DA6401:Introduction to Deep Learning
Module 1A-Artificial Neuron Models

Ganapathy Krishnamurthi

Department of Data Science and Artificial Intelligence , IIT Madras

Logistics & Schedule

Course Details

- **Instructor:** Prof. Ganapathy Krishnamurthi
- **Email:** gankrish@dsai.iitm.ac.in
- **Venue:** CRC302 (In-person)
- **Slot C:**
  - Mon (10:00), Tue (09:00)
  - Wed (08:00), Fri (12:00)

Communication

- **Channels:** Moodle
- **Dates:** Jan 20 – May 5, 2026

Note on Lectures

Classes are primarily in-person. Online links will be provided if physical attendance is not feasible.

Getting Ready for the Course

Course Prerequisites

- **Foundational Courses:**
  - DA5000: **Mathematical Foundations of Data Science**
  - DA5400: **Foundations of Machine Learning**

Getting Ready for the Course

Course Prerequisites

- **Foundational Courses:**
  - DA5000: **Mathematical Foundations of Data Science**
  - DA5400: **Foundations of Machine Learning**
- **Technical Proficiency:**
  - Strong command of **Python**.
  - Assignments will rely heavily on **NumPy** and **PyTorch**.

Grading Scheme & Key Dates

Grading Scheme & Key Dates

Component	Weight	Date
Quiz 1 (MCQ)	20%	**Feb 18, 2026**
Quiz 2 (MCQ)	20%	**Mar 25, 2026**

Grading Scheme & Key Dates

Component	Weight	Date
Quiz 1 (MCQ)	20%	**Feb 18, 2026**
Quiz 2 (MCQ)	20%	**Mar 25, 2026**
Assignments ($4 \times 5\%$)	20%	*Throughout Sem*

Grading Scheme & Key Dates

Component	Weight	Date
Quiz 1 (MCQ)	20%	**Feb 18, 2026**
Quiz 2 (MCQ)	20%	**Mar 25, 2026**
Assignments ($4 \times 5\%$)	20%	*Throughout Sem*
Final Exam*	**40%**	**May 8, 2026**

**The final exam will be handwritten.*

Course Roadmap: Fundamentals (Modules 1-3)

- **Module 1: Introduction**
  - History, Perceptrons, and the XOR problem.
  - **Multilayer Perceptrons (MLPs)** and representation power.

Course Roadmap: Fundamentals (Modules 1-3)

- **Module 1: Introduction**
  - History, Perceptrons, and the XOR problem.
  - **Multilayer Perceptrons (MLPs)** and representation power.
- **Module 2: Optimization**
  - Backpropagation derivation.
  - Optimizers: **SGD, Adam, RMSProp**.
  - Initialization techniques (Xavier/He).

Course Roadmap: Fundamentals (Modules 1-3)

- **Module 1: Introduction**
  - History, Perceptrons, and the XOR problem.
  - **Multilayer Perceptrons (MLPs)** and representation power.
- **Module 2: Optimization**
  - Backpropagation derivation.
  - Optimizers: **SGD, Adam, RMSProp**.
  - Initialization techniques (Xavier/He).
- **Module 3: Training Methodology**
  - Regularization: L2, Dropout, Early Stopping.
  - Handling Class Imbalance.

Course Roadmap: Specialized Architectures (Modules 4-6)

- **Module 4: Computer Vision (CNNs)**
  - Convolutions, Pooling, LeNet to ResNet.
  - Object Detection (YOLO) & Segmentation (U-Net).

Course Roadmap: Specialized Architectures (Modules 4-6)

- **Module 4: Computer Vision (CNNs)**
  - Convolutions, Pooling, LeNet to ResNet.
  - Object Detection (YOLO) & Segmentation (U-Net).
- **Module 5: Sequence Models (NLP)**
  - Word Embeddings (Word2Vec).
  - RNNs, LSTMs, and GRUs.
  - Introduction to Attention.

Course Roadmap: Specialized Architectures (Modules 4-6)

- **Module 4: Computer Vision (CNNs)**
  - Convolutions, Pooling, LeNet to ResNet.
  - Object Detection (YOLO) & Segmentation (U-Net).
- **Module 5: Sequence Models (NLP)**
  - Word Embeddings (Word2Vec).
  - RNNs, LSTMs, and GRUs.
  - Introduction to Attention.
- **Module 6: Generative AI**
  - **Transformers** & Self-Attention.
  - Generative Models: VAEs, GANs, and **Diffusion Models**.

References & Resources

Textbooks

- **Deep Learning** – Ian Goodfellow, Yoshua Bengio, Aaron Courville (MIT Press).
- **Dive into Deep Learning** – Zhang et al. (d2l.ai).

Questions?

Machine Learning Landscape

From Traditional Machine Learning to Deep Learning

Machine Learning

Arthur Samuel (1959)

Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.

Machine Learning

Arthur Samuel (1959)

Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.

Learn directly from data - No explicitly programmed rules

Machine Learning

Arthur Samuel (1959)

Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.

Learn directly from data - No explicitly programmed rules

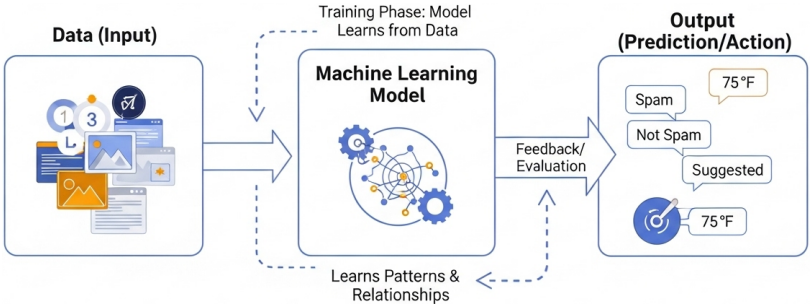


Figure: ML: Learning from Data (Source: ImageGen)

Common Machine Learning Applications

Figure: Housing Price Prediction

Common Machine Learning Applications

Figure: Housing Price Prediction

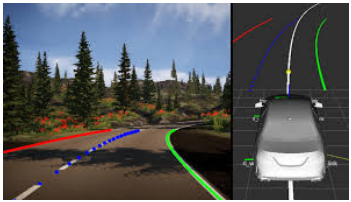


Figure: Autonomous vehicles(Lane following)

Common Machine Learning Applications

Figure: Housing Price Prediction

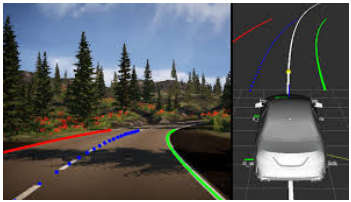


Figure: Autonomous vehicles(Lane following)

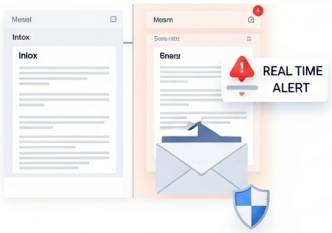


Figure: Message Sentiment Analysis

Common Machine Learning Applications

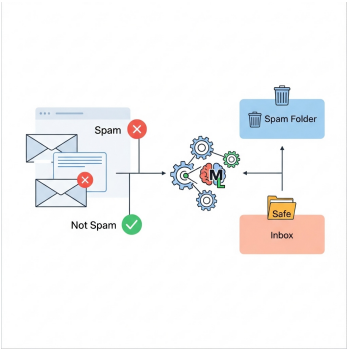


Figure: Spam Email Filtering

Common Machine Learning Applications

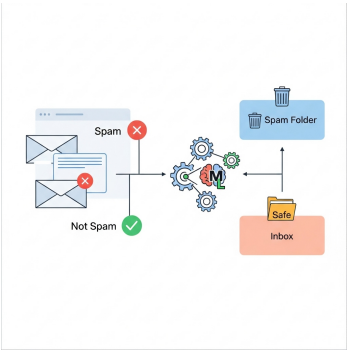


Figure: Spam Email Filtering

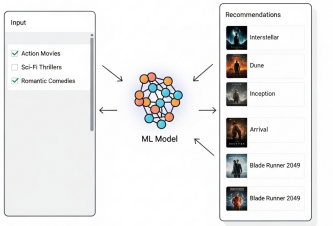


Figure: Recommendation engines

Common Machine Learning Applications

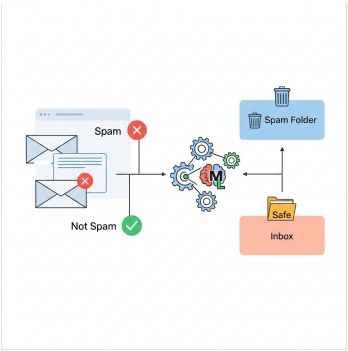


Figure: Spam Email Filtering

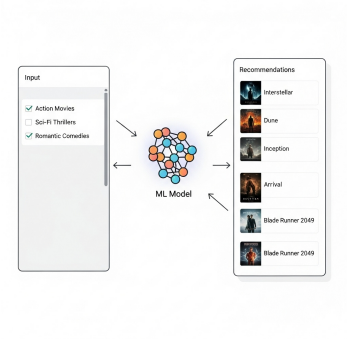


Figure: Recommendation engines

Figure: Image Recognition

Machine Learning: Key Concepts

The Data & Process

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.
- Model learns from these engineered features.

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.
- Model learns from these engineered features.
- Success depends entirely on feature quality.

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.
- Model learns from these engineered features.
- Success depends entirely on feature quality.

The Models

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.
- Model learns from these engineered features.
- Success depends entirely on feature quality.

The Models

Linear Models

- *Core Idea:* Assumes linear relationships.

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.
- Model learns from these engineered features.
- Success depends entirely on feature quality.

The Models

Linear Models

- *Core Idea:* Assumes linear relationships.
- *Examples:* Linear/Logistic Regression.

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.
- Model learns from these engineered features.
- Success depends entirely on feature quality.

The Models

Linear Models

- *Core Idea:* Assumes linear relationships.
- *Examples:* Linear/Logistic Regression.
- *Traits:* Fast, interpretable, but simple.

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.
- Model learns from these engineered features.
- Success depends entirely on feature quality.

The Models

Linear Models

- *Core Idea:* Assumes linear relationships.
- *Examples:* Linear/Logistic Regression.
- *Traits:* Fast, interpretable, but simple.

Tree-Based Models

- *Core Idea:* Learns ‘if-then-else‘ rules.

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.
- Model learns from these engineered features.
- Success depends entirely on feature quality.

The Models

Linear Models

- *Core Idea:* Assumes linear relationships.
- *Examples:* Linear/Logistic Regression.
- *Traits:* Fast, interpretable, but simple.

Tree-Based Models

- *Core Idea:* Learns ‘if-then-else‘ rules.
- *Examples:* Random Forest, Gradient Boosting.

Machine Learning: Key Concepts

The Data & Process

Focus: Structured Data

- Data in tables (rows & columns).
- Predict a target from features.

Method: Manual Feature Engineering

- Human experts design features.
- Model learns from these engineered features.
- Success depends entirely on feature quality.

The Models

Linear Models

- *Core Idea:* Assumes linear relationships.
- *Examples:* Linear/Logistic Regression.
- *Traits:* Fast, interpretable, but simple.

Tree-Based Models

- *Core Idea:* Learns ‘if-then-else‘ rules.
- *Examples:* Random Forest, Gradient Boosting.
- *Traits:* Captures non-linearity, powerful.

Limitations of Traditional Machine Learning

Limitations of Traditional Machine Learning

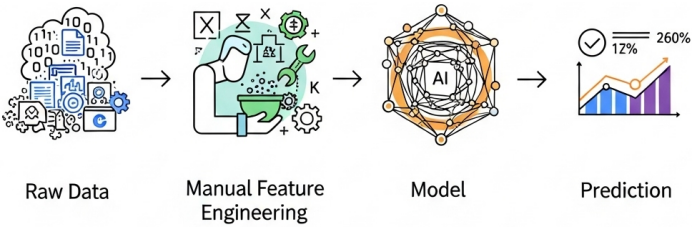


Figure: Feature Engineering Bottleneck (Source: ImageGen)

- **The Feature Engineering Bottleneck**: Time-consuming, requires domain expertise, and is often suboptimal.

Limitations of Traditional Machine Learning

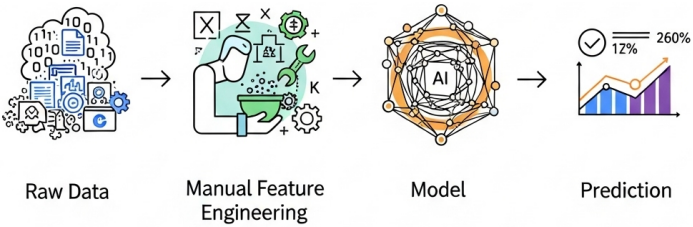


Figure: Feature Engineering Bottleneck (Source: ImageGen)

- **The Feature Engineering Bottleneck**: Time-consuming, requires domain expertise, and is often suboptimal.
- **Inability to handle high-dimensional data**: Struggles with images, audio, or raw text.

Limitations of Traditional Machine Learning

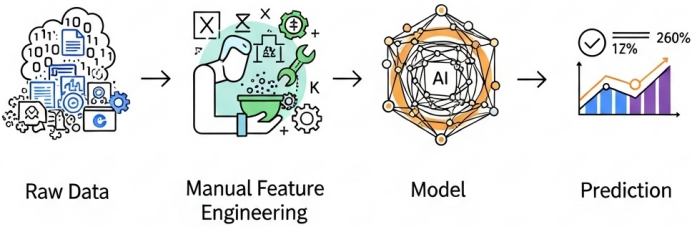


Figure: Feature Engineering Bottleneck (Source: ImageGen)

- **The Feature Engineering Bottleneck**: Time-consuming, requires domain expertise, and is often suboptimal.
- **Inability to handle high-dimensional data**: Struggles with images, audio, or raw text.
- **Limited representation learning**: Learns shallow patterns, not deep, hierarchical features.

Limitations of Traditional Machine Learning

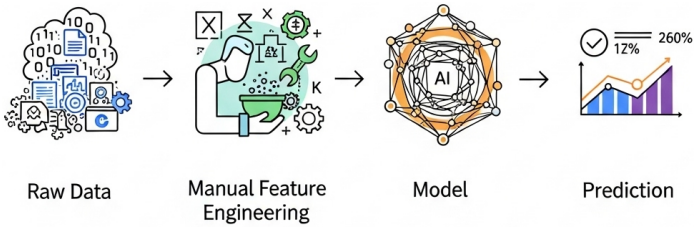


Figure: Feature Engineering Bottleneck (Source: ImageGen)

- **The Feature Engineering Bottleneck**: Time-consuming, requires domain expertise, and is often suboptimal.
- **Inability to handle high-dimensional data**: Struggles with images, audio, or raw text.
- **Limited representation learning**: Learns shallow patterns, not deep, hierarchical features.
- **Performance plateaus**: Adding more data yields diminishing returns after a certain point.

Deep Learning - A New Paradigm

From Traditional Machine Learning to Deep Learning

The Shift to Deep Learning: A New Paradigm

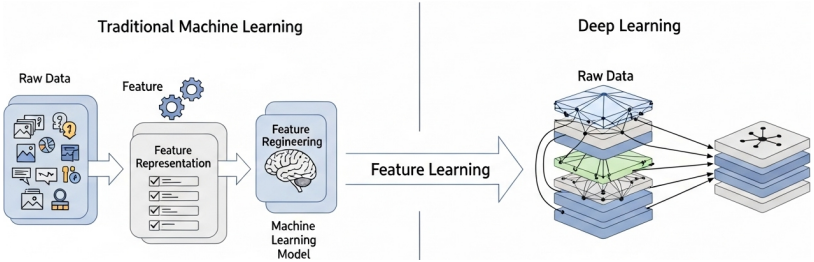


Figure: ML needs manual feature extraction; DL learns them automatically.

The Shift to Deep Learning: A New Paradigm

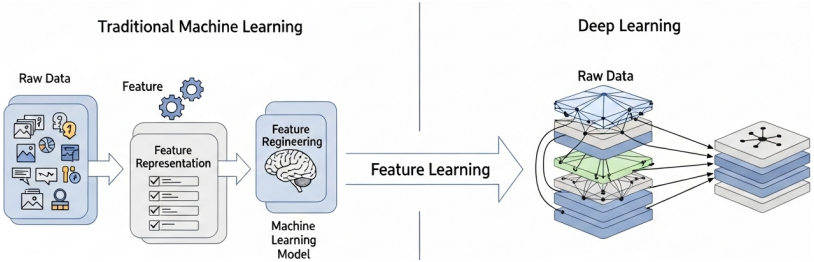


Figure: ML needs manual feature extraction; DL learns them automatically.

Deep Learning: The Core Idea

- Models learn directly from raw data.

The Shift to Deep Learning: A New Paradigm

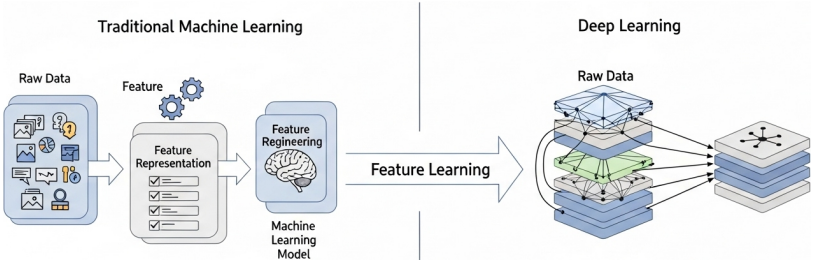


Figure: ML needs manual feature extraction; DL learns them automatically.

Deep Learning: The Core Idea

- Models learn directly from raw data.
- Handles images, text, and audio.

The Shift to Deep Learning: A New Paradigm

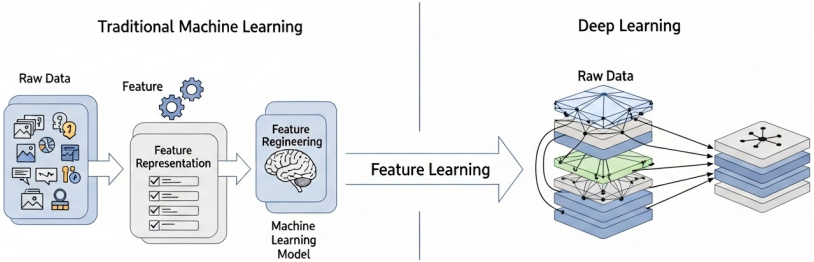


Figure: ML needs manual feature extraction; DL learns them automatically.

Deep Learning: The Core Idea

- Models learn directly from raw data.
- Handles images, text, and audio.
- **No manual feature engineering.**

The Shift to Deep Learning: A New Paradigm

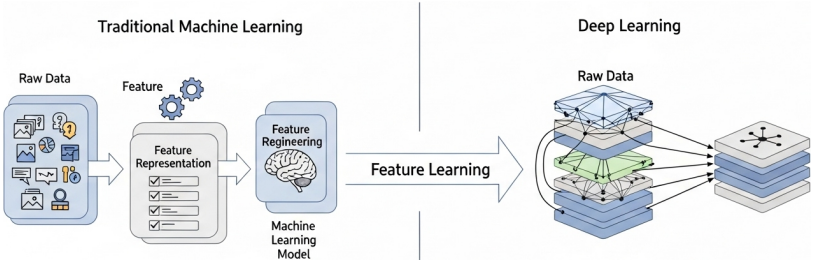


Figure: ML needs manual feature extraction; DL learns them automatically.

Deep Learning: The Core Idea

- Models learn directly from raw data.
- Handles images, text, and audio.
- **No manual feature engineering.**

Why this Paradigm Shift?

The Shift to Deep Learning: A New Paradigm

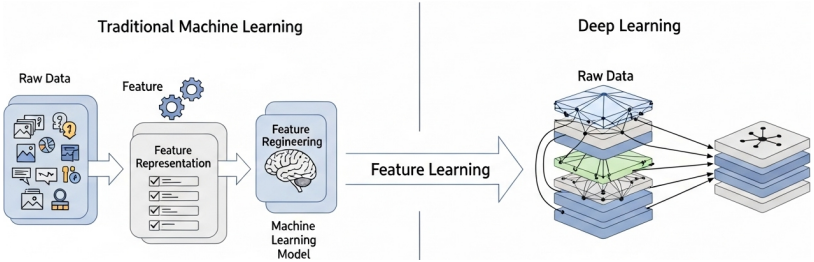


Figure: ML needs manual feature extraction; DL learns them automatically.

Deep Learning: The Core Idea

- Models learn directly from raw data.
- Handles images, text, and audio.
- **No manual feature engineering.**

Why this Paradigm Shift?

- Solves the feature engineering bottleneck.

The Shift to Deep Learning: A New Paradigm

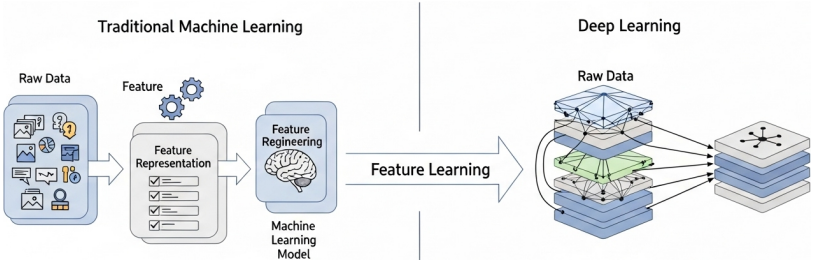


Figure: ML needs manual feature extraction; DL learns them automatically.

Deep Learning: The Core Idea

- Models learn directly from raw data.
- Handles images, text, and audio.
- **No manual feature engineering.**

Why this Paradigm Shift?

- Solves the feature engineering bottleneck.
- Automates representation learning.

The Shift to Deep Learning: A New Paradigm

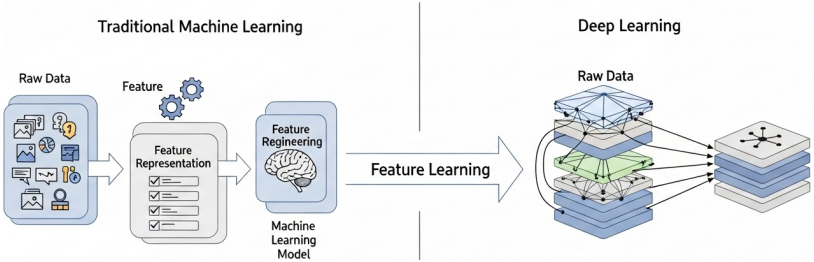


Figure: ML needs manual feature extraction; DL learns them automatically.

Deep Learning: The Core Idea

- Models learn directly from raw data.
- Handles images, text, and audio.
- **No manual feature engineering.**

Why this Paradigm Shift?

- Solves the feature engineering bottleneck.
- Automates representation learning.
- Unlocks new, complex capabilities.

Popular Deep Learning Applications

Popular Deep Learning Applications

Figure: Computer vision

Popular Deep Learning Applications

Figure: Computer vision

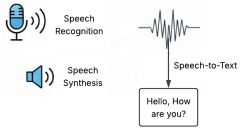


Figure: Speech recognition and Audio synthesis

Popular Deep Learning Applications

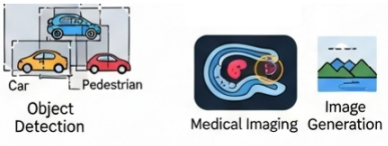


Figure: Computer vision

Figure: Natural language processing

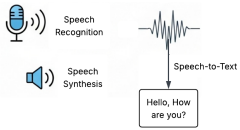


Figure: Speech recognition and Audio synthesis

Popular Deep Learning Applications

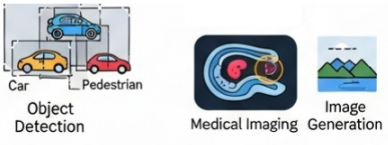


Figure: Computer vision

Figure: Natural language processing

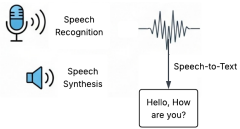


Figure: Speech recognition and Audio synthesis

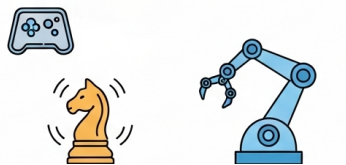


Figure: Game playing and robotics

Key Differences: ML vs. DL

Machine Learning

Deep Learning

Key Differences: ML vs. DL

Machine Learning

- **Feature Extraction:**
  Requires manual engineering.

Deep Learning

- **Feature Extraction:**
  Learns features automatically.

Key Differences: ML vs. DL

Machine Learning

- **Feature Extraction:**
  Requires manual engineering.
- **Representation:**
  Learns shallow patterns.

Deep Learning

- **Feature Extraction:**
  Learns features automatically.
- **Representation:**
  Builds hierarchical features.

Key Differences: ML vs. DL

Machine Learning

- **Feature Extraction:**
  Requires manual engineering.
- **Representation:**
  Learns shallow patterns.
- **Data Scalability:**
  Works well on small to medium data.

Deep Learning

- **Feature Extraction:**
  Learns features automatically.
- **Representation:**
  Builds hierarchical features.
- **Data Scalability:**
  Excels with very large datasets.

Key Differences: ML vs. DL

Machine Learning

- **Feature Extraction:**
  Requires manual engineering.
- **Representation:**
  Learns shallow patterns.
- **Data Scalability:**
  Works well on small to medium data.

Deep Learning

- **Feature Extraction:**
  Learns features automatically.
- **Representation:**
  Builds hierarchical features.
- **Data Scalability:**
  Excels with very large datasets.
- **Training Paradigm:**
  Enables end-to-end learning.

What Enabled the Deep Learning?

A convergence of three key factors created the "perfect environment"

What Enabled the Deep Learning?

A convergence of three key factors created the "perfect environment"

What Enabled the Deep Learning?

A convergence of three key factors created the "perfect environment"

Big Data

- Massive labeled datasets.
- The "fuel" for hungry models.
- e.g., ImageNet, Wikipedia.

What Enabled the Deep Learning?

A convergence of three key factors created the "perfect environment"

Big Data

- Massive labeled datasets.
- The "fuel" for hungry models.
- e.g., ImageNet, Wikipedia.

Computational Power

What Enabled the Deep Learning?

A convergence of three key factors created the "perfect environment"

Big Data

- Massive labeled datasets.
- The "fuel" for hungry models.
- e.g., ImageNet, Wikipedia.

Computational Power

- Rise of GPUs (parallel power).
- Made deep training feasible.
- Drastically cut training time.

What Enabled the Deep Learning?

A convergence of three key factors created the "perfect environment"

Big Data

- Massive labeled datasets.
- The "fuel" for hungry models.
- e.g., ImageNet, Wikipedia.

Computational Power

- Rise of GPUs (parallel power).
- Made deep training feasible.
- Drastically cut training time.

Better Algorithms

What Enabled the Deep Learning?

A convergence of three key factors created the "perfect environment"

Big Data

- Massive labeled datasets.
- The "fuel" for hungry models.
- e.g., ImageNet, Wikipedia.

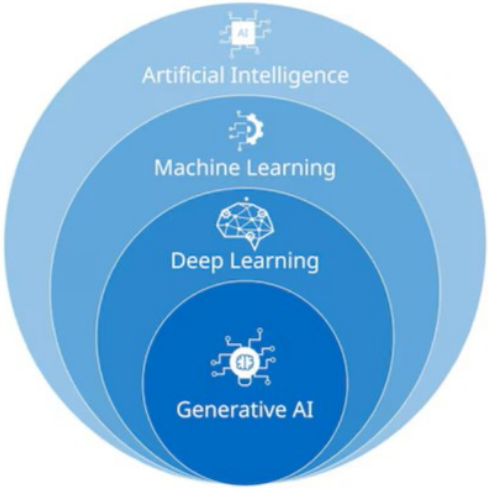
Computational Power

- Rise of GPUs (parallel power).
- Made deep training feasible.
- Drastically cut training time.

Better Algorithms

- Architectural innovations.
- e.g., CNNs, Transformers.
- Smarter training methods.

The AI Venn Digram



From Biology to Mathematics

Building Blocks of a Neural Network

Inspiration from the Brain

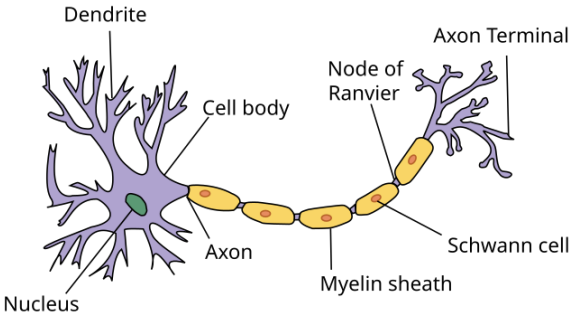


Figure: A biological neuron (Source: Wikipedia)

Inspiration from the Brain

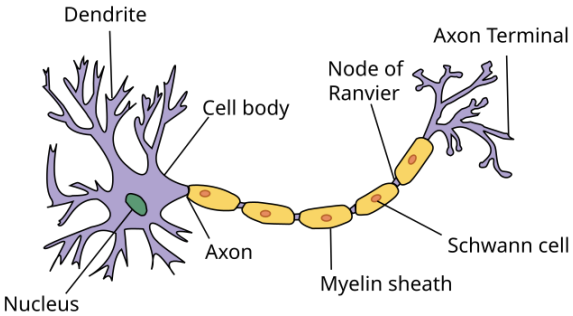


Figure: A biological neuron (Source: Wikipedia)

A Simple Model of a Neuron:

- Receives signals via Dendrites.

Inspiration from the Brain

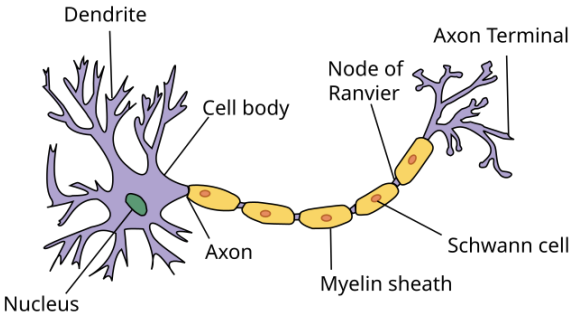


Figure: A biological neuron (Source: Wikipedia)

A Simple Model of a Neuron:

- Receives signals via Dendrites.
- Integrates signals in the Soma/Cell body.

Inspiration from the Brain

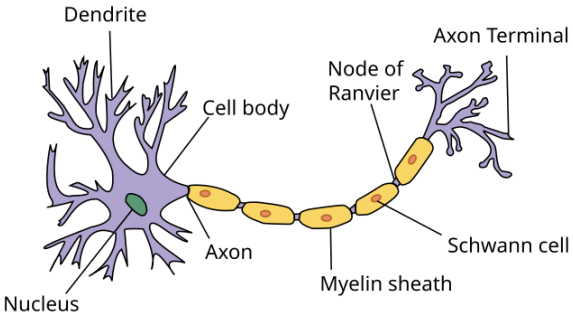


Figure: A biological neuron (Source: Wikipedia)

A Simple Model of a Neuron:

- Receives signals via Dendrites.
- Integrates signals in the Soma/Cell body.
- If threshold is met, it "Fires".

Inspiration from the Brain

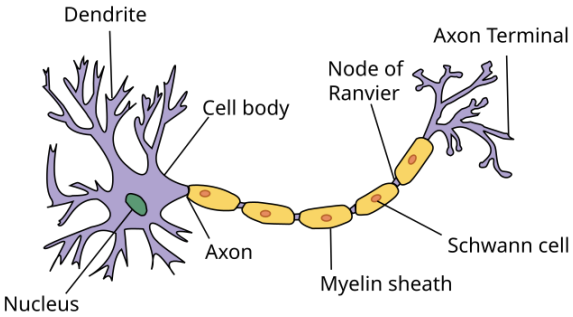


Figure: A biological neuron (Source: Wikipedia)

A Simple Model of a Neuron:

- Receives signals via Dendrites.
- Integrates signals in the Soma/Cell body.
- If threshold is met, it "Fires".
- Sends output signal via Axon.

Inspiration from the Brain

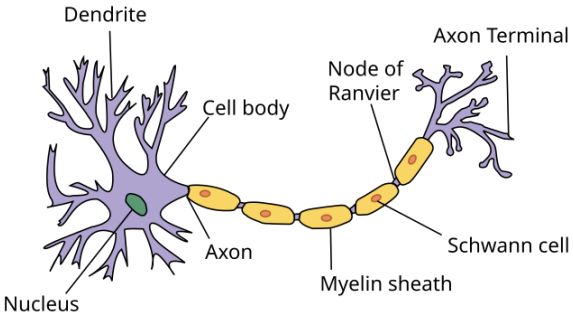


Figure: A biological neuron (Source: Wikipedia)

A Simple Model of a Neuron:

- Receives signals via Dendrites.
- Integrates signals in the Soma/Cell body.
- If threshold is met, it "Fires".
- Sends output signal via Axon.

Important: Inspiration, Not a Replica

- Brain is vastly more complex.

Inspiration from the Brain

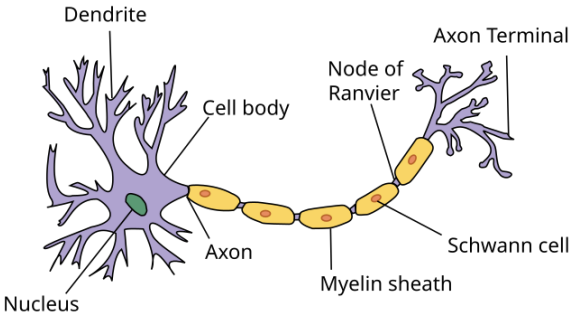


Figure: A biological neuron (Source: Wikipedia)

A Simple Model of a Neuron:

- Receives signals via Dendrites.
- Integrates signals in the Soma/Cell body.
- If threshold is met, it "Fires".
- Sends output signal via Axon.

Important: Inspiration, Not a Replica

- Brain is vastly more complex.
- Biological learning is not backpropagation.

Inspiration from the Brain

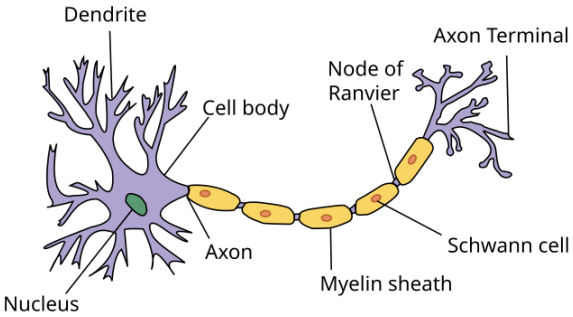


Figure: A biological neuron (Source: Wikipedia)

A Simple Model of a Neuron:

- Receives signals via Dendrites.
- Integrates signals in the Soma/Cell body.
- If threshold is met, it "Fires".
- Sends output signal via Axon.

Important: Inspiration, Not a Replica

- Brain is vastly more complex.
- Biological learning is not backpropagation.
- This is a **useful abstraction**, not a perfect model.

The McCulloch-Pitts Neuron (1943)

The McCulloch-Pitts Neuron (1943)

The Core Idea: A Simple Biological Logic Gate

The first mathematical model of a neuron. It operates on a simple principle: if enough inputs are active, the neuron "fires."

The McCulloch-Pitts Neuron (1943)

The Core Idea: A Simple Biological Logic Gate

The first mathematical model of a neuron. It operates on a simple principle: if enough inputs are active, the neuron "fires."

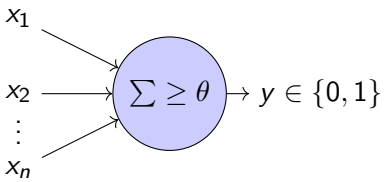


Figure: The M-P Neuron Model: Sum inputs and compare to a threshold θ.

The McCulloch-Pitts Neuron (1943)

The Core Idea: A Simple Biological Logic Gate

The first mathematical model of a neuron. It operates on a simple principle: if enough inputs are active, the neuron "fires."

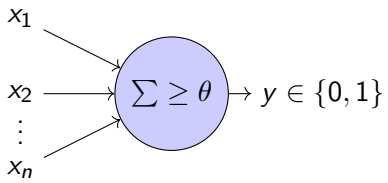


Figure: The M-P Neuron Model: Sum inputs and compare to a threshold θ.

How it Works:

- Takes multiple binary inputs ($x_i \in \{0, 1\}$).

The McCulloch-Pitts Neuron (1943)

The Core Idea: A Simple Biological Logic Gate

The first mathematical model of a neuron. It operates on a simple principle: if enough inputs are active, the neuron "fires."

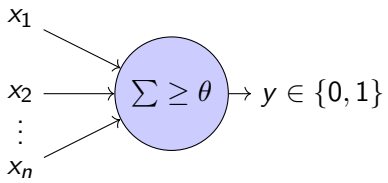


Figure: The M-P Neuron Model: Sum inputs and compare to a threshold θ.

How it Works:

- Takes multiple binary inputs ($x_i \in \{0, 1\}$).
- Calculates their sum.

The McCulloch-Pitts Neuron (1943)

The Core Idea: A Simple Biological Logic Gate

The first mathematical model of a neuron. It operates on a simple principle: if enough inputs are active, the neuron "fires."

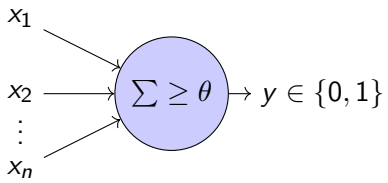


Figure: The M-P Neuron Model: Sum inputs and compare to a threshold θ.

How it Works:

- Takes multiple binary inputs ($x_i \in \{0, 1\}$).
- Calculates their sum.
- The output is 1 if the sum is greater than or equal to a fixed threshold (θ).

The McCulloch-Pitts Neuron (1943)

The Core Idea: A Simple Biological Logic Gate

The first mathematical model of a neuron. It operates on a simple principle: if enough inputs are active, the neuron "fires."

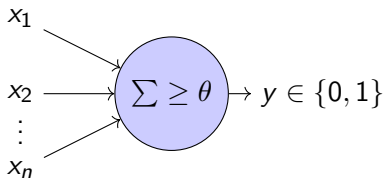


Figure: The M-P Neuron Model: Sum inputs and compare to a threshold θ.

How it Works:

- Takes multiple binary inputs ($x_i \in \{0, 1\}$).
- Calculates their sum.
- The output is 1 if the sum is greater than or equal to a fixed threshold (θ).
- The output is 0 otherwise.

The McCulloch-Pitts Neuron (1943)

The Core Idea: A Simple Biological Logic Gate

The first mathematical model of a neuron. It operates on a simple principle: if enough inputs are active, the neuron "fires."

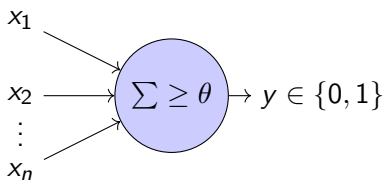


Figure: The M-P Neuron Model: Sum inputs and compare to a threshold θ.

How it Works:

- Takes multiple binary inputs ($x_i \in \{0, 1\}$).
- Calculates their sum.
- The output is 1 if the sum is greater than or equal to a fixed threshold (θ).
- The output is 0 otherwise.

Significance:

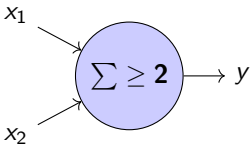
- Proved that networks of these simple units could compute any logical function (e.g., AND, OR).

M-P Neuron Examples: Logical Functions

- Inputs are binary: $x \in \{0, 1\}$.

1. The AND Function

(Fires only if all inputs are 1)

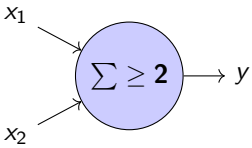


M-P Neuron Examples: Logical Functions

- Inputs are binary: $x \in \{0, 1\}$.

1. The AND Function

(Fires only if all inputs are 1)



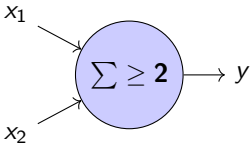
x_1	x_2	Sum	y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	**2**	**1**

M-P Neuron Examples: Logical Functions

- Inputs are binary: $x \in \{0, 1\}$.

1. The AND Function

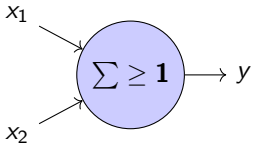
(Fires only if all inputs are 1)



x_1	x_2	Sum	y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	**2**	**1**

2. The OR Function

(Fires if at least one input is 1)

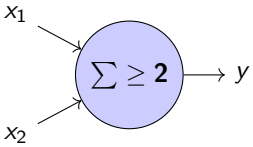


M-P Neuron Examples: Logical Functions

- Inputs are binary: $x \in \{0, 1\}$.

1. The AND Function

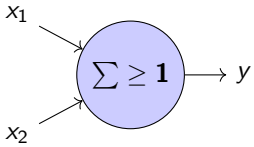
(Fires only if all inputs are 1)



x_1	x_2	Sum	y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	**2**	**1**

2. The OR Function

(Fires if at least one input is 1)



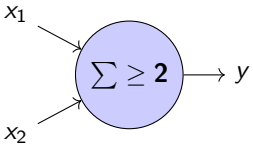
x_1	x_2	Sum	y
0	0	0	0
0	1	1	**1**
1	0	1	**1**
1	1	2	**1**

M-P Neuron Examples: Logical Functions

- Inputs are binary: $x \in \{0, 1\}$.

1. The AND Function

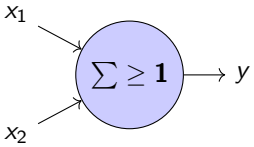
(Fires only if all inputs are 1)



x_1	x_2	Sum	y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	**2**	**1**

2. The OR Function

(Fires if at least one input is 1)



x_1	x_2	Sum	y
0	0	0	0
0	1	1	**1**
1	0	1	**1**
1	1	2	**1**

Observation

By simply changing the **threshold** (θ), the same physical structure performs a completely different logical function.

Real-World Analogy: The "Movie Night" Decision

The Scenario:
You are deciding whether to go to a movie tonight (y).

Real-World Analogy: The "Movie Night" Decision

The Scenario:
You are deciding whether to go to a movie tonight (y).

The Inputs (Factors):

- x_1: Is the movie a comedy? (1=Yes, 0=No)
- x_2: Is a friend coming? (1=Yes, 0=No)
- x_3: Is the ticket cheap? (1=Yes, 0=No)

Real-World Analogy: The "Movie Night" Decision

The Scenario:
You are deciding whether to go to a movie tonight (y).

The Inputs (Factors):

- x_1: Is the movie a comedy? (1=Yes, 0=No)
- x_2: Is a friend coming? (1=Yes, 0=No)
- x_3: Is the ticket cheap? (1=Yes, 0=No)

The Personality (Threshold):
You are **easy-going**. You only need **2 out of 3** things to go right to be convinced.

$$\theta = 2$$

Real-World Analogy: The "Movie Night" Decision

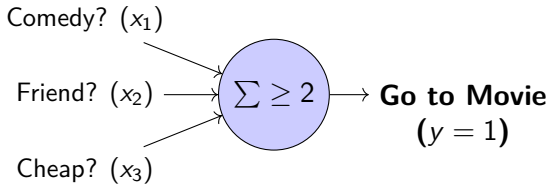
The Scenario:
You are deciding whether to go to a movie tonight (y).

The Inputs (Factors):

- x_1: Is the movie a comedy? (1=Yes, 0=No)
- x_2: Is a friend coming? (1=Yes, 0=No)
- x_3: Is the ticket cheap? (1=Yes, 0=No)

The Personality (Threshold):
You are **easy-going**. You only need **2 out of 3** things to go right to be convinced.

$$\theta = 2$$



Real-World Analogy: The "Movie Night" Decision

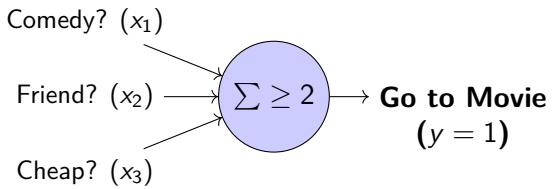
The Scenario:
You are deciding whether to go to a movie tonight (y).

The Inputs (Factors):

- x_1: Is the movie a comedy? (1=Yes, 0=No)
- x_2: Is a friend coming? (1=Yes, 0=No)
- x_3: Is the ticket cheap? (1=Yes, 0=No)

The Personality (Threshold):
You are **easy-going**. You only need **2 out of 3** things to go right to be convinced.

$$\theta = 2$$



Situation

It's a Horror movie ($x_1 = 0$), but your friend is coming ($x_2 = 1$) and it's cheap ($x_3 = 1$).

Real-World Analogy: The "Movie Night" Decision

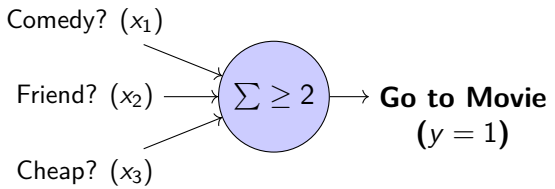
The Scenario:
You are deciding whether to go to a movie tonight (y).

The Inputs (Factors):

- x_1: Is the movie a comedy? (1=Yes, 0=No)
- x_2: Is a friend coming? (1=Yes, 0=No)
- x_3: Is the ticket cheap? (1=Yes, 0=No)

The Personality (Threshold):
You are **easy-going**. You only need **2 out of 3** things to go right to be convinced.

$$\theta = 2$$



Situation

It's a Horror movie ($x_1 = 0$), but your friend is coming ($x_2 = 1$) and it's cheap ($x_3 = 1$).

$$\text{Sum} = 0 + 1 + 1 = \mathbf{2}$$

$$2 \geq 2 \implies \textbf{Fire (Go to Movie)}$$

Geometric Interpretation: Drawing a Line

The Mathematical Boundary

Recall the firing condition for 2 inputs:

$$x_1 + x_2 \geq \theta$$

Geometric Interpretation: Drawing a Line

The Mathematical Boundary

Recall the firing condition for 2 inputs:

$$x_1 + x_2 \geq \theta$$

This inequality defines a **half-plane**. The boundary is the line:

$$x_1 + x_2 = \theta \quad \Rightarrow \quad \mathbf{x_2} = -\mathbf{x_1} + \theta$$

Geometric Interpretation: Drawing a Line

The Mathematical Boundary

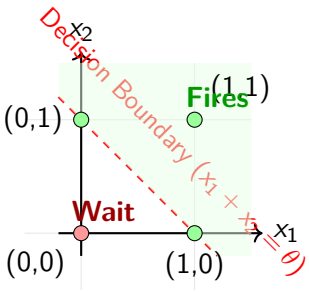
Recall the firing condition for 2 inputs:

$$x_1 + x_2 \geq \theta$$

This inequality defines a **half-plane**. The boundary is the line:

$$x_1 + x_2 = \theta \quad \Rightarrow \quad \mathbf{x_2 = -x_1 + \theta}$$

- Points **above** the line: Neuron Fires (1).
- Points **below** the line: Neuron Silent (0).



Geometric Interpretation: Drawing a Line

The Mathematical Boundary

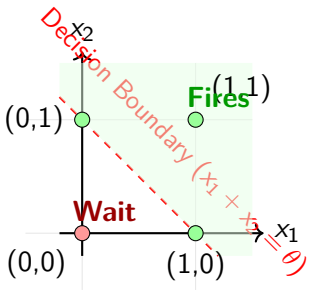
Recall the firing condition for 2 inputs:

$$x_1 + x_2 \geq \theta$$

This inequality defines a **half-plane**. The boundary is the line:

$$x_1 + x_2 = \theta \quad \Rightarrow \quad \mathbf{x_2 = -x_1 + \theta}$$

- Points **above** the line: Neuron Fires (1).
- Points **below** the line: Neuron Silent (0).



Key Takeaway: Linear Separability

The MP Neuron is a **Linear Classifier**. It works by drawing a *single straight line* to separate the "Yes" answers from the "No" answers.

Where the Model Breaks: The "Used Car" Dilemma

The Scenario:

You are buying a used car. You have a threshold of **2** "Yes" factors to say "I'll buy it."

Where the Model Breaks: The "Used Car" Dilemma

The Scenario:
You are buying a used car. You have a threshold of **2** "Yes" factors to say "I'll buy it."

The Features (x):

1. Is the Engine good? (x_1)
2. Are the Brakes working? (x_2)
3. Is there a Cup Holder? (x_3)

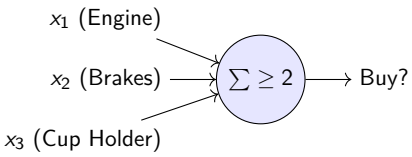
Where the Model Breaks: The "Used Car" Dilemma

The Scenario:
You are buying a used car. You have a threshold of **2** "Yes" factors to say "I'll buy it."

The Features (x):

1. Is the Engine good? (x_1)
2. Are the Brakes working? (x_2)
3. Is there a Cup Holder? (x_3)

MP Neuron Logic ($\theta = 2$)



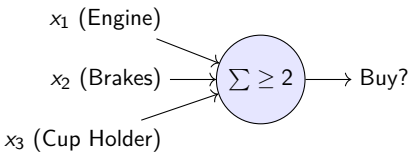
Where the Model Breaks: The "Used Car" Dilemma

The Scenario:
You are buying a used car. You have a threshold of **2** "Yes" factors to say "I'll buy it."

The Features (x):

1. Is the Engine good? (x_1)
2. Are the Brakes working? (x_2)
3. Is there a Cup Holder? (x_3)

MP Neuron Logic ($\theta = 2$)



In the real world, Brakes $\gg$ Cup Holder.

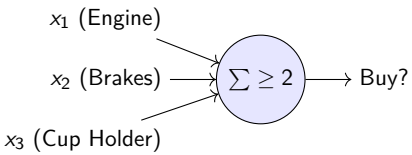
Where the Model Breaks: The "Used Car" Dilemma

The Scenario:
You are buying a used car. You have a threshold of **2** "Yes" factors to say "I'll buy it."

The Features (x):

1. Is the Engine good? (x_1)
2. Are the Brakes working? (x_2)
3. Is there a Cup Holder? (x_3)

MP Neuron Logic ($\theta = 2$)



In the real world, Brakes $\gg$ Cup Holder.

Failure Mode

Consider a car with **No Brakes**, but a Good Engine and a Cup Holder:

- $x_1 = 1, x_2 = 0, x_3 = 1$
- Sum $= 1 + 0 + 1 = \mathbf{2}$

Result: The MP Neuron says **"BUY"** because it treats a *Cup Holder* as equal to *Brakes*.

Critical Limitations of the M-P Neuron

Critical Limitations of the M-P Neuron

A Calculator, Not a Learner

The M-P neuron was a fixed logic gate. It could compute, but it could not **learn**.

Critical Limitations of the M-P Neuron

A Calculator, Not a Learner

The M-P neuron was a fixed logic gate. It could compute, but it could not **learn**.

No Feature Importance

- Equally important features.
- Can't prioritize key features.

Critical Limitations of the M-P Neuron

A Calculator, Not a Learner

The M-P neuron was a fixed logic gate. It could compute, but it could not **learn**.

No Feature Importance

- Equally important features.
- Can't prioritize key features.
- *Missing piece: Weights.*

Critical Limitations of the M-P Neuron

A Calculator, Not a Learner

The M-P neuron was a fixed logic gate. It could compute, but it could not **learn**.

No Feature Importance

- Equally important features.
- Can't prioritize key features.
- *Missing piece: Weights.*

No Automatic Learning

Critical Limitations of the M-P Neuron

A Calculator, Not a Learner

The M-P neuron was a fixed logic gate. It could compute, but it could not **learn**.

No Feature Importance

- Equally important features.
- Can't prioritize key features.
- *Missing piece: Weights.*

No Automatic Learning

- Threshold (θ) is fixed.
- Must be set manually.

Critical Limitations of the M-P Neuron

A Calculator, Not a Learner

The M-P neuron was a fixed logic gate. It could compute, but it could not **learn**.

No Feature Importance

- Equally important features.
- Can't prioritize key features.
- *Missing piece: Weights.*

No Automatic Learning

- Threshold (θ) is fixed.
- Must be set manually.
- *Missing piece: A Learning Rule.*

Critical Limitations of the M-P Neuron

A Calculator, Not a Learner

The M-P neuron was a fixed logic gate. It could compute, but it could not **learn**.

No Feature Importance

- Equally important features.
- Can't prioritize key features.
- *Missing piece: Weights.*

No Automatic Learning

- Threshold (θ) is fixed.
- Must be set manually.
- *Missing piece: A Learning Rule.*

Restricted to Binary I/O

Critical Limitations of the M-P Neuron

A Calculator, Not a Learner

The M-P neuron was a fixed logic gate. It could compute, but it could not **learn**.

No Feature Importance

- Equally important features.
- Can't prioritize key features.
- *Missing piece: Weights.*

No Automatic Learning

- Threshold (θ) is fixed.
- Must be set manually.
- *Missing piece: A Learning Rule.*

Restricted to Binary I/O

- Inputs must be 0 or 1.
- Outputs are only 0 or 1.
- Can't handle real-valued data.

The Leap to Learning: The Perceptron (1958)

The Leap to Learning: The Perceptron (1958)

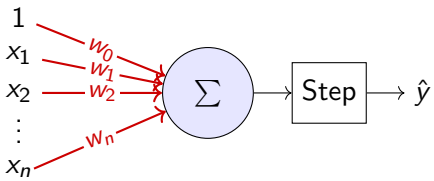


Figure: The Perceptron: Introduces learnable weights (w_i) on each input.

The Leap to Learning: The Perceptron (1958)

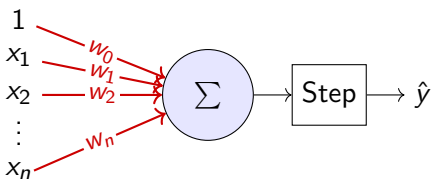


Figure: The Perceptron: Introduces learnable weights (w_i) on each input.

Innovation 1: Weighted Inputs

- Each input is multiplied by a weight.
- Weights represent feature importance.
- The model can now **prioritize** signals.

The Leap to Learning: The Perceptron (1958)

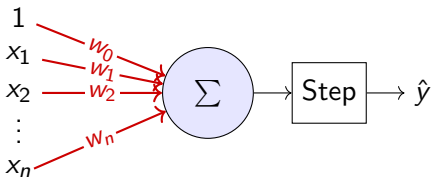


Figure: The Perceptron: Introduces learnable weights (w_i) on each input.

Innovation 1: Weighted Inputs

- Each input is multiplied by a weight.
- Weights represent feature importance.
- The model can now **prioritize** signals.

Innovation 2: The Learning Rule

- Compares prediction ($\hat{y}$) to true label (y).
- If wrong, it adjusts the weights to correct the error.

The Leap to Learning: The Perceptron (1958)

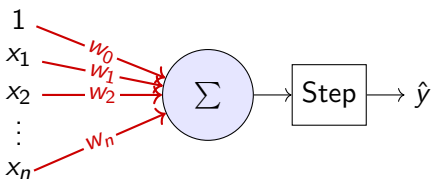


Figure: The Perceptron: Introduces learnable weights (w_i) on each input.

Innovation 1: Weighted Inputs

- Each input is multiplied by a weight.
- Weights represent feature importance.
- The model can now **prioritize** signals.

Innovation 2: The Learning Rule

- Compares prediction ($\hat{y}$) to true label (y).
- If wrong, it adjusts the weights to correct the error.

Significance: The Birth of Learning Machines

For the first time, **a machine could automatically learn** to classify patterns, laying the foundation for all of modern deep learning.

Mathematical Formulation: From Fixed to Flexible

McCulloch-Pitts (The Old Way)

- **Inputs:** Binary $x_i \in \{0, 1\}$
- **Weights:** None (all 1)
- **Threshold:** Fixed θ

$$y = 1 \quad \text{if} \quad \sum_{i=1}^{n} x_i \geq \theta$$

Mathematical Formulation: From Fixed to Flexible

McCulloch-Pitts (The Old Way)

- **Inputs:** Binary $x_i \in \{0, 1\}$
- **Weights:** None (all 1)
- **Threshold:** Fixed θ

$$y = 1 \quad \text{if} \quad \sum_{i=1}^{n} x_i \geq \theta$$

The Perceptron (The New Way)

- **Inputs:** Real numbers $x_i \in \mathbb{R}$
- **Weights:** Learnable $w_i \in \mathbb{R}$
- **Bias:** Learnable b (or w_0)

$$y = 1 \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i + b \geq 0$$

Mathematical Formulation: From Fixed to Flexible

McCulloch-Pitts (The Old Way)

- **Inputs:** Binary $x_i \in \{0, 1\}$
- **Weights:** None (all 1)
- **Threshold:** Fixed θ

$$y = 1 \quad \text{if} \quad \sum_{i=1}^{n} x_i \geq \theta$$

The Perceptron (The New Way)

- **Inputs:** Real numbers $x_i \in \mathbb{R}$
- **Weights:** Learnable $w_i \in \mathbb{R}$
- **Bias:** Learnable b (or w_0)

$$y = 1 \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i + b \geq 0$$

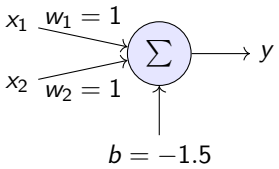
- We often rewrite the threshold θ as a **bias term** $b = -\theta$.
- Moving it to the left side: $\sum w_i x_i - \theta \geq 0 \implies \mathbf{w} \cdot \mathbf{x} + b \geq 0$.
- **Compact Notation:** $z = \mathbf{w}^T \mathbf{x} + b$.

Implementing Logic Gates with Weights

By adjusting weights (w) and bias (b), the same architecture performs different tasks.

1. The AND Gate

$y = 1$ if $x_1 = 1, x_2 = 1$

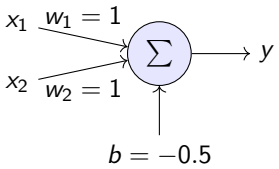


$$1(1) + 1(1) - 1.5 = \mathbf{0.5} \geq 0 \implies 1$$

$$1(1) + 1(0) - 1.5 = -\mathbf{0.5} < 0 \implies 0$$

2. The OR Gate

$y = 1$ if any $x = 1$



$$1(0) + 1(1) - 0.5 = \mathbf{0.5} \geq 0 \implies 1$$

$$1(0) + 1(0) - 0.5 = -\mathbf{0.5} < 0 \implies 0$$

Real World Solution: The "Used Car" Revisited

The Problem: In the MP model, a Cup Holder (x_3) was equal to Brakes (x_2).

The Perceptron Solution: Assign **Weights** based on importance.

- w_1 (Engine) $= 5$
- w_3 (Cup Holder) $= 1$
- w_2 (Brakes) $=$ **10** (Critical!)
- Bias $b = -8$ (Threshold of 8)

Real World Solution: The "Used Car" Revisited

The Problem: In the MP model, a Cup Holder (x_3) was equal to Brakes (x_2).

The Perceptron Solution: Assign **Weights** based on importance.

- w_1 (Engine) $= 5$
- w_3 (Cup Holder) $= 1$
- w_2 (Brakes) $=$ **10** (Critical!)
- Bias $b = -8$ (Threshold of 8)

Scenario 1: No Brakes

Engine Good (1), No Brakes (0), Cup Holder (1).

$$\sum = (5 \times 1) + (10 \times 0) + (1 \times 1) - 8$$

$$\sum = 6 - 8 = -2$$

$$-2 < 0 \implies \textbf{Don't Buy (0)}$$

Real World Solution: The "Used Car" Revisited

The Problem: In the MP model, a Cup Holder (x_3) was equal to Brakes (x_2).

The Perceptron Solution: Assign **Weights** based on importance.

- w_1 (Engine) $= 5$
- w_2 (Brakes) $= \mathbf{10}$ (Critical!)
- w_3 (Cup Holder) $= 1$
- Bias $b = -8$ (Threshold of 8)

Scenario 1: No Brakes

Engine Good (1), No Brakes (0), Cup Holder (1).

$$\sum = (5 \times 1) + (10 \times 0) + (1 \times 1) - 8$$

$$\sum = 6 - 8 = -2$$

$$-2 < 0 \implies \textbf{Don't Buy (0)}$$

Scenario 2: Good Car

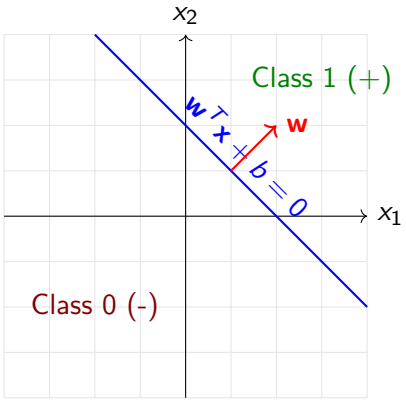
No Cup Holder (0), but Good Brakes (1).

$$\sum = (5 \times 1) + (10 \times 1) + (1 \times 0) - 8$$

$$\sum = 7$$

$$7 \geq 0 \implies \textbf{Buy (1)}$$

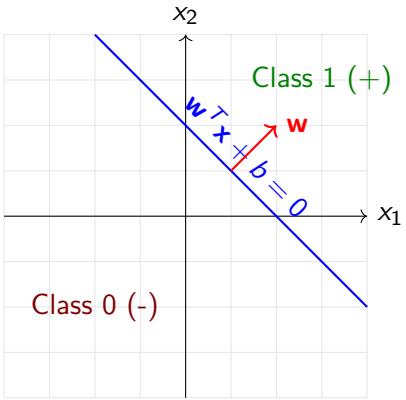
Geometric Interpretation: The Hyperplane



The Decision Boundary

- The equation $\mathbf{w} \cdot \mathbf{x} + b = 0$ defines a **line** (in 2D) or a **hyperplane** (in nD).

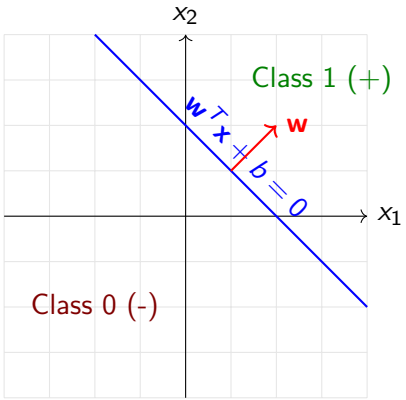
Geometric Interpretation: The Hyperplane



The Decision Boundary

- The equation $\mathbf{w} \cdot \mathbf{x} + b = 0$ defines a **line** (in 2D) or a **hyperplane** (in nD).
- This separates the input space into two regions:
  - $\mathbf{w} \cdot \mathbf{x} + b > 0 \implies$ Class 1
  - $\mathbf{w} \cdot \mathbf{x} + b < 0 \implies$ Class 0

Geometric Interpretation: The Hyperplane



The Decision Boundary

- The equation $\mathbf{w} \cdot \mathbf{x} + b = 0$ defines a **line** (in 2D) or a **hyperplane** (in nD).
- This separates the input space into two regions:
  - $\mathbf{w} \cdot \mathbf{x} + b > 0 \implies$ Class 1
  - $\mathbf{w} \cdot \mathbf{x} + b < 0 \implies$ Class 0

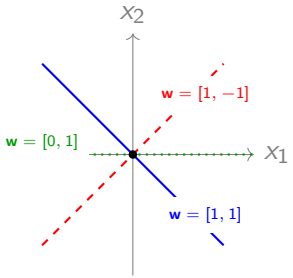
The Weight Vector (w)

- The vector $\mathbf{w}$ is always **orthogonal** (perpendicular) to the decision boundary.
- It points in the direction of the "Yes" (1) class.

Geometric Intuition: Rotation vs. Translation

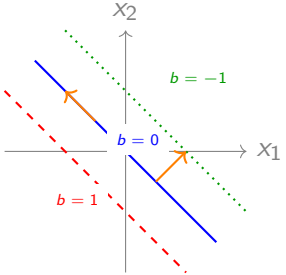
1. Changing Weights (w)

*Controls the **Orientation***



2. Changing Bias (b)

*Controls the **Position***



Rotation

Changing **w rotates** the boundary around the center. It changes *which feature matters more*.

Translation

Changing b **shifts** the boundary without rotating it. It changes the *threshold required to fire*.

The Role of Bias (w_0): Shifting the World

Why do we need a Bias?

- Without a bias ($b = 0$), the decision boundary **must pass through the origin** $(0, 0)$.
- The equation becomes $\mathbf{w} \cdot \mathbf{x} = 0$.
- Problem: What if the data isn't centered at the origin?

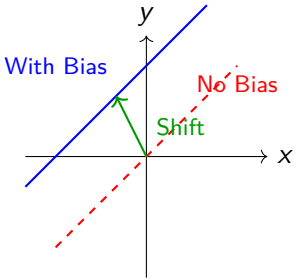
The Role of Bias (w_0): Shifting the World

Why do we need a Bias?

- Without a bias ($b = 0$), the decision boundary **must pass through the origin** $(0, 0)$.
- The equation becomes $\mathbf{w} \cdot \mathbf{x} = 0$.
- Problem: What if the data isn't centered at the origin?

The Solution:

- The Bias b acts as an "offset."
- It shifts the activation function left or right.
- It allows the decision boundary to move **off the origin** to fit the data better.



An Intuitive "Personality" Check

Think of the weighted sum ($\sum w_i x_i$) as
External Evidence.

- "Is the food good?"
- "Is it cheap?"

An Intuitive "Personality" Check

Think of the weighted sum ($\sum w_i x_i$) as **External Evidence**.

- "Is the food good?"
- "Is it cheap?"

Think of the **Bias (**b **or** w_0**)** as your **Internal Disposition**.

- It represents how easy (or hard) it is to trigger you, *regardless of the evidence*.

An Intuitive "Personality" Check

Think of the weighted sum ($\sum w_i x_i$) as **External Evidence**.

- "Is the food good?"
- "Is it cheap?"

Think of the **Bias (b or w_0)** as your **Internal Disposition**.

- It represents how easy (or hard) it is to trigger you, *regardless of the evidence*.

The Golden Rule

$\text{Evidence} + \text{Bias} > 0 \implies \text{Action}$

An Intuitive "Personality" Check

Think of the weighted sum ($\sum w_i x_i$) as **External Evidence**.

- "Is the food good?"
- "Is it cheap?"

Think of the **Bias (b or w_0)** as your **Internal Disposition**.

- It represents how easy (or hard) it is to trigger you, *regardless of the evidence*.

The Golden Rule

Evidence + Bias $> 0 \implies$ Action

Example: "Should I go to the Party?"

An Intuitive "Personality" Check

Think of the weighted sum ($\sum w_i x_i$) as **External Evidence**.

- "Is the food good?"
- "Is it cheap?"

Think of the **Bias (b or w_0)** as your **Internal Disposition**.

- It represents how easy (or hard) it is to trigger you, *regardless of the evidence*.

The Golden Rule

Evidence + Bias $> 0 \implies$ Action

Example: "Should I go to the Party?"

The Extrovert (High Positive Bias)

Bias (b) = +5

"I love parties! I'm already halfway out the door."

Even if the music is bad (Input $x = -2$),
$-2 + 5 = \mathbf{3} > 0 \implies$ **GO!**

An Intuitive "Personality" Check

Think of the weighted sum ($\sum w_i x_i$) as **External Evidence**.

- "Is the food good?"
- "Is it cheap?"

Think of the **Bias (b or w_0)** as your **Internal Disposition**.

- It represents how easy (or hard) it is to trigger you, *regardless of the evidence*.

The Golden Rule

Evidence + Bias $> 0 \implies$ Action

Example: "Should I go to the Party?"

The Extrovert (High Positive Bias)

Bias (b) = +5

"I love parties! I'm already halfway out the door."

Even if the music is bad (Input $x = -2$), $-2 + 5 = \mathbf{3} > 0 \implies$ **GO!**

The Homebody (High Negative Bias)

Bias (b) = -5

"I'd rather stay home. You need to convince me."

The music must be AMAZING (Input $x = +6$) just to get them to move, $6 - 5 = \mathbf{1} > 0 \implies$ **GO.**

An Intuitive "Personality" Check

Think of the weighted sum ($\sum w_i x_i$) as **External Evidence**.

- "Is the food good?"
- "Is it cheap?"

Think of the **Bias (b or w_0)** as your **Internal Disposition**.

- It represents how easy (or hard) it is to trigger you, *regardless of the evidence*.

The Golden Rule

Evidence + Bias $> 0 \implies$ Action

Example: "Should I go to the Party?"

The Extrovert (High Positive Bias)

Bias (b) = +5

"I love parties! I'm already halfway out the door."

Even if the music is bad (Input $x = -2$), $-2 + 5 = \mathbf{3} > 0 \implies$ **GO!**

The Homebody (High Negative Bias)

Bias (b) = -5

"I'd rather stay home. You need to convince me."

The music must be AMAZING (Input $x = +6$) just to get them to move, $6 - 5 = \mathbf{1} > 0 \implies$ **GO.**

*Bias adjusts the **threshold of activation**. It allows the neuron to say "Yes" easily or make it very strict.*

The Learning Problem

The Learning Problem

The Goal:

Find a weight vector $\mathbf{w}$ and bias b such that:

- For all Positive examples ($x \in P$): $\mathbf{w} \cdot \mathbf{x} + b \geq 0$
- For all Negative examples ($x \in N$): $\mathbf{w} \cdot \mathbf{x} + b < 0$

The Learning Problem

The Goal:
Find a weight vector $\mathbf{w}$ and bias b such that:

- For all Positive examples ($x \in P$): $\mathbf{w} \cdot \mathbf{x} + b \geq 0$
- For all Negative examples ($x \in N$): $\mathbf{w} \cdot \mathbf{x} + b < 0$

The Challenge:
We start with random weights. The line is wrong. How do we move it to the right place?

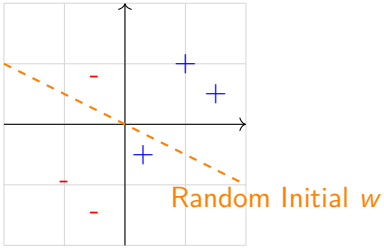
The Learning Problem

The Goal:
Find a weight vector $\mathbf{w}$ and bias b such that:

- For all Positive examples ($x \in P$): $\mathbf{w} \cdot \mathbf{x} + b \geq 0$
- For all Negative examples ($x \in N$): $\mathbf{w} \cdot \mathbf{x} + b < 0$

The Challenge:
We start with random weights. The line is wrong. How do we move it to the right place?

The Approach:
Iterative Error Correction. We loop through the data, and every time the model makes a mistake, we "nudge" the weights.



The Perceptron Learning Algorithm

Input: Training data $D = \{(x, y)\}$

Initialize: $\mathbf{w} \leftarrow \mathbf{0}$ (or random small numbers)

Loop until convergence (no errors/mistake):

For each pair $(\mathbf{x}, y)$ in D:

1. **Predict:** $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$ *(1 if ≥ 0, else 0)*
2. **Update:** If $\hat{y} \neq y$ (Mistake!):
   - If True is Positive ($y = 1$) but predicted Negative: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}$
   - If True is Negative ($y = 0$) but predicted Positive: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}$

Note: We absorb bias b into $\mathbf{w}$ *by adding a '1' to every input vector* $\mathbf{x}$.

The Intuition: Steering the Weight Vector

The Goal

We want the weight vector **w** to point in the "correct" direction relative to our input data **x**.

- For **Positive** examples ($y = 1$), **w** should point *roughly* in the same direction as **x** (angle $< 90°$).
- For **Negative** examples ($y = 0$), **w** should point *away* from **x** (angle $> 90°$).

When the model makes a mistake, we simply "nudge" w to fix the angle.

Too far away?	**Too close?**
Pull it closer $(+)$	Push it away $(-)$

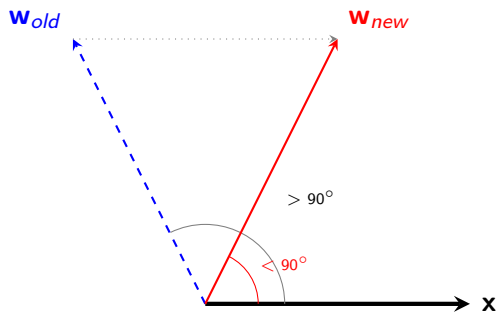
Case 1: False Negative (The "Pull" Effect)

The Error:

- Input $\mathbf{x}$ is **Positive** ($y = 1$).
- Model predicted **Negative** ($\mathbf{w} \cdot \mathbf{x} < 0$).
- Problem: Angle is too wide ($> 90°$).

The Fix: $\mathbf{w}_{new} \leftarrow \mathbf{w} + \mathbf{x}$

- We **add** the input vector to the weights.
- This pulls **w towards x**.
- **Result:** Angle decreases, making $\mathbf{w} \cdot \mathbf{x}$ more positive.



Angle Reduced!

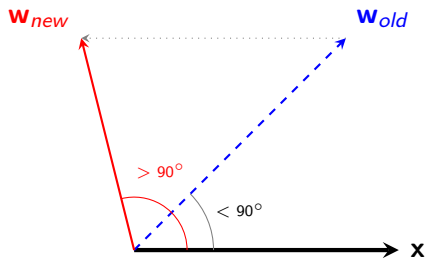
Case 2: False Positive (The "Push" Effect)

The Error:

- Input $\mathbf{x}$ is **Negative** ($y = 0$).
- Model predicted **Positive** ($\mathbf{w} \cdot \mathbf{x} > 0$).
- Problem: Angle is too narrow ($< 90°$).

The Fix: $\mathbf{w}_{new} \leftarrow \mathbf{w} - \mathbf{x}$

- We **subtract** the input vector.
- This pushes $\mathbf{w}$ **away** from $\mathbf{x}$.
- **Result:** Angle increases, making $\mathbf{w} \cdot \mathbf{x}$ more negative.



Angle Increased!

Mathematical Proof: Why the Dot Product Improves

Let $\mathbf{w}_{new}$ be the updated weight vector. We check the new dot product $\mathbf{w}_{new} \cdot \mathbf{x}$ to see if it moved in the right direction.

Mathematical Proof: Why the Dot Product Improves

Let $\mathbf{w}_{new}$ be the updated weight vector. We check the new dot product $\mathbf{w}_{new} \cdot \mathbf{x}$ to see if it moved in the right direction.

1. False Negative Update ($\mathbf{w} + \mathbf{x}$)

We want the score to **increase** (become more positive).

$$\mathbf{w}_{new} \cdot \mathbf{x} = (\mathbf{w}_{old} + \mathbf{x}) \cdot \mathbf{x}$$

$$= \mathbf{w}_{old} \cdot \mathbf{x} + \mathbf{x} \cdot \mathbf{x}$$

$$= \underbrace{\mathbf{w}_{old} \cdot \mathbf{x}}_{\text{Old Score}} + \underbrace{\|\mathbf{x}\|^2}_{\text{Always Positive}}$$

Conclusion: The score **increases**.

Mathematical Proof: Why the Dot Product Improves

Let $\mathbf{w}_{new}$ be the updated weight vector. We check the new dot product $\mathbf{w}_{new} \cdot \mathbf{x}$ to see if it moved in the right direction.

1. False Negative Update ($\mathbf{w} + \mathbf{x}$)

We want the score to **increase** (become more positive).

$$\mathbf{w}_{new} \cdot \mathbf{x} = (\mathbf{w}_{old} + \mathbf{x}) \cdot \mathbf{x}$$

$$= \mathbf{w}_{old} \cdot \mathbf{x} + \mathbf{x} \cdot \mathbf{x}$$

$$= \underbrace{\mathbf{w}_{old} \cdot \mathbf{x}}_{\text{Old Score}} + \underbrace{\|\mathbf{x}\|^2}_{\text{Always Positive}}$$

Conclusion: The score **increases**.

2. False Positive Update ($\mathbf{w} - \mathbf{x}$)

We want the score to **decrease** (become more negative).

$$\mathbf{w}_{new} \cdot \mathbf{x} = (\mathbf{w}_{old} - \mathbf{x}) \cdot \mathbf{x}$$

$$= \mathbf{w}_{old} \cdot \mathbf{x} - \mathbf{x} \cdot \mathbf{x}$$

$$= \underbrace{\mathbf{w}_{old} \cdot \mathbf{x}}_{\text{Old Score}} - \underbrace{\|\mathbf{x}\|^2}_{\text{Always Positive}}$$

Conclusion: The score **decreases**.

Mathematical Proof: Why the Dot Product Improves

Let $\mathbf{w}_{new}$ be the updated weight vector. We check the new dot product $\mathbf{w}_{new} \cdot \mathbf{x}$ to see if it moved in the right direction.

1. False Negative Update ($\mathbf{w} + \mathbf{x}$)

We want the score to **increase** (become more positive).

$$\mathbf{w}_{new} \cdot \mathbf{x} = (\mathbf{w}_{old} + \mathbf{x}) \cdot \mathbf{x}$$

$$= \mathbf{w}_{old} \cdot \mathbf{x} + \mathbf{x} \cdot \mathbf{x}$$

$$= \underbrace{\mathbf{w}_{old} \cdot \mathbf{x}}_{\text{Old Score}} + \underbrace{\|\mathbf{x}\|^2}_{\text{Always Positive}}$$

Conclusion: The score **increases**.

2. False Positive Update ($\mathbf{w} - \mathbf{x}$)

We want the score to **decrease** (become more negative).

$$\mathbf{w}_{new} \cdot \mathbf{x} = (\mathbf{w}_{old} - \mathbf{x}) \cdot \mathbf{x}$$

$$= \mathbf{w}_{old} \cdot \mathbf{x} - \mathbf{x} \cdot \mathbf{x}$$

$$= \underbrace{\mathbf{w}_{old} \cdot \mathbf{x}}_{\text{Old Score}} - \underbrace{\|\mathbf{x}\|^2}_{\text{Always Positive}}$$

Conclusion: The score **decreases**.

Since $\mathbf{w} \cdot \mathbf{x} \propto \cos\theta$, increasing the dot product decreases the angle θ, and vice versa.

Real Inputs, Rigid Boundaries

The Misconception

"Since we use real numbers $(0.75, -1.2, \dots)$ instead of just 0 and 1, can't we solve complex problems?"

Real Inputs, Rigid Boundaries

The Misconception

"Since we use real numbers $(0.75, -1.2, \dots)$ instead of just 0 and 1, can't we solve complex problems?"

The Reality Check: Recall the Perceptron equation:

$$\sum w_i x_i + b = 0$$

- This is the definition of a **Linear Hyperplane**.
- In 2D, it is a straight line ($y = mx + c$).
- It **cannot bend**, curve, or encircle data.

Real Inputs, Rigid Boundaries

The Misconception

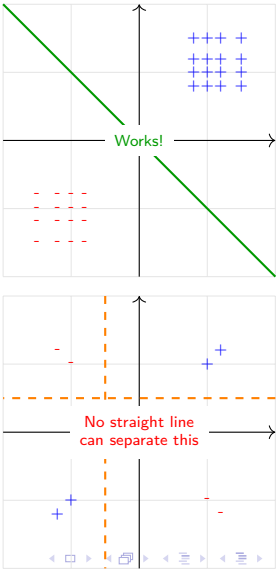
"Since we use real numbers $(0.75, -1.2, \dots)$ instead of just 0 and 1, can't we solve complex problems?"

The Reality Check: Recall the Perceptron equation:

$$\sum w_i x_i + b = 0$$

- This is the definition of a **Linear Hyperplane**.
- In 2D, it is a straight line ($y = mx + c$).
- It **cannot bend**, curve, or encircle data.

Changing the domain from Binary to Real only fills the space; it doesn't bend the ruler.



Perceptron: Strengths & Limitations

Strengths

Perceptron: Strengths & Limitations

Strengths

- **It Learns!**
  An automatic learning rule adjusts weights from data.

Perceptron: Strengths & Limitations

Strengths

- **It Learns!**
  An automatic learning rule adjusts weights from data.
- **Feature Importance:**
  Learns which inputs are more important via weights.

Perceptron: Strengths & Limitations

Strengths

- **It Learns!**
  An automatic learning rule adjusts weights from data.
- **Feature Importance:**
  Learns which inputs are more important via weights.
- **Convergence Guarantee:**
  If data is linearly separable, it's guaranteed to find a solution.

Perceptron: Strengths & Limitations

Strengths

- **It Learns!**
  An automatic learning rule adjusts weights from data.
- **Feature Importance:**
  Learns which inputs are more important via weights.
- **Convergence Guarantee:**
  If data is linearly separable, it's guaranteed to find a solution.

Limitations

Perceptron: Strengths & Limitations

Strengths

- **It Learns!**
  An automatic learning rule adjusts weights from data.
- **Feature Importance:**
  Learns which inputs are more important via weights.
- **Convergence Guarantee:**
  If data is linearly separable, it's guaranteed to find a solution.

Limitations

- **Linear Separability** Only problems where a single straight line (or hyperplane) can separate the classes can be solved.

Perceptron: Strengths & Limitations

Strengths

- **It Learns!**
  An automatic learning rule adjusts weights from data.
- **Feature Importance:**
  Learns which inputs are more important via weights.
- **Convergence Guarantee:**
  If data is linearly separable, it's guaranteed to find a solution.

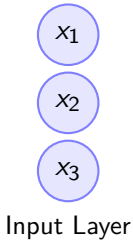
Limitations

- **Linear Separability** Only problems where a single straight line (or hyperplane) can separate the classes can be solved.
- **Harsh Threshold:** The step function activation is not differentiable, preventing modern gradient-based training.

From Neuron to Network: Multilayer Perceptrons (MLP)

1. Input Layer

- The raw data ($\mathbf{x}$).
- No computation happens here; it just passes values forward.



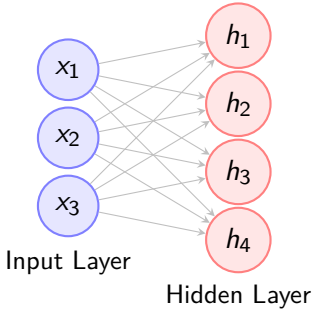
From Neuron to Network: Multilayer Perceptrons (MLP)

1. Input Layer

- The raw data (**x**).
- No computation happens here; it just passes values forward.

2. Hidden Layer(s)

- The core innovation.
- These neurons are "hidden" from the outside world.
- They transform the input into **new features**.



From Neuron to Network: Multilayer Perceptrons (MLP)

1. Input Layer

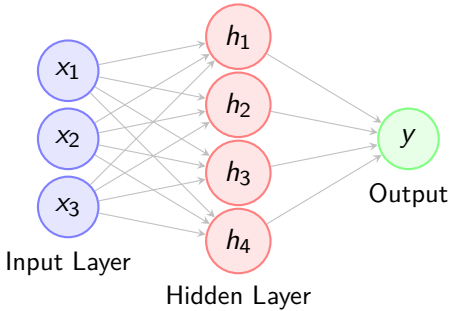
- The raw data (**x**).
- No computation happens here; it just passes values forward.

2. Hidden Layer(s)

- The core innovation.
- These neurons are "hidden" from the outside world.
- They transform the input into **new features**.

3. Output Layer

- The final decision ($\hat{y}$).
- Combines the features from the



How Hidden Layers Solve the Impossible

The Logic: "Divide and Conquer"

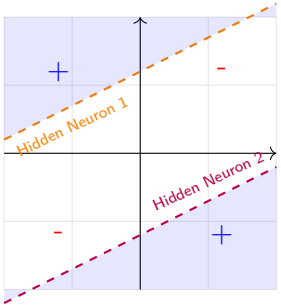
A single Perceptron draws **one line**. It fails at XOR.

How Hidden Layers Solve the Impossible

The Logic: "Divide and Conquer"
A single Perceptron draws **one line**. It fails at XOR.

A Network draws multiple lines:

1. **Hidden Neuron 1:** Draws Line A to separate the top-left.
2. **Hidden Neuron 2:** Draws Line B to separate the bottom-right.
3. **Output Neuron:** Combines them (e.g., "Active if Line A **OR** Line B says yes").



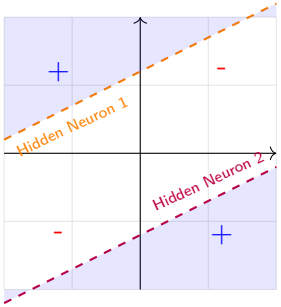
Two lines creating a non-linear decision boundary.

How Hidden Layers Solve the Impossible

The Logic: "Divide and Conquer"
A single Perceptron draws **one line**. It fails at XOR.

A Network draws multiple lines:

1. **Hidden Neuron 1:** Draws Line A to separate the top-left.
2. **Hidden Neuron 2:** Draws Line B to separate the bottom-right.
3. **Output Neuron:** Combines them (e.g., "Active if Line A **OR** Line B says yes").



Two lines creating a non-linear decision boundary.

Key Insight

Hidden layers transform the space so that the problem becomes linearly separable for the output layer.

Visualizing the Transformation: XOR Solved

Let's use a simple MLP (one hidden layer and two neurons) with step activation functions (output is 0 or 1).

Visualizing the Transformation: XOR Solved

Let's use a simple MLP (one hidden layer and two neurons) with step activation functions (output is 0 or 1).

- **Hidden 1 (h_1):** Acts like OR ($x_1 + x_2 \geq 0.5$) - $w_0 = -0.5, w_1 = 1, w_2 = 1$

Visualizing the Transformation: XOR Solved

Let's use a simple MLP (one hidden layer and two neurons) with step activation functions (output is 0 or 1).

- **Hidden 1 (h_1):** Acts like OR ($x_1 + x_2 \geq 0.5$) - $w_0 = -0.5, w_1 = 1, w_2 = 1$
- **Hidden 2 (h_2):** Acts like AND ($x_1 + x_2 \geq 1.5$) - $w_0 = -1.5, w_1 = 1, w_2 = 1$

Visualizing the Transformation: XOR Solved

Let's use a simple MLP (one hidden layer and two neurons) with step activation functions (output is 0 or 1).

- **Hidden 1 (h_1):** Acts like OR ($x_1 + x_2 \geq 0.5$) - $w_0 = -0.5, w_1 = 1, w_2 = 1$
- **Hidden 2 (h_2):** Acts like AND ($x_1 + x_2 \geq 1.5$) - $w_0 = -1.5, w_1 = 1, w_2 = 1$
- **Output (y):** Computes $h_1 - h_2 \geq 0.5$ (essentially "h_1 AND NOT h_2")

Visualizing the Transformation: XOR Solved

Let's use a simple MLP (one hidden layer and two neurons) with step activation functions (output is 0 or 1).

- **Hidden 1 (h_1):** Acts like OR ($x_1 + x_2 \geq 0.5$) - $w_0 = -0.5, w_1 = 1, w_2 = 1$
- **Hidden 2 (h_2):** Acts like AND ($x_1 + x_2 \geq 1.5$) - $w_0 = -1.5, w_1 = 1, w_2 = 1$
- **Output (y):** Computes $h_1 - h_2 \geq 0.5$ (essentially "h_1 AND NOT h_2")

1. The Computation Trace

Input Space		**Hidden Space**		**Output**
x_1	x_2	h_1	h_2	y
0	0	**0**	**0**	**0**
0	1	**1**	**0**	**1**
1	0	**1**	**0**	**1**
1	1	**1**	**1**	**0**

Visualizing the Transformation: XOR Solved

Let's use a simple MLP (one hidden layer and two neurons) with step activation functions (output is 0 or 1).

- **Hidden 1 (h_1):** Acts like OR ($x_1 + x_2 \geq 0.5$) - $w_0 = -0.5, w_1 = 1, w_2 = 1$
- **Hidden 2 (h_2):** Acts like AND ($x_1 + x_2 \geq 1.5$) - $w_0 = -1.5, w_1 = 1, w_2 = 1$
- **Output (y):** Computes $h_1 - h_2 \geq 0.5$ (essentially "h_1 AND NOT h_2")

1. The Computation Trace

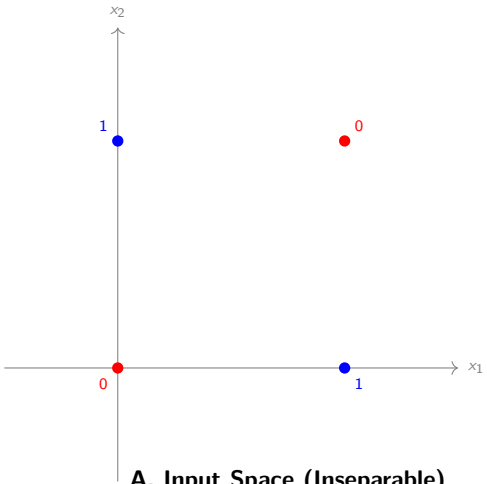
Input Space		Hidden Space		Output
x_1	x_2	h_1	h_2	y
0	0	**0**	**0**	**0**
0	1	**1**	**0**	**1**
1	0	**1**	**0**	**1**
1	1	**1**	**1**	**0**

Key Observation

Look at the inputs $(0, 1)$ and $(1, 0)$. They are distant in input space, but the hidden layer maps them to the **exact same point** $(1, 0)$ in hidden space.

Visualizing the Transformation: XOR Solved

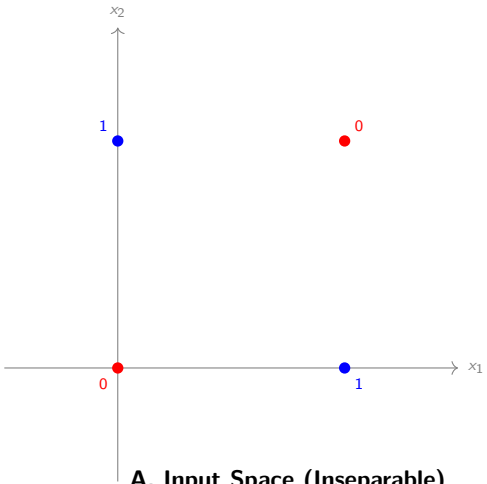
2. The Geometric Transformation



A. Input Space (Inseparable)

Visualizing the Transformation: XOR Solved

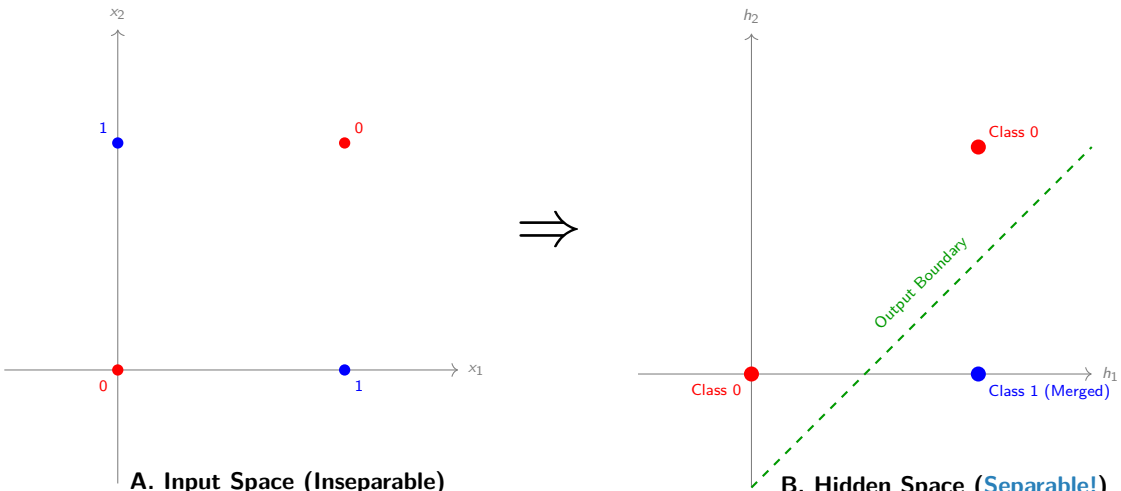
2. The Geometric Transformation



A. Input Space (Inseparable)

Visualizing the Transformation: XOR Solved

2. The Geometric Transformation



Weights in a Network: Enter Linear Algebra

In a single perceptron, weights were a **vector** ($\mathbf{w}$).

In a network, every layer is connected to the next.

- If Layer 1 has m neurons...
- And Layer 2 has n neurons...
- We need $m \times n$ connections.

Weights in a Network: Enter Linear Algebra

In a single perceptron, weights were a **vector** ($\mathbf{w}$).

In a network, every layer is connected to the next.

- If Layer 1 has m neurons...
- And Layer 2 has n neurons...
- We need $m \times n$ connections.

The Weight Matrix W:

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Where $\mathbf{W}$ is a matrix of size $(n \times m)$.

- Row i contains the weights for Neuron i in the hidden layer.
- It learns to detect **one specific feature**.

Weights in a Network: Enter Linear Algebra

In a single perceptron, weights were a **vector** ($\mathbf{w}$).

In a network, every layer is connected to the next.

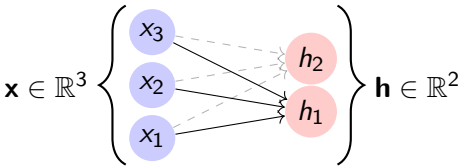
- If Layer 1 has m neurons...
- And Layer 2 has n neurons...
- We need $m \times n$ connections.

The Weight Matrix W:

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Where $\mathbf{W}$ is a matrix of size $(n \times m)$.

- Row i contains the weights for Neuron i in the hidden layer.
- It learns to detect **one specific feature**.



$\mathbf{W}$ is 2×3

A Provocative Thought: "The lego block of intelligence?"

Think about what we just did.

- We used **2 neurons** to carve out a specific shape (XOR) by combining straight lines.
- We essentially created a "cut-out" in space.

A Provocative Thought: "The lego block of intelligence?"

Think about what we just did.

- We used **2 neurons** to carve out a specific shape (XOR) by combining straight lines.
- We essentially created a "cut-out" in space.

The Big Question

If 2 neurons can carve out an XOR shape... what could we carve out with **100 neurons**? Or **1,000,000**?

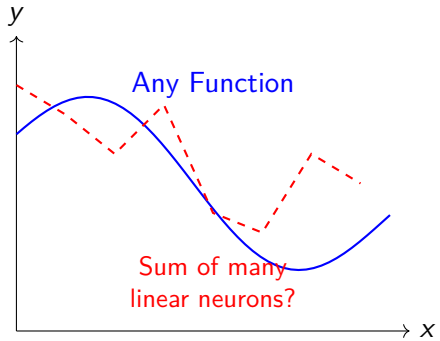
A Provocative Thought: "The lego block of intelligence?"

Think about what we just did.

- We used **2 neurons** to carve out a specific shape (XOR) by combining straight lines.
- We essentially created a "cut-out" in space.

The Big Question

If 2 neurons can carve out an XOR shape... what could we carve out with **100 neurons**? Or **1,000,000**?



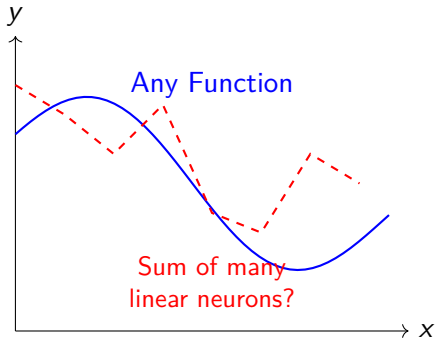
A Provocative Thought: "The lego block of intelligence?"

Think about what we just did.

- We used **2 neurons** to carve out a specific shape (XOR) by combining straight lines.
- We essentially created a "cut-out" in space.

The Big Question

If 2 neurons can carve out an XOR shape... what could we carve out with **100 neurons**? Or **1,000,000**?



The Hypothesis: Maybe, just maybe, if we have enough neurons, we can approximate *any* shape, *any* decision boundary, or *any* mathematical function in the universe.

Looking Ahead: From Logic to Calculus

Module 1B: Representation & Optimization

- **The Universal Approximation Theorem:**
  Formalizing the idea that MLPs can approximate *any* continuous function.
- **The "Modern" Neuron:**
  Why Step functions are dead ends for learning (Derivative is 0 or ∞).
- **Enter Calculus:**
  Transitioning to differentiable activation functions: **Sigmoid**, Tanh, and ReLU.

Next Lecture: How do we teach the network automatically?