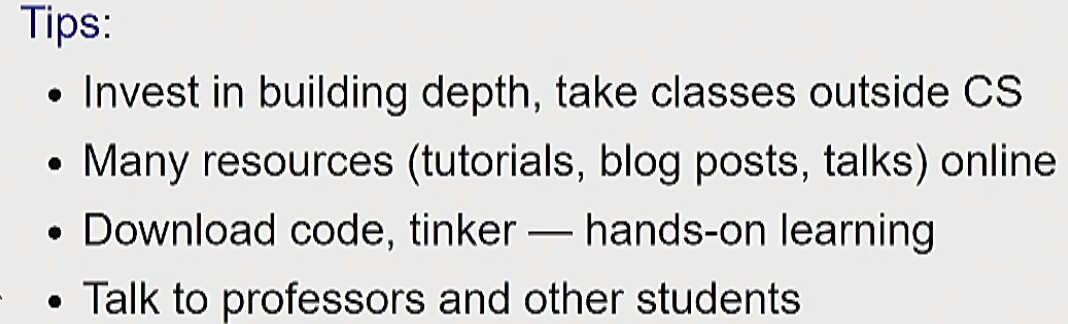
I NEVER READ THE QUESTION CORRECTLY. BRING A HIGHLIGHTER FOR GOODNESS SAKE, UNDERLINE, RESTATE THE PROBLEM BEFORE DIVING IN.I seem to miss important details in the question: like 1/3, max{0, expression}



The field is empirically driven. Try what works and then figure out why it works.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | ML: Loss minimisation  State based: recurrences. Order matters.  Variables: reduction to factors and solving. Order doesn’t matter.  Logic: Represent complex facts compactly | | | | | | | | | | | |
| Reflexive | | | | Models: linear models, neural networks, nearest neighbours  Inference: feed-forward  Learning: SGD, k-means | | | | | | | | | | | |
| Machine learning: define loss, minimise loss via stochastic gradient descent (SGD) which is now standard | | | | | | | | | | | | | | | |
| Hypothesis class: the space of allowed predictors  Predictor:  Weight vector (also called the parameters of the model): **w**  Features: x simply represent what properties might be useful for prediction  Feature extractor (also called feature map): ϕ, which takes *x* and converts it into the feature vector.  Hyper-parameters: design decisions (hypothesis class, training objective, optimisation algorithm) that need to be made before running the learning algo  Residual: diff between predicted f**w**(x) vs. actual output *y*  Loss function with respect to **w** (on per example/data point basis): measures how badly the function f screwed up on that example | | | | | | | | | | | | | | | |
| Algorithm: SGD | | | | | | | | | | | | | | | |
| Gradient: The gradient of TrainLoss(**w**) is the direction that increase the training loss the most. Gradients are like derivatives so it is linear and gradient of sum is the sum of the gradient. We can also apply the chain rule.  Training loss (also known as training error): simply the average of the per-example losses. The best predictor is the one that minimises training loss.  Stochastic gradient descent: update **w** after each example. Large step size faster training but less stability, smaller step size = slower training but more stability  Group distributional robust optimisation: maximum group loss != average loss. Minimise maximum group loss. | | | | | | | | | | | | | | | |
| Classification | | | | | | | | | | | | | | | |
| Binary classifier  Predicted label = sign(w. ϕ (x))  Score = **w**. ϕ (x) how confident are we in predicting +1?  Margin = **w**. ϕ (x).y how correct we are  Hinge loss:  Logistic loss: no matter how correct your prediction is you will have non-zero loss so there is an incentive (although diminishing one) to push the loss down by increasing the margin. | | | | | | | |  | | | | | | | |
| Non-linear features: purpose to fit more complex data. Key takeaway: non-linear predictors with linear machinery. | | | | | | | | | | | | | | | |
| Features: simply represent what properties might be useful for prediction. We can get non-linear predictors just by changing ϕ if x is non-linear! You can even get piecewise constant predictors! If feature not useful then weight assigned as 0 for that feature.  More features: makes learning harder because need to eval weight for each feature.  Hypothesis class of quadratic predictors is superset of hypothesis class of linear predictors. | | | | | | | |  | | | | | | | |
| Feature template: a group of features all computed in a similar way | | | | | | | | | | | | | | | |
| Note people often use the word feature when they really mean feature template.  Two feature vector implementations: arrays good for dense, dictionary good for sparse | | | | | | | | | |  | | | | | |
| Neural nets: ‘automated feature extractors’ | | | | | | | | | | | | | | | |
| NNs are a way to build complex nonlinear predictors without creating a large number of complex feature extractors by hand. It’s a common misconception that NNs are more expressive than other models but isn’t necessarily true. NNs yield non-linear predictors in a more compact way. You may not need O(d2) features to represent desired non-linear predictor.  NNs can’t deal with zero gradients, so we replace the threshold function with Activation functions with non-zero gradients. Logistic function σ (z). But doesn’t fix everything, still a problem when gradients are very small which makes optimising logistic function difficult. In 2012 ReLu works well enough and is the activation function of choice.  Choosing hyperparameters based on training error = bad because overfit, doesn’t include regularisation, trains forever.  Choosing hyperparameters that optimise test error is bad = unreliable estimate of a model’s error on unseen data.  Validation set is important to optimise hyperparams, which are chosen based on that which yields lowest error on validation set. Prevents over fitting on training and test set.  Controlling the norm   * Regularisation: adds an additional term to the objective function which penalises the norm(length) of W. This is probably the most common way to control the norm. Purpose is to keep the weights small and has the effect of keeping w close to the origin that it otherwise would be. Increasing regularisation weight has the effect of reducing the hypothesis class by forcing the algo to pick weights from a small set that is closer to zero. * Early stopping: | | | | | | | |  | | | | | | | |
| Computational graphs: DAG whose root node represents the final math expression and each not represents intermediate sub-expressions | | | | | | | | | | | | | | | |
|  | | | | | | | |  | | | | | | | |
| Generalisation theory: machine learning is not about minimising loss. Key idea is to minimise error on unseen future examples | | | | | | | | | | | | | | | |
| Key idea: what matters it the size of the hypothesis class, not how ‘complex’ the predictors in the hypothesis class look.  Approximation vs. estimation error trade off: increase hypothesis class means you can get closer to magical f\* but that increases estimation error as hypothesis class size increases.  How to control hypothesis class size to reduce estimation error?   1. Reduce dimensionality d (number of features) . Manually drop unhelpful features. Automatic feature selection: forward selection, boosting, L1 regularisation 2. L2 Regularisation: add a term to penalise the length of **w**. Note: SVMs are exactly hinge loss + L2 regularisation. 3. Early stopping | | | | | | | | | |  | | | | | |
|  | | | | | | | | | |  | | | | | |
| Unsupervised: clustering, PCA | | | | | | | | | |  | | | | | |
| State-Based Models | | | | | Models: search problems vs. MDP (stochastic version of search), games  Objective: minimum cost paths vs. maximum value policies  Inference (graph-based): UCS/A\*, DP vs. value iteration, minimax  Inference (tree-based): backtracking vs. minimax/expectimax  Learning: structured perception, Q-Learning, TD learning | | | | | | | | | | |
| Search | | | | | | | | | |  | | | | | |
| Search is powerful iff well understood world states and actions. So key skill is defining and decomposing problems into states.  State is a summary of past actions sufficient to choose future actions optimally. State collapses tree into having only info that we use to choose future actions optimally to avoid the exponential blow ups  Search problem is an abstraction that provides a clean interface to the world to find optimality  Model local interactions. Use Inferenceto find globally optimal solutions | | | | | | | | | | Definition of search problem   1. Sstart Start state 2. Action(s): possible actions from states 3. Cost(s,a): cost of the action from state s 4. Succ(s,a): successor state from state s given action a 5. IsEnd(s): reached end state? | | | | | |
| **1. Tree search**   * Enumerating all states and actions but not done in practice so we build algorithms to help instantiate a search tree. * Tree search is memory efficiency but exponential time complexity | | | | | | | | | | **2. DP**  DFS with reuse | | | | | **3. UCS** The analog of DP for BFS |
| **1a. Exhaustive**  If b actions per state and max depth of D | **1b. DFS**  Key idea is backtracking search plus terminate when it finds first end state | **1c. BFS**  Key idea: explore all nodes in order of increasing depth  Space > DFS because @ lowest level have to remember the queue of nodes to explore | | | | **1d. DFS-ID** | | | | Definition of DP   1. Recurse 2. If already computed for s, return cached answer   Effect: recasts tree search problem as a DAG. If not DAG then DP breaks! | | | | | 1. Expand sates close to the start 2. Use past-cost to re-use computation   Key idea: UCS enumerates sates in order of increasing past cost.  Implementation diff: UCS start to end, Djikstras all nodes |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Exhaustive | DFS | BFS | DFS-ID | | Edge Cost | Any | 0 **Strict**! | ≥ 0 | ≥ 0 | | Time | O(bD) | O(bD) | O(bd) | O(bd) | | Space | O(D) which is small | O(D) | O(bD) > DFS | O(d) | | | | | | | | | | | |  |  |  | | --- | --- | --- | |  | DP | UCS | | Cycles | No. **Strict**! | Yes | | Edge cost | Any | ≥ 0 | | Time | O(n) | O(nlogn) | | Space | ? | ? | | | | | | |
| **A\*** Key idea is to distort edge costs to favour certain end states  Heuristic h(s) is any estimate of FutureCost(s). If h(s) =0 means I’m on the optimal path.  Ideal: explore in order of PastCost(s) + FutureCost(s)  A\*: explore in order of PastCost(s) + h(s). Practically, just modify the edge cost of UCS so you don’t have to modify UCS code: Cost’(s,a) = cost(s,a) + h(succ(s,a)) –h(s)  A heuristic h is consistent if: cost’(s,a) = cost(s,a) + h(succ(s,a))-h(s) >= 0, h(send)=0. In practice this means modified UCS edge costs are non-negative (the main property).  State ordering: is the sequence of visited nodes. Recall A\* is modified UCS so from start state, needs to do BFS at depth=1 to evaluate the initial options. | | | | | | | | | |  | | | | | |
| **Learning: Structured perceptron for search** | | | | | | | | | |  | | | | | |
| The structured perception can find such costs | | | | | | | | | |  | | | | | |
| Modelling: MDP | | | | | | | | | |  | | | |  | |
| Relationship to search. Many similarities but one main one and one minue one. Major is the transition probabilities minor is changed from minimising cost to maximising rewards.   1. succ(s,a) becomes T(s,a,s’). Transition distribution for each state, s is . Introduced via the chance nodes. 2. cost(s,a) becomes Reward(s,a,s’). Reward may be positive or negative 3. isEnd(s) also includes conditions when you lose the game     Definitions   * Policy: A mapping from each state s \in States to an action a \in Actions * Utility: Following a policy yields a random path. The utility of a policy is the discounted sum of the rewards on the path. This is a random variable. * Path: s0 → a1,r1,s1 → a2,r2,s2 … (action, reward, new state) * Utility: r1 + r2 + r3 + … * Value: The value of a policy at a state is the expected utility. Also known as expected utility | | | | | | | | | |  | | | | | |
| Modelling | | | | | | | | | |  | | | | | |
| Value: the expected utility received by **following** policy pi from state s  Q-value: expected utility of **taking action a** from state s following policy pi.  What is a solution for MDP?   * Policy * Utility   Policy evaluation Vpi(s) | | | | | | | | | | You can solve recursion algebraically by hand | | | | | |
| Learning: for MDPs | | | | | | | | | |  | | | |  | |
| Reinforcement learning is online whereas MDP is offline. MDP requires a mental model of how the world works and then we find a policy to collect maximum rewards. RL doesn’t know how the world works. Perform actions and collect rewards.    Model-Based Monte Carlo: key idea is to estimate the MDP transitions and rewards by counting.  Model Free Monte Carlo: key idea is to try estimate Qopt(s,a) directly  SARSA: evaluates Q-value of a policy  Q-Learning:  Epsilon Greedy: exploitation vs. exploration. Can get Qopt using SARSA if we explore a lot to generate data then transition to exploiting the optimal actions after estimate of Q is more accurate.  Function Approximation | | | | | | | | | |  | | | | | |
| Modelling: Adversarial Games | | | | | | | | | |  | | | |  | |
|  | | | | | | | | | |  | | | | | |
| Minimax | | | | | | | | | |  | | | | | |
| Expectiminimax | | | | | | | | | |  | | | | | |
| Alpha beta pruning is a technique to speed-up search of minimax (exact).   1. Initialise alpha = -inf, beta = +inf. Traverse to first leaf. 2. Hit leaf node: update beta if minNode, update alpha if maxNode 3. Backtrack:    1. If Max node: update alpha = max(currentAlpha, prevAlpha, prevBeta)    2. If min node, update beta = min(currentBeta, prevAlpha, prevBeta) 4. Prune when alpha >= beta   Ordering: matters:  Evaluation functions use domain specific knowledge to compute approximate. Very weak estimate of the value of minimax. Analogy is FutureCost(s) in search problems. Pacman! | | | | | | | | | |  | | | | | |
| Variable-based models | | | Models: CSPs, Markov networks, Bayesian networks  Inference: backtracking, beam search, gibbs sampling, forward-backward, particle filtering  Learning: maximum likelihood (closed form, EM) | | | | | | | | | | | | |
| Modelling: CSPs: “The cool thing with CSPs is that we define local structures and run algorithms to find global scores!” | | | | | | | | | | | | | | | |
| Factor graphs: more general | | | | | | | | | |  | | | | | |
| Variables:  Values: Value is an element in the domain  Domain: Set of remaining values for each variable. Domain mutates whereas values don’t. It can reduce with each look ahead.  Assignment: lock in a value for the variable from the domain  Weight the product of all factors | | | | | | | | | |  | | | | | |
| CSP: Hard constraint | | | | | | | | | |  | | | | | |
| A CSP is a factor graph where all factors are constraints  Constraint is satisfied iff f(x) = 1  An assignment is consistent iff weight(x) = 1 (i.e. all constraints are satisfied)  Dependent factors D(x, Xi) returns a set of all the factors affecting Xi given the partial assignments x  Constraints/factor: scope  Constraints/factor: expression  Consistent s  Number of consistent values = num of values that satisfies factors  Most consistent var = var with most num of vals in domain  Least consistent variable = var with fewest num of vals in domain | | | | | | | | | |  | | | | | |
| How to find assignment? | | | | | | | | | |  | | | | | |
| Backtrack(x, w, Domain**s**): where x are the partial assignments, w are the weights, a Domain**s** is the set of domains for each remaining variable. Backtracking search is *exponential* time with number of variables n.  How to choose unassigned variable?   * Most constrained variable. Choose the variable that has the *fewest* consistent values i.e. which variable has the fewest remaining options left in their domain? Must assign every variable = fail early = more pruning. When to use? **When SOME factors are constraints.**   How to order values of domain of chosen Xi?   * Least constrained variable.Order values of selected variable by decreasing number of consistent values of neighbouring variables. Need to choose some value most likely to lead to a solution. **Use when ALL factors are constraints (CSPs).**   How to update domain?   * Arc consistency: eliminate values from domain = reduce branching. * Forward checking: Figure out what values would violate the factors and remove them from the domain of the set for that particular neighbour. Limitations: If constraints are far from each other or rely on many variables no good because it allows you to exit early out of incompatible scenarios. If domain empty for a variable then terminate early * AC3 is extreme look ahead by enforcing arc consistency on neighbours and their neighbours etc. Can solve large problems. Limitation: only looks at pairwise constraints (local structure) not global structures. | | | | | | | | | | This notations is about updating the partial assignment of x with extra variables. | | | | | |
| Modelling you get very different asymptotic complexities depending on how you model | | | | | | | | | | | | | | | |
| Must convert n-ary constraint into binary or unary constraint to run look ahead functions. Think about every CSP as a binary constraint problem. Technique is to add more dimensions to the CSP.  Why? CSP is used to impose logical constraints on probabilistic inferences from ML. Why? Classification decisions are generally related. Local classifiers output evidence (unary factors), binary factors enforce the outputs to be consistent**.** | | | | | | | | | |  | | | | | |
| Search: two methods. Extend partial assignments or modify complete assignments | | | | | | | | | | | | | | | |
| **Extend partial assignments**  Beam search Considers K candidates of partial assignments rather than one partial assignment. Beam size K controls tradeoff between efficiency and accuracy (most important algos for language. Used by Google translate to hold K num of best translation solutions). Extends partial assignments   * Exhaustive, K = inf is BFS so tie O(bn). **Greedy** **K=1**: not global optimum but fast. DFS only one path for one partial assignment. K=1. Time = O(nb) * n variables (depth), branching factor: b = domain len, beam size = K, Time = O(nKblogK)   **Modify complete assignments**  Local search: Iterative Conditional Modes (ICM). Not guaranteed to find optimal, when stuck then stuck forever! Exploits key idea is to exploit locality structure of graphs: When evaluating possible re-assignments to Xi only need to consider the factors that depend on Xi.    Gibbs sampling (probabilistic). ICM but random. Unlike ICM which returns the value of the assignment, GS returns the normalised distribution. The use of randomness helps get out of local minimum by bumping you out of a greedy pick. Think of stochastic exploration.  Initialise all assignments. For each variable, evaluate assignment based on probability while holding all the other variables fixed. Count each assignment. You can make direct inferences about anything in the entire MN after running gibbs sampling because we’ve eventually converged to the right answer (spring final 2021) | | | | | | | | | | | | |  | | |
| Conditional independence makes running CSPs search algorithms easier because fewer constraints  Conditioning: goal try to disconnect graph. By picking a value for a variable and then removing that variable from the graph and add new factor that accounts for new factors. Why? Accelerates algorithms. Subproblems are much smallers structures such that you can brute force. *“this technique makes markov network and CSPs in real life tractable”*  Markov blanket of a node A in a Factor Graph: what is the minimum set of nodes to condition on to disconnect A from the graph? | | | | | | | | | | | | | Independence. Let A and B be a partitioning of variables X. A and B are independent if there are no edges between A and B.  Conditioning: To condition on variable X =v, consider all factors f1, …, fk that depend on Xi. Remove Xi and f1,…,fk. Add g(x) = fj(xU{Xi=v}) for j=1,…,k  Conditional independence: Let A, B, C be a partitioning of variables. A and B are conditionally independent given C if conditioning on C produces a graph in which A and B are independent. In other words, every path from A to B goes through C.  Markov blanket of a BN = all its parents and children, and all the parents of its children. What to condition on to make A conditionally independent of the rest. Why? Its purpose is to find minimal set without losing useful info for feature extraction. | | |
| Hints from section   * Valid assignment = multiplication of all factors is non-zero * find probability of CSP= multiplication of all factors / Z, where z is the sum of all factors for all values for all variables. * State the arity of each factor: the number of variables as parameters into the factor function * Factors are always functions of the variables! F(X1, X2) * Domains can add a **null** to domain reflect no choice. Remember that scheduling appended [0] to list?   Which CSP technique will guarantee the maximum weight solution? Least constrained value, most constrained variable, iterated conditional modes, backtracking search. Answer: Backtracking only and MCV. | | | | | | | | | | | | | * Express logical or as indicator functions: 1[ 1[expression a] + 1[expression b] != 0] * ‘each artist must be matched to a gallery’ is encoded as an n-ary indicator factor: j(G1,…,Gn) = 1[{A1,…,Am} subset {G1,…,Gn}] | | |
| **Bayesian Networks. Why? Handle heterogeneously missing info, incorporate prior knowledge, interpret all intermediate variables. Precursor to causal models** | | | | | | | | | | | | | | | |
| ELI5: BN = world model. Inference is asking a query using that world model. How to create BN? Requires domain knowledge.  BNs are a special case of factor graph – BNs use the language of factor graphs. BNs need to be reduced to a markov network (with normalisation constant Z=1). Unlike Markov Networks have arbitrary factors to represent a set of preferences. BN have local conditional probabilities. BNs allow us to define a joint probability over many variables e.g P(C,A,H,I) by specifying local conditional probabilities e.g p(i|a).  Markov models are limiting because they don’t have a way of talking about noisy sensor readings. Hence Hidden Markov Models are its extension. HMMs are a special case of BNs.  Unlike HMM, BNs are directed and acyclic. BNs are a generative story (arrows have direction). It evolves states (Not necessarily).  Unlike NN, BN’s vertices and edges have meaning. The structure describes conditional dependence. In NNs structures does not tell you anything. BN > NN for explainability – you can interrogate the conditional probabilities.  Unlike NN, BNs goes from outputs to inputs. NNs goes from inputs to outputs | | | | | | | | If two cause variables are an OR conditions for an effect, it’s more likely that only one of the causes occurred than both causes occuring at the same time. In otherwords, one cause probabilty actually decreases in this case P(B=1) | | | | | | | |
| Fundamentals: it’s important to think about an assignment to random variables as capturing the state of the world | | | | | | | | | | | | | | | |
| Joint distribution: probabilty for each assignment to state of the world. Upper case to denote the random variable. Lower case to denote value P(S=s, R=r), which is a number whereas P(S,R) is a distribution (table of probabilities).  Marginal distribution: Pick subset of variables we care about (query variables) and induce a distribution over that. E.g. P(S,R,T,A) but query is P(R=1 | T=1, A=1) means S is marginalised out.  Conditional distribution: **YOU MUST NORMALISE**  Local conditional probability:  Conditional independence⫫ If checking A B independent, does knowing B tell me something about A? vice versa because it’s birectional. It’s sufficient to ask one way.  If checking A⫫B | C: ‘Given C, does the extra knowledge of knowing B tell me something new about A and vice versa? Be careful of v-structures, and inverted v-structures.  Probabilistic Inference definition: P(R=query| conditions )  Bayes rule P(A|B) = P(B|A)P(A) / P(B). In other words, the probability of my query is the same as the probability of my query given the evidence for all valuesof the | | | | | | | | | | | |  | | | |
| Markov network: Markov networks connect factor graphs with probability. • The connection is very natural: factor graphs already provide a way of specifying non-negative weights over assignments, which gets us most of the way there. We then normalize the weights to make them sum to 1 to get a probability distribution. • Once we have a joint probability distribution, we can compute marginal probabilities of individual (or subsets of) variables. We can compare CSPs with Markov networks. Variables become random variables, which means that they have probabilities associated with them. Instead of weights, we have their normalized versions, a.k.a., probabilities. The big difference is that instead of focusing on just finding the maximum weight assignment, which might be not representative of the full set of possibilities, the goal is to look at marginal probabilities | | | | | | | | |  | | | | | | |
| BN definition: Let X = (X\_1, … , X\_N), BN is a DAG that specifies joint distribution over X as a product of local conditional distributions, one for each node. | | | | | | | | |  | | | | | | |
| HMM definition: HMM goal is to infer what the hidden object/state is given observable evidence. | | | | | | | | | End-to-End HMM | | | | | | |
| **Probabalistic Inference on generic bayesian networks** | | | | | | | | |  | | | | | | |
| Inference by converting BNs to markov networks: ‘it’s important to remember that a single factor connects all parents. There isn’t one per arrow - this is a common mistake. We run any inference algo for markov networks (e.g. gibbs sampling) on this so called markov network. But one important thing is missing, which is the ability to condition on the evidence. | | | | | | | | | If Query is on node Q. Can ignore leaf Z but can’t ignore parents X and Y. When converted to MN get 2 unary, 2 binary, 1 trinary | | | | | | |
| **Probabilitistic inference on HMMs:** three techniques to ask two types of queries: filtering (conditioned on evidence up to that point and can’t see the future e.g. object tracking) and smoothing (conditioned on all evidence, when you have collected all the data).  What is the belief distribution of position given our observations? | | | | | | | | | | | | | | | |
| Forward-backward for HMMs (**exact**). Use DP to exhaustively enumerate all assignments. First convert HMM to lattice representation. Smoothing question = the nomalised version of Si = the weighted fraction of paths pathing through a given node.    Particle filtering query  for HMMs (**approx)**. Exhaustive search is expensive.Akin to beam search but random sampling. **Very lightweight, practical algorithm because it only depends on the last timestep.** Partial assignement because each particle = guess of current state.  1) Propose (transitions): given currParitialAssignments get transition probability because it’s fast  2) Weight (emisions) old particles based on evidence: w(hi) = ∏P(ej|hi) for all emissions 1,…j at hi. Weights are assigned to each particle as a tuple pair.  3) Resample with replacement kills off particles with low weights, then particles behave as if they were sampled from the distribution we care about p(H=h| latest evidence). Reallocate the *same number of particles* at the higher weighted locations.      Gibbs sampling (GS)(**general**, **approx**): GS isn’t practical for HMMs because it  GS works best when variables are independent from each other. GS can only update variables by a small amount so takes a long time for convergence. HMMs are the case where variables are highly dependent on each other.  For GS to work on BNs, need to initialise the first hiddden variable in the BN either as a fixed value or specify a prior distribution P(B1) from which to sample.  PF: Linear wrt to particle counts. Set that to be at least as big as the domain size. Having more particles is always better.  GS: slow to converge | | | | | | | | | | |  | | | | |
| Learning Where do the local conditional probabilities come from? These local conditional distributions are the parameters of the BN.  Three learning techniques. Surprisingly for BNs, learning in fully-supervised setting is easier than inference. Parameter sharing. | | | | | | | | | | | | | | | |
| **Supervised:** where training data is complete assignments. Just need naively *count* and *normalise* = Maximum likelihood technique.  **Laplace Smoothing:** mechanism to guard against overfitting. Involves small tweak to existing algorithms Just add lambda to each possible value whether observed or not.  **Unsupervised with EM:** This is the ONLY algorithm that can handle missing data setting where training data consists of partial assignments. The workhorse is the EM algorithm which breaks the problem into probabilistic inference + supervised algorithm.   * Estep: evaluate the expected joint distributions for the missing variable. Then evaluate the P(missing variable|evidence) * Mstep: add the weighted number of times that param was used and recount instances with halllucinated weight and then normalize to get probabilities. | | | | | | |  | | | | | | | | |
| Logic | | | | | | |  | | | | | | |  | |
| Propositional logic | | | | | | |  | | | | | | |  | |
| Reinterpret implications as CNF  **Conjunctive Normal Form**. Is a re-expression with conjunctions (ands). (v …v) ∧ (v… v)… we want the conjunctions.Modus Ponens isn’t permitted. Rewrite P→Q into ¬P v Q  **De morgans’ law:** ¬(¬want ∧ ¬ invite). If you have negation outside an expression with you can push negation into it (want v invite) and flip the inside operator  Inference  **Resolution rule. Get rid of p and** ¬P and join the two statements. It is a single inference rule that operates on CNFs.  **Prove f is entailed KB via resolution**  **Equivalent to proving KB U** ¬f. Add ¬f to the KB then use resolution to prove contradiction. | | | | | | | KB entails f means that f didn’t add any more info. It is already known from KB. f is a superset of KB. No satisfying assignment.    Tricky question: if asked to test this, first write f as first order logic, then CNF into propositional, then look for most similar statement in KB then try prove equivalence by rearranging it. | | | | | | | Resolution tree | |
| First order logic | | | | | | |  | | | | | | |  | |
| Has predicates, variables and quantifiers.   * If ∀ then comes with implication * If ∃ then comes with conjunction or disjunction * If you see a verb then it is a predicate         De morgans’ law flips ∀ into ∃ and vice versa  Propositionisation: converts first order logic into propositional logic by converting infinite predictes into instances, otherwise intractable.   * ∀ becomes a lot of conjunctions     CNFs permitted by ignoring the ∀ and ∃  To check if new statement changes size of KB need to check if it is equivalent to an existing statement in KB. If so then adds nothing. If distinct then adds something.  Resolution doesn’t work for first order logic | | | | | | |  | | | | | | | | |