Redes Unidireccionales

Deisy Chaves
Oficina 10, 4rto Piso, Edificio B13
deisy.chaves@correounivalle.edu.co

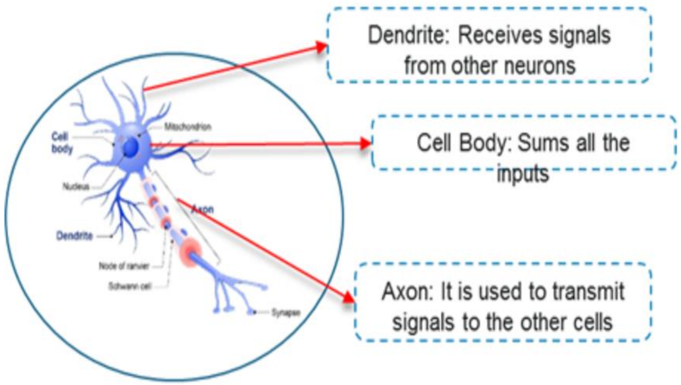
Contenido

- Redes unidireccionales
  - Perceptrón simple: Aprendizaje
  - Gradiente descendente

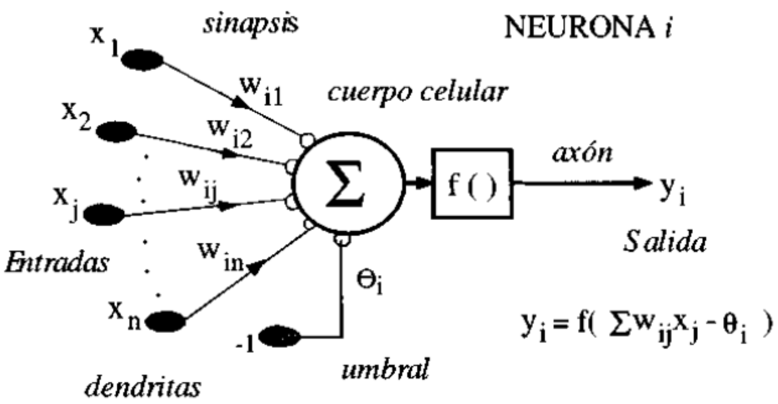
Neural networks

- A neural network is a collection of non-linear connected units called artificial neuron (analogous to a neuron in a biological brain)
  - Organized in layers of signaling cascades
  - Each neuron transmits a signal to another neuron

Biological neuron

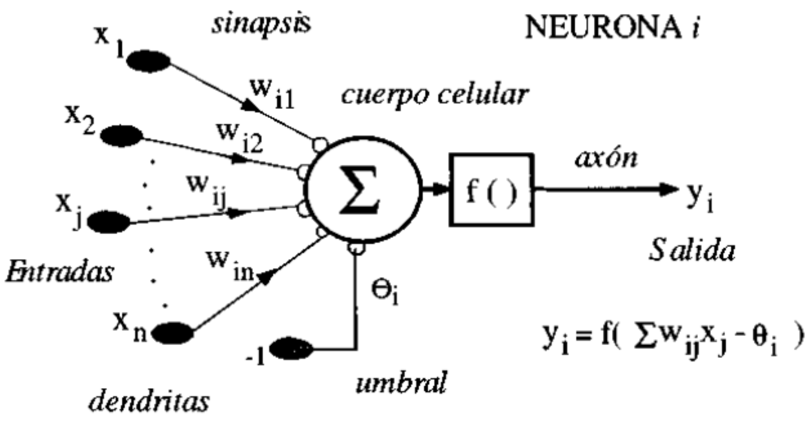


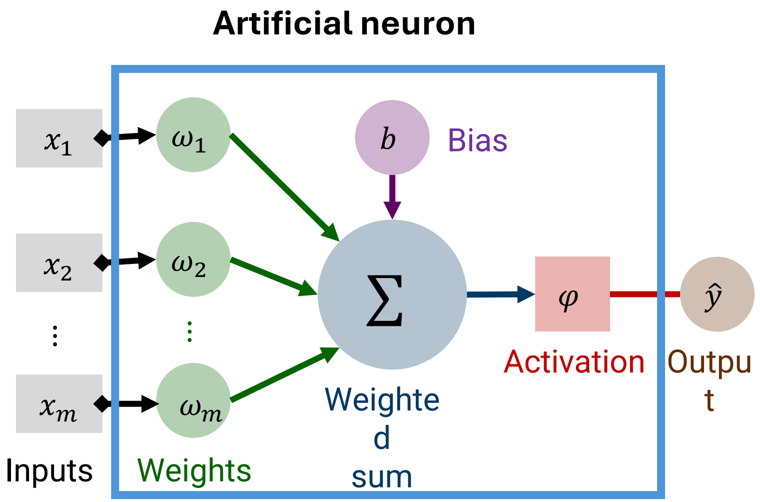
Artificial neuron



Artificial neuron

- Input features, *xj*
- Weights, *wi*
- Weighted sum (linear function):
  $w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n + b = Z$
- Activation function, φ :
  $\hat{y} = \varphi\,(w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n + b)$
- Bias





Perceptron

Cornell Aeronautical Laboratory

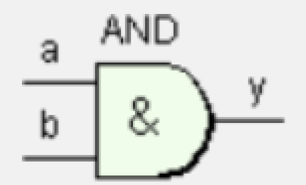
Rosenblatt & Mark I Perceptron: the first machine that could "learn" to recognize and identify optical patterns.

- It is the simplest neural network
- Invented by Frank Rosenblatt in 1957 in an attempt to understand human memory, learning, and cognitive processes.
- The first neural network model by computation, with a remarkable learning algorithm:
  - If function can be represented by perceptron, the learning algorithm is guaranteed to quickly converge to the hidden function!

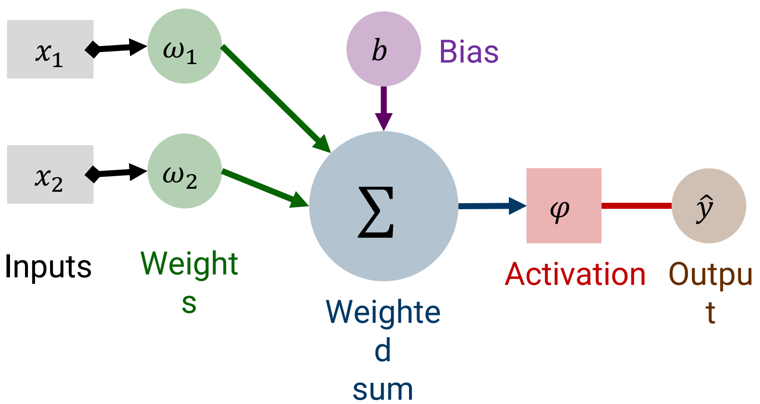
https://www.youtube.com/watch?v=BRXQiBHaO_I

Example

- Parameters to compute AND



a	b	y
0	0	0
0	1	0
1	0	0
1	1	1



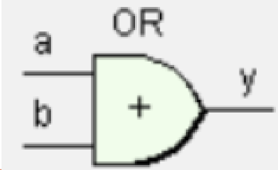
- Input: x_1 and x_2
- Bias: $b = -1$ for AND
- Weights: $w = [1, 1]$

Activation function

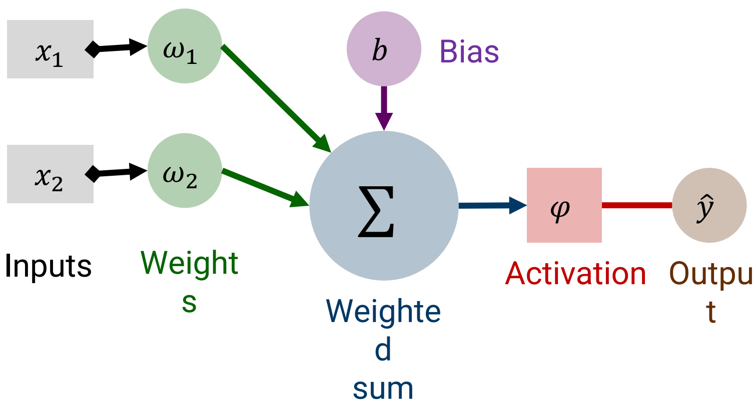
$$\hat{\boldsymbol{y}} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Example

- Parameters to compute OR



a	b	y
0	0	0
0	1	1
1	0	1
1	1	1



Input: x_1 and x_2

Bias: $b = 0$

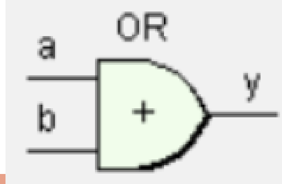
Weights: $w = [1, 1]$

Activation function

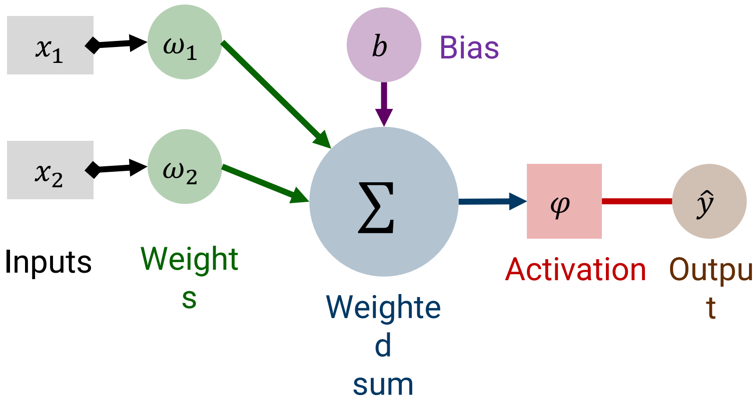
$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Example

- Parameters to compute OR



a	b	y
0	0	0
0	1	1
1	0	1
1	1	1



Input: x_1 and x_2

Bias: $b = -1$

Weights: $w = [1.5, 1.5]$

Activation function

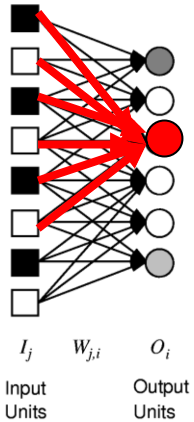
$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Single Layer Feed-forward Neural Networks

- Single-layer neural network (perceptron network)

A network with all the inputs connected directly to the outputs

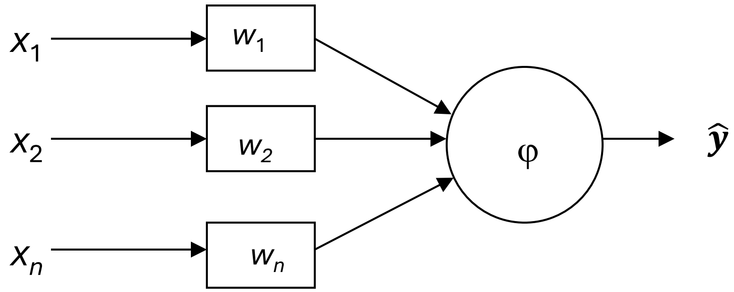
– Output units all operate separately: no shared weights



Perceptron Network

Since each output unit is independent of the others, we can limit our study to single output perceptrons.

Perceptron network

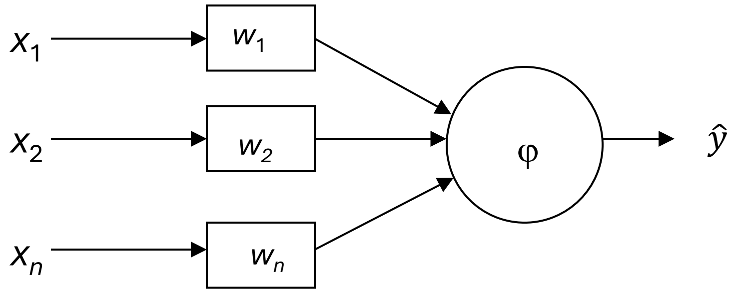


- **Learn weights such that an objective function is maximized**

Activation function, φ

$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Perceptron network



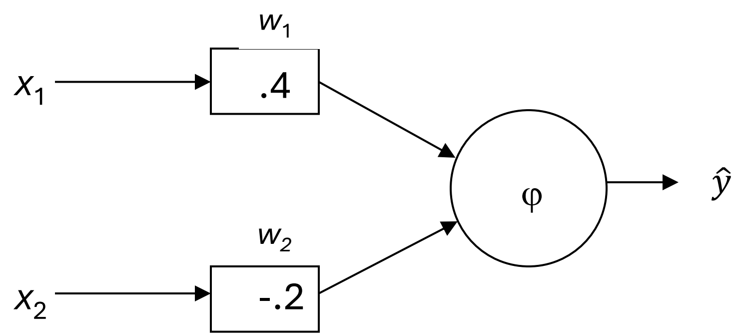
- **Learn weights such that an objective function is maximized**

- What objective function should we use?
- What learning algorithm should we use?

Activation function, φ

$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Perceptron learning algorithm



Training set

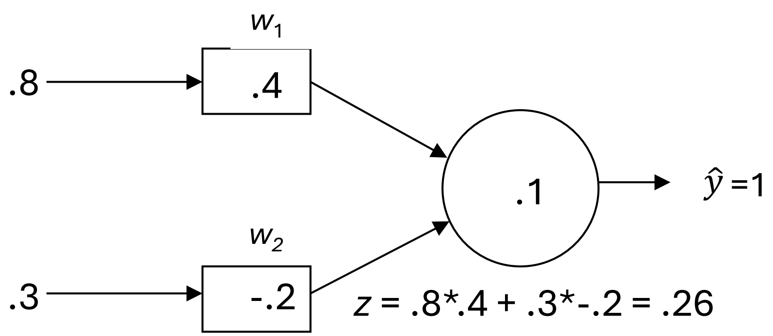
x_1	x_2	t
.8	.3	1
.4	.1	0

Target, Ground Truth (t)

Activation function, φ

$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

First training instance

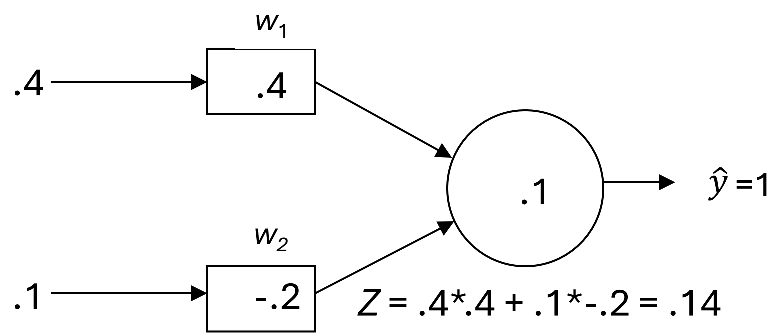


x_1	x_2	t
.8	.3	1
.4	.1	0

Activation function

$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Second training instance

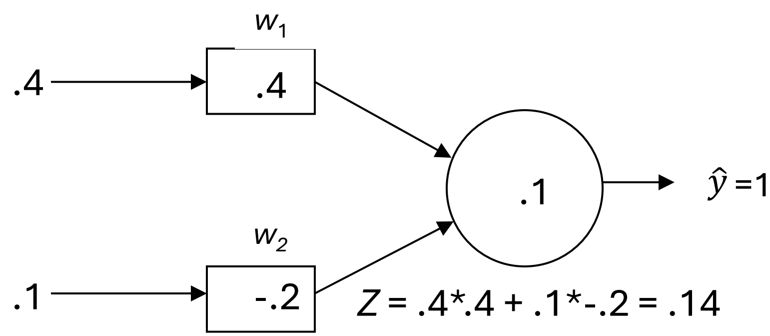


x_1	x_2	t
.8	.3	1
.4	.1	0

Activation function

$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Second training instance



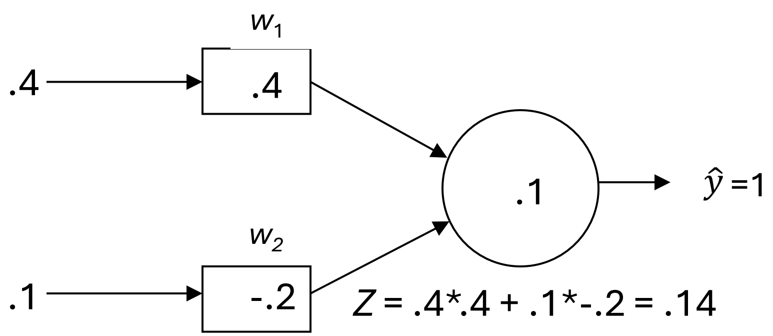
There are an error in de prediction, it is necesary to adjust the weights

x_1	x_2	t
.8	.3	1
.4	.1	0

Activation function

$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Second training instance



There are an error in de prediction, it is necesary to adjust the weights

$$\Delta w_i = (t - \hat{y}) * \alpha * x_i$$

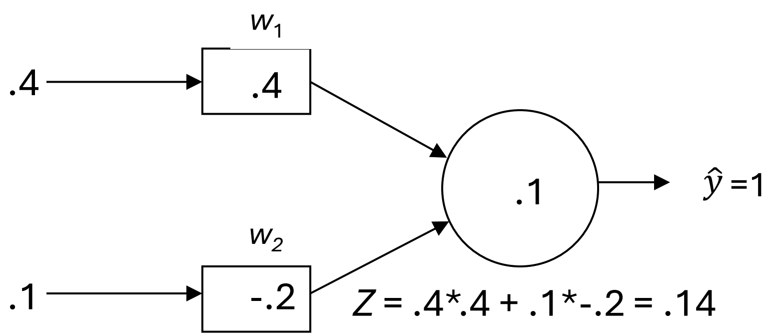
$$w_i = w_i + \Delta w_i$$

x_1	x_2	t
.8	.3	1
.4	.1	0

Activation function

$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Second training instance



There are an error in de prediction, it is necesary to adjust the weights

$$\Delta w_i = (t - \hat{y}) * \alpha * x_i$$

$$w_i = w_i + \Delta w_i$$

- α regulates the *learning rate* of the network

x_1	x_2	t
.8	.3	1
.4	.1	0

Activation function

$$\hat{y} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Perceptron rule learning

$$\Delta w_i = (t - \hat{y}) * \alpha * x_i$$

- Where
  - w_i is the weight from input i to the perceptron node
  - α is the learning rate
  - t is the target for the current instance
  - $\hat{y}$ is the current output
  - x_i is i^{th} input
- Least perturbation principle
  - Only change weights if there is an error
  - small α rather than changing weights sufficient to make current pattern correct
  - Scale by x_i

Perceptron rule learning

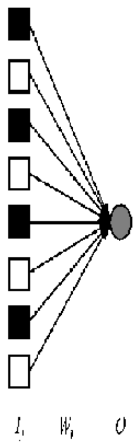
$$\Delta w_i = (t - \hat{y}) * \alpha * x_i$$

- Where
  - w_i is the weight from input i to the perceptron node
  - α is the learning rate
  - t is the target for the current instance
  - $\hat{y}$ is the current output
  - x_i is i^{th} input

- Given a perceptron node with n inputs
  - Iteratively apply a pattern from the training set and apply the perceptron rule
  - **Each iteration through the training set** is an ***epoch***
  - Continue **training until total training set error ceases to improve**

Perceptron rule learning

- **Weight Update**
- → Input X_j (j=1,2,...,n)
- → Single output $\hat{y}$: target output, T.
- Consider some initial weights
- Define example/instance error: $Err = T - \hat{y}$
- Now just move weights in right direction!
- If the error is positive, then we need to increase $\hat{y}$.
- Err >0 → need to increase $\hat{y}$;
- Err <0 → need to decrease $\hat{y}$
- Each input unit j, contributes $W_j X_j$ to total input:
- if X_j is positive, increasing W_j tends to increase $\hat{y}$;
- if X_j is negative, decreasing W_j tends to increase $\hat{y}$;
- So, use:
- $$W_j \leftarrow W_j + \alpha \times X_j \times Err$$
- Perceptron Learning Rule (Rosenblatt 1960)



Perceptron learning: Simple example

- Let's consider an example (adapted from Patrick Wintson book, MIT)
- Framework and notation:
- 0/1 signals
- Input vector: $\vec{X} = < x_0, x_1, x_2 \cdots, x_n >$
- Weight vector: $\vec{W} = < w_0, w_1, w_2 \cdots, w_n >$
- $x_0 = 1$ and $\theta_0 = -w_0$, simulate the threshold.
- $\hat{y}$ is output (0 or 1) (single output).
- Activation function: $z = \sum_{k=0}^{k=n} w_k x_k \quad z > 0 \; then \; \hat{y} = 1 \quad else \quad \hat{y} = 0$
- Learning rate = 1.

Perceptron learning: Simple example

$Err = T - O$

$W_j \leftarrow W_j + \alpha \times X_j \times Err$

- Set of examples, each example is a pair $(\vec{x}_i, y_i)$
- i.e., an input vector and a label y (0 or 1).

This procedure provably converges (polynomial number of steps) if the function is represented by a perceptron (i.e., linearly separable)

- Learning procedure, called the *"error correcting method"*

- Start with all zero weight vector.
- Cycle (repeatedly) through examples and for each example do:
  - If perceptron is 0 while it should be 1,
- add the input vector to the weight vector
  - If perceptron is 1 while it should be 0,
- subtract the input vector to the weight vector
  - Otherwise do nothing.

Intuitively correct, (e.g., if output is 0 but it should be 1, the weights are increased) !

Perceptron learning: Simple example

- Consider learning the logical OR function.
- Our examples are:

Sample	x0	x1	x2	label
1	1	0	0	0
2	1	0	1	1
3	1	1	0	1
4	1	1	1	1

Activation Function $z = \sum_{k=0}^{k=n} w_k x_k \quad z > 0 \ then \ \hat{y} = 1 \quad else \quad \hat{y} = 0$

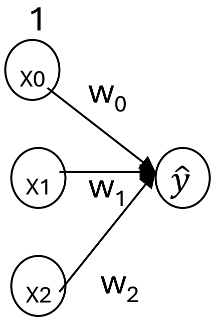
Perceptron learning: Simple example

$$z = \sum_{k=0}^{k=n} w_k x_k \qquad z > 0 \ then\ \hat{y} = 1 \quad else \quad \hat{y} = 0$$

Error correcting method

If perceptron is 0 while it should be 1,
add the input vector to the weight vector
If perceptron is 1 while it should be 0,
subtract the input vector to the weight vector
Otherwise do nothing.

- We’ll use a single perceptron with three inputs.
- We’ll start with all weights 0 W= <0,0,0>



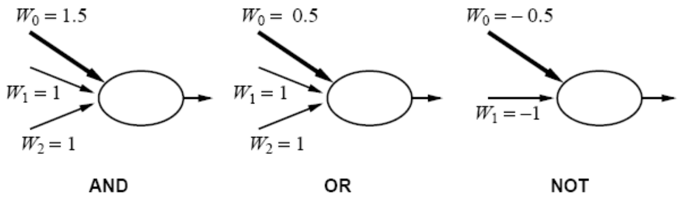
- Example 1 X= < 1 0 0> label=0 W= <0,0,0>
- Perceptron (1×0+ 0×0+ 0×0 =0, z=0) output → 0
- →it classifies it as 0, so correct, do nothing

- Example 2 X=<1 0 1> label=1 W= <0,0,0>
- Perceptron (1×0+ 0×0+ 1×0 = 0) output →0
- →it classifies it as 0, while it should be 1, so we add input to weights
- W = <0,0,0> + <1,0,1>= <1,0,1>

Epoch	x0	x1	x2	Desired Target	w0	w1	w2	Output	Error	New w0	New w1	New w2
1 example 1	1	0	0	0	0	0	0	0	0	0	0	0

Expressiveness of Perceptrons

What hypothesis space can a perceptron represent?



Even more complex Booelan functions such as majority function.

But can it represent any arbitrary Boolean function?

Expressiveness of Perceptrons

A perceptron with sign activation returns 1 iff the weighted sum of its inputs (including the bias) is positive, i.e.,:

$$\sum_{j=0}^{n} W_j x_j > 0 \quad \text{or} \quad \mathbf{W} \cdot \mathbf{x} > 0$$

I.e., iff the input is on one side of the hyperplane it defines.

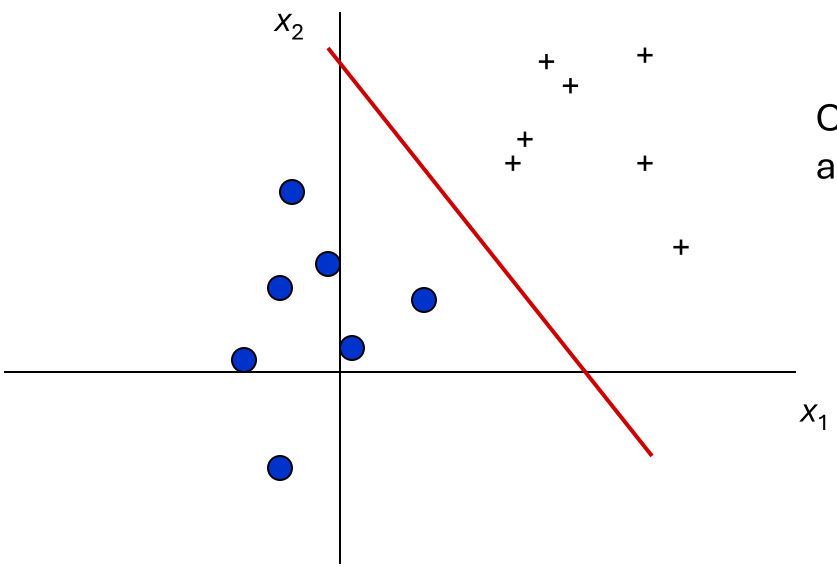
Perceptron → Linear Separator

Linear discriminant function or linear decision surface.

Weights determine slope and bias determines offset.

Linear Separability

Consider example with two inputs, x1, x2:



Can view trained network as defining a "separation line".

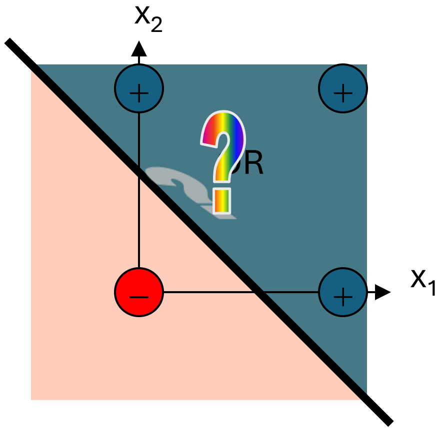
What is its equation?

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

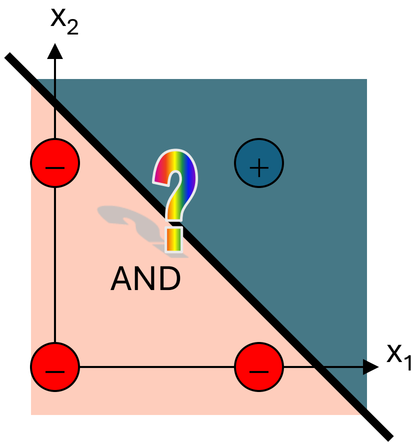
$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{w_0}{w_2}$$

Percepton used for classification

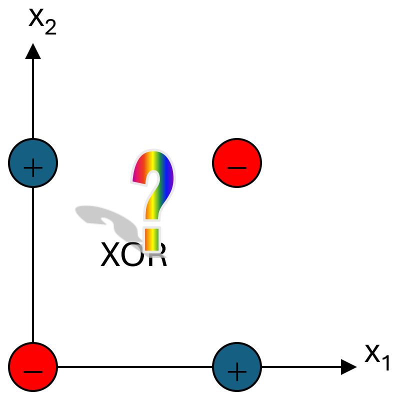
Linear Separability



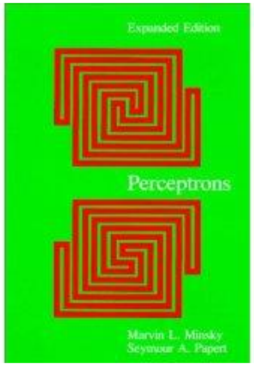
Linear Separability

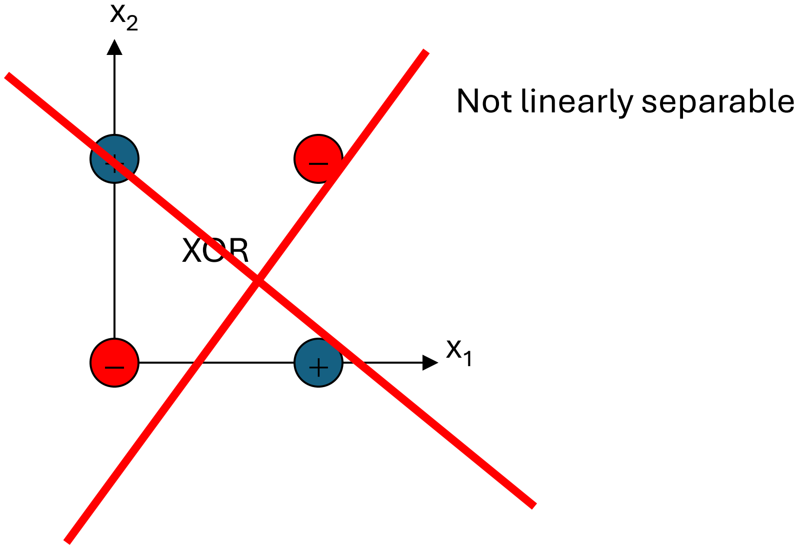


Linear Separability



Linear Separability





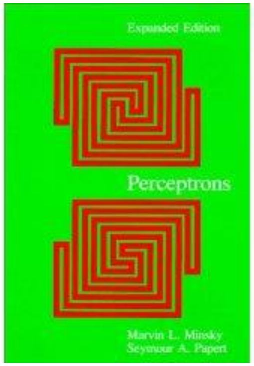
Minsky & Papert (1969)

Bad News: Perceptrons can only represent linearly separable functions.

Convergence of Perceptron Learning Algorithm

Perceptron converges to a consistent function, if…

- … training data linearly separable
- … step size α sufficiently small
- … no "hidden" units

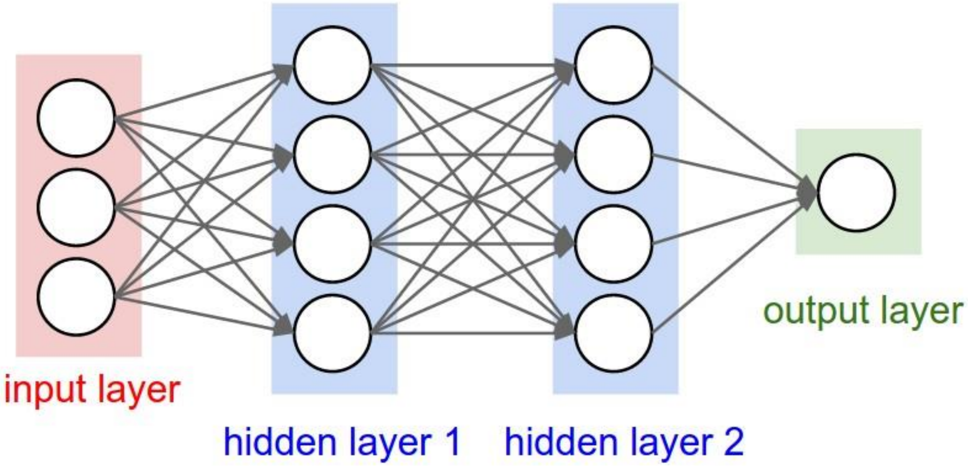


Minsky & Papert (1969)

Good news: Adding hidden layer allows more target functions to be represented.

Multilayer Perceptron (MLP)

- **MPL** is a neural networks contain more than one computational layer
- The additional intermediate layers (between input and output) are *hidden layers* because the computations performed are not visible to the user



Derivation of a learning rule for perceptrons Minimizing Squared Errors

- Threshold perceptrons have some advantages , in particular
- → Simple learning algorithm that fits a threshold perceptron to any
- linearly separable training set.

- Key idea: Learn by adjusting weights to reduce error on training set.
- → update weights repeatedly (epochs) for each example.

- We'll use:
- →Sum of squared errors (e.g., used in linear regression), classical error measure
- →Learning is an optimization search problem in weight space.

Derivation of a learning rule for perceptrons Minimizing Squared Errors

- Let S = {($\mathbf{x}_i$, y_i): i = 1, 2, …, m} be a training set. (Note, **x** is a vector of inputs, and y is the vector of the true outputs.)

- Let $h_{\mathbf{w}}$ be the perceptron classifier represented by the weight vector **w**.

- Definition:

$$E(\mathbf{x}) = Squared\ Error(\mathbf{x}) = \frac{1}{2}(y - h_{\mathbf{w}}(\mathbf{x}))^2$$

Derivation of a learning rule for perceptrons Minimizing Squared Errors

- The squared error for a single training example with input **x** and true output y is:

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 ,$$

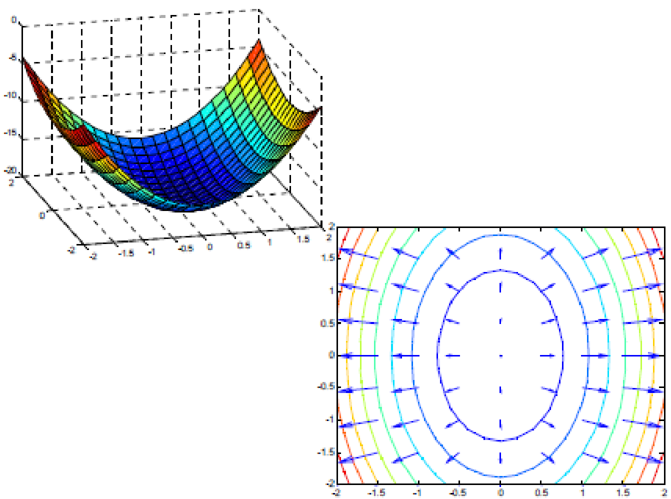
- Where $h_{\mathbf{w}}(\mathbf{x})$ is the output of the perceptron on the example and y is the true output value.
- We can use the gradient descent to reduce the squared error by calculating the partial derivatives of E with respect to each weight.

Gradient descent

- Method to find local optima of differentiable a function f
  - Intuition: **gradient tells us direction of greatest increase**, negative gradient gives us direction of greatest decrease
    - Take steps in directions that reduce the function value

$$\nabla f(\bar{x}_0) = \left(\frac{\partial f(\bar{x}_0)}{\partial x_1}, \frac{\partial f(\bar{x}_0)}{\partial x_2}, \cdots, \frac{\partial f(\bar{x}_0)}{\partial x_n} \right)$$

Gradient of f

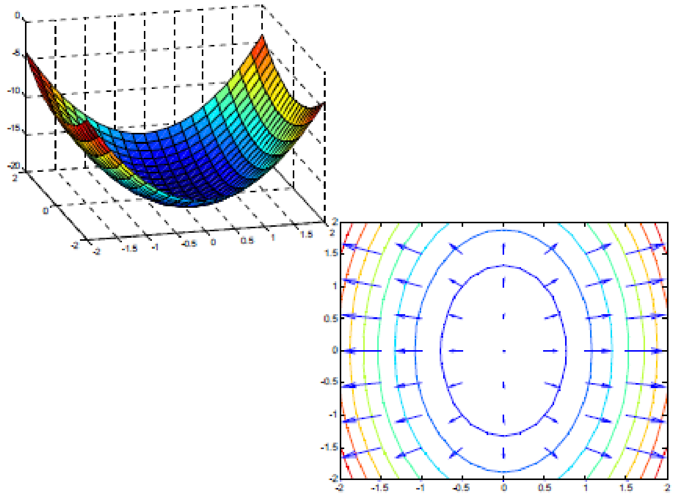


Gradient descent

- Method to find local optima of differentiable a function f
  - Intuition: **gradient tells us direction of greatest increase**, negative gradient gives us direction of greatest decrease
    - Take steps in directions that reduce the function value

$$\nabla f(\bar{x}_0) = \left(\frac{\partial f(\bar{x}_0)}{\partial x_1}, \frac{\partial f(\bar{x}_0)}{\partial x_2}, \dots, \frac{\partial f(\bar{x}_0)}{\partial x_n} \right)$$

Gradient of f



- Definition of derivative guarantees that if we take a small enough step in the direction of the negative gradient, the function will decrease in value
  - How small is small enough?

Gradient descent algorithm

- Pick an initial point x_0
- Iterate until convergence

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

where α_t is the t^{th} step size (sometimes called learning rate)

Gradient descent algorithm

- Pick an initial point x_0
- Iterate until convergence

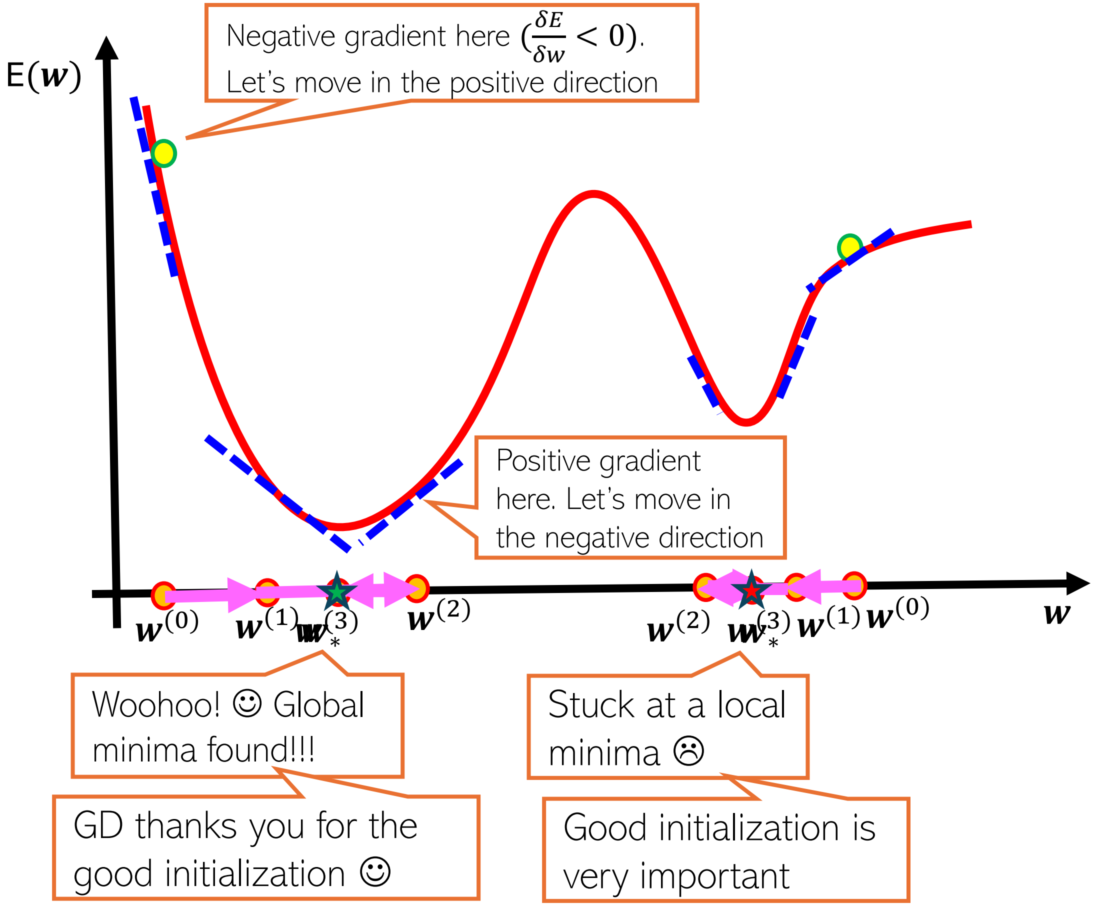
$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

where α_t is the t^{th} step size (sometimes called learning rate)

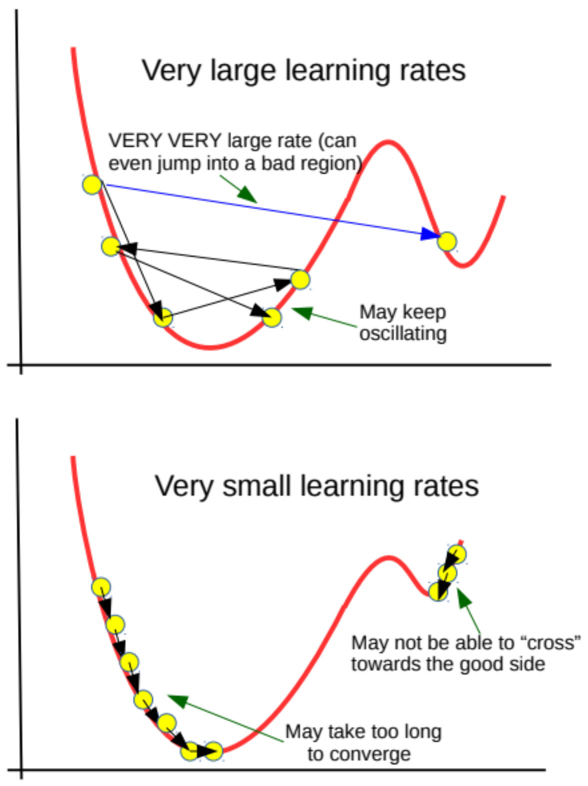
Possible Stopping Criteria: iterate until
$\|\nabla f(x_t)\| \leq \epsilon$ for some $\epsilon > 0$

Gradient descent: An Illustration

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{w}}(\mathbf{x}))^2\,,$$



Learning rate is very important



Slide credit: CS771: Intro to ML

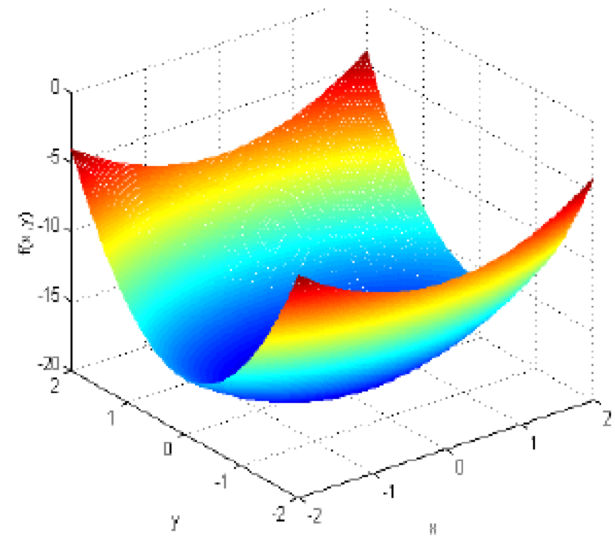
Gradient descent: Example

- Use gradient descent to minimize the function
  - $f(x) = x^2$
  - $\frac{\delta f}{\delta x} = 2x$
  - Learning rate, α=0.8
  - Initial value X_0= -0.4

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

Gradient descent: Exercise

- Use gradient descent to minimize the function
  - $f(x, y)=20 + 3x^2 + y^2$
  - $\frac{\delta f}{\delta x} = ?$
  - $\frac{\delta f}{\delta y} = ?$
  - Learning rate, α=0.25
  - Initial value X_0= -1.7, y_0= 1.7

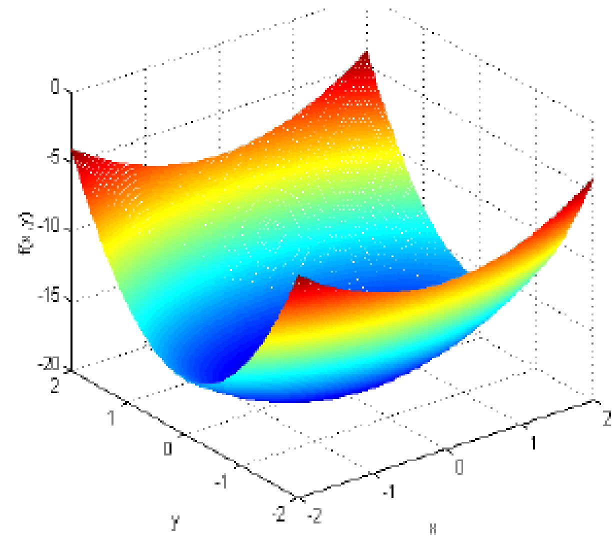


$$x_{t+1} = x_t - \alpha \nabla f(x_t, y_t)$$

$$y_{t+1} = y_t - \alpha \nabla f(x_t, y_t)$$

Gradient descent: Exercise

- Use gradient descent to minimize the function
  - $f(x, y) = 20 + 3x^2 + y^2$
  - $\frac{\delta f}{\delta x} = 6x$
  - $\frac{\delta f}{\delta y} = 2y$
  - Learning rate, $\alpha = 0.25$
  - Initial value $X_0 = -1.7$, $y_0 = 1.7$



The minimum is at (0,0)

$$x_{t+1} = x_t - \alpha \nabla f(x_t, y_t)$$

$$y_{t+1} = y_t - \alpha \nabla f(x_t, y_t)$$

Derivation of a learning rule for perceptrons Minimizing Squared Errors

- The squared error for a single training example with input **x** and true output y is:

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 ,$$

$$\frac{\partial E}{\partial W_j} = -Err \times g'(in) \times x_j$$

- **Gradient descent algorithm** → we want to reduce , *E, f*or each weight w_i , change weight in direction of steepest descent:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

α learning rate

$W_j \leftarrow W_j + \alpha \times X_j \times Err$

- Intuitively:

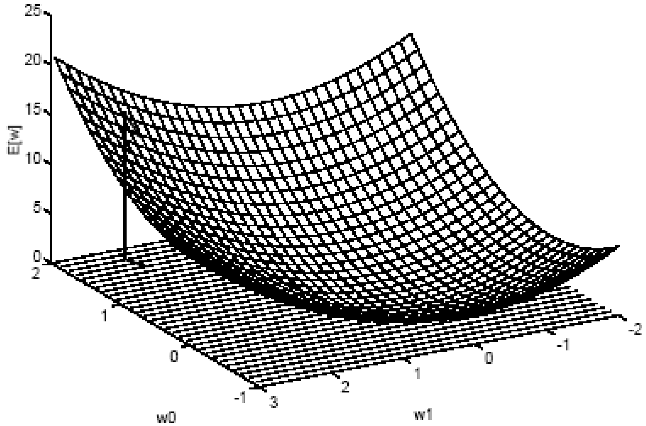
  Err = y – h_W(x) positive

  output is too small → weights are increased for positive inputs and decreased for negative inputs.

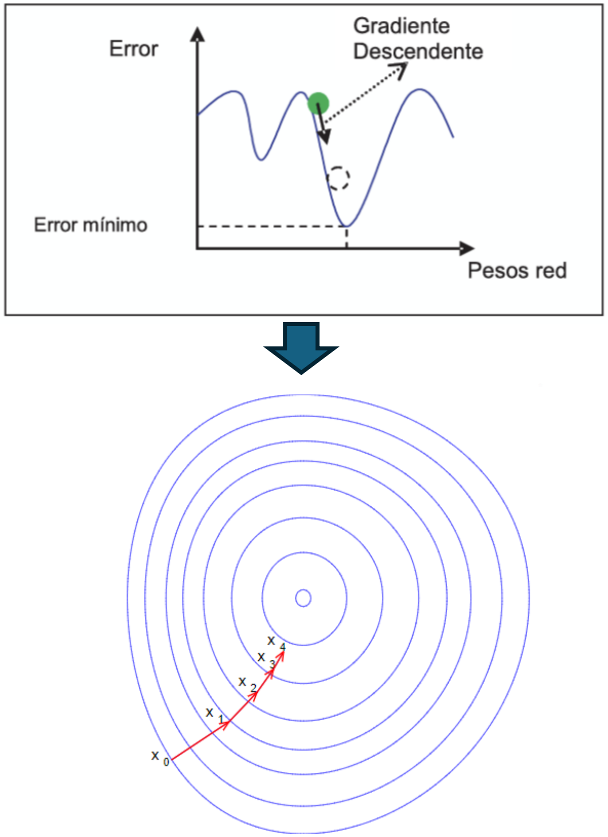
  Err = y – h_W(x) negative → opposite

Gradient descent in weight space

$$E(\mathbf{x}) = Squared\ Error(\mathbf{x}) = \frac{1}{2}(y - h_{\mathbf{w}}(\mathbf{x}))^2$$



From T. M. Mitchell, *Machine Learning*



Perceptron learning: Intuition

- Rule is intuitively correct!
- Greedy Search:
- Gradient descent through weight space!
- Surprising proof of convergence:
- Weight space has no local minima!
- With enough examples, it will find the target function!
- (provide α not too large)

Perceptron learning: Gradient descent learning algorithm

$W_j \leftarrow W_j + \alpha \times X_j \times Err$

Perceptron learning rule:

1. Start with random weights, **w** = (w_1, w_2, … , w_n).
2. Select a training example (**x**,y) ∈ S.
3. Run the perceptron with input **x** and weights **w** to obtain g
4. Let α be the training rate (a user-set parameter).

$$\forall w_i,\ w_i \leftarrow w_i + \Delta w_i,$$

where

$$\Delta w_i = \alpha(y - g(in))g'(in)x_i$$

5. Go to 2.

Epoch → cycle through the examples

Epochs are repeated until some stopping criterion is reached—typically, that the weight changes have become very small.

The stochastic gradient method selects examples randomly from the training set rather than cycling through them.

Perceptron learning: Gradient descent learning algorithm

function PERCEPTRON-LEARNING(*examples*, *network*) **returns** a perceptron hypothesis
inputs: *examples*, a set of examples, each with input $\mathbf{x} = x_1, \ldots, x_n$ and output y
network, a perceptron with weights $W_j,\ j = 0 \ldots n$, and activation function g

repeat
for each e **in** *examples* **do**
$in \leftarrow \sum_{j=0}^{n} W_j\, x_j[e]$
$Err \leftarrow y[e] - g(in)$
$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$
until some stopping criterion is satisfied
return NEURAL-NET-HYPOTHESIS(*network*)

Figure 20.21 The gradient descent learning algorithm for perceptrons, assuming a differentiable activation function g. For threshold perceptrons, the factor $g'(in)$ is omitted from the weight update. NEURAL-NET-HYPOTHESIS returns a hypothesis that computes the network output for any given example.

Readings

Chapra, Steven C. & Canale, Raymond P. Métodos Numéricos para Ingenieros,

- Capítulo 14: Optimización multidimensional no restringida

Martín del Brío, B. & Sanz Molina, A. Redes neuronales y sistemas borrosos,

- Capítulo 2.3: Perceptron simple

Credits

Slides adapt from Carla P. Gomes “CS 4700: Foundations of Artificial Intelligence”