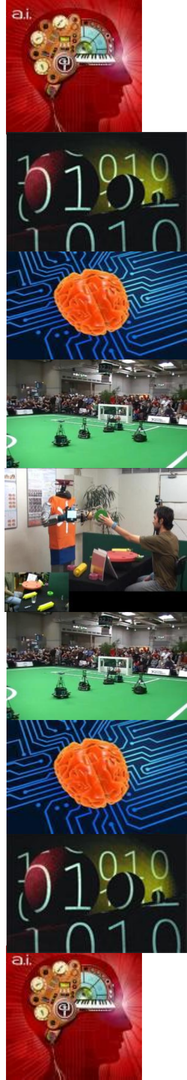
Lecture Outline

- Introduction to Neural Networks.
- Architecture.
- History.
- Perceptron.

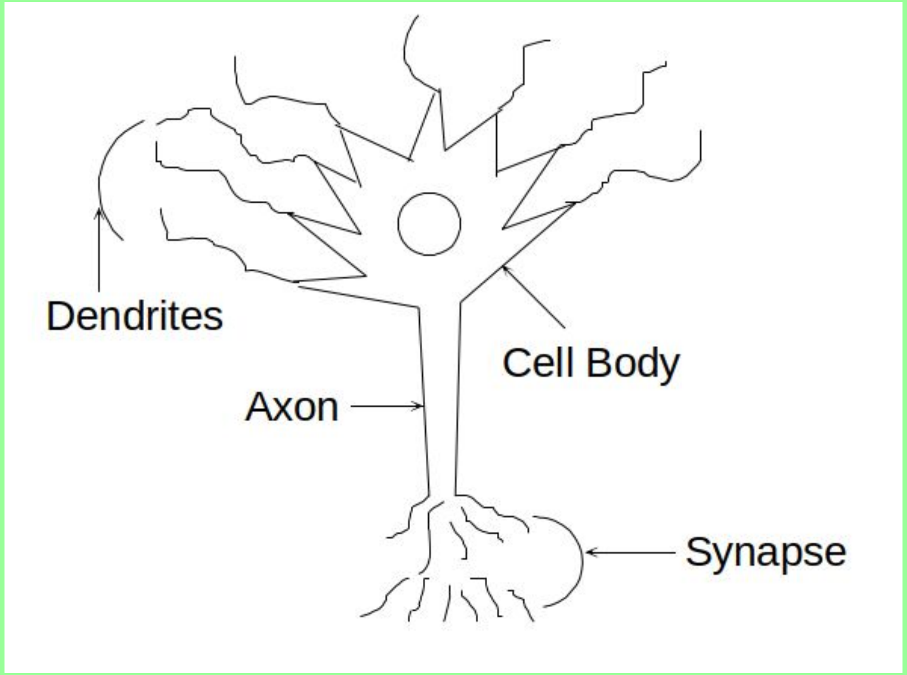
Neural Networks

Introduction to Neural Networks

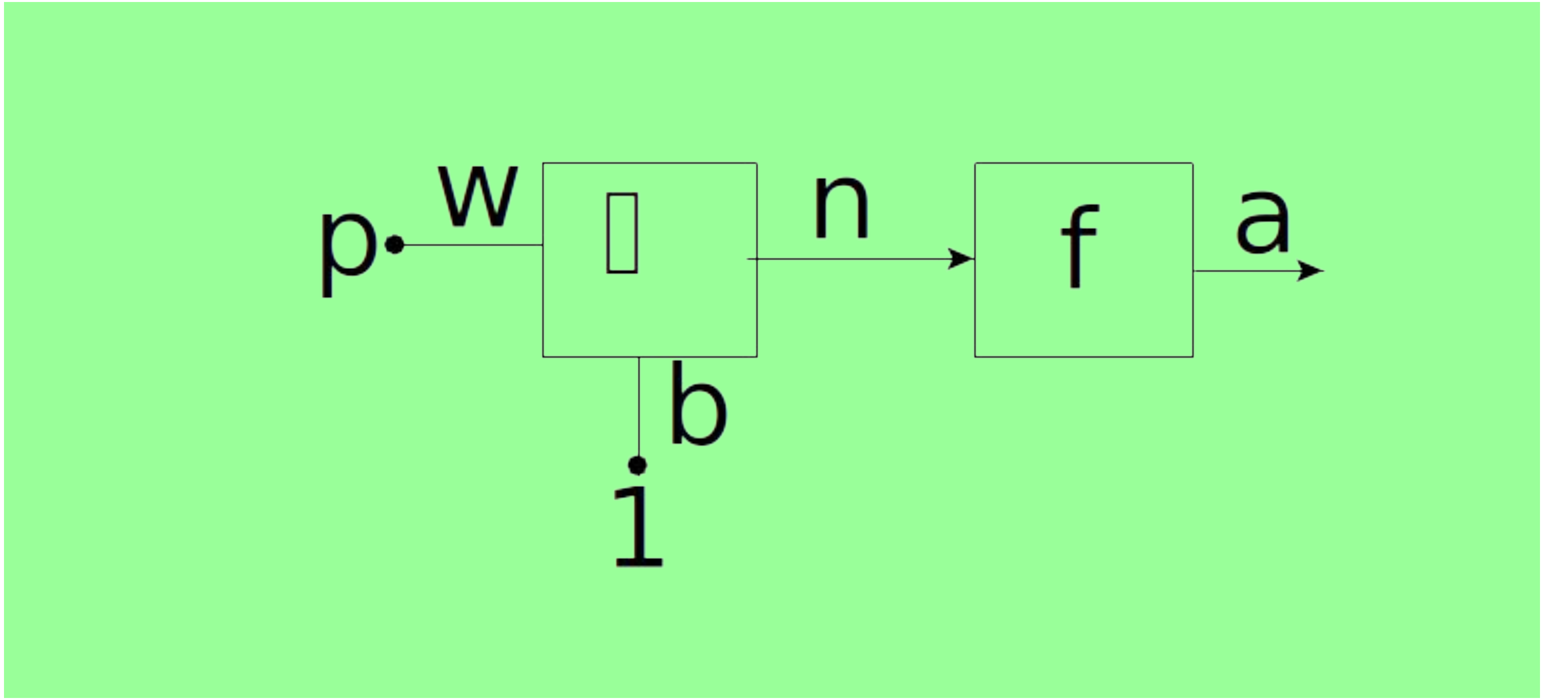
- Analogy from how the brain works.
- Parallel processing.
- Software and hardware implementations.
- Number of neural network architectures and learning algorithms developed over time.

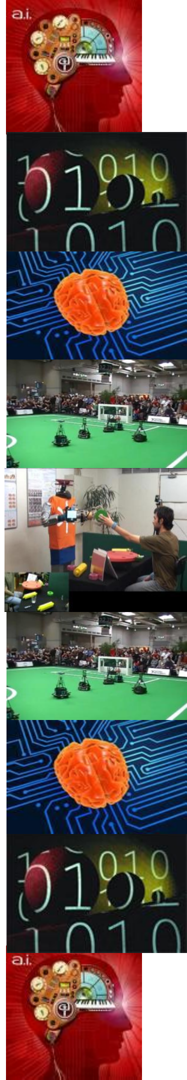


Biological Neuron



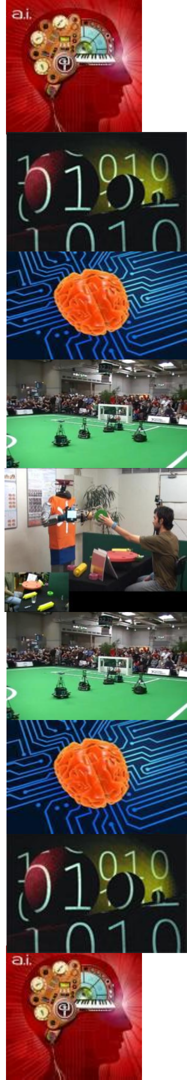
Computational Neuron





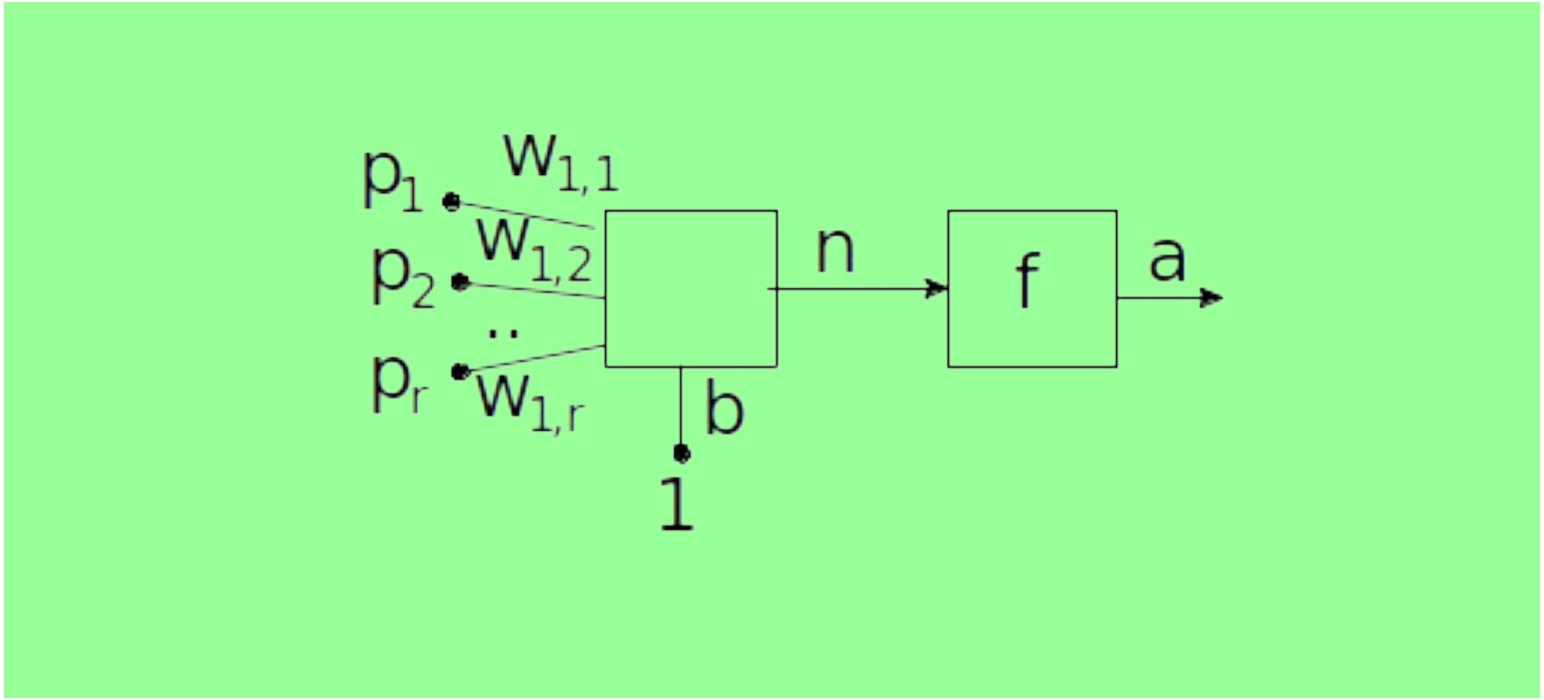
Computational Neural Network

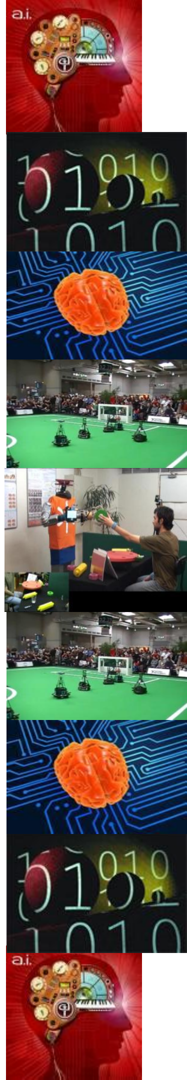
- Processing
- Weights and bias
- Training and learning algorithms
- Training set
- Activation functions
- Testing the neural network



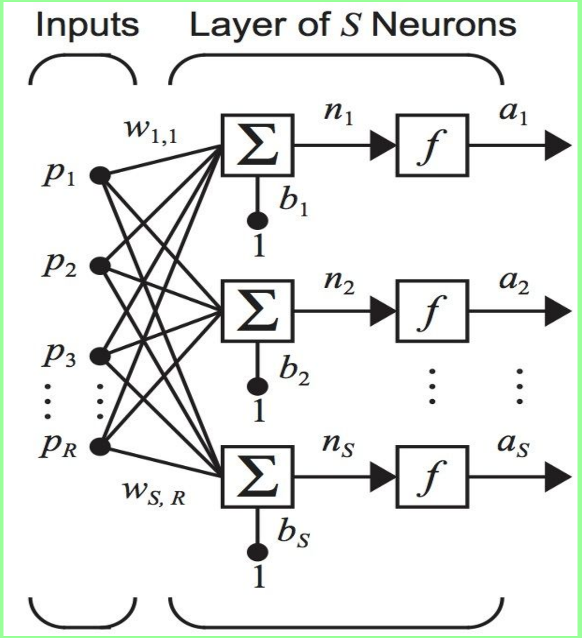
Neural Networks Architectures

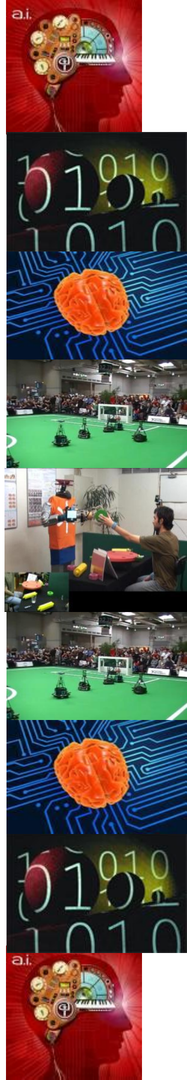
Single Neuron Single Layer



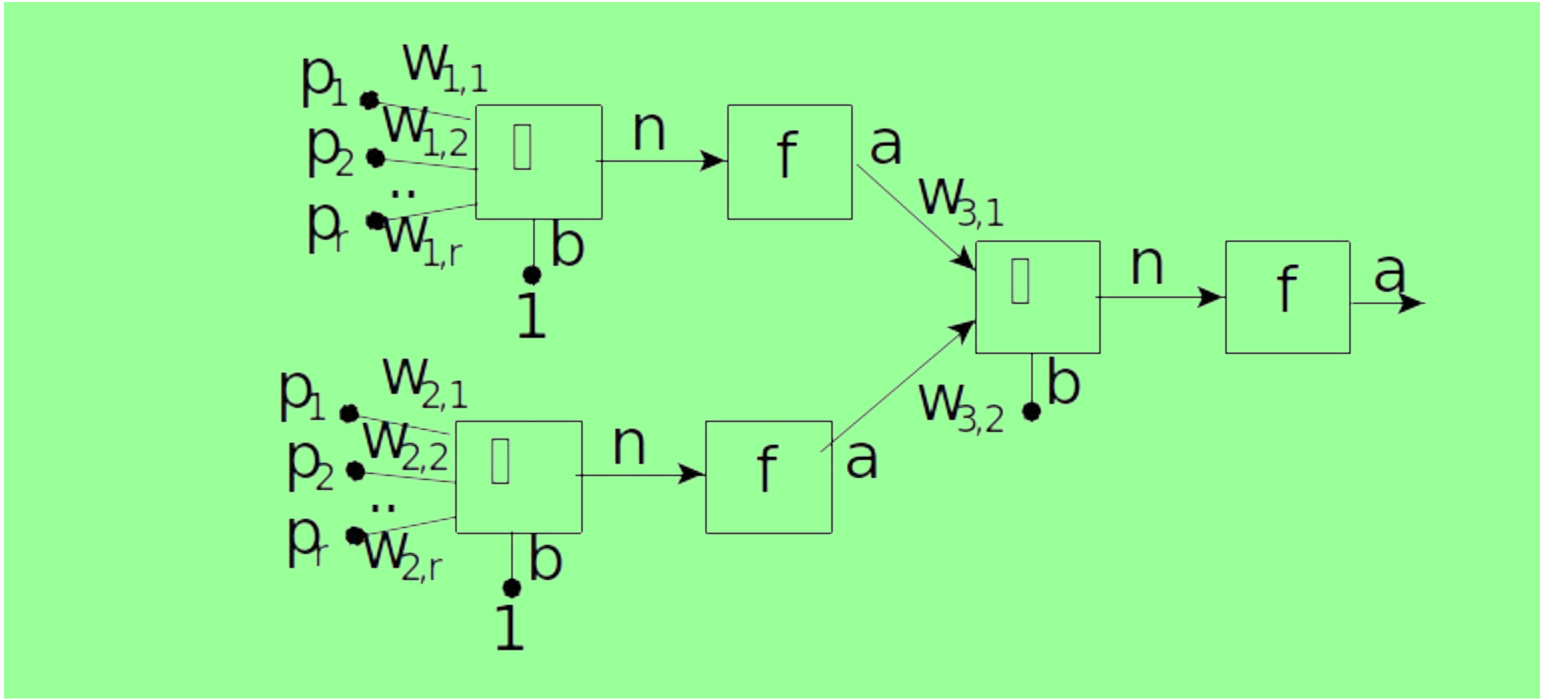


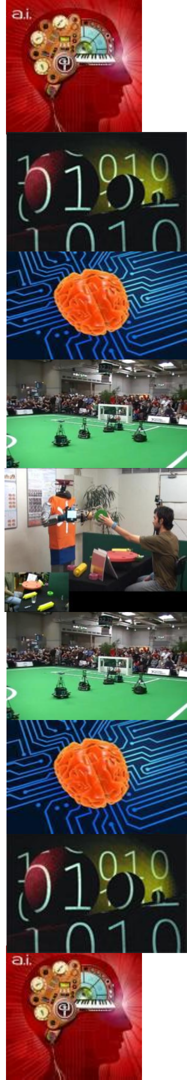
Multiple Neurons Single Layer



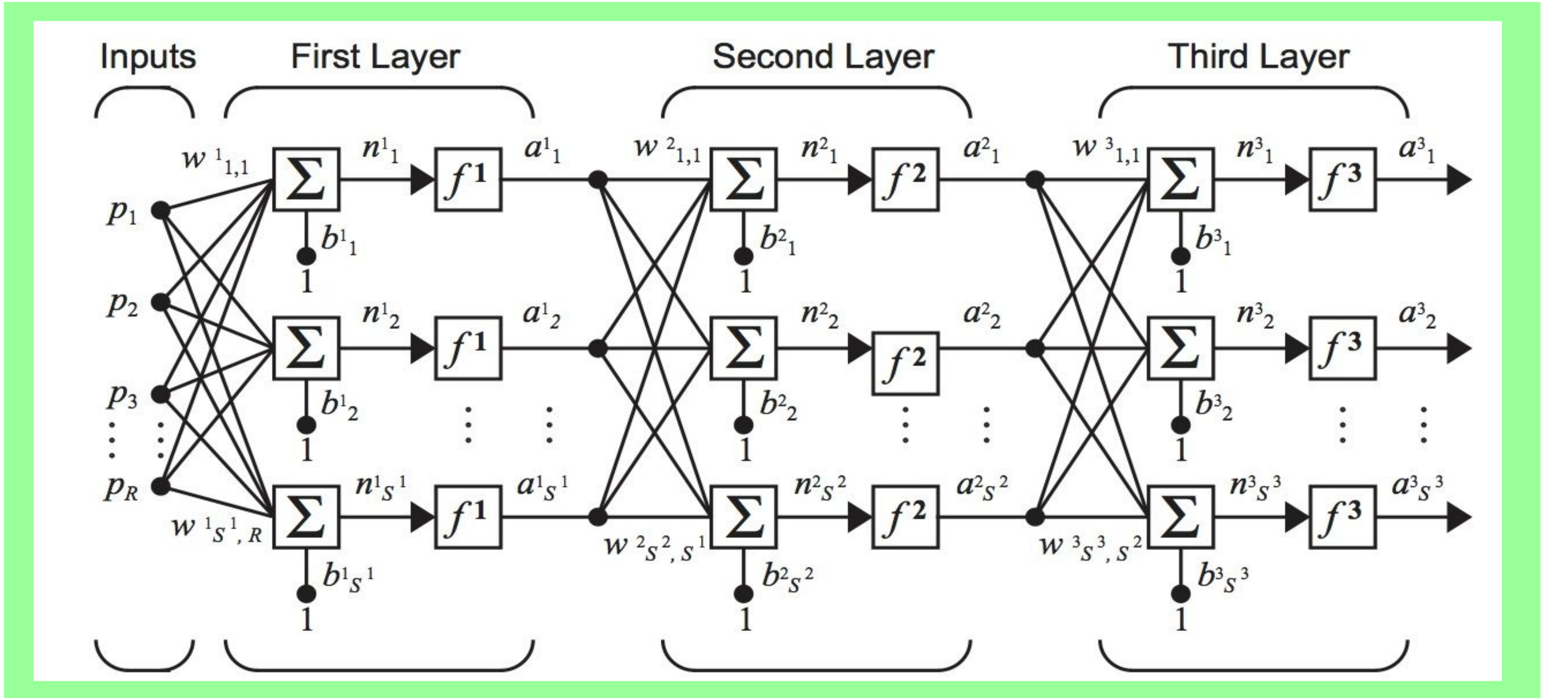


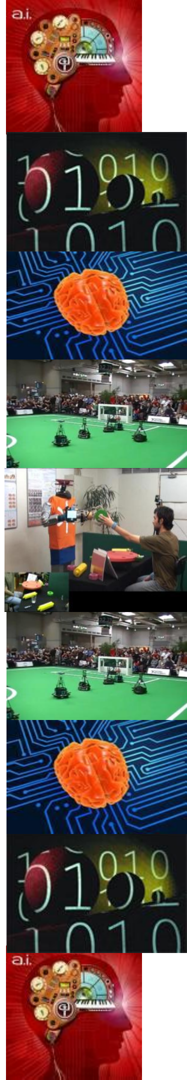
Two Layer Neural Network





Three Layer Neural Network





History of Neural Networks

History of Neural Networks

- The Beginning of Neural Networks (1940's):McCulloch Pitts Neuron, Hebbian Learning
- The First Golden Age of Neural Networks (1950's and 1960's): Perceptrons, Adaline
- The Quiet Years-1970's}: Kohenen, Anderson,Grossberg, Carpenter.

History of Neural Networks

- Renewed Enthusiasm-1980’s: Backpropagation, Hopfield nets, Neocognitron,Boltzman machine, hardware Implementation.
- Recurrent Neural Networks and Gradient-Based Learning (1990's): Long Short-Term Memory (LSTM), gradient-based learning.
- Currently: Deep learning, convolutional neural networks and transformer Networks.

Types Neural Networks

- Pattern recognition function,
  - pattern classification and
  - pattern association
    - autoassociation
    - hetero-association.

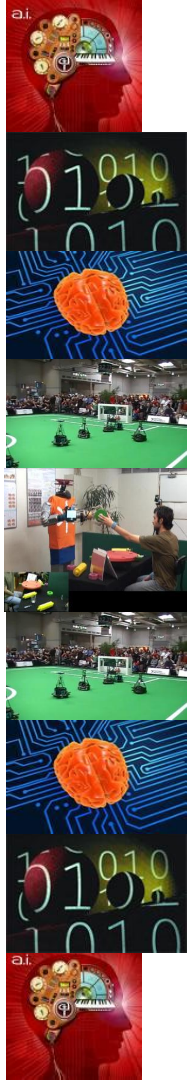
McCulloch-Pitts Neuron

McCulloch-Pitts Neuron

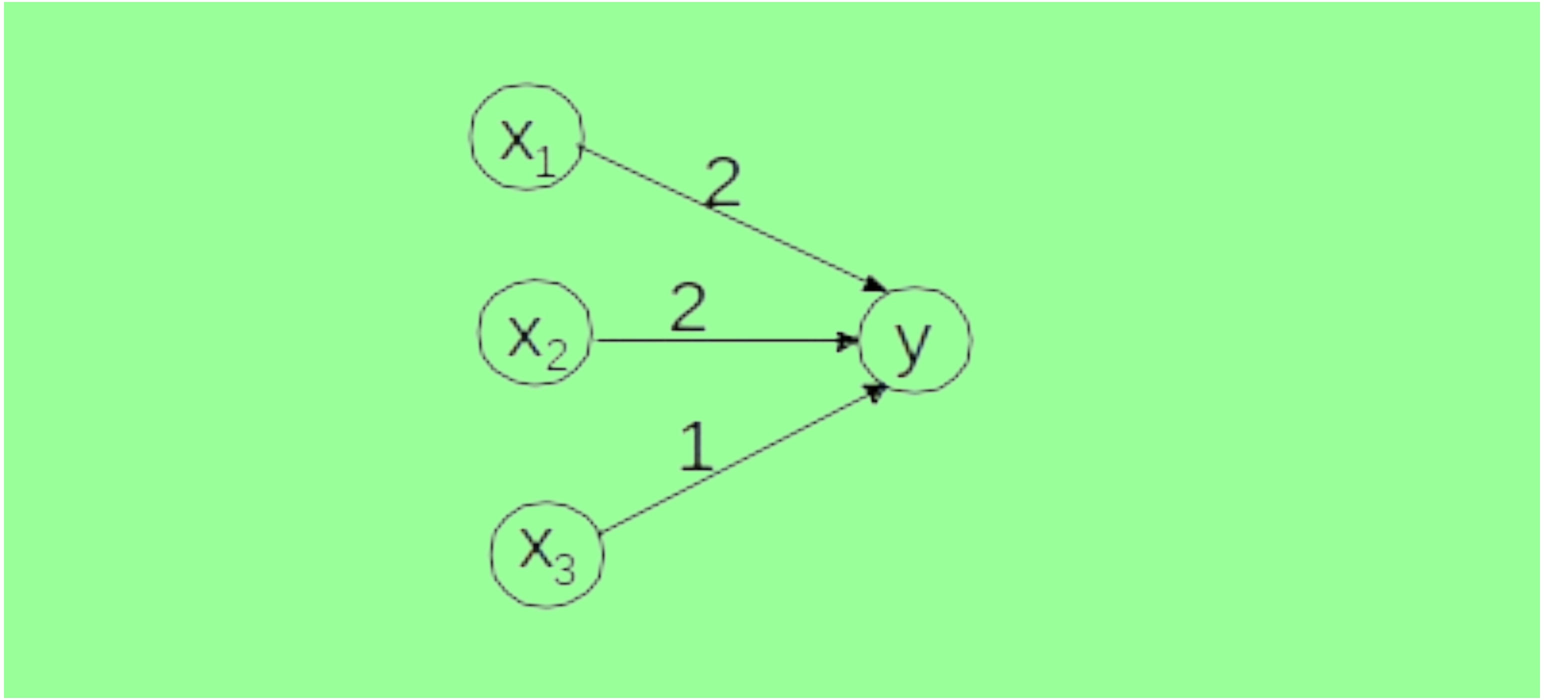
- The first neuron
    - The McCulloch-Pitts neuron takes binary inputs
    - The activation function used is:

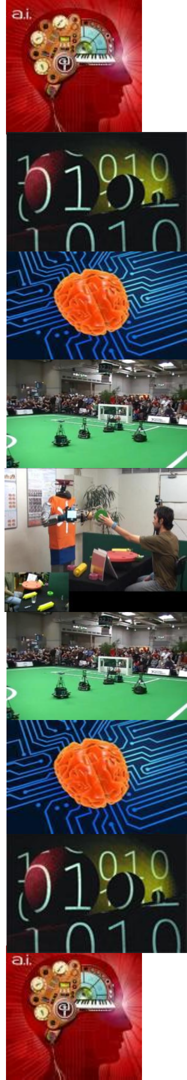
      $f(n) = 1$ if $n >= \theta$

      $= 0$ if $n < \theta$
- Theta is a parameter value



McCulloch-Pitts Neuron Example

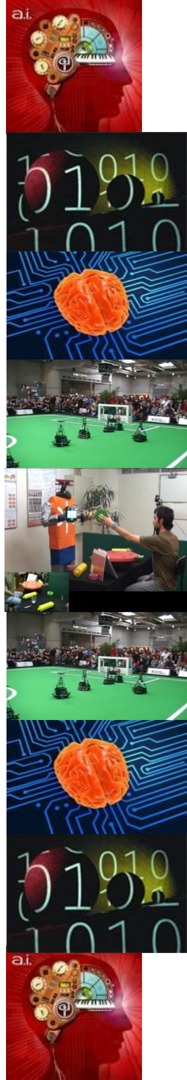




McCulloch-Pitts Neuron Example

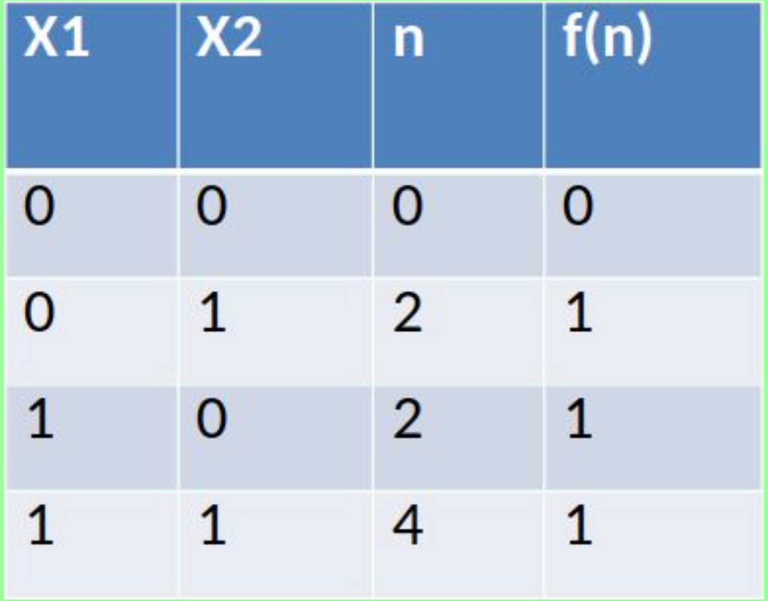
OR - function

X1	X2	Target
0	0	0
0	1	1
1	0	1
1	1	1

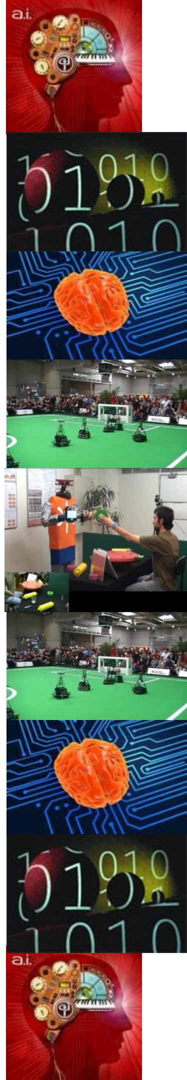


McCulloch-Pitts Neuron Example

McCulloch-Pitts Neuron to Perform the OR Function, w1=2, w2=2, θ=2



X1	X2	n	f(n)
0	0	0	0
0	1	2	1
1	0	2	1
1	1	4	1

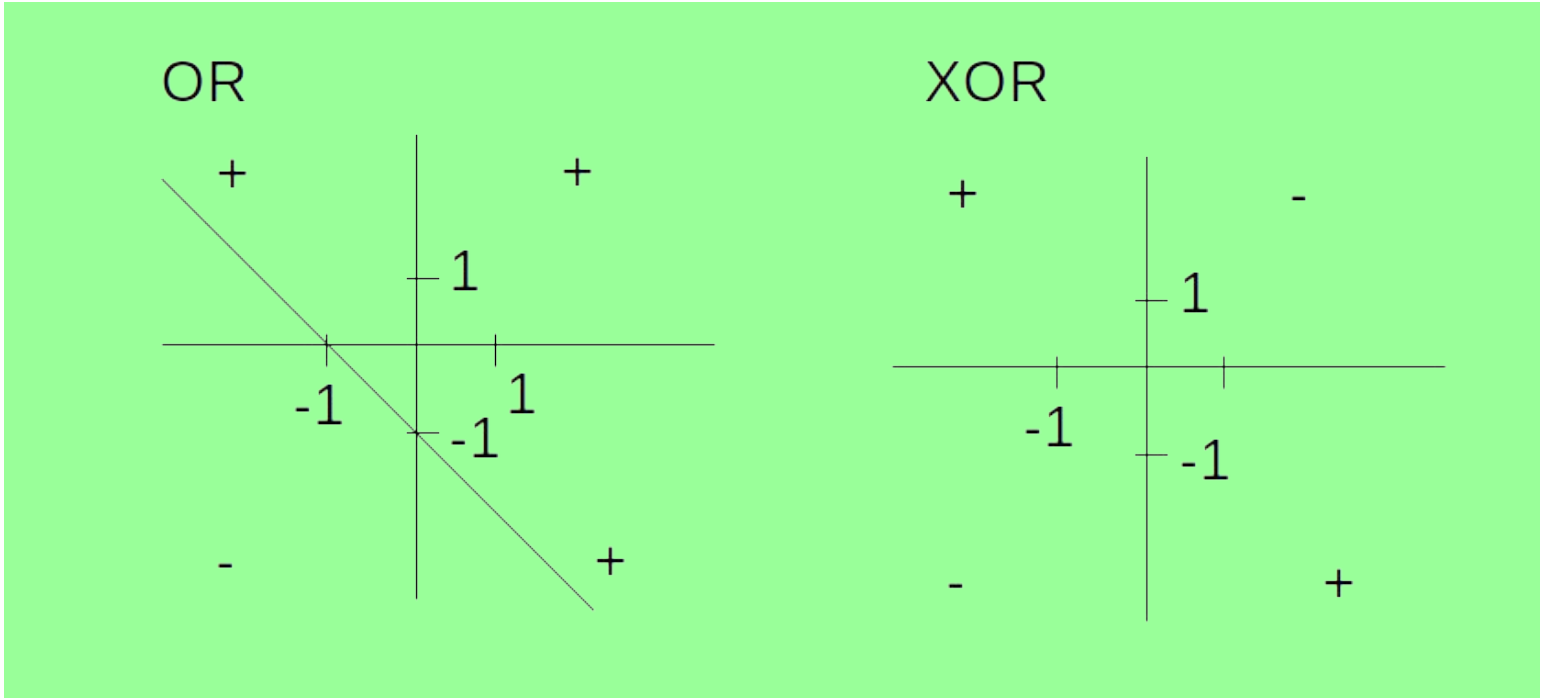


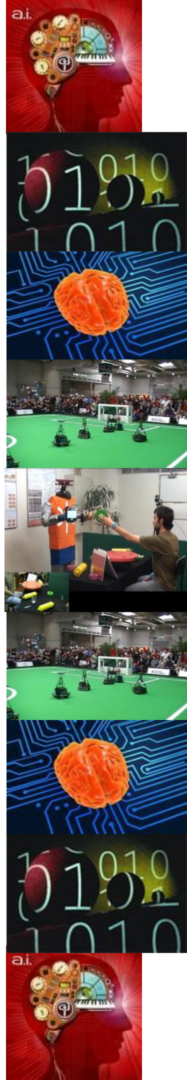
Linear Separability

Linear Separability

Is it possible to train a McCulloch-Pitts neuron to perform the XOR logical function?

Linear Separability





Perceptron

Introduction

- Performs pattern classification
- Feedforward neural network
- Single layered or multilayered
- Training

  -Determining weights

  -Determining bias

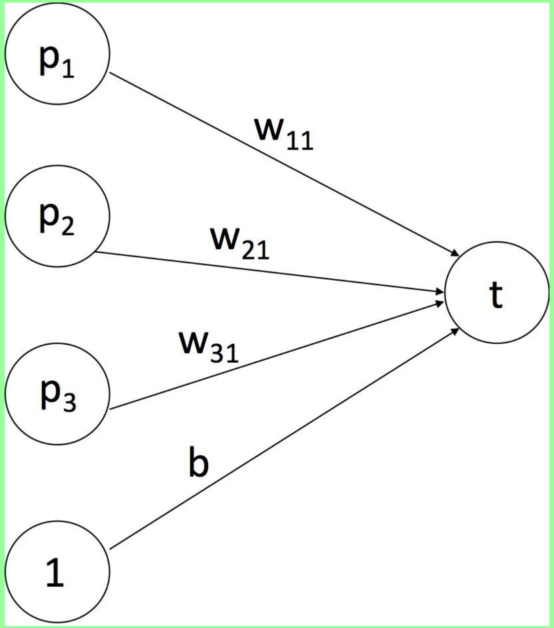
Introduction

- Learning algorithm
- Epochs
- Convergence of learning algorithms
- Classification and training set
  - Inputs
  - Outputs
  - Binary vs. bipolar

Binary Classification Example

- Conveyor belt to separate fruit
- Attributes
  - Shape
  - Texture
  - Weight
- Orange
  - Input: [1 -1 -1]
  - Output: -1
- Apple
  - Input: [1 1 -1]
  - Outputs: 1

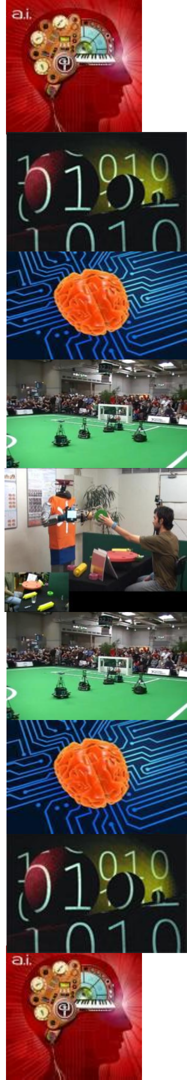
Example



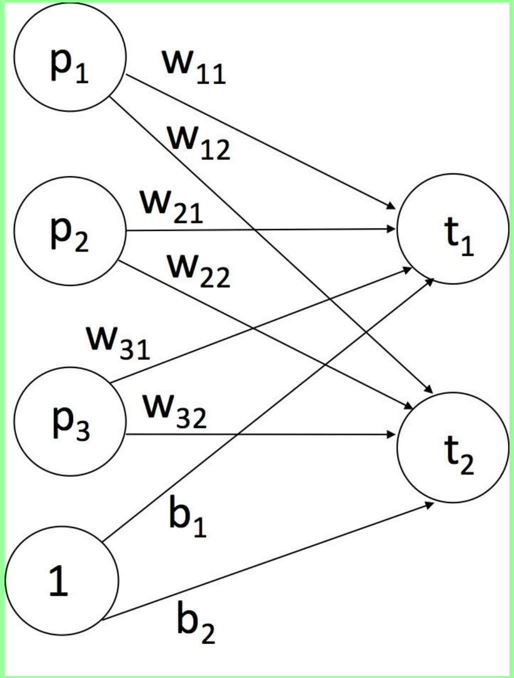
$$W = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Multi-classification Example

- Grapefruit
  - -Input: [1 -1 1]
  - -Output: [1 1]
- Orange
  - - Input: [1 -1 -1]
  - - Output: [-1 1]
- Apple
  - -Input: [1 1 -1]
  - -Outputs: [1 -1]



Multi-classification Example



$$W = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix}$$

$$b = [b_1 \, b_2]$$

Activation Function

- **Binary**

  $f(n) = 1 \quad \text{if } n >= 0$

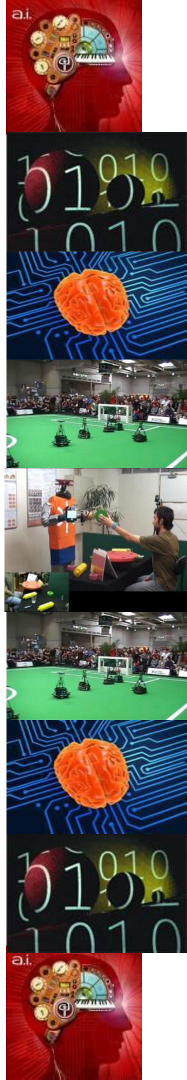
  $f(n) = 0 \quad \text{if } n < 0$

- **Bipolar**

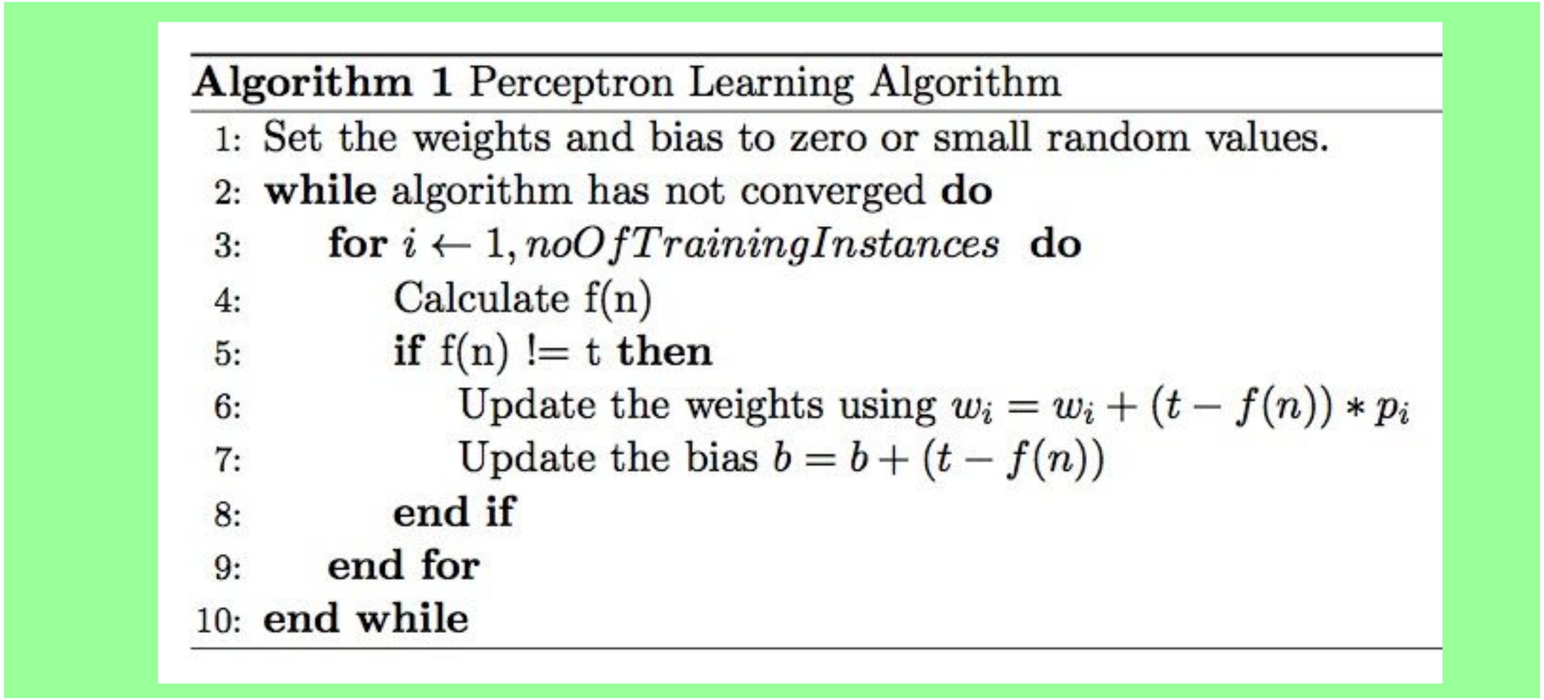
  $f(n) = 1 \quad \text{if } n >= 0$

  $f(n) = -1 \quad \text{if } n < 0$

- **+ b**

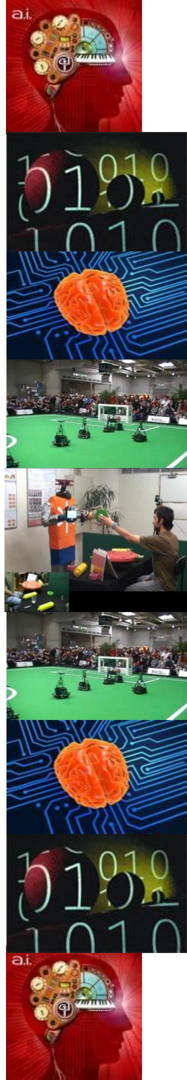


Perceptron Learning Algorithm



Algorithm 1 Perceptron Learning Algorithm

```
1: Set the weights and bias to zero or small random values.
2: while algorithm has not converged do
3:     for $i \leftarrow 1, noOfTrainingInstances$ do
4:         Calculate f(n)
5:         if f(n) != t then
6:             Update the weights using $w_i = w_i + (t - f(n)) * p_i$
7:             Update the bias $b = b + (t - f(n))$
8:         end if
9:     end for
10: end while
```



Learning rule

Used to update the weights and biases

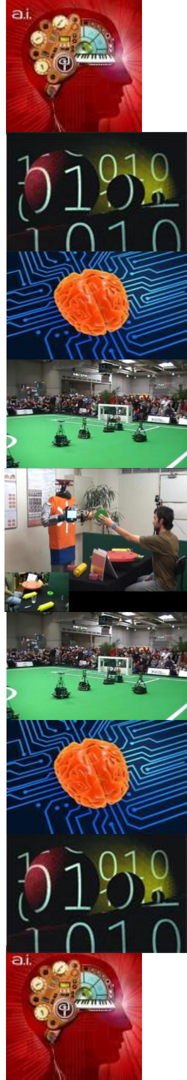
- $wi = wi + \ (t - f(n)) * pi$
- $b = b + (t - f(n))$

α- learning rate

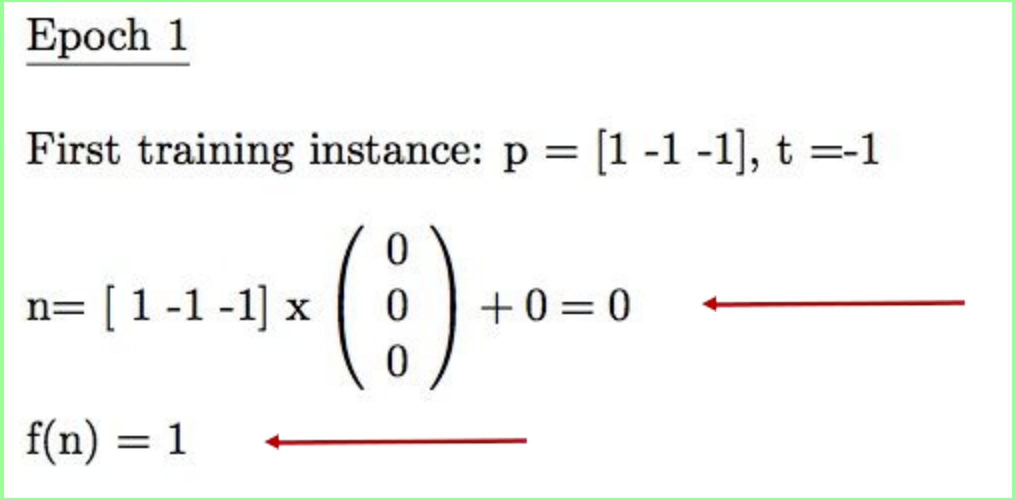
- $wi = wi + \alpha * (t - f(n)) * pi$
- $b = b + \alpha(t - f(n))$

Example

p1	p2	p3	t
1	-1	-1	-1
1	1	-1	1



Example

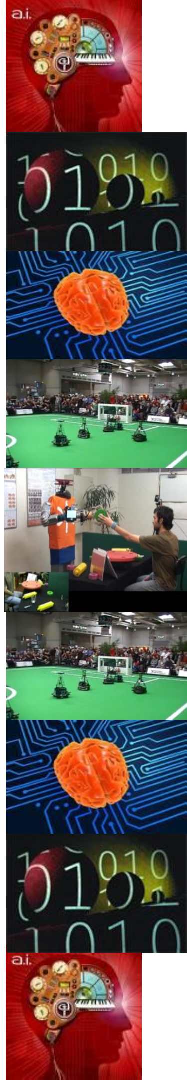


Epoch 1

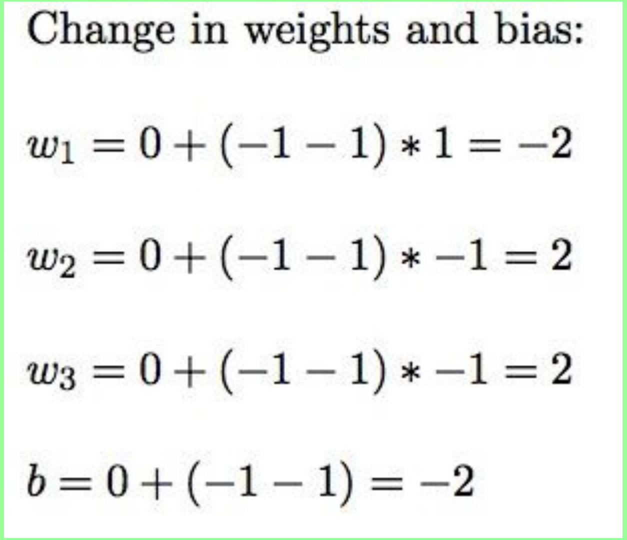
First training instance: $p = [1\ {-1}\ {-1}]$, $t = -1$

$$n = [\,1\ {-1}\ {-1}\,] \times \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + 0 = 0 \longleftarrow$$

$$f(n) = 1 \longleftarrow$$



Binary Classification Example



Change in weights and bias:

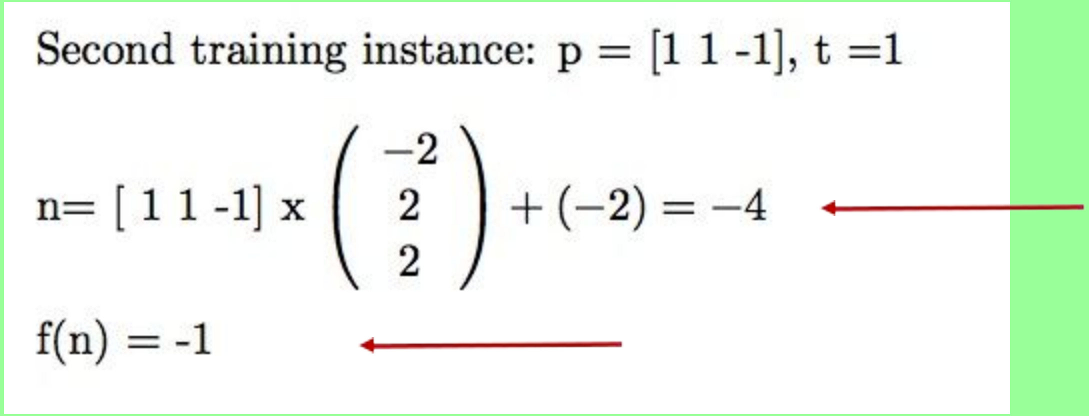
$$w_1 = 0 + (-1 - 1) * 1 = -2$$

$$w_2 = 0 + (-1 - 1) * -1 = 2$$

$$w_3 = 0 + (-1 - 1) * -1 = 2$$

$$b = 0 + (-1 - 1) = -2$$

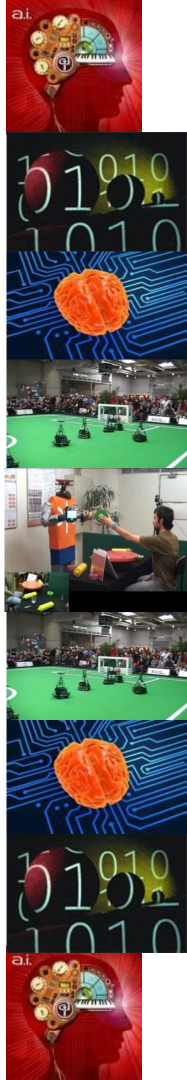
Binary Classification Example



Second training instance: p = [1 1 -1], t =1

$$n = [\,1\ 1\ {-1}\,] \times \begin{pmatrix} -2 \\ 2 \\ 2 \end{pmatrix} + (-2) = -4$$

$$f(n) = -1$$



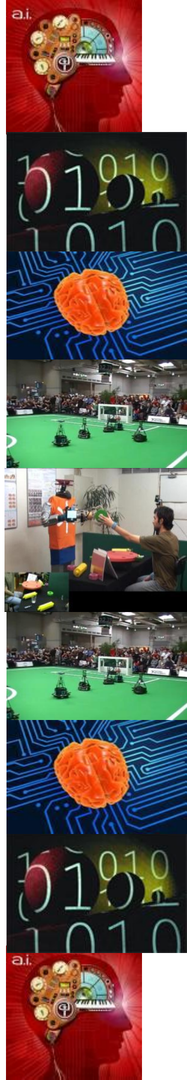
Binary Classification

$$w_1 = -2 + (1 - (-1)) * 1 = 0$$

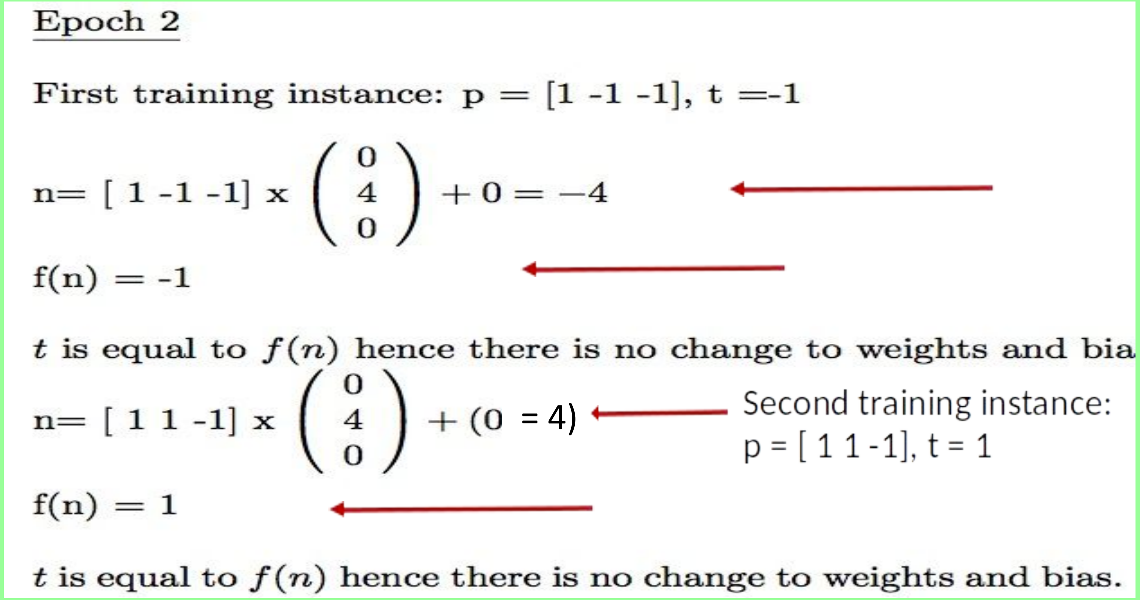
$$w_2 = 2 + (1 - (-1)) * 1 = 4$$

$$w_3 = 2 + (1 - (-1)) * -1 = 0$$

$$b = -2 + (1 - (-1)) = 0$$



Classification Example



Epoch 2

First training instance: p = [1 -1 -1], t =-1

$$n = [\,1\ {-1}\ {-1}\,] \times \begin{pmatrix} 0 \\ 4 \\ 0 \end{pmatrix} + 0 = -4$$

f(n) = -1

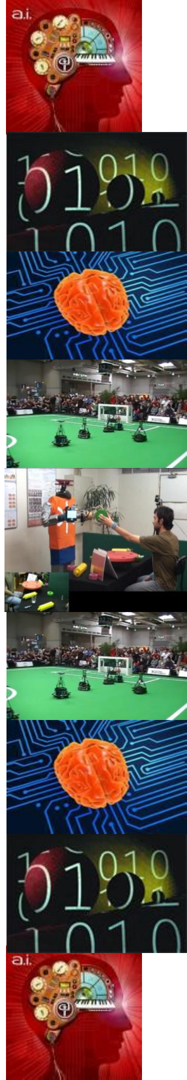
t is equal to $f(n)$ hence there is no change to weights and bia

$$n = [\,1\ 1\ {-1}\,] \times \begin{pmatrix} 0 \\ 4 \\ 0 \end{pmatrix} + (0\ = 4)$$

Second training instance:
p = [1 1 -1], t = 1

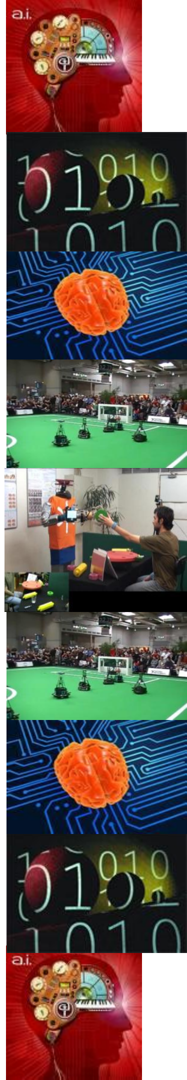
f(n) = 1

t is equal to $f(n)$ hence there is no change to weights and bias.



Multi-layer Perceptron

- Single layer
- Multilayer
- Learning algorithms
  - backpropagation



Terminology

- **Batch size** refers to the number of training examples used in one iteration of the training algorithm
    - If batch size is equal to the total number of training examples, the training algorithm is referred to as **batch gradient descent.**
    - If the batch size is equal to 1, the training algorithm is referred to as **stochastic gradient descent (SGD)**
    - If the batch size is between 1 and the total number of training examples, the training algorithm is referred to as **mini-batch gradient descent.**

Terminology

- A larger batch size can result in more stable updates to the weights and biases, but may also result in slower convergence and worse generalization performance

- A smaller batch size can result in faster convergence and better generalization performance and noisy updates.

- **Epoch:** refers to a single iteration during which the entire training dataset is processed.

Next Lecture - Backpropagation

QUESTIONS