Redes Neuronales

U2 Red Neuronal Simple

Héctor Maravillo

Section 1

Hebb Net

Biases and Thresholds

A **bias** acts exactly as a weight on a connection from a unit whose activation (input) is always 1.

Increasing the bias increases the net input to the unit. If a bias is included, the basic activation function is typically taken to be

$$f(net) = \begin{cases} 1 & \text{if net } \geq 0 \\ -1 & \text{if net } < 0 \end{cases}$$

where

$$net = b + \sum_i x_i w_i$$

Biases and Thresholds

Some authors do not uses a bias weight, but instead use a **fixed threshold** θ for the activation function. In that case,

$$f(net) = \begin{cases} 1 & \text{if net } \geq \theta \\ -1 & \text{if net } < \theta \end{cases}$$

where

$$net = \sum_i x_i w_i$$

Classification

In **classification**, the intent is to **train** the net, i.e., **adaptively determine its weights**, so that it will respond with the desired classification when presented with an input patter that it was trained on or when presented with one that is sufficiently similar to one of the training patterns.

For a particular output unit, the desired response is a **yes** if the input pattern is a member of its class and a **no** if it is not. For **bipolar signals**, a **yes** response is represented by an output signal of 1, a **no** by an output signal of -1.

Classification

Since we want one of two responses, the activation (or output) function is taken to be a **step function**.

The value of the function is 1 if the net input is positive and -1 if the net input is negative.

Since the net input to the output unit is

$$y_{in} = b + \sum_i x_i w_i$$

it's easy to see that the boundary between the region where $y_{in} > 0$ and the region where $y_{in} < 0$, which we call the **decision boundary**, is determined by the **hyperplane**

$$H = \{x \in \mathbb{R}^n \ : \ b + w^T x = 0\}$$

Linear Separability

If there are weights (and a bias) so that all of the training input vectors for which the correct response is +1 lie on one side of the decision boundary and all of the training input vectors for which the correct response is -1 lie on the other side of the decision boundary, we say that the problem is **linearly separable**.

Minsky and Papert showed that **a single-layer net can learn only linearly separable problems**.[1]

[1] Furthermore, it is easy to extend this result to show that **multilayer nets with linear activation functions** are no more powerful than a single layer nets (since the composition of linear functions is linear).

Data Representation

The form of the data may change the problem from one that can be solved by a simple neural net to one that cannot.

Binary representation is also not as good as **bipolar** if we want the net to generalize (i.e., respond to input data similar but not identical to, training data).

Using **bipolar input**, missing data can be distinguished from mistaken data. Missing values can be represented by 0 and mistakes by reversing the input value from +1 to -1, or vice versa.

Hebb net

The earliest and simplest learning rule for a neural net is generally known as the **Hebb rule**.

Hebb proposed that learning occurs by modification of the synapse strengths (weights) in a manner such that *if two interconnected neurons are both "on" at the same time, then the weight between those neurons should be increased*.[2]

We shall refer to a single layer (feedforward) neural net trained using the (extended) Hebb rule or correlational learning as a **Hebb net**.

[2] The original statement only talks about neurons firing at the same time. However, a stronger form of learning occurs if we also increase the weights if both neurons are "off" at the same time.

Hebb rule

Algorithm:

- Step 0. Initialize all weights and the bias

$$w_i(0) = 0, \quad i = 1, \ldots, n,$$

$$b(0) = 0$$

- Step 1. For each input training vector and target output pair (s, t), do steps 2-4.
- Step 2. Set activation for input units:

$$x_i = s_i, \qquad i = 1, \ldots, n$$

- Step 3. Set activation for output unit:

$$y = t$$

- Step 4.
  - Adjust the weights for

$$w_i(new) = w_i(old) + x_i y, \qquad i = 1, \ldots, n$$

  - Adjust the bias:

$$b(new) = b(old) + y$$

Hebb rule

Note that the bias is adjusted exactly like a weight from a *unit* whose output signal is always 1.

The weight update can also be expressed in vector form as

$$w(new) = w(old) + xy$$

The Hebb rule is extremely simple but limited (even for linearly separable problems).

Example: Logic functions

A **Hebb net** for the **AND** function: binary inputs and targets

INPUT		**TARGET**
$(s_1$	$s_2)$	t
(1	1)	1
(1	0)	0
(0	1)	0
(0	0)	0

Table: AND function

For each training input: target, the weight change is the product of the input vector and the target value, i.e.,

$$\triangle w_1 = x_1 t$$

$$\triangle w_2 = x_2 t$$

$$\triangle b = t$$

Example: Logic functions

The new weights are the sum of the previous weights and the weight change. Only one iteration through the training vector is required.

The weight updates for the first input are as follows:

$$\triangle w_1 = 1 \cdot 1$$

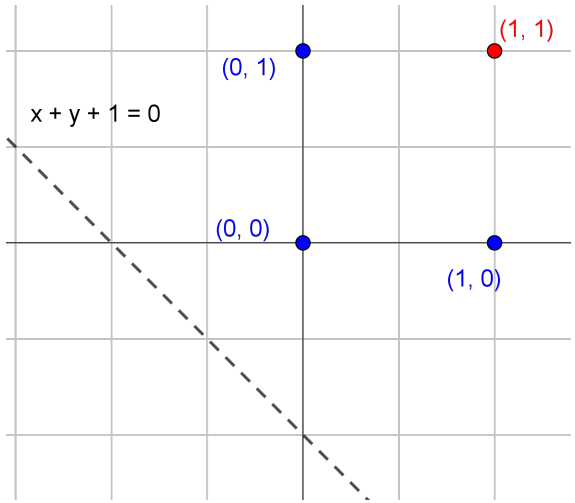
$$\triangle w_2 = 1 \cdot 1$$

$$\triangle 1 = 1$$

Example: Logic functions

So, the separating line becomes:

$$\{(x_1, x_2) \in \mathbb{R}^2 : \ x_1 + x_2 + 1 = 0\}$$



The second, third and fourth training inputs do not change the weights, because the target value is 0, no learning ocurrs. Thus, using **binary target** values prevent the net from learning any pattern for which the target is *off*.

Example: Bipolar inputs and targets

Consider the bipolar representation for inputs and targets

INPUT $(s_1 \ \ s_2)$	**TARGET** t
$(1 \ \ 1)$	1
$(1 \ \ -1)$	-1
$(-1 \ \ 1)$	-1
$(-1 \ \ -1)$	-1

Table: AND function (bipolar representation)

Example: Bipolar inputs and targets

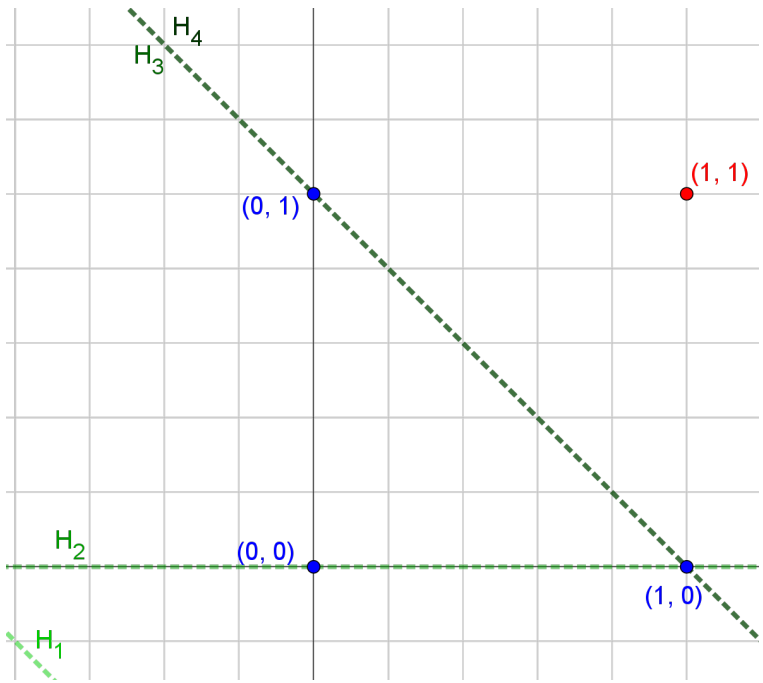
The weight updates in each iteration are as follows

$$w_1(1) = 1, \quad w_2(1) = 1, \quad b(1) = 1$$

$$w_1(2) = 0, \quad w_2(2) = 2, \quad b(2) = 0$$

$$w_1(3) = 1, \quad w_2(3) = 1, \quad b(3) = -1$$

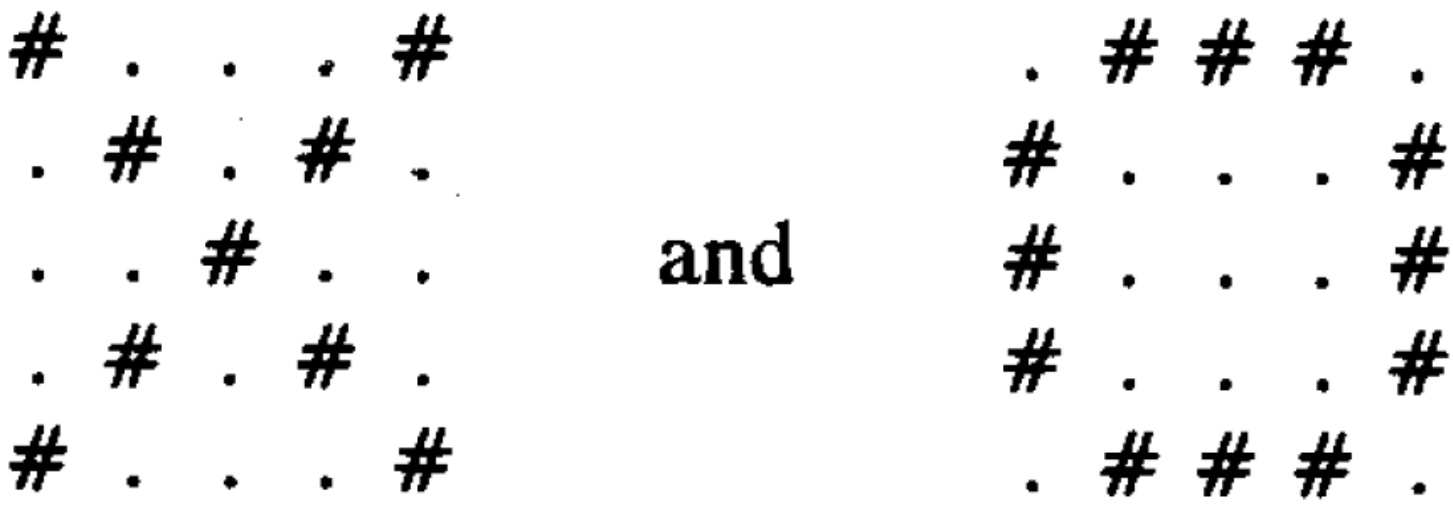
$$w_1(4) = 2, \quad w_2(4) = 2, \quad b(4) = -2$$



Example: 2D patterns

A Hebb net to classify two-dimensional input patterns (representing letters.

A simple example of using the Hebb rule for character recognition involves training the net to distinguish between the pattern "X" and the pattern "O". The patterns can be represented as



Pattern 1 Pattern 2

Example: 2D patterns

To treat this example as a pattern classification problem with one output class we will designate that class "X" and take the patter "O" to be an example of the output that is not "X".

The first thing we need to do is to convert the patterns to input vectors. That is easy do by assigning each # the value 1 and each "." the value -1.

To convert from the two-dimensional pattern to an input vector, we simply concatenate the rows.

References

- Fausset, L. *Fundamentals of Neural Networks. Architectures, Algorithms and Applications*, Pearson Education, 1994.