Chapter 8 (10)
Innleiðing í tilgjørdum viti

Notes

Contents

A search problem is the foundational concept in AI problem-solving. The agent must decide which actions to take to transition from the initial state to a goal state, while considering the path cost and possible state transitions.

1 Elements of a Search Problem (Common exam question)

- Initial State : The starting point of the agent in the problem space.
- Actions : A set of all possible actions the agent can take from a state.
- Transition Model: A function $Result(s, a)$ that returns the resulting state from performing action a in state s.
- States: All possible configurations or situations the agent may encounter.
- Goal Test: A function that checks if a given state is a goal state.
- Path Cost: A numerical value that defines the cost associated with a path from the initial state to a given state.

Key point: A solution is a sequence of actions that leads from the initial state to the goal state.

1.1 Additional Key Concepts

State space: The complete set of all possible states reachable from the initial state, given the actions and transition model.

Search Tree: A structure where nodes represent states, and branches represent actions leading to those states.

Search Graph (vs. Tree): Includes cycles and shared states — more efficient than trees but requires cycle detection.

Node (in a search tree/graph): A data structure that typically contains:

- State
- Parent Node
- Action taken to reach this state
- Path cost so far
- Depth (optional)

Branching Factor (b): The average number of successors (child nodes) per state.

Belief State: A representation of all the information an agent has about the world (useful in partially observable environments). Example: tracking visited locations.

Directed Acyclic Graph (DAG): A graph with directed edges and no cycles.

2 Search Strategies

2.1 Uninformed Search

Uninformed (Blind) Search has no domain-specific knowledge is used.

Strategies:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Uniform-Cost Search (Lowest-Cost-First)
- Iterative Deepening DFS

2.2 Search Algorithm Structure

Generic Graph Search

- Initialize:
  - Frontier = Initial State
  - Explored set = empty set
- Loop
  - If Frontier is empty return no solution
  - Remove node from Frontier
  - If node.state is goal return the solution
  - add node.state to Explored set
  - Expand node and add resulting nodes to Frontier (Only if not in Frontier or Explored set)

Depth-First Search (DFS):

- Frontier: Stack (Last in first out)
- Explores the deepest node first.
- Pros: Low memory usage
- Cons: Can get stuck in deep or infinite paths
- Time complexity: $O(b^m)$
- Space complexity: $O(b \cdot m)$

Where b is the branching factor and m is the maximum depth.

Iterative Deepening DFS (IDDFS)

- Combines benefits of BFS and DFS.
- Repeatedly runs DFS with increasing depth limits.

- Pros: Finds optimal solution with less space
- Redundant expansions

Simplified Pseudocode:

```
for depth = 0 to infinity:
    result = Depth-Limited-Search(initial, depth)
    if result not equal failure:
        return result
```

Breadth-First Search (BFS)

- Frontier: Queue (FIFO)
- Explores the shallowest nodes first.
- Guarantees: Finds the solution with the fewest steps (minimum depth)
- Time complexity: $O(b^d)$
- Space complexity: $O(b^d)$

Where b is the branching factor and d is depth of the shallowest solution.

Uniform-Cost Search (UCS)

- Expands the node with the lowest path cost.
- Uses a priority queue, ordered by path cost.
- Guarantees:Finds optimal solution (lowest cost)
- Time complexity: Depends on cost granularity (can be exponential)

-

Informed (Heuristic) Search

- Uses domain-specific knowledge (heuristics) to guide search.
- Examples: $A*$ Search and Greedy Best-First Search

3 Heuristic (Informed) Search

Heuristic search leverages problem-specific knowledge to guide exploration more efficiently than uninformed strategies.

3.1 Heuristic Function $h(n)$

A heuristic is a non-negative estimate of the cost from node n to the goal.

- Example: Euclidean (straight-line) distance in a map.

3.2 Admissibility

A heuristic is **admissible** if it never overestimates the true cost:

$$h(n) \leq \text{true cost from } n \text{ to goal}$$

- Examples: Euclidean distance, Manhattan distance (in grid environments without obstacles).

3.3 Consistency (Monotonicity)

A heuristic is **consistent** if:

$$h(n) \leq c(n, n') + h(n')$$

where $c(n, n')$ is the cost of moving from n to n'.

- Equivalent to the triangle inequality.
- Ensures the total estimated cost $f(n) = g(n) + h(n)$ is non-decreasing along a path.

4 Summary of Key Terms

- Agent: Entity making decisions
- Environment: The world the agent operates in
- Action: Operation an agent can take
- Frontier: Set of nodes available for expansion
- Explored Set: Set of visited states
- Goal State: Target state for the agent
- Path Cost: Numerical cost of reaching a node
- Optimal Solution: The solution with the lowest total path cost

Exam Tips:

- List and explain the 5 elements of a search problem
- Distinguish between search tree and search graph
- Compare time and space complexity for BFS, DFS, IDDFS
- Know pros/cons of different search methods

5 Types of Heuristic Search Algorithms

5.1 Greedy Best-First Search

- Expands the node that appears closest to the goal (lowest $h(n)$).
- Frontier: Priority Queue ordered by $h(n)$.
- Pros: Fast in many cases.
- Cons: Not optimal; can fail with misleading heuristics.

5.2 A^* Search

Evaluates nodes using:

$$f(n) = g(n) + h(n)$$

where $g(n)$ = cost so far, $h(n)$ = estimated cost to goal.

- Frontier: Priority Queue ordered by $f(n)$.
- Combines UCS (optimality) and Greedy Best-First (efficiency).
- Guarantees:
  - Completeness: Finds a solution if one exists.
  - Optimality: Finds least-cost path if $h(n)$ is admissible.

5.3 Time Complexity

- Generally exponential in solution depth.
- Strongly affected by branching factor b.

6 Search Complexity Concepts

- **Forward Branching Factor**: Number of successors from a node.
- **Backward Branching Factor**: Number of predecessors.
- Lower branching factor $\Rightarrow$ better time complexity.
- Example: $2 \cdot b^{k/2} \ll b^k$

7 Multi-Agent Systems & Game Theory

7.1 Overview

Game theory studies the behavior of multiple agents that may be cooperative, competitive, or mixed.

7.2 Utility

Utility measures future rewards from a state or action.

- Agents may have different utility functions.
- They act autonomously with limited/incomplete information.
- Outcomes depend on other agents' actions.

8 Types of Games

8.1 Cooperative Games

Agents coordinate to achieve shared goals.

8.2 Competitive Games

Agents have conflicting goals. Often modeled as zero-sum games.

9 Game Representations

9.1 Normal Form (Strategic Form)

- Payoff matrix shows utilities for each agent given action profiles.
- Action Profile: Assignment of an action to each agent.
- Utility Function: Maps action profiles to payoffs.

9.2 Extensive Form (Game Tree)

Represents sequential games with perfect information.

9.3 Imperfect Information & Simultaneous Actions

- Partially Observable Games: Agents lack full information (e.g., card games).
- Simultaneous Move Games: Players act without knowing others' choices.
- Information Set: States indistinguishable to a player at decision time.

10 Strategy and Planning

- **Strategy**: A full plan of actions for all possible situations.
- **Strategy Profile**: One strategy per agent.
- Strategies may be deterministic or probabilistic.

11 Adversarial Search

11.1 Minimax Algorithm

- MAX tries to maximize utility; MIN tries to minimize it.
- Works recursively from terminal states backward.

Key Functions:

- s_0: Initial state.
- Player(s): Returns whose turn it is.
- Actions(s): Legal actions.
- Result(s, a): Next state.
- Terminal(s): Checks if game is over.
- Utility(s): Value of terminal state.

Simplified Pseudocode:

```
def minimax(s):
    if Terminal(s):
        return Utility(s)
    if Player(s) == MAX:
        return max(minimax(Result(s,a)) for a in Actions(s))
    else:
        return min(minimax(Result(s,a)) for a in Actions(s))
```

12 Optimizations

12.1 Alpha-Beta Pruning

Prunes branches of the minimax tree that cannot affect the final decision, greatly reducing node evaluations.

13 Evaluation Function

Used when search is cut off early (e.g., depth limit). Estimates utility of a non-terminal state for approximate reasoning.

14 Monte Carlo Tree Search & AlphaGo

14.1 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

14.2 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.

15 Limitations of Traditional Approaches

15.1 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

16 Monte Carlo Tree Search (MCTS)

16.1 Overview

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

16.2 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.

16.3 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.

17 Exploration vs Exploitation (Multi-Armed Bandit Problem)

17.1 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.

17.2 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.

18 Deep Learning in AlphaGo

18.1 Policy and Value Networks

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

18.2 Self-Play and Reinforcement Learning

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

19 Symbolic vs Sub-Symbolic Strategies

19.1 Symbolic AI

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

19.2 Sub-Symbolic AI

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

20 Knowledge-Based Agents

20.1 Definition

Agents using internal knowledge representation to reason and derive conclusions.

20.2 Knowledge Base (KB)

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

20.3 Declarative vs Procedural Approaches

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

21 Logic and Inference

21.1 Syntax and Semantics

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

21.2 Logical Inference

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

21.3 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.

22 Propositional Logic

22.1 Syntax

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

22.2 Semantics (Models)

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

22.3 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.

23 Inference Algorithms

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

24 Advanced First-Order Logic and Optimization Techniques

24.1 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.

24.2 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.

24.3 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.

24.4 Soundness and Completeness in FOL

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

24.5 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

24.6 Tautology

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

25 Optimization Concepts

25.1 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.

25.2 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.

25.3 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.

25.4 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.

25.5 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.

25.6 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).

26 Logic and Inference: Advanced Concepts

26.1 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.

26.2 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.

26.3 Inference Rules

- **Modus Ponens:** If $\alpha \to \beta$ 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:** $\alpha \to \beta \equiv \neg\alpha \vee \beta$.
- **Biconditional Elimination:** $\alpha \leftrightarrow \beta \equiv (\alpha \to \beta) \wedge (\beta \to \alpha)$.
- **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.

26.4 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

26.5 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

26.6 Proof by Contradiction

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

26.7 Horn Clauses and Definite Clauses

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

26.8 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.

26.9 Limitations of Propositional Logic

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

26.10 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: $\exists x\, P(x)$

27 Introduction to AI Learning

Artificial Intelligence (AI) learning refers to the process by which an AI system improves its performance over time based on data, experience, or feedback. This process can:

- Expand the range of behaviors.
- Improve accuracy on tasks.
- Increase speed of task completion.

28 Core Components of Learning

- **Task:** The specific behavior or function being improved.
- **Data:** The experiences (examples, feedback, etc.) used to improve performance.
- **Measure of Improvement:** The metric used to quantify learning progress.

29 Large Data Requirements

AI systems often require large datasets to generalize effectively. The learner operates as a black box that takes:

- Experiences
- Problem description
- Background knowledge or bias
- And produces answers or decisions

Learner → Models → Reasoner

30 Common Learning Paradigms

- Supervised Classification
- Unsupervised Learning
- Reinforcement Learning
- Analytic Learning
- Inductive Logic Programming
- Statistical Relational Learning

31 Feedback and Bias

Feedback: Determines how success is measured.

- P: Assumes only observed negative examples are negative.
- N: Assumes only observed positive examples are positive.

Bias: Different assumptions about the distribution of examples, influencing the hypothesis space. Example: A linear model or a polynomial fit may result from differing biases over the same data.

32 Learning as Search

Learning can be viewed as searching through a space of hypotheses or models.

- The search space is often very large.
- Techniques like gradient descent and stochastic simulation are used.

Learning algorithm = Search Space + Evaluation Function + Search Method

33 Challenges in Learning

Data Issues:

- Inadequate features
- Missing or noisy data
- Incorrectly labeled features

Types of Error:

- Limited representation (representation bias)
- Limited search (search bias)
- Limited data (variance)
- Noisy features (noise)

34 Model Representation and Bias-Variance Tradeoff

- Richer representations enable better problem-solving.
- But they are harder to learn and more prone to overfitting.
- Tradeoff exists between bias (underfitting) and variance (overfitting).

35 Supervised Learning

Given a dataset of input-output pairs, the goal is to learn a function f that maps inputs to outputs.

35.1 Terminology

- Input Features
- Target Features
- Training Examples

35.2 Classification

Learning a function to map input to a discrete category (e.g., spam or not spam). Target features Y_i are discrete.

Nearest Neighbour Classification: Choose the class of the nearest training example.

K-Nearest Neighbour Classification: Choose class by majority vote among k nearest examples.

35.3 Linear Classification

A linear decision boundary:

$$x_1 = \text{humidity}, \quad x_2 = \text{pressure}$$

$$h(x_1, x_2) = \begin{cases} \text{rain} & \text{if } w_0 + w_1 x_1 + w_2 x_2 \geq 0 \\ \text{no rain} & \text{otherwise} \end{cases}$$

$$\text{Input Vector: } x = (1, x_1, x_2), \quad \text{Weight Vector: } w = (w_0, w_1, w_2)$$

$$h_w(x) = \begin{cases} 1 & \text{if } w \cdot x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron Learning Rule:

$$w_i \leftarrow w_i + \alpha(y - h_w(x))x_i$$

35.4 Activation Functions

- **Step function:** Hard threshold.
- **Sigmoid function:** Soft threshold.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Logistic Regression: Minimize the error of the logistic function by adjusting weights.

35.5 Support Vector Machines

- Adjusts the decision boundary to maximize the margin.
- **Linearly Separable:** Exists a hyperplane where classes are separable.
- Projects data into higher-dimensional space if necessary.

36 Regression

Supervised learning task to predict continuous values.

$$\text{Goal: predict target } Y \text{ from inputs } X_1, X_2, \ldots, X_n$$

Linear Regression:

$$\text{Minimize Sum of Squared Errors (SSE):} \quad SSE(E, w) = \sum_{e \in E} (o_e - p_e)^2$$

Stochastic Gradient Descent: Update weights incrementally after each example.

37 Evaluating Hypotheses

- o_e: Observed value
- p_e: Predicted value
- Error: $|o_e - p_e|$

37.1 Loss Functions

- **0-1 Loss:**

$$L(o, p) = \begin{cases} 0 & \text{if } o = p \\ 1 & \text{otherwise} \end{cases}$$

- L_1 **Loss:** Absolute error
- L_2 **Loss:** Squared error

$$L = (o - p)^2$$

- L_∞ **Loss:** Max error

38 Overfitting and Regularisation

Overfitting: Model captures noise in training data, failing to generalize.

Regularisation:

$$\text{Cost}(h) = \text{Loss}(h) + \lambda \cdot \text{Complexity}(h)$$

- Encourages simpler models.
- Feature selection: Removing irrelevant features.

39 Model Evaluation Techniques

- Holdout Validation: Train/test split.
- K-Fold Cross Validation: Split into k parts and test on each part iteratively.

40 Practical Tools

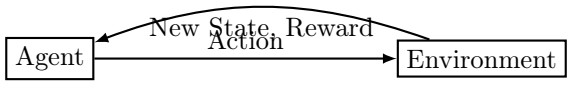
Scikit-learn: Python machine learning library offering tools for classification, regression, clustering, and evaluation.

41 Reinforcement Learning

Definition: Reinforcement learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives rewards or punishments based on the actions it takes, and learns to maximize cumulative rewards over time.

Basic Loop:

- Agent observes the current **state** of the environment.
- Agent chooses an **action**.
- Environment returns a new **state** and a **reward**.
- The agent updates its policy based on the experience.



RLHF: Reinforcement Learning from Human Feedback

Used for training large language models. The steps:

1. Prompt is given to the model.
2. Model generates multiple responses.
3. Human annotators rank the responses.
4. A reward model is trained to predict human preferences.
5. The main model is updated using reinforcement learning to improve future responses.

42 Unsupervised Learning

Definition: Learning patterns in data without labeled outcomes.

Clustering

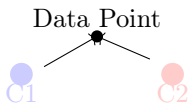
Grouping objects such that similar ones are in the same cluster.

Applications:

- Genetic research
- Image segmentation
- Market research
- Medical imaging
- Social network analysis

K-Means Clustering

1. Initialize k cluster centers randomly.
2. Assign each point to the nearest cluster center.
3. Recalculate the cluster centers.
4. Repeat until convergence.



43 Neural Networks

Biological Neurons

Neurons receive electrical impulses from others, process them, and may activate (fire). Artificial Neural Networks (ANNs) are inspired by this concept.

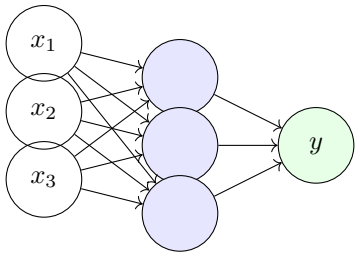
Artificial Neural Network (ANN)

A neural network is a parameterized function that maps input features $x_1, \ldots, x_n$ to an output y:

$$y = F(x_1, \ldots, x_n)$$

Key Properties:

- **Input layer:** Accepts features.
- **Hidden layers:** Process and transform input.
- **Output layer:** Produces prediction.



Activation Function: ReLU

$$\text{ReLU}(x) = \max(0, x)$$

44 Training Neural Networks

Gradient Descent

An optimization algorithm used to minimize the loss function by updating weights in the opposite direction of the gradient.

- Initialize weights randomly.
- Compute gradient of loss with respect to weights.
- Update weights:

  $$w := w - \eta \cdot \nabla L(w)$$

  where η is the learning rate.

Mini-batch Gradient Descent

Instead of using the entire dataset, a small random batch is used to compute the gradient.

Backpropagation

Used to train networks with hidden layers.

1. Forward pass to compute output.
2. Compute loss (error).
3. Backward pass to propagate error and compute gradients.
4. Update weights using gradient descent.

45 Dense Neural Networks

- Each neuron is connected to every neuron in the previous and next layer.
- Can learn linear patterns.
- Limit: Only learns linearly separable functions.

Solution: Use multiple layers (Multilayer Perceptrons) to learn complex patterns.
Depth: Number of layers.
Width: Number of neurons per layer.
Size: Maximum width of all layers.

46 Training Strategies

Train/Validation Split

- 80% for training
- 20% for validation

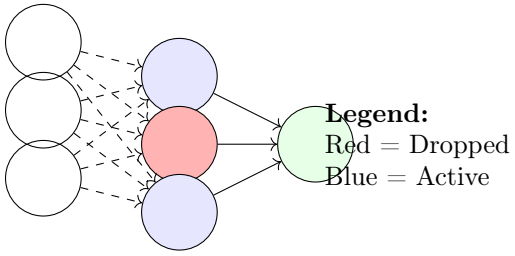
Overfitting: Model memorizes training data and fails to generalize.
Underfitting: Model is too simple or lacks data.
Solution:

- Get more data or augment existing data.
- Tune hyperparameters (e.g., number of layers, learning rate).

Dropout Regularization

Dropout randomly disables a fraction of neurons during training to prevent overfitting.

- Drop rate = proportion of units set to zero.
- During inference, all units are used, but activations are scaled by $\frac{1}{1-\text{rate}}$.



47 Uncertainty

Uncertainty arises when an agent lacks complete information about the world. It reflects the agent's **degree of belief** in various propositions.

- Uninformed agents must make decisions with incomplete information.
- Probability is used to quantify uncertainty.
- **Example:** Betting on a dice roll involves assigning probabilities to each possible face of the die.

48 Probability

Probability represents a **subjective measure of belief**. It helps agents reason under uncertainty.

Random Variables

A random variable can take different values due to randomness.

- Dice: domain$(X) = \{1, 2, 3, 4, 5, 6\}$
- Weather: domain(Weather) = {sun, cloud, rain, wind, snow}
- Traffic: {none, light, heavy}
- Flight status: {on time, delayed, cancelled}

Possible Worlds: A complete assignment of values to all variables. Denoted by ω, with $P(\omega)$ representing the probability of that world.

49 Axioms of Probability

1. Non-negativity: $0 \leq P(a)$
2. Normalization: $P(\text{true}) = 1$
3. Additivity (Mutual exclusivity): $P(a \vee b) = P(a) + P(b)$ if $a \wedge b = \text{false}$

50 Types of Probability

Unconditional (Prior) Probability

Belief in a proposition before seeing any evidence. **Example:** $P(\text{Rain}) = 0.3$

Conditional Probability

Belief in a proposition given some evidence. Notation: $P(a \mid b)$ **Example:** $P(\text{Traffic} = \text{heavy} \mid \text{Rain})$

Conditioning and Evidence

Evidence e eliminates possible worlds where e is false. We update our beliefs using:

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$$

- **Prior**: Belief before evidence
- **Likelihood**: Probability of evidence given a hypothesis
- **Posterior**: Updated belief

51 Probability Distribution

Definition: A function assigning probabilities to each value in a random variable's domain.

Flight Status	Probability
On Time	0.6
Delayed	0.3
Cancelled	0.1

52 Independence

Two events are independent if knowledge of one does not affect the other.

$$P(a \wedge b) = P(a) \cdot P(b)$$

Example: Rolling two dice.

53 Bayes' Rule

$$P(a \mid b) = \frac{P(b \mid a) \cdot P(a)}{P(b)}$$

Example: Medical Diagnosis

- $P(\text{Disease}) = 0.01$
- $P(\text{Positive Test} \mid \text{Disease}) = 0.99$
- $P(\text{Positive Test}) = 0.05$

$$P(\text{Disease} \mid \text{Positive}) = \frac{0.99 \cdot 0.01}{0.05} = 0.198$$

54 Causal vs. Evidential Reasoning

Causal Reasoning

Inferring effects from causes. **Example:** $P(\text{Wet} \mid \text{Rain})$

Evidential Reasoning

Inferring causes from effects. **Example:** $P(\text{Rain} \mid \text{Wet})$

55 Rules of Probability

- Negation: $P(\neg a) = 1 - P(a)$
- Inclusion-Exclusion: $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$
- Marginalisation: $P(a) = P(a \wedge b) + P(a \wedge \neg b)$
- Law of Total Probability: $P(a) = P(a \mid b)P(b) + P(a \mid \neg b)P(\neg b)$

56 Conditional Independence

Variables A and B are conditionally independent given C if:

$$P(A \mid B, C) = P(A \mid C)$$

Example: Given a disease C, symptoms A and B may be conditionally independent.

57 Bayesian Networks

A Bayesian network is a **Directed Acyclic Graph (DAG)** representing dependencies between variables.

- Nodes: Random variables
- Arrows: Causal/statistical dependencies
- Each node has a conditional probability table (CPT)

Structure:

For each node X:

$$P(X \mid \text{Parents}(X))$$

Example Network:

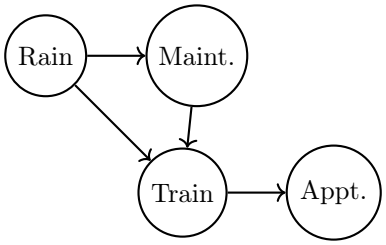
$$\text{Rain} \rightarrow \text{Maintenance} \rightarrow \text{Train} \rightarrow \text{Appointment}$$

Parents:

- Train: Rain, Maintenance
- Appointment: Train

CPTs:

- $P(\text{Rain})$
- $P(\text{Maintenance} \mid \text{Rain})$
- $P(\text{Train} \mid \text{Rain}, \text{Maintenance})$
- $P(\text{Appointment} \mid \text{Train})$



58 Joint Probability using Chain Rule

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \text{Parents}(X_i))$$

Example with Previous Network:

$P(\text{Rain}, \text{Maintenance}, \text{Train}, \text{Appointment}) = P(\text{Rain}) \cdot P(\text{Maintenance} \mid \text{Rain}) \cdot P(\text{Train} \mid \text{Rain}, \text{Maintenance}) \cdot P(\text{Appointment}$

59 Inference in Bayesian Networks

Inference involves using known knowledge (evidence) to deduce new knowledge (unknown or hidden variables).

Steps to Construct a Bayesian Network

1. **What is observed?** (Evidence)
2. **What do you want to find out?** (Query)
3. **What other variables simplify the model?** (Hidden or latent variables)

Hidden Variables

Hidden variables are unobserved variables that affect the probabilities of observed or queried variables.

Example:

$$\alpha \cdot P(\text{Appointment}, \text{Light}, \text{No})$$

Here, α is a normalization constant that ensures the resulting distribution sums to 1.

60 Marginalisation and Normalisation

- **Marginalisation:**
$$P(a) = P(a, b) + P(a, \neg b)$$
- **Conditional Inference:**
$$P(X \mid e) = \alpha \cdot P(X, e) = \alpha \cdot \sum_{y} P(X, e, y)$$
where y ranges over hidden variables.
- α is the normalisation constant.

Challenges

- Inference by enumeration scales poorly with many variables. - The complexity of exact inference depends heavily on the network's structure.

61 Exact and Approximate Inference Methods

Exact Inference Methods

- **Inference by Enumeration**
- **Variable Elimination**

Approximate Inference Methods

- **Stochastic Simulation**
- **Variational Methods**
- **Sampling Techniques**

62 Sampling Techniques

Sampling is used to approximate distributions by drawing random values from them.

Cumulative Probability Distribution

- Example: Weather distribution

$$\begin{aligned} P(X = v_1) &= 0.3 \quad \text{(Sunny)} \\ P(X = v_2) &= 0.4 \quad \text{(Rainy)} \\ P(X = v_3) &= 0.1 \quad \text{(Cloudy)} \\ P(X = v_4) &= 0.2 \quad \text{(Foggy)} \end{aligned}$$

Steps:

1. Order the values of the domain X
2. Generate the cumulative distribution
3. Draw a random number $y \sim \text{Uniform}(0, 1)$
4. Select the value of X based on where y falls in the cumulative distribution

Rejection Sampling

- Generate full samples.
- Reject samples that do not match the evidence.
- Inefficient if evidence is rare.

Likelihood Weighting

- Fix the evidence variables.
- Sample other variables conditioned on evidence.
- Weight each sample by its likelihood.

Importance Sampling

- Draw samples from an easier distribution.
- Weight the samples to correct the bias.

63 Uncertainty Over Time: Markov Models

Markov Assumption

The current state depends only on a finite number of previous states.

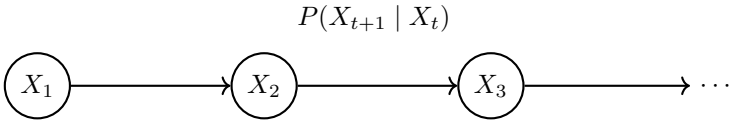
Markov Chain

A sequence of random variables $X_1, X_2, \ldots, X_n$ satisfying:

$$P(X_{t+1} \mid X_t, X_{t-1}, \ldots, X_1) = P(X_{t+1} \mid X_t)$$

Key Assumption: The future is independent of the past given the present.

Transition Model



64 Hidden Markov Models (HMMs)

Definition

A Markov model with hidden (unobservable) states.

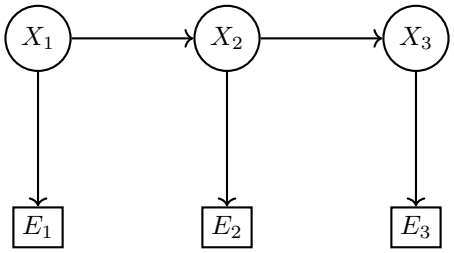
- **Transition Probability:** $P(X_{t+1} \mid X_t)$
- **Emission (Sensor) Probability:** $P(E_t \mid X_t)$

Sensor Markov Assumption

Current observation depends only on the current state.

Tasks in HMMs

- **Filtering:** Estimate current state given past observations
- **Prediction:** Estimate future states
- **Smoothing:** Estimate past states
- **Most Likely Explanation:** Find the most likely sequence of hidden states



65 Language Models

Simple Models: Bag of Words and Unigrams

- Words are treated as independent.
- $P(w_i)$: Probability distribution over words.
- Frequency vector indicates word usage.

Bigram Models

$$P(w_i \mid w_{i-1})$$

- The domain of each variable is the vocabulary.
- Models word sequences using adjacent word pairs.

Trigram Models

$$P(w_i \mid w_{i-1}, w_{i-2})$$

N-gram Models

- Extend bigrams/trigrams to any n-length sequence.
- Capture local dependencies in language.

Limitations of N-gram Models

- **Sparsity:** Many possible sequences are never seen in training data.
- **Data Size:** Large corpora are needed to get reliable estimates.
- **Lack of Semantic Understanding:** Words are treated as symbols, ignoring meaning and context.
- **Lack of Global Information:** Limited context prevents understanding long-distance dependencies.

66 Overview of NLP

Natural Language Processing (NLP) is a field at the intersection of linguistics, computer science, and artificial intelligence. It enables machines to understand, interpret, and generate human language.

66.1 Key Applications

- Automatic Summarisation
- Information Extraction
- Machine Translation
- Question Answering
- Text Classification

67 Challenges in NLP

- **Syntax:** Structure of a sentence.
- **Semantics:** Meaning of words and sentences.

68 Approaches in NLP

68.1 Rule-Based

- Hand-coded rules based on expert knowledge.
- Easy to interpret and debug.
- Does not require large volumes of data.

68.2 Data-Driven

- Infers rules from data.
- Accuracy improves with more data.
- May be biased or hard to interpret.
- More flexible and scalable.

69 Probabilistic Models

69.1 Markov Chains

Used in both rule-based and data-driven systems depending on how transition probabilities are derived (e.g., n-gram frequency).

69.2 Bayes' Rule

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$

69.3 Naive Bayes Classifier

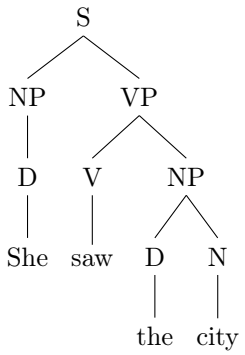
Assumes independence among features to compute the probability of a class given the features.

70 Formal Grammars

70.1 Context-Free Grammar (CFG)

Abstracts meaning from text to represent structure. It uses production rules to define possible sentence structures.

70.2 Syntax Tree Example



71 Word Representation

71.1 Tokenisation

- **Character-level:** Good for spell-checking, compact
- **Word-level:** Intuitive, but memory-intensive
- **Subword-level:** Balanced approach, most commonly used

71.2 One-Hot Encoding

Represents words as binary vectors.

71.3 Word2Vec

Learns vector representations (embeddings) of words based on their context.

71.3.1 Continuous Bag of Words (CBOW)

Predicts target word from context.

CBOW Architecture

- **Input:** One-hot encoded vectors of context words
- **Hidden Layer:** Trains word embeddings
- **Output:** Softmax over vocabulary

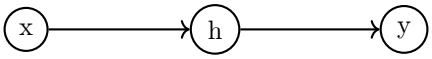
72 Neural Networks in NLP

72.1 Perceptron

A basic unit in neural networks:

$$y = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

72.2 Feedforward Neural Network



72.3 Encoder-Decoder Architecture

Used for sequence-to-sequence tasks like translation.

72.4 Recurrent Neural Networks (RNNs)

Processes sequences step by step and maintains a hidden state.

- Captures sequential dependencies.
- Suffers from vanishing gradients and depth issues.

72.5 Attention Mechanism

Weighs hidden states by importance.

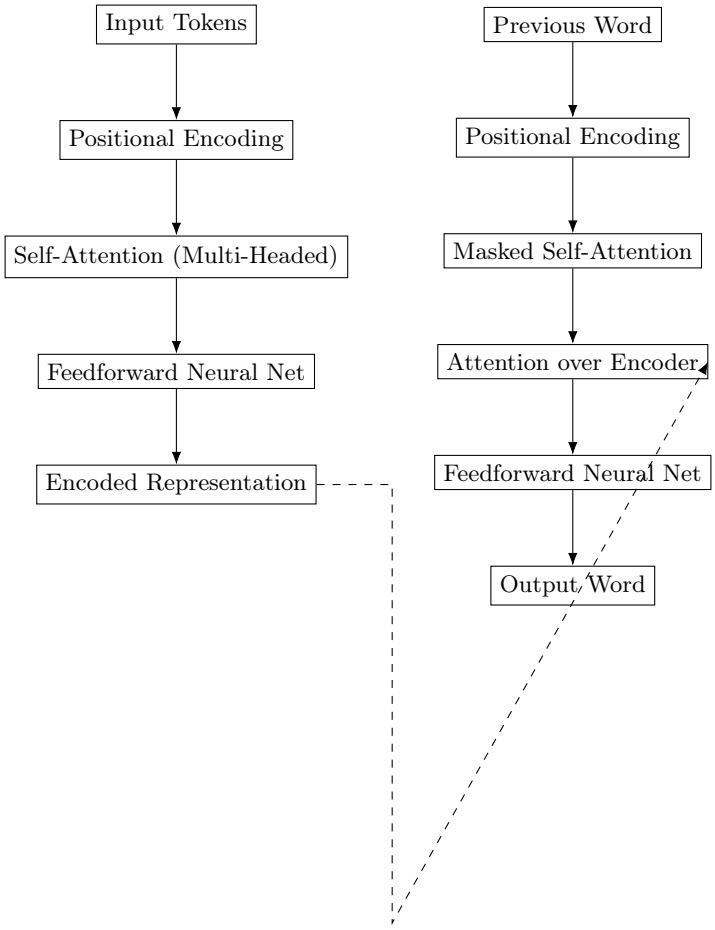
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

73 Transformers

73.1 Key Features

- Self-attention
- Parallel processing
- Positional encoding

73.2 Architecture Diagram



73.3 Advantages

- Enables parallel training.
- Captures long-term dependencies.

73.4 Limitations

- Memory intensive for long sequences.

74 Exam Preparation Summary

- Activation Functions: Step, Sigmoid
- Perceptron = Linear function + Step
- NLP Language Models: N-grams, Neural, Transformers