Redes Neuronales

U2 Red Neuronal Simple

Héctor Maravillo

Section 1

The Perceptron

Perceptron

The **Perceptron**, created by **Rosenblatt in 1957**, is the simplest configuration of an artificial neural network ever created, whose purpose was to implement a computational model based on the retina.

The **simplicity** of the Perceptron network is due to its condition of being constituted by just **one neural layer**.

Perceptrons had perhaps the most far-reaching impact of any of the early neural nets. The **Perceptron learning rule** is a more powerful learning rule than the Hebb rule: Under suitable assumptions, its iterative learning procedure can be proved **to converge** to the correct weights.

Geometric Patterns

One application of the Perceptron was to identify geometric patterns.

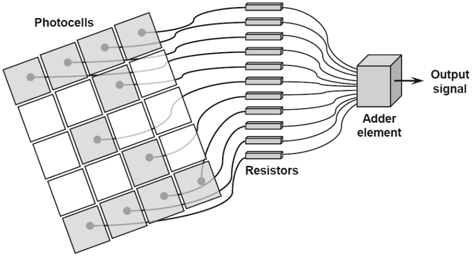

Figure: Illustrative model of the Perceptron for pattern recognition.[1]

[1] In the original Perceptron signals from photocells, used to map geometric patterns, were pondered by tunable resistors, which could be adjusted during the training process After that, an additive element would combine all pondered signals. In this way, the Perceptron could **recognize** different geometric patterns, such as numbers and letters.

Perceptron algorithm

This algorithm is suitable for either **binary** or **bipolar input** vectors (*n*-tuples), with a **bipolar target**, fixed θ, and adjustable bias.

The algorithm is **not particularly sensitive** to the **initial values** of the weights or the value of the learning rate α.

Perceptron algorithm

The **activation function** for each associator unit was the **binary step function** with an arbitrary, but fixed threshold.

The output of the perceptron is $y = f(y_{in})$, where the **activation function** is

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \leq y_{in} \leq \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

Perception algorithm

The weights from the associator units to the response (or output) unit were **adjusted** by the perceptron learning rule.

For **each training input**, the net would **calculate** the **response** of the output unit. Then the net would determine whether an **error occurred** for this patter (by comparing the calculated output with the target value).

The net **did not distinguish** between an error in which the calculated output was zero and the target -1, as opposed to an error in which the calculated output was $+1$ and the target -1. In either of these cases, the sign of the error denotes that the weights should be changed in the direction indicated by the target value.

Perception algorithm

If an **error occurred** for a particular training input patter, the weights would be changed according to the formula (**Perceptron learning rule**):

$$w_i(new) = w_i(old) + \alpha t x_i$$

where the target value t is $+1$ or -1 and α is the learning rate.

If an error did not occur, the weights would not be changed.

Training would continue until no error occurred.

- **Step 0**.
  Initialize weights and bias (for simplicity, set weights and bias to zero).
  Set learning rate $\alpha \in (0, 1]$ (for simplicity, α can be set to 1).
- **Step 1.** While stopping condition is false, do Steps 2-6.
  - **Step 2.** For each training pair $(s : t)$ do Steps 3-5
    - **Step 3.** Set activations of input units:
      $$x_i = s_i$$
    - **Step 4.** Compute response of output unit
      $$y_{in} = b + \sum_i x_i w_i$$
    - **Step 5.** Update weights and bias if an error occurred for this pattern.
      If $y \neq t$:
      $$w_i(new) = w_i(old) + \alpha t x_i \quad (1)$$
      $$b(new) = b(old) + \alpha t \quad (2)$$
      Else
      $$w_i(new) = w_i(old) \quad (3)$$
      $$b(new) = b(old) \quad (4)$$
  - **Step 6.** Test stopping condition:
    If no weights change in Step 2, stop; else, continue.

Remarks

Note that only weights **connecting active input** units ($x_i \neq 0$) are **updated**.

Also, weights are **updated only** for patterns that do not produce the **correct value** of y.

This means that as more training patterns produce the correct response, less learning occurs.

This is in contrast to the training of the **ADALINE units**, in which learning is based on the difference between y_{in} and t.

Remarks

The **threshold** on the activation function for the response unit is a **fixed**, non-negative value θ.

The form of the activation function for the output unit (response unit) is such that there is an **undecided band** (of fixed width determined by θ) separating the region of positive response from that of negative response.

Several output classes

The perceptron can be extended to the case where the input vectors belongs to one (or more) of **several categories**.

In this type of application, there is n output unit representing each of the categories to which the input vectors may belong.

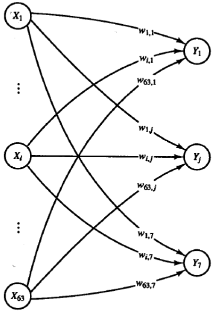

Figure: Perceptron to classify input into seven categories

- **Step 0**.
  Initialize weights and bias (0 or small random values)).
  Set learning rate $\alpha \in (0, 1]$
- **Step 1.** While stopping condition is false, do Steps 2-6.
  - **Step 2.** For each training pair $(s : t)$ do Steps 3-5
    - **Step 3.** Set activation of **each input units** $i = 1, \dots, n$:

      $$x_i = s_i$$

    - **Step 4.** Compute activation of **each output unit** $j = 1, \dots, m$:

      $$y_{in_j} = b_j + \sum_i x_i w_{ij}$$

      $$y_j = \begin{cases} 1 & \text{if } y_{in_j} > \Theta \\ 0 & \text{if } -\Theta \leq y_{in_j} \leq \Theta \\ -1 & \text{if } y_{in_j} < -\Theta \end{cases}$$

    - **Step 5.** Update weights and bias $j = 1, \dots, m$; $i = 1, \dots, n$:
      If $t_j \neq y_j$:

      $$\begin{aligned} w_{ij}(new) &= w_{ij}(old) + \alpha t_j x_i & (5) \\ b_j(new) &= b_j(old) + \alpha t_j & (6) \end{aligned}$$

      Else, biases and weights remain unchanged
  - **Step 6.** Test stopping condition:
    If no weights change in Step 2, stop; else, continue.

References

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