

Detailed Mathematical Presentation of the Simple Perceptron

1. Introduction

The **Perceptron** is the foundational building block of neural networks, introduced by Frank Rosenblatt in 1958. It is a binary classifier that maps an input vector to a binary output using a linear decision boundary. Despite its simplicity, it illustrates key concepts in supervised learning, linear separability, and iterative weight adjustment.

2. Perceptron Model Components and Notation

- **Input vector:**

$$\mathbf{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$$

where each x_i is a feature or input value.

- **Weight vector:**

$$\mathbf{w} = (w_1, w_2, \dots, w_n)^T \in \mathbb{R}^n$$

representing the synaptic weights associated with each input.

- **Bias:**

$$b \in \mathbb{R}$$

a scalar offset that shifts the decision boundary.

- **Net input (combination function):**

The perceptron computes a weighted sum of inputs plus bias:

$$z = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^n w_i x_i + b$$

- **Activation function:**

The perceptron uses a step function (Heaviside function) to produce the output:

$$y = f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

The output $y \in \{0,1\}$ classifies the input into one of two classes.

3. Geometric Interpretation: Linear Separability

The decision boundary is defined by the hyperplane:

$$\mathbf{w}^T \mathbf{x} + b = 0$$

- Points with $\mathbf{w}^T \mathbf{x} + b > 0$ are classified as 1.
- Points with $\mathbf{w}^T \mathbf{x} + b < 0$ are classified as 0.

This hyperplane separates the input space into two half-spaces, assuming the data is **linearly separable**.

4. Perceptron Learning Algorithm

The perceptron learns the weights $\mathbf{w}$ and bias b through an iterative process based on labeled training data:

- Training set: $\{(\mathbf{x}^{(t)}, d^{(t)})\}_{t=1}^T$ where $d^{(t)} \in \{0,1\}$ is the desired output (label) for input $\mathbf{x}^{(t)}$.

4.1 Prediction and Error

For each training example:

- Compute net input:

$$z^{(t)} = \mathbf{w}^T \mathbf{x}^{(t)} + b$$

- Compute output:

$$y^{(t)} = f(z^{(t)}) = \begin{cases} 1 & z^{(t)} \geq 0 \\ 0 & z^{(t)} < 0 \end{cases}$$

- Compute error:

$$e^{(t)} = d^{(t)} - y^{(t)}$$

4.2 Weight and Bias Update Rule

Weights and bias are updated only if the prediction is incorrect ($\hat{e}^{(t)} \neq 0$):

$$\mathbf{w} \leftarrow \mathbf{w} + \eta e^{(t)} \mathbf{x}^{(t)}$$

$$b \leftarrow b + \eta e^{(t)}$$

where $\eta > 0$ is the **learning rate**, a small positive constant controlling update magnitude.

4.3 Intuition Behind the Update

- If $\hat{y}^{(t)} = 0$ but $d^{(t)} = 1$ (false negative), increase weights in direction of $\mathbf{x}^{(t)}$ to make $z^{(t)}$ larger next time.
- If $\hat{y}^{(t)} = 1$ but $d^{(t)} = 0$ (false positive), decrease weights in direction of $\mathbf{x}^{(t)}$ to reduce $z^{(t)}$.

5. Vectorized Formulation

To simplify notation, augment the input vector with bias as an additional input fixed to 1:

$$\tilde{\mathbf{x}} = (x_1, x_2, \dots, x_n, 1)^T, \tilde{\mathbf{w}} = (w_1, w_2, \dots, w_n, b)^T$$

Then,

$$z = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}$$

and the update rule becomes:

$$\tilde{\mathbf{w}} \leftarrow \tilde{\mathbf{w}} + \eta e^{(t)} \tilde{\mathbf{x}}^{(t)}$$

6. Convergence Theorem

If the training data is **linearly separable**, the perceptron learning algorithm converges in a finite number of steps to weights that correctly classify the training set (Rosenblatt, 1958).

7. Example Calculation

Suppose:

- Inputs: $\mathbf{x} = (1, 0, 1)$
- Initial weights: $\mathbf{w} = (0.7, 0.6, 0.5)$
- Bias: $b = -1.5$
- Learning rate: $\eta = 0.1$
- Desired output: $d = 1$

Calculate:

$$z = 0.7 \times 1 + 0.6 \times 0 + 0.5 \times 1 - 1.5 = 0.7 + 0 + 0.5 - 1.5 = -0.3$$

Output:

$$y = f(z) = 0 \text{ (since } z < 0 \text{)}$$

Error:

$$e = 1 - 0 = 1$$

Update weights:

$$w_1 \leftarrow 0.7 + 0.1 \times 1 \times 1 = 0.8$$

$$w_2 \leftarrow 0.6 + 0.1 \times 1 \times 0 = 0.6$$

$$w_3 \leftarrow 0.5 + 0.1 \times 1 \times 1 = 0.6$$

Bias:

$$b \leftarrow -1.5 + 0.1 \times 1 = -1.4$$

8. Summary of Key Equations

Concept	Equation
Net input (weighted sum)	$z = \mathbf{w}^{\text{top}} \mathbf{x} + b$
Activation function	$y = f(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$
Error	$e = d - y$
Weight update	$\mathbf{w} \leftarrow \mathbf{w} + \eta e \mathbf{x}$
Bias update	$b \leftarrow b + \eta e$
Augmented vector update	$\tilde{\mathbf{w}} \leftarrow \tilde{\mathbf{w}} + \eta e \tilde{\mathbf{x}}$

9. Extensions and Limitations

- The perceptron can only learn **linearly separable** functions.
- Non-linear problems require multi-layer perceptrons (MLPs) or other architectures.
- Activation functions can be generalized beyond step functions to sigmoid, ReLU, etc., enabling gradient-based training.

References

- [Neural Designer: Mathematics of the Perceptron](#)
- [Simplilearn: Perceptron Function](#)
- [W3Schools: Perceptron Algorithm](#)
- Serge Desmedt, *The Math behind Neural Networks: The Rosenblatt Perceptron*, CodeProject, 2019
- Brendon Schultz, *Machine Learning: The mathematics behind the Perceptron model* (YouTube)