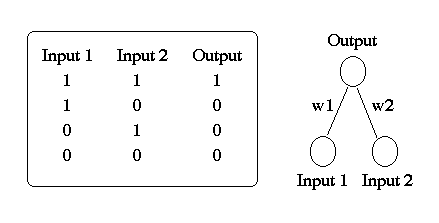
**NNET and Perceptron Learning**

Apply the perceptron learning rule to solve the following problem for w1 = -0.5, w2 = 0.5, and θ = 1.5. Record the weights and threshold after each step of learning, applying the input patterns in the same order as in the Figure below. Make a table of the results. After one step of learning your table should look like:

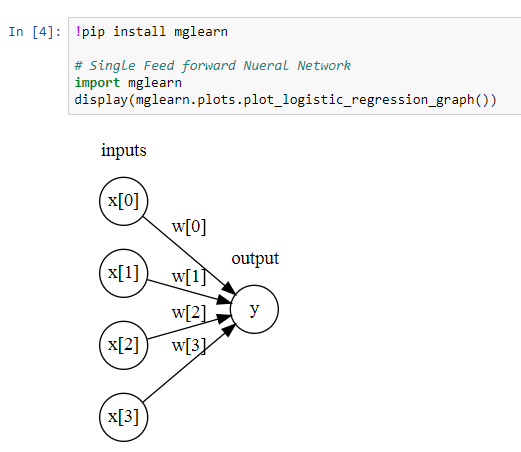
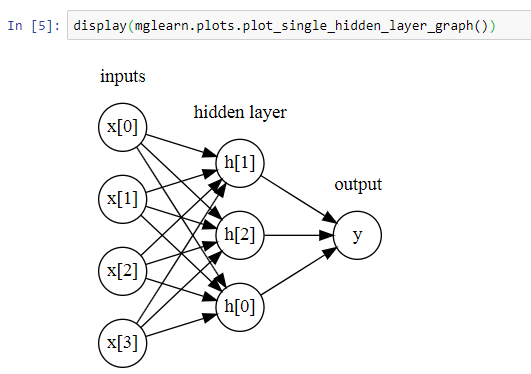
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input1** | **Input2** | **Target** | **Output** | **Weight1** | **Weight2** | **Threshold** |
| 1 | 1 | 1 | 0 | -0.5 | 0.5 | 1.5 |
| 1 | 0 | 0 | 0 | 0.5 | 1.5 | 0.5 |

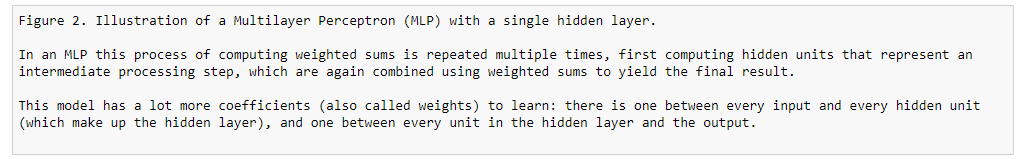
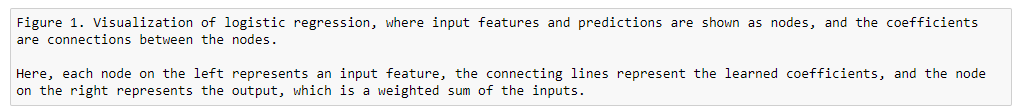




A Perceptron is a single layer feed-forward Neural Network used to determine whether an [input](https://brilliant.org/wiki/input-output/) belongs to one [class](https://brilliant.org/wiki/classification/) or another. For example, the perceptron algorithm can determine the [AND operator](https://brilliant.org/wiki/common-operators/?wiki_title=AND%20operator) - given [binary](https://brilliant.org/wiki/binary-numbers/) inputs  and , is ( AND ) equal to 0 or 1?

Multi Layered Perceptrons (MLP) can be viewed as generalizations of linear models that perform multiple stages of processing to come to a decision.



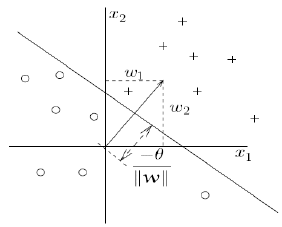
**Linearly Separable –**

The Perceptron algorithm focuses on binary classified data, objects that are either members of one class or another. Additionally, it allows for [online learning](https://brilliant.org/wiki/online-algorithm/?wiki_title=online%20learning), which simply means that it processes elements in the training dataset one at a time (which can be useful for large datasets). Furthermore, the perceptron algorithm is a type of [linear classifier](https://brilliant.org/wiki/linear-classifier/), which classifies data points by using a linear combination of the variables used.

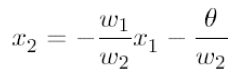
 The algorithm takes binary classified input data, along with their [class membership](https://brilliant.org/wiki/classification/), and outputs a line that attempts to separate data of one class from data of the other: data points on one side of the line are of one class and data points on the other side are of the other.



*ŷ* is a weighted sum of the input features *x*[0] to *x*[*p*], weighted by the learned coefficients *w*[0] to *w*[*p*].



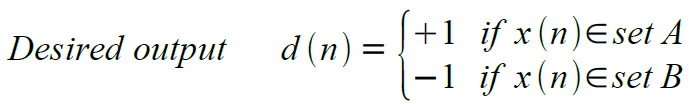




* Weights determine the Slope of the line.
* The Bias is proportional to the offset of the plane from the Origin.
* The Weight vector is perpendicular to the Plane.

**Perceptron Learning Algorithm**

The classification of inputs correctly is accomplished by adjusting the connected weights and bias. During Training both Weights and Bias are modified accordingly.



If Classification is correct, do nothing. If incorrect, modify the Weights using 

Repeat this procedure until the entire Training set is classified correctly.

**Final Result of Perceptron Learning and Iterations**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ITERATION | INPUT 1 | INPUT 2 | TARGET | OUTPUT (NET>THETA THEN 1 ELSE 0) | WEIGHT 1 | WEIGHT 2 | THRESHOLD (THETA) | NET (W1X1+W2X2) | DELTA THETA (-d) | DETLA WEIGHT (d) | X1 AND X2 |
| 1 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 | 1 |
| 2 | 1 | 0 | 0 | **0** | 0.5 | 1.5 | 0.5 | 0.5 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 | 0 |
| 4 | 0 | 0 | 0 | **0** | -0.5 | 0.5 | 1.5 | 0 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 | 1 |
| 6 | 1 | 1 | 1 | **1** | 0.5 | 1.5 | 0.5 | 2 | 0 | 0 | 1 |
| 7 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 | 0 |
| 8 | 0 | 1 | 0 | **0** | -0.5 | 0.5 | 1.5 | 0.5 | 0 | 0 | 0 |

Input Pairs {(1,0), (0,0), (1,1), (0,1)}

Incorrect Classification – Need for Weights and Theta Adjustment.

Correct Classification – Do Nothing.

|  |
| --- |
| **Iteration 1** |
| The Network computation is defined as Net = W1X1 + W2X2. If Net > Theta Then Output = 1 otherwise Output = 0 |
| Perceptron Learning Rules – |



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ITERATION | INPUT 1 | INPUT 2 | TARGET | OUTPUT (NET>THETA THEN 1 ELSE 0) | WEIGHT 1 | WEIGHT 2 | THRESHOLD (THETA) | NET (W1X1+W2X2) | DELTA THETA (-d) | DETLA WEIGHT (d) |
| 1 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |

Since Net is not greater than Theta (0 < 1.5), the Output is 0. There is a misclassification, so the parameters need to be updated.

As we have misclassified Class 1 as Class 0, the Theta and Weights need to be updated. Delta Theta for **Threshold = -(1-0) = -1.**

The feedback signal for updating Weights is just the opposite. In this case, **d = 1 for Weights.**

Now, we have new set of Weights and Threshold –

THETA = (1.5 - 1) = 0.5

W1 = (-0.5 + 1) = 0.5

W2 = (0.5 + 1) = 1.5

Let's iterate with the new set of Weights and Threshold

|  |
| --- |
| **Iteration 2** |
| The Network computation is defined as Net = W1X1 + W2X2. If Net > Theta Then Output = 1 otherwise Output = 0 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ITERATION | INPUT 1 | INPUT 2 | TARGET | OUTPUT (NET>THETA THEN 1 ELSE 0) | WEIGHT 1 | WEIGHT 2 | THRESHOLD (THETA) | NET (W1X1+W2X2) | DELTA THETA (-d) | DETLA WEIGHT (d) |
| 1 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 2 | 1 | 0 | 0 | **0** | 0.5 | 1.5 | 0.5 | 0.5 | 0 | 0 |

Since Net is not greater than Theta (0.5 = 0.5), the Output is 0. It had properly classified and there is no need to update the Weights as there is no misclassification. We will continue with the same Weights.

**Iteration 3**

The Network computation is defined as Net = W1X1 + W2X2.If Net > Theta Then Output = 1 otherwise Output = 0

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ITERATION | INPUT 1 | INPUT 2 | TARGET | OUTPUT (NET>THETA THEN 1 ELSE 0) | WEIGHT 1 | WEIGHT 2 | THRESHOLD (THETA) | NET (W1X1+W2X2) | DELTA THETA (-d) | DETLA WEIGHT (d) |
| 1 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 2 | 1 | 0 | 0 | **0** | 0.5 | 1.5 | 0.5 | 0.5 | 0 | 0 |
| 3 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 |

Since Net is greater than Theta (1.5 > 0.5), the Output is 1. There is a misclassification, so the Weights need to be updated.

As we have misclassified Class 0 as Class 1, the Theta and Weights need to be updated. Delta Theta for **Threshold = -(0-1) = 1.** The feedback signal for updating Weights is just the opposite. In this case, **d = -1 for Weights.**

Now, we have new set of Weights and Threshold –

THETA = (0.5 + 1) = 1.5

W1 = (0.5 - 1) = -0.5

W2 = (1.5 - 1) = 0.5

Let's iterate with the new set of Weights and Threshold.

**Iteration 4**

The Network computation is defined as Net = W1X1 + W2X2.If Net > Theta Then Output = 1 otherwise Output = 0

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ITERATION | INPUT 1 | INPUT 2 | TARGET | OUTPUT (NET>THETA THEN 1 ELSE 0) | WEIGHT 1 | WEIGHT 2 | THRESHOLD (THETA) | NET (W1X1+W2X2) | DELTA THETA (-d) | DETLA WEIGHT (d) |
| 1 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 2 | 1 | 0 | 0 | **0** | 0.5 | 1.5 | 0.5 | 0.5 | 0 | 0 |
| 3 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 |
| 4 | 0 | 0 | 0 | **0** | -0.5 | 0.5 | 1.5 | 0 | 0 | 0 |

Since Net is not greater than Theta (0 < 1.5), the Output is 0. It had properly classified and there is no need to update the Weights as there is no misclassification. We will continue with the same Weights for the previous inputs.

**Iteration 5**

The Network computation is defined as Net = W1X1 + W2X2.If Net > Theta Then Output = 1 otherwise Output = 0

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ITERATION | INPUT 1 | INPUT 2 | TARGET | OUTPUT (NET>THETA THEN 1 ELSE 0) | WEIGHT 1 | WEIGHT 2 | THRESHOLD (THETA) | NET (W1X1+W2X2) | DELTA THETA (-d) | DETLA WEIGHT (d) |
| 1 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 2 | 1 | 0 | 0 | **0** | 0.5 | 1.5 | 0.5 | 0.5 | 0 | 0 |
| 3 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 |
| 4 | 0 | 0 | 0 | **0** | -0.5 | 0.5 | 1.5 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |

Since Net is not greater than Theta (0 < 1.5), the Output is 0. There is a misclassification, so the Weights need to be updated.

As we have misclassified Class 1 as Class 0, the Theta and Weights need to be updated. Delta Theta for **Threshold = -(1-0) = -1.**

The feedback signal for updating Weights is just the opposite. In this case, **d = 1 for Weights.**

Now, we have new set of Weights and Threshold –

THETA = (1.5 - 1) = 0.5

W1 = (-0.5 + 1) = 0.5

W2 = (0.5 + 1) = 1.5

Let's iterate with the new set of Weights and Threshold

**Iteration 6**

The Network computation is defined as Net = W1X1 + W2X2.If Net > Theta Then Output = 1 otherwise Output = 0.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ITERATION | INPUT 1 | INPUT 2 | TARGET | OUTPUT (NET>THETA THEN 1 ELSE 0) | WEIGHT 1 | WEIGHT 2 | THRESHOLD (THETA) | NET (W1X1+W2X2) | DELTA THETA (-d) | DETLA WEIGHT (d) |
| 1 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 2 | 1 | 0 | 0 | **0** | 0.5 | 1.5 | 0.5 | 0.5 | 0 | 0 |
| 3 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 |
| 4 | 0 | 0 | 0 | **0** | -0.5 | 0.5 | 1.5 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 6 | 1 | 1 | 1 | **1** | 0.5 | 1.5 | 0.5 | 2 | 0 | 0 |

Since Net is greater than Theta (2 > 0.5), the Output is 1. It had properly classified and there is no need to update the Weights as there is no misclassification. We will continue with the same Weights.

**Iteration 7**

The Network computation is defined as Net = W1X1 + W2X2.If Net > Theta Then Output = 1 otherwise Output = 0.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ITERATION | INPUT 1 | INPUT 2 | TARGET | OUTPUT (NET>THETA THEN 1 ELSE 0) | WEIGHT 1 | WEIGHT 2 | THRESHOLD (THETA) | NET (W1X1+W2X2) | DELTA THETA (-d) | DETLA WEIGHT (d) |
| 1 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 2 | 1 | 0 | 0 | **0** | 0.5 | 1.5 | 0.5 | 0.5 | 0 | 0 |
| 3 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 |
| 4 | 0 | 0 | 0 | **0** | -0.5 | 0.5 | 1.5 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 6 | 1 | 1 | 1 | **1** | 0.5 | 1.5 | 0.5 | 2 | 0 | 0 |
| 7 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 |

Since Net is greater than Theta (1.5 > 0.5), the Output is 1. There is a misclassification, so the Weights need to be updated.

As we have misclassified Class 0 as Class 1, the Theta and Weights need to be updated. Delta Theta for **Threshold = -(0-1) = 1.**

The feedback signal for updating Weights is just the opposite. In this case, **d = -1 for Weights.**

Now, we have new set of Weights and Threshold

THETA = (0.5 + 1) = 1.5

W1 = (0.5 - 1) = -0.5

W2 = (1.5 - 1) = 0.5

Let's iterate with the new set of Weights and Threshold –

**Iteration 8**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ITERATION | INPUT 1 | INPUT 2 | TARGET | OUTPUT (NET>THETA THEN 1 ELSE 0) | WEIGHT 1 | WEIGHT 2 | THRESHOLD (THETA) | NET (W1X1+W2X2) | DELTA THETA (-d) | DETLA WEIGHT (d) |
| 1 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 2 | 1 | 0 | 0 | **0** | 0.5 | 1.5 | 0.5 | 0.5 | 0 | 0 |
| 3 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 |
| 4 | 0 | 0 | 0 | **0** | -0.5 | 0.5 | 1.5 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | **0** | -0.5 | 0.5 | 1.5 | 0 | -1 | 1 |
| 6 | 1 | 1 | 1 | **1** | 0.5 | 1.5 | 0.5 | 2 | 0 | 0 |
| 7 | 0 | 1 | 0 | **1** | 0.5 | 1.5 | 0.5 | 1.5 | 1 | -1 |
| 8 | 0 | 1 | 0 | **0** | -0.5 | 0.5 | 1.5 | 0.5 | 0 | 0 |

Since Net is not greater than Theta (0.5 < 1.5), the Output is 0. It had properly classified and there is no need to update the Weights as there is no misclassification. We will continue with the same Weights.

The Perceptron Learning **converges** with all the given points being *classified correctly*.

**Visualizing the Decision Boundary for these Point Pairs**

The summation that Perceptron uses to determine its output is the **dot product** of the Inputs (x1, x2…xn) and Weights vectors (w1, w2…wn), plus the Bias (b) – **w1 \* x1 + w2 \* x2 + b**

Considering x1 to be x and x2 to be y, the equation can be rewritten as – **w1x + w2y + b.**

This is like the Standard equation of a Straight line – **Ax + By – C = 0**

We can now solve for two points on our graph – **x-intercept and y-intercept.**

**x-intercept = -(b – w2y)/w1. If y = 0 then x = -b/w1**

**y-intercept = -(b – w1x)/w2. If x = 0 then y = -b/w2**

With intercepts, we can calculate the slope of the line, **m:**

Point 1 – (0, -b/w1); Point 2 – (-b/w2, 0)

m = (y2 – y1) / (x2 – x1)

m = (0 - -(b/w1)) / (-(b/w2) – 0)

m = -(b/w2) / (b/w1)

With all the above values, we can now construct our line in slope-intercept form –

Slope (m) = -(b/w2) / (b/w1)

y-intercept = (-(b / w2) / (b / w1))x + (-b / w2)

**Plotting the Decision Boundary for the given Input Pairs**

|  |  |
| --- | --- |
| **(x1, x2)** | **AND** |
| (0 , 0) | 0 |
| (0 , 1) | 0 |
| (1 , 0) | 0 |
| (1 , 1) | 1 |

We see that all pairs of inputs that return 0 are **blue** and on one side of the line, and the input that gives us 1 (**orange**), is on the other side of the line.

**Learning steps in the W1-W2 two-dimensional space with loss (count of misclassified points):**

*Error function for the AND logic Solution Region for AND function has a triangular shape when threshold is fixed at 1.*

