Artificial Intelligence

Comprehensive Course Notes

Gabriel Rovesti

May 3, 2025

Contents

Chapter 1

Introduction to Artificial Intelligence

1.1 Historical Overview

Artificial Intelligence emerged as a formal discipline in 1956 during a workshop at Dartmouth College, attended by pioneers such as John McCarthy, Marvin Minsky, and Claude Shannon. The field initially focused on general principles of intelligence and problem-solving.

1.1.1 Early Successes

Early achievements in AI included:

- Samuel's Checkers program (1952): Learned weights and played at a strong amateur level
- Logic Theorist (1955): Developed by Newell & Simon to prove theorems in Principia Mathematica using search and heuristics
- General Problem Solver (GPS): A more general approach to problem-solving

1.1.2 Expert Systems Era (1970s-1980s)

Expert systems represented a significant paradigm in AI development:

- Systems designed to elicit domain-specific knowledge from experts in the form of rules
- DENDRAL: Inferred molecular structure from mass spectrometry data
- MYCIN: Diagnosed blood infections and recommended antibiotics
- XCON: Converted customer orders into parts specifications, saving DEC $40 million annually by 1986

1.1.3 Neural Networks Development

Neural networks have had a complex history in AI:

- 1943: McCulloch and Pitts introduced artificial neural networks, connecting neural circuitry and logic
- 1969: Minsky and Papert's "Perceptrons" book showed limitations of linear models (unable to solve XOR), which slowed neural networks research
- 1986: Rumelhardt, Hinton, and Williams popularized backpropagation for training multi-layer networks
- 1989: LeCun applied convolutional neural networks to recognize handwritten digits for USPS

1.1.4 Deep Learning Revolution

Deep learning has transformed AI in recent years:

- AlexNet (2012): Made huge gains in object recognition, transforming computer vision
- AlphaGo (2016): Used deep reinforcement learning to defeat world champion Lee Sedol
- Foundation Models (since 2019): GPT series, DALL·E, and other large language and multimodal models have shown impressive capabilities across various domains

1.2 AI Agent Architecture

1.2.1 Definition of Intelligent Agents

An intelligent (or rational) agent is an entity that:

- Perceives its environment through sensors
- Acts upon the environment through actuators
- Aims to achieve its goals as much as possible given available information
- Can be viewed abstractly as a function mapping perception sequences to actions:

  $$f : P^* \to A$$

  where P^* represents all possible sequences of perceptions and A represents actions

1.2.2 PEAS Framework

To design a rational agent, we must specify the PEAS components:

- **P**erformance measure: How success is evaluated
- **E**nvironment: The context in which the agent operates
- **A**ctuators: How the agent can execute actions
- **S**ensors: How the agent perceives the environment

Example for an autonomous taxi:

Component	**Description**
Performance Measure	Safety, reaching destination, profit, legal operation, passenger comfort, etc.
Environment	Roads, traffic, pedestrians, passengers, weather conditions, etc.
Actuators	Steering, acceleration, braking, signals, display, etc.
Sensors	Cameras, LIDAR, GPS, speedometer, engine sensors, etc.

Table 1.1: PEAS example for an autonomous taxi

1.2.3 Environment Types

Environment properties significantly impact agent design:

Property	**Description**
Observable vs. Partially Observable	Whether the agent's sensors give access to the complete state of the environment
Deterministic vs. Stochastic	Whether the next state is completely determined by the current state and action
Episodic vs. Sequential	Whether the agent's experience is divided into independent episodes
Static vs. Dynamic	Whether the environment can change while the agent is deliberating
Discrete vs. Continuous	Whether the state of the environment has a finite number of distinct states
Single-agent vs. Multi-agent	Whether the agent is operating alone or with other agents

Table 1.2: Environment properties affecting agent design

1.2.4 Agent Types

Four main types of agents with increasing complexity:

Simple Reflex Agents

- Select actions based only on current percept
- Implement condition-action rules (if-then)
- No memory of past percepts
- Limited in partially observable environments

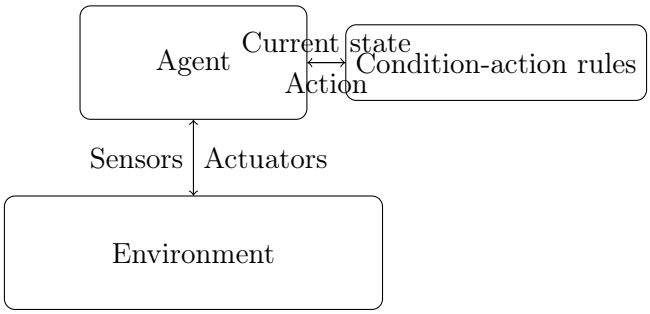


Figure 1.1: Simple reflex agent architecture

Model-Based Reflex Agents

- Maintain internal state tracking the unobserved aspects of the world
- Update state based on percept history and environment model
- Select actions using condition-action rules based on current state

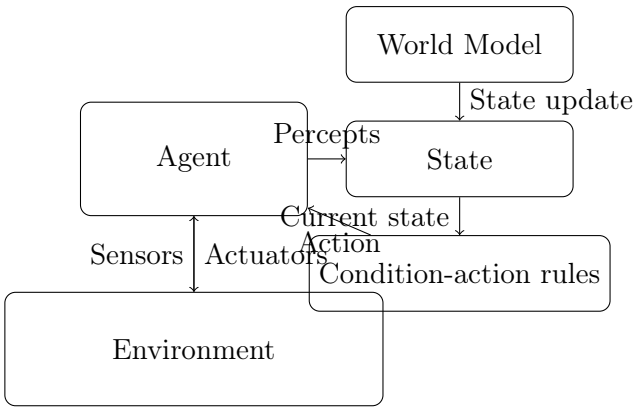


Figure 1.2: Model-based reflex agent architecture

Goal-Based Agents

- Consider future actions and the desirability of their outcomes
- Search and planning capabilities to find action sequences that achieve goals
- More flexible than reflex agents as explicit goals can be modified

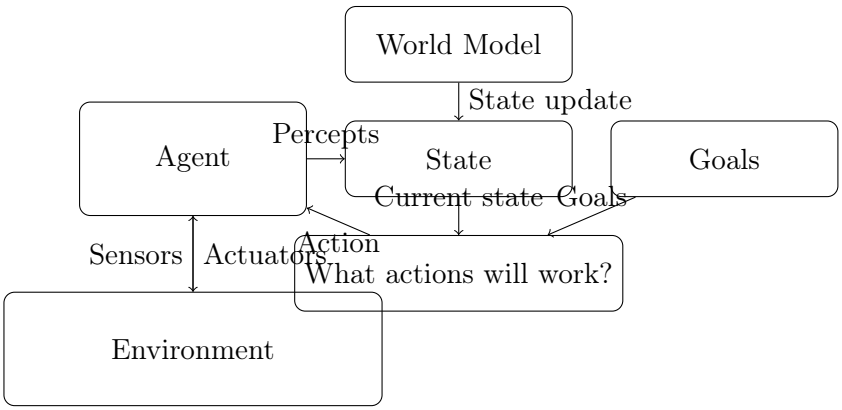


Figure 1.3: Goal-based agent architecture

Utility-Based Agents

- Maximize "happiness" or utility function that maps states to real numbers
- Can handle conflicting goals and uncertainty in action outcomes
- Makes optimal decisions in complex environments

Learning Agents

- Improve performance through experience
- Can start with little knowledge and become more competent over time
- Components: learning element (improves performance), performance element (selects actions), critic (provides feedback), problem generator (suggests exploratory actions)

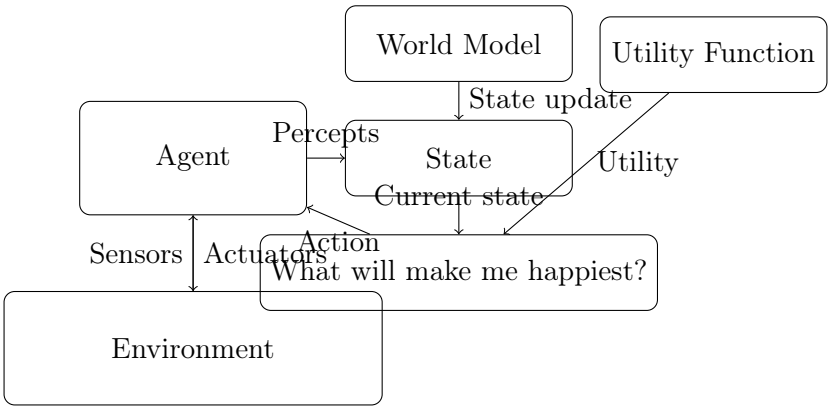


Figure 1.4: Utility-based agent architecture

1.3 Environment Representations

Different ways to represent the environment:

- **Atomic representation**: Each state is indivisible, no internal structure
- **Factored representation**: States split into a fixed set of variables or attributes
- **Structured representation**: Objects and their various relationships can be described explicitly

Chapter 2

Problem Solving and Search

2.1 Problem Formulation

A problem can be formally defined by four components:

- **Initial state**: The starting situation
- **Successor function**: Defines available actions and their results
- **Goal test**: Determines if a given state is a goal state
- **Path cost**: Function that assigns a cost to each path

Alternatively, we can separate:

- **Actions**: What can be performed in a given state
- **Transition model**: The effect of each action

A solution is a sequence of actions leading from the initial state to a goal state.

2.2 Uninformed Search Strategies

Uninformed (or blind) search strategies use only the information provided in the problem definition.

2.2.1 Breadth-First Search (BFS)

- Expands shallowest unexpanded node first
- Implementation: fringe is a FIFO queue
- Properties:
  - Complete: Yes (if branching factor b is finite)

  - Time complexity: $O(b^d)$ where d is solution depth
  - Space complexity: $O(b^d)$
  - Optimal: Yes (if all step costs are equal)

2.2.2 Uniform-Cost Search

- Expands least-cost unexpanded node first
- Implementation: fringe is a priority queue ordered by path cost
- Equivalent to BFS if all step costs are equal
- Properties:
  - Complete: Yes (if step costs $\geq \epsilon > 0$)
  - Time complexity: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ where C^* is the cost of optimal solution
  - Space complexity: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
  - Optimal: Yes

2.2.3 Depth-First Search (DFS)

- Expands deepest unexpanded node first
- Implementation: fringe is a LIFO queue (stack)
- Properties:
  - Complete: No (can get stuck in infinite loops)
  - Complete in finite spaces with cycle detection
  - Time complexity: $O(b^m)$ where m is maximum depth
  - Space complexity: $O(bm)$
  - Optimal: No

2.2.4 Iterative Deepening Search (IDS)

- Performs DFS with increasing depth limits
- Combines benefits of DFS (space efficiency) and BFS (completeness and optimality)
- Properties:
  - Complete: Yes (if b is finite)
  - Time complexity: $O(b^d)$
  - Space complexity: $O(bd)$
  - Optimal: Yes (if all step costs are equal)

2.2.5 Bidirectional Search

- Searches forward from initial state and backward from goal
- Terminates when the two searches meet
- Properties:
  - Complete: Yes (if b is finite)
  - Time complexity: $O(b^{d/2})$
  - Space complexity: $O(b^{d/2})$
  - Optimal: Yes (if using BFS in both directions)

2.3 Informed Search Strategies

Informed search strategies use problem-specific knowledge beyond the definition of the problem to find solutions more efficiently.

2.3.1 Best-First Search

- Uses an evaluation function $f(n)$ for each node
- Expands the node with the lowest $f(n)$ value
- Implementation: fringe is a priority queue ordered by $f(n)$

2.3.2 Greedy Search

- Evaluation function $f(n) = h(n)$ (heuristic function)
- $h(n)$ estimates cost from n to the nearest goal
- Expands node that appears to be closest to goal
- Properties:
  - Complete: No (can get stuck in loops)
  - Complete in finite space with repeated-state checking
  - Time complexity: $O(b^m)$, but a good heuristic can significantly improve this
  - Space complexity: $O(b^m)$
  - Optimal: No

2.3.3 A* Search

- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ is the cost from start to node n
- $h(n)$ is the estimated cost from n to goal
- Requires an admissible heuristic (never overestimates)
- Properties:
  - Complete: Yes (unless there are infinitely many nodes with $f \leq f(G)$)
  - Time complexity: Exponential in [relative error in $h \times$ length of solution]
  - Space complexity: Keeps all nodes in memory
  - Optimal: Yes, if $h(n)$ is admissible

2.3.4 Consistency in Heuristics

A heuristic is consistent (or monotonic) if:

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

where $c(n, a, n')$ is the cost of reaching n' from n via action a.

If h is consistent, then:

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

This means $f(n)$ is non-decreasing along any path, ensuring optimality in graph search.

2.3.5 Memory-Bounded Search Algorithms

A* has high memory requirements. Several alternatives exist:

Iterative Deepening A* (IDA*)

- Like iterative deepening but uses f-value cutoffs
- At each iteration, performs depth-first search with a cutoff on $f(n)$
- Memory complexity: $O(bd)$

Recursive Best-First Search (RBFS)

- Recursive algorithm that attempts to mimic A* but with linear space
- Keeps track of the f-value of the best alternative path
- Backs up when current node exceeds this alternative value
- Memory complexity: $O(bd)$

Simplified Memory-Bounded A* (SMA*)

- Expands best nodes like A* until memory is full
- When memory is full, drops the worst leaf node
- Stores the f-value of dropped nodes with their parents
- Properties:
  - Complete only if solution can be kept in memory
  - Optimal if any optimal solution is reachable (otherwise returns best reachable solution)

2.3.6 Heuristic Functions

- A heuristic function $h(n)$ estimates the cost from state n to the goal
- An admissible heuristic never overestimates the true cost
- A consistent heuristic satisfies the triangle inequality
- If $h_2(n) \geq h_1(n)$ for all n (both admissible), then h_2 dominates h_1 and is better for search
- Admissible heuristics can be derived from relaxed problems
- $h(n) = \max(h_a(n), h_b(n))$ combines admissible heuristics into a stronger one

Example: For the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = sum of Manhattan distances of each tile from its goal position

2.4 Local Search Algorithms

Local search algorithms keep only a current state and try to improve it. Useful for optimization problems where the path to the solution is irrelevant.

2.4.1 Hill Climbing

- Moves to neighbor with highest value
- Terminates at a local maximum
- Problems: local maxima, plateaus, ridges
- Variants:
  - Stochastic hill climbing: chooses randomly among uphill moves
  - First-choice hill climbing: generates successors randomly until finding an improvement
  - Random-restart hill climbing: performs a series of hill climbs from random initial states

2.4.2 Simulated Annealing

- Allows "bad" moves to escape local optima
- Probability of accepting a worse state depends on:
  - How much worse it is
  - A temperature parameter that decreases over time
- Probability of accepting a move that decreases value by ΔE at temperature T:
$$P(\text{accept}) = e^{-\Delta E/T}$$
- Guaranteed to find global optimum if cooled slowly enough

2.4.3 Local Beam Search

- Keeps track of k states rather than just one
- Generates all successors of these k states
- Selects the k best successors from the entire list
- Problem: Can still get stuck if all k states end up on same local hill

2.4.4 Genetic Algorithms

- Maintain a population of states (individuals)
- New generation formed by:
  - Selection: Choose pairs based on fitness
  - Crossover: Combine parts of two individuals

  - Mutation: Random changes to introduce diversity

- Works best when subcomponents of a solution are meaningful

2.5 Online Search

In online search, the agent doesn't know the transition model or what states exist.

2.5.1 Online Search Characteristics

- Agent knows only:
  - Actions available in the current state
  - Step cost function
  - Goal test
- Must interleave computation and action
- Exploration is required to find good solutions

2.5.2 Online Search Algorithms

Online Depth-First Search

- Applies DFS in an online setting
- Keeps track of unexplored actions and paths taken
- Backtracks when necessary
- Works only in safely explorable state spaces

Learning Real-Time A* (LRTA*)

- Maintains cost estimates $H(s)$ for each visited state
- Initially, $H(s) = h(s)$ (heuristic value)
- Updates $H(s)$ based on experience:

$$H(s) \leftarrow \min_a(c(s, a, s') + H(s'))$$

- Selects action with lowest estimated total cost
- Eventually learns accurate cost estimates and finds optimal paths

Chapter 3

Adversarial Search

3.1 Game Theory Basics

Adversarial search deals with environments where multiple agents have conflicting goals.

3.1.1 Game Types

Games can be categorized along several dimensions:

- Perfect vs. imperfect information
- Deterministic vs. stochastic
- Zero-sum vs. general-sum
- Sequential vs. simultaneous moves

3.1.2 Game Representation

Games are typically represented as:

- Initial state
- Players (often MAX and MIN in two-player games)
- Actions available to each player in each state
- Transition model defining results of actions
- Terminal test to identify end of game
- Utility function (or payoff) defining outcome value for each player

3.2 Minimax Algorithm

For deterministic, perfect-information, zero-sum games:

- MAX tries to maximize the score
- MIN tries to minimize the score
- Optimal strategy: Choose move that leads to best achievable outcome against optimal opponent

Algorithm 1 Minimax Algorithm

```
1: function MINIMAX(state)
2:     if Terminal-Test(state) then return Utility(state)
3:     end if
4:     if Player(state) = MAX then
5:         value ← −∞
6:         for all action in Actions(state) do
7:             value ← MAX(value, Minimax(Result(state, action)))
8:         end for
9:     else
10:        value ← ∞
11:        for all action in Actions(state) do
12:            value ← MIN(value, Minimax(Result(state, action)))
13:        end for
14:    end ifreturn value
15: end function
```

3.2.1 Minimax Properties

- Complete: Yes, if tree is finite
- Optimal: Yes, against optimal opponent
- Time complexity: $O(b^m)$ where b is branching factor and m is maximum depth
- Space complexity: $O(bm)$ with depth-first exploration

3.3 Alpha-Beta Pruning

Alpha-beta pruning is an optimization for the minimax algorithm that eliminates branches that cannot influence the final decision.

- α = best value found so far for MAX along the current path
- β = best value found so far for MIN along the current path
- Prune when current node's value is guaranteed to be worse than an alternative

Algorithm 2 Alpha-Beta Pruning Algorithm

```
1: function ALPHA-BETA-SEARCH(state)
2:     value ← Max-Value(state, −∞, ∞) return action that produces value
3: end function
4: function MAX-VALUE(state, α, β)
5:     if Terminal-Test(state) then return Utility(state)
6:     end if
7:     value ← −∞
8:     for all action in Actions(state) do
9:         value ← MAX(value, Min-Value(Result(state, action), α, β))
10:        if value ≥ β then return value                    ▷ Prune
11:        end if
12:        α ← MAX(α, value)
13:    end forreturn value
14: end function
15: function MIN-VALUE(state, α, β)
16:    if Terminal-Test(state) then return Utility(state)
17:    end if
18:    value ← ∞
19:    for all action in Actions(state) do
20:        value ← MIN(value, Max-Value(Result(state, action), α, β))
21:        if value ≤ α then return value                    ▷ Prune
22:        end if
23:        β ← MIN(β, value)
24:    end forreturn value
25: end function
```

3.3.1 Alpha-Beta Properties

- Does not affect final result compared to minimax
- Best-case time complexity: $O(b^{m/2})$ with perfect ordering
- Perfect ordering: examine best moves first
- With good move ordering, can effectively "double" the search depth

3.4 Resource Limits and Evaluation Functions

In practice, we can't search to terminal states for complex games like chess.

3.4.1 Evaluation Functions

- Replace utility function with heuristic evaluation function
- Estimate the expected utility of the game from a given position
- Often a weighted linear sum of features:

$$\text{eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- Example (chess): $\text{eval}(s) = w_1 \cdot \text{material} + w_2 \cdot \text{mobility} + \ldots$

3.4.2 Cutoff Test

- Replace terminal test with a cutoff test
- Typically based on maximum depth and sometimes other factors
- Apply evaluation function at cutoff nodes

3.4.3 Horizon Effect

- Problem where an agent delays inevitable negative events so they fall beyond the search horizon
- Solutions:
  - Quiescence search: Continue search at "unstable" positions
  - Singular extensions: Extend search for particularly promising or threatening moves

3.5 Games with Chance

For games with randomness (dice, cards, etc.), we need to consider expected outcomes.

3.5.1 Expectiminimax

3.5.2 Expectiminimax Properties

- Time complexity: $O(b^m \cdot n^m)$ where n is number of chance outcomes
- Exact utility values matter (not just their ordering)
- Alpha-beta pruning is less effective but still possible

Algorithm 3 Expectiminimax Algorithm

```
1: function EXPECTIMINIMAX(state)
2:     if Terminal-Test(state) then return Utility(state)
3:     end if
4:     if Player(state) = MAX then
5:         value ← −∞
6:         for all action in Actions(state) do
7:             value ← MAX(value, Expectiminimax(Result(state, action)))
8:         end for
9:     else if Player(state) = MIN then
10:        value ← ∞
11:        for all action in Actions(state) do
12:            value ← MIN(value, Expectiminimax(Result(state, action)))
13:        end for
14:    else if Player(state) = CHANCE then
15:        value ← 0
16:        for all outcome r with probability P(r) do
17:            value ← value + P(r) * Expectiminimax(Result(state, r))
18:        end for
19:    end ifreturn value
20: end function
```

3.6 Partially Observable Games

In games like poker or bridge, players don't have perfect information about the state.

3.6.1 Information Sets

- An information set is a collection of states that are indistinguishable to the player
- Players must use the same strategy for all states in an information set

3.6.2 Strategies for Partially Observable Games

- Determinization: Sample from possible game states and apply perfect-information techniques
- State evaluation: Compute expected value over all possible actual states
- Perfect information Monte Carlo: Repeatedly sample determinizations and search them

- Information set search: Consider all possible states consistent with the observed history

Chapter 4

Knowledge Representation: Propositional Logic

4.1 Knowledge-Based Agents

Knowledge-based agents use a knowledge base (KB) to represent facts about the world and use logical inference to make decisions.

4.1.1 Architecture

- TELL: Informs the KB what the agent perceives
- ASK: Queries the KB about what action to take
- The knowledge base consists of sentences in a formal language

4.2 Propositional Logic

4.2.1 Syntax

- Atomic sentences: True, False, P, Q, R, ...
- Complex sentences formed using logical connectives:
  - Negation: $\neg$ (not)
  - Conjunction: $\wedge$ (and)
  - Disjunction: $\vee$ (or)
  - Implication: $\Rightarrow$ (if-then)
  - Biconditional: $\Leftrightarrow$ (if and only if)

4.2.2 Semantics

- Truth values assigned to atomic sentences
- Truth values of complex sentences determined by truth tables
- A model m is an assignment of truth values to all propositional symbols
- m is a model of a sentence α if α is true in m
- $M(\alpha)$ is the set of all models of α

4.2.3 Entailment

- $KB \models \alpha$ means "α is entailed by KB"
- This holds if α is true in all models where KB is true
- Formally: $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

4.3 Inference Algorithms

4.3.1 Inference by Enumeration

- List all possible models
- Check each model to see if it satisfies KB
- For each such model, check if it satisfies α
- Time complexity: $O(2^n)$ for n symbols

4.3.2 Forward Chaining

For KB of Horn clauses (clauses with at most one positive literal):

- Applies Modus Ponens repeatedly
- Start with known atomic facts
- Apply rules whose premises are satisfied to derive new facts
- Continue until query is derived or no new facts can be derived
- Complete for Horn clause KBs
- Linear time complexity in the size of the KB

Algorithm 4 Forward Chaining Algorithm

```
1: function FC-Entails?(KB, q)
2:     count ← a table mapping clauses to number of unsatisfied premises
3:     inferred ← a table mapping symbols to boolean (all false initially)
4:     agenda ← a queue of symbols known to be true in KB
5:     while agenda is not empty do
6:         p ← Pop(agenda)
7:         if p = q then return true
8:         end if
9:         if inferred[p] = false then
10:            inferred[p] ← true
11:            for all clause c in KB where p is in c.PREMISE do
12:                decrement count[c]
13:                if count[c] = 0 then
14:                    add c.CONCLUSION to agenda
15:                end if
16:            end for
17:        end if
18:    end whilereturn false
19: end function
```

4.3.3 Backward Chaining

- Works backward from the query
- Find rules that could conclude the query
- Recursively prove the premises of these rules
- Depth-first recursive proof search
- Can be more efficient by avoiding irrelevant facts

4.3.4 Resolution

- Complete inference method for propositional logic
- Requires conversion to Conjunctive Normal Form (CNF)
- CNF: Conjunction of disjunctions of literals
- Resolution rule: $\frac{\ell_1 \vee \cdots \vee \ell_k,\ m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$ where ℓ_i and m_j are complementary literals

Algorithm 5 Backward Chaining Algorithm

```
1: function BC-ENTAILS?(KB, q) return BC-Or(KB, q, {})
2: end function
3: function BC-OR(KB, goal, substitution)
4:     if substitution = failure then return false
5:     end if
6:     if goal is empty then return substitution
7:     end if
8:     first, rest ← First(goal), Rest(goal)
9:     for all rule r in KB do
10:        (lhs ⇒ rhs) ← standardize-variables(r)
11:        for all θ in BC-And(KB, lhs, Unify(rhs, first, substitution)) do
12:            result ← BC-Or(KB, rest, θ)
13:            if result ≠ failure then return result
14:            end if
15:        end for
16:    end forreturn failure
17: end function
18: function BC-AND(KB, goals, substitution)
19:    if substitution = failure then return {}
20:    end if
21:    if goals is empty then return {substitution}
22:    end if
23:    first, rest ← First(goals), Rest(goals)
24:    return ∪_{θ∈BC-Or(KB,first,substitution)} BC-And(KB, rest, θ)
25: end function
```

Converting to CNF

Steps to convert a sentence to CNF:

1. Eliminate biconditionals: $\alpha \Leftrightarrow \beta$ becomes $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

2. Eliminate implications: $\alpha \Rightarrow \beta$ becomes $\neg\alpha \vee \beta$

3. Move negation inwards using De Morgan's laws:

   - $\neg(\alpha \wedge \beta)$ becomes $\neg\alpha \vee \neg\beta$
   - $\neg(\alpha \vee \beta)$ becomes $\neg\alpha \wedge \neg\beta$
   - $\neg\neg\alpha$ becomes α

4. Apply distributivity: $\alpha \vee (\beta \wedge \gamma)$ becomes $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Algorithm 6 Resolution Algorithm

```
1: function PL-Resolution(KB, α)
2:     clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
3:     new ← {}
4:     while true do
5:         for all pairs of clauses C_i, C_j in clauses do
6:             resolvents ← PL-Resolve(C_i, C_j)
7:             if resolvents contains the empty clause then return true
8:             end if
9:             new ← new ∪ resolvents
10:        end for
11:        if new ⊆ clauses then return false
12:        end if
13:        clauses ← clauses ∪ new
14:    end while
15: end function
```

Completeness of Resolution

Resolution is complete: if $KB \models \alpha$, then resolution will derive a contradiction from $KB \wedge \neg\alpha$.

This is proven through the ground resolution theorem:

- If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause
- The resolution closure is the set of all clauses derivable by repeated application of the resolution rule

Chapter 5

First-Order Logic

5.1 Limitations of Propositional Logic

- Cannot express general relationships about objects
- Cannot express properties that hold for all or some objects
- Requires separate proposition for each fact about each object

5.2 First-Order Logic Syntax

5.2.1 Basic Elements

- Constants: Objects in the domain (e.g., KingJohn, 2, UCB)
- Predicates: Relations or properties (e.g., Brother, $>$)
- Functions: Mappings from objects to objects (e.g., Sqrt, LeftLegOf)
- Variables: Stand for objects (e.g., x, y, a, b)
- Connectives: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$
- Equality: $=$
- Quantifiers: $\forall$ (universal), $\exists$ (existential)

5.2.2 Terms and Sentences

- Term: Constant, variable, or function applied to terms
- Atomic sentence: Predicate applied to terms, or equality between terms
- Complex sentences: Built from atomic sentences using connectives and quantifiers

Examples:

- $Brother(KingJohn, RichardTheLionheart)$
- $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
- $\exists x\ Crown(x) \wedge OnHead(x, John)$

5.3 First-Order Logic Semantics

5.3.1 Models

- Domain: Set of objects in the world
- Interpretation: Maps constants to objects, predicates to relations, functions to functional relations
- Model: Domain + Interpretation

5.3.2 Quantifiers

- Universal quantification: $\forall x\ P(x)$ is true iff $P(x)$ is true for all values of x in the domain
- Existential quantification: $\exists x\ P(x)$ is true iff $P(x)$ is true for at least one value of x in the domain

Properties of quantifiers:

- $\forall x \forall y$ is the same as $\forall y \forall x$
- $\exists x \exists y$ is the same as $\exists y \exists x$
- $\forall x \exists y$ is not the same as $\exists y \forall x$
- Quantifier duality:
  - $\forall x\ P(x) \equiv \neg \exists x\ \neg P(x)$
  - $\exists x\ P(x) \equiv \neg \forall x\ \neg P(x)$

5.4 Inference in First-Order Logic

5.4.1 Universal Instantiation (UI)

$$\frac{\forall v\ \alpha}{\alpha[v/g]}$$

where $\alpha[v/g]$ means α with variable v replaced by ground term g.

Example:

$$\forall x \; King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$

yields

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$

5.4.2 Existential Instantiation (EI)

$$\frac{\exists v \; \alpha}{\alpha[v/k]}$$

where k is a new constant symbol not occurring elsewhere in the KB.

Example:

$$\exists x \; Crown(x) \wedge OnHead(x, John)$$

yields

$$Crown(C_1) \wedge OnHead(C_1, John)$$

where C_1 is a new constant symbol (Skolem constant).

5.4.3 Unification

- Unification finds substitutions that make different logical expressions identical
- θ is a unifier for expressions p and q if $p\theta = q\theta$
- Most general unifier (MGU) is the unifier that makes the fewest commitments

5.4.4 Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', \; (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

where θ is a unifier such that $p_i'\theta = p_i\theta$ for all i.

5.4.5 Forward Chaining for FOL

- Similar to propositional forward chaining
- Uses unification to match rule premises with facts
- Complete for first-order definite clauses

Algorithm 7 Unification Algorithm

function UNIFY(α, β, θ)
    if θ = failure **then return** failure
    else if $\alpha = \beta$ **then return** θ
    else if α is a variable **then return** Unify-Var(α, β, θ)
    else if β is a variable **then return** Unify-Var(β, α, θ)
    else if α and β are compounds **then return** Unify(args(α), args(β), Unify(op(α), op(β), θ))
    else if α and β are lists **then return** Unify(rest(α), rest(β), Unify(first(α), first(β), θ))
    elsereturn failure
    end if
end function

5.4.6 Backward Chaining for FOL

- Similar to propositional backward chaining
- Uses unification to match goals with rule conclusions
- Forms the basis for logic programming (e.g., Prolog)

5.4.7 Resolution for FOL

- Generalizes propositional resolution
- Requires conversion to Conjunctive Normal Form
- Uses unification to match complementary literals
- Complete for first-order logic

Converting to CNF

Steps to convert a first-order logic sentence to CNF:

1. Eliminate implications and biconditionals
2. Move negation inwards
3. Standardize variables (each quantifier should use a different variable)
4. Skolemize (eliminate existential quantifiers)
5. Drop universal quantifiers
6. Distribute $\wedge$ over $\vee$

Skolemization

- Replace existentially quantified variables with Skolem functions or constants
- If $\exists x$ is not within the scope of any universal quantifier, replace x with a new constant
- If $\exists x$ is within the scope of universal quantifiers $\forall y_1, \ldots, \forall y_n$, replace x with a new function $f(y_1, \ldots, y_n)$

Example:

$$\forall x\ [\exists y\ Animal(y) \wedge Loves(x, y)] \Rightarrow \exists z\ Loves(z, x)$$

becomes

$$\forall x\ [Animal(F(x)) \wedge Loves(x, F(x))] \Rightarrow Loves(G(x), x)$$

where F and G are Skolem functions.

Chapter 6

Uncertainty and Probabilistic Reasoning

6.1 Motivation

Logical agents have significant limitations:

- Laziness: Too much work to list all possible exceptions to rules
- Theoretical ignorance: Not all domains have complete theories
- Practical ignorance: Not all relevant facts are knowable

6.2 Probability Theory Basics

6.2.1 Probability Model

- Sample space Ω: Set of all possible worlds
- Probability distribution P: Function assigning probabilities to worlds
- Event: Subset of possible worlds
- $P(a)$: Probability of proposition a being true

6.2.2 Axioms of Probability

- $0 \leq P(a) \leq 1$ for any proposition a
- $P(True) = 1$ and $P(False) = 0$
- $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$

6.2.3 Conditional Probability

- $P(a|b)$: Probability of a given that b is true
- Definition: $P(a|b) = \frac{P(a \wedge b)}{P(b)}$ if $P(b) \neq 0$
- Product rule: $P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a)$

6.3 Inference Using Full Joint Distributions

- The joint distribution specifies $P(X_1, X_2, \ldots, X_n)$ for all combinations of values of the random variables
- Any probability query can be answered by summing over the joint distribution
- For query $P(Y|e)$ where e is evidence:

$$P(Y|e) = \alpha \sum_{z} P(Y, z, e)$$

  where z represents all unobserved variables
- Normalization constant $\alpha = \frac{1}{P(e)}$ ensures probabilities sum to 1

6.4 Independence

- Variables X and Y are independent if $P(X|Y) = P(X)$ or equivalently $P(Y|X) = P(Y)$
- Independence can also be expressed as $P(X, Y) = P(X)P(Y)$
- Independence allows decomposing the joint distribution:

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

6.5 Conditional Independence

- Variables X and Y are conditionally independent given Z if $P(X|Y, Z) = P(X|Z)$
- Equivalent forms:
  - $P(Y|X, Z) = P(Y|Z)$
  - $P(X, Y|Z) = P(X|Z)P(Y|Z)$
- Allows more compact representations even when variables are not completely independent

6.6 Bayes' Rule

- Derived from the product rule:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

- Useful for computing diagnostic probabilities from causal probabilities
- E.g., $P(disease|symptoms) = \frac{P(symptoms|disease)P(disease)}{P(symptoms)}$

6.7 Naive Bayes Models

- Structure: Cause $\rightarrow$ Effects
- Assumes effects are conditionally independent given the cause:

$$P(Cause, Effect_1, \ldots, Effect_n) = P(Cause)\prod_{i=1}^{n} P(Effect_i|Cause)$$

- Total number of parameters is linear in n
- Widely used for classification despite simplistic assumptions

6.8 Bayesian Networks

6.8.1 Structure

- Directed acyclic graph representing dependencies among variables
- Each node corresponds to a random variable
- Each node X_i has a conditional probability distribution $P(X_i|Parents(X_i))$
- The network represents the joint distribution as:

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

6.8.2 Example Bayesian Network

6.8.3 Advantages of Bayesian Networks

- Compact representation: Only need to specify conditional probabilities for each node given its parents
- For n Boolean variables:

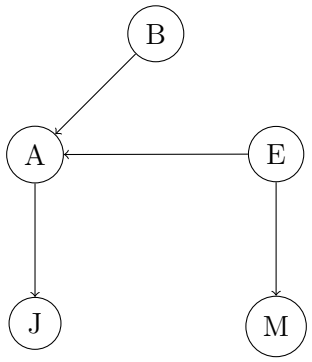


Figure 6.1: Example Bayesian network with 5 variables

  - Full joint distribution: $2^n - 1$ parameters
  - Bayesian network: If each node has at most k parents, only $n \cdot 2^k$ parameters

- Makes independence assumptions explicit in the graph structure

6.9 Inference in Bayesian Networks

6.9.1 Exact Inference

- Variable elimination algorithm:
  - Eliminate hidden variables one by one
  - For each variable, create a factor (conditional probability function)
  - Compute new factors by summing out variables
- Complexity depends on graph structure:
  - Singly connected networks (polytrees): $O(n \cdot d^k)$ where d is domain size and k is maximum number of parents
  - Multiply connected networks: NP-hard in general

6.9.2 Approximate Inference

- Direct sampling: Generate samples from the joint distribution and count
- Rejection sampling: Generate samples and reject those inconsistent with evidence
- Likelihood weighting: Fix evidence variables, sample other variables, weight by likelihood of evidence

- Markov Chain Monte Carlo (MCMC): Generate samples by randomly changing one variable at a time

Chapter 7

Machine Learning

7.1 Introduction to Machine Learning

Machine learning allows systems to improve from experience without being explicitly programmed.

7.1.1 When to Use Machine Learning

- Problems where humans don't know how to program a solution
- Problems where solutions need to adapt to changing environments
- Problems with large amounts of data containing implicitly encoded information

7.1.2 Components of Learning

Learning can be characterized by three key components:

- Task (T): What the system is trying to do
- Performance measure (P): How success is evaluated
- Experience (E): Data the system uses to improve

7.2 Learning Paradigms

7.2.1 Supervised Learning

- Learn a function from labeled examples (input-output pairs)
- Classification: Output is discrete (categories)
- Regression: Output is continuous (numbers)

7.2.2 Unsupervised Learning

- Learn patterns from unlabeled data
- Clustering: Group similar examples
- Density estimation: Find distribution of data
- Dimensionality reduction: Represent data using fewer features

7.2.3 Reinforcement Learning

- Learn by interacting with an environment
- Receive rewards or penalties for actions
- Goal: Learn policy that maximizes expected reward

7.3 Hypothesis Space and Learning Algorithms

7.3.1 Hypothesis Space

- The set of all possible functions/models the learning algorithm can represent
- Examples: Linear functions, decision trees, neural networks
- The choice of hypothesis space imposes an inductive bias

7.3.2 Empirical Risk Minimization

- Find hypothesis $h \in H$ that minimizes error on training data
- Empirical error: $E_{emp}(h) = \frac{1}{m} \sum_{i=1}^{m} L(h(x_i), y_i)$
- Loss function L measures disagreement between prediction and true output

7.4 Model Complexity and Generalization

7.4.1 Overfitting and Underfitting

- Underfitting: Model is too simple to capture patterns in data
- Overfitting: Model fits training data too well, captures noise
- Bias-variance tradeoff:
  - High bias: Model makes strong assumptions, tends to underfit
  - High variance: Model is very flexible, tends to overfit

7.4.2 VC Dimension

- Vapnik-Chervonenkis dimension: Measure of complexity for hypothesis class
- The largest number of points that can be shattered (classified in all possible ways) by the hypothesis class
- Higher VC dimension means more complex models

7.4.3 Confidence Intervals and Generalization

- True error $E_{true}(h)$ is bounded by empirical error plus confidence interval:

$$E_{true}(h) \leq E_{emp}(h) + \sqrt{\frac{VC(H) \cdot \log \frac{2m}{VC(H)} + \log \frac{4}{\delta}}{m}}$$

  with probability $1 - \delta$, where m is the number of training examples

7.4.4 Structural Risk Minimization

- Balance between empirical error and model complexity
- Choose hypothesis minimizing sum of empirical error and complexity penalty
- Regularization implements this principle by adding complexity penalty to loss function

7.5 Machine Learning in Practice

7.5.1 Dataset Splitting

- Training set: Used to learn model parameters
- Validation set: Used for hyperparameter tuning and model selection
- Test set: Used for final evaluation of model performance

7.5.2 Model Selection

- Choose model type (e.g., decision tree, neural network)
- Select hyperparameters (e.g., depth of tree, learning rate)
- Use cross-validation to estimate performance

7.5.3 Data Preprocessing

- Feature scaling: Normalize or standardize features
- Feature selection: Choose most relevant features
- Feature engineering: Create new features from existing ones

7.6 Neural Networks and Deep Learning

7.6.1 Artificial Neuron

- Basic computational unit in neural networks
- Given inputs $x_1, x_2, \ldots, x_n$ and weights $w_1, w_2, \ldots, w_n$:

$$net = \sum_{i=0}^{n} w_i x_i$$

  where $x_0 = 1$ and w_0 is the bias
- Output: $o = \sigma(net)$ where σ is an activation function
- Common activation functions:
  - Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
  - ReLU: $\sigma(x) = \max(0, x)$
  - Tanh: $\sigma(x) = \tanh(x)$

7.6.2 Feedforward Neural Networks

- Arranged in layers: input layer, hidden layers, output layer
- Each neuron connected to all neurons in adjacent layers
- Forward phase: Compute output for given input
- Information flows only in forward direction

7.6.3 Backpropagation Algorithm

- Learning algorithm for neural networks
- Uses gradient descent to minimize error
- Forward phase: Compute outputs of all neurons
- Backward phase: Compute gradients of error with respect to weights
- Update weights: $w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$

7.6.4 Deep Learning

- Neural networks with many hidden layers
- Automatically learns hierarchical representations
- Key innovations:
  - Better activation functions (ReLU)
  - Efficient initialization
  - Regularization techniques (dropout, batch normalization)
  - Advanced optimization methods
  - GPU acceleration

7.6.5 Convolutional Neural Networks (CNNs)

- Specialized architecture for processing grid-like data (images)
- Key components:
  - Convolutional layers: Apply filters to detect features
  - Pooling layers: Reduce spatial dimensions
  - Fully connected layers: Perform final classification
- Translation invariance: Detect features regardless of position
- Parameter sharing: Same filter applied throughout the image

7.6.6 Transformers

- Self-attention mechanism: Focus on relevant parts of input
- Parallelizable computation (unlike RNNs)
- Architecture:
  - Multi-head self-attention
  - Position-wise feed-forward networks
  - Residual connections and layer normalization
- Encoder-decoder structure or decoder-only (GPT) or encoder-only (BERT)
- Foundation for modern language models

Chapter 8

Reinforcement Learning

8.1 Introduction to Reinforcement Learning

Reinforcement learning is about learning by interaction with an environment to achieve a goal.

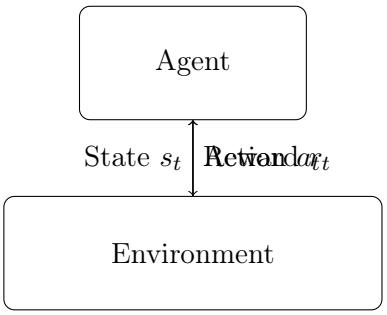


Figure 8.1: The agent-environment interaction in reinforcement learning

8.2 Markov Decision Processes (MDPs)

8.2.1 MDP Formulation

An MDP is defined by:

- Set of states S
- Set of actions A
- Transition model $P(s'|s,a)$
- Reward function $R(s,a,s')$
- Discount factor $\gamma \in [0,1)$

8.2.2 The Goal in MDPs

Find a policy $\pi : S \to A$ that maximizes expected cumulative discounted reward:

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

8.3 Value Functions

8.3.1 State-Value Function

The value of state s under policy π:

$$V^\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right]$$

8.3.2 Action-Value Function (Q-Function)

The value of taking action a in state s under policy π:

$$Q^\pi(s,a) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\right]$$

8.3.3 Bellman Equations

Recursive relationship for value functions:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$$
$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')]$$

8.3.4 Optimal Value Functions

The optimal value functions define the best possible performance:

$$V^*(s) = \max_\pi V^\pi(s) = \max_a Q^*(s,a)$$
$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

8.3.5 Optimal Policy

Once we have Q^*, the optimal policy is:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

8.4 Dynamic Programming Methods

8.4.1 Policy Evaluation

Compute V^π for a given policy π:

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$$

8.4.2 Policy Improvement

Improve the policy based on the current value function:

$$\pi'(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$$

8.4.3 Policy Iteration

Alternate between policy evaluation and policy improvement until convergence.

8.4.4 Value Iteration

Directly compute V^* without explicitly representing the policy:

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$$

8.5 Model-Free Learning

8.5.1 Monte Carlo Learning

- Learn directly from episodes of experience
- Update value estimates only at end of episode
- No bootstrapping, no need for model
- Every-visit MC:

$$V(s) \leftarrow V(s) + \alpha[G_t - V(s)]$$

  where $G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}$ is the return from time t

8.5.2 Temporal Difference Learning

- Learn from incomplete episodes using bootstrapping
- TD(0) update:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

- TD error: $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$

8.6 Q-Learning

8.6.1 Q-Learning Algorithm

- Off-policy TD control algorithm
- Learns Q^* directly without policy evaluation
- Update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- Convergence guaranteed in deterministic environments if all state-action pairs are visited infinitely often

Algorithm 8 Q-Learning Algorithm

```
Initialize Q(s, a) for all s ∈ S, a ∈ A arbitrarily
for each episode do
    Initialize state s
    for each step of episode do
        Choose action a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe reward r and next state s'
        Q(s, a) ← Q(s, a) + α[r + γ max_a' Q(s', a') − Q(s, a)]
        s ← s'
    end for
end for
```

8.6.2 Exploration vs. Exploitation

- Exploration: Try new actions to discover better policies
- Exploitation: Use current knowledge to maximize reward
- Common strategies:
  - ϵ-greedy: Choose best action with probability $1-\epsilon$, random action with probability ϵ
  - Softmax: Choose actions with probability proportional to their estimated values
  - Optimistic initialization: Initialize Q-values optimistically to encourage exploration

8.7 Function Approximation

8.7.1 Linear Function Approximation

- Represent Q-function as a linear function of features:

$$Q(s, a, \vec{w}) = \vec{w}^T \vec{\phi}(s, a)$$

- Update rule:

$$\vec{w} \leftarrow \vec{w} + \alpha[r + \gamma \max_{a'} Q(s', a', \vec{w}) - Q(s, a, \vec{w})] \nabla_{\vec{w}} Q(s, a, \vec{w})$$

8.7.2 Deep Q-Networks (DQN)

- Use neural networks to approximate Q-function
- Key innovations:
  - Experience replay: Store transitions and train on random batches
  - Target network: Separate network for generating targets
- Loss function:

$$L(\vec{w}) = E[(r + \gamma \max_{a'} Q(s', a', \vec{w}^-) - Q(s, a, \vec{w}))^2]$$

  where $\vec{w}^-$ are parameters of the target network

8.8 Policy Gradient Methods

8.8.1 Policy Parameterization

- Directly parameterize the policy:

$$\pi(a|s, \vec{\theta}) = P(a|s, \vec{\theta})$$

- For continuous actions, often use Gaussian policy:

$$\pi(a|s, \vec{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a - \mu(s, \vec{\theta}))^2}{2\sigma^2}\right)$$

8.8.2 Policy Gradient Theorem

$$\nabla_{\vec{\theta}} J(\vec{\theta}) = E_{\pi_{\vec{\theta}}}[\nabla_{\vec{\theta}} \log \pi(a|s, \vec{\theta}) Q^{\pi}(s, a)]$$

8.8.3 REINFORCE Algorithm

- Monte Carlo policy gradient method
- Update rule:

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha G_t \nabla_{\vec{\theta}} \log \pi(a_t|s_t, \vec{\theta})$$

- High variance, often combined with baseline for variance reduction

8.9 Advanced Topics

8.9.1 Deep Reinforcement Learning

- Combines deep learning with reinforcement learning
- Successfully applied to:
  - Games (AlphaGo, AlphaZero)
  - Robotics
  - Resource management

8.9.2 Multi-Agent Reinforcement Learning

- Multiple agents learning simultaneously
- Challenges:
  - Non-stationary environment (other agents are learning)
  - Coordination vs. competition
  - Credit assignment

8.9.3 Hierarchical Reinforcement Learning

- Decompose complex tasks into hierarchies of subtasks
- Options framework: Temporally extended actions
- Helps with exploration and transfer learning

Chapter 9

Natural Language Processing

9.1 Introduction to NLP

Natural Language Processing (NLP) is the field focused on enabling computers to understand, interpret, and generate human language.

9.1.1 Applications of NLP

- Machine translation
- Question answering
- Text summarization
- Sentiment analysis
- Speech recognition
- Conversational agents (chatbots)

9.2 Language Models

9.2.1 N-gram Models

- Model sequences of words using Markov assumption
- Probability of word depends only on n-1 previous words
- Unigram: P(w)
- Bigram: $P(w_i|w_{i-1})$
- Trigram: $P(w_i|w_{i-2}, w_{i-1})$

- Maximum likelihood estimation:

$$P(w_i|w_{i-1}) = \frac{Count(w_{i-1}, w_i)}{Count(w_{i-1})}$$

9.2.2 Smoothing

- Addresses zero probability problem for unseen n-grams
- Laplace (add-one) smoothing:

$$P(w_i|w_{i-1}) = \frac{Count(w_{i-1}, w_i) + 1}{Count(w_{i-1}) + V}$$

  where V is vocabulary size
- Backoff models: Fall back to lower-order n-grams for unseen sequences
- Interpolation: Combine different order n-grams

$$P(w_i|w_{i-2}, w_{i-1}) = \lambda_3 P(w_i|w_{i-2}, w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_1 P(w_i)$$

9.2.3 Evaluation: Perplexity

- Measures how well a probability model predicts a sample
- Lower perplexity means better model

$$Perplexity(W) = P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}}$$

9.3 Text Classification

9.3.1 Naive Bayes Classifier

- Applies Bayes' rule with conditional independence assumption

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \propto P(c) \prod_{i=1}^{n} P(x_i|c)$$

  where d is a document with features $x_1, \ldots, x_n$ and c is a class
- Used for spam detection, sentiment analysis, topic classification

9.3.2 Bag-of-Words Model

- Represents text as unordered collection of words
- Ignores grammar and word order
- Each document becomes a vector of word counts or frequencies
- Simple but effective for many classification tasks

9.4 Word Embeddings

9.4.1 Distributional Semantics

- "You shall know a word by the company it keeps" (Firth, 1957)
- Words with similar contexts have similar meanings
- Words represented as dense vectors in continuous space

9.4.2 Word2Vec

- Learn word vectors by predicting context (Skip-gram) or predicting word from context (CBOW)
- Captures semantic and syntactic relationships
- Vector arithmetic works: king - man + woman $\approx$ queen

9.4.3 GloVe (Global Vectors)

- Combines global matrix factorization with local context window methods
- Uses word co-occurrence statistics from corpus
- Learns vectors such that dot product equals log co-occurrence probability

9.5 Parts of Speech and Syntax

9.5.1 Parts of Speech (POS)

- Lexical categories: noun, verb, adjective, adverb, etc.
- Closed-class words: prepositions, determiners, pronouns, etc.
- Open-class words: nouns, verbs, adjectives, etc.
- Penn Treebank tagset: 45 tags (NN, VB, JJ, etc.)

9.5.2 POS Tagging

- Assigning part of speech to each word in text
- Challenges: words can have multiple POS depending on context
- HMM tagging: uses Viterbi algorithm to find most likely tag sequence
- Modern approaches: bidirectional RNNs, transformers

9.5.3 Syntactic Parsing

Constituency Parsing

- Represents sentence structure as nested constituents
- Based on phrase structure grammar
- Context-Free Grammar (CFG) formalism
- Algorithms: CKY, Earley parsing

Dependency Parsing

- Represents direct relationships between words
- Each word (except root) has exactly one head
- Labeled edges indicate grammatical relationships (subject, object, etc.)
- Algorithms: transition-based, graph-based parsing

9.6 Modern NLP with Deep Learning

9.6.1 Subword Models

- Address vocabulary limitations by breaking words into smaller units
- Byte Pair Encoding (BPE): Iteratively merge most frequent character pairs
- WordPiece: Similar to BPE but uses likelihood improvement for merges
- Advantages:
  - Handles out-of-vocabulary words
  - Effective for morphologically rich languages
  - Reduces vocabulary size while maintaining expressiveness

9.6.2 Transformer Architecture

- Introduced in "Attention is All You Need" (Vaswani et al., 2017)
- Key components:
  - Multi-head self-attention
  - Position-wise feed-forward networks
  - Positional encodings

  - Residual connections and layer normalization

- Captures long-range dependencies without recurrence
- Highly parallelizable

9.6.3 Pre-trained Language Models

Decoder-only Models (GPT)

- Generative Pre-trained Transformer
- Unidirectional (left-to-right) attention
- Pre-trained using language modeling objective
- Auto-regressive generation
- Examples: GPT-1, GPT-2, GPT-3, GPT-4

Encoder-only Models (BERT)

- Bidirectional Encoder Representations from Transformers
- Bidirectional attention over all tokens
- Pre-trained using masked language modeling
- Strong performance on understanding tasks
- Examples: BERT, RoBERTa, DeBERTa

Encoder-Decoder Models (T5)

- Text-to-Text Transfer Transformer
- Treats all NLP tasks as text-to-text problems
- Pre-trained using span corruption
- Versatile for both understanding and generation
- Examples: T5, BART, mT5

9.7 Evaluating NLP Systems

9.7.1 Intrinsic Evaluation

- Evaluates system on isolated task
- Metrics for classification: accuracy, precision, recall, F1 score
- Metrics for generation: BLEU, ROUGE, METEOR
- Metrics for language modeling: perplexity

9.7.2 Extrinsic Evaluation

- Evaluates system as part of larger application
- Measures real-world performance
- Example: evaluating a POS tagger by its impact on a parser
- More meaningful but more expensive

Chapter 10

Computer Vision

10.1 Introduction to Computer Vision

Computer vision is the field that enables computers to gain high-level understanding from digital images or videos.

10.1.1 Why Computer Vision is Useful

- Images and videos constitute a vast portion of digital data (approximately 90% of web data)
- Enables machines to extract meaningful information from visual inputs
- Applications span numerous domains:
  - Medical imaging and diagnostics
  - Autonomous vehicles and robotics
  - Security and surveillance
  - Augmented and virtual reality
  - Content retrieval and organization
  - Manufacturing quality control
  - Human-computer interaction
- Growing importance with the advent of deep learning and access to large visual datasets

10.2 Challenges in Computer Vision

Computer vision is inherently difficult due to several fundamental challenges:

10.2.1 Variability in Appearance

- **Lighting conditions**: Same object appears different under varying illumination
- **Viewpoint changes**: Objects look different when viewed from different angles
- **Scale variations**: Objects can appear at different sizes in images
- **Intra-class variation**: Objects within the same category can differ significantly
- **Deformations**: Non-rigid objects can change shape
- **Occlusions**: Objects may be partially hidden by other objects

10.2.2 Semantic Gap

- Bridging the gap between low-level pixel data and high-level semantic understanding
- Computers see pixels; humans see meaning
- The challenge of moving from "what" (detection) to "why" (interpretation)

10.2.3 Computational Challenges

- Processing high-resolution images and videos requires significant computational resources
- Real-time vision applications demand efficient algorithms
- Balancing accuracy with computational efficiency

10.3 Image Formation and Representation

10.3.1 Image Formation Process

- **Pinhole camera model**: The simplest model of image formation
  - Light rays pass through a small aperture (pinhole) and project onto an image plane
  - Creates an inverted image of the scene
- **Lens-based cameras**: Gather more light, but introduce distortions

- **Perspective projection**: 3D world points are mapped to 2D image points

$$x' = f\frac{X}{Z}, \quad y' = f\frac{Y}{Z} \tag{10.1}$$

where (X, Y, Z) is a 3D point, (x', y') is its 2D projection, and f is the focal length

10.3.2 Digital Image Representation

- **Grayscale images**: Single channel, intensity values typically in range [0, 255]
  - Represented as a 2D matrix of intensity values
  - $I(x, y)$ gives the intensity at position (x, y)
- **Color images**: Multiple channels, typically RGB (Red, Green, Blue)
  - Represented as a 3D tensor with dimensions height × width × channels
  - Each pixel has three values representing color components
- **Image as a function**: $f : [a, b] \times [c, d] \to [0, 255]$ or $[0, 1]$

10.4 Traditional Computer Vision Approaches

10.4.1 Filtering and Convolution

- **Convolution operation**: Basic building block of many vision algorithms

$$(f * h)[n, m] = \sum_{k,l} f[k, l] \cdot h[n - k, m - l] \tag{10.2}$$

where f is the image, h is the kernel, and $*$ denotes convolution

- **Linear filters**: Weighted sum of pixel values in a neighborhood
  - Gaussian filter: Blurring/smoothing to reduce noise
  - Box filter: Simple averaging filter
  - Sobel filter: Edge detection

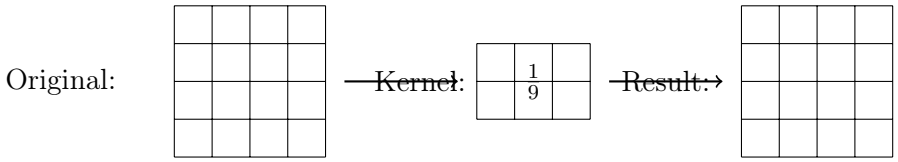


Figure 10.1: Illustration of image convolution with a 3×3 averaging kernel.

10.4.2 Edge Detection

- **Edges**: Significant changes in image intensity that often correspond to object boundaries
- **First-order edge detection**: Based on image gradients
  - Sobel operator: Approximates the gradient magnitude and direction
  - Prewitt operator: Similar to Sobel but with different kernel weights
- **Second-order edge detection**: Based on the Laplacian (second derivative)
  - Laplacian of Gaussian (LoG): Combines Gaussian smoothing with Laplacian
  - Difference of Gaussians (DoG): Approximation of LoG
- **Canny edge detector**: Multi-stage algorithm for robust edge detection
  1. Gaussian smoothing to reduce noise
  2. Gradient computation using Sobel filters
  3. Non-maximum suppression to thin edges
  4. Hysteresis thresholding to eliminate weak edges

10.4.3 Feature Extraction

- **Local features**: Distinctive points in the image that are useful for matching and recognition
- **Scale Invariant Feature Transform (SIFT)**:
  - Detects keypoints that are invariant to scale, rotation, and moderate illumination changes
  - Process:
    1. Scale-space extrema detection using Difference of Gaussians

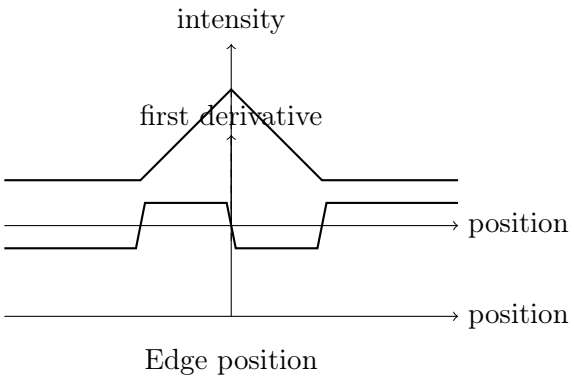


Figure 10.2: Edge detection using first derivatives. The edge corresponds to a peak in the first derivative of the intensity function.

2. Keypoint localization and filtering
3. Orientation assignment based on local gradients
4. Keypoint descriptor computation (128-dimensional vector)

- **Other feature detectors/descriptors**:
  - SURF: Speeded-Up Robust Features, faster alternative to SIFT
  - ORB: Oriented FAST and Rotated BRIEF, efficient binary descriptor
  - BRIEF: Binary Robust Independent Elementary Features
  - FAST: Features from Accelerated Segment Test, corner detector

10.5 Traditional Object Recognition Paradigms

10.5.1 Bag of Visual Words

- Inspired by the Bag of Words model in text processing
- Process:
  1. Extract local features (e.g., SIFT) from training images
  2. Cluster features to create a "visual vocabulary" (codebook)
  3. Represent each image as a histogram of visual word occurrences
  4. Train a classifier (e.g., SVM) on these histogram representations
- Advantages: Simple, rotation and scale-invariant
- Disadvantages: Ignores spatial relationships between features

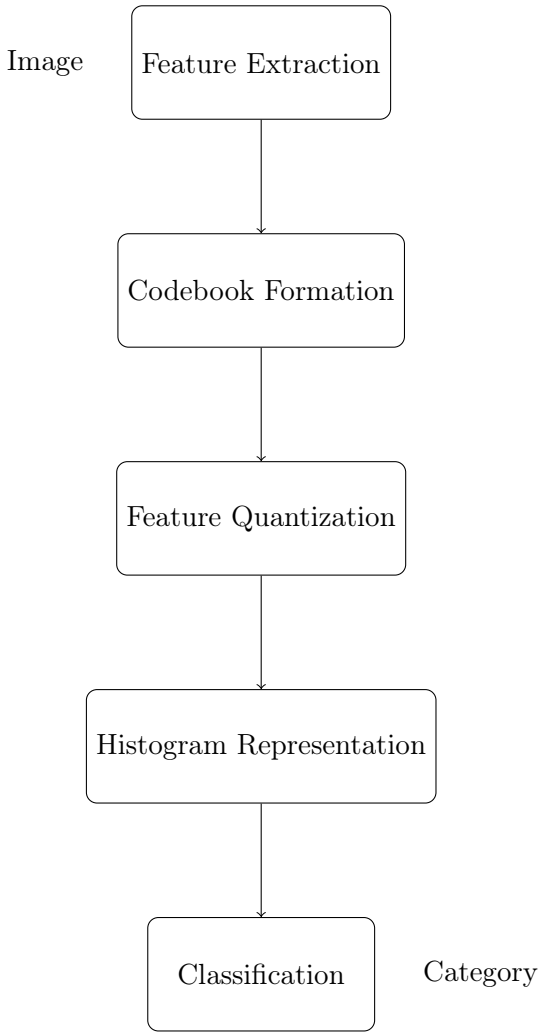


Figure 10.3: Bag of Visual Words pipeline for object recognition.

10.5.2 Part-Based Models

- Represent objects as collections of parts with spatial relationships
- **Deformable Part Models (DPM)**:
  - Object represented by a root filter and a set of part filters
  - Parts can move relative to the root with deformation costs
  - Detection uses dynamic programming to find optimal part configurations
- Advantages: Capture structural relationships, handle pose variations
- Disadvantages: Complex model, computationally intensive

10.5.3 Template Matching

- Slide a template over the image and measure similarity
- Similarity measures: Sum of Squared Differences (SSD), Normalized Cross-Correlation (NCC), etc.
- Advantages: Simple, intuitive
- Disadvantages: Sensitive to scale, rotation, and appearance variations

10.6 Deep Learning Approaches

10.6.1 Convolutional Neural Networks (CNNs)

- Specialized neural networks for processing grid-like data (e.g., images)
- Key components:
  - **Convolutional layers**: Apply filters to learn spatial hierarchies of features
  - **Pooling layers**: Downsample feature maps to reduce spatial dimensions
  - **Activation functions**: Introduce non-linearity (typically ReLU)
  - **Fully connected layers**: Perform final classification based on extracted features
- Properties:
  - Local connectivity: Neurons connect to a local region of the input
  - Parameter sharing: Same filter applied throughout the input
  - Translation invariance: Detect features regardless of their position

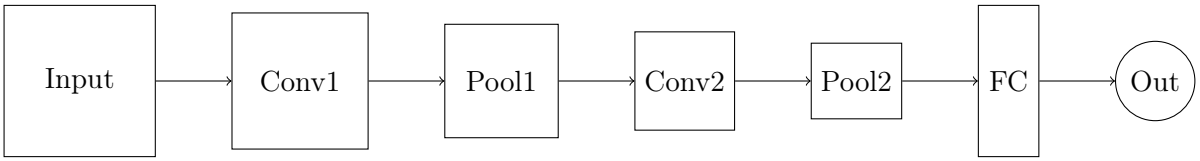


Figure 10.4: Typical architecture of a Convolutional Neural Network (CNN).

10.6.2 Popular CNN Architectures

- **AlexNet** (2012): First CNN to win ImageNet challenge, 8 layers
- **VGG** (2014): Deeper networks with small 3×3 filters, 16-19 layers
- **GoogLeNet/Inception** (2014): Introduced inception modules with parallel convolutions
- **ResNet** (2015): Introduced residual connections to enable much deeper networks (up to 152 layers)
- **MobileNet** (2017): Lightweight architecture for mobile and embedded devices
- **EfficientNet** (2019): Systematically scales depth, width, and resolution for optimal performance

10.6.3 Tasks in Computer Vision

- **Image classification**: Assign a label to an entire image
- **Object detection**: Locate and classify multiple objects in an image
  - R-CNN family: Region-based CNNs
  - YOLO: You Only Look Once, single-pass detection
  - SSD: Single Shot MultiBox Detector
- **Semantic segmentation**: Classify each pixel in the image
  - FCN: Fully Convolutional Networks
  - U-Net: Encoder-decoder architecture with skip connections
- **Instance segmentation**: Detect and segment individual object instances
  - Mask R-CNN: Extension of Faster R-CNN with mask prediction

10.7 Vision Transformers

10.7.1 From CNNs to Transformers

- CNNs dominated computer vision for nearly a decade
- Transformer architecture (originally for NLP) adapted for vision in 2020
- Advantages over CNNs:

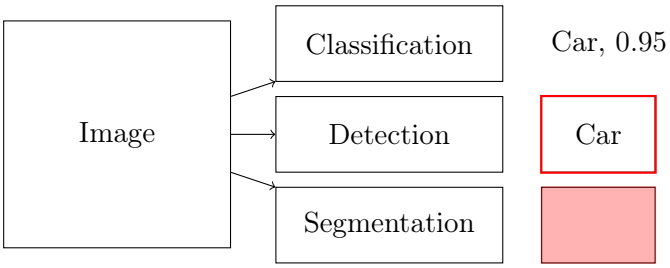


Figure 10.5: Major computer vision tasks: classification, detection, and segmentation.

  - Global receptive field from the start
  - Better at capturing long-range dependencies
  - More scalable with data and model size

10.7.2 Vision Transformer (ViT) Architecture

- **Image tokenization**:
  - Divide image into fixed-size patches (e.g., 16×16)
  - Flatten patches and linearly embed them into tokens
  - Add position embeddings to preserve spatial information
  - Prepend a special [CLS] token for classification
- **Transformer encoder**:
  - Stack of transformer blocks
  - Each block contains multi-head self-attention and MLP layers
  - Layer normalization and residual connections around each block
- **Classification head**:
  - MLP applied to the [CLS] token representation

10.7.3 Recent Developments

- **Hierarchical Vision Transformers**:
  - Swin Transformer: Shifted windows for efficient attention
  - PVT: Pyramid Vision Transformer with multi-scale feature maps
- **Hybrid architectures**:
  - ConvNeXt: CNN design inspired by transformers

  - CoAtNet: Combines convolution and self-attention

- **Self-supervised learning**:
  - DINO: Self-distillation with no labels
  - MAE: Masked Autoencoders for visual pre-training

10.8 Evaluation Metrics in Computer Vision

10.8.1 Classification Metrics

- **Accuracy**: Proportion of correct predictions
- **Precision**: True positives / (True positives + False positives)
- **Recall**: True positives / (True positives + False negatives)
- **F1 Score**: Harmonic mean of precision and recall
- **Top-k Accuracy**: Correct if true class is among the k highest-probability predictions

10.8.2 Detection and Segmentation Metrics

- **Intersection over Union (IoU)**: Overlap between predicted and ground truth regions
- **Average Precision (AP)**: Area under the Precision-Recall curve at specific IoU thresholds
- **Mean Average Precision (mAP)**: Mean of AP across classes and/or IoU thresholds
- **Pixel Accuracy**: Percentage of correctly classified pixels
- **Mean IoU**: Average IoU across all classes

10.9 Challenges and Future Directions

10.9.1 Current Challenges

- **Robustness**: Models can be sensitive to adversarial examples, domain shifts, and unusual scenes
- **Explainability**: Deep models often function as black boxes, making it difficult to understand their decisions
- **Few-shot learning**: Learning from limited examples remains challenging

- **Computational efficiency**: State-of-the-art models require significant computational resources
- **3D understanding**: Moving beyond 2D to comprehensive 3D scene understanding

10.9.2 Future Directions

- **Multimodal learning**: Integrating vision with language, audio, and other modalities
- **Self-supervised learning**: Reducing reliance on labeled data
- **Neural architecture search**: Automatically discovering optimal model architectures
- **Neuro-symbolic approaches**: Combining neural networks with symbolic reasoning
- **Edge AI**: Deploying computer vision on resource-constrained devices
- **Foundation models for vision**: Large-scale pre-trained models adaptable to many tasks

10.10 Applications of Computer Vision

10.10.1 Healthcare

- Medical image analysis (X-rays, CT scans, MRIs)
- Disease diagnosis and prognosis
- Surgical planning and assistance
- Monitoring patient condition

10.10.2 Autonomous Systems

- Self-driving vehicles
- Robot navigation and manipulation
- Drone surveillance and delivery
- Industrial automation

10.10.3 Security and Surveillance

- Facial recognition
- Anomaly detection
- Object tracking
- Biometric authentication

10.10.4 Augmented and Virtual Reality

- Scene understanding
- Object recognition and tracking
- Hand and body pose estimation
- Spatial mapping

10.10.5 Retail and E-commerce

- Visual search
- Virtual try-on
- Inventory management
- Customer behavior analysis

10.11 Conclusion

Computer vision has evolved dramatically over the past decades, from handcrafted features and models to deep learning approaches that learn representations directly from data. While significant progress has been made, the field continues to face challenges in creating systems that truly understand visual content at a human level. The integration of computer vision with other AI disciplines, particularly through multimodal learning, represents a promising direction toward more comprehensive visual intelligence.

As hardware capabilities improve and algorithms become more efficient, computer vision applications will continue to proliferate across industries, transforming how we interact with technology and enhancing our ability to solve complex problems. The future of computer vision lies not just in improved accuracy on benchmark datasets, but in developing systems that can reason about the visual world, understand context, and make decisions that are explainable, fair, and robust.

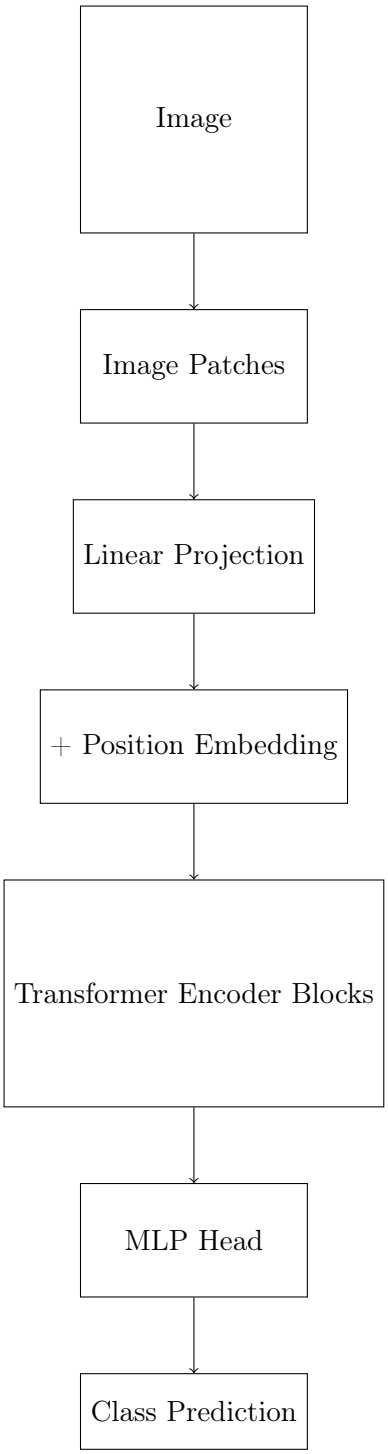


Figure 10.6: Vision Transformer (ViT) architecture.

Chapter 11

Constraint Satisfaction Problems

11.1 Introduction to CSPs

Constraint Satisfaction Problems (CSPs) represent a class of problems where the goal is to find a state that satisfies a set of constraints.

11.1.1 Definition

A CSP is formally defined by:

- A set of variables $X = \{X_1, X_2, \ldots, X_n\}$
- Domains for each variable $D = \{D_1, D_2, \ldots, D_n\}$ where D_i is the set of possible values for X_i
- A set of constraints $C = \{C_1, C_2, \ldots, C_m\}$ that restrict the values variables can take simultaneously

A solution to a CSP is an assignment of values to all variables such that all constraints are satisfied.

11.1.2 Examples of CSPs

- **Map Coloring**: Assign colors to regions such that no adjacent regions have the same color
- **N-Queens**: Place N queens on an N×N chessboard such that no queen threatens another
- **Sudoku**: Fill a 9×9 grid with digits 1-9 such that each column, row, and 3×3 box contains all digits

- **Scheduling**: Assign resources (rooms, teachers) to tasks (classes) subject to constraints
- **Cryptarithmetic**: Find digit substitutions that make a mathematical equation true

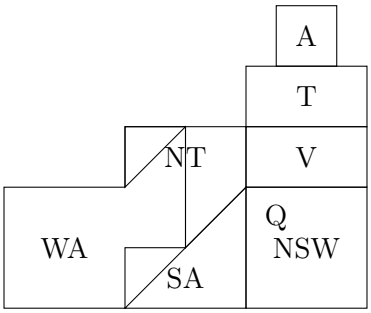


Figure 11.1: Example of a map coloring problem. Adjacent regions must have different colors.

11.1.3 Types of Constraints

- **Unary constraints**: Involve a single variable, e.g., $X_1 \neq \text{red}$
- **Binary constraints**: Involve pairs of variables, e.g., $X_1 \neq X_2$
- **Higher-order constraints**: Involve three or more variables
- **Preference constraints (soft constraints)**: Express desirability rather than requirements

11.1.4 Constraint Graphs

For binary CSPs, we can represent the problem structure as a constraint graph:

- Nodes represent variables
- Edges connect variables that participate in a constraint

The structure of this graph can provide insights into the complexity of the problem and suggest efficient solution strategies.

11.2 Backtracking Search for CSPs

11.2.1 Basic Backtracking Algorithm

Backtracking search is a depth-first search that assigns values to variables one at a time, backtracking when a variable has no legal values left to assign.

Algorithm 9 Backtracking Search for CSPs

```
1: function BACKTRACK(assignment, csp)
2:     if assignment is complete then return assignment
3:     end if
4:     var ← Select-Unassigned-Variable(csp, assignment)
5:     for all value in Order-Domain-Values(csp, var, assignment) do
6:         if value is consistent with assignment then
7:             add {var = value} to assignment
8:             result ← Backtrack(assignment, csp)
9:             if result ≠ failure then return result
10:            end if
11:            remove {var = value} from assignment
12:        end if
13:    end forreturn failure
14: end function
```

11.2.2 Improving Backtracking Efficiency

Several heuristics can significantly improve backtracking performance:

Variable Selection Heuristics

- **Minimum Remaining Values (MRV)**: Choose the variable with the fewest legal values
- **Degree Heuristic**: Choose the variable involved in the most constraints with unassigned variables (as a tie-breaker)

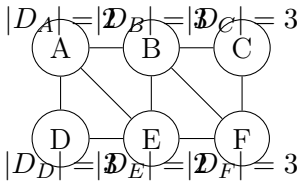


Figure 11.2: Example constraint graph with domain sizes. MRV would select variable A or E.

Value Ordering Heuristics

- **Least Constraining Value**: Choose the value that rules out the fewest choices for neighboring variables

11.2.3 Constraint Propagation

Constraint propagation reduces the search space by using constraints to eliminate inconsistent values:

Forward Checking

- When a variable is assigned, check immediate constraints and remove inconsistent values from neighboring unassigned variables
- Terminates search when any variable has no legal values

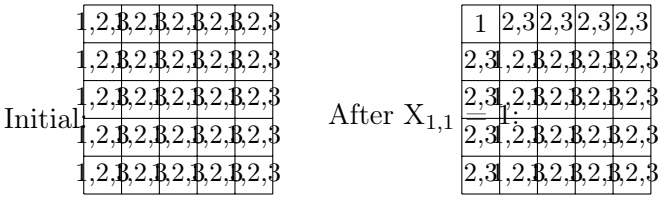


Figure 11.3: Forward checking in a CSP. After assigning 1 to $X_{1,1}$, the value 1 is removed from the domains of variables in the same row and column.

Arc Consistency (AC-3)

- A constraint is arc consistent if for every value in the domain of one variable, there exists a consistent value in the domain of the other variable
- Ensures that every variable has values consistent with each binary constraint

11.3 Problem Structure and Decomposition

11.3.1 Tree-Structured CSPs

- If the constraint graph forms a tree (no loops), the CSP can be solved in $O(nd^2)$ time
- Algorithm:
  1. Choose a variable as root, order variables from root to leaves
  2. For j = n down to 2, apply RemoveInconsistent(Parent(X_j), X_j)
  3. For j = 1 to n, assign X_j consistent with Parent(X_j)

Algorithm 10 AC-3 Algorithm

```
1: function AC-3(csp)
2:     queue ← all arcs in csp
3:     while queue is not empty do
4:         (X_i, X_j) ← Remove-First(queue)
5:         if Remove-Inconsistent-Values(X_i, X_j) then
6:             for each X_k in Neighbors(X_i) - {X_j} do
7:                 add (X_k, X_i) to queue
8:             end for
9:         end if
10:    end while
11: end function
12: function REMOVE-INCONSISTENT-VALUES(X_i, X_j)
13:    removed ← false
14:    for each x in Domain[X_i] do
15:        if no value y in Domain[X_j] satisfies constraint(X_i=x, X_j=y)
    then
16:            delete x from Domain[X_i]
17:            removed ← true
18:        end if
19:    end forreturn removed
20: end function
```

11.3.2 Nearly Tree-Structured CSPs

- Many real-world problems are "almost" trees
- Can be solved efficiently using techniques like:
  - **Cutset conditioning**: Instantiate a subset of variables such that the remaining constraint graph is a tree
  - **Tree decomposition**: Decompose the constraint graph into interconnected subproblems that form a tree

11.4 Local Search for CSPs

11.4.1 Min-Conflicts Algorithm

- Start with a complete assignment (possibly with conflicts)
- Iteratively select a variable involved in conflicts and reassign it to minimize conflicts

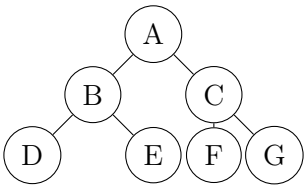


Figure 11.4: A tree-structured constraint graph. This can be solved efficiently in linear time.

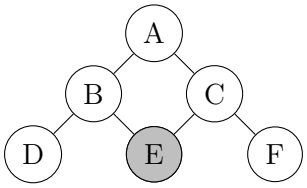


Figure 11.5: Almost tree-structured constraint graph. Conditioning on variable E (shaded) turns the graph into a tree.

11.4.2 Applications to N-Queens

- For n-queens, min-conflicts can find solutions for extremely large n (e.g., n = 1,000,000)
- Success due to the fact that the state space has relatively few local minima

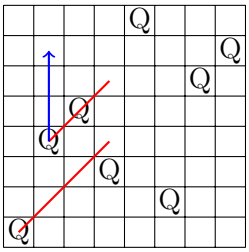


Figure 11.6: Min-conflicts applied to 8-queens. The conflicted queen in column 2 is moved to minimize conflicts.

11.4.3 Local Search for Optimization Problems

Many CSPs are actually optimization problems:

- **Constraint optimization problems**: Maximize an objective function while satisfying hard constraints

Algorithm 11 Min-Conflicts Algorithm

```
1: function MIN-CONFLICTS(csp, max-steps)
2:     current ← initial complete assignment for csp
3:     for i = 1 to max-steps do
4:         if current is a solution then return current
5:         end if
6:         var ← a randomly selected conflicted variable in current
7:         value ← the value v for var that minimizes conflicts
8:         set var = value in current
9:     end forreturn failure
10: end function
```

- **Weighted CSPs**: Each constraint has a weight; goal is to minimize the sum of weights of violated constraints

Local search techniques like simulated annealing, genetic algorithms, and tabu search can be effective for such problems.

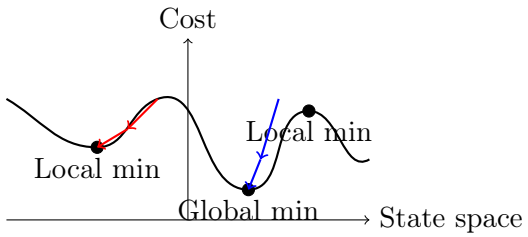


Figure 11.7: Local search in a cost landscape with multiple minima. Red arrows show hill climbing to a local minimum. Blue arrows show a pathway to the global minimum.

Chapter 12

Multimodal Large Language Models

12.1 Introduction to Multimodal Learning

12.1.1 From Unimodal to Multimodal AI

Traditional AI systems focused on single modalities (text, vision, audio), but human intelligence integrates multiple sensory inputs and outputs. Multimodal AI aims to bridge this gap by processing and generating information across different modalities.

- **Unimodal**: Systems that handle a single type of input/output (e.g., text-only or image-only)
- **Multimodal**: Systems that process, reason with, and generate across multiple modalities (text, images, audio, video)

12.1.2 Challenges in Multimodal Learning

- **Representation alignment**: Different modalities have different statistical properties and semantics
- **Fusion**: How to effectively combine information from different modalities
- **Translation**: Converting information from one modality to another
- **Alignment**: Determining which parts of different modalities correspond to each other
- **Co-learning**: Using knowledge from one modality to improve learning in another

12.1.3 Multimodal Applications

- **Visual question answering**: Answering questions about images
- **Image captioning**: Generating textual descriptions of images
- **Text-to-image generation**: Creating images from textual descriptions
- **Visual reasoning**: Making inferences about visual scenes
- **Multimodal chatbots**: Conversational agents that can discuss visual content
- **Video understanding**: Comprehending actions and events in videos

12.2 Building Blocks of Multimodal Models

12.2.1 CLIP: Contrastive Language-Image Pre-training

CLIP is a neural network trained on a variety of image-text pairs from the internet, establishing strong visual-semantic connections.

- **Architecture**: Consists of an image encoder and a text encoder trained jointly
- **Training objective**: Contrastive learning to maximize similarity between matching image-text pairs and minimize for non-matching pairs
- **Key innovation**: Zero-shot transfer to new visual classification tasks without specific training

12.2.2 Diffusion Models

Diffusion models are a class of generative models that learn to generate data by gradually denoising a Gaussian noise pattern.

- **Forward process**: Gradually add noise to an image until it becomes pure noise
- **Reverse process**: Learn to remove noise step by step to generate an image
- **Conditioning**: Can be conditioned on text, images, or other modalities
- **Applications**: Text-to-image generation (DALL·E, Stable Diffusion), image editing, super-resolution

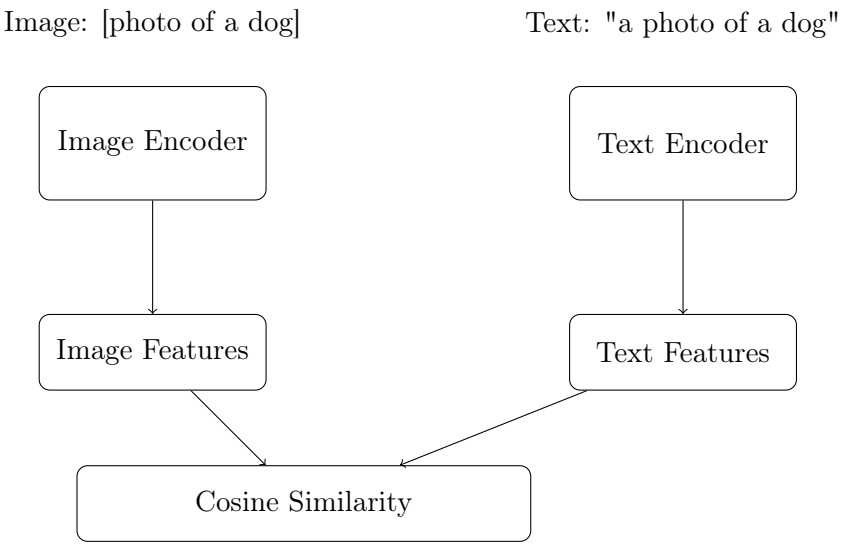


Figure 12.1: Simplified architecture of CLIP, showing contrastive learning between image and text encoders.

12.2.3 One for All (OFA)

OFA is a unified multimodal model that handles various tasks with a single sequence-to-sequence architecture.

- **Unified architecture**: Single model for multiple tasks across modalities
- **Task formulation**: All tasks formulated as sequence-to-sequence problems
- **Capabilities**: Image understanding, text understanding, image generation, cross-modal tasks

12.3 Efficient Multimodal Models

12.3.1 BLIP-2: Bootstrapping Language-Image Pre-training

BLIP-2 is an efficient approach for multimodal learning that bridges pre-trained vision and language models through a lightweight Querying Transformer.

- **Architecture**:

Diffusion Model: Forward process (adding noise) and reverse process (denoising)

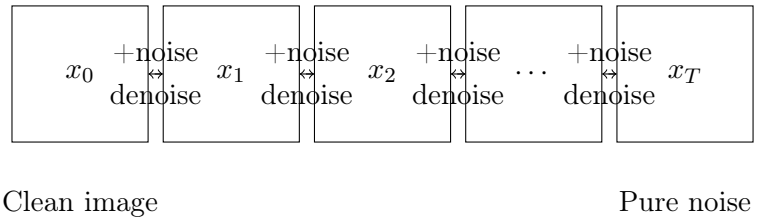


Figure 12.2: Illustration of the diffusion process, showing forward noising and reverse denoising steps.

  - Frozen image encoder (typically a vision transformer)
  - Frozen large language model
  - Trainable Q-Former (Querying Transformer) that connects them
- **Advantages**:
  - Parameter-efficient: Only trains the connecting module
  - Leverages strong pre-trained models without modifying them
  - Reduces computational costs significantly

12.3.2 Training Objectives in BLIP-2

- **Image-Text Contrastive Learning**: Align image and text representations
- **Image-Grounded Text Generation**: Generate captions conditioned on images
- **Image-Text Matching**: Predict whether image-text pairs match

12.3.3 Efficiency Comparisons

12.4 MLLM Architectures

12.4.1 Architectural Paradigms

There are several approaches to building Multimodal Large Language Models (MLLMs):

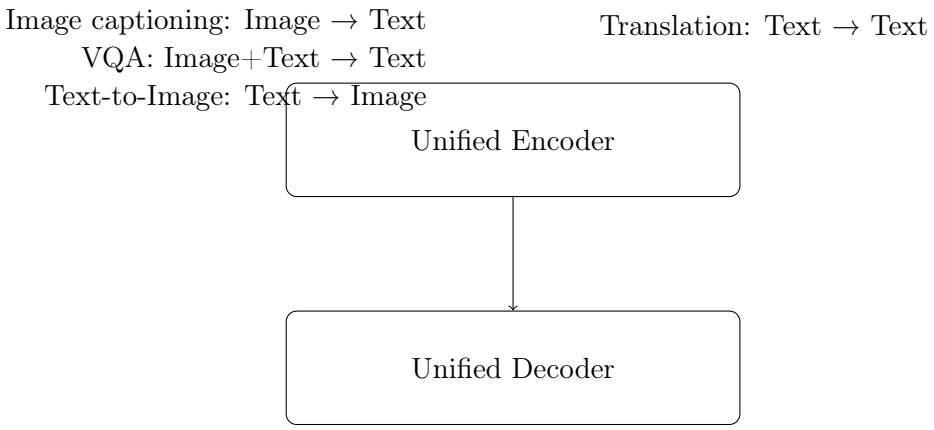


Figure 12.3: OFA's unified sequence-to-sequence architecture for multimodal tasks.

Model	Trainable Parameters	COCO Caption	VQA
Full fine-tuning	1-7B	High	High
BLIP-2	100-200M	High	High
Adapter-based	10-50M	Medium	Medium

Table 12.1: Comparison of different multimodal training strategies by efficiency and performance.

- **End-to-end integration**: Train a single model on multimodal data from scratch
- **Modular composition**: Combine specialized models for different modalities
- **Projection-based**: Project non-text modalities into the language model's embedding space

12.4.2 LLM as Discrete Controller

One effective approach uses LLMs as controllers or orchestrators:

- **LLM-as-a-controller**: LLM decides which specialized models to call and when
- **Tool use**: Visual modules are accessed as tools by the LLM
- **Benefits**: Modularity, reusability, easier debugging

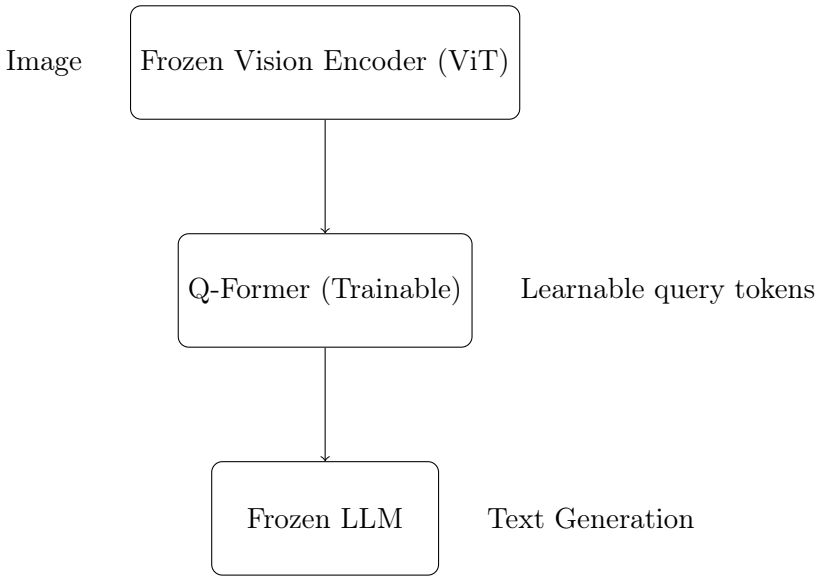


Figure 12.4: BLIP-2 architecture with frozen image encoder, trainable Q-Former, and frozen language model.

12.4.3 LLM as Joint Part of System

Alternative approach where the LLM is more deeply integrated with other modalities:

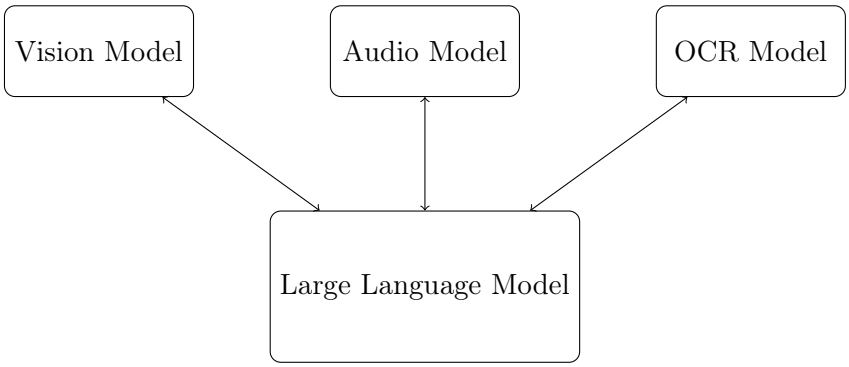
- **Projectors**: Map other modalities into LLM's token space
- **LLM as backbone**: Other modalities feed into the LLM's layers
- **Cross-attention**: Allow interaction between modalities at different levels

12.5 Image Tokenization and Processing

12.5.1 Methods for Image Tokenization

In order for LLMs to process images, they must be converted into a format compatible with text tokens:

- **Grid-based**: Divide image into grid cells and encode each cell
- **Patch-based**: Split image into patches and encode each patch (used in vision transformers)
- **Object-based**: Detect objects and tokenize their features
- **Visual vocabulary**: Learn a discrete codebook of visual tokens



LLM as a controller for specialized modules

Figure 12.5: LLM as a discrete controller architecture, coordinating specialized perceptual models.

12.5.2 Challenges in Image Tokenization

- **Sequence length**: Images can generate thousands of tokens, challenging LLM context windows
- **Information density**: Visual and linguistic information have different densities
- **Spatial relationships**: Preserving spatial information in sequence format
- **Interleaving**: How to effectively interleave image and text tokens

12.5.3 Tokenization in Leading MLLMs

Model	Method	Approach
GPT-4V	Projection	Proprietary visual encoding projected to language space
LLaVA	CLIP+Projection	CLIP embeddings projected to language model embeddings
Flamingo	Perceiver	Cross-attention between visual features and language model
ImageBind	Joint embedding	Map multiple modalities to a shared embedding space

Table 12.2: Image tokenization approaches in different multimodal language models.

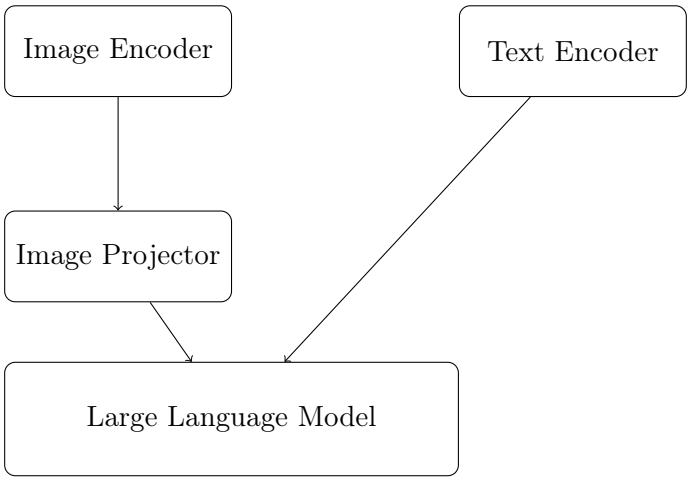


Figure 12.6: Joint architecture where visual features are projected into the language model's space.

12.6 Multimodal Instruction Tuning

12.6.1 From Pre-training to Instruction Tuning

- **Pre-training**: General capability acquisition on large-scale data
- **Instruction tuning**: Aligning the model to follow human instructions
- **Multimodal instruction tuning**: Teaching models to follow instructions involving multiple modalities

12.6.2 Creating Multimodal Instruction Datasets

- **Human annotation**: Experts create instruction-response pairs with images
- **Synthetic data generation**: Using existing models to generate instruction-response pairs
- **Data augmentation**: Variations of instructions for the same image
- **Multi-turn conversations**: Instruction datasets with follow-up questions

12.6.3 LLaVA: Large Language and Vision Assistant

LLaVA is a representative example of instruction-tuned multimodal models:

- **Architecture**: CLIP visual encoder + projection layer + instruction-tuned LLM

- **Training process**:
  1. Pre-train on image-text pairs to learn visual-language connection
  2. Instruction-tune on multimodal instruction-response pairs
  3. Optionally, human preference alignment with RLHF
- **Capabilities**: Visual understanding, reasoning, conversation about images

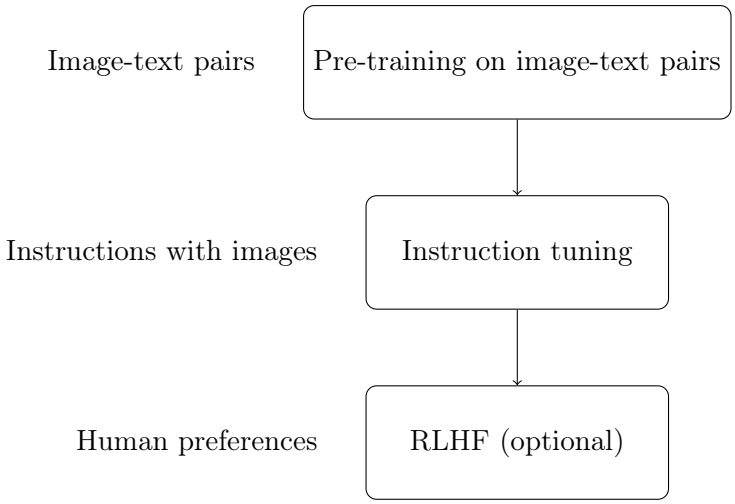


Figure 12.7: LLaVA training process, showing progression from pre-training to instruction tuning.

12.7 Future Directions and Challenges

12.7.1 Expanding Modalities

- **Audio integration**: Speech, sounds, music understanding
- **Video understanding**: Dynamic visual content over time
- **3D and physical world**: Understanding spatial relationships and physics
- **Tactile and other sensory**: Incorporating touch, smell, taste representations

12.7.2 Key Challenges

- **Hallucinations**: MLLMs can generate plausible but false information about visual content

- **Grounding**: Ensuring models' responses are actually grounded in visual inputs
- **Evaluation**: Developing robust benchmarks for multimodal capabilities
- **Cross-modal reasoning**: Integrating information across modalities for complex reasoning
- **Efficiency**: Making multimodal models computationally efficient
- **Bias and fairness**: Addressing biases in multimodal systems

12.7.3 Applications and Impact

- **Accessibility**: Making digital content accessible to diverse users
- **Education**: Multimodal learning aids and tutoring systems
- **Creative tools**: Assisting in content creation across modalities
- **Healthcare**: Medical image analysis with explanations
- **Robotics**: Grounding language instructions in physical world

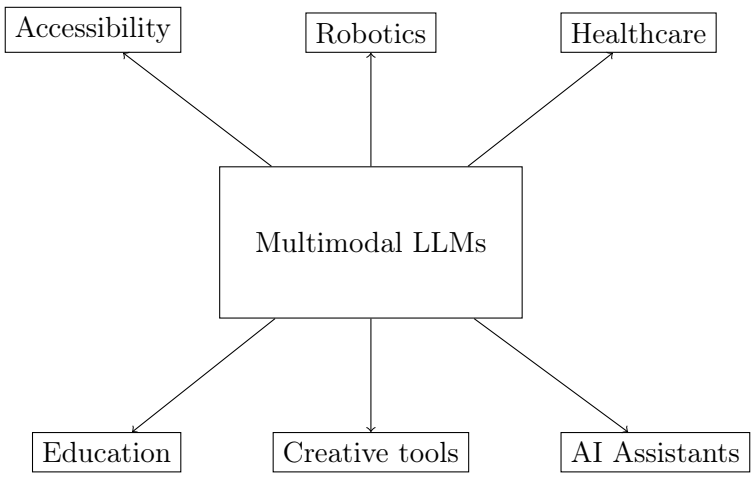


Figure 12.8: Potential application areas of multimodal large language models.

Bibliography

[1] Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.

[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[3] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

[4] Jurafsky, D., & Martin, J. H. (2023). *Speech and Language Processing* (3rd ed. draft). Retrieved from https://web.stanford.edu/~jurafsky/slp3/

[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998-6008).

[6] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning* (pp. 8748-8763).

[7] Li, J., Li, D., Savarese, S., & Hoi, S. (2023). BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International Conference on Machine Learning*.

[8] Liu, P., Yuan, W., Fu, J., et al. (2022). OFA: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International Conference on Machine Learning*.

[9] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*.

[10] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*.

[11] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

[12] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 1-67.

[13] Liu, H., Li, C., Wu, Q., & Lee, Y. J. (2023). Visual instruction tuning. In *Advances in Neural Information Processing Systems*.

[14] Deng, C., Liu, X., et al. (2024). A Survey on Multimodal Large Language Models. *arXiv preprint arXiv:2306.13549*.