CS2109S
AY22/23 Sem 2
github.com/jasonqiu212

01. Introduction

- **Agent** - Anything that can perceive its environment through sensors and acting upon that env. through actuators
- **Agent Function** - Maps from percept histories to actions
- **Rational Agent** - Chooses an action that is expected to maximize its performance measure, given by percept sequence and built-in knowledge
- **Autonomous Agent** - If behavior is determined by its own expereince

Performance Measure of Function

- Motivation: For an agent to do the right thing, need a measure of goodness

Defining the Problem: PEAS

1. Performance measure
2. Environment
3. Actuators
4. Sensors

Characterizing the Environment

1. **Fully observable** - (vs. Partially) Agent's sensors can access complete state of env. all the time
2. **Deterministic** - (vs. Stochastic) Next state of env. is determined by **current state** and **action executed by agent**
   - **Strategic** - If env. is deterministic except for actions of other agents
3. **Episodic** - (vs. Sequential) Agent's experience is divided into atomic **episodes**, where each episode includes perceiving and an action, and **action depends on episode** itself
4. **Static** - (vs. Dynamic) Env. is unchanged while agent is deciding
   - **Semi** - Time does not affect env., but affects performance score
5. **Discrete** - (vs. Continuous) Discrete num. of percepts and actions
6. **Single Agent** - (vs. Multi-agent) Agent operating by itself in an env.

Implementing Agents (in ascending complexity)

1. **Simple Reflex Agents** - Fixed conditional rules
2. **Model-based Reflex Agents** - Stores percept history to make decisions about internal model of world with conditional rules. Eg. Roomba
3. **Goal-based Agents** - Keep in mind a goal and action aims to achieve it
4. **Utility-based Agents** - Find best way to achieve goal
5. **Learning Agents** - Learn from previous experiences

Exploitation vs. Exploration

- **Exploitation** - Maximize expected utility using current knowledge of world
- **Exploration** - Learn more about the world to improve future gains. May not always maximize performance measure.

02. Uninformed Search

- Deterministic, fully observable
- **Tree Search** - Can revisit nodes
- **Graph Search** - Tracks visited (Tree Search + Memoization)
- **Uninformed Search** - Uses only information available in problem definition

Formulating the Problem

1. How to represent state in problem?
2. Initial state
3. Actions: Successor function
4. Goal test
5. Path cost

- **Abstraction Function** - Maps abstracted representation to real world state
- **Representation Invariant** - $I(c) = \textbf{True} \rightarrow \exists a \textbf{ s.t. } AF(c) = a$

Breadth-first Search

- Idea: Expand shallowest unexpanded node using **queue**
- Given: b: Branching factor and d: Depth of optimal solution
- Complete: Yes (if tree is finite)
- Time: $O(b^{d+1})$
- Space: $O(b^d)$
- Optimal: Yes (if cost = 1)
- BFS is Uniform-cost Search with same cost

Uniform-cost Search

- Idea: Expand least-cost unexpanded node using **priority queue** (Dijkstra's)
- Given: C^*: Cost of optimal solution
- Complete: Yes (if step cost $\geq \epsilon$ where $\epsilon \geq 0$)
- Time: $O(b^{(C^*/\epsilon)})$ (C^*/ϵ is approx. number of layers)
- Space: $O(b^{(C^*/\epsilon)})$
- Optimal: Yes

Depth-first Search

- Idea: Expand deepest unexpanded node using **stack**
- Given: m: Maximum depth of tree
- Complete: No (fails with infinite depth or loops)
- Time: $O(b^m)$
- Space: $O(bm)$ (better than BFS)
- Optimal: No

Depth-limited Search

- Motivation: How to handle infinite depth for DFS?
- Idea: DFS with depth limit l where nodes at depth l have no children
- Time: $b^0 + b^1 + ... + b^{(d-1)} + b^d = O(b^d)$

Iterative Deepening Search

- Motivation: How to determine depth limit? We don't.
- Idea: Try different depths for depth-limited search
  - BFS pretending to be DFS to save space
- Complete: Yes
- Time: $(d+1)b^0 + db^1 + ... + b^d = O(b^d)$ (More overhead than DLS)
- Space: $O(bd)$

Summary

	BFS	Uniform Cost	DFS	DLS	IDS
Complete	Yes	Yes	No	No	Yes
Time	$O(b^d)$	$O(b^{C^*/\epsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{C^*/\epsilon})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal	Yes	Yes	No	No	No

Bidirectional Search

- Idea: Search both forwards from initial state and backwards from goal state. Stop when searches meet.
- Time: $O(2b^{d/2})$
- Operators must be reversible
- Can have many goal states
- How to check if node intersects with other half?

03. Informed Search

Heuristic

- **Heuristic** - Estimated cost from n to goal
- **Admissible** - $h(n)$ is admissible if, for every node n, $h(n) \leq h^*(n)$ where h^* is the true cost
  - if h is admissible, then A* using tree search is optimal
- **Consistent** - $h(n)$ is consistent if, for every node n and every successor n' of n generated by action a, $h(n) \leq c(n, a, n') + h(n')$
  - Triangle inequality
  - If h is consistent, $f(n)$ is non-decreasing along any path ($f(n') \geq f(n)$)
  - If h is consistent, then h is admissible
  - if h is admissible, then A* using graph search is optimal

Dominance

- If $h_2(n) \geq h_1(n)$ for all n, then h_2 **dominates** h_1
- If h_2 **dominates** h_1 and both are admissible, then h_2 is better for search

How to invent admissible heuristic?

- Set fewer restrictions on actions
- E.g. Number of misplaced tiles, Total manhattan distance

Best-first Search

- Idea: Expand most desirable node using priority queue
- Evaluation Function: $f(n) = h(n)$
- Complete: No. Possible to be stuck in loop
- Time and space: $O(b^m)$
- Optimal: No

A* Search

- Idea: Take note of cost so far and heuristic
- Evaluation Function: $f(n) = g(n) + h(n)$ where $g(n)$ is cost to reach n
- Complete: Yes, unless non-increasing, since cost is factored in
- Time and space: Same as BFS
- Optimal: Yes, depending on the heuristic

Iterative Deepening A* Search (IDA*)

- Motivation: How can we save space?
- Idea: Have a cutoff for f and remember the best f that exceeds cutoff
  - Similar to IDS. Linear space complexity.
- Optimal and complete

Simplified Memory A* Search (SMA*)

- Motivation: How can we save space?
- Idea: Do normal A*. If memory is full, drop node with worst f.
- Lose completeness

Local Search

- Motivation: What if the goal state is the solution? The path is irrelevant.
- Idea: Keep single current state and try improving it
- Formulating the problem:
  1. Initial state
  2. Actions: Successor function
  3. Good heuristic
  4. Goal test

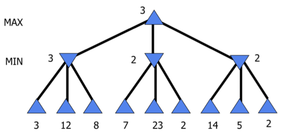
Hill-climbing

- Idea: Generate successors from current and pick the best using heuristic
- What if stuck into local minima?
  - Introduce randomness
  - **Simulated Annealing Search** - Allow some bad moves and gradually decrease frequency
- Why only keep 1 best state?
  - **Beam Search** - Perform k hill-climbing searches in parallel
- How to generate successors?
  - **Genetic Algorithms** - Successsor is generated by combining 2 parent states

04. Adversarial Search

- Assumption: Opponents reacts rationally
- Formulating the problem:
  - Initial state
  - Successor function
  - Terminal test
  - Utility function: Measures how good the move is for a player

Minimax



- Idea: Choose move that yields highest minimax value using DFS
- Complete: Yes (if tree is finite)
- Time: $O(b^m)$
- Space: $O(bm)$
- Optimal: Yes (against an optimal opponent)

Alpha-Beta Pruning

- Motivation: How to save time for Minimax?
- Idea: By tracking max. and min. values so far, can prune some paths that we would never choose
  1. α contains max. and β contains min.
  2. Initially, $\alpha = -\infty$ and $\beta = \infty$
  3. When going down, copy α and β
  4. Prune if $\alpha \geq \beta$
  5. When going up, depending on MIN/MAX level, update α or β
- With perfect ordering, time complexity: $O(b^{m/2})$. Doubles search depth.

Resource Limits

- In reality, search space for games can be very large. α-β pruning also not fast enough.
- Solution: Limit depth (Only see a finite moves ahead) and determine best move using evaluation function to estimate desirability of position (Heuristic)
- Other hacks:
  - **Transpoisitions** - Memoize equivalent states
  - Pre-computation of opening/closing moves

05. Introduction to Machine Learning

- A machine learns if it improves performance P on task T based on experience E. Where T must be fixed, P must be measurable, E must exist

Types of Feedback

- **Supervised** - Correct answer given for each example
  - **Regression** - Predict results within continuous output
  - **Classification** - Predict results in discrete output
- **Unsupervised** - No answers given
- **Weakly supervised** - Answer given, but not precise
- **Reinforcement** - Occasional rewards given

Decision Trees

- DT can express any function of input attributes, if data is consistent
- Goal: Make DT **compact**. How?

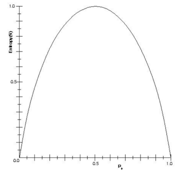
Information Theory

- Idea: Choose attribute that splits examples into subsets that are ideally 'all positive' or 'all negative'
- **Entropy** - Measure of randomness in set of data

$$I(P(v_1), ..., P(v_n)) = -\sum_{i=1}^{n} P(v_i)\log_2 P(v_i)$$

- For data with p positive examples and n negative examples:

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$



- **Information Gain** - (IG) Reduction in entropy from attribute test
- Goal: Choose attribute with largest information gain
- Intuition: IG = Entropy of this node - Entropy of children nodes
- Given chosen attribute A with v distinct values:

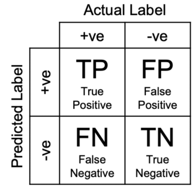
$$\text{remainder}(A) = \sum_{i=1}^{v}\frac{p_i+n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

$$IG(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - \text{remainder}(A)$$

- **Decision Tree Learning** - Recursively choose attributes with highest IG
- IG is not the only way. Can use whatever objective function that achieves the criteria we want.

Performance Measurement

- **Correctness** - Correct if $\hat{y} = y$
- **Accuracy** - $\frac{1}{m}\sum_{j=1}^{m}(\hat{y_j} = y_j)$
- Confusion Matrix:



- Accuracy $= \frac{TP+TN}{TP+FN+FP+TN}$
- **Precision** - $\frac{TP}{TP+FP}$ **Recall** - $\frac{TP}{TP+FN}$
- Type I Error: FP Type II Error: FN
- FP Rate $= \frac{FP}{FP+TN}$ TP Rate $= \frac{TP}{TP+FN}$

Pruning

- Motivation: DT overfits to training set, but performs poorly on test set
- Occam's Razor: Simple hypothesis preferred
- **Pruning** - Ignores outliers, which reduces overfitting
  - E.g. Min-sample, Max-depth

06. Linear Regression

Notation

- m = Number of training examples
- n = Number of features
- $x_j^{(i)}$ = Input feature j of ith training example
- y = Output variables

Hypothesis

$$h_w(x) : w_0 + w_1 x$$

Cost Function (Square Error Function)

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)})^2$$

- Goal: Minimize cost function. Thus, hypothesis is close to training samples
- Why squared error? Convenience, since we need to differentiate later

Gradient Descent

- Start at some (w_0, w_1). Pick nearby point that reduces $J(w_0, w_1)$.
- Algorithm: Repeat until convergence:

$$w_j := w_j - \alpha \frac{dJ(w_0, w_1, ...)}{dw_j}$$

- All updates done at end
- How to do $\frac{dJ(w_0,w_1)}{dw_j}$? Partial derivative: Hold everything else constant
  - $\frac{dJ(w_0,w_1)}{dw_j} = \frac{d}{dw_j}(\frac{1}{2m}\sum_{i=1}^{m}(w_0 + w_1 x^{(i)} - y^{(i)})^2)$
  - $\frac{dJ(w_0,w_1)}{dw_0} = \frac{1}{m}\sum_{i=1}^{m}(w_0 + w_1 x^{(i)} - y^{(i)})$ (Note: Chain rule)
  - $\frac{dJ(w_0,w_1)}{dw_1} = \frac{1}{m}\sum_{i=1}^{m}(w_0 + w_1 x^{(i)} - y^{(i)})x^{(i)}$
- Time complexity: $O(kmn)$ where k is number of iterations

Learning Rate

- If α too small, then descent is too slow. If α too big, then might overshoot.
- Given constant α, descent will grow smaller as we approach minimum

Variants of Gradient Descent

- Batch gradient descent: Consider all training examples when updating
- Stochastic gradient descent: Consider 1 random data point at a time (Cheaper and more randomness)
- Mini-batch gradient descent

Using Matrices

- Given: $w = \begin{pmatrix} w_0 \\ \vdots \\ w_n \end{pmatrix}$ and $x = \begin{pmatrix} x_0 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ x_n \end{pmatrix}$
- $h_w(x) : w^T x$

Feature Scaling

- Motivation: Gradient descent does not work well if features have different scales
- **Mean Normalization** - $x_i \leftarrow \frac{x_i - \mu_i}{\sigma_i}$

Normal Equation

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

- No need to choose α and feature scaling
- $X^T X$ needs to be invertible
- Time complexity: $O(n^3)$. Slow if n is big

07. Logistic Regression