PANE Journal of Physics Education P. J. Phys. Edu. Vol 1(2), October 2026

FROM DATA TO DECISION: A TUTORIAL ON ARTIFICIAL INTELLIGENCE

Upashana Sharma and Rupak Mukherjee*

Department of Physics, Sikkim University, Gangtok, India

*Corresponding Author: rmukherjee@cus.ac.in

Abstract: Artificial intelligence (AI) and machine learning (ML) are widely used in modern science and technology. Among various classification algorithms, the Rosenblatt perceptron is one of the earliest and simplest trainable models, forming the foundation of modern neural networks. This work presents a concise introduction to the perceptron model, including its structure, signum activation function, and error-correction learning rule. A physics-motivated worked example drawn from Tesla coil wireless power transfer demonstrates successful training and convergence of the algorithm: the perceptron learns to predict whether a receiver LED glows ($+1$) or does not glow (-1) based solely on the transmitter voltage and the coil-to-LED distance. A Python implementation is also provided to illustrate the development of a basic AI/ML classifier from scratch. The results show that the perceptron is an effective and interpretable method for binary classification, and the learned weight vector acquires a direct physical meaning in terms of the electromagnetic coupling strength.

Keywords: Artificial Intelligence; Machine Learning; Supervised Learning; Perceptron; Error-Correction Learning; Binary Classification; Neural Networks; Linear Separability; Convergence Theorem; Tesla Coil; Wireless Power Transfer; Electromagnetic Induction

(Received: 21 May 2026; accepted: 6 October 2026; published: 30 October 2026)

1 Introduction

Artificial intelligence (AI) refers broadly to the ability of a computational system to perform tasks that would ordinarily require human intelligence, such as recognising patterns, making decisions, and learning from experience [1, 2]. Machine learning (ML) is the subfield of AI concerned with building such systems automatically from data, rather than through hand-crafted rules [3]. Today, ML algorithms are used to detect disease in medical images [4], translate languages [5], recommend content [6], and drive autonomous vehicles [7]. The range of available algorithms is wide: decision trees, k-nearest neighbours, support vector machines, random forests, and deep neural networks, to name a few [8, 9]. Each of these can be thought of as a different strategy for teaching a computer to generalise from examples.

Despite this diversity, nearly every modern ML algorithm for classification shares a common ancestor: the *perceptron*, proposed by Frank Rosenblatt in 1958 [10]. The perceptron is not just a historical curiosity. It is the conceptual and mathematical template from which all later classifiers, including today's deep learning models, were derived [11, 12]. For those writing their first AI/ML program, the perceptron is the ideal starting point: it is simple enough to understand completely, yet rich enough to illustrate every essential idea in supervised learning [13]. By the end of this tutorial, the reader will have implemented a working AI/ML classifier in Python from scratch, one that takes labelled data as input, automatically learns a decision rule, and correctly classifies new examples. This is, in miniature, exactly what every modern ML system does.

The question of whether a machine can learn from experience has occupied scientists and engineers for most of the twentieth century [14]. Traditionally, rules governing physical or computational systems were derived analytically, from first principles or from careful experimental observation. This approach works well when the rules are known in advance, but it becomes difficult to apply when the system is complex, poorly understood, or when the governing structure must be inferred entirely from data [15, 16].

Working at the Cornell Aeronautical Laboratory in the late 1950s, Rosenblatt introduced the *perceptron*: a mathematical model of a single neuron that could not only process inputs and produce outputs, but could also *adjust its own internal parameters* based on whether its output was correct or incorrect. This was the first explicit learning algorithm for an artificial neural unit, and its publication in 1958 is widely regarded as the starting point of the field of neural networks [10, 11].

The perceptron was inspired by two earlier ideas. McCulloch and Pitts (1943) had shown that a neuron could be modeled as a threshold logic unit (a device that fires when the weighted sum of its inputs exceeds a certain value) [17]. Hebb (1949) had proposed that the strength of a synaptic connection increases when two neurons are repeatedly active at the same time [18]. Rosenblatt combined both ideas and added the crucial ingredient that was missing: a systematic rule to modify the weights whenever the neuron made a mistake [10].

The result is an algorithm that is elegant in its simplicity. The perceptron computes a weighted sum of its inputs, applies a sign function to decide which of two classes the input belongs to, and then corrects its weights by a small amount whenever it is wrong. Repeated over many training examples, this process is guaranteed to converge to a correct solution provided that one exists [10, 11, 19].

Although the perceptron is now considered a foundational rather than a state-of-the-art model, understanding it properly is essential for any student of neural networks or machine learning [12, 20]. Every modern architecture, from multi-layer perceptrons to convolutional and recurrent networks and transformers [21, 22, 5], is built from generalized versions of this same unit. As Haykin (2009) puts it, the perceptron is *"the simplest form of a neural network used for the classification of patterns said to be linearly separable"* [11].

In this tutorial, we present a complete and self-contained exposition of the Rosenblatt perceptron as a gateway into supervised ML, motivated by a compelling physics experiment: wireless power transfer using a Tesla coil. The task the perceptron must learn is whether a small LED placed at various distances from the coil will glow or remain dark, given only the transmitter voltage and the separation distance as inputs.

2 The Perceptron: A Foundational ML Algorithm

Before diving into the mathematics, it is worth situating the perceptron within the broader ML landscape [8]. In any supervised classification task, the goal is the same: given a set of labelled training examples, find a function that maps inputs to correct class labels [9]. The perceptron achieves this through iterative, error-driven weight updates, a strategy that underpins virtually every modern neural network [11, 12]. The three key ingredients shared by all such systems are already present here: (i) a *parameterised model* (the weight vector), (ii) a *training signal* (the classification error), and (iii) an *update rule* that improves the model each time it makes a mistake.

2.1 Biological Motivation

Before describing the mathematics, it is worth understanding where the model comes from. A biological neuron receives signals from other neurons through its dendrites, integrates those signals in its cell body, and produces an output signal along its axon if the combined input exceeds a threshold [17, 23]. The key feature is that the strength of each incoming connection (the synaptic weight) can change over time in response to experience. It is this adaptability that makes the brain capable of learning [18].

The perceptron is a mathematical abstraction of this process. Instead of dendrites, it has a set of input lines $x_1, x_2, \ldots, x_m$. Instead of synaptic strengths, it has numerical weights $w_1, w_2, \ldots, w_m$. Instead of a threshold mechanism, it has the sign function. And instead of Hebbian plasticity, it has the error-correction rule [10, 11]. The analogy is not perfect, but it is close enough to be useful [23].

2.2 Structure of the Model

The signal-flow graph of the Rosenblatt perceptron is shown in Fig. 1. The model has three components: a set of input nodes, a linear combiner (the summing junction), and a hard limiter (the sign function) [11]. An additional input $x_0 = +1$ is fixed and is connected to the summing junction through a weight $w_0 = b$, which acts as the bias.

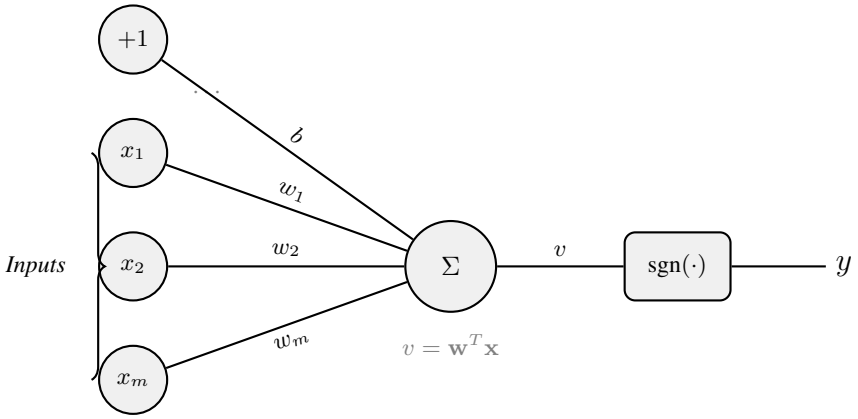


Figure 1: Signal-flow graph of the Rosenblatt perceptron. Each input x_i is multiplied by its corresponding weight w_i and summed to produce the induced local field v. The bias b is modelled as a weight w_0 connected to a fixed input $x_0 = +1$. The sign function then maps v to a binary output $y \in \{+1, -1\}$ [11].

The components of the model and their roles are summarised in Table 1.

Table 1: Components of the Rosenblatt perceptron and their roles.

Symbol	Name	Role
$x_1, \ldots, x_m$	Inputs	Feature values of the current pattern
$w_1, \ldots, w_m$	Synaptic weights	Determine the importance of each input; these are updated during training
b	Bias	Shifts the decision boundary; treated as weight w_0 with fixed input $x_0 = +1$
v	Induced local field	The weighted sum $v = \sum_i w_i x_i + b$
$\mathrm{sgn}(\cdot)$	Hard limiter	Converts v into a binary decision
y	Output	The perceptron's class prediction ($+1$ or -1)

3 Mathematical Formulation

3.1 The Induced Local Field

In a binary classification task there are two classes: $\mathcal{C}_1$ (class 1, with desired output $+1$) and $\mathcal{C}_2$ (class 2, with desired output -1). For a pattern with m input features, the perceptron first computes the weighted sum of all inputs together with the bias:

$$v = \sum_{i=1}^{m} w_i x_i + b = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m + b. \tag{1}$$

Here m is the number of input features, x_i is the i-th input, w_i is its weight, and b is the bias. This quantity v is called the *induced local field* [11]. It tells the perceptron how strongly the input pattern activates the neuron. A large positive value of v suggests strong evidence for class $\mathcal{C}_1$, while a large negative value suggests class $\mathcal{C}_2$.

To write Eq. (1) in a more compact vector form, we absorb the bias term into the weight vector by defining an augmented input vector and an augmented weight vector:

$$\mathbf{x}(n) = [+1,\ x_1(n),\ x_2(n),\ \ldots,\ x_m(n)]^T, \qquad \mathbf{w}(n) = [b,\ w_1(n),\ w_2(n),\ \ldots,\ w_m(n)]^T, \tag{2}$$

where n denotes the current iteration step and the superscript T denotes the transpose (turning a column vector into a row vector and vice versa). The induced local field then simplifies to the inner product

$$v(n) = \mathbf{w}^T(n)\,\mathbf{x}(n). \tag{3}$$

This notation is used throughout the rest of this paper. When we write $\mathbf{w}$ and $\mathbf{x}$, we always mean the augmented versions including the bias weight and the fixed $+1$ entry.

The surface $\mathbf{w}^T\mathbf{x} = 0$ is a hyperplane in the input space (a straight line when $m = 2$). This hyperplane is the *decision boundary* of the perceptron: patterns on one side are assigned to class $\mathcal{C}_1$, and patterns on the other side are assigned to class $\mathcal{C}_2$ [8, 11].

3.2 The Signum Activation Function

The value of v can be any real number. To convert it into a binary class label, the perceptron applies the *signum* (sign) function [11]:

$$\operatorname{sgn}(v) = \begin{cases} +1 & \text{if } v > 0, \\ 0 & \text{if } v = 0, \\ -1 & \text{if } v < 0. \end{cases} \tag{4}$$

In practice, when $v = 0$ we treat the output as -1 and apply the update rule [11]. The actual output of the perceptron at step n is therefore

$$y(n) = \operatorname{sgn}\big(\mathbf{w}^T(n)\,\mathbf{x}(n)\big)\,. \tag{5}$$

The signum function is shown in Fig. 2. It is the simplest possible activation function: a step from -1 to $+1$ at the origin. More sophisticated activation functions such as the sigmoid, hyperbolic tangent, and rectified linear unit (ReLU) are used in deeper networks [12, 24, 25], but the signum function captures the core idea of a threshold decision.

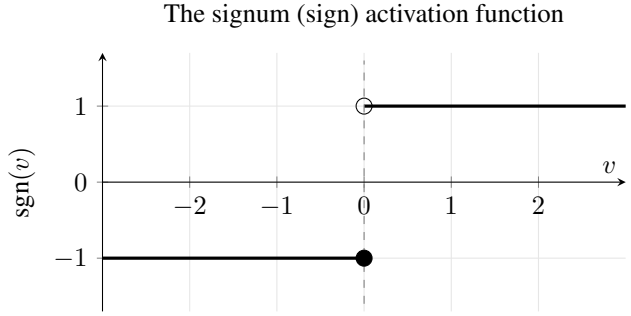


Figure 2: The signum function used as the activation (hard limiter) in the Rosenblatt perceptron. It outputs $+1$ for any positive induced local field and -1 for any negative value. At $v = 0$ the convention used in practice is $\operatorname{sgn}(0) = -1$ (filled circle), i.e. the pattern is classified into class $\mathcal{C}_2$ and the update rule is applied [11].

3.3 The Desired Output and the Error Signal

In supervised learning, each training pattern $\mathbf{x}(n)$ is accompanied by a *desired output* $d(n)$, which is the correct class label assigned [3, 9]. For the perceptron:

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \in \mathcal{C}_1, \\ -1 & \text{if } \mathbf{x}(n) \in \mathcal{C}_2. \end{cases} \tag{6}$$

The *error signal* at step n is the difference between the desired output and the actual output:

$$e(n) = d(n) - y(n). \tag{7}$$

As shown in Table 2, $e(n)$ can take only three possible values: 0 (correct prediction), $+2$ (predicted -1 but should be $+1$), and -2 (predicted $+1$ but should be -1).

Table 2: All possible combinations of desired output d and actual output y, and the resulting error signal and weight-update action.

Case	$d(n)$	$y(n)$	$e(n) = d - y$	Action
Correct	$+1$	$+1$	0	No weight update
Correct	-1	-1	0	No weight update
Wrong	$+1$	-1	$+2$	Weights pushed toward $\mathbf{x}(n)$
Wrong	-1	$+1$	-2	Weights pushed away from $\mathbf{x}(n)$

4 The Error-Correction Learning Algorithm

4.1 The Weight Update Rule

In ML terminology, *training* is the process of adjusting a model's parameters so that it performs correctly on labelled data [3, 9]. The perceptron training algorithm is the simplest example of this process, and it introduces the student to two concepts that appear in every ML system: a **loss signal** (the error $e(n)$) and a **parameter update rule** (Eq. 8 below). The central idea is straightforward: do nothing when the prediction is correct, and make a small correction when it is wrong. When the perceptron misclassifies pattern $\mathbf{x}(n)$, the weight vector is updated according to

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\, e(n)\, \mathbf{x}(n), \tag{8}$$

where $\eta > 0$ is the *learning-rate parameter* [10, 11]. This rule was proposed by Rosenblatt (1958) and is sometimes called the *fixed-increment adaptation rule* when η is held constant throughout training [11]. This update rule is a precursor to the broader family of gradient-based optimisation methods, including stochastic gradient descent, which underlies the training of modern deep learning models [12, 26].

The learning rate η controls the size of each correction. A value of $\eta = 1$ is standard and is the choice used in Haykin's analysis of the convergence theorem [11]. When the initial weight vector is zero, as in this tutorial, changing η merely rescales the final weight vector by a constant factor and does not change the resulting decision boundary; for non-zero initial weights or other learning algorithms, however, the choice of η can affect the training behaviour.

4.2 Why the Update Rule Makes Sense

It is worth understanding geometrically why Eq. (8) moves the perceptron towards a correct solution [11, 19]. Suppose the perceptron predicts $y = -1$ for a pattern $\mathbf{x}$ that should be $+1$. Then $e = +2$, and the update becomes

$$\mathbf{w} \leftarrow \mathbf{w} + 2\eta\, \mathbf{x}.$$

The new weight vector $\mathbf{w}(n+1)$ has a larger inner product with $\mathbf{x}$:

$$\mathbf{w}(n+1)^T \mathbf{x} = \mathbf{w}(n)^T \mathbf{x} + 2\eta\, \|\mathbf{x}\|^2.$$

Here $\|\mathbf{x}\|$ is the Euclidean norm (length) of the vector $\mathbf{x}$, defined by $\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x} = \sum_i x_i^2$ for a column vector $\mathbf{x}$. Since $2\eta\|\mathbf{x}\|^2 > 0$, the induced local field v, which is currently negative, increases and gradually moves towards becoming positive. As per Eq. (4), the perceptron is now more likely to output $+1$ for this same pattern in future. The opposite holds when the perceptron predicts $+1$ for a pattern that should be -1. In short, *each update makes the perceptron slightly more correct on the pattern it just misclassified* [11].

4.3 The Perceptron Convergence Theorem

The most important theoretical result concerning the perceptron is the following, first stated and proved by Rosenblatt (1958) and later sharpened by Novikoff (1962) [19]:

If the training patterns $\{\mathbf{x}(n)\}$ *originate from two linearly separable classes, there exists a weight vector* $\mathbf{w}^*$ *such that*

$$(\mathbf{w}^*)^T\mathbf{x} > 0 \quad \forall\, \mathbf{x} \in \mathcal{C}_1, \qquad (\mathbf{w}^*)^T\mathbf{x} \leq 0 \quad \forall\, \mathbf{x} \in \mathcal{C}_2, \tag{9}$$

and the perceptron algorithm with any initialisation converges to such a solution in a finite number of weight updates. Here $\forall$ means "for all", $\in$ means "belongs to", and $\mathbf{w}^*$ denotes the ideal (separating) weight vector.

The proof proceeds by showing two things simultaneously [11, 19]:

1. Each update increases the inner product (dot product) $\mathbf{w}(n) \cdot \mathbf{w}^*$ by at least a fixed positive amount $\alpha > 0$. This means the weight vector is making measurable progress toward $\mathbf{w}^*$ with every mistake.

2. The magnitude $\|\mathbf{w}(n)\|$ grows no faster than a bounded linear rate with the number of updates. This prevents the weights from growing without bound.

Together, these two facts imply, via the Cauchy–Schwarz inequality, that the cosine of the angle between $\mathbf{w}(n)$ and $\mathbf{w}^*$ approaches 1 after finitely many updates [11]. Once the angle is small enough, the perceptron classifies everything correctly and the algorithm stops.

4.4 Step-by-step Algorithm

The complete procedure is summarised in Table 3.

Table 3: Step-by-step algorithm for the Rosenblatt perceptron.

Step	Description
1	Collect labelled training patterns $\{(\mathbf{x}(n),\ d(n))\}_{n=1}^{N}$ and augment each input with $x_0 = +1$.
2	Initialise the weight vector $\mathbf{w}(0) = \mathbf{0}$ and choose learning rate $\eta > 0$.
3	For each training pattern in sequence, compute the induced local field $v(n) = \mathbf{w}^T(n)\,\mathbf{x}(n)$ and the actual output $y(n) = \text{sgn}(v(n))$.
4	Compute the error $e(n) = d(n) - y(n)$.
5	If $e(n) \neq 0$, update the weight vector: $\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\, e(n)\,\mathbf{x}(n)$. Otherwise, leave $\mathbf{w}$ unchanged.
6	Repeat Steps 3–5 over all training patterns until a complete pass is made with no weight update (convergence).

5 Illustrative Example: Wireless Power Transfer with a Tesla Coil

5.1 Physical Background

A Tesla coil operates by driving its primary coil with a high-frequency alternating voltage, which induces a strongly oscillating magnetic field in the surrounding space [27]. When a secondary coil is brought into the vicinity of the primary, electromagnetic induction transfers energy across the air gap without any physical contact, the principle of *wireless power transfer* [28]. If the transferred power exceeds the forward-voltage threshold of a small LED connected across the secondary coil, the LED glows. Otherwise it remains dark.

Two physical factors govern whether the coupling is strong enough to light the LED:

- **Transmitter voltage** x_1 (in normalised units, 1–10): a higher driving voltage increases the oscillating magnetic flux Φ (the magnetic field passing through the receiver coil), boosting the induced electromotive force (EMF), $\mathcal{E}$, in the receiver coil by Faraday's law, $\mathcal{E} = -d\Phi/dt$.

- **Coil-to-LED distance** x_2 (in normalised units, 1–10): the magnetic field of a small coil falls off approximately as r^{-3}, where r is the separation between the coils, for near-field coupling [28], so increasing the separation dramatically weakens the induced EMF.

The classification task is therefore:

$$\mathcal{C}_1 : \text{LED glows} \quad (d = +1),$$

$$\mathcal{C}_2 : \text{LED does not glow} \quad (d = -1).$$

High voltage *and* short distance cooperate to produce sufficient coupling; low voltage *or* large distance breaks the coupling below the LED's threshold.

5.2 Perceptron Inputs, Weights, and Activation

With only two physical features the perceptron model is

$$v = w_1 x_1 + w_2 x_2 + b, \tag{10}$$

where x_1 is the normalised voltage and x_2 is the normalised distance. The augmented input vector and weight vector are

$$\mathbf{x} = [1,\ x_1,\ x_2]^T, \qquad \mathbf{w} = [b,\ w_1,\ w_2]^T. \tag{11}$$

The physical interpretation of the two learned weights is immediate:

- Increasing the voltage x_1 promotes coupling and increases v, favouring the LED-glows class. So w_1 must be positive ($w_1 > 0$).
- Increasing the distance x_2 suppresses coupling and decreases v, favouring the LED-dark class. So w_2 must be negative ($w_2 < 0$).

The bias b captures the baseline coupling level: a sufficiently large negative bias encodes the fact that the LED requires a minimum power threshold even under the most favourable conditions.

The activation rule is unchanged from the general formulation:

$$y = \operatorname{sgn}(v) = \begin{cases} +1 & \text{(LED glows)}, \quad v > 0, \\ -1 & \text{(LED dark)}, \quad v \leq 0. \end{cases} \tag{12}$$

5.3 Decision Boundary Intuition

Setting $v = 0$ defines the critical coupling curve in the (x_1, x_2) plane:

$$w_1 x_1 + w_2 x_2 + b = 0 \quad \Longrightarrow \quad x_2 = -\frac{w_1}{w_2}\, x_1 - \frac{b}{w_2}. \tag{13}$$

Since $w_1 > 0$ and $w_2 < 0$, the slope $-w_1/w_2$ is positive: the boundary is a rising straight line in the voltage–distance plane. Experiments lying *above* this line (large distance relative to the voltage) correspond to insufficient coupling, so the LED stays dark. Experiments *below* the line (short distance relative to the voltage) correspond to sufficient coupling, and the LED glows. The perceptron therefore learns, from data alone, the threshold relationship between transmitter voltage and coil separation that determines whether wireless power delivery succeeds.

5.4 Problem Setup and Dataset

The training dataset consists of four experimental observations, summarised in Table 4. Voltage x_1 and distance x_2 are both scaled to the range 1–10 for numerical convenience.

Table 4: Training dataset for the Tesla coil LED classification task.

Sample	Voltage x_1	Distance x_2	Desired output d	Augmented vector $\mathbf{x}$
A	8	2	$+1$	$[1,\ 8,\ 2]^T$
B	3	8	-1	$[1,\ 3,\ 8]^T$
C	7	3	$+1$	$[1,\ 7,\ 3]^T$
D	2	7	-1	$[1,\ 2,\ 7]^T$

Samples A and C represent high-voltage, short-range configurations where the induced EMF easily exceeds the LED threshold ($d = +1$). Samples B and D represent low-voltage, long-range configurations where the coupling is too weak to light the LED ($d = -1$).

Each input vector is augmented with a leading 1 so that the perceptron can learn a bias term. The initial weight vector is chosen as

$$\mathbf{w}(0) = [0,\ 0,\ 0]^T,$$

and the learning rate is $\eta = 1$.

5.5 Epoch 1

Iteration 1 — Sample A

$$\mathbf{x}_A = [1,\ 8,\ 2]^T, \qquad d = +1.$$

Initial weights:

$$\mathbf{w} = [0,\ 0,\ 0]^T.$$

The local field is

$$\mathbf{w} = [0,\ 0,\ 0] + \eta\, e\, [1,\ 8,\ 2] = [0,\ 0,\ 0] + (1)(2)\, [1,\ 8,\ 2] = [2,\ 16,\ 4], \tag{14}$$

where $\eta = 1$ and $e = 2$.

Since $v = 0$, the signum convention gives $y = -1$. Hence,

$$e = d - y = 1 - (-1) = 2.$$

Physical meaning: the perceptron has not yet learnt that high voltage and close proximity should produce sufficient coupling; the weights are updated to encode this information:

$$\mathbf{w} = [0,\ 0,\ 0] + 1 \cdot 2 \cdot [1,\ 8,\ 2] = [2,\ 16,\ 4]. \tag{15}$$

Iteration 2 — Sample B

$$\mathbf{x}_B = [1,\ 3,\ 8]^T, \qquad d = -1.$$

$$v = [2,\ 16,\ 4] \cdot [1,\ 3,\ 8]^T = 2 + 48 + 32 = 82. \tag{16}$$

Therefore $y = +1$, but $d = -1$, so

$$e = -1 - 1 = -2.$$

Physical meaning: the perceptron wrongly predicts that a low-voltage, long-range setup can light the LED. The update pushes the weights away from this configuration:

$$\mathbf{w} = [2,\ 16,\ 4] + \eta\, e\, [1,\ 3,\ 8] = [2,\ 16,\ 4] + (1)(-2)\,[1,\ 3,\ 8] = [0,\ 10,\ -12], \tag{17}$$

where $\eta = 1$ and $e = -2$.

Notice that after this correction w_2 has become negative, correctly encoding the physical insight that large distance suppresses coupling.

Iteration 3 — Sample C

$$\mathbf{x}_C = [1,\ 7,\ 3]^T, \qquad d = +1.$$

$$v = [0,\ 10,\ -12] \cdot [1,\ 7,\ 3]^T = 0 + 70 - 36 = 34. \tag{18}$$

Thus $y = +1 = d$. The classification is correct, so no update occurs.

Physical meaning: the perceptron now correctly identifies that high voltage at short range ($x_1 = 7, x_2 = 3$) produces sufficient coupling to glow the LED.

Iteration 4 — Sample D

$$\mathbf{x}_D = [1,\ 2,\ 7]^T, \qquad d = -1.$$

$$v = [0,\ 10,\ -12] \cdot [1,\ 2,\ 7]^T = 0 + 20 - 84 = -64. \tag{19}$$

Hence $y = -1 = d$. The classification is correct; no update is needed.

Physical meaning: the perceptron correctly identifies that low voltage at large distance ($x_1 = 2, x_2 = 7$) is insufficient to light the LED.

5.6 Epoch 2

At the start of Epoch 2, the weight vector is

$$\mathbf{w} = [0,\ 10,\ -12]^T.$$

Table 5: Classification results during Epoch 2.

Sample	$v = \mathbf{w}^T\mathbf{x}$	y	d	Correct?	Update?
A	$0 + 80 - 24 = 56$	$+1$	$+1$	Yes	No
B	$0 + 30 - 96 = -66$	-1	-1	Yes	No
C	$0 + 70 - 36 = 34$	$+1$	$+1$	Yes	No
D	$0 + 20 - 84 = -64$	-1	-1	Yes	No

All four samples are classified correctly in Epoch 2. Since no weight update occurs during a complete pass, the algorithm has *converged*. The final weight vector is

$$\mathbf{w}^* = [0,\ 10,\ -12]^T,$$

corresponding to $b = 0$, $w_1 = 10$, and $w_2 = -12$. The decision boundary in the voltage–distance plane is therefore

$$10\,x_1 - 12\,x_2 = 0 \quad \Longrightarrow \quad x_2 = \tfrac{5}{6}\,x_1 \approx 0.83\,x_1. \tag{20}$$

5.7 Physical Interpretation of the Learned Weights

The converged weight vector $\mathbf{w}^* = [0,\ 10,\ -12]^T$ has a clear electromagnetic interpretation. The positive weight $w_1 = 10$ quantifies the benefit of increasing the driving voltage: a unit increase in x_1 contributes $+10$ to the coupling measure v. The negative weight $w_2 = -12$ quantifies the penalty imposed by separation: a unit increase in x_2 reduces v by 12. The magnitude $|w_2| > w_1$ indicates that, for this normalised dataset, a unit change in distance affects v more strongly than a unit change in voltage. This is qualitatively in line with the rapid fall-off of near-field magnetic coupling with separation ($\sim r^{-3}$) [28]. However, the weights depend on the chosen normalisation and the small dataset, and the dependence of coupling on driving voltage is not modelled here, so no quantitative comparison is implied.

The decision boundary $x_2 \approx 0.83\,x_1$ states that the LED glows whenever the normalised distance is less than 83% of the normalised voltage. In other words, the perceptron has automatically recovered, purely from labelled observations, a linear approximation to the threshold coupling condition of the Tesla coil system.

After convergence, the decision boundary takes the form

$$b + w_1 x_1 + w_2 x_2 = 0, \tag{21}$$

which is a straight line separating the LED-glows and LED-dark regions in the feature space [8, 11], as illustrated in Fig. 3.

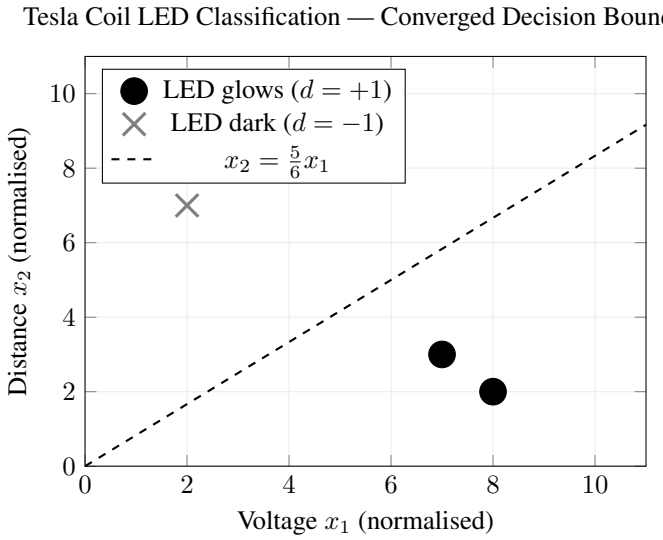


Figure 3: Converged decision boundary for the Tesla coil LED classification task. The dashed line $x_2 = \frac{5}{6}x_1$ correctly separates the LED-glows class (filled circles, high voltage / short range) from the LED-dark class (crosses, low voltage / long range). Points below the boundary receive sufficient wireless power to light the LED; points above do not.

6 Python Implementation

A clean implementation of the perceptron is straightforward. The core algorithm requires only three operations: computing the induced local field, applying the sign function, and applying the weight update when there is an error [29]. The `NumPy` library [30] is used for efficient array operations. Listing 1 first defines the training data (the feature matrix `X` and the desired outputs `d`), and Listing 2 then contains the training loop, which must be run after the data have been defined.

Listing 1: Defining the Tesla coil dataset (voltage and distance) and the desired outputs.

```
import numpy as np

# x1 = normalised voltage (1-10)
# x2 = normalised coil-to-LED distance (1-10)
X = np.array([
    [8, 2],    # Sample A - LED glows  (+1)
    [3, 8],    # Sample B - LED dark   (-1)
    [7, 3],    # Sample C - LED glows  (+1)
    [2, 7],    # Sample D - LED dark   (-1)
], dtype=float)

# desired outputs d(n): +1 = LED glows, -1 = LED dark
d = np.array([+1, -1, +1, -1])
```

Listing 2: Core perceptron training loop in Python.

```
def sgn(v):
    return +1 if v > 0 else -1

ones = np.ones((X.shape[0], 1))
X_aug = np.hstack([ones, X])       # augment inputs with x0 = +1

w   = np.zeros(X_aug.shape[1])     # w(0) = [0, 0, 0]
eta = 1.0                          # learning rate

for epoch in range(1, 101):
    errors = 0
    for i in range(len(X_aug)):
        v     = np.dot(w, X_aug[i])       # induced local field
        y     = sgn(v)                    # actual output
        error = d[i] - y                  # error signal
        if error != 0:
            w = w + eta * error * X_aug[i]   # weight update
            errors += 1
    if errors == 0:                       # converged
        break

print("Converged weights [b, w1, w2]:", w)
```

The decision boundary can be visualized after convergence by plotting the line $b + w_1 x_1 + w_2 x_2 = 0$. For the above dataset the converged boundary is $x_2 = \frac{5}{6} x_1 \approx 0.83\, x_1$, which cleanly separates the two classes in the voltage–distance plane. Visualisation is facilitated by the `Matplotlib` library [31].

6.1 Python Implementation Output

```
                                                PERCEPTRON LEARNING TABLE
=================================================================================================================
Epoch     Sample    v = W^Tx          y         d         Error     Updated?        New Weight Vector
-----------------------------------------------------------------------------------------------------------------
1         A         0.00              -1        1         2         Yes             [ 2. 16.  4.]
1         B         82.00             1         -1        -2        Yes             [  0.  10. -12.]
1         C         34.00             1         1         0         No              [  0.  10. -12.]
1         D         -64.00            -1        -1        0         No              [  0.  10. -12.]
2         A         56.00             1         1         0         No              [  0.  10. -12.]
2         B         -66.00            -1        -1        0         No              [  0.  10. -12.]
2         C         34.00             1         1         0         No              [  0.  10. -12.]
2         D         -64.00            -1        -1        0         No              [  0.  10. -12.]
-----------------------------------------------------------------------------------------------------------------

FINAL CONVERGED WEIGHT VECTOR
=================================================================================================================
W = [  0.  10. -12.]

DECISION BOUNDARY
=================================================================================================================
0.00 + (10.00)x₁ + (-12.00)x₂ = 0
x₂ = (0.83)x₁ + (0.00)


=================================================================================================================
                                                PREDICTION
=================================================================================================================
Input Data:
Voltage (x1) = 6
Distance (x2)     = 4

Computed value v = 12.00
Predicted Class : PASS (+1)

Training complete.
>>
```

Figure 4: Perceptron learning table showing the epoch-wise training process and weight updates for the Tesla coil wireless power classification task.

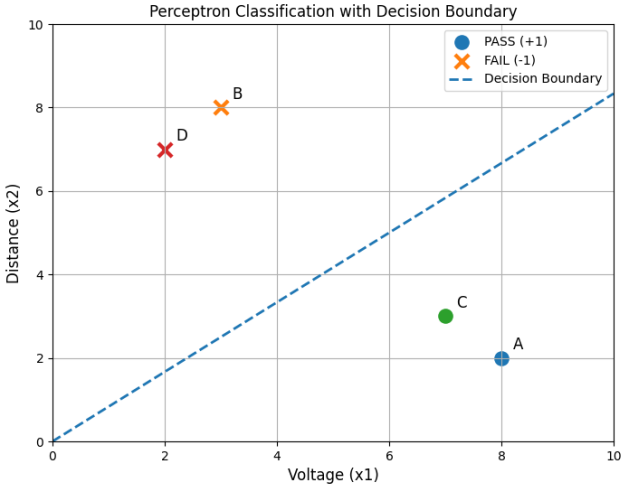


Figure 5: Perceptron classification with the final decision boundary $x_2 = \frac{5}{6}x_1$ separating the LED-glows class from the LED-dark class.

The final converged weight vector obtained after training is $\mathbf{w}^* = [0,\ 10,\ -12]^T$, corresponding to $b = 0$, $w_1 = 10$, and $w_2 = -12$, which gives the decision boundary $10x_1 - 12x_2 = 0$, or equivalently $x_2 = \frac{5}{6}x_1 \approx 0.83\,x_1$.

Using the trained perceptron, one can predict whether an LED placed at normalised distance $x_2 = 4$ from a coil driven at normalised voltage $x_1 = 6$ would glow. The induced local field is $v = 10(6) - 12(4) = 60 - 48 = 12 > 0$, so the perceptron predicts $y = +1$: the LED *glows*. This is physically consistent with the fact that moderately high voltage at close-to-medium range provides sufficient inductive coupling.

7 Geometric Interpretation

It is instructive to think about what the perceptron is doing in geometric terms, because this makes the convergence theorem feel natural rather than abstract [8, 11].

7.1 The Weight Vector is Perpendicular to the Boundary

The decision boundary is the set of all $\mathbf{x}$ satisfying $\mathbf{w}^T\mathbf{x} = 0$. This means every point on the boundary is orthogonal to $\mathbf{w}$. In other words, $\mathbf{w}$ is the *normal vector* to the decision hyperplane [11].

This has a useful implication: when we update the weight vector by adding $2\eta\,\mathbf{x}(n)$ (for a false negative), we are rotating the normal to the boundary toward the misclassified point. This rotates the boundary itself away from that point, making the point more likely to be on the correct side in the next pass [11, 19]. In the Tesla coil context, a false negative (predicting that the LED stays dark when it actually glows) causes the boundary to swing toward the high-voltage / short-distance region, correctly expanding the LED-glows classification zone.

7.2 Linear Separability in Picture Form

A dataset is linearly separable if and only if there exists at least one hyperplane with all class $\mathcal{C}_1$ points on one side and all class $\mathcal{C}_2$ points on the other [11, 32]. In two dimensions, the hyperplane is a straight line. Fig. 6 compares a linearly separable dataset with a non-separable one.

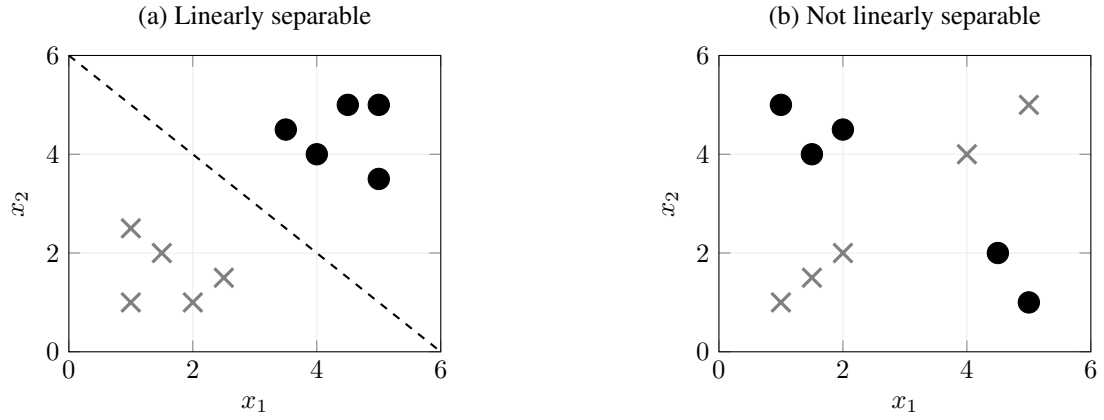


Figure 6: (a) A linearly separable dataset: a single straight line can separate the two classes. (b) A non-linearly separable dataset: no single straight line can separate the classes. The Rosenblatt perceptron is only guaranteed to converge for case (a) [11].

8 Limitations of the Perceptron

8.1 The XOR Problem

The most famous limitation of the single-layer perceptron was identified by Minsky and Papert (1969) in their monograph *Perceptrons* [33]. They showed that a single perceptron cannot learn the exclusive-or (XOR) function, whose truth table is given in Table 6.

Table 6: XOR truth table. No single straight line can separate the class $+1$ entries from the class -1 entries.

x_1	x_2	XOR (d)
0	0	-1
0	1	$+1$
1	0	$+1$
1	1	-1

The reason is visible in Fig. 7: the two XOR classes are arranged in a chequered pattern in the input plane, and no

straight line can separate them. This means the data are not linearly separable, and the convergence theorem does not apply [19, 33]. The perceptron training loop will cycle indefinitely without ever reaching zero errors.

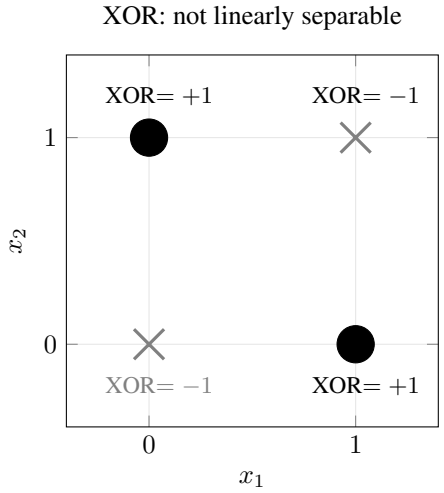


Figure 7: XOR plotted in the input plane. The class $+1$ points (filled circles) and the class -1 points (crosses) cannot be separated by any single straight line, making the XOR problem unsolvable by a single perceptron [33].

8.2 Impact and the Road Forward

The critique by Minsky and Papert led to a sharp decline in neural network research throughout the 1970s, a period sometimes referred to as the first AI winter [14]. The resolution came in the mid-1980s with the development of the backpropagation algorithm for training multi-layer perceptrons (MLPs) [34]. By stacking several layers of perceptron-like units and using a differentiable activation function, an MLP can approximate any continuous nonlinear boundary [35]. This made it possible to solve problems like XOR and far more complex ones. The modern deep learning revolution [12, 21] is a direct continuation of this line of development.

A few other practical limitations are also worth noting. The convergence theorem guarantees termination only for linearly separable data; for non-separable data the algorithm loops indefinitely and a maximum epoch limit must be imposed [11]. Additionally, large feature values can dominate the weight update and cause numerical instability; it is therefore standard practice to normalise all input features to the same range before training [9, 13].

The perceptron corresponds to what is now called a **linear classifier with a hard threshold**. The generalisation from one layer to many and from a hard threshold to smooth, differentiable activations [24, 25] is precisely what gives deep neural networks their power [12, 21]. Every concept introduced here: input features, weight vectors, bias terms, forward passes, error signals, and iterative updates, carries forward unchanged into those more advanced models [11, 16]. One who understands the perceptron already understands the skeleton of a deep learning system.

9 Conclusion

In this work, we presented a clear and step-by-step explanation of the Rosenblatt perceptron as a hands-on introduction to AI/ML classification, motivated by the physics of Tesla coil wireless power transfer. Starting from the biological inspiration [17, 18, 23] and covering the signal-flow model, the signum activation function, the error-correction learning rule, and the convergence theorem [10, 11, 19], the exposition was grounded throughout in a concrete electromagnetic scenario: can a given combination of transmitter voltage and coil-to-LED distance produce enough inductive coupling to light an LED? Among the many algorithms available in modern ML, including support vector machines [32], decision trees, and deep neural networks [12, 21], the perceptron stands out as the most transparent entry point, because every stage of its operation can be followed by hand and verified analytically.

The effectiveness of the algorithm was demonstrated using the four-sample Tesla coil dataset. The full iteration history shows how the weight vector is updated only when the perceptron makes a mistake, and how training terminates naturally once all patterns are correctly classified [11]. The converged weight vector $\mathbf{w}^* = [0, 10, -12]^T$ acquires a direct physical interpretation: the positive weight on voltage ($w_1 = 10$) encodes the benefit of stronger driving, while the larger-magnitude negative weight on distance ($w_2 = -12$) qualitatively reflects the rapid decrease of near-field magnetic coupling with separation [28]. The learned decision boundary $x_2 \approx 0.83\, x_1$ is the linear threshold surface that the perceptron has autonomously inferred from the labelled observations.

The limitations of the model were also discussed. The most important is that the perceptron can only separate linearly separable classes, and has no mechanism for learning nonlinear boundaries [33]. This limitation motivates the study of multi-layer perceptrons and, more broadly, deep neural networks [12, 34, 35], which are built from generalisations of the same basic unit.

Overall, the Rosenblatt perceptron is a simple, interpretable, and historically significant ML algorithm [14]. Understanding it thoroughly, as illustrated here through a vivid physics experiment, provides the conceptual foundation necessary to follow the subsequent development of neural network theory and to write real AI/ML code from scratch [29, 30, 31], consistent with the broader tradition of learning dynamical models from data [16, 15].

Acknowledgements

All computational work reported in this paper was carried out using the Brahmagupta High-Performance Computing (HPC) facility at Sikkim University. One of the authors, Rupak Mukherjee gratefully acknowledges the support provided by the Inter-University Centre for Astronomy and Astrophysics (IUCAA), Pune, through its Visiting Associate Programme.

AI Disclosure Statement

In preparing this work, the authors used ChatGPT to assist in overall English language of the manuscript. All scientific observations, computational work, analyses, and results were independently carried out and verified by the authors. The authors subsequently reviewed and revised the content as necessary and accept full responsibility for the final published work.

References

[1] C. M. Bishop, Pattern Recognition and Machine Learning, (Springer, New York, 2006).

[2] A. M. Turing, "Computing machinery and intelligence", Mind 59(236), 433–460 (1950).

[3] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects", Science 349(6245), 255–260 (2015).

[4] G. Litjens et al., "A survey on deep learning in medical image analysis", Medical Image Analysis 42, 60–88 (2017).

[5] A. Vaswani et al., "Attention is all you need", Advances in Neural Information Processing Systems 30, 5998–6008 (2017).

[6] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems", Computer 42(8), 30–37 (2009).

[7] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving", Proc. IEEE International Conference on Computer Vision (ICCV), 2015, pp. 2722–2730.

[8] T. M. Mitchell, Machine Learning, (McGraw-Hill, New York, 1997).

[9] E. Alpaydin, Introduction to Machine Learning, 4th ed. (MIT Press, Cambridge, MA, 2020).

[10] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", Psychological Review 65(6), 386–408 (1958).

[11] S. Haykin, Neural Networks and Learning Machines, 3rd ed. (Pearson Education, New Jersey, 2009).

[12] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, (MIT Press, Cambridge, MA, 2016).

[13] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, Learning from Data: A Short Course, (AMLBook), 2012.

[14] D. Crevier, AI: The Tumultuous History of the Search for Artificial Intelligence, (Basic Books, New York, 1993).

[15] S. H. Strogatz, Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering, (Westview Press, Boulder, 1994).

[16] S. L. Brunton and J. N. Kutz, Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control, (Cambridge University Press, Cambridge, 2019).

[17] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biophysics 5, 115–133 (1943).

[18] D. O. Hebb, The Organization of Behavior: A Neuropsychological Theory, (Wiley, New York, 1949).

[19] A. B. J. Novikoff, "On convergence proofs for perceptrons", Proceedings of the Symposium on the Mathematical Theory of Automata, vol. 12, Brooklyn Polytechnic Institute, 1962, pp. 615–622.

[20] B. Widrow and M. E. Hoff, "Adaptive switching circuits", IRE WESCON Convention Record, 1960, pp. 96–104.

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", Nature 521(7553), 436–444 (2015).

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory", Neural Computation 9(8), 1735–1780 (1997).

[23] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, Principles of Neural Science, 5th ed. (McGraw-Hill, New York, 2013).

[24] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines", Proceedings of the 27th International Conference on Machine Learning (ICML), 2010, pp. 807–814.

[25] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS), 2011, pp. 315–323.

[26] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning", SIAM Review 60(2), 223–311 (2018).

[27] N. Tesla, "System of transmission of electrical energy", U.S. Patent 645, 576, 1900.

[28] A. Kurs *et al.*, "Wireless power transfer via strongly coupled magnetic resonances", Science 317(5834), 83–86 (2007).

[29] F. Pedregosa et al., "Scikit-learn: Machine learning in Python", Journal of Machine Learning Research 12, 2825–2830 (2011).

[30] C. R. Harris et al., "Array programming with NumPy", Nature 585(7825), 357–362 (2020).

[31] J. D. Hunter, "Matplotlib: A 2D graphics environment", Computing in Science & Engineering 9(3), 90–95 (2007).

[32] V. N. Vapnik, The Nature of Statistical Learning Theory, (Springer, New York, 1995).

[33] M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry, (MIT Press, Cambridge, MA, 1969).

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", Nature 323(6088), 533–536 (1986).

[35] G. Cybenko, “Approximation by superpositions of a sigmoidal function”, Mathematics of Control, Signals and Systems 2(4), 303–314 (1989).