

31YB: LECTURE 5 :

Monday, 6th October, 2003

OVERVIEW:

Some mathematical preliminaries: Review of McCulloch-Pitts Neuron Model

Review of: 3-term Supervised Learning

Rosenblatt's Perceptron Learning Rule (PLR)

Principle of Gradient Descent

Widrow-Hoff Learning Rule: Delta Rule

Review: McCulloch-Pitts/Perceptron model

Neuron sums its weighted inputs:

$$w_0 + w_1 x_1 + \dots + w_n x_n = \sum_{i=0}^n w_i x_i = \mathbf{w}\mathbf{x}$$

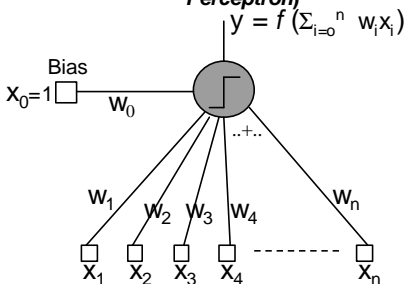
Neuron applies threshold activation function:

$$y = f(\mathbf{w}\mathbf{x})$$

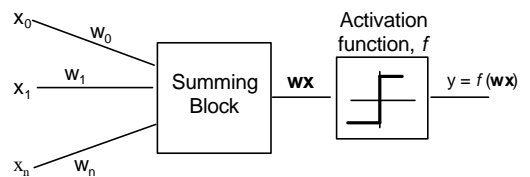
where, e.g., $f(\mathbf{w}\mathbf{x}) = +1$ if $\mathbf{w}\mathbf{x} > 0$

$$f(\mathbf{w}\mathbf{x}) = -1 \text{ if } \mathbf{w}\mathbf{x} \leq 0$$

Mathematical Preliminaries Review of McCulloch-Pitts model (also known as Perceptron)



Review: McCulloch-Pitts/Perceptron model



Mathematical preliminaries: Vector Notation

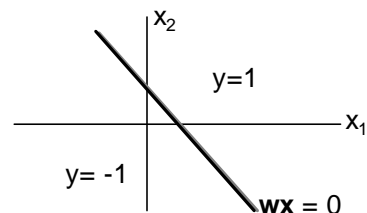
Vectors appear in lowercase **bold** type

e.g. input vector: $\mathbf{x} = [x_0 \ x_1 \ x_2 \ \dots \ x_n]$

Dot product of two vectors:

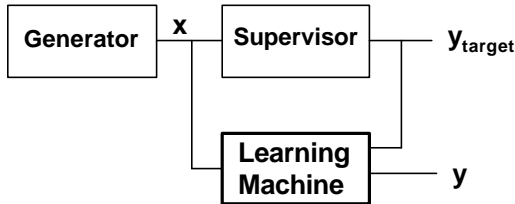
$$\begin{aligned} \mathbf{w}\mathbf{x} &= w_0 x_0 + w_1 x_1 + \dots + w_n x_n \\ &= \sum_{i=0}^n w_i x_i \end{aligned}$$

Review: Geometrical interpretation



Neuron defines two regions in input space where it outputs -1 and 1. These regions are separated by a hyperplane $\mathbf{w}\mathbf{x} = 0$

Review: 3-term Supervised Learning



Training: Learn from training pairs $(\mathbf{x}, \mathbf{y}_{\text{target}})$
Testing: Given any \mathbf{x} , output a value \mathbf{y} close to the supervisor's output $\mathbf{y}_{\text{target}}$

More mathematics: Define: An error function

We'd like to minimise the squared error (which is a function of the weights), for each training pair/pattern:

$$E(\mathbf{w}) = \frac{1}{2} (y_{\text{target}} - y)^2$$

Intuition:

- Square makes error positive and penalises large errors more
- 1/2 just makes some of the maths easier
- The total error will be the sum of errors across all patterns (see Hand-out)
- Need to change the weights in order to minimize the error – how?
 - Use principle of *gradient descent* - Calculate derivative (gradient) of the Error with respect to the weights, and then change the weights by a small increment in the opposite direction to the gradient *-more next!*

Supervised (or 3-term) Training

(Adjusting the weights)

Generate a *training pair or pattern*:

- an input $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]$
- a target output y_{target} (known/given)

1. Neuron computes an output y given \mathbf{x}
2. Compare y with y_{target} (to compute error)
3. Adjust weights, \mathbf{w} , to reduce error
4. Repeat 1-3 many times

How does a weight affect error?:

Principle of Gradient Descent

- Assume there is no output (threshold) activation function during Perceptron learning, i.e. $y = f(\mathbf{w}\mathbf{x}) = \mathbf{w}\mathbf{x}$
 - hence, y is now *not* restricted to having discrete values (due to absence of threshold/step activation function f), but can have any value
- Derivative of error with respect to the weights, \mathbf{w} , is (using partial differentiation - NOT Examenable!):

$$\frac{\partial E}{\partial \mathbf{w}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial \mathbf{w}} = -(y_{\text{target}} - y)\mathbf{x} = -\mathbf{d}\mathbf{x}$$
- So to reduce E by *gradient descent*, move/increment weights in the *opposite direction* to the gradient $-(-\mathbf{d}\mathbf{x})$ (see next!, and see Hand-out 1)

Rosenblatt's Perceptron Learning Rule (PLR) (1962) (for training the Perceptron/McCulloch Pitts Neuron Model)

1. Initialise weights at random
2. For each training pair/pattern $(\mathbf{x}, y_{\text{target}})$:
 - Compute output y (of Perceptron/McCulloch-Pitts Neuron Model)
 - Compute error, $d = (y_{\text{target}} - y)$
 - Use the error to update weights as follows:

$$\Delta \mathbf{w} = \mathbf{w} - \mathbf{w}_{\text{old}} = \eta \mathbf{d} \mathbf{x} \quad \text{or,} \quad \mathbf{w} = \mathbf{w}_{\text{old}} + \eta \mathbf{d} \mathbf{x}$$
 where η is a constant learning rate (see later)
3. Repeat 2 until convergence [i.e. error d is zero]

Example of PLR Update rule (assume $\eta = 0.5$):

| | | |
|--|------|---|
| if $y = +1$ & $y_{\text{target}} = -1$ | then | $\mathbf{w} = \mathbf{w}_{\text{old}} - \mathbf{x}$ |
| if $y = -1$ & $y_{\text{target}} = +1$ | then | $\mathbf{w} = \mathbf{w}_{\text{old}} + \mathbf{x}$ |
| if $y = y_{\text{target}}$ | then | $\mathbf{w} = \mathbf{w}_{\text{old}}$ |

Widrow-Hoff Learning Rule (*Delta Rule*)

$$\Delta \mathbf{w} = \mathbf{w} - \mathbf{w}_{\text{old}} = + \mathbf{h} \mathbf{d} \mathbf{x}$$

or, $\mathbf{w} = \mathbf{w}_{\text{old}} + \mathbf{h} \mathbf{d} \mathbf{x}$

where $\mathbf{d} = y_{\text{target}} - y$

η is a constant that controls the learning rate (amount of weight increment/update $\Delta \mathbf{w}$ at each training step), $\mathbf{h} > 0$
 Note that weight update is proportional to the error, i.e., difference between the output and the target (\mathbf{d}).

★**Note:** Delta Rule (DR) is similar to the Perceptron Learning Rule (PLR) (see earlier), but there are differences...

★e.g. Error (\mathbf{d}) in DR is now not restricted to having values of 0,1 or -1 (as in PLR), but may have any value whatever – in general DR can be derived for any *differentiable* output/activation function f , whereas PLR only works for the Perceptron/*threshold* output function

★ As we will see, the value of η is critical for DR: and this is certainly different from the case of PLR. – see later (assignment!)

For interest (NOT examinable):
Proving the Delta Rule

- To prove the learning rule for the DR (i.e. the error is reduced at each weight update step), we must show that it implements *gradient-descent* in a suitable measure of error, E .
- **See Handout 1 for details** - slides 16-23 (& Beale and Jackson, Chapters 1 & 3)
- **NOTE:**
- *There is NO lecture this Thursday, 9th Oct.*
- *NEXT LECTURE is on Monday, 13th Oct, covering: Multi-Layered Perceptrons (MLP) & Back-Propagation*
- *1st Tutorial during the lecture slot on 16th Oct, Thursday, 10:00, 2A87B*