CS188 MT1 CHEATSHEET

Note1: Search

admissible: nonnegative and underestimate
consistency: $h(a) - h(c) \le cost(A,C)$
- admissibility guarantees optimal-tree search
- consistency guarantees optimal tree/graph search
  - must give h=0 as 0 in goal state

Note2: Game Trees

- AlphaBetaPruning

```
def max-value(state,α,β):
    initialize v=-∞
    for each successor of state:
        v = max(v, value(successor,α,β))
        if v ≥ β return v:
        α = max(α,v)
    return v
def min-value(state,α,β):
    initialize(state,α,β)
    .....
        v = min(v, value(successor,α,β))
        if v ≤ α return v
        β = min(β,v)
    ...
```

Note3: MDPs (NON deterministic search) [know all info]

- Discounted utility guarantees convergence:
  - $R(s_0,a_0,s_1) + \gamma R(s_1,a_1,s_2) + \gamma^2(\dots) = \sum_{t=0}^{\infty} \gamma^t R(s_t,a_t,s_{t+1})$
  - $\le \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-R}$
- Markovianness (memoryless) MDP:
  - $T(s,a,s') = P(s' \mid s,a)$
- Solving MDPs (optimal policy π yields max expected total utility/reward)
  - $V^*(s)$: optimal value of state s; optimal expected utility starting at s
  - $Q^*(s,a)$: optimal value of q-state (s,a); optimal value of utility starting at s, taking action a, and acting optimally henceforth

Bellman Equation

- $V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
- optimal q-value:
  - $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
- simplified:
  - $V^*(s) = \max_a Q^*(s,a)$

★ $V^*(s)$ is simply the max expected utility over all possible actions from s.

Value Iteration: [returns the value of a state]
- max expected utility in state s given that the Markov decision process terminates in k timesteps
  1. $\forall s \in S$, initialize $V_0(s)=0$
  2. Repeat until convergence:
     $\forall s \in S, V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$
- Runtime: $O(|S|^2|A|)$

Policy Extraction [returns optimal action from each state]
- $\forall s \in S, \pi^*(s) = \arg\max_a Q^*(s,a) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
- it's better to have optimal q-values of states, then just need argmax it.

Policy Iteration
- faster than value iteration
1. define any random policy
2. Repeat until convergence:
   - assign $V^{\pi_i}(s)=0$ for all terminal states
   - evaluate current policy with policy evaluation.

Policy Evaluation: compute $V^\pi(s)$ for all states s, where $V^\pi(s)$ is expected utility starting at s, following π.
$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$] start with $V^\pi{}_0$

   - define π_i as policy iteration at i: each $V^{\pi_i}(s)$ can be solved through systems of equations
3. use policy improvement to generate better policy:
   - use policy extraction on the values generated by policy evaluation to generate improved policy.
   - $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$

- UCS, A* guarantee optimal solution
- BFS optimal if all actions same cost
- Greedy search relies all on heuristic
  - UCS - backward cost to origin
  - A* - use f(n) = g(n) + h(n)

Note4: RL

- definition:
  - sample: (s,a,s',R) tuple

Model Based Learning
- approximate transition function $\hat{T}(s,a,s')$ and $\hat{R}(s,a,s')$
- generate π using value iteration/policy iteration

Model Free Learning

- Passive Reinforcement Learning

1. Direct Evaluation
   1. follow a fixed policy, calculate multiple episodes, then calculate $V^\pi(s)$ as total reward/times visited
   - slow to converge, learns at the end

2. Temporal Difference Learning
   1. initialize $\forall s, V^\pi(s) = 0$ (converges to V^* only if executed policy $\pi = \pi^*$)
   2. update as you go:
      - $V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + \alpha \cdot sample$ where
        - $sample = R(s,\pi(s),s') + \gamma V^\pi(s')$
   - learns at every timestep
   - give exponentially less weight to older, potentially less accurate samples
   - converge to learning true state values faster

- Active Reinforcement Learning

1. Q Learning (converges to V^* when visit all states)
   - Q Value Iteration update rule: - all Qs start as 0.
     - $Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$
     - $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot sample$ where
       - $sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$

2. Approximate Q-Learning
   - $difference = [R(s,a,s') + \gamma \max_{a'} Q(s',a')] - Q(s,a)$
   - update: $w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$
   - $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot difference$

where $\vec{w} = [w_1, w_2 \dots]$ weight vector
$\vec{f}(s) = [f_1(s)\ f_2(s) \dots]^T$
$\vec{f}(s,a) = [f_1(s,a) \dots f_n(s,a)]^T$
are feature vectors for state s and q state (s,a)

if question asks "if Q(left) = Q(right), then it means $V^* = Q(left) = Q(right)$. Use system of eq.

Epsilon Greedy: takes random action & probability $(1-\epsilon)$ optimal.

$Q(s,a) = w_1 f_1(s,a) + \dots + w_n f_n(s,a)$

$Q_{new}(s,a) = R(s,a,s') + \gamma \max_{a'}$ - update each weight after each observation

Note 5: CSP

- identification problem: is state s a goal state
- CSP is NP-hard

Solving CSP

1. Backtracking search:
   - optimization of DFS
     1) Fix an ordering for variables, and select values for variables in this order. Assignments are commutative
     2) select variables that don't conflict with previously assigned values. Backtrack otherwise

Improvements:

1. Filtering
   1) Forward checking: whenever a value is assigned to a variable X_i, prune domains of unassigned variables that share a constraint with X_i that could violate constraint if assigned.

   ↓ generalize into ARC consistency Algorithm

   - interpret each undirected edge of constraint graph as two directed ones. Each directed edge is an arc
   1) Begin by storing all arcs in the constraint graph for the CSP in a queue Q.
   2) iteratively remove arcs from Q and enforce the condition that in each removed arc $X_i \to X_j$. For every remaining value v for the tail variable X_i, there is atleast one remaining value w for the head variable X_j such that $X_i = v, X_j = w$ doesn't violate constraints. If some value v for X_i wouldnt work with any remaining X_j values, we remove v from the set of possible values for X_i.
   3) if atleast one value is removed for X_i, when enforcing arc $X_i \to X_j$, add arcs $X_k \to X_i$ to Q, for all unassigned X_k.
   4) continue until Q empty, or domain empty and backtrack.

   Runtime: $O(ed^3)$ where e = # arcs: $\max(n^2 d^3)$

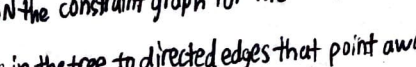
2) Ordering
   - more effective to compute next variable and corresponding value on the fly.
   1) MRV [minimum remaining values] — when selecting which variable to assign next, using an MRV policy chooses whichever unassigned variable has the fewest valid remaining values (most constrained variable). This is intuitive in the sense that the most constrained variable is most likely to run out of possible values and result in backtracking if left unassigned, so its best to assign a value to it sooner than later.
   2) LCV [least constraining value] — when selecting which value to assign next, a good policy to implement is to select the value that prunes the fewest values from the domains of the remaining unassigned values. Notably, this requires additional computation (e.g. rerunning arc consistency/forward checking or other filtering methods for each value to find the LCV), but still yield speed gains depend on usage.

Exploiting Structure of CSPs

- runtime from $O(d^n)$ to $O(nd^2)$

algorithm

1. First pick an arbitrary node in the constraint graph for the CSP to serve as the root of the ~~tree~~ tree
2. Convert all undirected edges in the tree to directed edges that point away from the root. Then linearize (topologically sort) the directed acyclic graph.

   ex) 

   $\to A \to B \to C \to D \to E \to F$

3) Perform a backwards pass of arc consistency. Iterating from $i = n$ to $i = 2$
4) Perform forward assignment, starting from X_1 to X_n, guarantees a correct sol.

* this algorithm can be extended to CSPs that are close to a tree structure, use cutset conditioning; remove that subset of variables that make it tree

2. Local Search Algorithm (Another CSP algorithm)

- start with a random assignment to values then iteratively select a random conflicted variable and reassign its value to one that violates fewest constraints til no more violations.
  - O(N) and high prob of success BUT
    1) incomplete
    2) suboptimal ⇒ won't necessarily converge and expensive on certain critical ratio: $R = \frac{\text{number constraints}}{\text{number variables}}$

Note 6: Propositional Logic

definitions:

valid: true in all models

satisfiable: atleast one model in which it is true

unsatisfiable: not true in all models

entailment: A entails B ($A \models B$) if in all models A is true, B is true:
models of A are subset of B

Theorems:
1) $(A \models B$ iff $A \Rightarrow B$ is valid)
2) $(A \models B$ iff $A \wedge \neg B$ is unsatisfiable)

DeMorgans
1) $\forall x \neg P \equiv \neg \exists x P$
2) $\neg \forall x P \equiv \exists x \neg P$
3) $\forall x P \equiv \neg \exists x \neg P$
4) $\exists x P \equiv \neg \forall x \neg P$

Logic Tricks
- $\vee$ and $\vee$ and $\wedge$ are associative!
- $A \Rightarrow B \equiv \neg A \vee B$
- $a \wedge (B \vee C) \equiv ((a \wedge B) \vee (a \wedge C))$
- $a \vee (B \wedge C) \equiv ((a \vee B) \wedge (a \vee C))$

Algorithms for showing $KB \models q$
1) simple model checking: enumerate all models $O(2^N)$
2) DPLL algorithm: backtracking with three improvements
   1) early termination
   2) pure symbols
   3) unit clauses
- CNF: $(P_1 \vee \cdots P_{i_1}) \wedge \cdots \wedge (P_j \vee \cdots \vee P_{i_n})$
3) Local Search (WALK-SAT)
   - randomly initialize all symbols and then flip some

Inferencing for showing $KB \models q$ (q is true in all models KB is true)
1. modus ponens: if KB contains $A \overset{A}{\&} A \Rightarrow B$, infer B
2. AND Elimination: $A \wedge B$ infer A, B
3. Resolution: A, B, infer $A \wedge B$
   - sound & complete

Note 7: Probabilistic Inference

Bayes Rule: $P[A|B] = P[A \wedge B]/P[B] = P[B|A]P[A]/P[B]$

- $P[C|B] = P[C|A \cap B]P[A|B] + P[C|\bar{A} \cap B)P[\bar{A}|B]$
- $P[\bar{B} \cap \bar{G}] = 1 - P[B \cup G]$

Total Probability Rule: $P[B] = P[A \wedge B] + P[\bar{A} \wedge B] = P[B|A]P[A] + P[B|\bar{A}](1 - P[A])$

Marginal Distribution:
- $P[X=a] = \sum_{b \in A} P[X=a, Y=b]$

Conditional Independence: $P(X|Y,Z) = P(X|Z)$, $P(Z,Y,X) = P(Z,Y)$

inference by enum:

$P(B|j,m) = \alpha P(B,j,m) = \alpha \sum_{e,a} P(B,e,a,j,m)$

$= \alpha \sum_{e,a} P(B)P(e)P(a|B,e)P(j|a)P(m|a)$

T/F
- arc consistency not guaranteed to reduce/guarantee backtracking in worst case! Still $O(dn)$
- set of values that remain doesn't depend on order in which arcs are processed from q.

Midterm 2 CS188

Note 6: Probabilistic Inference

Joint Distribution: captures prob. of every possible outcome
- given Joint PDF, we can trivially compute any desired $P(Q_1, \dots, Q_m | e_1, \dots, e_n)$ using slow, memory-heavy IBE
  - **Inference By Enumeration**: (IBE)
    - given Joint PDF, collect all rows consistent with observed evidence variables, sum over all the hidden variable, normalize

Bayes Nets
- Directed Acyclic Graph, small local conditional prob. tables
- Each node is conditionally independent of all its ancestors nodes in the graph given all of its parents (B)→(A)→(E) store B, A|B, E|A

- **Inference**
factors = unnormalized probability
1. Join (multiply together) all factors involving X: variable X elim
2. sum out X
- **Sampling**: (faster, efficient for prob. reasoning) count samples

1. Prior Sampling
- just naive sample, go down the bayesNet and sample

2. Rejection Sampling
- reject samples that disagree with query

3. Likelihood Weighting
- declare all variables equal to the evidence
- weight each sample; multiply by the probability each time we get to evidence variable

4. Gibbs Sampling
- first set all variables to totally random value
- repeatedly pick one variable at a time, clear value, resample it given the values currently assigned to all other variables
- pick a new variable from that conditional distributions
  * can calculate say $P(S|T, C, E)$ only using CPT's that connect S with neighbors

Note 8: Markov Models

Markov Models: time dependent, chain-like infinite Bayes Nets (W)→(W_1)→(W_2)→…
1. **Initial Distribution**: time=0, probability of each state $P(W_0)$
2. **Transition Model**: probability from one state to next: $P(W_{i+1} | W_i)$
- **Markov / Memoryless Property**: $t = i+1$ only dependent on $t = i$:
  - $\Pr(W_0, W_1, \dots W_n) = \Pr(W_0) \prod_{i=0}^{n-1} \Pr(W_{i+1} | W_i)$ (Joint PDF)

Calculating Distribution of weather on given day t

1. inefficient normal way
- compute joint distribution using Markov prop.
- marginalize / sum out. basically IBE: $O(d^t)$ if t vars, d vals.

2. Mini Forward Algorithm
- By marginalization, we know $\Pr(W_{i+1}) = \sum_{w_i} \Pr(W_i, W_{i+1})$
- iteratively compute $\Pr(W_0), \Pr(W_1) \dots \Pr(W_{i+1}) = \sum_{w_i} \Pr(W_{i+1} | W_i) \Pr(W_i)$

Stationary distribution: distribution that remains the same after the passage of time: $\Pr(W_{i+1}) = \Pr(W_i)$
$$\Pr(W_{i+1}) = \Pr(V_i) = \sum_{w_i} \Pr(W_{i+1} | W_i) \Pr(W_i)$$
$\Pr(W_{i+1})$ given
- solve system of eq.
$1 = \sum_{w_i} \Pr(w_i)$

Hidden Markov Models (HMM) - allows us to observe evidence at each timestep which can potentially affect belief dist. at each state (W0)→(W1)→(W2)→…, (W1)↓(F1), (W2)↓(F2)
- $F_i \perp\!\!\!\perp W_0 | W_i$, $\forall i = 2, \dots, n$, $W_i \perp\!\!\!\perp \{W_0, \dots, W_{i-2}, F_2, \dots, F_{i-1}\} | W_{i-1}$
- $\forall i = 2, \dots, n$; $F_i \perp\!\!\!\perp \{W_0, \dots, W_{i-1}, F_1, \dots, F_{i-1}\} | W_i$

Representation
1. initial distribution | 2. transition model | 3. sensor model

def: **belief distribution** at time i with all evi. $f_1, \dots, f_i$ observed:
$$B(W_i) = \Pr(W_i | f_1, \dots, f_i)$$
$$B'(W_i) = \Pr(W_i | f_1, \dots, f_{i-1})$$

Forward Algorithm (calculate Belief dist at given time i) $\theta(TS^2)$
1. Time Elapse Update - determining $B'(W_{i+1})$ from $B(W_i)$:
$$B'(W_{i+1}) = \sum_{w_i} \Pr(W_{i+1} | W_i) B(W_i)$$
2. Observation Update - determining $B(W_{i+1})$ from $B'(W_{i+1})$
$$B(W_{i+1}) = \Pr(f_{i+1} | W_{i+1}) B'(W_{i+1})$$

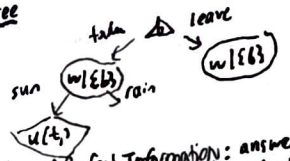
Particle Filtering Note 8 — $\theta(CT \cdot \#\text{particles})$
- sampling but for HMM
1. randomly initialize particles from distribution
2. Time Elapse Update:
   - for each particle in state t_i, update its state from prob. dist. given by $\Pr(T_{i+1} | t_i)$
3. Observation Update
   1) weight each particle with sensor model, $\Pr(F_i | T_i)$ — resampling bad if no evidence.
   2) sum all weight for each state
   3) normalize weights
   4) resample particles from this new distribution → gets you $B(T_{i+1})$ finalized

Note 7: Decision Networks

def: **Decision Network** is a combination of Bayes Nets & Expectimax: gives effect of various actions on utilities based on an overarching graphical probabilistic model

def: **MEU** (maximum expected utility)
1) instantiate all evidence that is known, run inference to calculate the posterior probabilities of all chance nodes parents of the utility node into which the action node feeds
2) go through each possible action and compute the expected utility of taking that action given the posterior probabilities computed in the previous step. The expected utility of taking an action a given evidence e and n chance nodes is computed with
$$EU(a|e) = \sum_{x_1 \dots x_n} P(x_1, \dots, x_n | e) U(a, x_1, \dots, x_n)$$
where each x_i represents a value that the i^{th} chance node can take on. Weights correspond to the probabilities of each outcome.
3) Finally select the action which yielded the highest utility to get the MEU.
ex) calculate optimal action, (should we leave or take umbrella) for weather:
$$EU(\text{leave}|\text{bad}) = \sum_w P(w|\text{bad}) U(\text{leave}, w)$$
$$EU(\text{take}|\text{bad}) = \sum_w P(w|\text{bad}) U(\text{take}, w)$$
$$MEU(F=\text{bad}) = \max_a EU(a|\text{bad})$$

Outcome Tree



def: **Value of Perfect Information**: answers is it worth it to invest in gaining new evidence. amount our max expected utility is expected to increase if we decide to observe new evidence.

General Formula:
$$MEU(e) = \max_a \sum_s P(s|e) U(s,a)$$
$$MEU(e, e') = \sum_{e'} P(e'|e) MEU(e, e')$$
$$MEU(\emptyset) = \max_a EU(a) = \max_a P(w) U(a, w)$$
$$MEU(e, e') = MEU(F) = \sum_f P(e'|e) MEU(e, e') = \sum_f P(F=f) MEU(F=f) = P(F=\text{good}) MEU(F=\text{good}) + P(F=\text{bad}) MEU(F=\text{bad})$$
$$VPI(E'|e) = MEU(e, E') - MEU(e)$$

VPI Properties
1) Nonnegative: $\forall E', e \; VPI(E'|e) \geq 0$
2) Nonadditivity: $VPI(E_j, E_k | e) \neq VPI(E_j | e) + VPI(E_k | e)$
3) Order-Independence: $VPI(E_j, E_k | e) = VPI(E_j | e) + VPI(E_k | e, E_j) = VPI(E_k | e) + VPI(E_j | e, E_k)$

Note 9: Machine Learning (Supervised Learning Algorithm (Bayes Structure))

def: **classification problem**: given datapoints, group them into one or more classes
- were also given training data and their classes

Naive Bayes Classifier
- assume feature F_i is independent of all other features given class label
- prediction $(F) = \arg\max_y P(Y = y) \prod_j P(F_j = f_j | Y = y_j)$ — remember $\theta = P(F_i = 1 | Y = \text{hgm})$

(Y)→(F_1), (F_2), (F_n)

How to get conditional probability tables used in our Bayes Net: **Parameter Estimation**
- given sample, youre MLE for outcome X that can take on $|X|$ values from sample size N is,
* $P_{MLE} = \text{count}(x) / N$

Laplace Smoothing w/ strength k: $P_{LAP,k}(x) = \frac{\text{count}(x) + k}{N + k|X|}$

similar with conditionals:
computing Laplace estimate for outcomes across diff. classes
$$P_{LAP,k}(x|y) = \frac{\text{count}(x, y) + k}{\text{count}(y) + k|X|}$$
$$P_{LAP,\infty}(x) = 1/|X|$$

example
1) $B(W_1 = \text{sun}) = \sum_{w_0} \Pr(W_1 = \text{sun} | w_0) B(w_0)$
   $B'(W_1 = \text{rain}) = \dots$ " " "
2) $B(W_1 = \text{sun}) = \Pr(F_1 = \text{good} | W_1 = \text{sun}) B'(W_1 = \text{sun})$
   Normalize after
∞) Iteratively compute up to $B(W_\infty)$ then normalize

Note 9: Machine Learning continued

Perceptron: Linear Classifiers

- doesn't estimate distribution, instead use linear combination of features called activations
- activation$_w(x) = \sum_i w_i f_i(x) = w^T f(x) = w \cdot f(x)$

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

Algorithm for Getting Perfect W

1. initialize all weights to 0. $w = 0$
2. For each training sample, with features $f(x)$ and true class label $y^* \in \{-1, +1\}$ do:
   (a) classify the sample using current weights, let y be the class predicted by your current w:
   $$y = \text{classify}(x) = \begin{cases} +1 & \text{if activation}_w(x) > 0 \\ -1 & < 0 \end{cases}$$
   (b) compared the predicted label y to the truth label y^*:
   - if $y = y^*$ do Nothing
   - otherwise if $y \neq y^*$, update weights $w \leftarrow w + y^* f(x)$
   - if you went through every training sample w/out updating weights, then terminate. Repeat step 2 (else)

Bias

- add feature to sample feature vector that is always 1.
- add extra weight for this feature to your weight vector.

$w^T f(x) + b = 0$.

Multiclass Perceptron

multiple weight vectors now. We predict it (sample) as the one with highest score.

update: - add feature vector to the weight vector for true class y^*
- subtract feature vector from weight vector corresponding to predicted class y

Note 10: Neural Networks

Non-linear Separators

Multilayer perceptron:

Theorem (Universal Function Approximators): A two-layer neural networks with a sufficient number of neurons can approximate any continuous functions to any desired accuracy.

Measuring Accuracy: the accuracy of the binary perceptron after making m predictions can be expressed as:

$$l^{acc}(w) = \frac{1}{m} \sum_{i=1}^{m} (\text{sgn}(w \cdot f(x^{(i)})) == y^{(i)})$$

where $x^{(i)}$ is datapoint i, w is our weight vector, f is our function that derives a feature vector from a raw datapoint, and $y^{(i)}$ is the actual class label of $x^{(i)}$. sgn(x) is indicator function which is 0 if x is negative, 1 if (+)

- # correct predictions / raw total # predictions

Loss Functions and Multivariate Optimization

gradient vector: $\nabla_w ll(w) = \left[\frac{\partial ll(w)}{\partial w_1}, \ldots, \frac{\partial ll(w)}{\partial w_n}\right]$

initialize weights w
For $i = 0, 1, 2 \ldots$
$w \leftarrow w + \alpha \nabla_w ll(w)$ ← changing over w each time with its derivative. want w to go to 0 with loss.
or (-) if minimize
- want stepsize α to be correct.

Logistic Regression Example

Given: $\hat{y} = h(x) = s(\sum_i w_i x_i) = 1/(1+\exp(-\sum_i w_i x_i))$ $\quad s(r) = 1/(1+\exp(-r))$

Loss func $L = -[y \ln h(x) + (1-y) \ln (1-h(x))]$

Problem: find dL/dw_i hint $(s'(r) = s(r)(1-s(r))$

Solution: Chain Rule/Backpropagation

given: $L = -[y \ln h(x) + (1-y) \ln(1-h(x))]$
$\hat{y} = h(x) = s(w^T x) = s(r) = 1/(1+e^{-r}) = 1/(1+e^{-w^T x})$

steps: $L = -[y \ln \hat{y} + (1-y) \ln(1-\hat{y})] \Rightarrow dL/d\hat{y} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$

$\hat{y} = s(r) = 1/(1+e^{-r}) \Rightarrow \frac{d\hat{y}}{dr} = \frac{ds(r)}{dr} = \hat{y}(1-\hat{y})$

$y = w^T x \Rightarrow \frac{dr}{dw_i} = \frac{d \sum_j w_j x_j}{dw_i} = x_i$

multiply:

$$\frac{dL}{dw_i} = \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{dr} \cdot \frac{dr}{dw_i} = \boxed{(-y + \hat{y}) x_i}$$

stochastic gradient descent update for w_i, w/ step size η

$$w_i = w_i - \eta \, dL/dw_i$$
$$= w_i - \eta(-y + \hat{y}) x_i = \boxed{w_i - \eta(h(x) - y) x_i}$$

Test Stuff

- IN IBE, the whole joint table is established: calculate the order from that
- CPT with 2 evidence, independent variables:
  - $P(X_{t+1} \mid D_1, \ldots, D_{t+1}, E_1, \ldots, E_{t+1}) = \frac{P(X_{t+1}, D_{t+1}, E_{t+1} \mid D_1, \ldots, D_t, E_1, \ldots, E_t)}{P(D_{t+1}, E_{t+1} \mid D_1, \ldots, D_t, E_1, \ldots, E_t)}$
  $= \frac{P(D_{t+1}, E_{t+1} \mid X_{t+1}, D_1, \ldots, D_t, E_1 \ldots E_t)}{P(D_{t+1}, E_{t+1} \mid D_{1:t}, E_{1:t})} \cdot (P(X_{t+1} \mid D_{1:t}, E_{1:t})$
  $= \frac{P(D_{t+1} \mid X_{t+1}) P(E_{t+1} \mid X_{t+1})}{P(D_{t+1}, E_{t+1} \mid D_{1:t}, E_{1:t})} \sum_x P(X_{t+1} \mid X_t = x, D_{1:t}, E_{1:t}) P(X_t = x \mid D_{1:t}, E_{1:t})$
  $= \frac{P(D_{t+1} \mid X_{t+1}) P(E_{t+1} \mid X_{t+1})}{P(D_{t+1}, E_{t+1} \mid D_1 \ldots D_t, E_1 \ldots E_t)} \sum_x P(X_{t+1} \mid X_t = x) P(X_t = x \mid D_{1:t}, E_{1:t})$

MDPs

- at convergence provide enough info to obtain optimal policy:
  - model based learning of T and R
  - Q-Learning to estimate Q(s,a)
  - NOT TD learning, or Direct Eval.

Q-Learning: converges to optimal Q-values for all states if:
- fixed policy taking actions uniformly at random
- AN ε greedy policy
- NOT greedy or fixed optimal policy

Probability

- $P(A \mid B, C, D) = \frac{P(A, B, C, D)}{P(B, C, D)} = \frac{\sim}{P(D \mid B, C) P(B \mid C)}$

A → B
↓
C
then $P(B, C) = \sum_a P(B \mid a) P(C \mid a) P(a)$

- given $P(A, B, C, D)$ joint table, can get $P(A, D)$ by $\sum_b \sum_c P(A, b, C, D)$
- use SUM and Product formulas/methods to play around.

DFS

- not complete

BFS

- complete

Test Stuff

HW8: Combining Factors Notes

- Joining over a variable: (Multiply)
  ex) given $P(A \mid C)$ and $P(B \mid A, C)$, join over C
  - $P(A \mid C) \cdot P(B \mid A, C) = P(A, B \mid C)$
- Joining and Summing over a variable: (Multiply and Sum)
  ex) given $P(A \mid B)$ and $P(B \mid C)$ and $P(C)$, join on C and sum over C
  1) $P(B \mid C) \cdot P(C) = P(B, C)$
  2) $\sum_c P(B, c) = P(B)$

  ex) given $P(A \mid C)$ and $P(B \mid A, C)$ join over A, sum over A
  1) $P(A \mid C) \cdot P(B \mid A, C) = P(A, B \mid C)$ 2) $\sum_a P(A, B \mid C) = P(B \mid C)$

- Joining over a variable formula
  1) take all variables that appear to the left in any factor and put these in the new factor
  2) take out all variables that are conditioned and did not appear on the left side and condition on these

RISK stuff

$U(EMV(L))$ vs. $U(L)$ so... lets say $U = x^2$
and $L = \frac{1}{3}(3)$ and $\frac{2}{3}(6)$
then $U(L) = \frac{1}{3}(3^2) + \frac{2}{3}(36) = 3 + 24 = 27$
$U(EMV(L)) = U(\frac{1}{3}(3) + \frac{2}{3}(6)) = U(5) = 25$

Risk prone: if second derivative is nonzero (+) then risk prone (-) then risk averse

Gradient Ascent Path

- orthogonal to contour, goes toward max
- Gradient Descent goes toward minimum.

CS188 Beginnings

Search Problems - Note 1

Completeness - if there exists a solution to the search problem, are we guaranteed to find it?
optimality - is the strategy optimal?
branching factor b - the increase in number of nodes on the fringe each time a fringe is dequeued and replaced with its children
- At depth k, (in the search tree), there are $O(b^k)$ nodes

Search Strategies

uninformed

Depth-First-Search
completeness: not complete, can be infinite loop if cycles exist
optimality: not optimal
time complexity: $O(b^m)$ where m = max depth
space complexity: $O(bm)$

Breadth-First-Search
completeness: complete
optimality: not optimal unless all edges are same cost.
time complexity: $1+b+b^2+\cdots+b^s = O(b^s)$ where s is our goal depth
space complexity: $O(b^s)$

Uniform Cost Search
- selects lowest cost fringe node from start for exploration
completeness: complete
optimality: optimal if all edge-costs are non-negative
time complexity: $O(b^{C^*/\varepsilon})$

informed

Greedy Search
- always selects fringe node with lowest heuristic value
completeness: not guaranteed
optimality: not guaranteed

A* Search
- selects lowest backward cost
completeness: complete
optimality: optimal

* IN Graph Search, we never expand a state twice. If we visit a node, we put it in a closed set to not visit again.
↳ ruins optimality, so we need consistency
* IN Tree Search, we can visit/expand same nodes repeatedly, but this leads to not always complete (inf. recursion), so we add the closed set condition which makes it a graph search

Some Neural Network functions

$f(x_1, x_2) = \max(x_1, -x_1) - 1$

$f(x_1, x_2) = x_2 - \max(x_1, x_1)$ or $x_2 - \max(x_1, x_1)$ OR $-\max(x_1, -x_2, x - x_2)$

MDP Questions

- When asking for Transition function and reward function, give $T(s,a,s')$ and $R(s,a,s')$ for as many possibilities
- MDP cannot be solved with expectimax search because self-loops = infinite tree
- Q-Learning can converge to optimal policy even if actions are suboptimal exploration: caveat is that you need to explore enough and learning rate small enough and not decrease too fast.
- TD learning learns values for a particular policy (on-policy)

Gradient Loss Func for Approximate Q-Learning

- suppose $L_1 = (y - \sum_k w_k f_k(x))$ then $\frac{dL_1}{dw_m} = \begin{cases} -f_m(x) \text{ if } y - \sum_k w_k f_k(x) > 0 \\ f_m(x) \text{ if } y - \sum_k w_k f_k(x) < 0 \end{cases}$

$w_m \leftarrow w_m - \alpha \frac{\partial L_1}{\partial w_m}$

- default update: we have $Q(s,a) = \sum_i w_i f_i(s,a)$

$L_2 = \frac{1}{2}(y - \sum_k w_k f_k(w))^2$; $\frac{dL_2}{dw_m} = -(y - \sum_k w_k f_k(x)) f_m(x)$ where

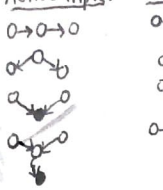
$y = r + \gamma \max_{a'} Q(s', a')$

so $w_m \leftarrow w_m + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s,a)) f_m(s,a)$

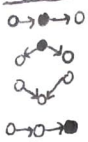
CS188 Test Stuff

D-Separation for Bayes Nets

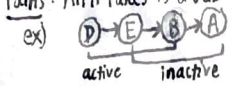
Active Triples



Inactive Triples



Paths: All it takes is a subset of a path to be inactive to be inactive

ex) 

but D→A is inactive because of just one inactive

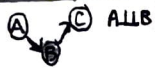
Conditional Independence: all paths (undirected) connecting the two variables need to be inactive

D-Separation Algorithm

given query $X_i \perp\!\!\!\perp X_j \mid \{X_{k_1}, \ldots, X_{k_n}\}$

1) shade all evidence nodes
2) check all undirected paths from X_i and X_j
   - exhaust all paths to check that all paths are inactive

Quick Bayes Nets Independence



A ⊥⊥ B | C

A ⊥⊥ E

Classification Problems

- Naïve Bayes can classify every dataset that perceptron can
- Naive Bayes
  - linear
  - circular?

  - CANNOT solve

Perceptrons

- guaranteed to learn separating decision boundary for separable dataset
- not guaranteed to find max-margin separating hyperplane

Loss functions

- must be differentiable (continuous range and well defined gradients)

ex) negative log-probability of correct digit

HMM updates

Standard HMM Elapse Time Update

$$P(X_t \mid e_{1:t-1}) = \sum_{x_{t-1}} P(X_t \mid x_{t-1}) P(x_{t-1} \mid e_{1:t-1})$$

observation update

$$P(X_t \mid e_{1:t}) \propto P(X_t \mid e_{1:t-1}) P(e_t \mid X_t)$$

More Probability!

$$P(+s \mid -o, -w, +e, +h) = \frac{P(+s, -o, -w, +e, +h)}{P(+s, -o, -w, +e, +h) + P(-s, -o, -w, +e, +h)}$$

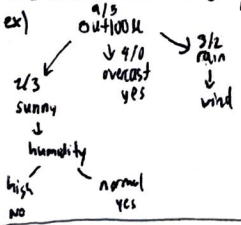
* Power of Bayes Nets: always know Joint Probability
* can always convert conditional probability with Joint Prob. stuff

VPI Questions

- if U(S,A), check to see if any VPI query have/relate to S and A. If they don't, VPI = 0.
- The more directly the query variables are related to S, A, the larger their VPI,
- check independence.

Decision Trees

- Suppose given training set data, divide the data by each feature and make a tree sorting by the label.

ex) 

depth: by how many features you use to classify into pure subsets

greedy splitting: choose things that (features) that split more, so 9/10.

Value iteration vs. Policy iteration

V.i: $O(|S|^2|A|)$

P.i: $O(|S|^3)$

Bayes Nets Inference

- once you have final factors, say $f_2(+d, +a)$ and $P(A)$, and you want to calculate $P(+a \mid +d)$, you just multiply and normalize.

ex) $$P(+a \mid +d) = \frac{f_2(+d, +a) \cdot P(+a)}{\sum_a P(a) f_2(a, +d)}$$

Neural Network Representation / Graph Questions

1) if the graph doesn't go through the origin, add, or the function must have the bias term b.
2) if a graph is strictly one line (straight), can't be a function with one ReLU. Must be a 2-Dimensional ReLU layer.
3) if graph is two lines connected, then must have 2-D relu layer
4) A 2D relu can do 2 slope changes

MDP Questions

- When they ask to define a game as an MDP, give ~~storage~~ space
  1) state space
  2) Transition model T(state, action, state')
  3) Reward Function

Epsilon Greedy

- if there are 3 available actions and one optimal one, there are these probabilities:
  - optimal one: $(1-\epsilon) + \epsilon/3$
  - two: $\epsilon/3$
  - three: $\epsilon/3$