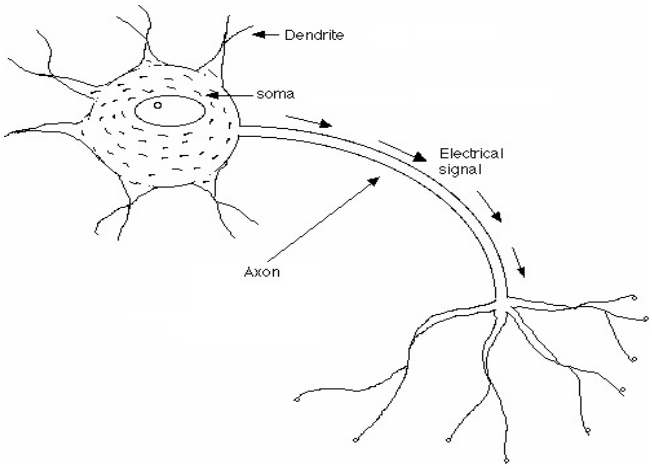
Perceptron

September 12, 2013



Perceptron is a simple model of neuron

Perceptron can only do the task that is *linear separable*

Review of biology of neuron

The unit of nerve cell is called *neuron*

A neuron is an information processing unit , it consists of input and ouput structure called *dendrites* and *axon*

Neuron process *plasticity* , which is the ability to *strengthen/weaken* the inter-neuron connection over time.

The neuron will *fire* the signal if the potential inside the cell body exceed a threshold

The artifical neuron

The artifical neuron can has lots of input : x_1 , x_2 , x_3 , ... , x_n

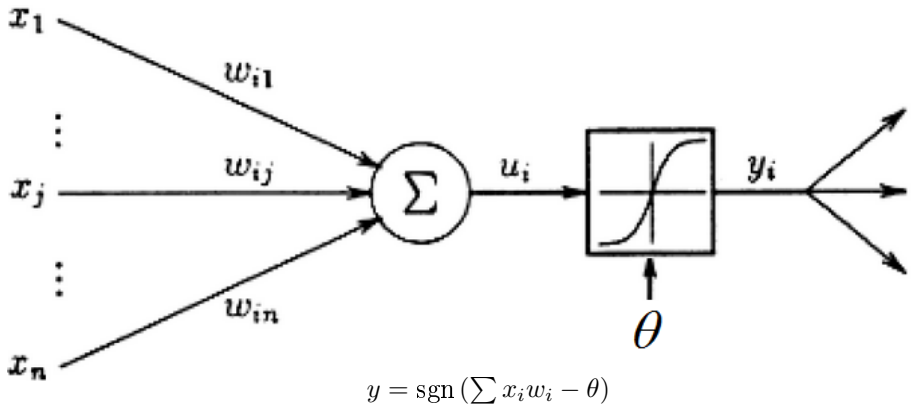
Each input has a weighting : w_1,w_2, w_3 , ... , w_n , all these w can be changed

The total input is the linear combination of all the inputs : $x_1w_1 + x_2w_2 + ... + x_nw_n = \sum x_iw_i$

Since the neuron will only fire if the sum inside exceed a "threshold", thus the equation now also include a threshold term $\sum x_iw_i - \theta$, notice that θ is also changeable

Since the neuron only fires if the sum inside "exceed" the threshold, so a *sgn* function can be used, and the equation now becomes $y = \text{sgn}\,(\sum x_iw_i - \theta)$

This model is called a perceptron



$$y = \text{sgn}\,(\sum x_iw_i - \theta)$$

What perceptron need for it to be able to do classification

For perceptron to be able to do classification, it needs *training set*

A training set is a set T with 2 members , the inputs and desired output (information about the classified type)

$$T = \left\{ \begin{array}{cc} x_1, x_2, ..., x_n & y_1 \\ x_2, x_2, ..., x_n & y_2 \\ \vdots & \vdots \\ x_1, x_2, ..., x_n & y_m \end{array} \right\}$$

Using vector notation

$$T = \{\mathbf{x}_i, y_i\}_{i=1}^{m}$$

Therefore, the perceptron is a kind of *supervised learning* , it need a *teacher* : the teacher knows the solution of certain practice example (the training set) . That is , the teacher provide the training set to the perceptron to train it .

How perceptron learn

The weighting w_i are all randomly assign some value in it, the perceptron is given the trainig set : inputs are feed into the perceptron, the perceptron process the input and generate the output by itself

But the perceptron will not know the output is "correctly" generated unless we *teach* them

How to *teach* them ? The method is to compare the ouput generated by the perceptron and the correct output (from the trainig set), and forms the *error term* : true output - output

Then using this error term to *correct* the weighting

That is , the perceptron learn by correcting the weighting, it is a *collective knowledge* , the network's weighting is the memoery of the knowledge

Then, for the perceptron to learn correctly, the perceptron should have a set of correct weighting

The perceptron algorithm

First initalize the weights $\{w_i\} = \text{random}(-0.5, +0.5)$, and initalize the threshold θ

Put the input into the perceptron and generate the output using $y = \text{sgn}\left(\sum x_i w_i - \theta\right)$

Find the error term $e = y^d - y$

Adjust the weighting and threshold : $w_i(\text{step\_k+1}) = w_i\,(\text{step\_k}) + \Delta w_i\,(\text{step\_k})$,

$$\theta(\text{step\_k+1}) = \theta\,(\text{step\_k}) + \Delta\theta\,(\text{step\_k})$$

Where $\Delta w_i = \alpha x_i e$, α is the learning rate constant, that $\alpha \in (0, +\infty)$ (α is non-negative)

For $\Delta\theta = \alpha(-1)e$

The Δw defined in this way can "correct" the w towards the "correct direction".

If w is too large that it over-estimate the output (output too larger) , then $e = y^d - y$ will be negative and thus $\Delta w = \alpha x e$ will be negative to "correct" the w to a smaller value

If w is too small that it under-estimate the output (output too small) , then $e = y^d - y$ will be positive and thus $\Delta w = axe$ will be positive to "correct" the w to a larger value

What is the learning rateα

The learning rate constant is a control of the speed of the learning process

For large α , the Δw will be large, that means the step size will be large and the improvement of the w may be oscillatory

For smaller α, the Δw will be small, it can make finer adjustment, but the speed may be too slow

The Perceptron Algorithm

1. Initialize the w_i and θ_i
2. Generate ouptut with training set input $y = \text{sgn}\left(\sum x_i w_i - \theta\right)$
3. Generate error $e = y^d - y$
4. Generate updating $w_i(k+1) = w_i\,(k) + \alpha x_i e, \quad \theta(k+1) = \theta\,(k) - \alpha e$
5. Loop step 2 to 4.

Other form of the artifical neuron

The Integration Function

The integration function determine how the inputs are summed up, there are other integration scheme
Since the θ is also changing, thus we can treat it as a weighting with input -1
The general form of the integration function is

$$I\left(x_1, x_2, ..., x_n, -1, w_1, w_2, ..., w_n, \theta\right)$$

When I is simple linear combination :

$$I_{Linear}\left(x_1, x_2, ..., x_n, -1, w_1, w_2, ..., w_n, \theta\right) = x_1 w_1 + ... + x_n w_n + (-1)\theta = \sum x_i w_i - \theta$$

When I is quadratic

$$I_{Quad}\left(x_1, x_2, ..., x_n, -1, w_1, w_2, ..., w_n, \theta\right) = x_1^2 w_1 + ... + x_n^2 w_n + (-1)\theta = \sum x_i^2 w_i - \theta$$

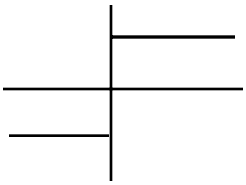
When I is spherical

$$I_{Sphe}\left(x_1, x_2, ..., x_n, -1, w_1, w_2, ..., w_n, \theta\right) = (x_1 - w_1)^2 + ... + (x_n - w_n)^2 + (-1)\theta = \sum (x_i - w_i)^2 - \theta$$

The Activation Function

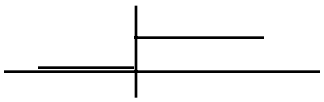
Apart form *sgn* function, other function can be used to generate the otuput
f is sgn function

$$f_{Sgn}(x) = \operatorname{sgn}(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases}$$



f is step function

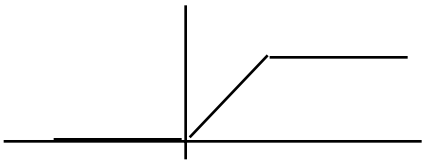
$$f_{Step}(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$



The functions above are called *hard limiter*

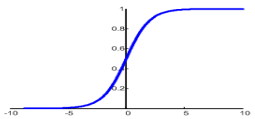
The following functions are *soft limiter*
f is ramp function

$$f_{Ramp}(x) = \begin{cases} 1 & x > 1 \\ x & 1 \geq x \geq 0 \\ 0 & x < 0 \end{cases}$$



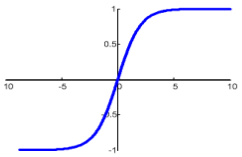
f is unipolar sigmoid function

$$f_{Sigmoid,A}(x) = \frac{1}{1 + e^{-Ax}}$$



f is bipolar sigmoid function

$$f_{Sigmoid,A}(x) = \frac{2}{1 + e^{-Ax}} - 1$$



f is modified arc tan

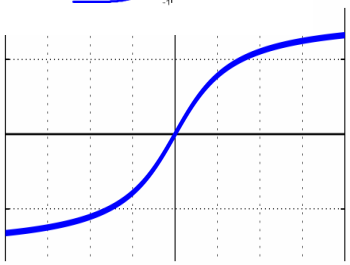
$$f_{modArcTan}(x) = \frac{2}{\pi} \tan^{-1} x$$

f is arc tanh

$$f_{arcTanh}(x) = \tanh^{-1} x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

f is squash function

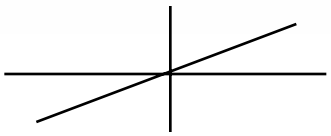
$$f_{squash}(x) = \frac{x}{1 + |x|}$$



These 3 look alike

f is linear

$$f_{Linear}(x) = x$$



Negative Threshold

Since $\Delta w_i(k) = \alpha x_i e, \quad$ but $\Delta\theta(k) = \alpha(-1)e$ are not quite the same form

Thus , if θ is treated as a weighting with input -1 , then

Let $\phi = -\theta$, the input will be $+1$, and then
$\Delta w_i(k) = \alpha x_i e$
$\Delta\phi(k) = \alpha(1)e$

$w_i(k+1) = w_i(k) + \alpha x_i e$
$\phi_i(k+1) = \phi(k) + \alpha e$
They become the same form

Vector Notation

If $x_1, x_2, ..., x_n$ are grouped into one vector $\mathbf{x}$ and vecotr notation is ued , the equation will become more compact

1. Initialize the $\mathbf{w}$ and ϕ
2. Generate ouptut with training set input $y = \text{sgn}\left(\mathbf{x^T w} + \phi\right)$
3. Generate error $e = y^d - y$
4. Generate updating $\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha \mathbf{x} e, \quad \phi(k+1) = \phi\,(k) + \alpha e$
5. Loop step 2 to 4.

$-END-$