

# Deep Learning

## Perceptron

Tiago Vieira

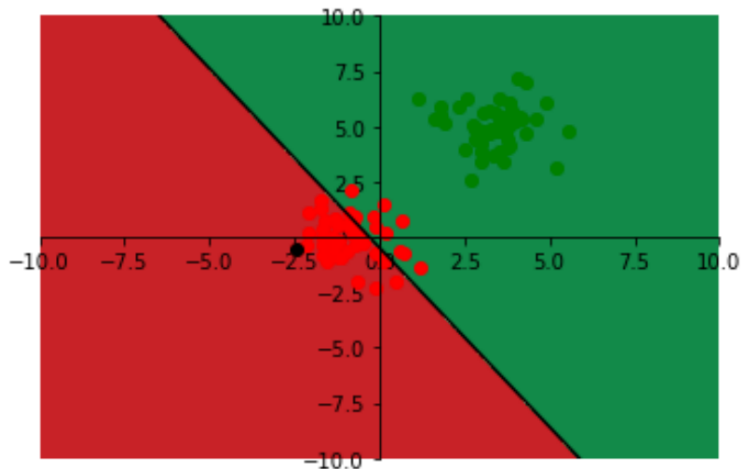
Institute of Computing  
Universidade Federal de Alagoas

February 5, 2023

# Motivation – ML w/ Perceptron



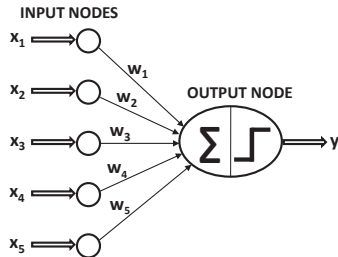
## Motivation



# Binary Classification and Linear Regression Problems

- ▶ In the binary classification problem, each training pair  $(\bar{X}, y)$  contains feature variables  $\bar{X} = (x_1, \dots, x_d)$ , and label  $y$  drawn from  $\{-1, +1\}$ .
  - Example: Feature variables might be frequencies of words in an email, and the class variable might be an indicator of spam.
  - Given labeled emails, recognize incoming spam.
- ▶ In linear regression, the *dependent* variable  $y$  is real-valued.
  - Feature variables are frequencies of words in a Web page, and the dependent variable is a prediction of the number of accesses in a fixed period.
- ▶ Perceptron is designed for the binary setting.

# The Perceptron: Earliest Historical Architecture



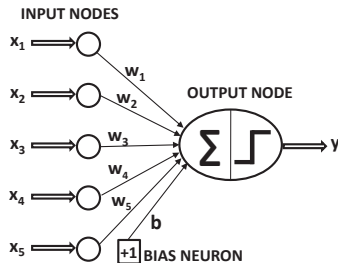
- ▶ The  $d$  nodes in the input layer only transmit the  $d$  features  $\overline{X} = [x_1 \dots x_d]$  without performing any computation.
- ▶ Output node multiplies input with weights  $\overline{W} = [w_1 \dots w_d]$  on incoming edges, aggregates them, and applies *sign activation*:

$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X}\} = \text{sign}\left\{\sum_{j=1}^d w_j x_j\right\}$$

## What is the Perceptron Doing?

- ▶ Tries to find a *linear separator*  $\overline{W} \cdot \overline{X} = 0$  between the two classes.
- ▶ Ideally, all positive instances ( $y = 1$ ) should be on the side of the separator satisfying  $\overline{W} \cdot \overline{X} > 0$ .
- ▶ All negative instances ( $y = -1$ ) should be on the side of the separator satisfying  $\overline{W} \cdot \overline{X} < 0$ .

## Bias Neurons



- In many settings (e.g., skewed class distribution) we need an invariant part of the prediction with bias variable  $b$ :

$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X} + b\} = \text{sign}\left\{\sum_{j=1}^d w_j x_j + b\right\} = \text{sign}\left\{\sum_{j=1}^{d+1} w_j x_j\right\}$$

- On setting  $w_{d+1} = b$  and  $x_{d+1}$  as the input from the bias neuron, it makes little difference to learning procedures  $\Rightarrow$  Often implicit in architectural diagrams

## Training a Perceptron

- ▶ Go through the input-output pairs  $(\bar{X}, y)$  one by one and make updates, if predicted value  $\hat{y}$  is different from observed value  $y \Rightarrow$  Biological readjustment of synaptic weights.

$$\bar{W} \Leftarrow \bar{W} + \alpha \underbrace{(y - \hat{y})}_{\text{Error}} \bar{X}$$

$$\bar{W} \Leftarrow \bar{W} + (2\alpha)y\bar{X} \text{ [For misclassified instances } y - \hat{y} = 2y]$$

- ▶ Parameter  $\alpha$  is the learning rate  $\Rightarrow$  Turns out to be irrelevant in the special case of the perceptron
- ▶ One cycle through the entire training data set is referred to as an *epoch*  $\Rightarrow$  Multiple epochs required
- ▶ How did we derive these updates?



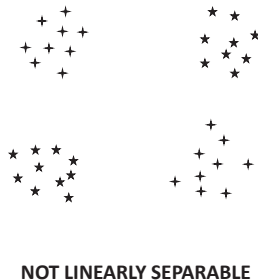
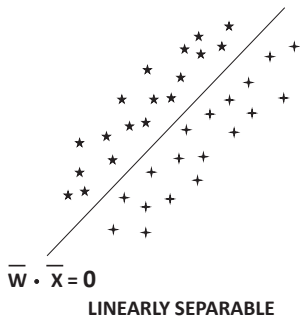
# What Objective Function is the Perceptron Optimizing?

- ▶ At the time, the perceptron was proposed, the notion of loss function was not popular  $\Rightarrow$  Updates were heuristic
- ▶ Perceptron optimizes the perceptron criterion for  $i$ th training instance:

$$L_i = \max\{-y_i(\overline{W} \cdot \overline{X}_i), 0\}$$

- Loss function tells us how far we are from a desired solution  $\Rightarrow$  Perceptron criterion is 0 when  $\overline{W} \cdot \overline{X}_i$  has same sign as  $y_i$ .
- ▶ Perceptron updates use *stochastic gradient descent* to optimize the loss function and reach the desired outcome.
  - Updates are equivalent to  $\overline{W} \leftarrow \overline{W} - \alpha \left( \frac{\partial L_i}{\partial w_1} \dots \frac{\partial L_i}{\partial w_d} \right)$

# Where does the Perceptron Fail?



- ▶ The perceptron fails at similar problems as a linear SVM
  - **Classical solution:** Feature engineering with Radial Basis Function network  $\Rightarrow$  Similar to kernel SVM and good for noisy data
  - **Deep learning solution:** Multilayer networks with nonlinear activations  $\Rightarrow$  Good for data with a lot of structure

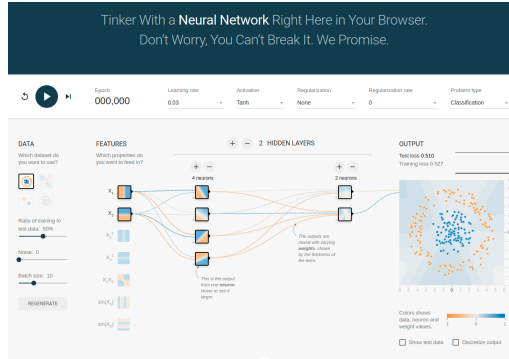
# Historical Origins

- ▶ The first model of a computational unit was the *perceptron* (1958).
  - Was roughly inspired by the biological model of a neuron.
  - Was implemented using a large piece of hardware.
  - Generated great excitement but failed to live up to inflated expectations.
- ▶ Was not any more powerful than a simple linear model that can be implemented in a few lines of code today.

# Perceptron Tutorial

Perceptron tutorial (`simple_perceptron.py`).

# Tensorflow Playground<sup>3</sup>

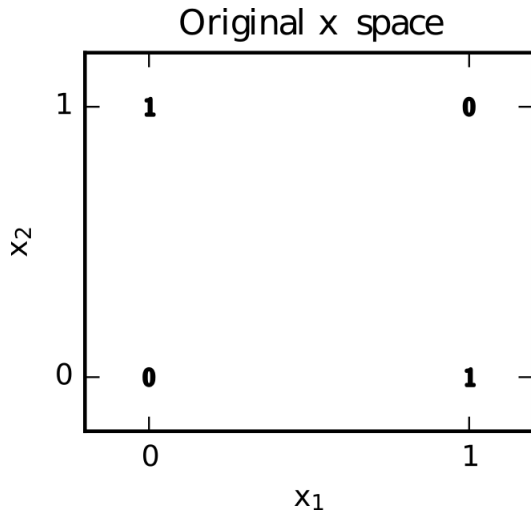


<sup>3</sup><https://playground.tensorflow.org/>

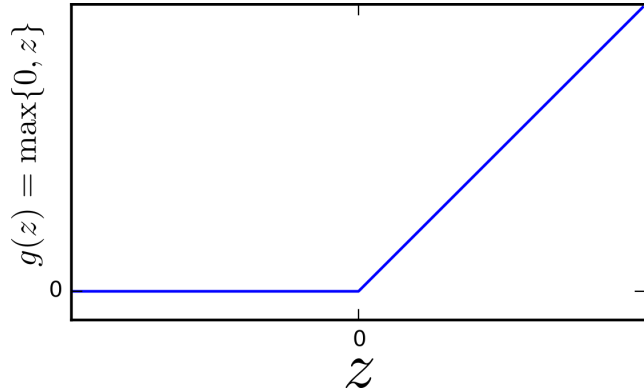
# The XOR Problem

- ▶ “*Perceptrons*” by Marvin Minsky and Seymour Papert (1969).
- ▶ Perceptrons cannot solve the XOR problem.
- ▶ Significant decline in interest and funding of neural network research.

## The XOR Problem

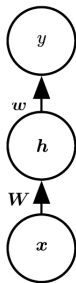
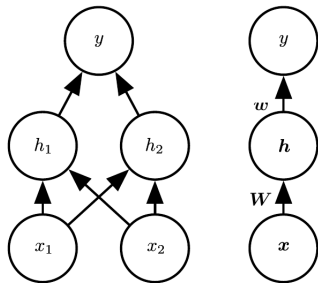


## Rectified Linear Activation



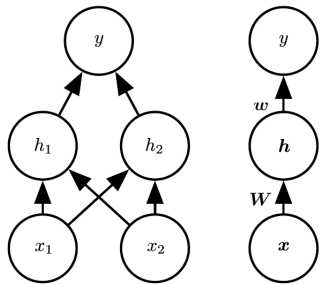


## Network Diagrams



$$\mathbf{h} = \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$
$$f(\mathbf{x}; (\mathbf{W}; \mathbf{c}); (\mathbf{w}, b)) = \mathbf{w}^T \mathbf{h} + b$$

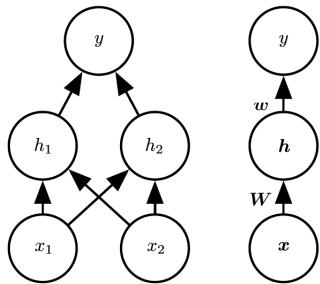
## Solving XOR



$$\mathbf{h} = \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$
$$f(\mathbf{x}; (\mathbf{W}; \mathbf{c}); (\mathbf{w}, b)) = \mathbf{w}^T \mathbf{h} + b$$

$$X = [\mathbf{x}]_{i=1}^4 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$
$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$
$$\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$
$$\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$
$$b = 0$$

## Solving XOR



$$\mathbf{h} = \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$
$$f(\mathbf{x}; (\mathbf{W}; \mathbf{c}); (\mathbf{w}, b)) = \mathbf{w}^T \mathbf{h} + b$$

$$H = \max(0, \mathbf{W}^T X + \mathbf{c})$$

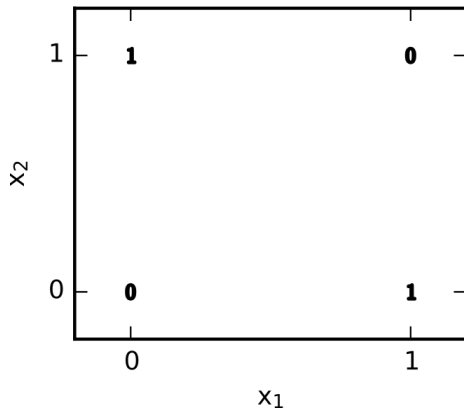
$$H = \max\left(0, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

$$H = \max\left(0, \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

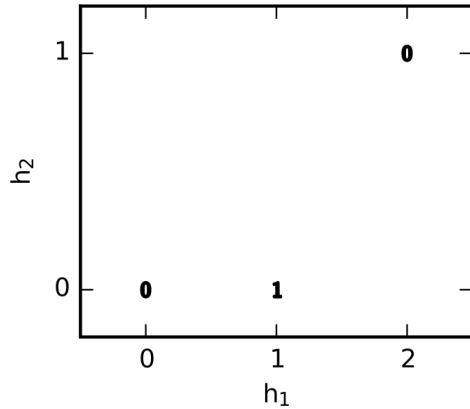
$$H = \max\left(0, \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}\right)$$

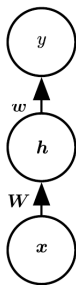
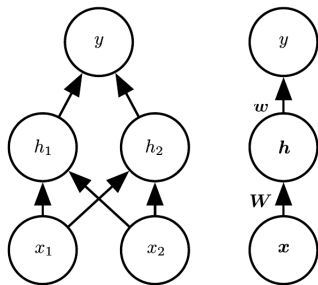
$$H = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Original x space



Learned h space





$$\mathbf{h} = \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$

$$f(\mathbf{x}; (\mathbf{W}; \mathbf{c}); (\mathbf{w}, b)) = \mathbf{w}^T \mathbf{h} + b$$

$$Y = \max(0, \mathbf{w}^T H + \mathbf{b})$$

$$Y = \max \left( \begin{bmatrix} 1 & -2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right)$$

$$Y = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$$

Thank you!  
tvieira@ic.ufal.br