Intelligent Agents

PEAS framework Performance measure, Environment, Actuators, Sensors

Environment	Description
Observable	access to state of environment at each point in time
Episodic (vs sequential)	experience is divided in different atomic episodes
Static (vs (semi-) dynamic)	environment unchanged while deliberating
Discrete (vs	limit on number of distinct actions
Single-agent (vs multi-agent)	operates by itself
Deterministic (vs stochastic)	state completely determined by current state and action

Exploration	Exploitation
Learning more about environment	Maximise gain based on current knowledge

Representation invariant Abstract states must have corresponding concrete states

Searching

Problem solving steps Goal formulation → problem formulation → search → execute

Problem formulation Find states, initial state, goal state/goal test, actions, transition model, action cost function

Problem vs search tree Graph structure to model the path (problem) vs path taken to search (search tree)

Path cost Cost of path from state to any state

Optimal path cost Cost of lowest-cost path cost from any state to any state

State space All possible configurations

Search space Subset of state space that is to be searched

Evaluation

Time complexity Number of nodes generated or expanded

Space complexity Maximum number of nodes in memory

Complete Algorithm is complete if for every problem instance it will find a solution if one exists

Optimal Algorithm is optimal if for every instance where it produces a solution, the best solution is produced

Uninformed Search

Breadth-first search

Frontier used: **Queue**

Time complexity	exponential w.r.t depth of optimal solution
Space complexity	exponential w.r.t depth of optimal solution
Complete	number of nodes finite
Optimal	step cost is the same

Time complexity $O(b^d)$, where b total number of notes, d depth

Uniform-cost search

Frontier used: **Priority queue**

Time complexity	exponential w.r.t tier of optimal solution
Space complexity	exponential w.r.t tier of optimal solution
Complete	step cost always positive and finite total cost
Optimal	step cost always positive

Space complexity and time complexity $O(b^{I+\lfloor C^{*}\epsilon \rfloor})$, where C^* refers to cost of optimal solution, ϵ lower bound of cost of each action.

Depth-first search

Frontier used: **Stack**

Time complexity	exponential w.r.t max. depth of search tree
Space complexity	polynomial w.r.t to max. depth of search tree
Not complete	when depth of search tree is infinite
Not optimal	solution may be in shallower depth

Depth-limited search

Limits search to depth l.

Time complexity, space complexity, completeness, optimality **dependent on search algorithm**.

Iterative-deepening search

Uses DLS, iteratively increasing depth limit from 0 to ∞ until solution found.

Space complexity and time complexity dependent on search algorithm. **Complete and optimal**.

Informed Search

Heuristics

Admissability Never overestimates cost

$$h(n) \leqslant h^*(n)$$

Consistency Fulfills triangle inequality

$$h(n) \leqslant c(n, a, n') + h(n'), h(G) = 0$$

Dominance A more dominant admissible heuristic function is better for search.

$$h_1(n) \geqslant h_2(n) \implies h_1 \text{ dominant}$$

Best-first search

Similar to the other search, with a **priority queue**, with the priority of each node calculated as $f(n)$ (greedy-best first search: $f(n) = g(n)$)

A*Star Search

Cost: $f(n) = g(n) + h(n)$

Time complexity	exponential
Space complexity	exponential w.r.t depth of optimal solution
Complete	edge costs are positive, branching factor finite
Optimal	depends on heuristics

$h(n)$ admissable $\implies$ optimal A* search without visited memory

Non-admissable can still be optimal - if cost-optimal path is where h_n is admissible on all nodes OR if optimal solution C_1 and second-best optimal solution costs C_2 - if $h(n)$ overestimates costs, but never by more than $C_2 - C_1$

$h(n)$ consistent $\implies$ optimal A* search with visited memory

Local Search

Used as they use very little memory, and finds reasonable solutions in large or infinite state spaces

Hill-Climbing Search

Idea: Keep climbing until value of neighbour is less than value of current

Prone to local maxima, ridges, plateau - can get stuck as the best move is always picked

Simulated Annealing

Idea: Counter possibility of getting stuck in Hill Climbing

Let T be the "temperature" - it goes down over time. Picks a random move - if it helps situation it is picked. Otherwise, it is accepted with some probability, which decreases exponentially with the badness of the move, and decreases as T goes down. Bad moves thus become more unlikely as T decreases.

Adversarial Search

Used for competitive environments - which are fully observable, deterministic, discrete, have terminal states, zero-sum, and turn-taking.

Minimax

Idea: Assuming players play optimally, the player picks the best move, and the opponent picks the move worst for the player. (Always optimal)

Terminal State	$utility(s, max)$
Max Player moving	Max of utility from possible actions
Min Player moving	Min of utility from possible actions

Alpha-beta pruning

Idea: No point to explore full game tree. Keep track of α, β, the highesst-value and lowest-value choices found.

Perfect ordering: $O(b^{\frac{m}{2}})$.

Decision Tree

Fully expressive. Size of hypothesis class is generalised to 2^{2^n} for n boolean attributes.

Entropy

The entropy $B(q)$ of a Boolean random variable, with probability q

$$B(q) = -(q log_2 q + (1-q) log_2(1-q))$$

Thus, the entropy of the output

$$H(output) = B(\frac{p}{p+n})$$

To get the information gain of an attribute A, find the entropy of everything else except for what is in A:

$$remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B(\frac{p_k}{p_k + n_k})$$

The information gain is then calculated as the reduction in entropy:

$$IG(A) = H(output) - remainder(A)$$

Methods to generalise more

Pruning: **min-sample-leaf**, **max-depth**
Data preprocessing

Linear Regression

Idea: Best fit linear function for continuous-valued input

Loss function

$$J_{MSE}(w) = \frac{1}{2m}\sum_{i=1}^{m}(h_w(x^i) - y^i)^2$$

In matrix form:

$$\frac{1}{2m}(Xw - Y)\cdot(Xw - Y)$$

where X are variables, w are weights, $Xw = \hat{Y}$ are the predictions, and Y are the actual values.

Minimising loss function

Idea: Differentiate the loss function.

With regard to the weight w_i, the partial derivative is:

$$\frac{\delta}{\delta w_1}J_{MSE}(w) = \frac{1}{m}\sum_{i=1}^{m}(w_i x^i - y^i)x^i$$

Normal Equation

$$w = (X^T X)^{-1}X^T Y$$

Does not require feature scaling, single iteration, with no need for a learning rate.

$O(n^3)$ time to calculate inverse of X^TX, and X^TX needs to be invertible (full column rank)

Gradient descent

Idea: Start at a weight and pick a weight that reduces the loss until minimum is found.

With learning rate (hyperparameter),

$$w_j \leftarrow w_j - \frac{\delta}{\delta w_j}J(w_0, ...)$$

In this scenario, the MSE loss function is convex for linear regression, and thus gradient descent can be done on it.

There are different kinds of gradient descent: Stochastic (randomly selects one at a time), Mini-batch (selects of size m).

As data points decrease, cost of gradient descent is cheaper, and randomness increases, causing possibility of escaping minima (bounce)

Logistic regression

Use continuous value output probability for classification

Logistic function

Idea: Naive threshold is not differentiable and is completely confident. Soften with sigmoid function.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Loss function

J_{MSE} is not convex due to exponential function. Use: cross-entropy.

$$CE(y, \hat{y}) = \sum_{i=1}^{C} -y_i log(\hat{y}_i)$$

The binary cross entropy then can be calculated:

$$BCE(y, \hat{y}) = -ylog(\hat{y}) - (1-y)log(1-\hat{y})$$

Minimising the loss

Differentiating, the partial derivative with respect to w_j is found as

$$\frac{\delta}{\delta w_j}J_{BCE}(w) = \frac{1}{m}\sum_{i=1}^{m}(h_w(x^i) - y^i)x_j^i$$

For many attributes

$$h_{w(x)} = \sigma(w_0 + w_1 f_1 + ...)$$

Multi-class classifiers

One vs All	Test probability of all classes, assign highest probability
One vs One	Fit classifier for each pair, highest wins assigned

Performance Measure

Confusion Matrix

TP	FP (Type 1)
FN (Type 2)	TN

$$TPR = \frac{TP}{FN}, FPR = \frac{FP}{FP+TN}$$

Correctness (Classification)

Accuracy:

$$accuracy = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}^i = y^i)$$

Precision $\mathcal{P}$. Maximise if false positives (Type I) error costly.

$$\mathcal{P} = \frac{TP}{TP+FP}$$

Recall $\mathcal{R}$. Maximise if false negatives (Type II) error costly.

$$\mathcal{R} = \frac{TP}{TP+FN}$$

$$F_1 = \frac{2}{\frac{1}{\mathcal{P}} + \frac{1}{\mathcal{R}}}$$

Bias and Variance

Bias

High bias can cause algorithms to miss relevant relations, resulting in **underfitting**.

(Too less features)

Variance

High variance can cause algorithms to model random noise, causing **overfitting**.

(Too much features)

Hyperparameter tuning

Grid search (exhaustively try all)
Random search (randomly search)
Successive halving (use all, successively increase with smaller set)
Bayesian optimisation
Evolutionary algorithms