# Boolean Automator

Presented by: Maxwell B. Hasbrouck

# The Problem

| $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ | $y_8$ | $y_9$ | $y_{10}$ | $y_{11}$ | $y_{12}$ | $y_{13}$ | $y_{14}$ | $y_{15}$ | $y_{16}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 2: All Boolean operations on two inputs

Most operations can be handled by a single layer Perceptron, but XOR and Logical Bitconditional can't be

# The Background

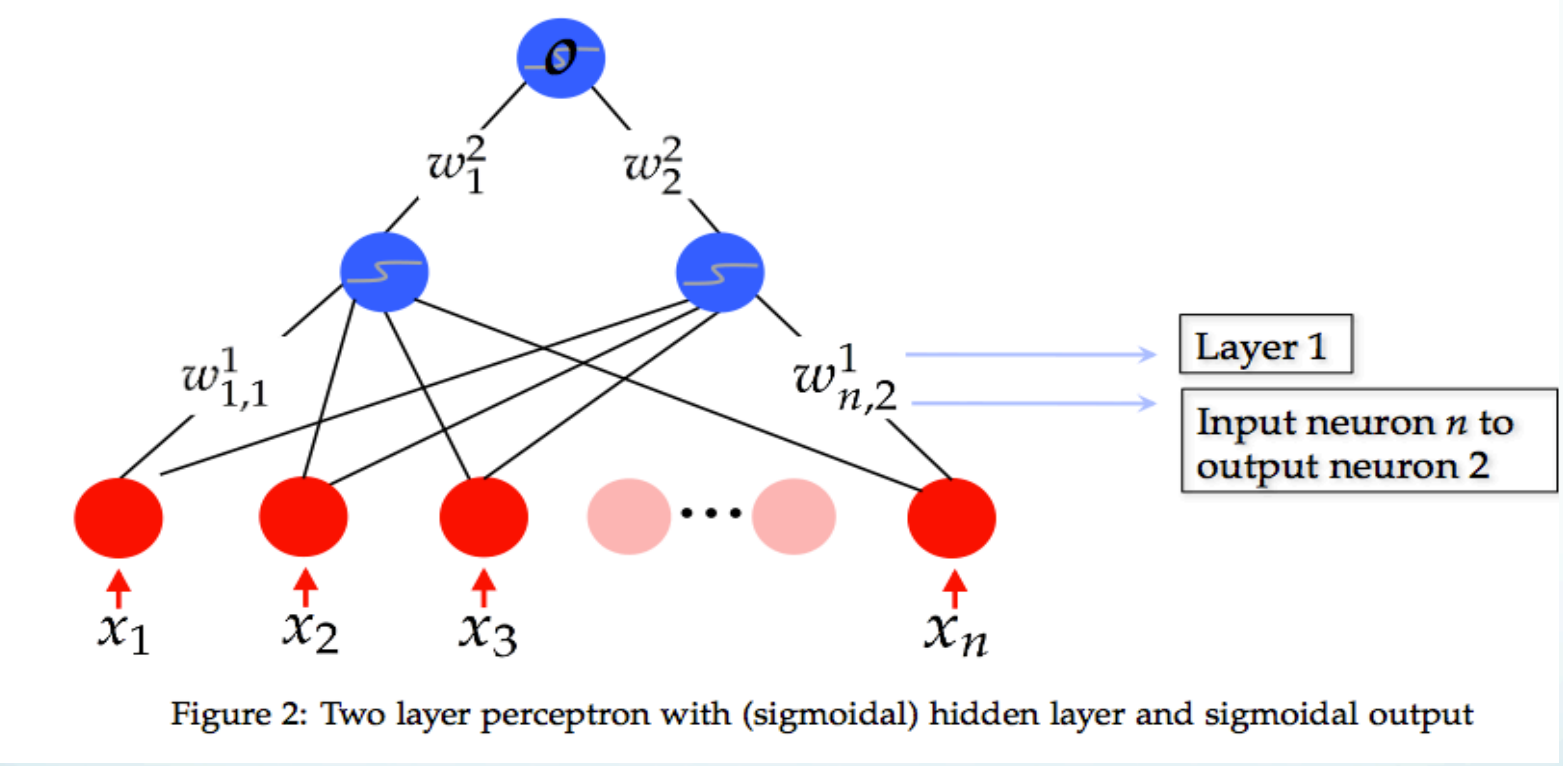


Figure 2: Two layer perceptron with (sigmoidal) hidden layer and sigmoidal output

The output at each layer (hidden and output) is calculated by passing a weighted sum of inputs through the sigmoid function. Each hidden neuron sums its inputs multiplied by their weights

# Steps 1-4

**Step One:** for input $x = [x_1....x_n]$, calculate output $h_j$ at each hidden layer neuron

$$h_j = \sigma(\sum_{i=1}^{n} w_{i,j}^1 x_i) \text{ where } \sigma(\theta) = \frac{1}{1+e^{-\beta\theta}} \quad (3)$$

Use $\beta = 1$.

**Step Two:** for input $h = [h_1...h_j]$ from hidden layer, calculate output of output neuron

$$o = \sigma(\sum_{j} w_j^2 h_j), \quad (4)$$

where $\sigma(\theta)$ is the same as above. Note that $w_j^2$ indicates weights in layer 2, not squares.

**Step Three:** calculate $\delta^2$, which will be used to update layer 2 weights

$$\delta^2 = (o-y)(o)(1-o) \quad (5)$$

because there is one output, $\delta^2$ is a scalar.

**Step Four:** calculate $\delta^1$, which will be used to update layer 1 weights

$$\delta_j^1 = (\delta^2)(w_j^2 h_j(1-h_j)) \quad (6)$$

because there are $j$ hidden neuron outputs, $\delta^1$ is a $j$-vector.

# Steps 5-8

**Step Five:** calculate $\Delta w^2$

$$\Delta w_j^2 = \ell \delta^2 h_j \tag{7}$$

where $\ell$ is the learning rate. Use $\ell = 0.1$.

**Step Six:** calculate $\Delta w^1$

$$\Delta w_{i,j}^1 = \ell \delta_j^1 x_i \tag{8}$$

$\Delta w_{i,j}^1$ can be conveniently stored in matrix form. If $\delta^1$ is a $j$ by 1 vector and $x$ is a 1 by $i$ vector, $\delta^1 * x$ will produce a matrix with $j$ rows and $i$ columns.

**Step Seven:** update layer 2 weights

$$w_j^2 = w_j^2 - \Delta w_j^2 \tag{9}$$

**Step Eight:** update layer 1 weights

$$w_{i,j}^1 = w_{i,j}^1 - \Delta w_{i,j}^1 \tag{10}$$

# Functions

`booletron`: a driver that runs `multiperceptron` on all 16 Boolean operations and checks its accuracy by comparing its output `[guess]` to the desired output `[y]`.

`guess = multiperceptron(x,y)`, where input matrix `[x]` remains the same for all 16 operations and `[y]` varies. `multiperceptron` trains the perceptron to perform the Boolean operation specified by `[y]` using the algorithm detailed previously and returns the perceptron's output after training.