Redes Neuronales

U2 Redes Simples

Héctor Maravillo

Section 1

ADALINE

The ADALINE

The **ADALINE (ADAptive Linear NEuron)** (Widrow & Hoff, 1960) typically uses bipolar $\{1, -1\}$ activations for its **input signals** and its **target output** (although it is not restricted to such values).

The **weights** on the connections from the input units to the ADALINE are **adjustable**.

The **ADALINE** also has a bias, which acts like an **adjustable weight** on a connection from a unit whose activation is always 1.

Delta rule

In general, an ADALINE can be trained using the **delta rule**, also known as the **Least Mean Squares (LMS)** or **Widrow-Hoff rule**.

The rule can also be used for single-layer nets with several output units; an ADALINE is a special case in which there is **only one output unit**.

During training, the activation of the unit is its net input, i.e., the **activation function** is the **identify function**.

The learning rule **minimize** the **mean squared error** between the **activation** and the **target value**.

Threshold function

This **allows** the net to continue **learning on all training patterns**, even after the correct output value is generated (if a **threshold function** is applied) for some patterns.

After training, if the net is being used for pattern classification in which the desired output is either a $+1$ or a -1, a **threshold function** is applied to the net input to obtain the activation.

- If the net input to the ADALINE is greater than or equal to 0, then its activation is set to 1.
- Otherwise it is set to -1.

ADALINE Architecture

An **ADALINE** is a **single unit (neuron)** that receives input from several units.

It also receives input from a *¡¡unit¿¿* whose signal is always +1, in order for the **bias weight** to be trained by the same process (the delta rule) as is used to train the other weights.

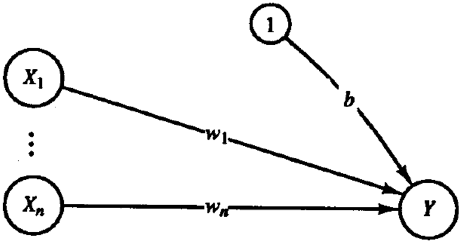

Figure: Architecture of an ADALINE

Algorithm

A **training algorithm** for an ADALINE is a follow:

- **Step 0**. Initialize weights (small random values are usually used). Set learning rate α.
- **Step 1**. While stopping condition is false, do Steps 2-6.
  - **Step 2**. For each bipolar training pair $s : t$, do Steps 3-5.
    - **Step 3**. Set activations of input units, $i = 1, \dots, n$:
$$x_i = s_i$$
    - **Step 4**: Compute net input to output unit:
$$y_{in} = b + \sum_i x_i w_i$$
    - **Step 5**. Update bias and weights, $i = 1, \dots, n$:
$$b(new) = b(old) + \alpha(t - y_{in}) \quad (1)$$
$$w_i(new) = w_i(old) + \alpha(t - y_{in})x_i \quad (2)$$
  - Test for stopping condition:
    - If the largest weight change that occurred in Step 2 is smaller than a specified tolerance, then stop.
    - Otherwise continue.

Learning rate

Setting the **learning rate** to a suitable value requires some care.

According to Hecht-Nielsen, an **upper bound** for its value can be found from the **largest eigenvalue** of the **correlation matrix** R of the input (row) vectors $x(p)$:

$$R = \frac{1}{P}\sum_{p=1}^{P} x(p)^T x(P)$$

namely,

$$\alpha < \text{one-half the largest eigenvalue of } R$$

Learning rate

However, since R does not need to be calculated to compute the weight updates, it is common simply to take a **small value** for α (such as $\alpha = 0.1$) initially.

- If **too large** a value is chosen, the learning process will **not converge**.
- If **too small** a value is chosen, **learning** will be **extremely slow**.

For a single neuron, a practical range for the learning rate α is

$$0.1 \leq n\alpha \leq 1.0$$

where n is the number of input units [Widrow, Winter & Baxter, 1988].

Derivations

The **delta rule** changes the weights of the neural connections so as to **minimize** the **difference** between the net input to the output unit, y_{in}, and the target value t.

The aim is to **minimize the errors** over all training patterns. However, this is accomplished by reducing the error for each pattern, one at a time.

Weight corrections can also be accumulated over a number of training patterns (so-called **batch updating**) if desired.

Derivations

The **delta rule** for adjusting the kth weight (for each pattern) is

$$\triangle w_k = \alpha(t - y_{in})x_k$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is the vector of activation of inputs units,

$$y_{in} = \sum_{i=1}^{n} x_i w_i$$

is the net input to output unit Y, and t is the target output.

Derivations

The **squared error** for a **particular training pattern** is

$$E = (t - y_{in})^2$$

E is a function of all the weights, $w_i, \ i = 1, \ldots, n$.

The gradient of E is the vector consisting of the **partial derivatives** of E with respect to each of the weights.

- The **gradient** gives the **direction** of most rapid **increase** in E.
- The **opposite direction gives** the most rapid **decrease** in the error.

The **error** can be reduced by adjusting the weight w_i in the direction of

$$-\frac{\partial E}{\partial w_k}$$

Derivations

Since $y_{in} = \sum_{i=1}^{n} x_i w_i$

$$
\begin{aligned}
\frac{\partial E}{\partial w_k} &= -2(t - y_{in})\frac{\partial y_{in}}{\partial w_k} \\
&= -2(t - y_{in})x_k
\end{aligned}
$$

Thus, the local error will be **reduced** most rapidly (for a given learning rate) by adjusting the weights according to the **delta rule**

$$\triangle w_k = \alpha(t - y_{in})x_k$$

Delta rule for several units

The derivation here allows for **more than one** output unit. The weights are changed to reduce the **difference** between the net input to the **output unit**, y_{in_J} and the **target value** t_J.

This formulation reduces the **error for each pattern**.

The d **delta rule** for adjusting the weight from the Ith **input unit** to the Jth **output unit** (for each pattern) is

$$\triangle w_{IJ} = \alpha(t_J - y_{in_J})x_I$$

The **squared error** for a particular training pattern is

$$E = \sum_{j=1}^{m} (t_j - y_{in_j})^2$$

E is a function of all the weights.

The **gradient** of E is a vector consisting of the partial derivatives of E with respect to each of the weights.

The error can be **reduced most rapidly** by adjusting the weight w_{IJ} in the direction of $-\partial E / \partial w_{IJ}$.

We now find an explicit formula for the gradient for the arbitrary weight w_{IJ}. First, note that

$$\begin{aligned}\frac{\partial E}{\partial w_{IJ}} &= \frac{\partial}{\partial w_{IJ}} \sum_{j=1}^{m} (t_j - y_{in_j})^2 \\ &= \frac{\partial}{\partial w_{IJ}} (t_J - y_{in_J})^2\end{aligned}$$

since the weight w_{IJ} **influences the error** only at output unit Y_j.

Furthermore, using the fact that

$$y_{in_J} = \sum_{i=1}^{n} x_i w_{iJ}$$

we obtain

$$\begin{aligned}\frac{\partial E}{\partial w_{IJ}} &= -2(t_J - y_{in_J})\frac{\partial y_{in_J}}{w_{IJ}} \\ &= -2(t_J - y_{in_J})x_I\end{aligned}$$

Thus, the local error will be reduced most rapidly (for a given learning rate) by adjusting the weights according to the delta rule.

References

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