Chapter 3-4 (chapter 2-3 in online book)
Innleiðing í tilgjørdum viti

Notes

Contents

1 Monte Carlo Tree Search & AlphaGo

1.1 AlphaGo Overview

AlphaGo is an AI developed to play the board game Go at a professional level. It defeated top human players using a combination of techniques:

- Deep Learning
- Monte Carlo Tree Search (MCTS)
- High-Performance Computing

1.2 The Game of Go

- Played on a 19×19 grid $\Rightarrow$ very high branching factor.
- Branching factor: ~ 250 moves on average per turn.
- For comparison, chess has a branching factor of ~ 35.
- Goal: Surround more territory than the opponent.
- Complexity:
  - Estimated state space: $\sim 10^{100}$
  - Very subtle mechanics $\Rightarrow$ traditional evaluation functions (like in chess) are ineffective.

2 Limitations of Traditional Approaches

2.1 Minimax with Alpha-Beta Pruning

- Works well for chess:
  - Modest branching factor
  - Good heuristic evaluation functions
- In Go:
  - Branching factor is too large
  - Evaluation functions are too subtle to model effectively

3 Monte Carlo Tree Search (MCTS)

3.1 Overview

- Heuristic search algorithm applied to decision trees.
- Common in Go (last $\sim$10 years).
- Used when no good evaluation function exists.

3.2 Core Components

- **Selection:** Choose promising moves recursively until a leaf node is reached.
- **Expansion:** Add child nodes (possible future states) at the leaf.
- **Simulation:** Play out the game randomly to the end; record win/loss.
- **Backpropagation:** Propagate results back up the tree; update selection policy.

3.3 Benefits and Drawbacks

Pros:

- Does not require a predefined evaluation function.
- Effective in large, complex spaces.

Cons:

- Requires many simulations.
- Slow for very large games like Go.

4 Exploration vs Exploitation (Multi-Armed Bandit Problem)

4.1 Analogy

- Faced with multiple slot machines ("arms").
- Each has an unknown reward distribution.
- Must balance:
  - **Exploitation:** Use the arm that seems best.
  - **Exploration:** Try others to discover potential improvements.

4.2 Upper Confidence Bound (UCB)

- Balances exploration and exploitation.
- Formula considers both:
  - Average reward of each arm.
  - Number of times each arm has been tried.
- Policy:
  - Try each option once.
  - Then choose the option maximizing the UCB formula.

5 Deep Learning in AlphaGo

5.1 Policy and Value Networks

- **Policy Network:** Suggests promising moves, reducing branching factor.
- **Value Network:** Estimates game state value without full simulation.

5.2 Self-Play and Reinforcement Learning

- AlphaGo trained by playing against itself.
- Reinforcement learning improved decision-making, reducing search depth and branching factor.

6 Symbolic vs Sub-Symbolic Strategies

6.1 Symbolic AI

- Based on explicit knowledge and logical reasoning.
- Uses rules, symbols, and logic.
- Examples: Knowledge bases, expert systems, logic inference.

6.2 Sub-Symbolic AI

- Uses numerical/statistical methods.
- Examples: Neural networks, genetic algorithms, deep learning.
- Learns patterns from data rather than explicit rules.

7 Knowledge-Based Agents

7.1 Definition

Agents using internal knowledge representation to reason and derive conclusions.

7.2 Knowledge Base (KB)

- Repository of information stored in sentences.
- Sentences: Assertions about the world in a formal language.

7.3 Declarative vs Procedural Approaches

- **Declarative:** Feed system facts; it reasons to conclusions.
- **Procedural:** Encode specific behaviours directly.
- Often combined in modern systems.

8 Logic and Inference

8.1 Syntax and Semantics

- **Syntax:** Rules for forming valid sentences.
- **Semantics:** Meaning of sentences; correspondence with the world.

8.2 Logical Inference

- Derive new sentences from known ones using rules.
- Goal: Use facts to answer queries or deduce new knowledge.

8.3 Properties

- **Entailment:** α is entailed by KB if α is true in all models where KB is true. Example: KB = {"All humans are mortal", "Socrates is a human"} $\Rightarrow$ "Socrates is mortal"
- **Completeness:** An inference algorithm is complete if it can derive all entailed sentences.

9 Propositional Logic

9.1 Syntax

- Atomic symbols: $P, Q, \ldots$
- Compound sentences:
  - Conjunction: $P \wedge Q$
  - Disjunction: $P \vee Q$
  - Implication: $P \rightarrow Q$

9.2 Semantics (Models)

- A model assigns truth values to atomic propositions.
- A model = one possible world configuration.

9.3 Entailment in Propositional Logic

- $KB \models \alpha \iff \alpha$ is true in all models where KB is true.
- **Truth Table:** Lists all possible truth assignments; used to check entailment.

10 Inference Algorithms

- Goal: Decide if $KB \models \alpha$
- Methods:
  - Truth table checking (exhaustive).
  - More efficient algorithms (e.g., resolution).

11 Logic and Inference: Advanced Concepts

11.1 Complexity of Model Checking

- Determining whether $KB \models Q$ (entailment) can be computationally expensive.
- Naive approach: test all possible interpretations $\rightarrow$ exponential in number of propositions.
- Efficient methods rely on syntactic manipulation rather than enumerating all models.

11.2 Deductive Systems

- **Soundness:** If $KB \vdash Q$ then $KB \models Q$ (only derives truths).
- **Completeness:** If $KB \models Q$ then $KB \vdash Q$ (can derive everything entailed).
- Deduction theorem connects semantic entailment with syntactic derivation.

11.3 Inference Rules

- **Modus Ponens:** If $\alpha \rightarrow \beta$ and α are true, then β is true.
- **Conjunction Elimination:** If $\alpha \wedge \beta$ is true, then α is true.
- **Double Negation Elimination:** $\neg\neg\alpha \equiv \alpha$.
- **Implication Elimination:** $\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$.

- **Biconditional Elimination:** $\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$.
- **De Morgan's Laws:** $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$, $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$.
- **Distributive Laws:** $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$, etc.

11.4 Search Problem Analogy for Theorem Proving

- **Initial State:** Knowledge Base (KB)
- **Actions:** Inference rules applied to current KB
- **Transition Model:** New KB after inference
- **Goal Test:** Statement Q to prove
- **Path Cost:** Number of steps or inferences used

11.5 Resolution Method

- **Principle:** Combine clauses to derive new information until goal or contradiction.
- Example of resolving literals:

$$P \vee Q, \quad \neg P \Rightarrow Q$$

$$P \vee Q, \quad \neg P \vee R \Rightarrow Q \vee R$$

- **Clause:** A disjunction of literals, e.g., $P \vee Q \vee R$.
- **Conjunctive Normal Form (CNF):** Conjunction of clauses, e.g., $(A \vee B \vee C) \wedge (D \vee \neg E)$.
- Conversion to CNF:
  - Eliminate biconditionals and implications
  - Move NOT inwards using De Morgan's laws
  - Apply distributive laws

11.6 Proof by Contradiction

- Assume $\neg Q$ and derive a contradiction from KB.
- Ensures KB consistency.

11.7 Horn Clauses and Definite Clauses

- Horn Clause: CNF with at most one positive literal. Form: $\neg P \lor Q \equiv P \to Q$
- Definite Clause: Exactly one positive literal (the "head"), possibly many negative literals (the "body").
- Basis for efficient logic programming and automated reasoning.

11.8 Forward and Backward Chaining

- **Forward Chaining:** Start from known facts, apply inference rules to derive new facts until goal.
- **Backward Chaining:** Start from query Q, work backwards applying rules until known facts are reached.
- Linear resolution strategies (e.g., SLD resolution) reduce search space using sub-goals.

11.9 Limitations of Propositional Logic

- Cannot represent objects, relations, or quantifiers.
- Limited expressiveness for complex domains.

11.10 First-Order Logic (FOL)

- Extends propositional logic with:
  - Constants: specific objects
  - Variables: placeholders
  - Predicates: properties/relations over objects
  - Functions: map objects to objects
- Terms: constants, variables, or functions applied to terms
- Atomic formulas: predicates applied to terms
- Quantifiers:
  - Universal: $\forall x\, P(x)$
  - Existential: $\exists x\, P(x)$

12 Advanced First-Order Logic and Optimization Techniques

12.1 Universal and Existential Quantification

- **Universal Quantification:** $\forall x\, P(x)$ — "for all x, $P(x)$ is true".
- **Existential Quantification:** $\exists x\, P(x)$ — "there exists an x such that $P(x)$ is true".
- These quantifiers are fundamental in First-Order Logic (FOL) for expressing general statements about objects.

12.2 First-Order Logic (FOL) Formulas

- **Arity:** Number of arguments a predicate or function takes.
- **Predicates with functions:** Allow expressing relationships between objects and their properties.
- **Interpretations:** Map constants, functions, and predicates to values or truth assignments. Example: $F = c_1 + c_3 > c_2$, domain $\{1, 2, 3, 4\}$ Relations are sets of tuples where the interpretation is true.

12.3 Key FOL Techniques

- **Skolemization:** Remove existential quantifiers by introducing Skolem constants or functions.
- **Unification:** Process of finding substitutions for variables to make terms or predicates match.
- **Resolution in FOL:** Extends propositional resolution to FOL using unification to derive new clauses.

12.4 Soundness and Completeness in FOL

- **Soundness:** Every derived formula is logically true.
- **Completeness:** Every logically entailed formula can be derived using the inference system.

12.5 Limitations of First-Order Logic

- Incompleteness in some domains
- Scalability challenges with large knowledge bases
- Difficult to represent spatial and temporal reasoning
- Handling uncertainty is non-trivial

12.6 Tautology

- A formula that is true under every possible interpretation.
- Important in both propositional logic and FOL.

13 Optimization Concepts

13.1 Definition and Objective

- Optimization: Selecting the best option from a set of alternatives.
- Formalization:
  - Variables with associated domains
  - Objective function mapping assignments to real numbers
  - Optimality criterion: find assignment minimizing or maximizing the objective function
- Example: Minimizing loss associated with a variable assignment.

13.2 Local Search Algorithms

- Maintain a single current state and explore neighboring states to improve the objective.
- **Example:** Placing houses and hospitals in 2D space to minimize total distance.
- **State space landscape:** Visualizes global/local maxima and minima, flat regions, and shoulders.

13.3 Hill Climbing Variants

- **Steepest-Ascent Hill Climbing:** Move to the neighbor with the highest improvement.
  - Issues: Local maxima, plateaus, ridges
- **Stochastic Hill Climbing:** Randomly select among improving neighbors.
- **First-Choice Hill Climbing:** Randomly evaluate neighbors and move to first improvement found.
- **Random-Restart Hill Climbing:** Restart from random initial states to escape local maxima.
- **Local Beam Search:** Maintain k states in parallel, keep best successors at each step.

13.4 Iterative Improvement Techniques

- **Simulated Annealing:** Accept worse neighbors with decreasing probability over time to escape local optima.
- **Gradient Descent:** For continuous domains, adjust variables proportionally to reduce the objective function.

13.5 Genetic Algorithms

- Maintain a population of candidate solutions.
- Randomly select pairs and perform crossover to produce offspring.
- Apply mutation to introduce variability.
- Iterate until an acceptable solution is found.

13.6 Key Considerations

- Local search is fast but can get stuck in local optima.
- Random restarts, stochastic methods, and population-based methods improve robustness.
- Continuous vs discrete domains may require specialized algorithms (e.g., gradient descent for continuous).