CS221 - Indented Outline (Notes 1-18)

Build: v0.2 • Date: 2026-01-13

Contents

CS221 - AI Principles and Techniques

Indented outline of Lectures 1, 5-13 (set 1 of your PDF notes)

This document is a cleaned-up, hierarchical outline meant for fast review. It preserves the lecture structure but emphasizes definitions, core algorithms, and the "why" behind assumptions.

How to use

- Skim headings first; then drill into the nested bullets for details.
- For algorithms, memorize: (i) assumptions, (ii) ordering/invariant, (iii) termination condition.
- If you want, we can extend this into a full course reader with worked examples and derivations.

Contents (this batch)

Lecture 1: Overview

What AI is, how the field got here, and the modeling-inference-learning paradigm.

Motivation and scope

- AI success stories exist, but benchmark performance does not always translate to real-world robustness.
- Two philosophical views
  - AI agents: building intelligent agents with perception, reasoning, learning.
  - AI tools: methods that solve problems and benefit society (often in non-human-like ways).

A brief history (very compressed)

- 1956 Dartmouth workshop and early optimism; rapid progress claims were premature.
- AI winters and shifts
  - Overreliance on brittle symbolic reasoning and limited compute/data contributed to setbacks.
  - Knowledge-based expert systems succeeded in narrow domains but were hard to scale/maintain.
- Neural networks thread
  - Backpropagation enabled training multi-layer networks; deep learning resurgence with large data and GPUs.

Course organizing principle

- Modeling
  - Turn a messy real-world situation into a formal model (often lossy but analyzable).
  - Example: navigation as a graph with nodes, edges, and costs.
- Inference
  - Given a model, answer questions: shortest path, best action sequence, most probable explanation, etc.
- Learning
  - Fit unknown parameters of a model from data rather than hand-specifying everything.

Roadmap of model families (high level)

- State-based models: search problems, MDPs, adversarial games.
- Variable-based models: CSPs, Bayesian networks, logic.
- Machine learning: linear models, neural nets, and deep learning.

Lecture 5: Search I

Search problems and basic search algorithms; why future consequences matter.

Search problem abstraction

- Key components
  - Start state: s_start
  - Actions(s): available actions in state s
  - Succ(s, a): deterministic successor
  - Cost(s, a): step cost
  - IsEnd(s): goal test
- Goal: find a minimum-cost path (action sequence) from s_start to any end state.

Tree search intuition ("what if?")

- Represent possible action sequences as root-to-leaf paths in a search tree.
- Branching factor b and depth D lead to exponential time in worst case: O(b^D).

Backtracking search (optimal, but expensive)

- Recursively enumerate paths, updating best solution upon reaching an end state.
- Space: O(D) due to recursion stack; time: O(b^D) in worst case.

Special cases: DFS, BFS, and DFS with iterative deepening

- DFS (depth-first search)
  - Assumes effectively zero action costs; stops at first goal.
  - Space: O(D); time: O(b^D) worst case.
- BFS (breadth-first search)
  - Assumes constant step cost; explores by increasing depth.
  - Finds shortest-length solution; space can blow up: O(b^d) where d is solution depth.
- DFS-ID (iterative deepening)
  - Runs depth-limited DFS with depth limits 1, 2, ..., d.
  - Retains O(d) space like DFS, but achieves BFS optimality under constant step cost.

Dynamic programming view (preview)

- FutureCost recurrence
  - FutureCost(s) = 0 if IsEnd(s)
  - FutureCost(s) = min_a [Cost(s,a) + FutureCost(Succ(s,a))] otherwise
- Key idea: define a state so that all relevant future costs depend only on the current state (not the full history).

Lecture 6: Search II

Uniform cost search, A*, and learning action costs via structured prediction ideas.

Dynamic programming recap

- DP works cleanly when the induced state graph is acyclic (topological ordering exists).
- If there are cycles, we need a different state-ordering mechanism.

Uniform cost search (UCS) / Dijkstra

- Assumption
  - All step costs are non-negative: Cost(s,a) >= 0.
- High-level strategy
  - Maintain: explored (finalized), frontier (seen but not finalized), unexplored.
  - Pop frontier state with smallest PastCost; relax/update its successors.
  - Stop when an end state is popped (moved to explored).
- Invariant (why it works)
  - When a state is popped from frontier, its priority equals the true minimum PastCost to that state.

DP vs UCS trade-off

- DP: handles negative costs but requires acyclic structure; explores all reachable states.
- UCS: handles cycles but requires non-negative costs; explores states in increasing PastCost, often fewer than DP.

A* search (UCS + heuristic)

- Idea
  - UCS prioritizes by PastCost(s) only; A* biases exploration using an estimate of remaining cost h(s).
  - Target ordering: PastCost(s) + h(s) (h approximates FutureCost(s)).
- Heuristic conditions (practical)
  - Avoid heuristics that create negative modified edge costs or violate consistency; otherwise optimality can fail.

Learning costs as an inverse problem (structured perceptron)

- Setup (simplified)
  - Assume Cost(s,a) = w[a] (or generalize to w · phi(s,a)).
  - Given training pairs (x, y_true), infer w so that y_true is the minimum-cost path.
- Update intuition
  - Decrease costs of actions on y_true and increase costs of actions on predicted y_hat (the argmin path under current w).

Lecture 7: MDPs I

Markov decision processes: modeling sequential decisions under uncertainty.

From search to MDPs

- Search: deterministic successors; MDPs: stochastic transitions and long-run utilities.
- Motivation: planning when actions have probabilistic outcomes.

MDP definition

- Components
  - States S, actions A(s)
  - Transition model: T(s,a,s') = P(s' | s,a)
  - Reward function: R(s,a,s') (or R(s,a))
  - Discount factor gamma in [0,1) for infinite-horizon problems
  - Terminal states (optional) with no outgoing actions
- Policy pi(s): maps states to actions (deterministic) or distributions (stochastic).

Value functions

- State value
  - V^pi(s): expected discounted return starting from s following pi.
- Action value
  - Q^pi(s,a): expected discounted return starting from (s,a) then following pi.

Bellman equations

- Policy evaluation
  - V^pi(s) = sum_{s'} T(s,pi(s),s') [R(s,pi(s),s') + gamma V^pi(s')]
- Optimality
  - V*(s) = max_a sum_{s'} T(s,a,s') [R(s,a,s') + gamma V*(s')]
  - pi*(s) = argmax_a ...

Planning algorithms

- Value iteration
  - Iteratively apply Bellman optimality backup until values converge.
  - Extract greedy policy with respect to V.
- Policy iteration (conceptual)
  - Alternate: (1) evaluate current policy, (2) improve policy greedily, repeat.

Lecture 8: MDPs II

Reinforcement learning: planning and control when the model is unknown.

Problem setting

- In RL, transitions T and rewards R are unknown; we learn from experience (samples).
- Two broad families
  - Model-based: estimate T/R then plan (e.g., via value iteration on the estimated model).
  - Model-free: learn values/policies directly from experience.

Exploration vs exploitation

- Agent must explore to learn (reduce uncertainty), but also exploit current knowledge to gain reward.
- Typical approach: epsilon-greedy or other stochastic policies during learning.

Model-free value learning

- Monte Carlo (MC)
  - Learn V or Q from complete episode returns; unbiased but high variance and needs episodes to finish.
- Temporal-difference (TD)
  - Bootstrap using current estimates; updates from partial trajectories.
  - TD(0) update idea: V(s) <- V(s) + alpha [r + gamma V(s') - V(s)].
- SARSA (on-policy)
  - Q(s,a) <- Q(s,a) + alpha [r + gamma Q(s',a') - Q(s,a)].
- Q-learning (off-policy)
  - Q(s,a) <- Q(s,a) + alpha [r + gamma max_{a'} Q(s',a') - Q(s,a)].

Function approximation (preview)

- When state spaces are large/continuous, represent V/Q with parameterized functions (linear or neural).
- Policy gradient methods optimize a parameterized stochastic policy directly.

Lecture 9: Games I

Adversarial search in deterministic, turn-taking, zero-sum games.

Game trees and utilities

- States: board configurations; actions: legal moves; terminal utilities define outcomes.
- Assume rational opponent; objective: maximize your utility (often - opponent utility).

Minimax

- Recursive definition
  - Max nodes choose action maximizing child value.
  - Min nodes choose action minimizing child value (opponent).
- Computes optimal play in deterministic perfect-information games (given full tree).

Depth limits and evaluation functions

- Full game trees are huge; use cutoff depth D and an evaluation function Eval(s) at leaves.
- Eval(s) approximates true utility from non-terminal positions.

Alpha-beta pruning

- Core idea
  - Keep bounds alpha (best for Max so far) and beta (best for Min so far).
  - Prune subtrees that cannot affect the final minimax decision.
- Same decision as minimax; can reduce effective branching dramatically with good move ordering.

Lecture 10: Games II

Stochastic games and learning from self-play; from minimax to TD methods.

Stochasticity and chance nodes

- Some games include random events (dice, card draws).
- Extend minimax to incorporate expectation over chance outcomes (expectiminimax).

Backgammon as a case study

- Large state space and stochasticity make exhaustive search infeasible.
- Historically important example: TD-Gammon used temporal-difference learning with neural networks.

Learning in games

- Self-play
  - Generate experience by playing against current or past versions of the agent.
  - Use bootstrapping targets derived from the agent's own value estimates.
- TD update (concept)
  - Update value estimates towards r + gamma V(s').
  - Comparable ideas show up in RL algorithms such as Q-learning.

Lecture 11: CSPs I

Constraint satisfaction problems: variables, domains, constraints; systematic search with propagation.

CSP definition

- Components
  - Variables X1..Xn
  - Domains Dom(Xi)
  - Constraints restricting joint assignments (binary or higher-order)
- Goal: find a complete assignment satisfying all constraints.

Backtracking search for CSPs

- Assign variables one by one; backtrack when any constraint is violated.
- Different from general search: cost is typically feasibility rather than path cost.

Variable and value ordering heuristics

- Choose next variable to assign to reduce branching (e.g., smallest remaining domain / most constrained).
- Choose value ordering to preserve flexibility (least-constraining value).

Constraint propagation

- Forward checking
  - After assigning Xi = v, remove inconsistent values from neighboring unassigned domains.
- Arc consistency (AC-3)
  - Enforce that for every value in one variable's domain, some compatible value exists in the neighbor's domain.
  - Iteratively revise arcs until no more domain reductions occur.

Scheduling as a canonical CSP application

- Variables: tasks/classes; domains: time slots; constraints: conflicts, capacities, prerequisites.

Lecture 12: CSPs II

More CSP inference and scaling tricks: conditioning and local search viewpoints.

Review: why inference matters

- Pure backtracking is exponential; propagation and problem structure can cut the search drastically.

Conditioning (case split) as an algorithmic pattern

- Pick a variable (or constraint) and branch on its possible values or cases.
- Each branch simplifies the remaining problem; combine results across branches.

Local search for CSPs

- Work with complete (possibly inconsistent) assignments and iteratively repair them.
- Typical moves: change one variable's value to reduce number/weight of violated constraints.
- Hill-climbing variants can escape local optima via randomness.

When to prefer which approach

- Systematic search + propagation: good for finding proofs of infeasibility or all solutions.
- Local search: often good for large, loosely constrained problems where a solution likely exists.

Lecture 13: Bayesian networks I

Probabilistic graphical models for reasoning under uncertainty.

Probability as a modeling language

- Random variables represent uncertain quantities; probability distributions quantify uncertainty.
- Bayes rule relates conditional probabilities via priors and likelihoods.

Bayesian networks (BNs)

- Definition
  - A directed acyclic graph over variables; edges encode direct probabilistic influence.
  - Each node Xi has a conditional probability table (CPT): P(Xi | Parents(Xi)).
- Factorization
  - Joint distribution: P(X1,...,Xn) = product_i P(Xi | Parents(Xi)).

Inference tasks

- Marginal inference
  - Compute P(Query | Evidence) by summing out hidden variables.
- MAP / MPE (preview)
  - Find most probable assignment to some variables given evidence.

Variable elimination

- Organize repeated computations by grouping factors and summing out variables one at a time.
- Elimination order controls complexity; good orders reduce intermediate factor size.

CS221 - Indented Outline (Notes 11-18)

Generated from your uploaded lecture PDFs (Overview, Search, MDPs, Logic, ML III).

Notes	Deck
11	Lecture 1: Overview
12	Lecture 5: Search I
13	Lecture 6: Search II
14	Lecture 7: MDPs I
15	Lecture 8: MDPs II
16	Lecture 16: Logic I
17	Lecture 17: Logic II
18	Lecture 4: Machine learning III

Notes 11: Lecture 1: Overview

Roadmap

- A brief history
    - Two views
- Course overview
- Course logistics
    - Optimization

Course plan

- "Low-level intelligence"
    - "High-level intelligence"
- Machine learning

Two views of AI

- AI agents: how can we create intelligence?
- AI tools: how can we benefit society?

An intelligent agent

- Perception
    - Robotics
    - Language
- Knowledge
- Reasoning
    - Learning

History: early AI

- 1956: Workshop at Dartmouth College; attendees: John McCarthy, Mar-
- vin Minsky, Claude Shannon, etc.
- Aim for general principles:
- Every aspect of learning or any other feature of intelligence can be so
- precisely described that a machine can be made to simulate it.

History: optimism and limits

- Machines will be capable, within twenty years, of doing any work a man
- can do. -Herbert Simon
- Within 10 years the problems of artificial intelligence will be substantially
- solved. -Marvin Minsky
- I visualize a time when we will be to robots what dogs are to humans,
- and I'm rooting for the machines. -Claude Shannon

Modeling paradigm

- Modeling
- Inference
- Learning

State-based models

- White to move

Variable-based models

- Constraint satisfaction problems:
- hard constraints (e.g.,
- Sudoku,
- scheduling)
- X1
- X2
- X3
- X4
- Bayesian networks: soft dependencies (e.g., tracking cars from sensors)
- H1
- H2
- H3

Knowledge-based systems

- Expert systems: elicit specific domain knowledge from experts in form
- of rules:
- if [premises] then [conclusion]

Deep learning

- AlexNet (2012): huge gains in object recognition; trans-
- formed computer vision community overnight
- AlphaGo (2016):
- deep reinforcement learning, defeat
- world champion Lee Sedol

Societal impact examples

- society => data => predictions

Summary

- History: roots from logic, neuroscience, statistics-melting pot!
- Modeling [reflex, states, variables, logic] + inference + learning
- paradigm
- AI has high societal impact, how to steer it positively?

Notes 12: Lecture 5: Search I

Roadmap

- Tree search
- Dynamic programming
- Uniform cost search

Search paradigm

- Modeling
- Inference
- Learning

State and planning abstraction

- Classifier (reflex-based models):
- single action y in {-1, +1
- Search problem (state-based models):
- action sequence (a1, a2, a3, a4, . . .)
- Key: need to consider future consequences of an action!

Tree search algorithms

- Legend: b actions/state, solution depth d, maximum depth D
- Algorithm
- Action costs
- Space
- Time
- O(bD)
- Backtracking
- any
- O(D)
- O(bD)
- DFS
- zero

Uniform cost search

- Key idea: state ordering
- UCS enumerates states in order of increasing past cost.
- Assumption: non-negativity
- All action costs are non-negative: Cost(s, a) >=0.
- UCS in action:

UCS bookkeeping

- Frontier

- Explored
- Unexplored
- Explored: states we've found the optimal path to
- Frontier: states we've seen, still figuring out how to get there
- cheaply
- Unexplored: states we haven't seen

UCS assumptions

- N total states, n of which are closer than end state
- Algorithm
- Cycles?
- Action costs
- Time/space
- DP
- no
- any
- O(N)
- >=0
- UCS
- yes

Summary

- State: summary of past actions sufficient to choose future actions
- optimally
- Dynamic programming: backtracking search with memoization
- potentially exponential savings
- cycles?

Notes 13: Lecture 6: Search II

Roadmap

- Learning costs
- A* search
- Relaxation

General framework: relaxation

- Removing constraints
- (knock down walls, walk/tram freely, overlap pieces)
- Reducing edge costs
- (from $\infty$to some finite cost)
- Example:
- Original: Cost((1, 1), East) = ∞
- Relaxed: Costrel((1, 1), East) = 1

A* as UCS with heuristic

- Algorithm: A* search [Hart/Nilsson/Raphael, 1968]
- Run uniform cost search with modified edge costs:
- Cost'(s, a) = Cost(s, a) + h(Succ(s, a)) -h(s)
- Intuition: add a penalty for how much action a takes us away from the
- end state
- Example:
- sstart
- send
- h(s) =
- Cost'(C, B) = Cost(C, B) + h(B) -h(C) = 1 + (3 -2) = 2

Heuristic admissibility

- Definition: admissibility
- A heuristic h(s) is admissible if
- h(s) <=FutureCost(s)
- Intuition: admissible heuristics are optimistic
- Theorem: consistency implies admissibility
- If a heuristic h(s) is consistent, then h(s) is admissible.
- Proof: use induction on FutureCost(s)

Heuristic consistency

- Definition: consistency
- A heuristic h is consistent if
- Cost'(s, a) = Cost(s, a) + h(Succ(s, a)) -h(s) >=0
- h(send) = 0.
- Condition 1: needed for UCS to work (triangle inequality).

- Cost(s, a)
- h(Succ(s, a))
- h(s)
- send
- Condition 2: FutureCost(send) = 0 so match it.

Search effort vs heuristic quality

- sstart
- send
- h = 0 (UCS)
- h = FutureCost
- If h(s) = 0, then A* is same as UCS.
- If h(s) = FutureCost(s), then A* only explores nodes on a mini-
- mum cost path.
- Usually h(s) is somewhere in between.

Learning as inverse problem

- Forward problem (search):
- Cost(s, a)
- (a1, . . . , ak)
- Inverse problem (learning):
- (a1, . . . , ak)
- Cost(s, a)

Cost tweaking intuition

- Costs: {walk:3, tram:2
- Costs: {walk:1, tram:3
- Minimum cost path:
- Minimum cost path:
- walk:3
- walk:1
- walk:3
- tram:2
- walk:1
- tram:3
- walk:3
- walk:1

Simple cost model

- Assume costs depend only on the action:
- Cost(s, a) = w[a]
- Candidate output path:
- a1 : w[a1]
- a2 : w[a2]

- a3 : w[a3]
- y:
- s0
- s1
- s2
- s3
- Path cost:

Structured perceptron learning

- Algorithm: Structured Perceptron (simplified)
- For each action: w[a] <- 0
- For each iteration t = 1, . . . T:
- For each training example (x, y) in Dtrain:
- Compute the minimum cost path y' given w
- For each action a in y: w[a] <- w[a] -1
- For each action a in y': w[a] <- w[a] + 1
- Try to decrease cost of true y (from training data)
- Try to increase cost of predicted y' (from search)
- [semi-live solution]

Applications

- Part-of-speech tagging
- Fruit flies like a banana.
- Noun Noun Verb Det Noun
- Machine translation
- la maison bleue
- the blue house

Summary

- Structured Perceptron (reverse engineering): learn cost functions
- (search + learning)
- A*: add in heuristic estimate of future costs
- Relaxation (breaking the rules): framework for producing consis-
- tent heuristics
- Next time: when actions have unknown consequences...

Notes 14: Lecture 7: MDPs I

Roadmap

- Markov decision process
- Policy evaluation
- Value iteration

From search to uncertainty

- deterministic
- state s, action a
- state Succ(s, a)

Markov decision process

- Definition: Markov decision process
- States: the set of states
- sstart in States: starting state
- Actions(s): possible actions from state s
- T(s'|s, a): probability of s' if take action a in state s
- Reward(s, a, s'): reward for the transition (s, a, s')
- IsEnd(s): whether at end of game
- 0 <=gamma <=1: discount factor (default: 1)

Utility and value

- Definition: 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
- Utility
- [in; stay, 4, end]
- [in; stay, 4, in; stay, 4, in; stay, 4, end]
- [in; stay, 4, in; stay, 4, end]
- [in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end] 16
- Definition: value (expected utility)
- The value of a policy at a state is the expected utility.

Policy evaluation

- Definition: value of a policy
- Let Vpi(s) be the expected utility received by following policy pi
- from state s.
- Definition: Q-value of a policy
- Let Qpi(s, a) be the expected utility of taking action a from state
- s, and then following policy pi.

- Qpi(s, pi(s))
- Vpi(s')
- s'
- T(s'|s, pi(s))
- Vpi(s)
- pi(s)

Summary so far

- MDP: graph with states, chance nodes, transition probabilities,
- rewards
- Policy: mapping from state to action (solution to MDP)
- Value of policy: expected utility over random paths
- Policy evaluation: iterative algorithm to compute value of policy

Optimality: values and policies

- Goal: try to get directly at maximum expected utility
- Definition: optimal value
- The optimal value V ∗(s) is the maximum value attained by any
- policy.

Value iteration

- Algorithm: value iteration [Bellman, 1957]
- Initialize V ∗(0)(s) <- 0 for all states s.
- For iteration t = 1, . . . , tVI:
- For each state s:
- V ∗(t)(s) <-
- T(s'|s, a)[Reward(s, a, s') + gammaV ∗(t-1)(s')]
- max
- a in Actions(s)
- s'
- Q∗(t-1)(s,a)
- Time: O(tVISAS')
- [semi-live solution]

Notes 15: Lecture 8: MDPs II

Roadmap

- Reinforcement learning
- Model-based value iteration
- Model-free policy evaluation & learning
- Covering the unknown
- Summary

From MDPs to reinforcement learning

- Markov decision process (offline)
- Have mental model of how the world
- works.
- Find policy to collect maximum rewards.
- Reinforcement learning (online)
- Don't know how the world works.
- Perform actions in the world to find out
- and collect rewards.

RL template

- action a
- agent
- environment
- reward r, new state s'
- Algorithm: reinforcement learning template
- For t = 1, 2, 3, . . .
- Choose action at = piact(st-1) (how?)
- Receive reward rt and observe new state st
- Update parameters (how?)

Model-based value iteration

- Data: s0; a1, r1, s1; a2, r2, s2; a3, r3, s3; . . . ; an, rn, sn
- Key idea: model-based learning
- Estimate the MDP: T(s'|s, a) and Reward(s, a, s')
- Transitions:
- ˆT(s'|s, a) = # times (s, a, s') occurs
- # times (s, a) occurs
- Rewards:
- Reward(s, a, s') = r in (s, a, r, s')

Model-free Monte Carlo

- Data (following policy pi):
- s0; a1, r1, s1; a2, r2, s2; a3, r3, s3; . . . ; an, rn, sn

- Recall:
    - Qpi(s, a) is expected utility starting at s, first taking action a, and
    - then following policy pi
- Utility:
    - ut = rt + gamma * rt+1 + gamma2 * rt+2 + * * *
- Estimate:
    - ˆQpi(s, a) = average of ut where st-1 = s, at = a
        - (and s, a doesn't occur in s0, * * * , st-2)

Q-learning

- Problem: model-free Monte Carlo and SARSA only estimate Qpi, but
- want Q∗to act optimally
    - Output
        - MDP
        - reinforcement learning
    - Qpi
        - policy evaluation
        - model-free Monte Carlo, SARSA
    - Q∗
        - value iteration

Exploration/exploitation

        - Key idea: balance
    - Need to balance exploration and exploitation.
- Examples from life: restaurants, routes, research

Epsilon-greedy

        - Algorithm: epsilon-greedy policy
        - arg maxa in Actions ˆQ∗(s, a)
        - probability 1 -■,
    - piact(s) =
        - random from Actions(s)
        - probability ■.
        - 99.8
        - 99.6
        - (2,1)
        - 50
- Run (or press ctrl-enter)
        - 50

Function approximation

        - Key idea: linear regression model
    - Define features ϕ(s, a) and weights w:
        - ˆQ∗(s, a; w) = w * ϕ(s, a)

- Example: features for volcano crossing
- $\phi_7(s, a) = 1[s = (5, *)]$
- $\phi_1(s, a) = 1[a = W]$
- $\phi_8(s, a) = 1[s = (*, 6)]$
- $\phi_2(s, a) = 1[a = E]$

Deep reinforcement learning

- just use a neural network for $\hat{Q}*(s, a)$, pi∗, T, etc
- Playing Atari [Google DeepMind, 2013]:
  - last 4 frames (images) => 3-layer NN => keystroke
  - ■-greedy, train over 10M frames with 1M replay memory
  - Human-level performance on some games (breakout), less good on
  - others (space invaders)

Summary so far

- Online setting: learn and take actions in the real world!
- Exploration/exploitation tradeoff
- Model-Based RL: estimate transitions & rewards, and use that
  - model of the MDP to find optimal policy
- Model-Free Monte Carlo: estimate Q-values from data
- Model-Free Bootstrapping: update towards target that depends
  - on estimate rather than just raw data

Notes 16: Lecture 16: Logic I

Motivation

- Tell information
- Ask questions
- Use natural language!
- [semi-live demo]
- Need to:
- Digest heterogenous information
- Reason deeply with that information

Logic in the AI stack

- question
- data
- Learning
- model
- Inference
- answer
- Examples: search problems, MDPs, games, CSPs, Bayesian networks

Ingredients of propositional logic

- Syntax
- Semantics
- formula
- models
- Inference
- rules

Syntax

- Propositional symbols (atomic formulas): A, B, C
- Logical connectives: not , and , or , -> , ↔
- Build up formulas recursively-if f and g are formulas, so are the fol-
- lowing:
- Negation: not f
- Conjunction: f and g
- Disjunction: f or g
- Implication: f -> g
- Biconditional: f ↔g

Semantics: truth and models

- {f : KB |= f

Model checking

- Checking satisfiability (SAT) in propositional logic is special case of solv-
- ing CSPs!
- Mapping:
    - =>
  - propositional symbol
    - variable
    - =>
    - formula
    - constraint
    - model
    - assignment

Knowledge bases

- M(Rain)
- M(Rain -> Wet)
- Wet
- Wet
- Rain
- Rain
- Intersection:
- M({Rain, Rain -> Wet)
- Wet
- Rain

Entailment and queries

- M(KB)
- M(f)
- Intuition: f added no information/constraints (it was already known).
- Definition: entailment
- KB entails f (written KB |= f) iff
- M(KB) $\subseteq$M(f).
- Example: Rain and Snow |= Snow

Tell/Ask interface

- Tell[f]
- KB
- Tell: It is raining.
- Tell[Rain]
- Possible responses:
- Already knew that: entailment (KB |= f)
- Don't believe that: contradiction (KB |= not f)
- Learned something new (update KB): contingent

Satisfiability

- Definition: satisfiability
- A knowledge base KB is satisfiable if M(KB) ■= ∅.
- Reduce Ask[f] and Tell[f] to satisfiability:
- Is KB ∪{not f satisfiable?
- yes
- no
- Is KB ∪{f satisfiable?
- entailment
- no
- yes
- contradiction
- contingent

Inference rules

- Example of making an inference:
- It is raining. (Rain)
- If it is raining, then it is wet. (Rain -> Wet)
- Therefore, it is wet. (Wet)
- Rain -> Wet
- Rain,
- (premises)
- (conclusion)
- Wet
- Definition: Modus ponens inference rule
- For any propositional symbols p and q:
- p -> q

Soundness and completeness

- The truth, the whole truth, and nothing but the truth.
- Soundness: nothing but the truth
- Completeness: whole truth

Historical note

- Logic was dominant paradigm in AI before 1990s
- Problem 1: deterministic, didn't handle uncertainty (probability
- addresses this)
- Problem 2: rule-based, didn't allow fine tuning from data (ma-
- chine learning addresses this)
- Strength: provides expressiveness in a compact way

Summary

- Syntax
- Semantics
- formula

- models
- Inference
- rules

Notes 17: Lecture 17: Logic II

Roadmap

- Resolution in propositional logic
    - First-order logic

Limits of propositional logic

- Alice and Bob both know arithmetic.
    - AliceKnowsArithmetic and BobKnowsArithmetic
- All students know arithmetic.
    - AliceIsStudent -> AliceKnowsArithmetic
    - BobIsStudent -> BobKnowsArithmetic
    - . . .
- Every even integer greater than 2 is the sum of two primes.
    - ???

First-order logic ingredients

        - Syntax
        - Semantics
- formula
        - models
- Inference
    - rules

Syntax of FOL

- Terms (refer to objects):
    - Constant symbol (e.g., arithmetic)
    - Variable (e.g., x)
    - Function of terms (e.g., Sum(3, x))
- Formulas (refer to truth values):
    - Atomic formulas (atoms):
        - predicate applied to terms (e.g.,
        - Knows(x, arithmetic))
    - Connectives
        - ->
        - applied
        - to

Quantifiers

- Universal quantification (forall):
    - Think conjunction: forall x P(x) is like P(A) and P(B) and * * *
- Existential quantification (exists):
    - Think disjunction: exists x P(x) is like P(A) or P(B) or * * *

- Some properties:
- not forall x P(x) equivalent to exists x not P(x)
- forall x exists y Knows(x, y) different from exists y forall x Knows(x, y)

Models in FOL

- Recall a model represents a possible situation in the world.
- Propositional logic: Model w maps propositional symbols to truth values.
- w = {AliceKnowsArithmetic : 1, BobKnowsArithmetic : 0
- First-order logic: ?

Definite clauses

- forall x forall y forall z (Takes(x, y) and Covers(y, z)) -> Knows(x, z)
- Note: if propositionalize, get one formula for each value to (x, y, z), e.g.,
- Definition: definite clause (first-order logic)
- A definite clause has the following form:
- forall x1 * * * forall xn (a1 and * * * and ak) -> b
- for variables x1, . . . , xn and atomic formulas a1, . . . , ak, b (which
- contain those variables).

Substitution

- Subst[{x/alice, P(x)] = P(alice)
- Subst[{x/alice, y/z, P(x) and K(x, y)] = P(alice) and K(alice, z)
- Definition: Substitution
- A substitution theta is a mapping from variables to terms.
- Subst[theta, f] returns the result of performing substitution theta on f.

Unification

- Unify[Knows(alice, arithmetic), Knows(x, arithmetic)] = {x/alice
- Unify[Knows(alice, y), Knows(x, z)] = {x/alice, y/z
- Unify[Knows(alice, y), Knows(bob, z)] = fail
- Unify[Knows(alice, y), Knows(x, F(x))] = {x/alice, y/F(alice)
- Definition: Unification
- Unification takes two formulas f and g and returns a substitution
- theta which is the most general unifier:
- Unify[f, g] = theta such that Subst[theta, f] = Subst[theta, g]
- or "fail" if no such theta exists.

Modus ponens

- Definition: modus ponens (first-order logic)
- a'
- 1, . . . , a'
- forall x1 * * * forall xn(a1 and * * * and ak) -> b
- b'

- Get most general unifier theta on premises:
- theta = Unify[a'
- 1 and * * * and a'
- k, a1 and * * * and ak]
- Apply theta to conclusion:
- Subst[theta, b] = b'

CNF conversion

- Input:
- forall x (forall y Animal(y) -> Loves(x, y)) -> exists y Loves(y, x)
- Output:
- (Animal(Y (x)) or Loves(Z(x), x)) and (not Loves(x, Y (x)) or Loves(Z(x), x))
- New to first-order logic:
- All variables (e.g., x) have universal quantifiers by default
- Introduce Skolem functions (e.g., Y (x)) to represent existential
- quantified variables

Resolution

- Recall: First-order logic includes non-Horn clauses
- forall x Student(x) -> exists y Knows(x, y)
- High-level strategy (same as in propositional logic):
- Convert all formulas to CNF
- Repeatedly apply resolution rule

Resolution algorithm

- Recall:
- relationship between entailment and contradiction (basically
- "proof by contradiction")
- KB |= f
- KB ∪{not f is unsatisfiable
- Algorithm: resolution-based inference
- Add not f into KB.
- Convert all formulas into CNF.
- Repeatedly apply resolution rule.
- Return entailment iffderive false.

Complexity

- forall x forall y forall z P(x, y, z)
- Each application of Modus ponens produces an atomic formula.
- If no function symbols, number of atomic formulas is at most
- (num-constant-symbols)(maximum-predicate-arity)
- If there are function symbols (e.g., F), then infinite...
- * * *
- Q(a)

- Q(F(a))
- Q(F(F(a)))
- Q(F(F(F(a))))

Summary

- Horn clauses
  - any clauses
- modus ponens
  - resolution
- linear time
  - exponential time
- less expressive
  - more expressive

Notes 18: Lecture 4: Machine learning III

Roadmap

- Generalization
- Unsupervised learning
- Summary

Evaluation and test discipline

- Dtrain
- Learner
- How good is the predictor f?
- Key idea: the real learning objective
- Our goal is to minimize error on unseen future examples.
- Don't have unseen examples; next best thing:
- Definition: test set
- Test set Dtest contains examples not used for training.

Generalization

- When will a learning algorithm generalize well?
- Dtrain
- Dtest

Bias/variance lens

- All predictors
- Learning
- f ∗
- approx. error
- Feature extraction
- est. error
- ˆf
- Approximation error: how good is the hypothesis class?
- Estimation error: how good is the learned predictor relative to
- the potential of the hypothesis class?
- Err(ˆf) -Err(g)
- + Err(g) -Err(f ∗)

Controlling model capacity

- All predictors
- Learning
- f ∗
- approx. error
- Feature extraction
- est. error

- ˆf
- As the hypothesis class size increases...
- Approximation error decreases because:
- taking min over larger set
- Estimation error increases because:
- harder to estimate something more complex

Mitigation: dimensionality

- w in Rd
- Reduce the dimensionality d:

Mitigation: regularization

- w in Rd
- Reduce the norm (length) ||w||:
- [whiteboard: x 7 -> w1x]

Hyperparameters and tuning

- Definition: hyperparameters
- Properties of the learning algorithm (features, regularization pa-
- rameter λ, number of iterations T, step size η, etc.).
- How do we choose hyperparameters?
- Choose hyperparameters to minimize Dtrain error? No - solution would
- be to include all features, set λ = 0, T -> ∞.
- Choose hyperparameters to minimize Dtest error? No - choosing based
- on Dtest makes it an unreliable estimate of error!

Development cycle

- Problem: simplified named-entity recognition
- Input: a string x (e.g., Governor [Gavin Newsom] in)
- Output: y, whether x contains a person or not (e.g., +1)
- Algorithm: recipe for success
- Split data into train, val, test
- Look at data to get intuition
- Repeat:
- Implement feature / adjust hyperparameters
- Run learning algorithm
- Sanity check train and val error rates, weights
- Look at errors to brainstorm improvements
- Run on test set to get final error rates

Unsupervised learning overview

- Data has lots of rich latent structures; want methods to discover
- this structure automatically.

Clustering

- Definition: clustering
- Input: training set of input points
- Dtrain = {x1, . . . , xn
- Output: assignment of each point to a cluster
- [z1, . . . , zn] where zi in {1, . . . , K
- Intuition: Want similar points to be in same cluster, dissimilar points to
- be in different clusters
- [whiteboard]

K-means objective

- Setup:
- Each cluster k = 1, . . . , K is represented by a centroid muk in Rd
- Intuition: want each point ϕ(xi) close to its assigned centroid muzi
- Objective function:
- ||ϕ(xi) -muzi||2
- Losskmeans(z, mu) =
- i=1
- Need to choose centroids mu and assignments z jointly

K-means algorithm

- min
- min
- mu Losskmeans(z, mu)
- Key idea: alternating minimization
- Tackle hard problem by solving two easy problems.

Local minima

- K-means is guaranteed to converge to a local minimum, but is not guar-
- anteed to find the global minimum.
- [demo: getting stuck in local optima, seed = 100]
- Solutions:
- Run multiple times from different random initializations
- Initialize with a heuristic (K-means++)

Embeddings and vectors

- [Mikolov et al., 2013]

Challenges

- Capabilities:
- More complex prediction problems (translation, generation)
- Unsupervised learning: automatically discover structure
- Responsibilities:

- Feedback loops: predictions affect user behavior, which generates
- data
- Fairness: build classifiers that don't discriminate?
- Privacy: can we pool data together
- Interpretability: can we understand what algorithms are doing?

Summary

- Feature extraction (think hypothesis classes) [modeling]
- Prediction (linear, neural network, k-means) [modeling]
- Loss functions (compute gradients) [modeling]
- Optimization
- (stochastic
- gradient,
- alternating
- minimization)
- [learning]
- Generalization (think development cycle) [modeling]