How to most likely pass CS 486

Jim Zhang

2018 Spring

1 Introduction

Intelligence is the capacity to think and learn. The **Newell-Simon hypothesis** says that a physical symbol system is necessary and sufficient for intelligence. The dual dichotomy: We might want a system that can [think|act] [like humans|rationally]. The **Turing test** says that if a human can't distinguish a machine from a human, the machine is intelligent.

2 Problem-solving

2.1 Examples

n-queens, crosswords, sliding puzzles, river crossing, propositional satisfiability, partition problem, TSP, set cover

2.2 By search

Formulate your problem as search on a graph of states. You have a start state and you want to reach a goal state. BFS, DFS, greedy, and A* are all search algorithms.

2.2.1 A*

An **admissible** A* heuristic doesn't overestimate the remaining cost to the goal. A **consistent** A* heuristic never decreases total cost with a hop. Admissible or consistent heuristics always find optimal paths.

An admissible A* heuristic **dominates** another if its estimates are always at least as pessimistic, and at least one estimate is more pessimistic; using a dominating heuristic never makes you search more nodes.

The time complexity of A* is

$$O(b^{\varepsilon d})$$

where ε is the maximum relative error $\max_n \frac{h^*(n)-h(n)}{h^*(n)}$, b is the branching factor, and d is the depth of the goal node.

Iterative-deepening A* saves memory by doing a complete DFS with the next worst heuristic value each iteration.

2.3 Constraint satisfaction

A CSP is a set of variables x_i with domains $\text{dom}(x_i)$ and constraints C_j on those variables. A constraint can be intensional (use propositions), extensionally (write out a table of all admissible variable assignments), or globally (alldifferent).

2.3.1 Arc consistency

A constraint is **arc consistent** if it's possible to assign any value to any variable and still satisfy the constraint. (Every value for every variable has domain support in the constraint.)

You can make a CSP arc consistent by repeatedly removing unsupported values in a domain of a variable.

2.4 Backtracking search

Depth-first search of the search tree. **Constraint propagation** is removing inconsistent values in variable domains during the search.

Types of backtracking algorithms: **Naïve backtracking** is doing no constraint propagation. **Forward checking** maintains arc consistency on all constraints with a single uninstantiated variable. **Maintaining arc consistency** maintains arc consistency on all constraints.

Variable ordering: You typically want to instantiate variables that are likely to fail first. **dom**: Pick variable with smallest domain size. **dom/deg**: Smallest domain size / degree.

2.5 Local search

Using a state definition, cost function, and neighborhood function, search a state's neighborhood until you can't find anything better.

Can vary stopping criteria, initial solution (random or "good"), neighborhood function (small vs. large), neighbor to move to (first-improvement or best-improvement), threshold accepting (accept diminishingly worse cost;

simulated annealing; tabu search), multiple starts, multiple levels (different neighborhoods).

An **exact neighborhood**'s local optima are global optima. TSP has no polynomially searchable exact neighborhood unless $P = NP$. A non-exact neighborhood's local optimum can be arbitrarily far from a global optimum and local search can take an exponential number of steps to find a local optimum. All that said, local search TSP algorithms are pretty good.

GSAT is a local search algorithm for propositional satisfiability where the cost is the number of unsatisfied clauses and the neighborhood is "flip a variable".

2.6 Genetic algorithms

You have functions for fitness, crossover and mutation and evolve a population of N individuals with parents weighted by fitness.

3 Knowledge, reasoning, and decision making

3.1 Probability

$$P(X,Y) = P(X|Y)P(Y) \quad \text{(Product rule)}$$

$$P(X) = \sum_{y \in dom(Y)} P(X|Y=y)P(Y=y) \quad \text{(Sum rule)}$$

$$P(Y|X) = P(X|Y)P(Y)/P(X) \quad \text{(Bayes' rule)}$$

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|X_{i-1}, ..., X_1) \quad \text{(Chain rule)}$$

3.2 Bayesian networks

A DAG where nodes are random variables, edges are direct influences, and nodes are annotated with conditional probability tables. Independence assumptions made: every node is independent of its non-descendants given its parents.

3.2.1 Types of inference

Diagnostic: P(cause|effect). **Causal**: P(effect|cause). **Intercausal**: P(cause1|cause2,effect). **Mixed**: Combined.

3.2.2 Three-layer model

You want your **root causes** pointing to **events** pointing to **tests and reports**.

3.2.3 Modeling time

Dynamic systems have one variable per discrete time slice and full future causality. **Markov chains** have directly observable states and causality only goes one forward. A **hidden markov model** has hidden states that you can only observe through observation variables.

Inference types in hidden markov models are **monitoring** (probability of states given current observations), **prediction** (probability of future states given current observations), **hindsight** (probability of previous states given observations), and **most probable explanation** (the most probable past state that given observations).

Dynamic Bayesian networks are also things; HMMs are just DBNs with one state variable and one observation variable.

3.3 Decision networks

It's Bayesian networks with decisions (rectangles) and a utility function (diamond). **Value of information** is increase in expected utility given new information.

3.3.1 Utility theory

States are capital letters. $[p, A; 1-p, B]$ is a lottery with outcome A with probability p and outcome B with probability $1-p$. $A > B$ means outcome A is preferred to B. $A \sim B$ is indifferent between A and B. Axioms are:

$$(A > B) \vee (A < B) \vee (A \sim B) \quad \text{(Orderability)}$$

$$(A > B) \wedge (B > C) \Rightarrow (A > C) \quad \text{(Transitivity)}$$

$$A > B > C \Rightarrow \exists p [p, A; a-p, C] \sim B \quad \text{(Continuity)}$$

$$A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C] \quad \text{(Monotonicity)}$$

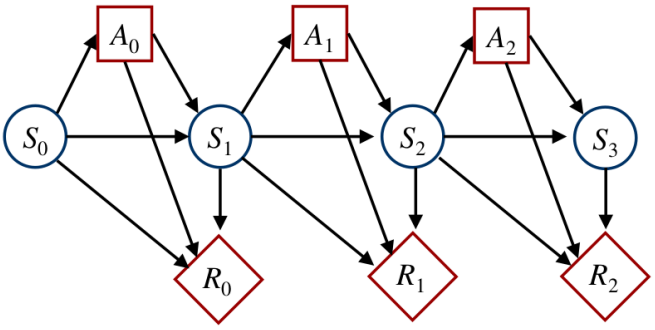
$$C := [p, A; 1-p, B] \text{ is allowed} \quad \text{(Decomposability)}$$

and if axiom are obeyed, then a consistent real-valued utility function exists.

3.4 Sequential decision making

It's where you have a sequence of decisions to make. Horizons can be indefinite or infinite. A Markov decision process (MDP) is a set of states S, a set of actions A, a transition model $P(S_t|A_{t-1}, S_{t-1})$, a reward model $R(S_t, A_{t-1}, S_{t-1})$, a discount factor $0 \leq \gamma \leq 1$, and a horizon h, and your goal as usual is to find an optimal policy and this seems quite hard.

The decision network for an MDP looks like this.



Define discounted overall utility as $U(S_0, ...) = \sum_t \gamma^t R(S_t, A_{t-1}, S_{t-1})$.

Partially-observable MDPs (POMDPs) are also things, where your states aren't fully observable.

3.5 Multiagent systems

Sometimes there are multiple agents and they have to take other agents into account. As an agent, your goal is to find a strategy that maximizes your payoff. A **pure** strategy means you always do one thing. A **stochastic/mixed** strategy has a distribution over actions.

A **strategy profile** σ is an assignment of a strategy to each agent. A **best response** for an agent to other agents with set strategies is the strategy for that agent that maximizes expected utility.

A **Nash equilibrium** is a strategy profile where each agent's strategy is the best response to the other agents' strategies. You can compute Nash equilibria by iterated elimination of dominated strategies (for pure equilibria) or with calculus (for mixed equilibria).

A **Pareto optimal** outcome cannot be strictly improved on by any other outcome.

A strategy **strictly dominates** another strategy if it's better for every other possible strategy profile for other agents.

4 Learning

Supervised learning is learning to predict a label given labelled data. **Unsupervised learning** just has data and you have to find structure in it.

Data can be real-valued or discrete (categorical/nominal, ordinal), and so can outcome (distinguishes **regression** from **classification**). Error on a data set is the sum of individual errors on each example, where the individual errors can be one-hot (for classification) or L_1 norm (abs diff) or L_2 norm (squared diff).

Don't overfit.

4.1 Clustering

A kind of unsupervised learning where you try to determine labels for unlabelled data. **Hard clustering** means classes are definitive; **soft clustering** means classes can be mixed.

You typically cluster with respect to some optimization criteria; for example, sum of distance to mean within cluster. Examples of distance: **Manhattan** (sum of differences along dimensions), **Euclidean** (sqrt of sum of squares of differences along dimensions), **Cosine similarity** (cosine of angle between vectors).

4.1.1 k-means clustering

Initialize k random means. Then, assign each example to the nearest mean, adjust the means based on the clusters, and repeat until done.

This can be used to cluster images quite convincingly (e.g. to create minimalist art, or to generate a color scheme from an image). Use Euclidean distance between color vectors. Results can be better if you convert to a different color space (e.g. CIELAB often better than RGB for perceptual uniformity).

4.2 Learning Bayesian networks

Four versions, from simplest to hardest:

1. Naïve Bayes, learn probability tables
2. Fixed structure, learn probability tables
3. Fixed nodes, learn arcs and probability tables
4. Node for attributes and classes, learn hidden nodes, arcs, and probabilities

4.2.1 Naïve Bayes

You just have your class point to all the attributes. This can be poor; can augment with m-estimate ($\frac{n_a+mp}{n+m}$).

4.2.2 Learning structure

Use a scoring function of the form $Score(G) = \sum Score(x_i, Parents(x_i))$. Some possible scoring functions: BIC, MDL, BDeu. Choose parent sets for each variable to minimize this score. (We did this for A1)

4.3 Decision trees

ID3: Repeatedly split the tree using the best feature. Best can be measured using information content

$$I(p) = \sum -p_i \log_2(p_i)$$

4.3.1 Extensions

Numeric attributes: Discretize them, or split on a numeric split point (however this can result in huge trees, many tests of the numeric attribute)

Missing values: When constructing decision tree: Use other instances to estimate missing attribute using simple majority, or create fractional examples. When using decision tree: Pretend example has all values of attribute, follow all possible branches, recombine answers by weighing.

Large discrete attributes: To prevent early split on this attribute, pick attribute of maximal gain ratio instead of gain

Attributes with costs: Pick attribute that maximizes $Gain^2/Cost$

Multiclass: Can adapt ID3

Overfitting: Stop growing the tree early using Chi-squared test, or post-prune the tree using validation set

4.4 Neural networks

A neuron's output is the thresholded weighted sum of its inputs. Thresholding functions include Heaviside step, sigmoid $(1/1 + e^{-x})$, hyperbolic $(\tanh x)$, ReLU $(\max(0, x))$, softplus $(\log(1+e^x))$. Can put together lots of these in interesting ways to fit very complex functions, then train them efficiently using backpropagation (super-high dimensional gradient descent).

4.4.1 Deep neural networks

Vanishing gradients: First few layers of a deep neural network learn super slowly with most thresholding functions. Solutions include pre-training, or using ReLU and maxout units.

Overfitting: Happens a lot more because huge number of parameters. Can fix using regularization, dropout, or data augmentation.

4.4.2 Evaluation

Split your labelled data set into a **training set** (that you will use to train your network) and a **test set** (that you will use to evaluate its performance). Cannot use test set in learning process in any way or there will be test set leakage; a "test-ish" set for the learning process has to be split off from the training set and is called a **validation set**.

With a limited amount of data, you can use **random resampling** (random training/test partitions), **cross validation** (iteratively leave out one group for test set and train on remaining groups; often $k = 10$).

Accuracy may not be a good performance measure if classes are imbalanced. Other accuracy measures are recall $(TP/(TP + FN)$; trues that are positive) and precision $(TP/(TP + FP)$; positives that are true).

4.4.3 Ensembles

Use many classifiers to vote on the answer. Best results if classifier errors are uncorrelated.

Can construct ensembles using bagging, cross-validated committees, boosting, or injecting randomness.

Bagging is learning multiple times with different training subsets. 63.2% is a good proportion to use.

Cross-validated committees is learning multiple times with cross-validation training subsets (drop a different one out each time).

Boosting is learning multiple times with different probability distributions over training examples, where subsequent distributions place more weight on misclassified examples.

Injecting randomness is exactly what it sounds like, and can be used in neural networks (random initial weights) or decision trees (pick from among top few features to split on).

Classifiers can be combined by unweighted voting, weighted voting (better classifiers' votes are worth more), or stacking (learn a classifier on the votes).

4.4.4 Overall steps for supervised ML

1. Choose (and possibly synthesize) good features
2. Collect data
3. Filter not-so-helpful features out
4. Select classifier and set parameters
5. Evaluate classifier

4.5 Reinforcement learning

It's a Markov decision process with unknown transition and reward models. It's interesting because the agent needs to guess which action resulted in a reward, balance exploration with exploitation, and learn throughout its entire lifetime. **Passive learning** is evaluating a fixed policy; **active learning** is determining a good policy.

Passive learning can be done using **adaptive dynamic programming** (ADP) and **temporal difference** (TD).

Active learning can be done using Q-learning.

Can decide between exploration and exploitation using ε-greedy (random action with probability ε, otherwise best action) or by Boltzmann exploration.

5 Other topics

5.1 Natural language

It's hard. Levels of analysis are **prosody**, **phonology**, **morphology**, **syntax**, **semantics**, and **pragmatics**. Plenty of ambiguity at every level, plenty of context dependence.