The Perceptron

Vincent Boyer

Graduate Program in Artificial Intelligence
and Optimization
vincent.boyer@uanl.edu.mx
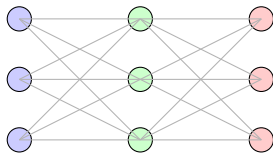
June 12, 2025

Why Study the Perceptron?

- The perceptron is the simplest form of a neural network.
- It serves as a foundation for understanding more complex models.
- Understanding the perceptron helps grasp key concepts in machine learning and AI.
- Mathematics as the foundation of AI

Brain　　　　　Neural Network

Figure: Brains and Neural Networks

A Brief History of AI and Neural Networks

- **1950:** Alan Turing proposes the concept of a thinking machine.
- **1957:** Frank Rosenblatt invents the perceptron, the first artificial neural network.
- **1969:** Marvin Minsky and Seymour Papert publish "Perceptrons," highlighting limitations (e.g., XOR problem), leading to the first AI Winter.
- **1974:** Paul Werbos introduces the concept of backpropagation for training neural networks.
- **1986:** Rumelhart, Hinton, and Williams popularize backpropagation, enabling multi-layer networks.

Turing Test	Perceptron	Perceptron Limitations	Backprop Concept	Backprop Popularized
1950	1957	1969	1974	1986
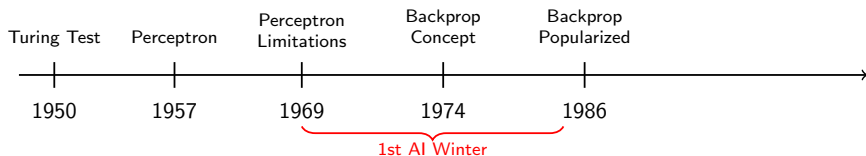
1st AI Winter

A Brief History of AI and Neural Networks

- **1987**-**1993:** Second AI Winter due to limited progress, funding, and computational power.
- **1998:** Yann LeCun develops LeNet-5, a pioneering convolutional neural network for digit recognition.
- **2006:** Geoffrey Hinton and colleagues introduce deep belief networks, sparking the deep learning revolution.
- **2012:** AlexNet wins the ImageNet competition, demonstrating the power of deep convolutional networks.
- **2018+:** Transformers and large language models (e.g., BERT, GPT) revolutionize AI applications.

Backprop Popularized   LeNet-5   Deep Belief Networks   AlexNet   Transformers

1986   1998   2006   2012   2018+

2nd AI Winter
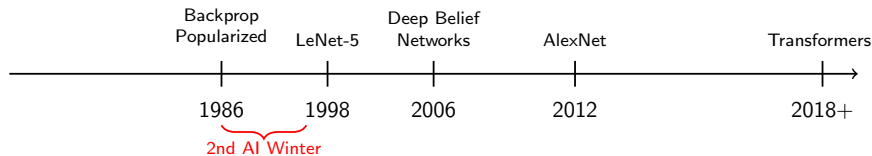
The Perceptron

- A perceptron is a simple model of a neuron
- Linear classifier
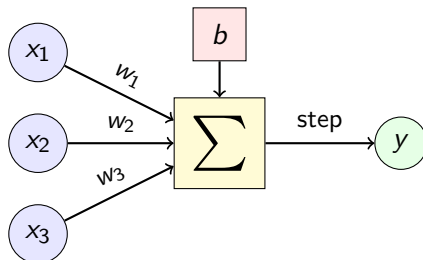- Foundation for modern neural networks

Figure: Perceptron

The Perceptron

- Inputs $x_1, x_2, ..., x_n$
- Weights $w_1, w_2, ..., w_n$, bias b
- Activation: $y = f(\sum w_i x_i + b)$
- f is the activation function (e.g. step function in the original perceptron)
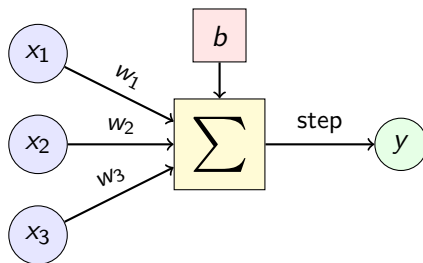

Figure: Perceptron
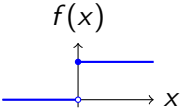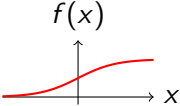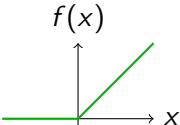
Classical Activation Functions

Activation Function	Graph
Step Function $$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$	$f(x)$ x
Sigmoid Function $f(x) = \frac{1}{1+e^{-x}}$	$f(x)$ x
ReLU (Rectified Linear Unit) $f(x) = \max(0, x)$	$f(x)$ x

Table: Classical Activation Functions and Their Graphs

Learning in a Perceptron

What is Learning?

- Learning in a perceptron involves adjusting **weights** based on input data to minimize error.
- It follows the rule:

$$w_{\text{new}} = w_{\text{old}} + \eta \cdot (\text{error}) \cdot X$$

where:

- η = learning rate
- error = $y_{\text{true}} - y_{\text{pred}}$
- X = input features

Evaluation:

- The perceptron learning algorithm evaluates its performance by checking how many inputs it classifies correctly.
- A loss function is used to quantify the error in predictions.

Challenges in Training

- **Linearly Separable Data:** The perceptron can only classify linearly separable data.
- **Overfitting:** The model may overfit to the training data if not properly regularized.
- **Underfitting:** The model may be too simple to capture the underlying patterns in the data, leading to poor performance on both training and test data.
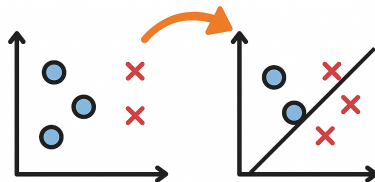
Loss Functions

The perceptron learns by adjusting weights based on the error in predictions. The loss function quantifies this error, guiding the learning process.

Loss Function	Typical Application
0-1 Loss $$L(y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & y \neq \hat{y} \end{cases}$$	Classification (Perceptron, SVM)
Squared Error Loss $$L(y, \hat{y}) = (y - \hat{y})^2$$	Regression, Linear Models
Log Loss (Cross-Entropy) $$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$	Logistic Regression, Neural Networks

Table: Common Loss Functions and Their Applications, where y is the true label and $\hat{y}$ is the predicted value.

Gradient Descent and Optimization

The gradient descent algorithm is a fundamental optimization technique used to minimize the loss function in machine learning models, including the perceptron. It works by iteratively adjusting the model parameters (weights) in the direction of the steepest descent of the loss function.

Gradient for Multivariable Functions

For a function $f(\mathbf{w})$ where $\mathbf{w} = (w_1, w_2, ..., w_n)$, the gradient is the vector of partial derivatives:

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \dots, \frac{\partial f}{\partial w_n} \right)$$

It points in the direction of steepest increase of f.

Gradient Descent and Optimization
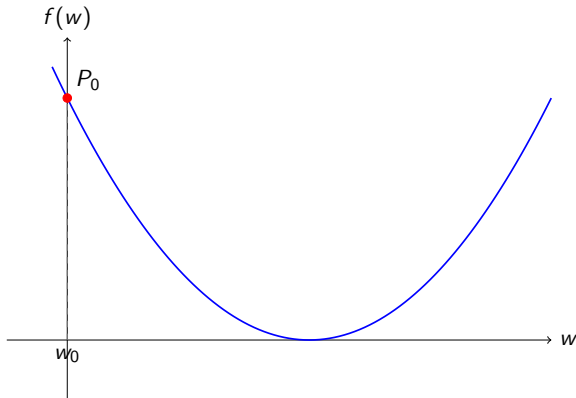
The gradient descent update rule is:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla f(\mathbf{w}_t)$$

where:

- η is the learning rate,
- $f(\mathbf{w})$ is the (loss) function.
- $\mathbf{w}_t$ is the weight vector at iteration t.
- $\mathbf{w}_{t+1}$ is the updated weight vector.

Gradient Descent Example

- Consider a simple quadratic function: $f(w) = (w-2)^2$.
- The gradient (derivative) is $f'(w) = 2(w-2)$.
- We set the learning rate η to a small value, e.g., $\eta = 0.5$.
- Starting at $w_0 = 0$ (Point $P_0 = (0, 4)$)

Gradient Descent Example

- Consider a simple quadratic function: $f(w) = (w-2)^2$.
- The gradient (derivative) is $f'(w) = 2(w-2)$.
- We set the learning rate η to a small value, e.g., $\eta = 0.5$.
- Then we go to $P_1(w_1, f(w_1))$ with

$$w_1 = w_0 - \eta f'(w_0) = 0 - 0.5 \cdot 2(0-2) = 0 + 1 = 1$$
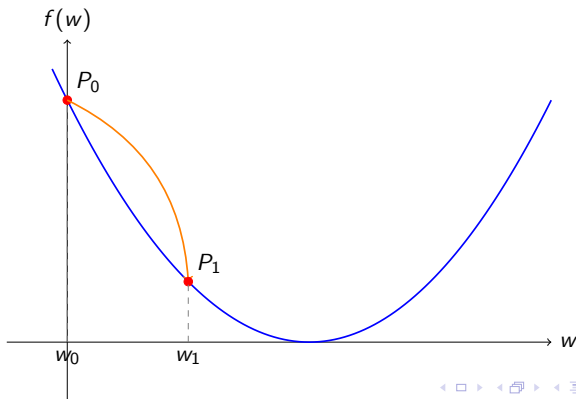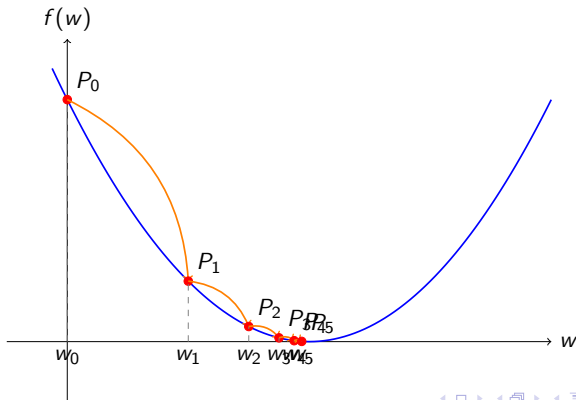
Gradient Descent Example

- Consider a simple quadratic function: $f(w) = (w-2)^2$.
- The gradient (derivative) is $f'(w) = 2(w-2)$.
- We set the learning rate η to a small value, e.g., $\eta = 0.5$.
- And we repeat this process for several iterations, updating w each time, until the change in w becomes negligible.
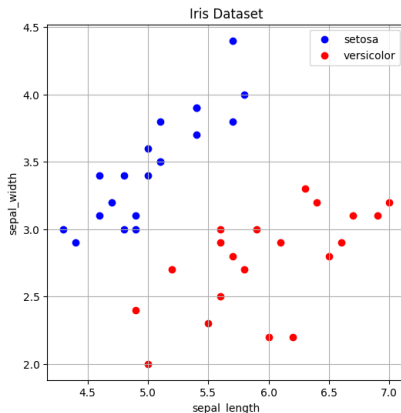
Example: Iris Dataset

The Iris dataset is a classic dataset in machine learning, consisting of 150 samples from three species of iris flowers: setosa, versicolor, and virginica. Each sample has four features: sepal length, sepal width, petal length, and petal width.

- **Objective:** Classify the species of an iris flower based on its measured features.
- **Features:** Sepal length, sepal width, petal length, petal width.
- **Classes:** Setosa, Versicolor, Virginica.

Example: Iris Dataset

For perceptron classification, we often select two classes (e.g., setosa vs. versicolor) and two features for visualization (sepal length and sepal width), aiming to find a linear boundary that separates the classes.

Example: Iris Dataset
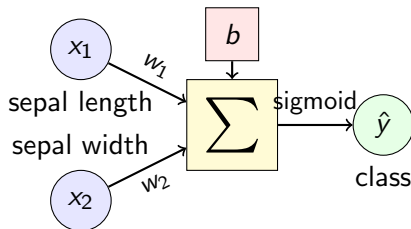
$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$$

For this example we will use the sigmoid activation function, defined as $\sigma(z) = \frac{1}{1+e^{-z}}$.

We will use the mean squared error (MSE) as the loss function, defined as:

$$L(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where y_i is the true label and $\hat{y}_i$ is the predicted label for the i-th sample.

Example: Iris Dataset

The gradient of the loss function with respect to the parameters (weights and bias) is a vector:

$$\nabla L = \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial b} \right)$$

where each partial derivative is computed as:

$$\frac{\partial L}{\partial w_1} = \frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_{1i} \sigma'(w_1 x_{1i} + w_2 x_{2i} + b)$$

$$\frac{\partial L}{\partial w_2} = \frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_{2i} \sigma'(w_1 x_{1i} + w_2 x_{2i} + b)$$

$$\frac{\partial L}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \sigma'(w_1 x_{1i} + w_2 x_{2i} + b)$$

where $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ is the derivative of the sigmoid function.

Example: Iris Dataset

- Initialize weights and bias randomly.
- For each training sample:
  - Compute the predicted output $\hat{y}$ using the perceptron formula.
  - Calculate the loss using the MSE.
  - Compute the gradient of the loss with respect to the weights and bias.
  - Update the weights and bias using the gradient descent rule:

$$w_j \leftarrow w_j - \eta \frac{\partial L}{\partial w_j}$$

$$b \leftarrow b - \eta \frac{\partial L}{\partial b}$$

  where η is the learning rate.
- Repeat for a number of epochs or until convergence.

Example: Iris Dataset

After training, the perceptron can classify new samples by computing the weighted sum of the features and applying the activation function. The decision boundary can be visualized in a 2D plot, where the two classes are separated by a line of the form $w_1 x_1 + w_2 x_2 + b = 0$.
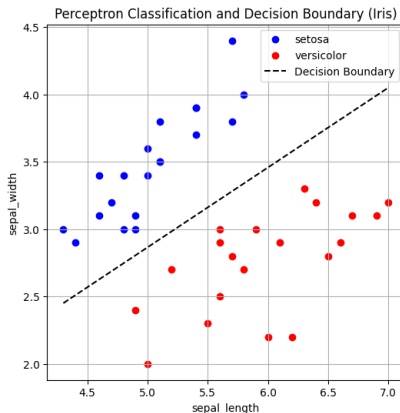
The XOR Limitation of the Perceptron

- The perceptron can only solve problems where the classes are linearly separable.
- The XOR (exclusive OR) problem is not linearly separable:
  - Inputs: $(0,0) \to 0$, $(0,1) \to 1$, $(1,0) \to 1$, $(1,1) \to 0$
  - No straight line can separate the 1s from the 0s in the input space.
- This limitation, highlighted by Minsky and Papert in 1969, led to reduced interest and funding for neural networks (the first AI Winter).
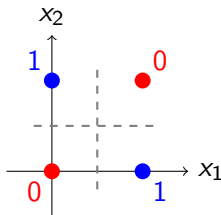

Figure: XOR: No line can separate the classes

From Perceptron to Modern AI

- The perceptron laid the groundwork for modern neural networks.
- Multi-layer networks (deep learning) can solve complex, non-linear problems.
- Key developments:
    - Backpropagation algorithm for training multi-layer networks.
    - Introduction of activation functions (ReLU, tanh, etc.) to introduce non-linearity.
    - Foundation for Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).
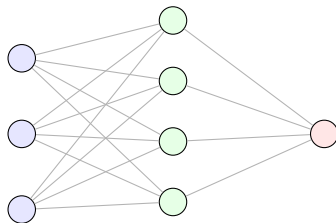

Figure: Evolution to Deep Neural Networks

From Math to Intelligence: The Journey of AI

- **Math Foundations $\rightarrow$ Perceptron $\rightarrow$ Deep Learning & AI**
- From simple neurons to complex networks—small ideas lead to big innovation!
- What are your thoughts? Let's discuss!

"The perceptron was the first step. . . today, AI is reshaping the world."