Reinforcement Learning

Def: Use a MDP: set of states, set of actions, transition model, reward function and γ. Don't know transition or reward functions (learn them).

Passive Learning Input is a policy, execute and learn the transitions and rewards.

Passive (Model): Count outcomes s' for each s, a, normalize to get transition model X. Discover $R(s, a, s')$ when we experience (s, a, s').

Passive (Model Free): Similar to the above but instead of calculating $E[X]$ from $P(X)$, it is calculated from the average of all of the samples.

Sample Evaluation: Execute policy n times from state s and observe reward given.

let $sample_i = R(s, \pi(s), s'_i) + \gamma V_k^\pi(s'_i)$ then $V_{k+1}^\pi \leftarrow \frac{1}{n} \sum_i^n sample_i$

Temporal Difference Learning: Update $V(s)$ each time we experience transition (s, a, s', r). Let $sample = r + \gamma V^\pi(s')$ then $V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + (\alpha)sample$

Active Reinforcement Learning: Learner makes choices and learns optimal policy.

Q-Learning: Learn $Q(s, a)$ as you go (off-policy learning). Receive a sample (s, a, s', r) let $sample = r + \gamma \max_{a'} Q(s', a')$ and update $Q(s, a) \leftarrow (1-\alpha)Q(s, a) + (\alpha)sample$

Epsilon Greedy: With probability ϵ act randomly, otherwise act on current policy.

Exploration Functions: $f(\mu, n)$ Takes a value estimate, μ, and visit count, n, and returns optimistic utility. Change $sample = r + \gamma \max_{a'} f(Q(s', a'), N(s', a'))$

Regret: Δ between your expected rewards while learning and the optimal rewards.

Approximate Q-Learning: Represent states with features and weights. $Q(s, a) = \sum_i w_i f_i(s, a)$ Update with $\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$ and $Q(s, a) \leftarrow Q(s, a) + (\alpha)\delta$ and $w_i \leftarrow w_i + (\alpha)\delta f_i(s, a)$. Online least squares (adjust weights of active features).

Policy Search: Learn policies that maximize rewards, not the values that predict them by hill climbing. (nudge feature weights up and down to see if improvement).

Probability

Rules: Product: $P(y)P(x|y) = P(x, y)$ Chain: $P(x_{1:n}) = \prod_i P(x_i|x_{1:i-1})$

Bayes': $P(x|y) = \frac{P(y|x)P(x)}{P(y)}$ Independence: $x \perp\!\!\!\perp y \leftrightarrow P(x|y) = P(x)P(y)$

Conditional Independence: $X \perp\!\!\!\perp Y|Z$ iff $\forall x, y, z : P(x, y|z) = P(x|z)P(y|z)$

Markov Models

Def: Value of X at a given time is called the state.

Independence: $X_{t+1} \perp\!\!\!\perp X_{1:t-1}|X_t \implies P(X_{1:T}) = P(X_1) \prod_{t=2}^{T} P(X_t|X_{t-1})$

Mini-Forward: $P(X_t = x_t) = \sum_{x_{t-1}} P(x_{t-1}, x_t) = \sum_{x_{t-1}} P(x_t|x_{t-1})P(x_{t-1})$

HMMs: Observable evidence variables E_n for each X_n

$P(X_{1:n}, E_{1:n}) = P(X_1)P(E_1|X_1) \prod_{i=2} P(X_i|X_{i-1})P(E_i|X_i)$

Computations: Filtering $P(X_t|e_{1:t})$, Smoothing $P(X_t|e_{1:n}), n > t$, and Most Probable Explanation $x^*_{1:n} = \text{argmax}_{x_{1:n}} P(x_{1:n}|e_{1:n})$

Online Belief Updates: Time: $P(x_t|e_{1:t-1}) = \sum_{-x_{t-1}} P(X_t|X_{t-1})P(X_{t-1}|e_{1:t-1})$ Observation: $P(X_t|e_{1:t}) = P(e_t|X_t)P(X_t|e_{1:t-1})$ Update belief for state X_t based on the previous evidence, then update on e_t

Forward Algorithm: Combine both updates into a single one (time and observation). $P(x_t, e_{1:t}) = P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1})P(x_{t-1}, e_{1:t-1})$ Time: $O(|X|^2)$ Space: $O(|X|)$

Particle Filtering: Track samples of X, not all of the values. Elapse Time: $x' = sample(P(X'|x))$, Observe: $w(x) = P(e|x)$, Resample (w/ weights and w/ replacement)

Bayes' Nets

Dynamic Bayes Nets: Abstraction of HMM, with multiple variables and inputs at once.

Probabilities: $P(x_{1:n}) = \prod_{i=1} P(x_i|parents(X_i))$ Every BN represents a join distribution, but not every distribution can be represented by a specific BN. Size: d^n where d is the dimension and n is the number of variables.

Causality: Topology may encode structure but it really encodes conditional independence.

Independence in BN: Given nodes $A, B \quad A \perp\!\!\!\perp B$ iff $\forall$ undirected paths between A and B, there is an inactive triple in every path.

Inference by Enumeration: Create one giant CPT with every single variable, and sum over the relevant conditions.

Variable Elimination (VE):

(c) Using ordering M, E, A because it had less size:

$f_1(A) = \sum_M P(M|A)$

	$f_1(A)$
$+a$	$0.7 + 0.3 = \mathbf{1}$
$-a$	$0.01 + 0.99 = \mathbf{1}$

$f_2(A, B) = \sum_E P(A|B, E)P(E)$

	$f_2(B, A)$
$+b, +a$	$(.95)(.002) + (.94)(.998) = \mathbf{.94002}$
$+b, -a$	$(.05)(.002) + (.06)(.998) = \mathbf{.05998}$
$-b, +a$	$(.29)(.002) + (.001)(.998) = \mathbf{.001578}$
$-b, -a$	$(.71)(.002) + (.999)(.998) = \mathbf{.998422}$

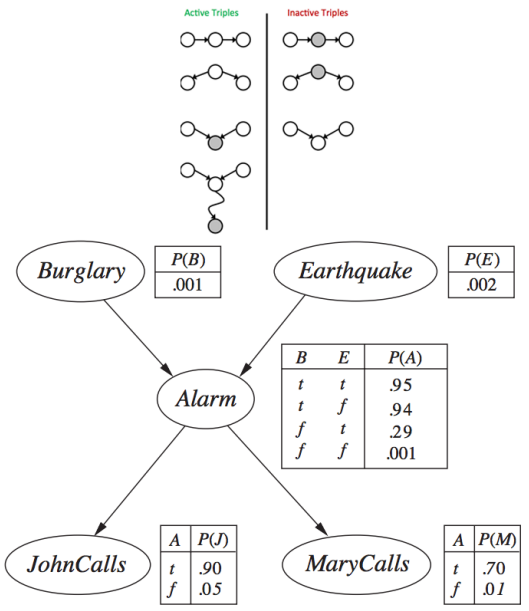
$f_3(B, +j) = \sum_A P(+j|A)f_1(A)f_2(A, B)$

	$f_3(B, +j)$
$+b, +j$	$(0.9)(0.94002) + (0.05)(0.05998) = \mathbf{0.849017}$
$-b, +j$	$(0.9)(0.001578) + (0.05)(0.998422) = \mathbf{0.0513413}$

$P(B, +j) = P(B)f_3(B, +j)$

	$P(B)f_3(B, +j)$
$+b$	$(0.001)(0.849017) = \mathbf{0.000849017}$
$-b$	$(0.999)(0.0513413) = \mathbf{0.0512899587}$

We can now normalize to get the probability $P(B = +b| + j) = \mathbf{0.01628}$



VE Cont.: NP-Hard, determined by size of largest factor (not always an efficient ordering).

Polytree A directed graph with no undirected cycles (efficient ordering for VE). Choose a set of variables such that if removed only a polytree remains.

Expectation: The expected outcome of the distribution given the observed data (the most likely). Requires uniform priors.

Maximum A Posteriori: The best hypothesis that fits observed data. Arbitrary prior.

Bayesian Estimate: Weighted combination of possibilities. Arbitrary Prior.

Prior: Binary variable (Beta(α, β)), discrete variable (Dirichlet)

Naive Bayes: Features are independent given class: $P(X_{1:n}|Y) = \prod_i P(X_i|Y)$

Laplace Smoothing: Pretend you saw every outcome K (1) more than you actually did.

Expectation Maximization: Randomly initialize K distributions. Compute the likelihood for each sample to be in each distribution and assign it to the most likely. Compute the MLE for the parameters of each distributions given the samples in it. Repeat.

Learning Structure: Search through the space of possible network structures. For each structure, learn the parameters and pick the one that fits observed data best. Use local search (to converge quickly) and add penalty term (regularization) proportional to model complexity.

Bayesian Information Criterion: $score = P(Data|BN) - penalty$ where $penalty = \alpha 0.5 * (|parameters|) * \log(|Data|)$. Never test with training data (k-fold cross validation).

Markov Decision Processes

Def: states, actions, Transition $T(s,a,s')$ or $P(s'|s,a)$, Reward $R(s,a,s')$ or $R(s)$
Policy: $\pi^* : S \to A$, gives an optimal action for all states
Discounting: Each time a level is descended multiply by another factor of γ
Bellman Equations: $V^*(s) = \max_a Q^*(s,a)$ = utility starting in s and then acting optimally, $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$ = utility starting after taking action a in s then acting optimally
Value Iteration: $V_0(s) = 0$, given $V_k(s)$ simultaneously solve one ply of expectimax from each state: $V_{k+1}(s) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k^*(s')]$ with Time: $O(|S|^2|A|)$
Policy Extraction: Given the optimal values, $V^*(s)$, $\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$
Problems: Slow, max at each state rarely changes and the policy often converges long before the values do
Asynchronous Value Iteration: Not essential to back up all states in each iteration as long as no state is starved. Backup with a heap of states ordered by the expected change in value (prefer backing a state whose successors had most change
Policy Evaluation: compute the utility of a state s under a fixed policy **Policy Iteration:** Initialize $\pi(s)$ to random actions, calculate utilities of π at each s using a nested loop, update policy using one-step look-ahead to see what's the best action I could execute, assuming I then follow $\pi(s)$

Initialize $\pi_i(s)$ to random actions and $i = 0$. Repeat:
Step 1: Policy Evaluation:
Initialize $k = 0$ and for each s, $V_0^\pi(s) = 0$ and repeat until V^π converges:
For each s $V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s,\pi_i(s),s')[R(s,\pi_i(s),s') + \gamma V_k^{\pi_i}(s')]$
$k++$
Step 2: Policy Improvement:
For each s, $\pi_{i+1}(s) = \operatorname{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$
If $\pi_i == \pi_{i+1}$, then it is optimal
Else let $i++$

Adversarial Search

Minimax: The opitmal utility with a rational adversary, Time: O(b^m), Space: O(bm)
Pruning: α is Max's best choise on a path to root. If the best value becomes worse than α, no point in exploring children. Similiar for β. Time: O($b^{m/2}$)

```
def value(state, agent, alpha, beta):
    for each successor of state:
        if agent == 0:
            v = max(v, value(successor, agent + 1, alpha, beta))
            if v >= beta: return v
            alpha = max(alpha, v)
        else:
            v = min(v, value(successor, agent + 1, alpha, beta))
            if v <= alpha: return v
            beta = min(beta, v)
    return v
```

Expectimax: Instead of the min value being chosen, the expected value for that state is returned: $\sum_i [value(successor_i) * P(successor_i)]$
Utility: Function from state to real numbers that describe an agent's preferences

Agents

Rational: Maximally achieving goals (actions that maximize utility function)
Reflex Based: Chooses action based on current percept (no future consideration)
Goal Based: Chooses action based on consequences (model of how the world reacts)
Utility Based: Goal based with trading off of multiple goals and uses probabilities

Constraint Satisfaction Problems (CSPs)

Def: Goal test is a set of constraints over the state's variables $x_i \in D_i$ *or* D
Forward Checking: Cross off values that violate a constraint when added to the existing assignment (Immediate neighbors and fail if the set of possible values is empty)
Arc Consistency: An arc $X \to Y$ is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constraint
Constraint Propagation: If X loses a value, neighbors of X need to be rechecked
Min Remaining Values: Choose the variable with fewest legal values in its domain
Max Degree: Choose the variable in the most constraints with remaining variables
Least Constraining Value: Given a variable, assign a value that rules out the fewest values in remaining variables

Stochastic Search

Hill Climbing++ Restarts: Generate random state when plateaued
Hill Climbing++ Walk: With prob p move to the neighbor with largest value, with $(1-p)$ move to a random neighbor
Hill Climbing++ (Both): Greedy move, random walk, or random restart
Simulated Annealing: Pick a random neighbor and calculate the change in 'energy' or objective function δ, if it is positive then move to that state. Otherwise, move to this state with probability $e^{\frac{\delta}{T}}$ where T is decreased as the algorithm runs longer. High T = greater P(bad)

Search

Def: Possible states, Successor function $f(n) \to (n'$, action, cost), start and goal state
Complete: Guaranteed to find a solution if one exists
Optimal: Guaranteed to find the least cost path
Properties: n= number of states, b= maximum branching factor, C^*= optimal cost, d= depth of shallowest solution, m= max depth, ϵ = min cost of all actions
Conformant Planning: Set of actions that always work (sterilizing surgical gear)

Blind Search

DFS: Fringe uses a Stack, complete iff finite, not optimal, time: O(b^m), space: O(bm)
BFS: Fringe uses a Queue, complete, optimal (constant), time and space: O(b^d)
IDDFS: Fringe uses a Stack, complete, optimal (constant), time: O(b^d), space: O(bm)

Heuristic Search

Admissible: Always an underestimate to the true lowest cost
Consistent: Always $h(n) <= h(n') + stepCost(n')$ where n' is a neighbor of n
Best First: Fringe uses a PriorityQueue with cost fuction $f(n)$ for each node
Uniform Cost: Best First with $f(n)$ = sum of edge costs from start to n (explores increasing contours), complete, optimal, time and space: O($b^{\frac{C^*}{\epsilon}}$)
Greedy: Best First with $f(n) = h(n)$ (suboptimal goal is common)
A*: Best First with $f(n) = g(n) + h(n)$ with $g(n)$ = sum of costs from start to n
IDA*: Depth bound is now $F_{limit} = h(start)$, prune if $f(n) > F_{limit}$,
$F'_{limit} = min(pruned\ nodes)$, uses space of DFS, time depends on # of unique F values
Beam: Best First with $|Fringe| = K$, not complete, time: O(b^d), space: O($b + K$)
Hill Climbing: Always choose best child (Beam Search with K = 1)
Tabu: Keep fixed length queue of states to not visit again (use with hill climbing)