CS 486 - W24
Introduction to Artificial Intelligence
Full Course Notes

With Prof Yuntian Deng

These are my in-class lecture notes. They cover all course content, besides example problems.

Josiah Plett

CS 486 Notes 1

Syllabus

- Search
- Uncertainty Estimation
- Markov Decision Process
- Machine + Deep Learning

② Uninformed Search

b = branching factor
m = max depth of nodes
d = depth of shallowest goal

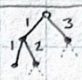
Search Problem:
↳ set of states
↳ initial state
↳ goal states or goal test
↳ successor function
↳ (optionally) cost function

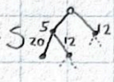
Algorithms

DFS 	$O(bm)$ space	$O(b^m)$ time
BFS 	$O(b^d)$ space	$O(b^d)$ time
IDS (DFS → BFS)	$O(bd)$ space	$O(b^d)$ time

③ Heuristic Search

Search Heuristic: $h(n) \approx d(goal)$

LCFS  exponential time, exponential space. C&O? yes & yes (mild)

GBFS  exponential time, exponential space. C&O? no & no.

A*S $f(n) = cost(n) + h(n)$ C&O? yes & yes with mild restrictions

if $h(n)$ is admissable, A* is optimal

Admissable: Never overestimates the cost of shortest path to goal.

Consistent: "Monotone" restriction. $h(m) - h(n) \le cost(m, n)$

Domination: $h_2(n)$ dominates $h_1(n)$ if for all n, $h_2(n) \ge h_1(n)$, and $h_2 \ne h_1$

Cycle Pruning: Ignore cycles!

Multiple-Path Pruning: Discard additional paths to the same node. (without considering relative costs)

④ CSP - Constraint Satisfaction Problems

Backtracking Search: On failure, backtrack (essentially DFS)

Arc Consistency: An arc $\langle X, c(X,Y)\rangle$ is arc-consistent iff $\forall v \in D_X, \exists w \in D_Y$ s.t. (v, w) satisfies $c(X,Y)$

AC-3: ① Put all arcs in S ② Remove all $v \in D_X$ that don't have $w \in D_Y$ that satisfies $c(X,Y)$ ③ Add arcs starting from Y ④ Repeat for all arcs till S is empty

Backtracking + AC-3: ① Do backtracking search ② After each assignment, do Arc Consistency. ③ If domain is empty or only has one option, we're done! ④ Keep backtracking.

⑤ Local Search

Local Search: non-systematic, non-guaranteed, neighbour-based local search that doesn't remember parent.
- State: assignment of all variables
- Neighbour Relation: next states?
- Cost Function: quality of state

Greedy Descent: Go to state with lower cost. Terminate otherwise.
↳ can add Random Restart or Random walk

Simulated Annealing: $A \to A' : e^{-\frac{\Delta C}{T}}$
T = temperature, gradually decreasing.
↳ geometrically

Population-Based: K states (not 1!)
- **Beam Search:** Use K best neighbours uniformly.
- **Stochastic Beam Search:** Chose K states probabilistically, $\propto e^{\frac{-cost(A)}{T}}$
- **Genetic Algorithm:** $2^{f(i)}$ parents → crossed-over child with possible mutation.

⑥ Probability

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

$$P(\neg A) = 1 - P(A)$$

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \leftarrow \text{Bayes'}$$

$$P(A \wedge B) = P(A) \times P(B|A)$$

Steps for Solving Anything ★

Conditional Probability: Convert into fraction (product / condition)

Joint Probability: Convert to summation of all variables

⑦ Independence + Bayesian Networks

Unconditional Independence

$P(X|Y) = P(X)$
$P(Y|X) = P(Y)$
$P(X \wedge Y) = P(X)P(Y)$

Conditional Independence

$P(X|Y \wedge Z) = P(X|Z)$
$P(Y|X \wedge Z) = P(Y|Z)$
$P(X \wedge Y|Z) = P(X|Z)P(Y|Z)$

Bayesian Network

DAG where a node is a random variable, each edge shows the child is conditional on the parent.

Full Joint Probability: $P(X_n \wedge \cdots \wedge X_1) = \prod_{i=1}^{n} P(X_i | Parents(X_i))$

⑧ D-Separation + Construction

D-Separation

$X \leftarrow E \rightarrow Y$

E d-separates X and Y iff E blocks every un-directed path between X and Y. → X & Y conditionally independent given E.

Constructing Bayesian Networks ★

1. Order variables $\{X_1, \dots, X_n\}$.
2. For each X_i:
   - 2.1. Choose the smallest set of parents from $\{X_1, \dots, X_{i-1}\}$ such that given $Parents(X_i)$, X_i is independent of all nodes $\{X_1, \dots, X_{i-1}\} - Parents(X_i)$.
   - 2.2. Link all parents to their child, X_i.
   - 2.3. Create conditional probability table $P(X_i | Parents(X_i))$.

example

Consider the net: $W \leftarrow A \rightarrow G$

Make a new net from order $\{W, G, A\}$

① W
② $W \rightarrow G$
③ $W \rightarrow G \rightarrow A$, $W \rightarrow A$

⑨ Statistical Inference (in Bayesian Networks)

$W \leftarrow A \rightarrow G$

$$P(A|W) = \frac{P(A \wedge W)}{P(W)} = \frac{\sum_g P(A)P(w|A)P(g|A)}{\sum_{a'}\sum_g P(a')P(w|a')P(g|a')}$$

← marginalized g

Using Variable Elimination ★ we then get:

$$= \frac{P(A)P(w|A)\sum_g P(g|A)}{\sum_{a'} P(a')P(w|a')\sum_g P(g|a')} = \frac{P(A)P(w|A)}{\sum_{a'} P(a')P(w|a')}$$

← normalization

Factor: Any function from random vars to a number.

Restricting a Factor: $f(X, Y, Z) \to f_2(Y, Z)$, $X = t$.

Summing out a Variable: $\sum_Y f(X, Y, Z) \to f_2(X, Z)$

VEA Summary

- Construct a factor for each conditional probability.
- Restrict observed variables to their observed values.
- Eliminate each hidden variable X_{h_i}.
  - Multiply all factors containing X_{h_i} to get new factor g_i.
  - Sum out variable X_{h_i} from the factor g_i.
- Multiply remaining factors + Normalize the result.

CS 486 Notes $\boxed{2}$

(10) Hidden Markov Models

Markov Assumption
The future is independent of the past given the present. (1st Order)

Stationary Process
Conditional probability of each step does not change over time.

Sensor Assumption
Each state is sufficient to generate its observation.

Inference Tasks (probabilities)

- Filtering: Current state?
  - ↳ posterior distribution w/all evidence to date
- Prediction: Next state?
- Smoothing: Previous state?
- Most Likely Explanation
  - ↳ Full sequence of states.

VEA with Enumeration $O(k \cdot 2^k)$
or Forward Recursion $O(k)$
Practice this! ★

(11) Inference w/Hidden Markov Models

Smoothing Calculations ★

⟹ Backward Recursion: $P(S_k | o_{0:(t-1)}) = \alpha P(S_k | o_{0:k}) P(o_{(k+1):(t-1)} | S_k)$
$= \alpha f_{0:k}\, b_{(k+1):(t-1)}$ f=forward, b=backward

① Base Case: $b_{t:(t-1)} = \vec{1}$

② Recursive Case: $b_{(k+1):(t-1)} = \sum_{s_{k+1}} P(o_{k+1} | S_{k+1})\, b_{(k+2):(t-1)} P(S_{k+1} | S_k)$

⟹ Forward-Backward Algorithm: that ↑ (1 forward pass, 1 backward)

⟹ Viterbi Algorithm: Given $\boxed{\pi_0}$ and probabilities $\boxed{P}$, return $\hat{S}$.

Dynamic Programming

① For $k = 1, \dots, t-1$
For $j = 0, \dots, n-1$
$\pi_k(j) = P(o_k | S_k = j) \max_z [\pi_{k-1}(z) P(S_k = j | S_{k-1} = z)]$
$\phi_k(j) = \arg\max_z [\pi_{k-1}(z) P(S_k = j | S_{k-1} = z)]$
Save last state $\hat{s}_{t-1} = \arg\max_j \pi_{t-1}(j)$

② For $k = t-1, \dots, 1$
$\hat{s}_{k-1} = \phi_k(\hat{s}_k)$

③ Return $\hat{S} = \hat{s}_0, \dots, \hat{s}_{t-1}$

(12) Decision Networks

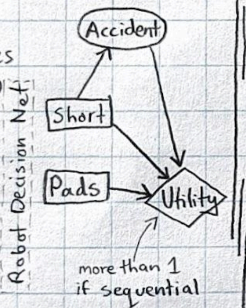
Decision Network = Bayesian Network + Actions + Utilities

Nodes

Chance Nodes
- Random Variables (like Bayesian nets)

Decision Nodes
- Actions (decision variables)

Utility Node
- Utility function for all states



(13) Markov Decision Processes

[A] Finite-stage: indefinite horizon; stops, but when?
[B] Ongoing: infinite horizon; may take forever.

[A] One-time Utility: only consider utility at the end.
[B] Sequential Rewards: rewards along the way.
↳ reward incorporates costs of actions/rewards/punishes.

Rewards

Total: $\sum_{t=0}^{\infty} R(S_t) = R(S_0) + R(S_1) + \dots$

Average: $\lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} R(S_t) = \lim_{n\to\infty} \frac{1}{n}(R(S_0) + R(S_1) + \dots)$

Discounted: $\sum_{t=0}^{\infty} \gamma^t R(S_t) = R(S_0) + \gamma R(S_1) + \dots;\ \gamma \in [0,1]$

(i) Partially-observable MDP (POMDP) = A MDP + HMM

Policies

π^* = optimal

A policy tells an agent what to do as a function of the current state.
- Non-stationary: $P(S, t)$
- Stationary: $P(S)$

$V^\pi(s)$: Expected utility of entering state s then following policy π.

$V^*(s)$: same as $V^\pi(s)$ but optimal π^*.

$Q^*(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$

$\pi^*(s) = \arg\max_a Q^*(s,a)$

(14) Value Iteration & Policy Iteration

$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a) = E_\pi\left[\sum_{j=0}^{T} \gamma^j R_{t+j+1} \middle| s = s_t\right]$

(informationally equivalent)

$Q^\pi(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s') = E_\pi\left[\sum_{j=0}^{T} \gamma^j R_{t+j+1} \middle| s_t = s, a_t = a\right]$

Bellman Equation ★

$V^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$

Value Iteration

① Arbitrary initial values $V_0(s)$
② $V_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s')$
③ Terminate when $\max_s |V_i(s) - V_{i+1}(s)| \approx 0$

Policy Iteration

① Alternate between Evaluation & Improvement
② Evaluation: $V^{\pi_i}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) V^{\pi_i}(s')$
③ Improvement: $\pi_{i+1}(s) = \arg\max_a \sum_{s'} P(s'|s,a) V^{\pi_i}(s')$

(15) Reinforcement Learning

PADP - Passive Adaptive Dynamic Programming

Repeat
① $\boxed{\text{Follow } \pi}$ to generate experience $\langle s, a, s', r \rangle$
② Update reward function $R(s) \leftarrow r$
③ Update transition probability
$N(s,a) \mathrel{+}= 1 \quad N(s,a,s') \mathrel{+}= 1$
$P(s'|s,a) = N(s,a,s') / N(s,a)$
④ Derive $V^\pi(s)$ w/ Bellman
$V(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V(s')$

Temporal Difference Error

$TD = (R(s_1) + \gamma \max_{a'} Q(s_2, a')) - Q(s_1, a)$
after experience $\langle s_1, a, r_1, s_2 \rangle$

Exploit: Action that maximizes $V(s)$

Explore: Action different from optimal one

↳ ε-greedy Exploration: $P(\text{explore}) = \varepsilon$, $\frac{d\varepsilon}{dt} < 0$

↳ Softmax using Gibbs/Boltzmann:
$P(a) = \dfrac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}}$, $T > 0$ is temperature

↳ Optimistic initial values to encarage exploration

AADP - Active Adaptive Dynamic Programming

Same as PADP, but instead of following a given policy, we determine optimal action a like below:

$a = \arg\max_a f\left(\sum_{s'} P(s'|s,a) V^+(s'), N(s,a)\right)$

$f(u,n) = \begin{cases} R^+; & \text{if } n < N_e \\ u; & \text{otherwise} \end{cases}$

↑ Repeat until each (s,a) is visited N_e times and all $V^+(s)$ converged.

(20) Gradient Descent (NN2) cont'd

Backpropagation Algorithm ★

(1) Given training examples $(\vec{x}_n, \vec{y}_n)$ and an error (loss) function $E(\hat{y}, y)$, perform:

(a) Forward pass: Compute error E.

(b) Backward pass: Compute gradients

$$\delta_k^{(2)} z_j^{(1)} = \frac{\partial E}{\partial W_{j,k}^{(2)}} \quad \text{and} \quad \delta_j^{(1)} x_i = \frac{\partial E}{\partial W_{i,j}^{(1)}} \leftarrow \text{Layer}$$

(2) Update each weight by the sum of the partial derivatives for all training examples.

Sigmoid Derivative

$$\frac{\partial g(x)}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2} = g(x)(1-g(x))$$

Error Function

$$E = \sum_k (\hat{y}_k - y_k)^2$$

Recursive Relationship

$$\delta_j^{(\ell)} = \begin{cases} \frac{\partial E}{\partial z_j^{(\ell)}} \times g'(a_j^{(\ell)}) \\ \left[\sum_k \delta_k^{(\ell+1)} W_{j,k}^{(\ell+1)}\right] \times g'(a_j^{(\ell)}) \end{cases}$$

(21) Batched, Momentum, Adaptive (NN 3)

(Batch) Gradient Descent

$$\hat{g} \leftarrow \frac{1}{N} \nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$$

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

Stochastic Gradient Descent

$$\hat{g} \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$$

$$\theta \leftarrow \theta - \underset{\text{learning rate}}{\epsilon} \hat{g}$$

Step size: $\epsilon \|g\|$

Momentum

$$V \leftarrow \alpha V - \epsilon \overbrace{\nabla_\theta L(f(x^{(i)};\theta), y^{(i)})}^{\hat{g}_{SGD}}$$

$$\theta_i \leftarrow \theta_{i-1} + V$$

α is usually set high, $\alpha \gg \epsilon$

Step size: $\epsilon \frac{\|\hat{g}\|}{1-\alpha}$

Nesterov Momentum

Same as momentum above, but when calculating V, use $\theta + \alpha V$.

Adaptive Gradient

$r=0$, init ϵ, θ, δ.

$$\hat{g} \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$$

$$r \leftarrow r + \hat{g} \odot \hat{g} \quad \text{(accumulation)}$$

$$\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g}$$

$$\theta \leftarrow \theta + \Delta\theta$$

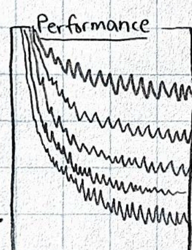
Root-Mean-Square

Same as AdaGrad, except for accumulation:

$$r \leftarrow \underset{\text{initial param}}{\rho} r + (1-\rho)\hat{g} \odot \hat{g}$$

Adaptive Delta

- Improved AdaGrad.
- Removes hand-set learning rate ϵ.
- Learning rate = diff between current and new weights.



ADAM (Adaptive Moments)

$s=0$, $r=0$, $t=0$, init ϵ, decay rates $\rho_1, \rho_2, \theta, \delta$

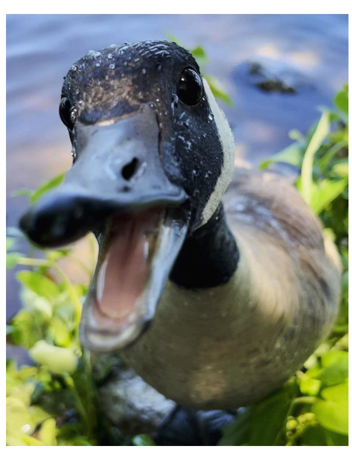
$$\hat{g} \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)}) \qquad t \leftarrow t+1$$

$$s \leftarrow \rho_1 s + (1-\rho_1)\hat{g} \qquad \hat{s} \leftarrow \frac{s}{1-\rho_1^t}$$

$$r \leftarrow \rho_2 r + (1-\rho_2)\hat{g} \odot \hat{g} \qquad \hat{r} \leftarrow \frac{r}{1-\rho_2^t}$$

$$\theta \leftarrow \theta - \epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$$

If you thank Mr. Goose for how easy this course was bf the final...



Mr. Goose will thank you and bless your weary mind with an easy final ❤️