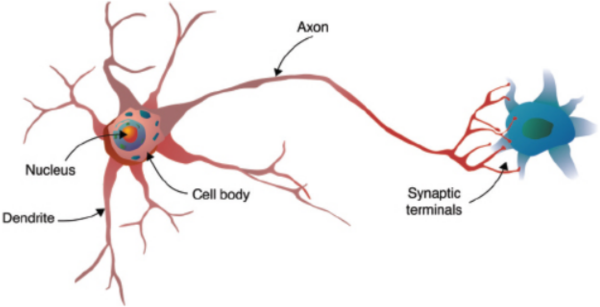
The Rosenblatt Perceptron

José Matheus

May 19, 2025

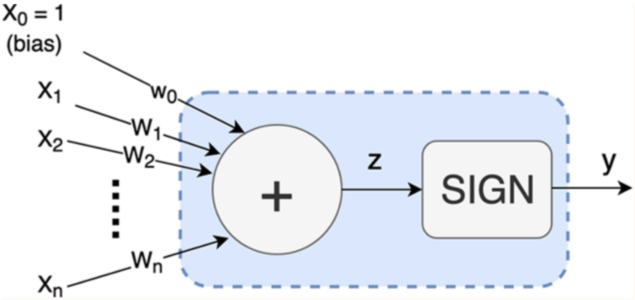
Introduction

- The perceptron is an artificial neuron, that is, a model of a biological neuron.
- A biological neuron consists of one cell body, multiple dendrites, and a single axon.



Perceptron

- The perceptron consists of a computational unit (dashed rectangle), a number of inputs (one of which is a special bias input), each with an associated input weight and a single output.



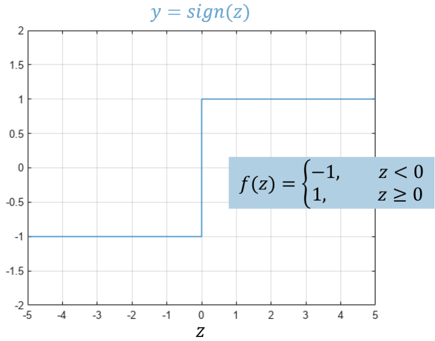
- The inputs and output correspond to the dendrites and the axon, and the unit computational corresponds to the cell body.

Perceptron

- A perceptron sums up the inputs to compute an intermediate value z, which is fed to an activation function.

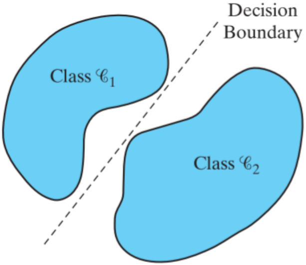
$$z = \sum_{i=0}^{n} w_i x_i$$

- The perceptron uses the sign function as an activation function.



Learning Algorithm

- The perceptron learning algorithm is what is called supervised learning algorithm, the model that is being trained is presented with both the input data and the desired output data (also known as ground truth).
- The perceptron is the simplest form of a neural network used for the classification of pattern said to be lineraly separable(i.e., patterns that lie on opposite sides of a hyperplane).



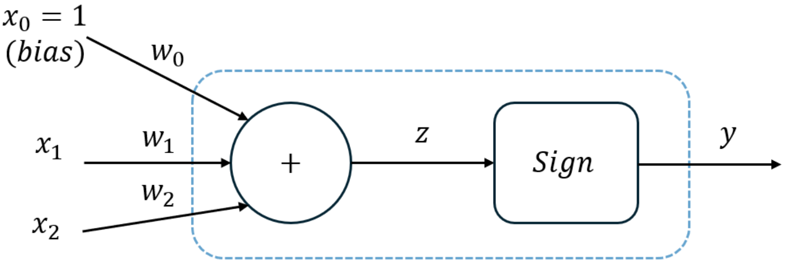
(a) (b)

Learning Algorithm

1. Randomly initialize the weights.
2. Select one input/output pair at random.
3. Present the values $x_1, ..., x_n$ to the perceptron to compute the output.
4. If the output y is different from the the ground truth for this input/output pair, adjust the weights it the following way:
    1. if $y < 0$, add ηx_i to each w_i.
    2. if $y > 0$, subtract ηx_i from each w_i.
5. Repeat steps 2, 3, and 4 until the perceptron predicts all examples correctly.

Two-Input Perceptron

- Let us study a perceptron with two inputs in addition to the bias input.



Two-Input Perceptron
Decision Boundary

$$z = \sum_{i=0}^{n} w_i x_i = w_0 x_0 + w_1 x_1 + w_2 x_2$$

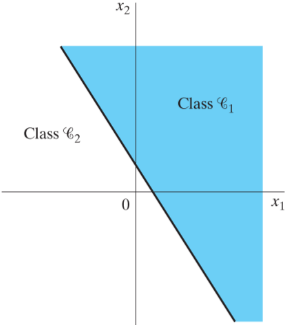
Boundary Condition

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$$

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} x_0$$

$$x_2 = -\frac{w_1}{w_2} x_1 + b$$

, in which $b = -\frac{w_0}{w_2} x_0$ (Intercept).



Example: Logical AND Gate

Truth Table

x_1	x_2	y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

- The values of the inputs and output can also be interpret as Boolean values, where -1 represents False and 1 represents True.

Example: Logical AND Gate

See https://github.com/JTeus/Apresenta-o-perceptron

Import

```
[1]: import random

     def show_learning(w):
         print('w0 =', '%5.2f' % w[0], ', w1 =', '%5.2f' % w[1],
               ', w2 =', '%5.2f' % w[2])
```

Define variables needed to control training process.

```
[2]: random.seed(7) # To make repeatable
     LEARNING_RATE = 0.1
     index_list = [0, 1, 2, 3] # Used to randomize order
```

η *(Hyperparameter)*

Define training examples.

```
[3]: x_train = [(1.0, -1.0, -1.0), (1.0, -1.0, 1.0),
         (1.0, 1.0, -1.0), (1.0, 1.0, 1.0)] # Inputs

     y_train = [-1.0, -1.0, -1.0, 1.0] # Output (ground truth)
```

y_{train}

x_1	x_2	y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

x_{train}

Define perceptron weights.

```
[4]: w = [0.2, -0.6, 0.25] # Initialize to some "random" numbers

     # Print initial weights.
     show_learning(w)

w0 =  0.20 , w1 = -0.60 , w2 =  0.25
```

Perceptron Function

[5]:

```
# First element in vector x must be 1.
# Length of w and x must be n+1 for neuron with n inputs.
def compute_output(w, x):
    z = 0.0
    for i in range(len(w)):
        z += x[i] * w[i] # Compute sum of weighted inputs
    if z < 0: # Apply sign function
        return -1
    else:
        return 1
```

$$z = \sum_{i=0}^{n} x_i w_i$$

$$f(z) = \begin{cases} -1, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

Example: Logical AND Gate

Training Loop and results

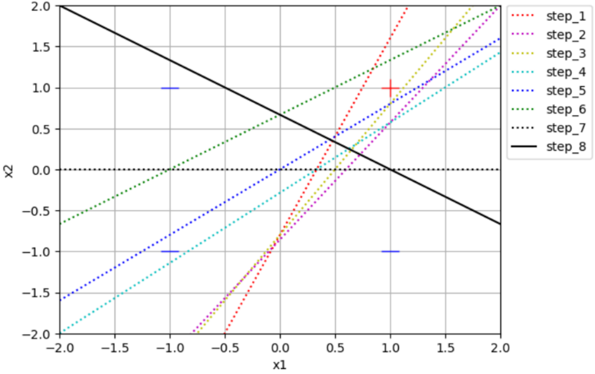
Perceptron Training Loop

[6]:
```python
# Perceptron training loop.
all_correct = False
while not all_correct:
    print(all_correct)
    all_correct = True
    random.shuffle(index_list) # Randomize order
    for i in index_list:
        x = x_train[i]
        y = y_train[i]
        p_out = compute_output(w, x) # Perceptron function
        if y != p_out: # Update weights when wrong
            for j in range(0, len(w)):
                w[j] += (y * LEARNING_RATE * x[j])
            all_correct = False
            show_learning(w) # Show updated weights
print(all_correct)
```

```
False
w0 =  0.30 , w1 = -0.50 , w2 =  0.35
w0 =  0.20 , w1 = -0.40 , w2 =  0.25
w0 =  0.10 , w1 = -0.30 , w2 =  0.35
False
w0 =  0.00 , w1 = -0.20 , w2 =  0.25
False
w0 = -0.10 , w1 = -0.10 , w2 =  0.15
w0 =  0.00 , w1 = -0.00 , w2 =  0.25
False
w0 = -0.10 , w1 =  0.10 , w2 =  0.15
False
True
```

Example: Logical AND Gate

Learning process



$$z = -0.1\overset{1}{\cancel{x_0}} + 0.1x_1 + 0.15x_2$$

$$0 = -0.1 + 0.1x_1 + 0.15x_2$$

$$\boxed{x_2 = \frac{2}{3}(1 - x_1)}$$

Example: Logical AND Gate

Results

$$z(x_1, x_2) = -0.1 + 0.1x_1 + 0.15x_2$$

$$\left.\begin{aligned} z(-1,-1) &= -0.35 \\ z(-1,1) &= -0.05 \\ z(1,-1) &= -0.15 \end{aligned}\right\} z < 0 \dashrightarrow f(z) = -1$$

$$z(1,1) = 0.15 \qquad z > 0 \dashrightarrow f(z) = 1$$

x_1	x_2	y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

Exercises
logical NAND Gate

x_1	x_2	y
-1	-1	1
-1	1	1
1	-1	1
1	1	-1

Exercises

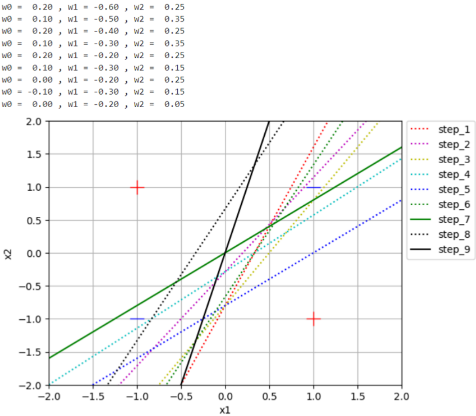
logical NOR Gate

x_1	x_2	y
-1	-1	1
-1	1	-1
1	-1	-1
1	1	-1

Limitations of the Perceptron

Logical XOR Gate

x_1	x_2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



Conclusion

- Perceptron can solve classification problems only where the classes are linearly separable.
- Perceptron is a binary classifier.
- The perceptron built around a single neuron is limited to performing pattern classification with only two classes. By expanding the output (computation) layer of the perceptron to include more than one neuron, we may correspondingly perform classification with more than two classes. However, the classes have to be linearly separable for the perceptron to work properly.

Bibliography