In the field of machine learning, the goal of statistical classification is to use an object's characteristics to identify which class (or group) it belongs to. A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics. An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector.

The perceptron is an algorithm for supervised classification of an input into one of several possible non-binary outputs. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time.

The perceptron is a binary classifier which maps its input x (a real-valued vector) to an output value f(x) (a single binary value):


f(x) = \begin{cases}1 & \text{if }w \cdot x + b > 0\\0 & \text{otherwise}\end{cases}


Where w is a vector of real-valued weights, w \cdot x is the dot product (which here computes a weighted sum), and b is the 'bias', a constant term that does not depend on any input value.

The perceptron learning algorithm does not terminate if the learning set is not linearly separable. If the vectors are not linearly separable learning will never reach a point where all vectors are classified properly. The most famous example of the perceptron's inability to solve problems with linearly non separable vectors is the Boolean exclusive-or problem.

**Definitions**

We first define some variables:

* y = f(\mathbf{z}) \, Denotes the *output* from the perceptron for an input vector\mathbf{z}.
* b \, Is the *bias* term, which in the example below we take to be 0.
* D = \{(\mathbf{x}_1,d_1),\dots,(\mathbf{x}_s,d_s)\} \, is the *training set* of s samples, where:
  + \mathbf{x}_j Is the n-dimensional input vector.
  + d_j \, Is the desired output value of the perceptron for that input.

We show the values of the nodes as follows:

* x_{j,i} \, Is the value of the ith node of the jth training *input vector*.
* x_{j,0} = 1 \,.

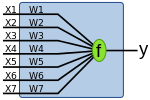
To represent the weights:

* w_i \, Is the ith value in the *weight vector*, to be multiplied by the value of the ith input node.
* Becausex_{j,0} = 1 \,, the w_0 \, effectively replaces the bias term.

To show the time-dependence of\mathbf{w}, we use:

* w_i(t) \, Is the weight i at timet.
* \alpha \, Is the *learning rate*, where0 < \alpha \leq 1.

Too high a learning rate makes the perceptron periodically oscillate around the solute

[](http://en.wikipedia.org/wiki/File:Perceptron.svg)

The appropriate weights are applied to the inputs, and the resulting weighted sum passed to a function that produces the output y.

**Algorithm**

1. Initialize the weights and the threshold. Weights may be initialized to 0 or to a small random value.

2. For each example j \, in our training setD \,, perform the following steps over the input \mathbf{x}_j \, and desired outputd_j \,:

2a. Calculate the actual output:

y_j(t) = f[\mathbf{w}(t)\cdot\mathbf{x}_j] = f[w_0(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \dotsb + w_n(t)x_{j,n}]

2b. Update the weights:

w_i(t+1) = w_i(t) + \alpha (d_j - y_j(t)) x_{j,i} \,, for all nodes0 \leq i \leq n.