Lecture 3: Delta Rule

Kevin Swingler

kms@cs.stir.ac.uk

Mathematical Preliminaries: Vector Notation

Vectors appear in lowercase **bold** font

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

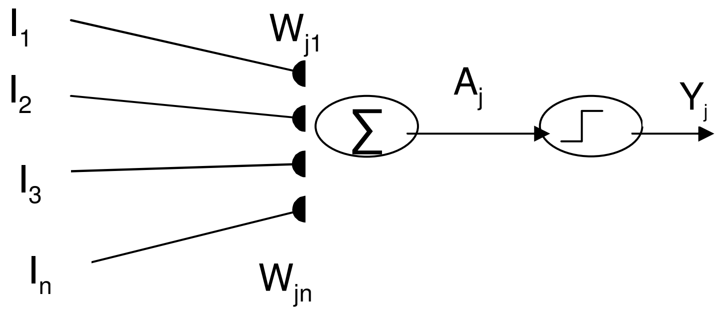
Dot product of two vectors:

$$\mathbf{wx} = w_0\, x_0 + w_1\, x_1 + \ldots + w_n\, x_n =$$

$$= \sum_{i=0}^{n} w_i x_i$$

E.g.: $\mathbf{x} = [1,2,3]$, $\mathbf{y} = [4,5,6]$ $\mathbf{xy} = (1*4)+(2*5)+(3*6) = 4+10+18 = 32$

Review of the McCulloch-Pitts/Perceptron Model



Neuron sums its weighted inputs:

$$w0\ x0 + w1\ x1 + \ldots + wn\ xn = \sum_{i=0}^{n} w_i x_i = \mathbf{w}\ \mathbf{x}$$

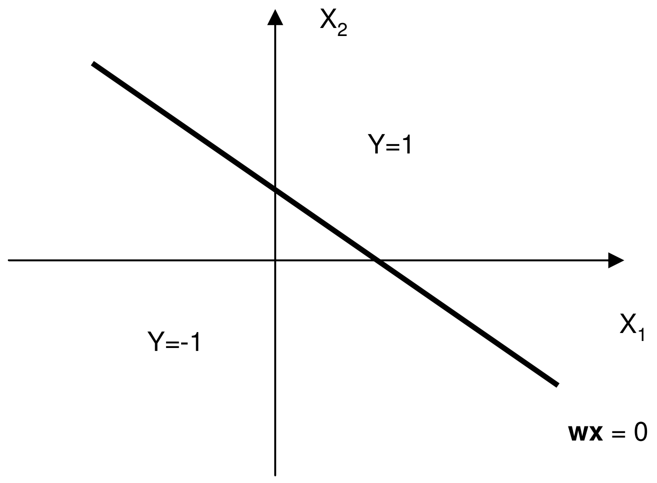
Neuron applies threshold activation function:

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

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

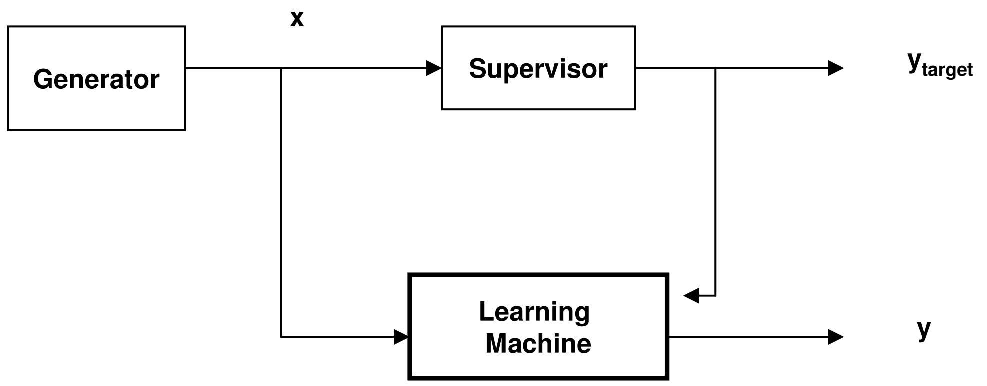
$f(\mathbf{w}\ \mathbf{x}) = -1$ if $\mathbf{w}\ \mathbf{x} \leq 0$

Review of Geometrical Interpretation



Neuron defines two regions in input space where it outputs -1 and 1. The regions are separated by a hyperplane **wx** = 0 (i.e. decision boundary)

Review of Supervised Learning



Training: Learn from training pairs (**x**, **y**$_{\mathbf{target}}$)

Testing: Given **x**, output a value y close to the supervisor's output y_{target}

Learning by Error Minimization

The Perceptron Learning Rule is an algorithm for adjusting the network weights **w** to minimize the difference between the actual and the desired outputs.

We can define a **Cost Function** to quantify this difference:

$$E(w) = \frac{1}{2}\sum_p \sum_j (y_{desired} - y)^2$$

Intuition:

- Square makes error positive and penalises large errors more
- ½ just makes the math easier
- Need to change the weights to minimize the error – How?
- Use principle of **Gradient Descent**

Principle of Gradient Descent

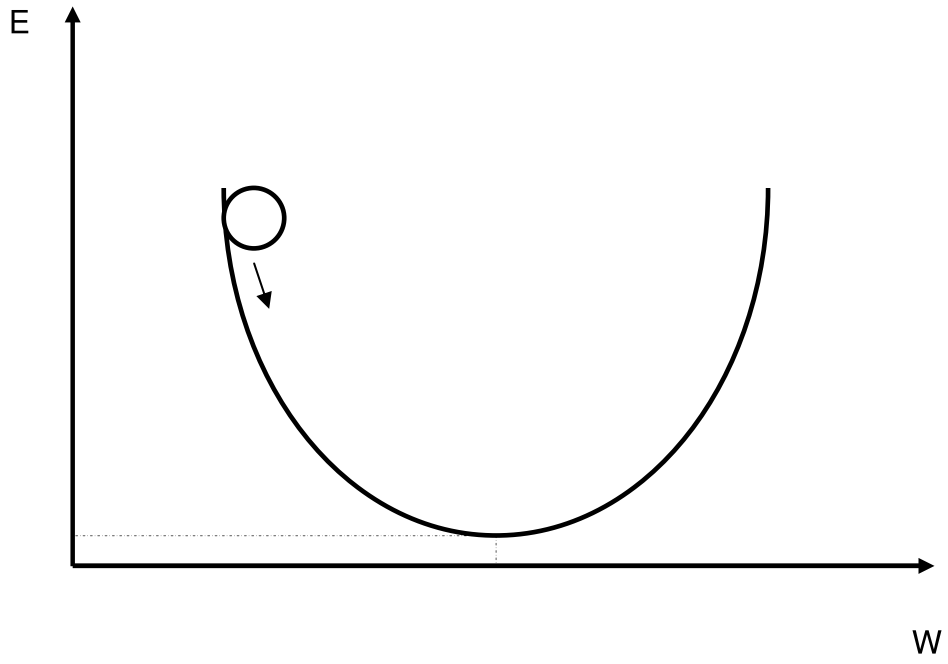
Gradient descent is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to the negative of the gradient of the function as the current point.

So, calculate the derivative (gradient) of the Cost Function with respect to the weights, and then change each weight by a small increment in the negative (opposite) direction to the gradient

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial w} = -(y_{desired} - y)x = -\delta x$$

To reduce E by gradient descent, move/increment weights in the negative direction to the gradient, $-(-\delta x) = +\delta x$

Graphical Representation of Gradient Descent



Widrow-Hoff Learning Rule (Delta Rule)

$$\Delta w = w - w_{old} = -\eta \frac{\partial E}{\partial w} = +\eta \delta x \quad \text{or} \quad w = w_{old} + \eta \delta x$$

where $\delta = y_{target} - y$ and η is a constant that controls the learning rate (amount of increment/update Δw at each training step).

Note: Delta rule (DR) is similar to the Perceptron Learning Rule (PLR), with some differences:

1. Error (δ) in DR is not restricted to having values of 0, 1, or -1 (as in PLR), but may have any value
2. DR can be derived for any *differentiable* output/activation function *f*, whereas in PLR only works for threshold output function

Convergence of PLR/DR

The weight changes Δw_{ij} need to be applied repeatedly for each weight w_{ij} in the network and for each training pattern in the training set.

One pass through all the weights for the whole training set is called an ***epoch*** of training.

After many epochs, the network outputs match the targets for all the training patterns, all the Δw_{ij} are zero and the training process ceases. We then say that the training process has ***converged*** to a solution.

It has been shown that if a possible set of weights for a Perceptron exist, which solve the problem correctly, then the Perceptron Learning rule/Delta Rule (PLR/DR) will find them in a finite number of iterations.

Furthermore, if the problem is linearly separable, then the PLR/DR will find a set of weights in a finite number of iterations that solves the problem correctly.