Neural Networks and Machine Learning

From Computational Models to Biological Intelligence

Your Name

Department of Computer Science

University Name

September 17, 2025

Contents

1 Introduction to Machine Learning and Neural Networks **1**

1.1 Introduction to Machine Learning 1

1.1.1 What is Learning? 1

1.1.2 What is Reasoning? 1

1.1.3 From Animal Learning to Machine Learning 1

1.1.4 Types of Reasoning: Comprehensive Overview 2

1.1.5 Reasoning Capabilities Across Intelligence Types 6

1.1.6 Limitations of Inductive Reasoning 7

1.1.7 Inductive Bias in Machine Learning 8

1.1.8 Inductive Bias in Specific Architectures 10

1.1.9 Machine Learning vs Other Approaches 11

1.1.10 Symbolic AI vs Machine Learning 12

1.1.11 Role of Prior Knowledge 13

1.1.12 Key Takeaways 14

1.2 Neural Networks: From Computation to Biology 15

1.2.1 What is Computation? 15

1.2.2 Computational Models: Theoretical Foundations 16

1.2.3 Four Fundamental Computational Models 18

1.2.4 Modern Computing Systems: From Theory to Practice 20

1.2.5 Biological Neural Networks: Nature's Computational Model 21

1.2.6 Other Biological Computation Models 25

1.2.7 Comparative Analysis of Computational Models 26

1.2.8 Key Insights and Takeaways 27

1.2.9 The Journey from Computation to Intelligence 28

1.2.10 Artificial Neural Networks 29

List of Figures

1.1 Structure of a biological neuron 22
1.2 Cortical layers organization 24
1.3 Abstract Neuron Architecture 30
1.4 Taylor Series Neural Network: $F(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \ldots + a_n(x - x_0)^n$ 44
1.5 Fourier Series Neural Network 45

List of Tables

1.1 Reasoning Capabilities Across Intelligence Types 6
1.2 Machine Learning vs Statistics 12
1.3 Symbolic AI vs Machine Learning Comparison 13
1.4 Cortical Layers and Their Functions 24
1.5 Key Differences Between Computational Models 26
1.6 Modern Technology Mapping 27
1.7 Computing Models Comparison 29

Chapter 1

Introduction to Machine Learning and Neural Networks

1.1 Introduction to Machine Learning

1.1.1 What is Learning?

Learning is the process of acquiring new knowledge, skills, or behaviors through experience. This process transforms inputs—such as data, experiences, or information—into useful capabilities like expertise, new skills, or predictive models.

Key Questions in Learning

- What are the essential inputs for the learning process?
- How do we measure the effectiveness and success of learning?
- What are the underlying mechanisms and processes by which learning occurs?

1.1.2 What is Reasoning?

Reasoning is the ability to draw logical conclusions from known facts or learned knowledge. Unlike learning, reasoning relies on logical inference rather than large amounts of data.

1.1.3 From Animal Learning to Machine Learning

Example: Bait Shyness in Rats

Rats demonstrate a fundamental learning principle through their feeding behavior:

- They sample novel food cautiously
- If the food causes illness, they avoid it in the future

- Past experience directly informs future decisions

This natural learning process parallels challenges in machine learning.

Parallel: Spam Email Filtering

Consider how this biological learning principle applies to spam detection:

- **Naive approach**: Memorize all past spam emails
- **Problem**: Cannot classify previously unseen emails
- **Solution**: Extract generalizable patterns (like words, phrases, or sender patterns)
- **Key insight**: Both rats and spam filters must generalize from specific experiences to handle new, similar situations

This ability to generalize leads us to examine the different types of reasoning that enable learning and decision-making.

1.1.4 Types of Reasoning: Comprehensive Overview

1. Inductive Reasoning

Definition: Inductive reasoning extracts patterns from observed data to make predictions about future or unseen cases. This approach moves from specific observations to general conclusions, yielding probable rather than certain results.

Key Characteristics:

- Most prevalent form of reasoning in the animal kingdom and primary mode in machine learning
- Forms the basis of most learned behaviors in animals
- Used extensively in deep learning and LLMs

Examples Across Different Contexts **Human Example**: Every cat I have ever seen has four legs. Therefore, all cats have four legs.

Animal Example:

- A dog learns that when its owner picks up the leash, it will probably go for a walk (experienced hundreds of times)
- A squirrel learns that acorns are edible after eating many without getting sick

Machine Learning Example: A spam classifier learns from previously labeled emails and generalizes patterns to detect new spam messages

LLM Example: When asked to complete "The sky is blue because...", the model has "observed" this pattern countless times in training data and induces probable completions based on statistical patterns

Applications in AI/ML

- Deep learning model training
- Pattern recognition systems
- Predictive analytics
- Natural language processing

2. Deductive Reasoning

Definition: Deductive reasoning moves from general rules and premises to reach specific, guaranteed conclusions. It starts with a general rule and a specific case to reach a logical conclusion.

Key Characteristics:

- Provides certainty when premises are true
- Animals generally lack this capability for abstract reasoning
- LLMs can only mimic this through pattern matching

Examples Across Different Contexts Human Example: All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

Mathematical Example: If all squares have four sides and a shape is a square, it must have four sides.

Animal Limitation: Animals cannot perform abstract syllogistic reasoning, such as deducing that "because all felines are carnivores and a tiger is a feline, then a tiger is a carnivore."

LLM Mimicry: When given "All mammals have hair. A dolphin is a mammal. Therefore...", the model completes with "a dolphin has hair"—not through true logical syllogism, but by recognizing learned textual patterns from training data.

Applications in AI

- Expert systems (traditional AI)
- Symbolic reasoning systems
- Theorem proving
- Rule-based systems

Limitations

- LLMs lack strict logical reasoning capabilities
- Most modern AI systems don't use true deductive reasoning

3. Abductive Reasoning (Inference to Best Explanation)

Definition: Abductive reasoning starts with an observation and seeks to find the simplest and most likely explanation. It's the process of finding a hypothesis that, if true, would best explain the observation.

Key Characteristics

- Often described as "inference to the best explanation"
- Guesses the most probable explanation given incomplete data
- Can be demonstrated in simple forms by animals
- Simulated effectively by modern LLMs

Examples Across Different Contexts **Human Example**: The grass is wet. A plausible explanation is that it rained (most likely, though sprinklers are possible).

Medical Example: A doctor observes symptoms like fever and cough and infers the patient likely has the flu.

Animal Example:

- A squirrel hears rustling and sees movement, "abduces" it's a predator and climbs a tree
- A raven sees a human place a rock over food, infers the food is under the rock when human leaves

LLM Example: When asked "Why is the road wet?", generates explanations like "It rained," "Water main broke," or "Street cleaner passed" by ranking probable explanations from training data.

Applications in AI/ML

- Medical diagnosis systems
- Troubleshooting AI
- Natural language understanding
- Creative writing and content generation

In Machine Learning Systems Hypothesis Generation: ML models generate predictions based on learned patterns - they present the most probable explanation without "knowing" it's true.

Bayesian Inference: AI systems use Bayesian models to start with prior beliefs and update them as new evidence is presented.

4. Analogical Reasoning (Pattern Transfer)

Definition: Analogical reasoning applies knowledge from one context to another by recognizing similar patterns or relationships.

Applications in AI/ML

- AI-powered tutoring systems
- Cross-domain learning
- Transfer learning in neural networks

Example AI that learns human speech patterns in English and transfers that learning to generate speech in another language.

5. Bayesian Reasoning (Probabilistic Prediction)

Definition: Bayesian reasoning uses probability to predict outcomes by updating beliefs based on new evidence.

Applications in AI/ML

- Spam filtering systems
- AI language models
- Uncertainty quantification

Example A Bayesian spam filter assigns probabilities to words appearing in spam emails and calculates the likelihood that a new email is spam.

6. Causal Reasoning (Understanding Cause-and-Effect)

Definition: Causal reasoning determines causal relationships rather than just correlations.

Limitations in Current AI

- LLMs struggle with true causality
- Most AI systems identify correlations rather than causes

Example In healthcare, researchers identify that smoking causes lung cancer, rather than just observing that smokers have higher cancer rates.

7. Counterfactual Reasoning (What-If Thinking)

Definition: Counterfactual reasoning explores hypothetical scenarios and alternative possibilities.

Applications in AI

- Risk analysis systems
- AI decision-making
- Simulation and planning

Example A self-driving car AI simulates different driving scenarios to decide the safest course of action in an emergency.

1.1.5 Reasoning Capabilities Across Intelligence Types

Reasoning Type	**Animals**	**Large Language Models**	**Traditional AI**
Inductive	Primary (survival-focused)	Primary (pattern-based)	Limited
Deductive	Absent (complex forms)	Simulated (pattern matching)	Primary (rule-based)
Abductive	Limited (simple forms)	Effective (learned patterns)	Limited
Causal	Basic	Limited	Rule-dependent
Adaptability	High (within domain)	High (pattern recognition)	Low (manual updates)

Table 1.1: Reasoning Capabilities Across Intelligence Types

While inductive reasoning is powerful, it has inherent limitations that both animals and AI systems must navigate.

1.1.6 Limitations of Inductive Reasoning

Pigeon Superstition Experiment (B.F. Skinner)

This experiment demonstrated how animals can form false associations through inductive reasoning.

Garcia & Koelling Experiment (1966)

This landmark experiment studied **selective associative learning** in rats and demonstrated that **not all stimuli are equally associated** with consequences.

Experimental Design:

Researchers used a compound stimulus approach:

- **Taste component**: Saccharin-flavored water
- **Audiovisual component**: Lights and sounds during drinking

Rats were then exposed to different aversive consequences:

- **Group 1**: Illness (nausea from mild radiation or toxin)
- **Group 2**: Physical discomfort (mild electric shocks)

Results:

Illness-Induced Group:

- Developed strong aversion to taste cues (saccharin water)
- Showed minimal aversion to audiovisual cues

Shock-Induced Group:

- Developed strong aversion to audiovisual cues (lights and sounds)
- Showed no aversion to taste cues

Key Finding: Rats selectively associated specific stimuli with appropriate consequences—taste with illness, external cues with physical danger.

Scientific Impact:

This experiment revolutionized learning theory by:

- **Challenging equipotentiality**: Not all stimulus-response associations are equally learnable
- **Demonstrating biological constraints**: Evolution shapes what animals can easily learn
- **Revealing adaptive biases**: Learning mechanisms evolved to enhance survival
  - Taste naturally links to internal consequences (poisoning)
  - External cues (sounds, lights) link to external threats (predators)

Implications for Machine Learning:

- **Learning requires inductive bias**: Not all associations are equally learnable
- **Feature relevance varies**: Some inputs are more informative than others
- **Domain knowledge matters**: Evolutionary or expert-designed constraints improve learning
- **No universal learner exists**: All learning algorithms must make assumptions (No-Free-Lunch theorem)

These biological insights directly inform machine learning design, where inductive bias plays a crucial role in model performance.

1.1.7 Inductive Bias in Machine Learning

What is Inductive Bias?

Definition: Inductive bias refers to the set of assumptions that a learning algorithm makes to generalize from limited training data to unseen data.

Why is it Critical?

Inductive bias is essential because:

- Machine learning models have limited training data
- Models must generalize from past observations to unseen cases
- Without appropriate bias, models may overfit (memorizing training data without learning generalizable patterns)

Types of Inductive Biases

1. Preference for Simpler Models (Occam's Razor)

- **Assumption**: Simpler explanations are preferred over complex ones
- **Example**: Decision trees with fewer splits are preferred because they generalize better
- **In Deep Learning**: Regularization techniques (L_1, L_2) penalize complex models

2. Smoothness Assumption

- **Assumption**: Data points that are close together should have similar outputs
- **Example**: In image classification, two similar images should belong to the same class
- **In ML**: K-Nearest Neighbors (KNN) assumes nearby data points have the same label

3. Similar Features Should Have Similar Effects

- **Assumption**: If two features are related, their effects should be similar
- **Example**: In linear regression, correlated features often have similar coefficients

4. Prior Knowledge About the Task (Domain-Specific Bias)

- **Assumption**: Certain relationships are more likely in specific tasks
- **Example**: In NLP, word order matters
- **In ML**: Transformers use positional embeddings to capture sentence structure

5. Invariance Bias (Translation, Rotation, Scale Invariance)

- **Assumption**: Some transformations should not change predictions
- **Example**: Rotating an image of a cat should still classify it as a cat
- **In ML**: CNNs use convolutional filters to enforce translation invariance

6. Sparsity Assumption

- **Assumption**: Only a few features are truly important
- **Example**: In text classification, most words are irrelevant
- **In ML**: L_1 regularization forces models to select important features

These general principles manifest differently across various neural network architectures, each designed with specific inductive biases for particular tasks.

1.1.8 Inductive Bias in Specific Architectures

Convolutional Neural Networks (CNNs)

CNNs are designed for image processing and rely on three key inductive biases:

1. Locality Bias (Local Connectivity)

- **Assumption**: Nearby pixels are more relevant than distant pixels
- **Example**: In facial recognition, CNN detects eyes, nose, mouth before recognizing entire face

2. Translation Invariance

- **Assumption**: An object should be recognized regardless of position
- **How it works**: CNNs use shared convolutional filters
- **Example**: Handwritten digit "3" recognized anywhere in the image

3. Hierarchical Feature Learning

- **Assumption**: Complex patterns learned by stacking abstraction layers
- **Example**: Lower layers detect edges → middle layers detect shapes → deeper layers detect objects

Recurrent Neural Networks (RNNs & LSTMs)

RNNs are designed for sequential data and rely on:

1. Temporal Dependency Bias

- **Assumption**: Recent information is more important than distant past
- **Example**: In "The cat sat on the mat", nearby words are more related

2. Order Sensitivity Bias

- **Assumption**: The order of input elements matters
- **Example**: "Dog bites man" $\neq$ "Man bites dog"

Transformers (BERT, GPT)

1. Attention-Based Bias (Self-Attention)

- **Assumption**: Important words can be anywhere in a sentence
- **Example**: In "The dog chased the ball...which was blue", "which" refers to "ball"

2. Context-Aware Learning Bias

- **Assumption**: Word meaning depends on context
- **Example**: "Bank" can mean financial institution or riverbank

3. Positional Encoding Bias

- **Assumption**: Order matters even without sequential processing
- **Example**: “She ate an apple” $\neq$ “An apple ate she”

1.1.9 Machine Learning vs Other Approaches

Why Machine Learning?

- For many problems, it’s difficult to program correct behavior by hand
- Examples: recognizing objects in images, understanding human speech
- Machine learning approach: program an algorithm to automatically learn from data

Reasons to Use Learning Algorithms

- Hard to code up a solution by hand (e.g., vision, speech)
- System needs to adapt to changing environment (e.g., spam detection)
- Want system to perform better than human programmers
- Privacy/fairness considerations (e.g., ranking search results)

Machine Learning vs Statistics

Relations to AI

- AI does not always imply a learning-based system
- **Non-learning AI approaches**:
  - Symbolic reasoning

Aspect	Statistics	Machine Learning
Primary Goal	Draw valid conclusions for scientists/policymakers	Build autonomous predictive systems
Focus	Interpretability and mathematical rigor	Predictive performance and scalability
Approach	Hypothesis testing and inference	Pattern recognition and automation

Table 1.2: Machine Learning vs Statistics

  - Rule-based systems
  - Tree search
- **Learning-based systems**:
  - Learned based on data
  - More flexibility
  - Good at solving pattern recognition problems

1.1.10 Symbolic AI vs Machine Learning

What is Symbolic AI?

- Also known as **Good Old-Fashioned AI (GOFAI)**
- Represents knowledge using **symbols, rules, and logic**
- Uses **explicitly programmed rules** for reasoning
- Based on **formal logic, tree search, and knowledge representation**
- **Example**: If "All humans are mortal" and "Socrates is human", then "Socrates is mortal"

Tree Search in Symbolic AI

- Fundamental approach for solving problems by exploring decision trees
- Uses algorithms like DFS, BFS, and A* Search
- **Example**: Chess AI searches possible future board states
- Does **not generalize from data** but computes solutions using logical steps

Rule-Based AI

- Subset of Symbolic AI using **explicit if-then rules**

- Rules manually defined by experts
- **Examples**:
  - IF patient has fever AND cough → Diagnose as flu
  - IF transaction > $10,000 → Flag as potential fraud
- **Limitation**: Struggles with exceptions and uncertain scenarios

Symbolic AI vs Machine Learning Comparison

Feature	Symbolic AI	Machine Learning
Knowledge Source	Rules & logic	Data & patterns
Interpretability	Highly explainable	Often a black box
Adaptability	Rigid (manual updates)	Can generalize from data
Data Requirements	Minimal	Requires large datasets
Best Use Cases	Theorem proving, expert systems	NLP, computer vision, recommendations

Table 1.3: Symbolic AI vs Machine Learning Comparison

Limitations of Symbolic AI

- Difficult to scale—requires manual updates
- Cannot handle unstructured data (images, speech)
- Struggles with uncertainty—rules don't adapt
- Machine Learning outperforms in perception tasks

The Future: Hybrid AI (Neuro-Symbolic AI)

- Combines **Symbolic AI (structured logic)** with **Machine Learning (pattern recognition)**
- **Example**: AI lawyer uses Symbolic AI for legal rules + ML for case history analysis
- Enhances **explainability, adaptability, and reasoning**

1.1.11 Role of Prior Knowledge

The effectiveness of any learning system—biological or artificial—depends heavily on the prior knowledge and assumptions it brings to new situations.

Key Principles:

- **Prior knowledge accelerates learning**: Domain expertise makes learning from examples more efficient
- **Trade-offs exist**: Stronger assumptions improve performance on expected tasks but reduce flexibility for unexpected scenarios
- **Balance is crucial**: The optimal learning system finds the right equilibrium between prior knowledge and data-driven adaptation
- **Tool development matters**: Creating methods to incorporate domain expertise remains central to advancing machine learning theory

1.1.12 Key Takeaways

Fundamental Principles

- **Learning transforms experience into generalizable knowledge** through pattern recognition and abstraction
- **Inductive bias is essential** for effective learning—all successful learning systems require appropriate assumptions about their domain
- **No universal learner exists**—every learning algorithm must make trade-offs based on its intended use case

Reasoning in AI Systems

- **Biological systems** primarily use inductive reasoning, with limited abductive capabilities
- **Traditional AI systems** excel at deductive reasoning but struggle with adaptation
- **Modern ML/LLMs** are powerful at inductive reasoning and can simulate other reasoning types through pattern matching
- **Future AI systems** will likely combine symbolic reasoning with machine learning for enhanced interpretability and robust decision-making

Practical Implications

- **Domain knowledge matters**—incorporating appropriate inductive biases improves learning efficiency and generalization
- **Architecture design is crucial**—different neural network architectures embed specific assumptions about data structure (CNNs for images, Transformers for sequences)

- **Learning and reasoning are complementary**—effective AI systems need both the ability to learn from data and to reason with acquired knowledge

1.2 Neural Networks: From Computation to Biology

1.2.1 What is Computation?

Computation is the process of performing calculations, manipulating data, or executing a sequence of operations to solve problems or transform inputs into desired outputs. It encompasses both the theoretical and practical aspects of processing information.

Core Elements of Computation

Every computational process involves these fundamental components:

1. **Input**: Data or information that enters the system
2. **Processing**: The manipulation or transformation of that data according to specific rules or algorithms
3. **Output**: The result or solution produced by the processing
4. **Algorithm**: The step-by-step procedure or set of rules that defines how the processing should occur
5. **Storage/Memory**: The ability to retain information for use during or after processing

Types of Computation

Mathematical Computation:

- Numerical calculations (arithmetic, calculus, statistics)
- Symbolic manipulation (algebra, logic)
- Optimization problems

Logical Computation:

- Boolean operations (AND, OR, NOT)
- Decision-making processes
- Pattern matching and recognition

Data Processing:

- Sorting and searching
- Data transformation and analysis
- Information retrieval

Algorithmic Computation:

- Following predefined procedures
- Recursive processes
- Iterative methods

1.2.2 Computational Models: Theoretical Foundations

A **computational model** is a mathematical or conceptual framework that defines how computation is carried out. It provides the rules and structure for describing:

- **What can be computed**
- **How it can be computed**
- **The limits of computation**

Elements Common to All Computational Models

1. **Representation of Information (Data Model)**
   Defines how information is stored (symbols, numbers, states, molecules, etc.)
2. **Operators / Rules**
   Defines how information is transformed (functions, state transitions, reactions, etc.)
3. **Control Mechanism**
   Defines the order of operations (sequential steps, parallel updates, probabilistic choices)
4. **Storage / Memory**
   Defines how intermediate information is kept and reused (tape, registers, neuron activations)
5. **Input / Output**
   Defines how external data enters and results leave the system

Elementary Components of Any Computing Model

What are the elementary components of any conceivable computing model? In the theory of general recursive functions, for example, it is possible to reduce any computable function to some composition rules and a small set of primitive functions. For a universal computer, we ask about the existence of a minimal and sufficient instruction set.

For an arbitrary computing model, the following metaphoric expression has been proposed:

$$\textbf{computation} = \textbf{storage} + \textbf{transmission} + \textbf{processing} \tag{1.1}$$

The mechanical computation of a function presupposes that these three elements are present:

- **Storage**: Data can be stored and maintained over time
- **Transmission**: Information can be communicated to the functional units of the model
- **Processing**: Data can be transformed according to computational rules

It is implicitly assumed that a certain **coding** of the data has been agreed upon. Coding plays an important role in information processing because, as Claude Shannon showed in 1948, when noise is present information can still be transmitted without loss, if the right code with the right amount of redundancy is chosen.

How Different Models Implement These Components Modern Computers:

- Transform storage of information into a form of information transmission
- Static memory chips store a bit as a circulating current until the bit is read
- Separation between storage (RAM/disk) and processing (CPU) with data buses for transmission

Turing Machines:

- Store information on an infinite tape
- Transmission is performed by the read-write head moving along the tape
- Processing occurs through state transitions based on current symbol and state

Cellular Automata:

- Store information in each cell

- Each cell acts simultaneously as storage and a small processor
- Transmission occurs through local neighborhood interactions between adjacent cells
- No separation between storage, transmission, and processing - they are unified in each cell

1.2.3 Four Fundamental Computational Models

Turing Machine (Alan Turing, 1936)

The Foundation of Algorithmic Computation

- **Style:** Imperative / mechanical model of computation
- **Core Idea:** A machine reads/writes symbols on an infinite tape with a finite set of rules
- **Representation:**
  - Infinite tape divided into cells
  - Head that can read/write and move left or right
  - Finite state machine controlling transitions
- **Strengths:**
  - Canonical model for algorithmic computability
  - Basis of the Church–Turing Thesis
  - Directly models sequential execution
- **Limitations:**
  - Low-level, not efficient
  - Sequential by nature, doesn't capture parallelism well

Example: A Turing Machine can simulate any algorithm you'd run on a modern computer (given enough tape).

Lambda Calculus (Alonzo Church, 1930s)

The Foundation of Functional Computation

- **Style:** Functional model of computation

- **Core Idea:** Everything is a function. Computation = function application + substitution
- **Representation:**
  - Variables (x)
  - Function definitions (λx.expression)
  - Function application ((f x))
- **Strengths:**
  - Basis of functional programming (Haskell, Lisp)
  - Good for reasoning about higher-order functions, abstraction, recursion
- **Limitations:**
  - Abstract and symbolic; not naturally tied to hardware
  - Efficiency is not modeled, just computability

Example: Addition can be defined entirely in terms of functions (Church numerals).

Cellular Automata (Stanislaw Ulam, John von Neumann, later Conway)

The Foundation of Distributed/Parallel Computation

- **Style:** Spatial / distributed model of computation
- **Core Idea:** Computation arises from simple local rules applied to a grid of cells over time
- **Representation:**
  - Infinite (or finite) grid of cells
  - Each cell has a finite state (e.g., alive/dead)
  - Transition rules depend only on the local neighborhood
- **Strengths:**
  - Good for modeling parallel, distributed, physical systems
  - Supports universal computation (Conway's Life is Turing-complete)
- **Limitations:**
  - Not natural for symbolic or algebraic computation

- More suited for simulating dynamics

Example: Conway's *Game of Life* shows how simple rules produce complex, even universal, behaviors.

Biological Computation (Inspired by Nature)

The Foundation of Adaptive/Learning Computation

Biological models are inspired by living systems and emphasize parallelism, adaptability, and learning.

Key Characteristics:

- **Learns from data** (training) rather than using fixed rules
- Massive parallelism and fault tolerance
- Self-organization and adaptation
- Pattern recognition and generalization

1.2.4 Modern Computing Systems: From Theory to Practice

Central Processing Unit (CPU) - *Based on Turing Machine Model*

- **Representation:** Binary data in registers and memory
- **Operators:** Instruction set (add, move, compare, jump)
- **Control:** Program counter + clock cycle (sequential execution)
- **Storage:** Cache, RAM, hard disk
- **Input/Output:** Peripherals, buses, I/O ports

Graphics Processing Unit (GPU) - *Based on Cellular Automata Model*

- **Representation:** Data stored in many cores and VRAM
- **Operators:** SIMD (single-instruction, multiple-data) kernels running in parallel
- **Control:** Thousands of threads execute simultaneously under scheduling
- **Storage:** VRAM, shared memory on cores
- **Input/Output:** Typically streams of data for graphics or AI workloads

Neural Network Systems - *Based on Biological Computation Model*

- **Representation:** Weights and activations (usually floating-point numbers)
- **Operators:** Linear algebra (matrix multiply, convolution, activation functions)
- **Control:** Layer-by-layer propagation (forward/backward pass)
- **Storage:** Model parameters + learned weights
- **Input/Output:** Input features (image, text) → Output predictions

1.2.5 Biological Neural Networks: Nature's Computational Model

Characteristics of Biological Neural Networks

The following are key characteristics that make biological neural networks powerful computational systems:

- **(a) Highly interconnected:** Neurons form a complex web of connections
- **(b) Robustness and Fault Tolerance:** The decay of nerve cells does not affect the overall function of the network significantly
- **(c) Flexibility:** The ability to reorganize and adapt to new situations
- **(d) Handling incomplete information:** Ability to infer appropriate outputs even when some inputs are missing or noisy
- **(e) Parallel processing:** Multiple neurons can process information simultaneously

Neuron Structure

Components of a Neuron

- **Fundamental unit:** neuron (cell body / soma, dendrites, axon, synapses)
- **Dendrites** receive inputs; **axon** transmits output and branches to many synapses (often thousands)
- **Synapse:** junction between axon terminal and target cell
- **Synaptic junctions** form between presynaptic axon terminals and postsynaptic dendrites or the cell body
- **Typical sizes:**
  - soma $\sim$ 10–80 μm

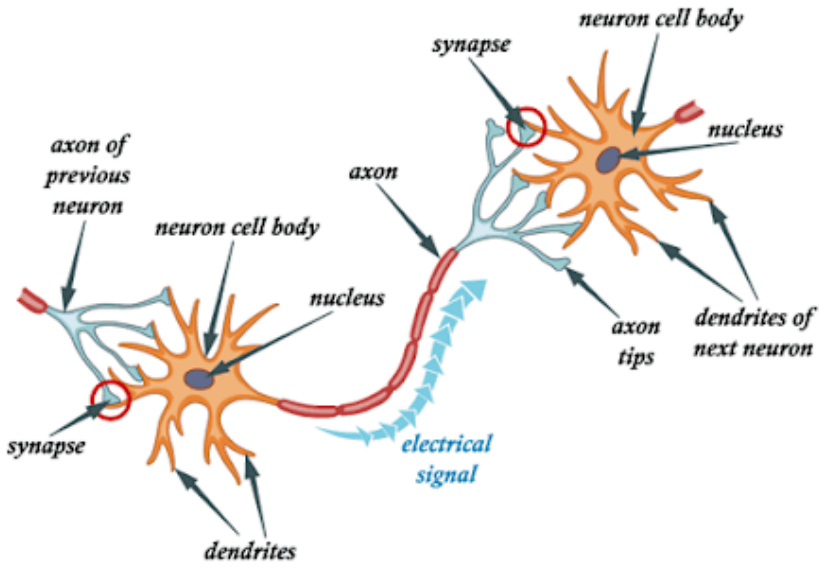

Figure 1.1: Structure of a biological neuron

- synaptic gap $\sim$ 200 nm
- neuron length from 0.01 mm to 1 m

Signal Transmission and Firing

- **Resting potential** $\sim$ -70 mV; depolarization above threshold (roughly $\sim$10 mV) triggers firing
- **Action potentials** are all-or-none pulses sent down the axon; information is encoded in firing rate ($\sim$1–100 Hz)
- **Propagation speed** in brain tissue $\sim$ 0.5–2 m/s; synaptic transmission delay $\sim$ 0.5 ms
- After firing the membrane recovers (**refractory period**); synaptic effects decay with time constant $\sim$5–10 ms

Synapses: Chemistry and Types

- **Transmission** across synapse is chemical: neurotransmitters released from presynaptic terminal
- **Postsynaptic effect** can be excitatory (depolarizing) or inhibitory (hyperpolarizing)
- All endings of a given axon are typically either excitatory or inhibitory
- **Synaptic strength** depends on activity and can change over time (basis for learning)

Plasticity and Learning

- Active synapses that repeatedly contribute to postsynaptic firing tend to strengthen; inactive ones weaken
- **Hebb's rule** ("cells that fire together, wire together") describes this activity-dependent plasticity
- Continuous modification of synaptic strengths underlies learning and memory formation

Network-Scale Properties

- **Convergence/divergence:** neurons receive many inputs and send outputs to many others
- **Average inputs per neuron:** on the order of 10^4 synapses; total synaptic connections in human brain estimated $\sim 10^{15}$
- The cortex contains extremely dense, layered networks with vast numbers of interconnected neurons
- Studying simple, identical units helps understand complex brain functions, but full understanding remains far off

Cortical Layers: Input/Output Organization

The cerebral cortex is organized into six distinct layers, each with specialized roles in processing and routing information. This laminar organization is crucial for understanding how biological neural networks process information hierarchically.

Cortical Layers & Their Input/Output Functions

Functional Specialization by Cortical Area **Sensory Processing Areas** (e.g., visual cortex):

- Have a very prominent Layer IV (strong input) because they process raw sensory data
- Dense thalamic inputs require robust input processing capabilities

Motor Areas:

- Layer V is very prominent (strong output) since these regions send commands out to body/motor systems

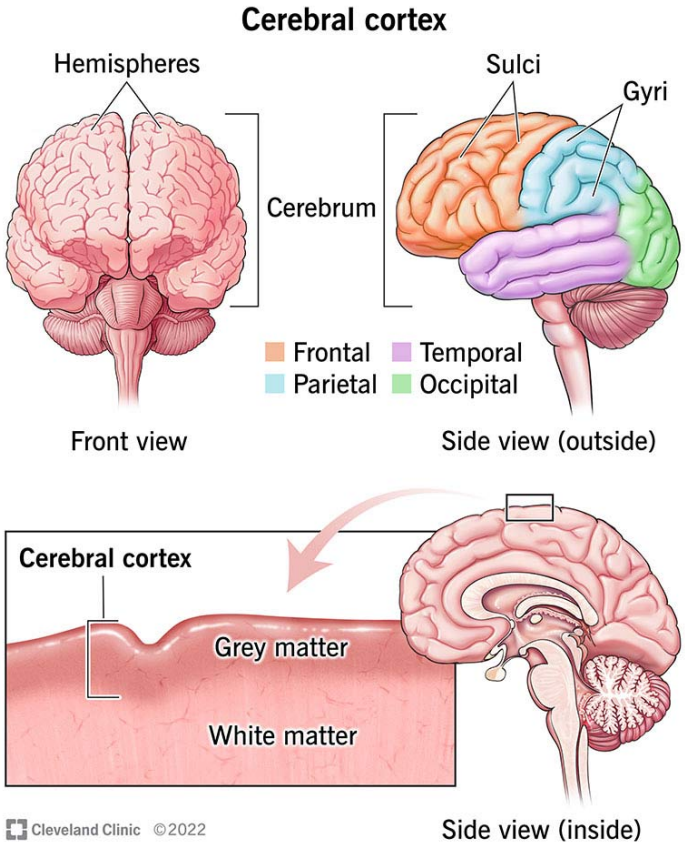

Figure 1.2: Cortical layers organization

Layer	Name (Common)	Main Cell Types / Features	Input Roles	Output / Projection Roles
I	Molecular (Plexiform) layer	Few neurons; apical dendritic tufts of pyramidal neurons; horizontal fibers; Cajal-Retzius cells; glial cells	Receives feedback / modulatory inputs; integrates signal from other cortical areas; contributes to modulation of how input is processed	Minimal direct long-range output; mainly modulatory interactions
II	External Granular layer	Stellate (granular) cells + small pyramidal cells	Inputs from other cortical areas; association fibers	Outputs to adjacent cortical columns; corticocortical communication
III	External Pyramidal layer	Pyramidal cells of small-to-medium size; many horizontal/corticocortical fibers	Receives input from other cortex; association inputs	Sends outputs to other parts of cortex; communicates between cortical areas
IV	Internal Granular (Lamina IV)	Stellate cells mainly; some pyramidal cells; receiving thalamic inputs	Major input hub: thalamic sensory afferents predominantly arrive here	Sends processed signals upwards to layers II & III and across column
V	Internal Pyramidal layer	Large pyramidal cells; size/shape suited to long projection	Receives processed cortical input	Major output layer to subcortical targets (e.g., brainstem, spinal cord), other cortical areas
VI	Multiform / Fusiform layer	Mixed cell types: fusiform, pyramidal, interneurons; connects deeper to thalamus	Receives inputs from other cortical layers & some feedback loops	Sends feedback to thalamus; helps modulate incoming signals; also outputs to other cortical/subcortical regions

Table 1.4: Cortical Layers and Their Functions

- Large pyramidal neurons in Layer V can project long distances to motor targets

Association Areas:

- Rely heavily on Layers II & III for inter-cortical communication and integration
- Process and integrate information from multiple sensory and cognitive sources

Thalamic Feedback:

- Layer VI helps regulate what inputs are emphasized (gain control, feedback)
- Provides top-down modulation of sensory processing

Example Information Flow in Cortical Processing

1. **Sensory Input**: Sensory afferents from thalamus → hit Layer IV in primary sensory cortex
2. **Vertical Processing**: Signals go up → processed and passed through Layers II & III to adjacent/association cortical areas
3. **Integration & Decision**: Higher-level processing / decision made → Layers V / VI send back outputs (motor commands, feedback to thalamus)
4. **Modulation**: Modulatory feedback (Layer I and VI) helps adjust sensitivity, attention, integration

This layered organization demonstrates how biological neural networks implement hierarchical processing, with clear separation of input processing, integration, output generation, and modulatory control - principles that have influenced the design of artificial neural network architectures.

1.2.6 Other Biological Computation Models

DNA Computing

- Uses DNA strands and biochemical reactions to encode and solve problems
- Enables **massive parallelism** (billions of molecules interacting at once)
- Example: Adleman (1994) solved a small Hamiltonian Path problem with DNA

Membrane Computing (P Systems)

- Inspired by biological cells with membranes
- Computation modeled as molecules passing between compartments with rules

Immune System Computation

- Inspired by adaptive immune systems recognizing pathogens
- Used in anomaly detection, cybersecurity, adaptive algorithms

Swarm Intelligence

- Inspired by ants, bees, and bird flocks
- Simple agents interacting lead to complex global solutions
- Example: Ant Colony Optimization for shortest path problems

1.2.7 Comparative Analysis of Computational Models

Key Differences Between Models

Feature	Lambda Calculus	Cellular Automata	Turing Machine	Biological Computation
Paradigm	Functional	Distributed / Parallel	Imperative / Sequential	Adaptive / Parallel / Learning
Representation	Functions + substitution	Grid of cells + local rules	Tape + head + state machine	Neurons, DNA, agents, molecules
Best for	Reasoning about functions & programs	Modeling physical systems	Defining algorithms & computability	Learning, adaptation, biological processes
Computational Power	Universal (Turing-complete)	Universal (e.g., Game of Life)	Universal (baseline model)	Universal (many models are Turing-complete)
Origin	Church (1930s)	Ulam, von Neumann, Conway (1940s–70s)	Turing (1936)	1990s onward (Adleman, Hopfield, etc.)

Table 1.5: Key Differences Between Computational Models

Mapping to Modern Computing Technologies

Lambda Calculus → Functional Programming Languages

- **Influence:** Functional languages like Haskell, Lisp, OCaml, Scala
- **Practice:** Used in compilers, parallelism, AI/ML frameworks (TensorFlow graphs)
- **Example:** Python's `lambda` functions (`map(lambda x: x*2, list)`)

Cellular Automata → Parallel / Distributed Computing

- **Influence:** Inspiration for GPU architectures, parallel algorithms, simulations
- **Practice:** Physics simulations, cryptography, GPU computations
- **Example:** Conway's Game of Life on GPUs

Turing Machine → CPUs & Algorithms

- **Influence:** CPUs ≈ optimized Turing machines (memory = tape, registers = head)

- **Practice:** Von Neumann architecture, algorithm design, compilers
- **Example:** Any code on a CPU is compiled down to Turing-like steps

Biological Computation → AI and Unconventional Computing

- **Influence:** Inspired deep learning, DNA-based computation, swarm optimization
- **Practice:** Neural networks in AI, bioinformatics, optimization, cybersecurity
- **Example:** Deep learning models (vision, NLP), DNA algorithms, ant colony optimization

Modern Technology Mapping Overview

Model	Modern Counterpart	Where It Shows Up
Lambda Calculus	Functional languages, ML frameworks	Haskell, Lisp, TensorFlow, PyTorch
Cellular Automata	GPUs, Parallel computing, Simulation	Physics engines, cryptography
Turing Machine	CPUs, von Neumann architecture, Algorithms	Everyday programming, OS, compilers
Biological Computation	Neural networks, DNA computing, swarm intelligence	AI, optimization, bioinformatics

Table 1.6: Modern Technology Mapping

1.2.8 Key Insights and Takeaways

From Biological Neural Networks

- Neurons are simple units whose structure (dendrite/axon/synapse) enables complex computation
- Signals are electrical within neurons and chemical at synapses; timing and rate carry information
- Synaptic plasticity provides a biological basis for learning (Hebbian adaptation)
- Massive connectivity (many synapses per neuron, $\sim 10^{15}$ total) creates powerful, distributed processing

From Computational Models

- **Lambda Calculus:** Computation = function evaluation
- **Turing Machine:** Computation = manipulating symbols step by step

- **Cellular Automata:** Computation = local rule interactions on grids
- **Biological Computation:** Computation = adaptive processes inspired by nature

Universal Principles

All four computational models are **Turing-complete**, meaning they can solve the same class of problems, but they represent different metaphors for thinking about computation:

1. **Sequential vs. Parallel:** Some models naturally express parallel computation (cellular automata, biological systems) while others are inherently sequential (Turing machines)
2. **Static vs. Adaptive:** Traditional models use fixed rules, while biological models can learn and adapt their behavior
3. **Abstract vs. Physical:** Some models are mathematical abstractions (lambda calculus) while others are inspired by physical or biological processes
4. **Efficiency vs. Universality:** All models can theoretically solve the same problems, but they differ greatly in practical efficiency for different types of tasks

1.2.9 The Journey from Computation to Intelligence

This document traces the path from fundamental computational concepts to sophisticated biological neural networks:

1. **Computation** provides the basic framework for information processing
2. **Computational Models** offer different paradigms for organizing computation
3. **Modern Computing Systems** implement these models in physical hardware
4. **Biological Systems** demonstrate how computation can emerge naturally and adapt
5. **Artificial Neural Networks** attempt to capture the power of biological computation in engineered systems

Understanding this progression helps us appreciate both the theoretical foundations of computation and the remarkable achievements of biological evolution in creating intelligent systems. It also provides insight into why artificial neural networks, inspired by biology but implemented on digital computers, have become such powerful tools for machine learning and artificial intelligence.

1.2.10 Artificial Neural Networks

Introduction: From Biology to Computation

Artificial Neural Networks (ANNs) represent one of the most successful attempts to harness the computational principles observed in biological neural systems for solving complex problems. Unlike traditional algorithmic approaches that follow explicit step-by-step instructions, neural networks learn to solve problems through experience and pattern recognition, much like biological brains.

Fundamental Architecture: Primitive Functions and Composition Rules

To understand artificial neural networks, we must first examine their core computational elements. Every computational model requires:

1. **Primitive Functions**: Basic operations that cannot be decomposed further
2. **Composition Rules**: Ways to combine primitive functions to create complex behaviors

Primitive Functions in Neural Networks In artificial neural networks, **primitive functions are located in the nodes (neurons) of the network**. Each node implements a specific mathematical transformation that processes incoming information and produces an output.

Composition Rules in Neural Networks The **composition rules are contained implicitly in**:

- **Interconnection pattern of the nodes**: How neurons are connected determines information flow
- **Synchrony or asynchrony of information transmission**: Whether neurons update simultaneously or in sequence
- **Presence or absence of cycles**: Whether information can flow in loops (recurrent networks) or only forward (feedforward networks)

This differs fundamentally from traditional computing models:

Computing Model	Primitive Functions	Composition Rules
von Neumann Processor	Machine instructions (ADD, MOVE, JUMP)	Program sequence + control flow
Artificial Neural Networks	Neuron activation functions	Network topology + connection weights + timing

Table 1.7: Computing Models Comparison

The Abstract Neuron: Building Block of Intelligence

Structure of an Abstract Neuron An abstract neuron with n inputs processes information through the following components:

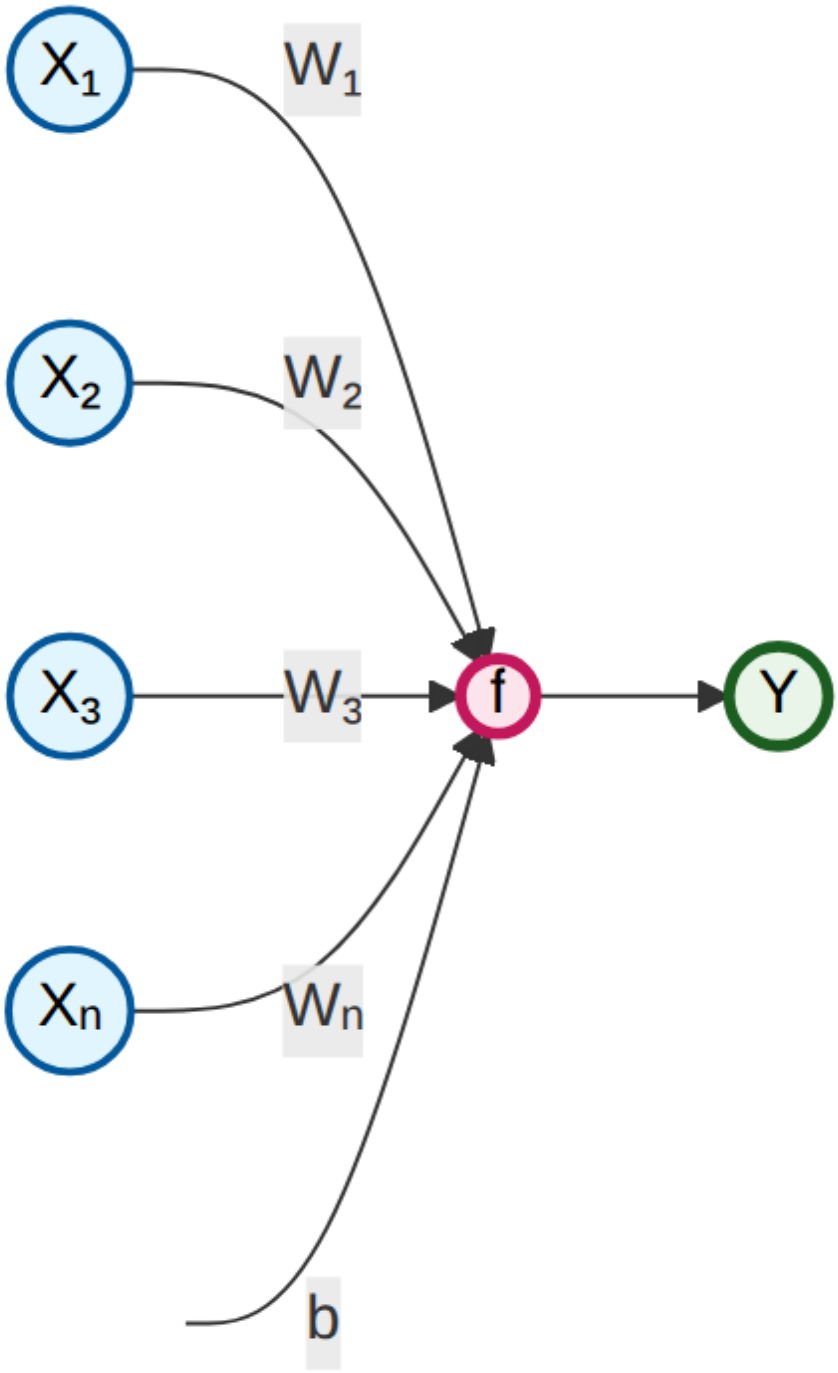

Figure 1.3: Abstract Neuron Architecture

Abstract Neuron Architecture

$$Y = f\left(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n\right) \tag{1.2}$$

Key Components:

1. **Input Channels**: Each input channel i can transmit a real value X_i
2. **Weights**: Each input has an associated weight W_i that multiplies the incoming information
3. **Integration**: The weighted signals are combined (usually by summation)
4. **Primitive Function**: An activation function f transforms the integrated signal
5. **Output**: The result is transmitted to connected neurons

Mathematical Representation The output of a neuron can be expressed as:

$$Y = f\left(\sum_{i=1}^{n} W_i X_i + b\right) \tag{1.3}$$

Where:

- Y: Output of the neuron
- f: Activation function (primitive function)
- W_i: Weight associated with input i
- X_i: Value of input i
- b: Bias term (optional constant)

Neural Networks as Function Approximators

Networks of Primitive Functions Artificial neural networks are fundamentally *networks of primitive functions*. Each node in the network transforms its input into a precisely defined output according to a specific mathematical function. The power of neural networks emerges from the combination of these simple transformations, which collectively create complex computational behaviors.

The key insight is that while individual neurons perform simple operations, their interconnected arrangement allows the network to approximate arbitrarily complex functions through the composition of these elementary building blocks.

The Network Function Consider a neural network that takes inputs (x, y, z) and produces an output through nodes implementing primitive functions f_1, f_2, f_3, f_4. The entire network can be conceptualized as implementing a single *network function* ϕ:

$$\phi(x, y, z) = f_4\left(a_4 \cdot f_3\left(a_3 \cdot f_2\left(a_2 \cdot f_1(a_1 \cdot x)\right)\right) + \ldots\right) \tag{1.4}$$

where $a_1, a_2, \ldots, a_5$ are the weights of the network. Different selections of weights produce different network functions, enabling the network to adapt its behavior to solve various problems.

Three Critical Elements Different models of artificial neural networks differ mainly in three fundamental aspects:

1. **Structure of the Nodes**
   - Choice of activation function (sigmoid, ReLU, tanh, etc.)
   - Input integration method (weighted sum, product, etc.)
   - Presence of bias terms
2. **Topology of the Network**
   - Feedforward vs. recurrent connections
   - Number of layers and neurons per layer
   - Connection patterns (fully connected, sparse, convolutional)
3. **Learning Algorithm**
   - Method for finding optimal weights
   - Supervised vs. unsupervised vs. reinforcement learning
   - Optimization techniques (gradient descent, evolutionary algorithms)

Function Approximation: The Classical Problem

Historical Context Function approximation is a classical problem in mathematics: **How can we reproduce a given function $F : \mathbb{R} \to \mathbb{R}$ either exactly or approximately using a given set of primitive functions?**

Traditional approaches include:

- **Polynomial approximation**: Using powers of x (Taylor series)

- **Fourier approximation**: Using trigonometric functions (sine and cosine)
- **Spline approximation**: Using piecewise polynomials

Neural Networks as Universal Approximators Neural networks provide a revolutionary approach to function approximation:

Key Insight: With sufficient neurons and appropriate activation functions, neural networks can approximate any continuous function to arbitrary precision (Universal Approximation Theorem).

Advantages of Neural Network Approximation

1. **Adaptive**: Networks learn the approximation from data rather than requiring explicit mathematical formulation
2. **Flexible**: Can handle high-dimensional inputs and complex, non-linear relationships
3. **Robust**: Can generalize to unseen data and handle noise
4. **Parallel**: Multiple neurons can process different aspects of the input simultaneously

Comparison: Classical vs. Neural Approximation

Taylor Series Neural Networks As demonstrated in our earlier diagrams, neural networks can explicitly implement classical approximation methods:

Taylor Series: $F(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \ldots + a_n(x - x_0)^n$

- **Basis Functions**: Powers of $(x - x_0)$
- **Coefficients**: Learned as weights in the network
- **Best For**: Local approximation around expansion point x_0

Fourier Series Neural Networks **Fourier Series**: $F(x) = \sum_{i=0}^{\infty} [a_i \cos(ix) + b_i \sin(ix)] = \sum_{i=0}^{\infty} w_i \sin(k_i x + d_i)$

- **Basis Functions**: Trigonometric functions (sine/cosine)
- **Coefficients**: Learned as weights and phase factors
- **Best For**: Periodic functions and signal processing

General Neural Networks **Modern ANNs**: $F(x) = f_n\left(W_n \cdot f_{n-1}\left(W_{n-1} \cdot \ldots f_1(W_1 \cdot x + b_1) + b_{n-1}\right.\right.$

- **Basis Functions**: Learned automatically through hidden layers

- **Coefficients**: All weights and biases learned from data
- **Best For**: Complex, high-dimensional problems without known mathematical structure

Learning in Neural Networks

The Learning Problem Given:

- A set of input-output pairs: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$
- A network architecture

Goal: Find weights W and biases b such that the network function $\phi(x; W, b)$ approximates the target function as closely as possible.

Learning as Optimization **Objective**: Minimize the error between network outputs and desired outputs:

$$E(W, b) = \sum_{i=1}^{m} \|\phi(x_i; W, b) - y_i\|^2 \tag{1.5}$$

Method: Use gradient-based optimization (backpropagation) to iteratively adjust weights:

$$W \leftarrow W - \eta \nabla_W E(W, b) \tag{1.6}$$

$$b \leftarrow b - \eta \nabla_b E(W, b) \tag{1.7}$$

Where η is the learning rate.

Types of Neural Network Architectures

Feedforward Networks

- Information flows in one direction from input to output
- No cycles or loops
- Examples: Perceptrons, Multi-layer Perceptrons (MLPs)

Recurrent Networks

- Contain cycles that allow information to persist

- Can process sequences and exhibit temporal dynamics
- Examples: RNNs, LSTMs, GRUs

Convolutional Networks

- Specialized for processing grid-like data (images)
- Use shared weights and local connectivity
- Examples: CNNs, ResNets, Vision Transformers

Specialized Architectures

- **Autoencoders**: Learn compressed representations
- **GANs**: Generate new data similar to training examples
- **Transformers**: Process sequences with attention mechanisms

Modern Applications and Impact

Computer Vision

- **Image Classification**: Recognizing objects in photographs
- **Object Detection**: Finding and localizing objects
- **Image Generation**: Creating realistic synthetic images

Natural Language Processing

- **Machine Translation**: Converting text between languages
- **Language Models**: Understanding and generating human language
- **Sentiment Analysis**: Determining emotional tone of text

Scientific Computing

- **Physics Simulations**: Solving partial differential equations
- **Drug Discovery**: Predicting molecular properties
- **Climate Modeling**: Understanding complex environmental systems

Key Insights and Future Directions

Fundamental Principles

1. **Distributed Representation**: Information is stored across many neurons rather than in specific locations
2. **Emergent Computation**: Complex behaviors arise from simple neuron interactions
3. **Adaptive Learning**: Networks modify themselves based on experience
4. **Fault Tolerance**: Performance degrades gracefully with neuron damage

Current Challenges

1. **Interpretability**: Understanding how networks make decisions
2. **Data Efficiency**: Learning from fewer examples
3. **Robustness**: Handling adversarial inputs and distribution shifts
4. **Energy Efficiency**: Reducing computational requirements

Future Prospects

- **Neuromorphic Computing**: Hardware that mimics brain architecture
- **Quantum Neural Networks**: Leveraging quantum mechanics for computation
- **Biological-Artificial Hybrids**: Integrating living neurons with artificial systems
- **Continual Learning**: Networks that learn continuously without forgetting

Conclusion: The Power of Artificial Neural Networks

Artificial neural networks represent a paradigm shift in computation - from explicit algorithmic programming to learning from data. By abstracting the key principles of biological neural computation (weighted connections, nonlinear transformations, and adaptive learning), ANNs have become one of the most powerful tools for solving complex problems in science, engineering, and everyday life.

The journey from understanding individual neurons to building networks capable of human-level performance in specific domains illustrates the power of **composition** - how simple primitive functions, when properly connected and trained, can give rise to remarkable computational abilities.

As we continue to develop more sophisticated architectures and learning algorithms, artificial neural networks promise to play an increasingly central role in our technological future, potentially leading us toward artificial general intelligence and beyond.

Network Architecture:

- **Input Layer**: Two inputs - x (variable) and x_0 (expansion point/bias term)
- **Polynomial Computation Layer**: Calculate basis functions $z^0, z^1, z^2, z^3, \ldots, z^n$
  - $z^0 = (x - x_0)^0 = 1$ (constant term)
  - $z^1 = (x - x_0)^1$ (linear term)
  - $z^2 = (x - x_0)^2$ (quadratic term)
  - $z^3 = (x - x_0)^3$ (cubic term)
  - $z^n = (x - x_0)^n$ (nth-order term)
- **Coefficient Neurons**: Learn the Taylor series coefficients $a_0, a_1, a_2, \ldots, a_n$
- **Output**: Sum all weighted terms to approximate $F(x)$

Mathematical Mapping:

- a_0 **neuron**: Learns the constant coefficient $\times$ z^0 (bias term)
- a_1 **neuron**: Learns the linear coefficient $\times$ z^1
- a_2 **neuron**: Learns the quadratic coefficient $\times$ z^2
- a_3 **neuron**: Learns the cubic coefficient $\times$ z^3
- a_n **neuron**: Learns the nth-order coefficient $\times$ z^n

Key Features:

- **Dual Input Design**: x and x_0 as separate inputs allows the network to learn both the function value and the optimal expansion point
- **Explicit Basis Functions**: $z^0, z^1, z^2, \ldots, z^n$ clearly show the polynomial basis being computed
- **Learnable Expansion Point**: x_0 can be treated as a learnable parameter (bias) or fixed reference point
- **Modular Structure**: Each polynomial degree has its own computation and coefficient pathway

This demonstrates how neural networks can implement **universal function approximation** by learning to combine polynomial basis functions, directly representing the mathematical foundation of Taylor series expansion through a structured network architecture.

Fourier Series Neural Network

$$F(x) = \sum_{i=0}^{\infty} a_i \cos(ix) + b_i \sin(ix) = \sum_{i=0}^{\infty} w_i \sin(k_i x + d_i) \tag{1.8}$$

Fourier Series Neural Network: $F(x) = \sum_{i=0}^{\infty} [a_i \cos(ix) + b_i \sin(ix)]$

Network Architecture:

- **Input Layer**: Single input x (the variable)
- **Trigonometric Basis Layer**: Calculate basis functions
  - **Cosine terms**: $\cos^0(x) = 1, \cos^1(x) = \cos(x), \cos^2(x) = \cos(2x), \ldots, \cos^n(x) = \cos(nx)$
  - **Sine terms**: $\sin^1(x) = \sin(x), \sin^2(x) = \sin(2x), \sin^3(x) = \sin(3x), \ldots, \sin^n(x) = \sin(nx)$
- **Coefficient Neurons**: Learn the Fourier series coefficients
  - **a-coefficients**: $a_0, a_1, a_2, a_3, \ldots, a_n$ for cosine terms
  - **b-coefficients**: $b_1, b_2, b_3, \ldots, b_n$ for sine terms
- **Output**: Sum all weighted trigonometric terms to approximate $F(x)$

Mathematical Mapping:

- a_0 **neuron**: Learns the DC component (constant term) $\times \cos^0(x) = 1$
- a_i **neurons**: Learn cosine coefficients $\times \cos(ix)$ for harmonic frequencies
- b_i **neurons**: Learn sine coefficients $\times \sin(ix)$ for harmonic frequencies

Key Features:

- **Frequency Domain Representation**: Each neuron pair represents a specific harmonic frequency
- **Orthogonal Basis Functions**: Sine and cosine functions form an orthogonal basis for periodic signals
- **Harmonic Analysis**: Network learns to decompose signals into fundamental and harmonic components
- **Periodic Function Approximation**: Particularly effective for modeling periodic and oscillatory phenomena

Comparison with Taylor Series:

- **Taylor**: Local polynomial approximation using powers of $(x - x_0)$
- **Fourier**: Global trigonometric approximation using sine/cosine harmonics
- **Taylor**: Best for smooth functions around expansion point
- **Fourier**: Best for periodic functions and signal processing applications

Both architectures demonstrate how neural networks can implement classical mathematical series expansions, providing structured approaches to **universal function approximation** through learned basis function combinations.

Learning from Data: The Key Difference

The main difference between Taylor or Fourier series and artificial neural networks is, however, that **the function F to be approximated is given not explicitly but implicitly through a set of input-output examples.** We know F only at some points but we want to generalize as well as possible. This means that we try to adjust the parameters of the network in an optimal manner to reflect the information known and to extrapolate to new input patterns which will be shown to the network afterwards. This is the task of the learning algorithm used to adjust the network's parameters.

Classical Series vs. Neural Networks: A Fundamental Distinction Classical Mathematical Series (Taylor/Fourier):

- **Explicit Function Definition**: The function $F(x)$ is mathematically defined and known
- **Analytical Coefficients**: Series coefficients can be computed directly using calculus
  - Taylor: $a_n = \frac{F^{(n)}(x_0)}{n!}$ (nth derivative at expansion point)
  - Fourier: a_n, b_n computed via integration over the function's period
- **Perfect Representation**: Given enough terms, the series can represent the function exactly
- **No Learning Required**: Coefficients are determined mathematically, not learned

Artificial Neural Networks:

- **Implicit Function Definition**: The function F is unknown but represented by data points
- **Learned Parameters**: Network weights and biases are learned from examples

- **Approximation from Samples**: Must generalize from finite training data to unknown inputs
- **Adaptive Learning**: Parameters adjust through iterative optimization algorithms

The Learning Process: From Examples to Generalization Given:

- Training dataset: $\{(x_1, F(x_1)), (x_2, F(x_2)), \ldots, (x_m, F(x_m))\}$
- Neural network architecture with parameters θ (weights and biases)

Goal: Find optimal parameters θ^* such that the network can:

1. **Fit the training data**: Network output $\approx F(x_i)$ for known examples
2. **Generalize to new inputs**: Network output $\approx F(x)$ for unseen x values

Learning Algorithm Steps:

1. **Initialize**: Set random initial values for all network parameters
2. **Forward Pass**: Compute network output for training examples
3. **Error Calculation**: Measure difference between network output and target values
4. **Backward Pass**: Compute gradients of error with respect to parameters
5. **Parameter Update**: Adjust parameters to reduce error
6. **Iteration**: Repeat until convergence or satisfactory performance

Generalization: The Ultimate Test The true power of neural networks lies in their ability to **extrapolate to new input patterns**:

Training Phase:

- Network learns from limited examples
- Parameters adjusted to minimize training error
- Network discovers underlying patterns in the data

Testing Phase:

- Network encounters completely new inputs
- Must produce reasonable outputs based on learned patterns
- Success measured by generalization performance

Key Challenges:

- **Overfitting**: Learning training data too specifically, poor generalization
- **Underfitting**: Insufficient learning capacity, poor performance on both training and test data
- **Bias-Variance Tradeoff**: Balancing model complexity with generalization ability

Caveat: Biological Realism vs. Mathematical Abstraction

In the theory of artificial neural networks we do not consider the whole complexity of real biological neurons. We only abstract some general principles and content ourselves with different levels of detail when simulating neural ensembles. The general approach is to conceive each neuron as a primitive function producing numerical results at some points in time. However, we can also think of artificial neurons as computing units which produce pulse trains in the way that biological neurons do. We can then simulate this behavior and look at the output of simple networks. This kind of approach, although more closely related to the biological paradigm, is still a very rough approximation of the biological processes.

Levels of Biological Abstraction Level 1: Mathematical Abstraction (Most Common)

- **Neuron Model**: Simple mathematical function $f\left(\sum w_i x_i + b\right)$
- **Activation**: Continuous values (real numbers) representing firing rates
- **Computation**: Matrix operations and gradient-based learning
- **Advantages**: Computationally efficient, mathematically tractable
- **Disadvantages**: Far removed from biological reality

Level 2: Rate-Based Models

- **Neuron Model**: Firing rate as a function of input
- **Activation**: Continuous values representing average firing frequency
- **Computation**: Differential equations modeling neural dynamics
- **Advantages**: Captures some temporal dynamics
- **Disadvantages**: Ignores individual spike timing

Level 3: Spiking Neural Networks

- **Neuron Model**: Explicit spike generation and propagation
- **Activation**: Discrete spikes (pulse trains) with precise timing
- **Computation**: Event-driven simulation of individual spikes
- **Advantages**: More biologically realistic, captures temporal coding
- **Disadvantages**: Computationally expensive, harder to train

Level 4: Detailed Biophysical Models

- **Neuron Model**: Multi-compartment models with ion channels
- **Activation**: Detailed membrane potential dynamics
- **Computation**: Partial differential equations for each neuron compartment
- **Advantages**: High biological fidelity
- **Disadvantages**: Extremely computationally intensive, limited scalability

The Abstraction Trade-off Why We Abstract:

1. **Computational Feasibility**: Detailed biological models are too complex for large networks
2. **Mathematical Tractability**: Simple models allow theoretical analysis and efficient learning algorithms
3. **Practical Performance**: Highly abstracted models often perform better on artificial tasks
4. **Understanding**: Simple models help us understand fundamental principles

What We Lose:

1. **Temporal Dynamics**: Real neurons have complex temporal behavior and memory
2. **Spike Timing**: Information may be encoded in precise timing of spikes
3. **Metabolic Constraints**: Real neurons have energy limitations affecting computation
4. **Plasticity Mechanisms**: Biological learning is far more complex than gradient descent
5. **Network Topology**: Real brain networks have complex, evolved architectures

Current Research Directions Bridging the Gap:

- **Neuromorphic Computing**: Hardware that mimics brain architecture and dynamics
- **Spiking Neural Networks**: Incorporating temporal dynamics while maintaining trainability
- **Biologically Plausible Learning**: Developing learning rules that could work in real brains
- **Brain-Inspired Architectures**: Using insights from neuroscience to design better artificial networks

The Ongoing Challenge: Finding the right level of biological realism that maintains:

- **Computational efficiency** for practical applications
- **Theoretical understanding** of fundamental principles
- **Biological inspiration** for novel architectures and algorithms
- **Performance advantages** over purely mathematical approaches

Implications for Understanding This abstraction means that while artificial neural networks are **inspired by** biological systems, they should not be considered **models of** biological systems. They represent a successful engineering approach that captures some essential computational principles while sacrificing biological realism for practical performance.

Key Takeaway: Artificial neural networks are best understood as powerful mathematical tools for pattern recognition and function approximation, rather than as accurate simulations of brain function. Their success demonstrates that some principles of neural computation can be abstracted and applied effectively, even when divorced from their biological context.

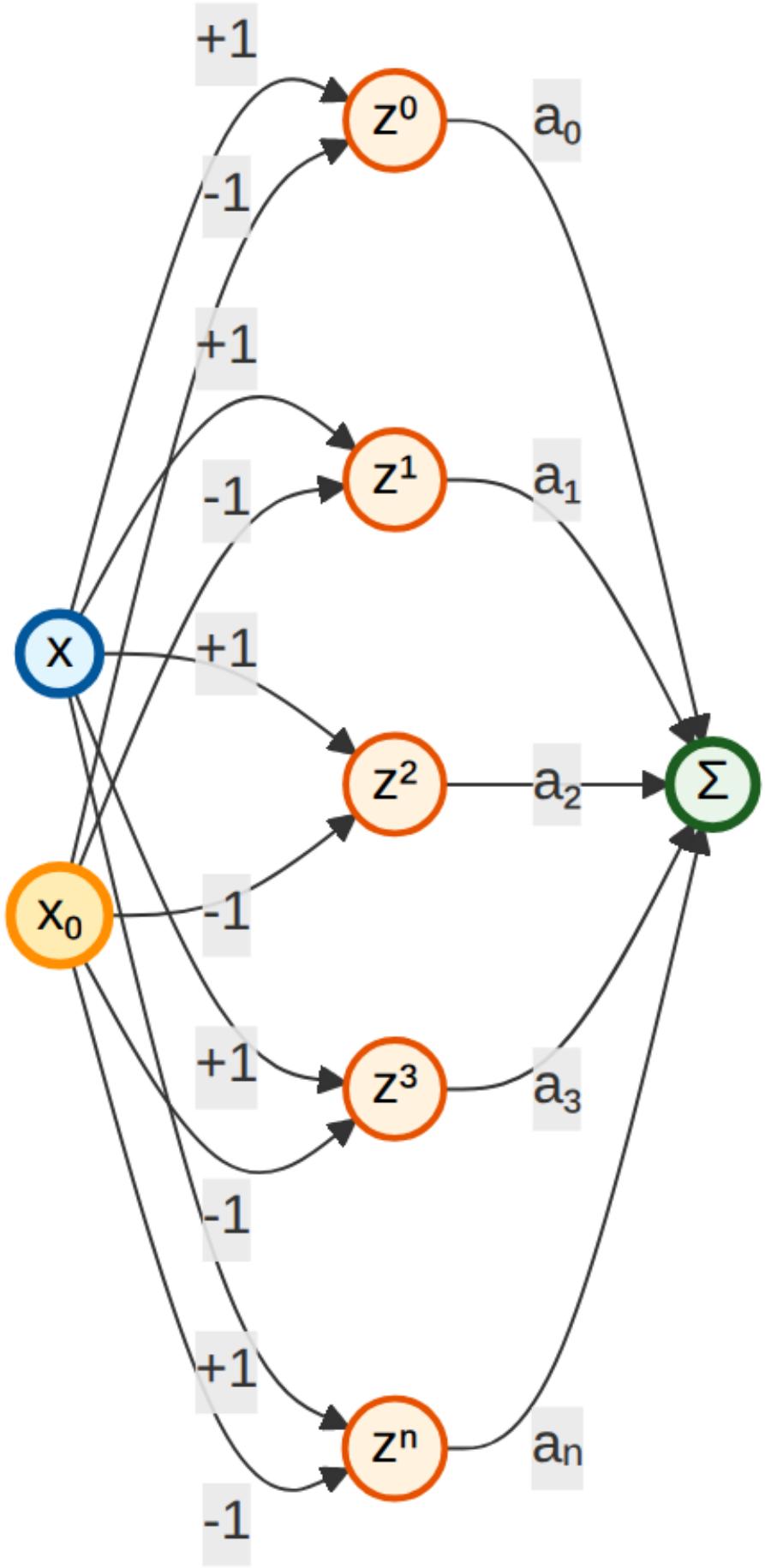

Figure 1.4: Taylor Series Neural Network: $F(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \ldots + a_n(x - x_0)^n$

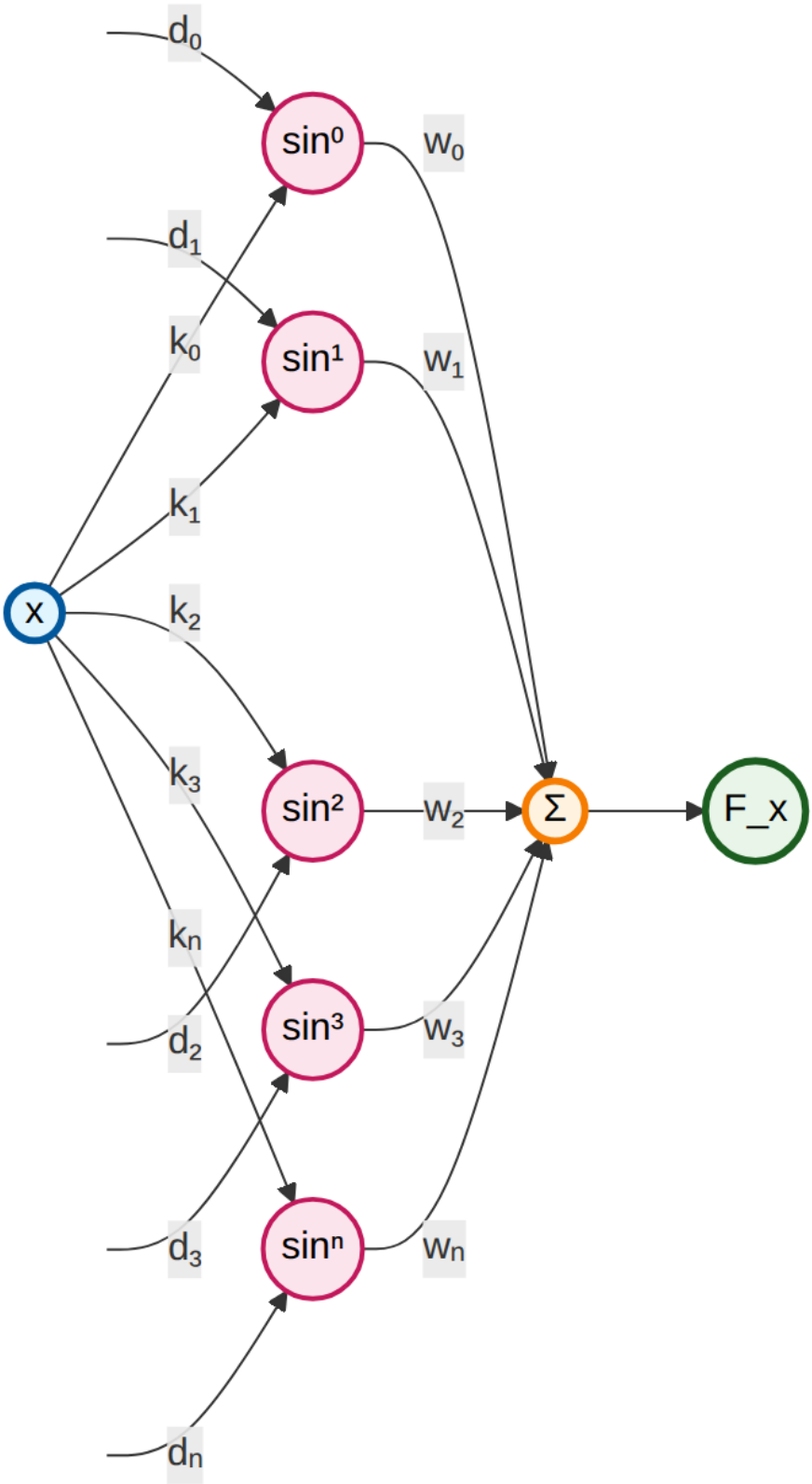

Figure 1.5: Fourier Series Neural Network