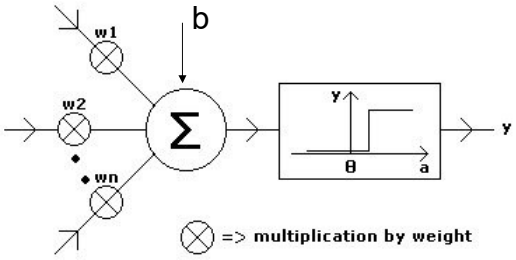
A Simple Neurone Model (The Perceptron)

- A perceptron has analogue inputs but binary output.
- Each input has an associated weight.
- Positive weights correspond to excitatory inputs and negative weights to inhibitory inputs.



$$out = f(net)$$

$$net = \sum_{i=1}^{n} w_i \cdot x_i + b$$

$$f = \begin{cases} 1 & if\ net > threshold \\ 0 & if\ net \le threshold \end{cases}$$

Human and Artificial Neurons (Investigating similarities)

- The human brain →most sophisticated device
- Learn from previous experience
- Great success in dealing unforeseen situations utilising the knowledge from previous experience
- Computer technology rely entirely on human pre-defined instructions
- Bugs may cause all sorts of unexpected results

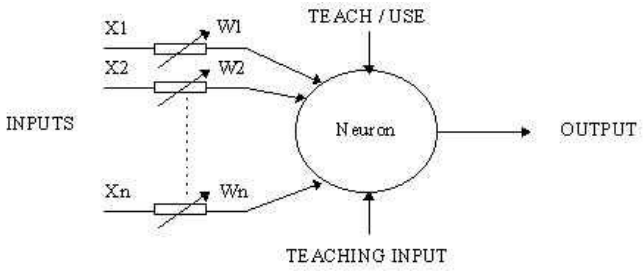
From Human neurons to artificial neurons

- We deduce essential feature and interconnections of human neurons
- Program a computer to simulate them
- As Knowledge is incomplete and computation power is limited
  - Gross idealization of real neurons

DIFFERENT APPROACHES TO PRE-PROCESS & LEARNING ALGORITHM

ANOTHER APPROACH

- A simple perceptron
  - Training mode
  - Testing mode



ANOTHER APPROACH

- For example, a 3-input (X_1,X_2 and X_3) perceptron is taught
  - Output 1 when the input is 111 or 101 &
  - Output 0 when the input is 000 or 001
- Truth table can be made to see all possibilities

INITIAL EFFORT

X1:		0	0	0	0	1	1	1	1
X2:		0	0	1	1	0	0	1	1
X3:		0	1	0	1	0	1	0	1
OUT:		0	0	0/1	0/1	0/1	1	0/1	1

TRAINING RULE

- Training
  - Hamming distance technique
  - Generalization to ensure all possible patterns are responded with appropriate outpu

AFTER TRAINING (GENERALIZATION)

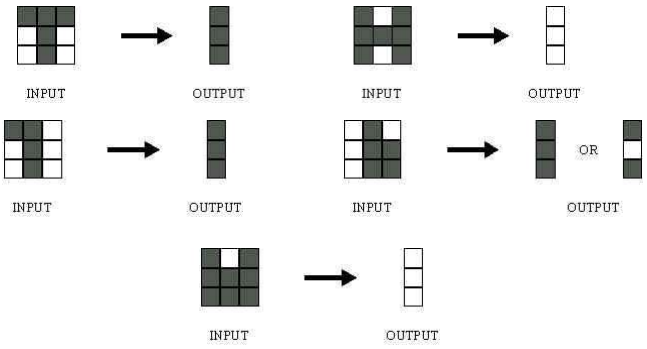
X1:	0	0	0	0	1	1	1	1
X2:	0	0	1	1	0	0	1	1
X3:	0	1	0	1	0	1	0	1
OUT:	0	0	0	0/1	0/1	1	1	1

ANOTHER APPROACH

- The difference between the two truth tables is called the *generalisation of the neuron*
- The firing rule gives the neuron a sense of similarity and enables it to respond 'sensibly' to patterns not seen during training

EXAMPLE TO SOLVE

- Pattern recognition – an example



TOP NEURON

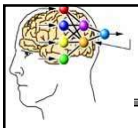
X11:		0	0	0	0	1	1	1	1
X12:		0	0	1	1	0	0	1	1
X13:		0	1	0	1	0	1	0	1
OUT:		0	0	1	1	0	0	1	1

MIDDLE NEURON

X21:		0	0	0	0	1	1	1	1
X22:		0	0	1	1	0	0	1	1
X23:		0	1	0	1	0	1	0	1
OUT:		1	0/1	1	0/1	0/1	0	0/1	0

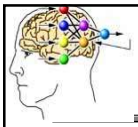
BOTTOM NEURON

X21:		0	0	0	0	1	1	1	1
X22:		0	0	1	1	0	0	1	1
X23:		0	1	0	1	0	1	0	1
OUT:		1	0	1	1	0	0	1	0



The Neuron

- The neuron is the basic information processing unit of an ANN. It consists of:
  1. A set of links, describing the neuron inputs, with weights $W_1, W_2, \ldots, W_m$
  2. An adder function (linear combiner) for computing the weighted sum of the inputs (real numbers): $u = \sum_{j=1}^{m} w_j x_j$
  3. Activation function (squashing function) φ for limiting the amplitude of the neuron output. $y = \varphi(u + b)$

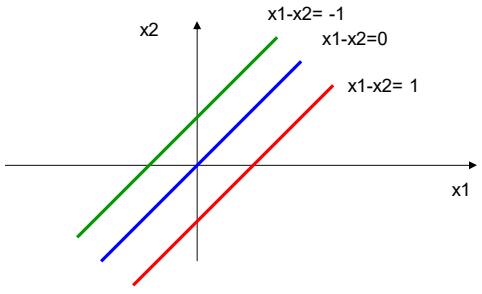


Bias of a Neuron

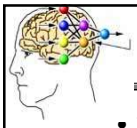
- The bias ***b*** has the effect of applying an affine transformation to the weighted sum ***u***

  v = u + b

- ***v*** is called **induced field** of the neuron



$u = x1 - x2$

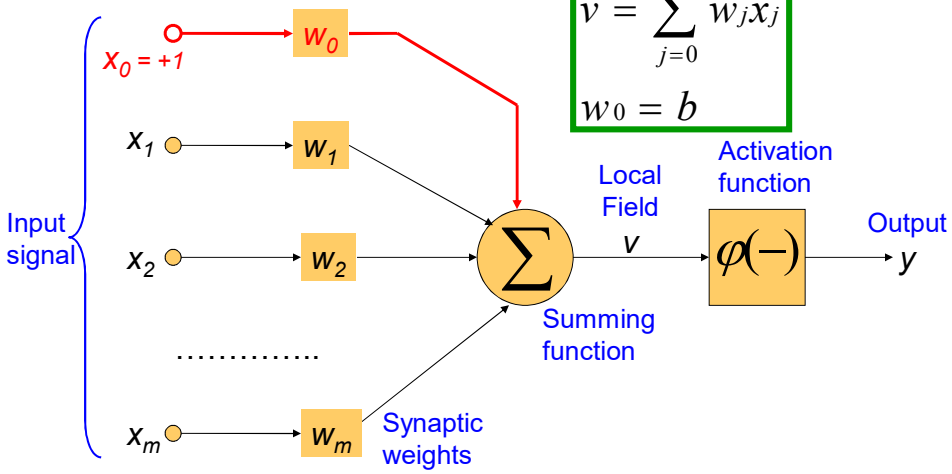


Bias as extra input

• The bias is an external parameter of the neuron. *It* can be modeled by adding an extra input.

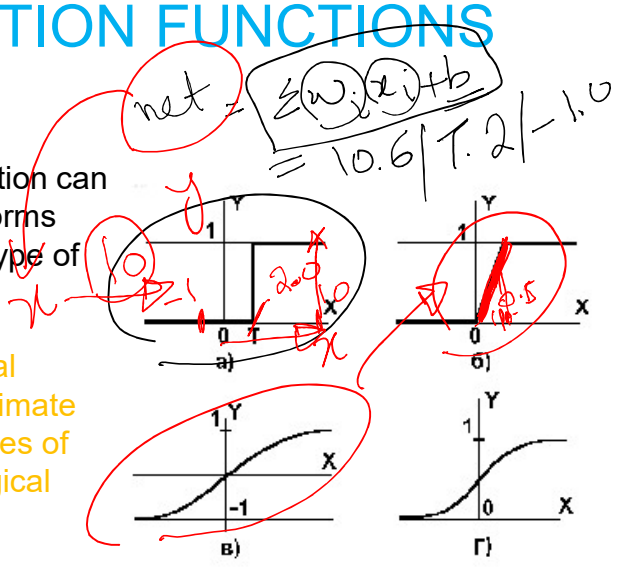
$$v = \sum_{j=0}^{m} w_j x_j$$

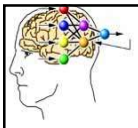
$$w_0 = b$$



ACTIVATION FUNCTIONS

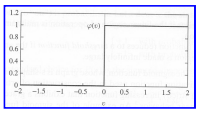
- The activation function can take a number of forms depending on the type of application.
- These mathematical alternatives approximate with different degrees of accuracy the biological neurone activity.





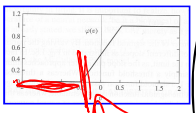
Activation Function φ

There are different activation functions used in different applications. The most common ones are:



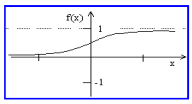
Hard-limiter

$$\varphi(v) = \begin{cases} 1 & if\ v \geq 0 \\ 0 & if\ v < 0 \end{cases}$$



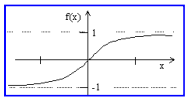
Piecewise linear

$$\varphi(v) = \begin{cases} 1 & if\ v \geq 1/2 \\ v & if\ 1/2 > v > -1/2 \\ 0 & if\ v \leq -1/2 \end{cases}$$



Sigmoid

$$\varphi(v) = \frac{1}{1+\exp(-av)}$$



Hyperbolic tangent

$$\varphi(v) = \tanh(v)$$

Neuron Models

- The choice of φ determines the neuron model. Examples:
- step function:

$$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > c \end{cases}$$

- ramp function:

$$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > d \\ a + ((v-c)(b-a)/(d-c)) & \text{otherwise} \end{cases}$$

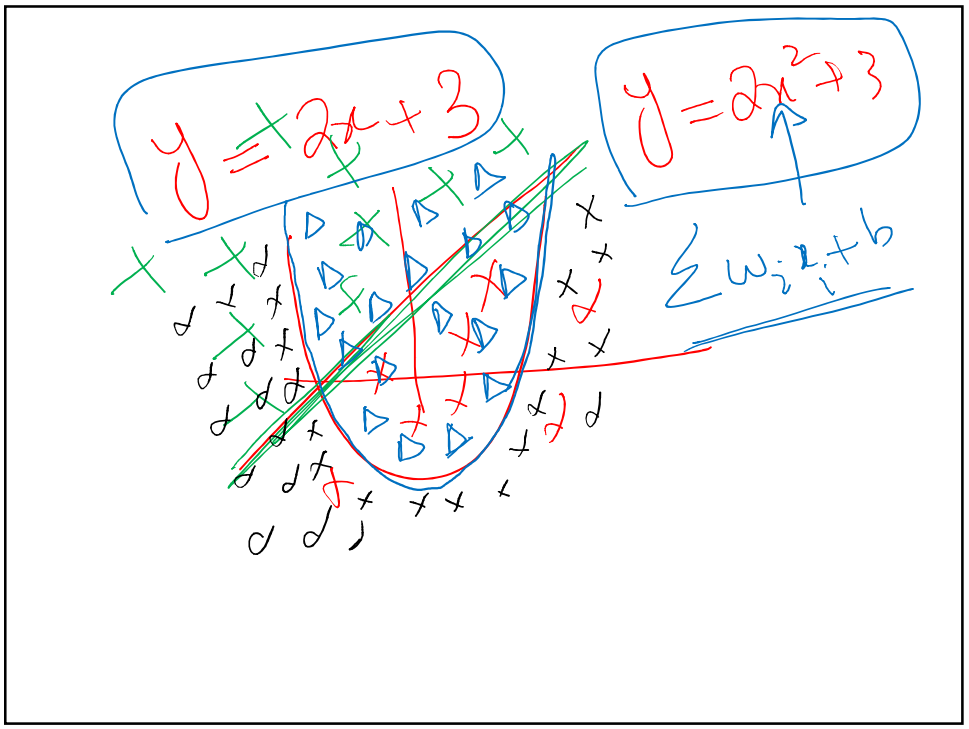
- sigmoid function:

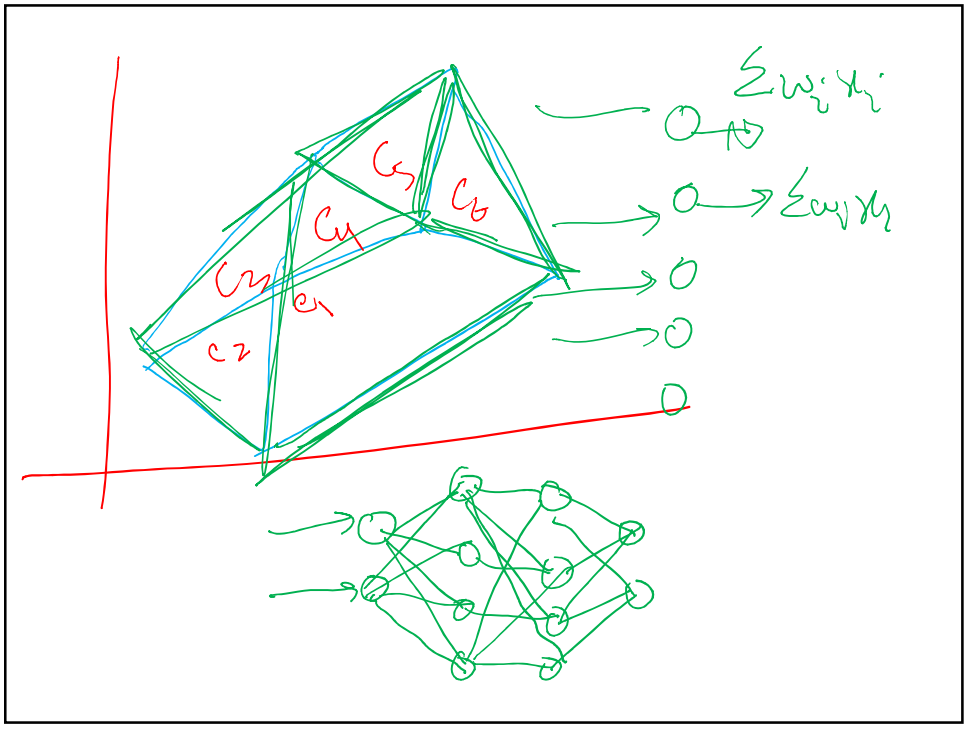
with z,x,y parameters

$$\varphi(v) = z + \frac{1}{1+\exp(-xv+y)}$$

- Gaussian function:

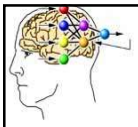
$$\varphi(v) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\left(\frac{v-\mu}{\sigma}\right)^2\right)$$


$y = x + 2x + 3$
$y = 2x^2 + 3$
$\sum w_i x_i + b$

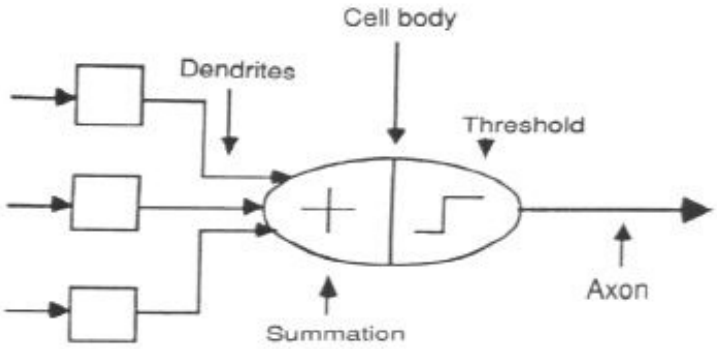

$\sum w_i x_i$
$\sum w_i x_i$
C1
C2
C3
C4
C5
C6

So WHAT EXPERTS DID?

- They devised ANN by deducing essential feature and interconnections of human neurons
- Then Program a computer to simulate them
- As Knowledge is incomplete and computation power is limited
  - We reach to Gross idealization of real neurons

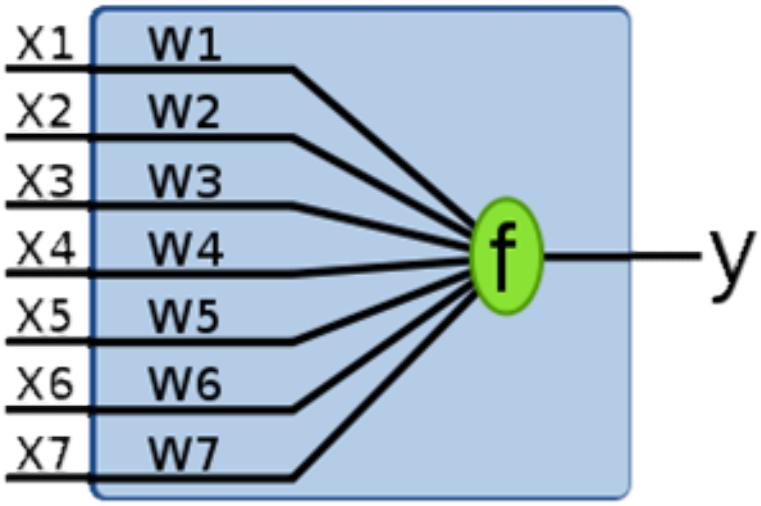


Coming back to Neuron Model



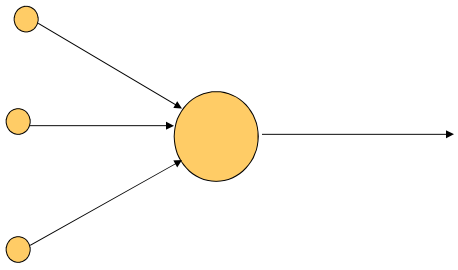
We relate this to A better term PERCEPTRON

Perceptron



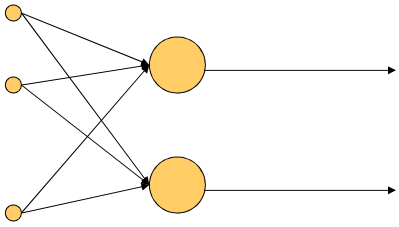
Perceptron: architecture

- We consider the architecture: feed-forward NN with one layer
- It is sufficient to study single layer Perceptron with just one neuron:



Single layer perceptrons

- Generalization to single layer Perceptrons with more neurons is easy because:

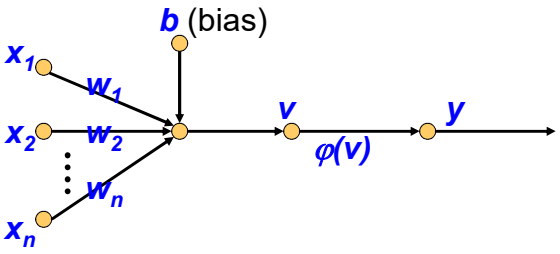


•The output units are independent among each other
•Each weight only affects one of the outputs

Perceptron: Neuron Model

- The (McCulloch-Pitts) Perceptron is a single layer NN with a non-linear φ, the sign function

$$\varphi(v) = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$



Perceptron for Classification

- The perceptron is used for binary classification
- Given training examples of classes C_1, C_2 train the Perceptron in such a way that it classifies correctly the training examples:
  - ***If the output of the Perceptron is +1 (>0) then the input is assigned to class C_1***
  - ***If the output is -1 (<0)then the input is assigned to C_2***

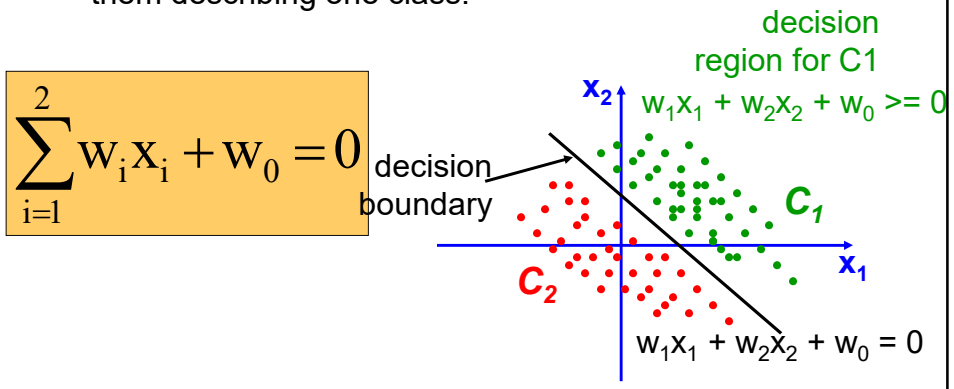
Perceptron Training

- How can we train a Perceptron for a classification task?
- We try to find suitable values for the weights in such a way that the training examples are correctly classified
- Geometrically, we try to find a hyper-plane that separates the examples of the two classes

Perceptron Geometric View

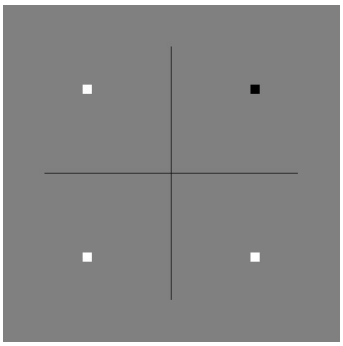
The equation below describes a (hyper-)plane in the input space consisting of real valued 2D vectors. The plane splits the input space into two regions, each of them describing one class.

$$\sum_{i=1}^{2} w_i x_i + w_0 = 0$$



Example: AND

- Here is a representation of the AND function
- White means *false*, black means *true* for output
- -1 means *false*, +1 means *true* for the input



-1 AND -1 = false

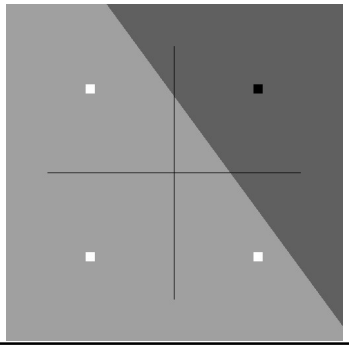
-1 AND +1 = false

+1 AND -1 = false

+1 AND +1 = true

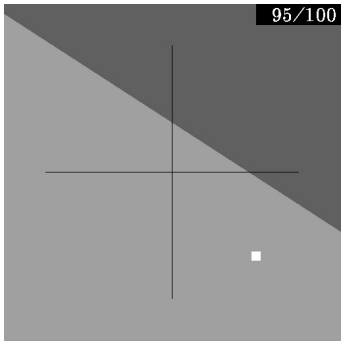
Example: AND continued

- A linear decision surface (a plane in 3D space) intersecting the feature space (the 2D plane where z=0) separates *false* from *true* instances



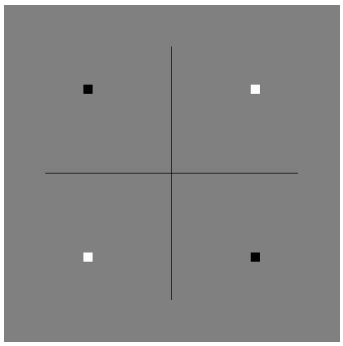
Example: AND continued

- Watch a Perceptron learn the AND function:



Example: XOR

- Here's the XOR function:



-1 XOR -1 = *false*

-1 XOR +1 = *true*

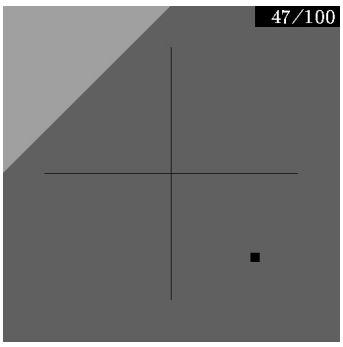
+1 XOR -1 = *true*

+1 XOR +1 = *false*

Perceptrons cannot learn such *linearly inseparable* functions

Example: XOR continued

- Watch a Perceptron try to learn XOR



FAILS?

- How to train the NEURON ?

Fixed increment learning algorithm

- Step 0: Initialize weight and bias.
  - (For simplicity, set weight & bias to zero.)
  - Set learning rate α ($0 < \alpha \leq 1$).
    - (For simplicity, α can be set to 1.)
- Step 1: While stopping condition is false,
  - do steps 2- 5.
- Step 2: For each input set (training pair, input & target), do steps 3-4

Continue…

- Step 3: Compute response of output unit:

$$y\_in = b + \sum_i x_i w_i;$$

$$y = \begin{cases} 1 & \text{if y\_in } \geq \theta \\ -1 & \text{if } < \theta \end{cases}$$

Continue….

- Step 4: Update weights and bias if an error occurred for this pattern.

$$if\ y \neq t,$$

$$w_i(new) = w_i(old) + \alpha t x_i$$

$$b(new) = b(old) + \alpha\ t.$$

$$else$$

$$w_i(new) = w_i(old)$$

$$b(new) = b(old)$$

Continue..

- Step 5. **Test the stopping condition**: If no weights changed in step2, stop; else, continue.

Perceptron: Learning Algorithm

- Variables and parameters at iteration n of the learning algorithm:

  x (n) = input vector
  $= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$

  w(n) = weight vector
  $= [b(n), w_1(n), w_2(n), \ldots, w_m(n)]^T$

  b(n) = bias

  y(n) = actual response

  d(n) = desired response

  η = learning rate parameter

The fixed-increment learning algorithm

n=1;
initialize **w**(n) randomly;
while (there are misclassified training examples)
    Select a misclassified augmented example (**x**(n),*d(n)*)
    w(n+1) = **w**(n) + ηd(n)**x**(n);
    n = n+1;
end-while;

η = learning rate parameter (real number)