Parceptron

Akira Imada

Most recently renewed on

May 8, 2010

1 Single Parceptron

Assume we have 10 inputs $x_1, x_2, \cdots, x_{10}$ and one output y. Also assume each input is binary 1 or -1. All the input is connected to output with synapse whose weight is w_i ($i = 1, 2, \cdots 10$). If $w_1x_1 + w_2x_2 + \cdots + w_{10}x_{10}$ is larger than a threshold θ then $y = 1$, otherwise $y = -1$. That is,

$$y = sgn(\sum_{i=1}^{10} w_i x_i - \theta). \tag{1}$$

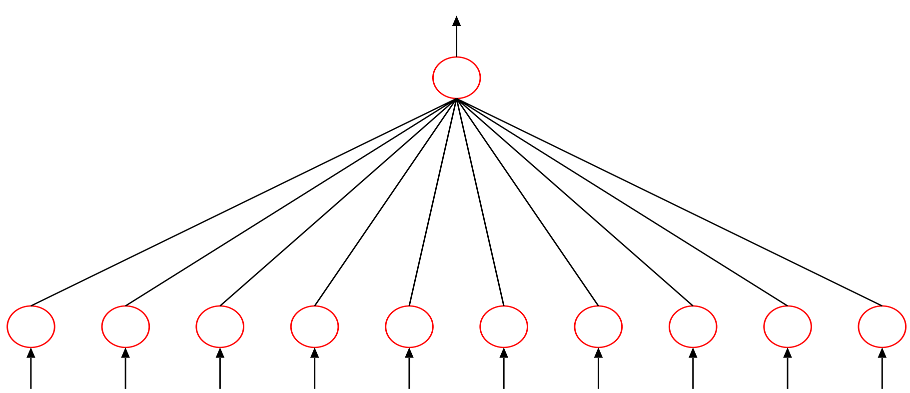


Figure 1: Schematic diagram of Single Parceptron.

Task 1 (A Framework) *Assigning random weight value from -1 to 1, create a program which accepts M binary inputs from keyboard, calculate y, and display 10 input and output with their color being green if the value is 1 or red if the value is -1. Also show weighted sum of the inputs $\sum_{i=1}^{10} w_i x_i$ on the screen like the following Figure.*

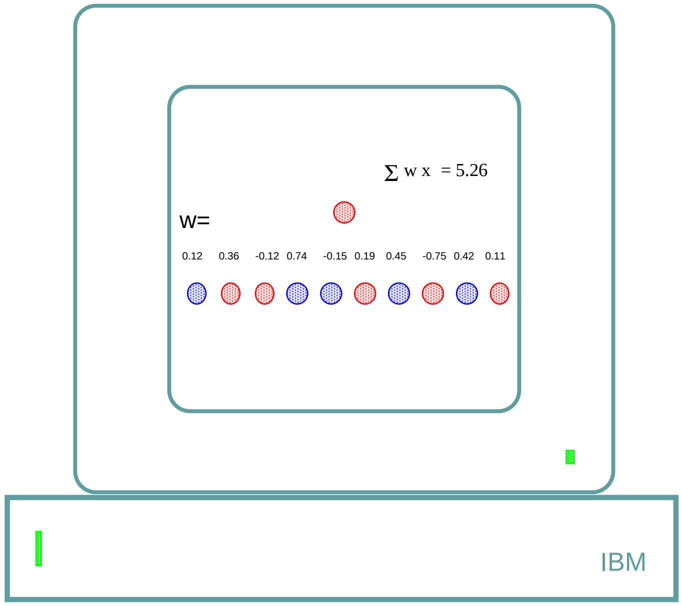


Figure 2: An example of display of the task.

Examples of how can it work.

One of the most typical example is to realise a Boolean function. For example, output is 1 if and only if all the input is 1, otherwise -1.

Table 1: An example of Boolean Function assuming 4 inputs case. Note that we call this AND logic when the number of input is 2.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	y
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	1	-1
...	...	...	...	...	...	...	...
1	1	1	1	1	1	1	-1
1	1	1	1	1	1	1	1

Task 2 (Extended AND) *Discover a set of weights and threshold so that $y = 1$ if and only if input are all 1, otherwise $y = -1$.*

Task 3 (Extended OR) *Discover a set of weights and threshold so that $y = 1$ if at least one input is 1 and $y = -1$ if all input is -1.*

Task 4 (EVEN PARITY) *What about the case where $y = 1$ if the number of 1 is odd, otherwise $y = -1$?*

2 Perceptron Learning

Assume now we have m input and n output w_{ij} is a weight value from input j to output neuron i. X^p is p-th input vector for training, that is,

$$X^p = (x_1^p, x_2^p, x_3^p, ...x_n^p),$$

$\hat{Y}^p$ is target output vector when p-th training input X^p is given, that is,

$$\hat{Y}^p = (\hat{y}_1^p, \hat{y}_2^p, \hat{y}_3^p, \cdots, \hat{y}_m^p)$$

Y^p is actual output vector, that is,

$$Y^p = (y_1^p, y_1^p, y_1^p, \cdots y_m^p)$$

Then we can describe a learning of weights as

$$W(t+1) = W(t) + \eta(\hat{Y}^p - Y^p)^t(X^p).$$

Note that $(\hat{Y}^p - Y^p)^t(X^p)$ is an outer product of two vectors, and result is a matrix of a same size of W, and η is a learning coefficient set to a small real number ranging $(0 < \eta < 1$. The learning is repeated until the change in weight value becomes neglectable. In other words, we have to preset p enough a large number so that all of $w_{ij}(t+1) - w_{ij}(t)$ becomes almost 0, after repeating above $p$times. This is also called *Widrow-Hoff Learning Algorithm*

Note that outer product of this specific two vectors is calculated as below.

$$\begin{pmatrix} a \\ b \end{pmatrix} (xyz) = \begin{pmatrix} ax & ay & az \\ bx & by & bz \end{pmatrix}$$

3 Multi-layer Parceptron

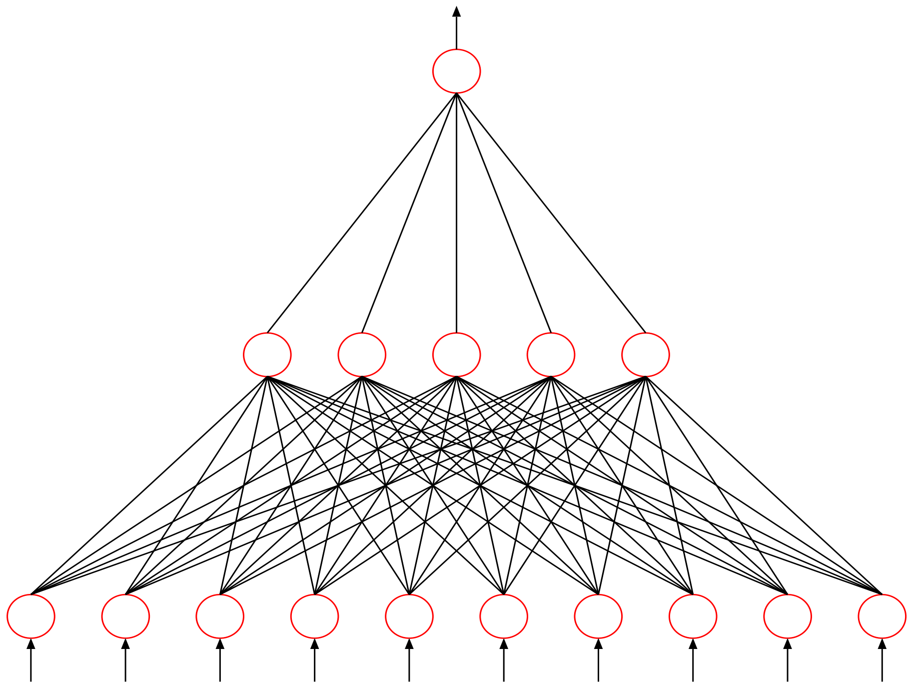


Figure 3: Schematic diagram of Multi-layer Parceptron.

4 Robot Navigation in a Grid-world

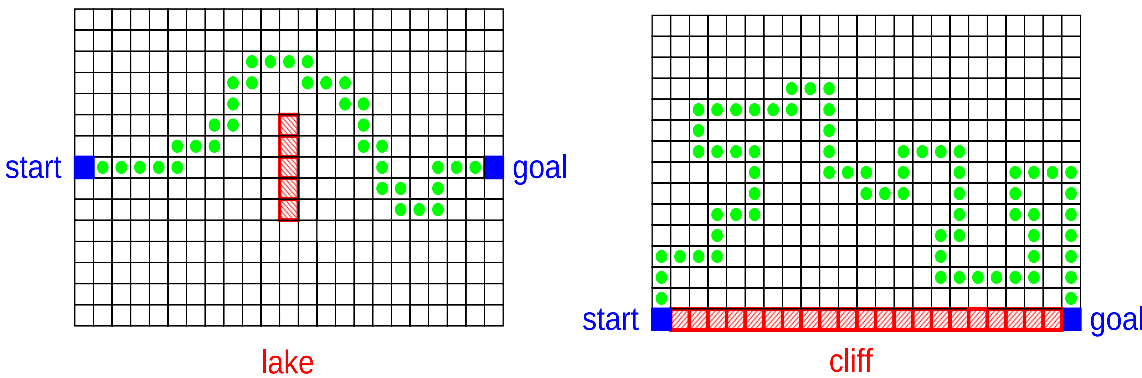
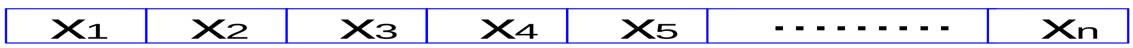


Figure 4: Two examples of grid-world to be explored.

5 Let's try an evolution of chromosom of lucky dog.

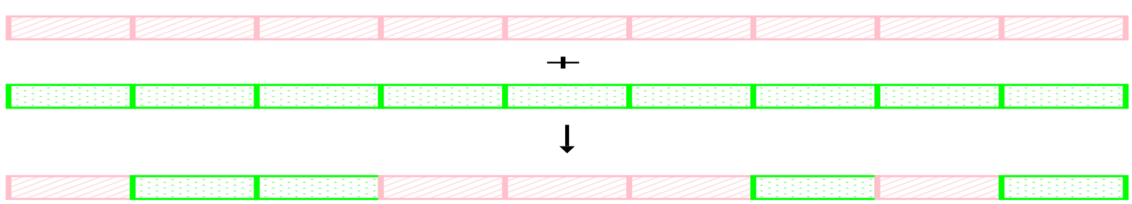
- Represent a series of x_i as a *population* of strings. Each of these strings is refered to as *chromosome* or sometimes called *individual*.



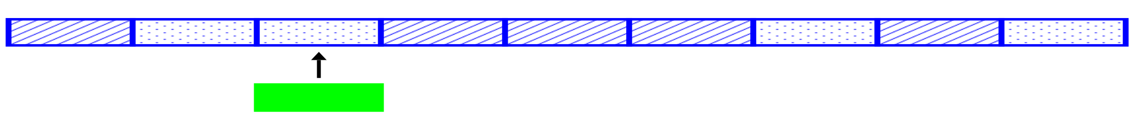
- Define a *fitness* evaluation by "How good is each individual?"

Then we start an evolution as follows, expecting better solutions from generation to generation.

1. **(Initialization)** Generate an initial *population* of p individuals at random.
2. **(Fitness Evaluation)** Evaluate fitness of each chromosome and sort the chromosomes according to its fitness from the best to the worst.
3. **(Selection)** Select two chromosomes
   - Here, from the best half of the population at random, which is called a *Truncate Selection*.
4. **(Reproduction)** Produce a child by the following two operations:
   - *Uniform Crossover*, for example



   - *Mutation*



5. Create the next generation by repeating the steps from 3 to 4 n times.
6. Repeat the steps from 2 to 5 until (near) optimal solution is obtained.