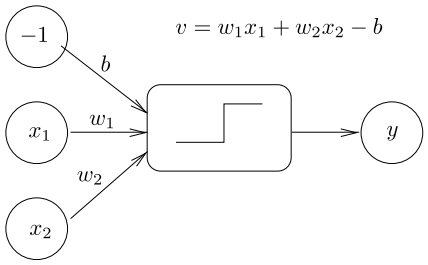
Tutorial 5 and 6: Perceptrons

Perceptrons

1. Implement a perceptron, with a threshold activation function. The perceptron should have two inputs x_1 and x_2 connected by weights w_1 and w_2. There should be also a bias input of -1 weighted by parameter b.



   The output of the perceptron is given by a threshold function:

$$y = f_{\text{hardlim}}(v) = \begin{cases} 1 & v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

2. **Quiz question:** Extend your perceptron code to solve the following classification task (perceptron training pseudocode is provided below):

   Train the perceptron to classify the following 5 numbers (0,1,2,3,4) into two classes: even or odd. The training set consists of these patterns (examples):

```
01110   00100   01110   01110   01000
10001   01100   10010   00001   01000
10001   00100   00100   01110   01010
10001   00100   01000   00001   01111
01110   01110   11111   01110   00010
```

   Thus, the first input vector is (011101000110001100010111 0). Plus the bias input. Desired output values will be +1 for odd numbers and 0 for even numbers.

   Testing the generalisation ability: take the same input vectors and flip 5 random bits in each, i.e. $1 \rightarrow 0$ and $0 \rightarrow 1$. See whether the perceptron still recognises the numbers and classifies them correctly as odd/even.

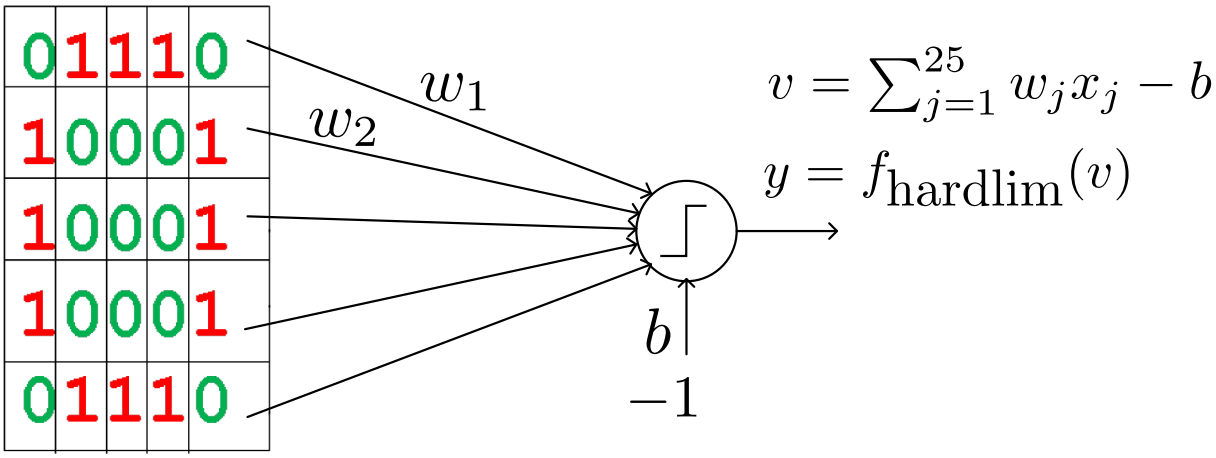


Figure 1: Setup of the input to the perceptron.

IMPORTANT: Submit a copy of your program and its output for the classification task. The program output should show the evolution of performance during training and performance on the testing sets in terms of number of correct or wrong classifications and the decrease of error.

Pseudocode:

(a) Initialise:

```
for each weight w[j]:
     w[j] = random(-1,1) // small random number
b = random(-1,1)         // small random number
epoch = 0
```

(b) Start a new epoch:

```
example = 0
CORRECT = 0
```

(c) For the current training example, calculate the perceptron's output

```
result(input[example]) {
     v = -1 * b                  // bias
     for each input j
          v = v + input[example][j] * w[j]
     if v >= 0                   // hardlim function
          return 1
     else
          return 0
}
```

(d) Calculate the error:

```
error = target_otuput[example] - result(input[example])
```

(e) Update the weights (`alpha` is the learning rate parameter that you need to set to some small value):

```
for each weight w[j]:
     w[j] = w[j] + alpha * error * input[example][j]
b = b + alpha * error * -1
```

(f) If there are more training examples, select the next `example++` and go to Step (d). Else continue.

(g) Freeze the parameters and do the testing.

```
if result(input[example])==target_output[example]
     CORRECT++
```

(h) If CORRECT==number of training examples, then stop. Else set `epochs++` and go to Step (b).