Chapter 1

What is Perceptron?

In machine learning, 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 algorithm dates back to the late 1950s; its first implementation, in custom hardware, was one of the first artificial neural networks to be produced.

1.1 A Bit of History

The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt, funded by the United States Office of Naval Research.

More on the web!

1.2 Definition

In the modern sense, the perceptron is an algorithm for learning a **binary classifier**: a function that maps its input x (a real-valued vector) to an output value f(x) (a single binary value):

$$f(x) = \begin{cases} 1: & w.x + b > 0 \\ 0: & otherwise \end{cases}$$

where...

- **w** is a vector of real-valued weights,
- $\mathbf{w} \cdot \mathbf{x}$ 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.

If b is negative, then the weighted combination of inputs must produce a positive value greater than $|b|$ in order to push the classifier neuron over the 0 threshold. Spatially, the bias alters the position (though not the orientation) of the decision boundary. 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 nonseparable vectors is the Boolean exclusive-or problem.

In the context of neural networks, a perceptron is an **artificial neuron** using the **Heaviside step function** as the activation function. The perceptron algorithm is also termed the **single-layer perceptron**, to distinguish it from a multilayer perceptron, which is a misnomer for a more complicated neural network. As a linear classifier, the single-layer perceptron is the **simplest feedforward neural network**.

1.3 Learning Algorithm

Below is an example of a learning algorithm for a (single-layer) perceptron. For multilayer perceptrons, where a hidden layer exists, more sophisticated algorithms such as **backpropagation** must be used. Alternatively, methods such as the **delta rule** can be used if the function is **non-linear** and **differentiable**, although the one below will work as well.

When multiple perceptrons are combined in an artificial neural network, each output neuron operates independently of all the others; thus, learning each output can be considered in isolation.

1.3.0.1 Definition

We first define some variables:
- $y = f(\vec{\mathbf{z}})$ denotes the output from the perceptron for an input vector $\vec{\mathbf{z}}$.
- $\mathbf{b}$ is the bias term, which in the example below we take to be 0.
- $D = \{\vec{\mathbf{X}}_1, d_1),\ldots,(\vec{\mathbf{X}}_s, d_s)\}$ is the training set of s samples, where:
  - $\vec{\mathbf{X}}_j$ is the n-dimensional input vector.
  - $\mathbf{d_j}$ is the desired output value of the perceptron for that input.

We show the values of the features as follows:
- $x_{j,i}$ is the value of the ith feature 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 feature.
- Because $x_{j,0} = 1$, the w_0 is effectively a learned bias that we use instead of the bias constant b.

To show the time-dependence of $\mathbf{w}$, we use:
- $w_i(t)$ is the weight i at time t.
- α is the learning rate, where $0 < \alpha \leq 1$.
  Too high a learning rate makes the perceptron periodically oscillate around the solution unless additional steps are taken.

1.3.1 Steps

1. Initialise the weights and the threshold. Weights may be initialised to 0 or to a small random value. In the example below, we use 0.
2. For each example j in our training set D , perform the following steps over the input $\mathbf{x}_j$ and desired output d_j :
   a) Calculate the actual output:
   $\mathbf{y_j(t) = f[w(t) \cdot x_j] = f[w_0(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}]}$
   b) Update the weights:
   $\mathbf{w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}}$ for all feature $\mathbf{0 \leq i \leq n}$.
3. For offline learning, the step 2 may be repeated until the iteration error $\mathbf{\frac{1}{s}\sum_{j=1}^{s}|d_j - y_j(t)|}$ is less than a user-specified error threshold γ, or a predetermined number of iterations have been completed.

The algorithm updates the weights after steps $2a$ and $2b$. These weights are immediately applied to a pair in the training set, and subsequently updated, rather than waiting until all pairs in the training set have undergone these steps.

1.4 Convergence

The perceptron is a linear classifier, therefore it will never get to the state with all the input vectors classified correctly if the training set **D** is not linearly separable, i.e. if the positive examples can not be separated from the negative examples by a hyperplane.

1.5 Pocket Algorithm

The pocket algorithm with ratchet (Gallant, 1990) solves the stability problem of perceptron learning by keeping the best solution seen so far "in its pocket". The pocket algorithm then returns the solution in the pocket, rather than the last solution. It can be used also for non-separable data sets, where the aim is to find a perceptron with a small number of misclassifications.

1.6 Example

A perceptron learns to perform a binary NAND function on inputs x_1 and x_2 .

- Inputs: x_0 , x_1 , x_2 , with input x_0 held constant at 1.
- Threshold (θ): 0.5
- Bias (b): 0
- Learning rate (r): 0.1
- Training set, consisting of four samples: $\{((1,0,0),1),((1,0,1),1),((1,1,0),1),((1,1,1),0)\}$

In the following, the final weights of one iteration become the initial weights of the next. Each cycle over all the samples in the training set is demarcated with heavy lines.

Table 1.1: Example of weight updates for the Perceptron Network

I/P		Intial Weights	O/P			Error	Correction	Final Weights	
Sensor Values	Desired O/P		Per sensor	Sum	Network				
		x_0	x_1	x_2	z	w_0	w_1	w_2	c_0
w_2									

1.7 References

Without following links Ctrl + c and Ctrl + v would have not happened!

http://en.wikipedia.org/wiki/Perceptron