-a relationships

COMPUTER VISION – COGNITIVE VISION

- Vision V, Reasoning R
- V -> R: visual data informs R
- R -> V: R task prompts V
- V -> R -> V: V informs R, R prompts V for clarity
- R -> V -> R: R determines requirement for V to collect and feed specific data back to R
- R -> VV..V: R imagines or envisions multiple possible Vs

LANGUAGE PROCESSING – FL and GRAMMAR

- For Grammars, (T)* means 0 or more T, [T] means optional T
- Unrestricted Grammars → both sides of rules can have any number of symbols
- Context-sensitive Grammars → RHS contains at least as many symbols as LHS
- Context-Free Grammars → LHS is a single non-terminal
- Regular Grammars → LHS is a single non-terminal, RHS is a terminal optionally followed by a non-terminal

LANGUAGE PROCESSING – REGEXP

- Regex errors can be type 1 (false positives/matches) or type 2 (false negatives/non-matches)
- Capture groups are enclosed in parentheses
- Adding a **?:** after the first parentheses of a capture group means it doesn’t capture
- The xth capture groups that does capture can be referenced with \x

LANGUAGE PROCESSING – NLP

- Lemma is the base/dictionary form of a word → words can belong to a lemma
- Wordforms are the surface form of words in text
- Morphemes are small meaningful units making up words → stem is core unit, affixes stick onto stem
- Tokens are individual occurrences of words/symbols in text → different occurrences of a word are different tokens
- Types are unique instances of words → types of a text represent its distinct vocabulary
- NLP text normalisation tokenises words, normalises their form and segments sentences

LANGUAGE PROCESSING – N-gram Models

- Language models compute the probability of an entire sequence of words, or the likely next word in a sequence → uses conditional probability
- Generally, the probability of a sequence is P(w1) * P(w2|w1) * P(w3|w2,w1)... → chain rule expansion of P(w1,w2,w3,...)
- LMs learn patterns and probabilities from datasets
- Markov assumption → next word depends only on prev k words → practical way of calculating conditional probabilities for LMs
- Unigram uses k=1 → probability of a sequence is the product of the probabilities of each individual word
- Bigram uses k=2 → probability of a word depends only on the previous one → probability of a sequence is product of probabilities of each word pair (wi w{i-1})
- Probability of P(wi | w{i-1}) can be estimated through count(wi w{i-1}) / count(w{i-1}) in corpus

SUMMARY

UNCERTAIN REASONING

- Confidence factors quantify the certainty/belief in particular evidence → assign to rules and inputs
- IF A THEN B → CF(B) = CF(A) * CF(IF A THEN B)
- CF(A and B) = min(CF(A), CF(B))
- CF(A or B) = CF(A) + CF(B) - (CF(A) * CF(B))
- Inference networks are sequences of relationships between facts and confidence factors
- Bayesian Networks are space efficient representations of full joint probability distributions as DAGs
- Directed edge from cause to immediate effects → each variable/fact/event gets a node
- Events are only conditionally dependent if they are immediate neighbours

UNCERTAIN REASONING – FUZZY LOGIC

- Degrees of truth instead of binary boolean logic → e.g. a temp of 25 is 0.8 warm and 0.3 hot
- Fuzzy sets have membership function A(x)
- Support of a FS is set of all x where A(x) != 0
- Crossover points for FS is x where A(x) = 0.5
- Core of FS is set of all x where A(x) = 1.0
- Height of FS is max A(x)
- Alpha-cut is set of all x where A(x) > alpha (>= for strong alpha-cut (alpha+))
- For OR membership, we take the max membership value
- For AND membership, we take the min membership value
- Rules use linguistic variables → IF x is A THEN y is B
- When aggregating results of rules for a particular case, we take the maximum output for each membership
- Defuzzification → convert fuzzy set into single crisp (clearly defined, binary) value for decision making
- Fuzzy Inference → use fuzzy logic to transform and map crisp inputs to crisp outputs → fuzzify inputs using membership functions, evaluate rules on fuzzified inputs, defuzzify results to obtain outputs

HUMAN-ALIGNED ROBOTICS

- Physics Engines simulate physical interactions in the environment → e.g. gravity, force, etc.
- Simulation Systems combine physics engines with additional tools to create full simulations of robotic systems that can be platform-specific or generic
- Human-in-the-loop learning → use human input in training and decision-making process of AI
- Consistent feedback always follows the same rules and provides the same clear guidance → good for training
- Policy shaping → trainer can change action selected by learner
- Reward shaping → trainer can modify reward for learner's action
- Persistent Rule-Based Interactive Learning introduces info retention and reuse → helps learner stay aligned with critical behaviours or safety protocols

HUMAN-ALIGNED ROBOTICS – Multimodal Integration

- Multi-Modal Associative Architectures are AI systems designed to process and integrate info from multiple sensory modalities
- Each modality is processed separately and outputs their own integrated label value (λ) and confidence value (γ)
- Outputs are combined in the Association/Fusion layer

HUMAN-ALIGNED ROBOTICS – Contextual Affordances

- Affordances represent characteristics of relationships between agent and object
- Objects have physical properties that suggest certain uses
- Context (location, time, etc.) influences which affordances are relevant (goal shapes interaction)
- Contextual Affordances consider the robot's state and context with the object's affordances

HUMAN-ALIGNED ROBOTICS – XAI

- Non-experts must be able to understand robots → we need to be able to answer why or why not regarding a robot or Ai's behaviour
- Featured-based or goal-driven explanation, but NO technical jargon and stuff
- Memory-Based method involves storing decisions and actions from previous episodes to answer "why"
- Hierarchical goals are split into multiple tasks of different levels in the hierarchy, and tasks must be completed in the hierarchical order
- Goal-driven explanations → explanation relates specifically to achieving a target/goal

Simple Behaviours	Lvl1	Rule-based logic
ML	Lvl2	Basic ML w/o complex understanding
	Lvl3	Complex pattern Recognition
	Lvl4	Understand and generate human text
Optimisation	Lvl5	Static optimisation of fixed goals
	Lvl6	Sequential decision problems; planning
Frontier of General	Lvl7	Reasoning, creativity and judgement; most advanced AI

EXTRA STUFF/NOTE TO SELF

- Overparameterisation → too many params relative to training data → NN learns noise instead of patterns
- Overfitting → NN learns patterns overly-specific to training data and does not generalise well
- Grammar G = (V, Sigma, R, S), where V is non-terminal set, Sigma is terminal set, R is rule set, S is start symbol

EXAMPLES

RL - SOFTMAX

Consider an RL agent navigating a gridworld, with four possible action: up (U), down (D), left (L), and right (R). The agent uses the softmax action selection method. Remember this method computes the probability of selecting an action using a Boltzmann distribution, as follows:

$$P(s_t, a) = \frac{e^{Q(s_t,a)/T}}{\sum_{a_i \in A} e^{Q(s_t,a_i)/T}}$$

In a particular given state S_t, the agent has the following Q-values to decide what action to take next:

Q(U)	Q(D)	Q(L)	Q(R)
0.7698	0.6501	0.0252	-0.7698

What would be the action selected by the agent if the temperature T used is 0.9 and the random number drawn is 0.9021.

Answer:

a. Up
b. Down
c. Left
d. Right
e. The agent would stay at the same position

Explanation (do not use in Moodle):

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}

RL - DISCOUNTED RETURN

Consider the return equation shown below with a discount factor γ = 0.9 and a reward sequence c
7, 3, 1, 10, -10. The return G_0 is equal to:

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

Answer:

a. 4.71
b. 1
c. 1.9
d. 11.239
e. 9.2

Explanation (do not use in Moodle):

G_5 = 0 (and all the following ones)

G_4 = -10

G_3 = 10 + 0.9 x G4 = 10 – 9 = 1

G_2 = 1 + 0.9 x G3 = 1 + 0.9 = 1.9

G_1 = 3 + 0.9 x G2 = 3 + 1.71 = 4.71

G_0 = 7 + 0.9 x G1 = 7 + 4.239= = 11.239

COMPUTER VISION – Averaging example with 3x3 window using cropping

$$\begin{array}{|ccc|c} \hline 9 & 9 & 9 & 3 \\ 9 & 9 & 3 & 3 \\ 9 & 3 & 3 & 3 \\ \cline{1-3} 3 & 3 & 3 & 3 \end{array} \longrightarrow \begin{array}{cccc} 9 & 9 & 9 & 3 \\ 9 & 7 & 5 & 3 \\ 9 & 5 & 4 & 3 \\ 3 & 3 & 3 & 3 \end{array}$$

RL - TD PREDICTION

Consider an agent using Temporal Difference (TD) learning to estimate the value function V for states S1 and S2. The agent follows a policy π that chooses actions based on the current state. The observed transitions and rewards are:

- Transition from S1 to S2 with a reward R = 4
- Transition from S2 back to S1 with a reward R = 1

Assume the current value estimates are V(S1) = 2 and V(S2) = 3. Using a learning rate α = 0.5 and a discount factor γ = 0.9, update the value of V(S1) using the TD(0) method, as follows:

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

What is the updated value of V(S1)?

Given:

- V(s1) = 2 (current estimate)
- V(s2) = 3
- R = 4 (reward for transition from s1 to s2)
- α = 0.5 (learning rate)
- γ = 0.9 (discount factor)

For s1:

V(s1) = 2 + 0.5[4 + 0.9 × 3 − 2]

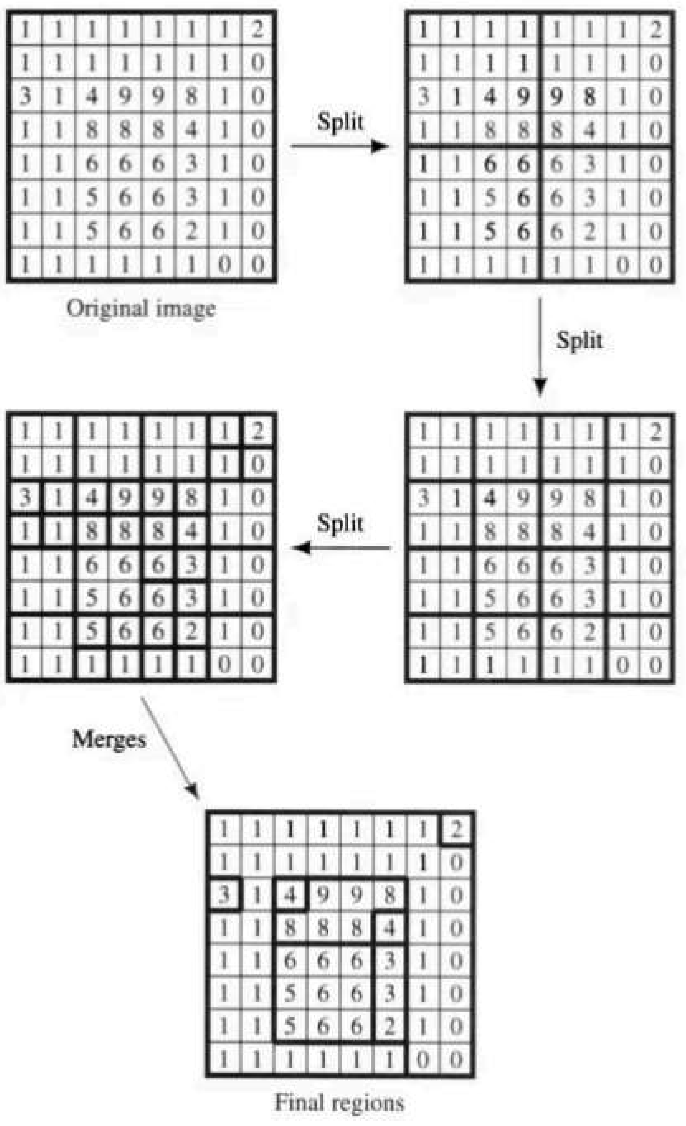
= 2 + 0.5[4 + 2.7 − 2]

= 2 + 0.5[4.7]

= 2 + 2.35

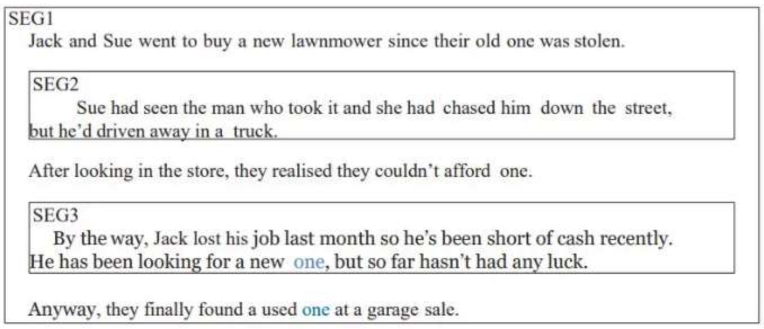
= 4.35

COMPUTER VISION - Split-Merge region finding example with ε=1



EXAMPLES

LANGUAGE PROCESSING



SEG1
Jack and Sue went to buy a new lawnmower since their old one was stolen.

SEG2
Sue had seen the man who took it and she had chased him down the street, but he'd driven away in a truck.

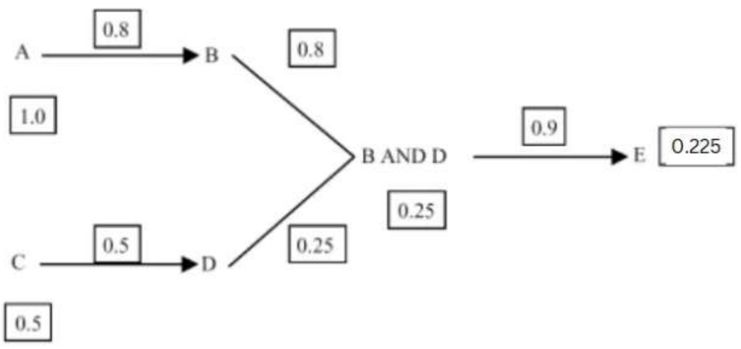
After looking in the store, they realised they couldn't afford one.

SEG3
By the way, Jack lost his job last month so he's been short of cash recently. He has been looking for a new one, but so far hasn't had any luck.

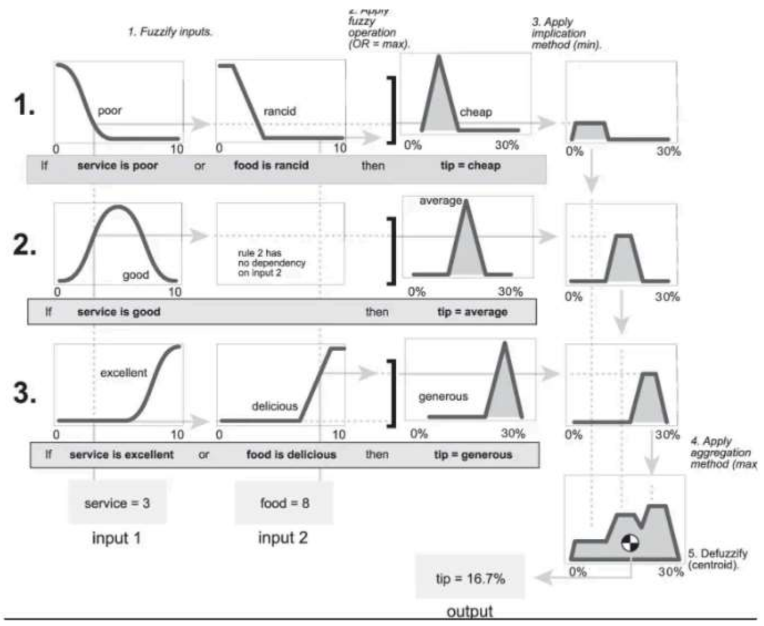
Anyway, they finally found a used one at a garage sale.

- SEG1 is the main narrative
- SEG2 provides background information about why the main narrative is happening
- SEG3 provides contextual information to support reasoning throughout the main narrative

UNCERTAIN REASONING – INFERENCE NETWORK

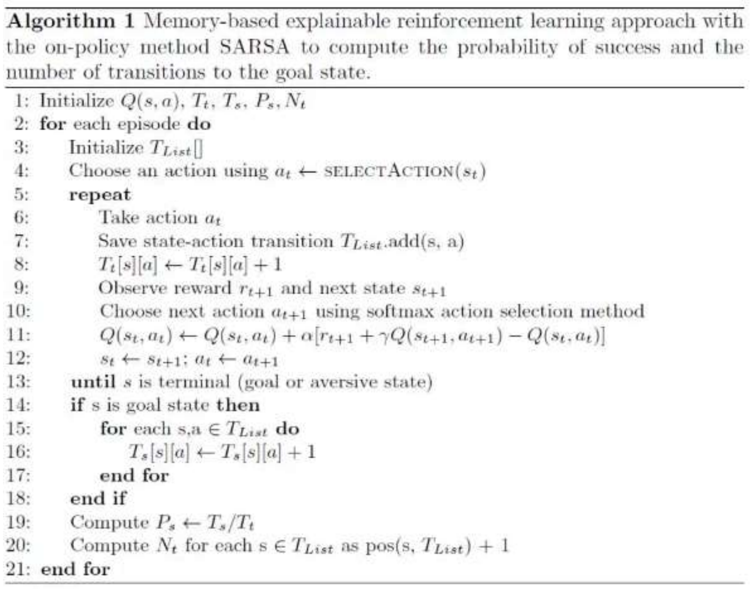


UNCERTAIN REASONING – FUZZY INFERENCE EXAMPLE



1. Fuzzify inputs.
2. Apply fuzzy operation (OR = max).
3. Apply implication method (min).
1.
poor
rancid
cheap
If service is poor or food is rancid then tip = cheap
2.
good
rule 2 has no dependency on input 2
average
If service is good then tip = average
3.
excellent
delicious
generous
If service is excellent or food is delicious then tip = generous
4. Apply aggregation method (max
service = 3
food = 8
input 1
input 2
5. Defuzzify (centroid).
tip = 16.7%
output
0
10
0%
30%

HUMAN-ALIGNED – MXRL

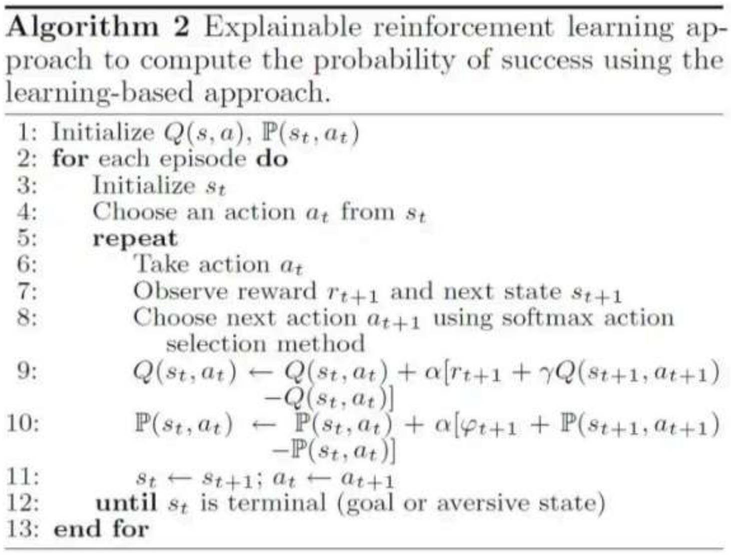


Algorithm 1 Memory-based explainable reinforcement learning approach with the on-policy method SARSA to compute the probability of success and the number of transitions to the goal state.

```
Initialize $Q(s,a)$, $T_t$, $T_s$, $P_s$, $N_t$
for each episode do
    Initialize $T_{List}[]$
    Choose an action using $a_t \leftarrow$ SELECTACTION($s_t$)
    repeat
        Take action $a_t$
        Save state-action transition $T_{List}$.add(s, a)
        $T_t[s][a] \leftarrow T_t[s][a] + 1$
        Observe reward $r_{t+1}$ and next state $s_{t+1}$
        Choose next action $a_{t+1}$ using softmax action selection method
        $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
        $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
    until $s$ is terminal (goal or aversive state)
    if $s$ is goal state then
        for each s,a $\in T_{List}$ do
            $T_s[s][a] \leftarrow T_s[s][a] + 1$
        end for
    end if
    Compute $P_s \leftarrow T_s/T_t$
    Compute $N_t$ for each s $\in T_{List}$ as pos(s, $T_{List}$) + 1
end for
```

- T_t is the total number of times each state-action pair is visited
- T_s is the total number of successful occurrences of each state-action pair (how many eps we visit the state-action pair in end in success)
- P_s is the probability of success for taking specific actions in state s
- N_t is the number of transitions required to reach the goal state from a given state-action pair

HUMAN-ALIGNED – LEARNING-BASED METHODS



Algorithm 2 Explainable reinforcement learning approach to compute the probability of success using the learning-based approach.

```
Initialize $Q(s,a)$, $\mathbb{P}(s_t,a_t)$
for each episode do
    Initialize $s_t$
    Choose an action $a_t$ from $s_t$
    repeat
        Take action $a_t$
        Observe reward $r_{t+1}$ and next state $s_{t+1}$
        Choose next action $a_{t+1}$ using softmax action
            selection method
        $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1})$
            $- Q(s_t,a_t)]$
        $\mathbb{P}(s_t,a_t) \leftarrow \mathbb{P}(s_t,a_t) + \alpha[\varphi_{t+1} + \mathbb{P}(s_{t+1},a_{t+1})$
            $- \mathbb{P}(s_t,a_t)]$
        $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
    until $s_t$ is terminal (goal or aversive state)
end for
```

Learning-Based (algo2) methods are approaches where an AI agent learns directly from interactions with the environment → RL algos are learning-based

- $\mathbb{P}(s_t, a_t)$ represents the probability of success for taking action a_t in state s_t
- $\mathbb{P}$ allows agents to provide explanations based on actual experiences
- Relies on RL to iteratively improve Q-values and update probability of success for each state-action pair

EXAMPLES

HUMAN-ALIGNED – INTROSPECTION-BASED

Algorithm 3 Explainable reinforcement learning approach to compute the probability of success using the introspection-based approach.

```
1: Initialize $Q(s,a)$, $\hat{P}_s$
2: for each episode do
3:     Initialize $s_t$
4:     Choose an action $a_t$ from $s_t$
5:     repeat
6:         Take action $a_t$
7:         Observe reward $r_{t+1}$ and next state $s_{t+1}$
8:         Choose next action $a_{t+1}$ using softmax action
               selection method
9:         $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1})$
               $-Q(s_t,a_t)]$
10:        $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
11:    until $s_t$ is terminal (goal or aversive state)
12:    $\hat{P}_s \approx \left[(1-\sigma)\cdot\left(\frac{1}{2}\cdot log_{10}\frac{Q(s_t,a_t)}{R^T}+1\right)\right]_{\hat{P}_s \geq 0}^{\hat{P}_s \leq 1}$
13: end for
```

Introspection-Based (algo3) methods are approaches where an AI agent uses internal reasoning or self-assessment to learn?

- Uses internal metrics (e.g. Q-values) and theoretical approximations to estimate the success probability
- $\hat{P}_s$ represents the introspective estimated success probability → allows agents to explain that an action was believed to be likely based on its internal reasoning of potential outcomes

RL – VALUE FUNCTIONS

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

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

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

Answer: The optimal policy is:

$\pi^*(S_1) = a_2$
$\pi^*(S_2) = a_2$

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. $V^*(S_2) = 5 + 0.9 * 3.5/0.19 = 21.58$

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:

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

UNCERTAIN REASONING – BAYESIAN NETWORK

Consider the following random variables:

rsp: regular sport practice.

bd: balanced diet.

hbp: high blood pressure.

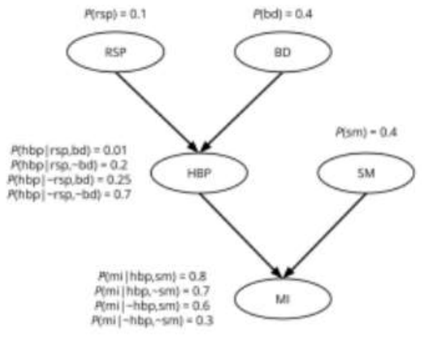
sm: smoker.

mi: has suffered a myocardial infarction.

Answer:

a. 0.4853
b. 0.4
c. 0.6872
d. 0.7
e. 0.5147

The causal relationships and the probabilistic knowledge are shown in the Bayes network below. Using the network, compute the probability of being a smoker given that a myocardial infarction has been suffered and no regular sport is practised, i.e., P(SM|MI, ~RSP).



P(S|M, ~R) = P(S|M) (by Conditional Independence)

P(S|M) = P(S,M)/P(M)

P(S,M) = P(M,S) = P(M,S,H) + P(M,S,~H) = P(M|H,S)P(H,S) + P(M|~H,S)P(~H,S) =
P(M|H,S)P(H|S)P(S) + P(M|~H,S)P(~H|S)P(S)
= P(M|H,S)P(H)P(S) + P(M|~H,S)P(~H)P(S) (by Conditional Independence)
= 0.8 * 0.4 * P(H) + 0.6 * 0.4 * P(~H)
= 0.32 * P(H) + 0.24 * P(~H)

P(M) = P(M,S,H) + P(M,S,~H) + P(M,~S,H) + P(M,~S,~H) = 0.74 * P(H) + 0.42 * P(~H)

P(H) = P(H,R,B) + P(H,R,~B) + P(H,~R,B) + P(H,~R,~B)
= P(H|R,B)P(R)P(B) + P(H|R,~B)P(R)P(~B) + P(H|~R,B)P(~R)P(B) + P(H|~R,~B)P(~R)P(~B)
= 0.4804

P(~H) = 1 - P(H) = 0.5196

P(S,M) = 0.278432
P(M) = 0.573728

So P(S|M) = P(S,M)/P(M) = 0.4853031402