Artificial Neural Networks

Artificial Intelligence

Intelligent Systems - ANNs

Aim example

binary classification for any input data (discrete or continuous)

- data can be separated by:
  - a line → $ax + by + c = 0$ (if $m = 2$)
  - a plan → $ax + by + cz + d = 0$ (if $m = 3$)
  - a hyperplan → $\sum a_i x_i + b = 0$ (if $m > 3$)
- How do we identify the optimal values of a, b, c, d, a_i?
  - Artificial Neural Networks (ANNs)
  - Support Vector Machines (SVMs)

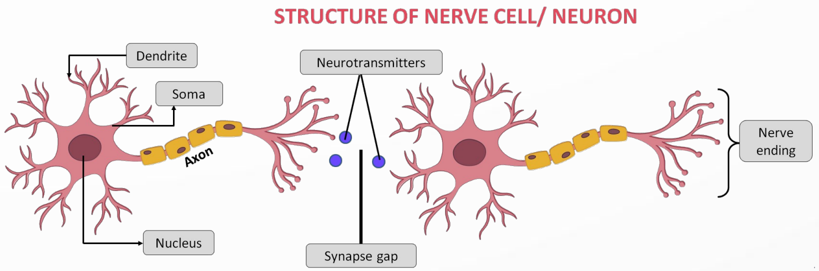
Why ANN?

some tasks can be easy done by humans, but they are difficult to be encoded as algorithms

- Shape recognition
  - Old friends
  - Handwritten
  - Voice
- Rational processes
  - Car driving
  - Piano playing
  - Basketball playing
  - Swimming

such tasks are to difficult to be formalized and done by a rational process

The learning process (brain)



Human brain

- ~10.000.000.000 of neurons connected through synapses

A neuron

- has a body (soma), an axon, and more dendrites
- can be in a given state
  - Active state – if the input information is over a given stimulation threshold
  - Passive state – otherwise

Synapse

- link between the axon of a neuron and the dendrites of other neurons
- take part to information exchange between neurons
- 5.000 connections/neuron (average)
- during a life, new connections can appear

Brain → *Neural network*

- complex system, non-linear and parallel that processes information
- information is stored and processed by the entire network, not only by a part of network

Models for information processing:

- learning
- storing
- memorizing

Learning

- a basic characteristic
- useful connections become permanent (others are eliminated)

Memory

Short time memory

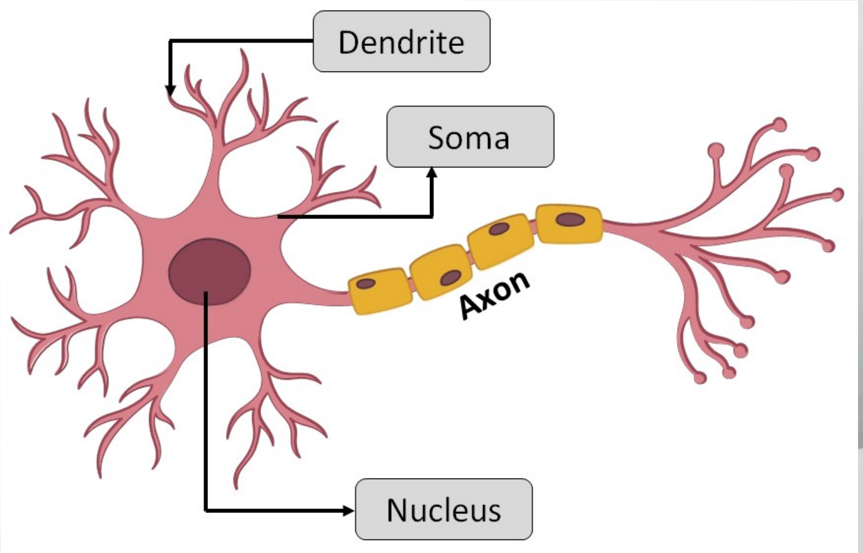
- immediately → 30 sec.
- working memory

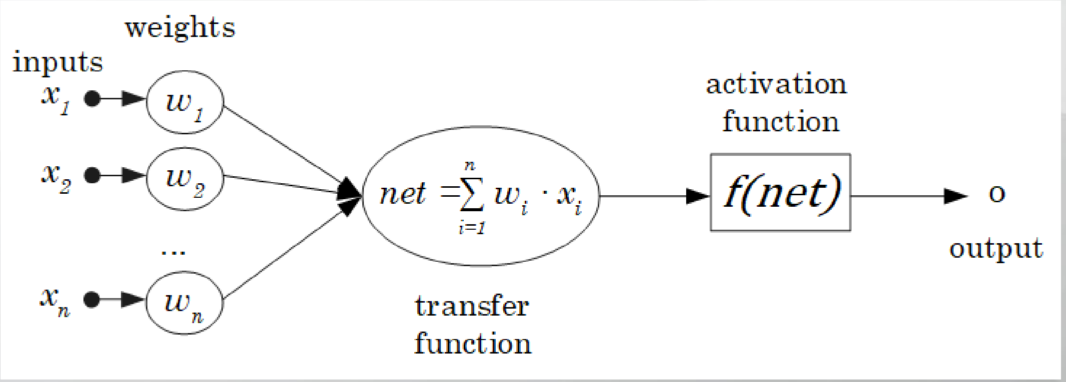
Long term memory

- capacity
- increasing along life
- limited → learning a poetry strophe by strophe
- influenced by emotional states

Biology versus artificial

BNN	ANN
Soma	Node
Dendrite	Input
Axon	Output
Activation	Processing
Synapse	Weighted connection



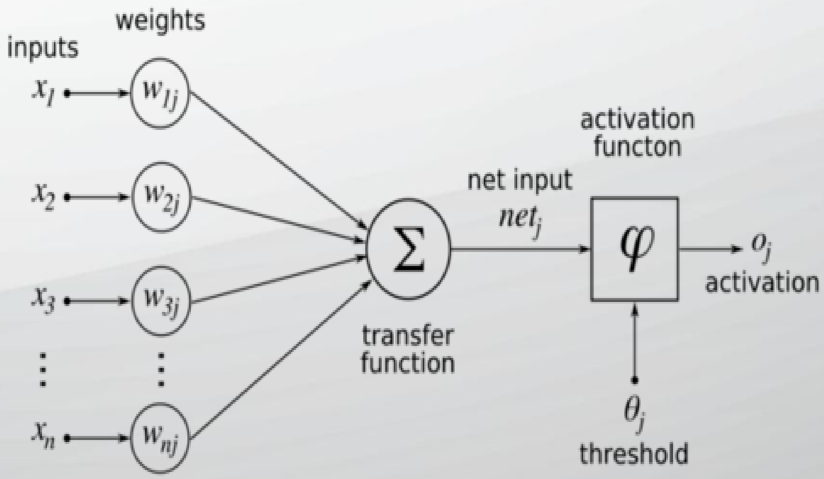


The perceptron

Learning Objectives

By the end of this section you should be able to answer there questions:

1. what is a perceptron?
2. what activation function has a perceptron?
3. what problems can (and can't) be solved by a perceptron?
4. how is trained a perceptron?



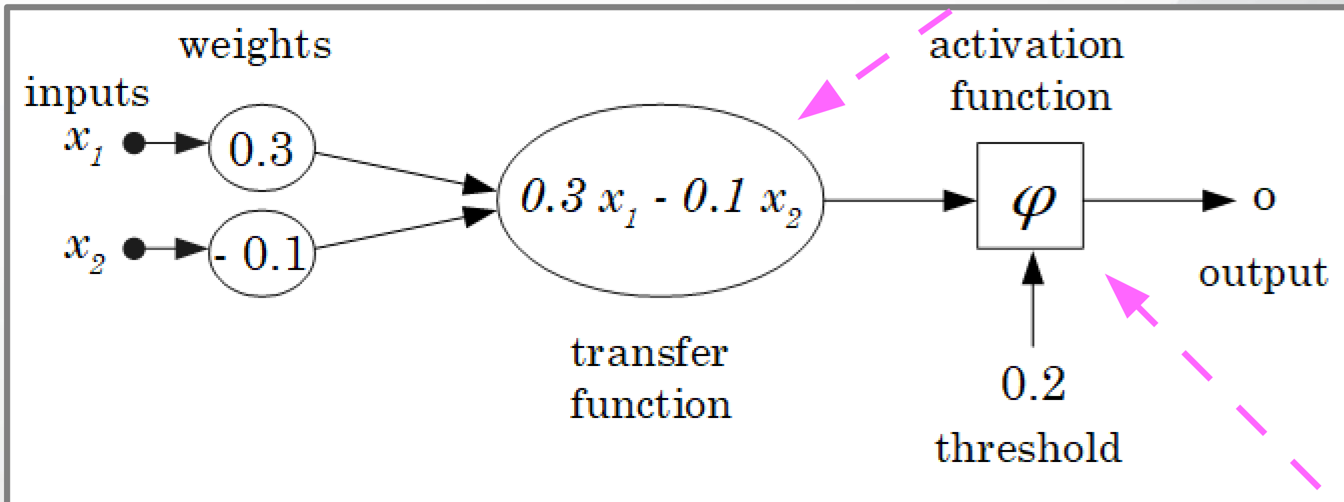
It is the first model of a neuron

- invented by Frank Rosenblatt (1928 – 1971) an American psychologist
- the original perceptron was designed to take a number of binary inputs, and produce one binary output (0 or 1)
- uses different ***weights*** to represent the importance of each ***input***, and that the sum of the values should be greater than a ***threshold value*** before making a ***decision*** like true or false (*0* or *1*)
- the original algorithm:
  1. Set a threshold value
  2. Multiply all inputs with its weights
  3. Sum all the results
  4. Activate the output

Perceptron – example

Consider the parameters: $w_1 = 0.3 \quad w_2 = -0.1 \quad \theta = 0.2$

$$net = \sum_{i=1}^{n} w_i * x_i$$



For the input: $\boldsymbol{x} = (x_1, x_2) = (1,0)$

We get the output: $\varphi(0.3 \times 1 - 0.1 \times 0) = \varphi(0.3) = 1$

Activation function

$$\varphi(net) = \begin{cases} 0, & \text{if } net < \theta \\ 1, & \text{if } net \geqslant \theta \end{cases}$$

threshold function

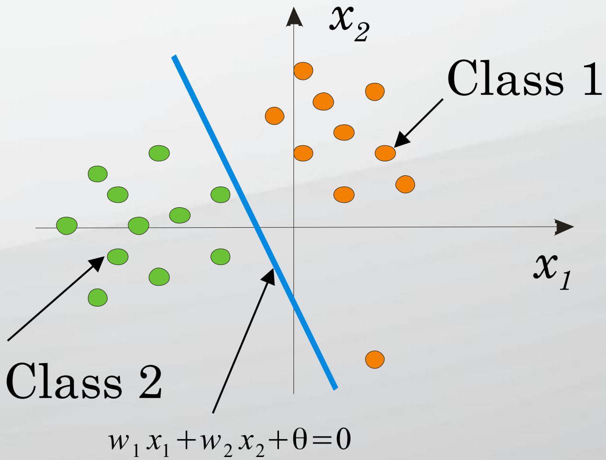
Solving capacity

Observe!

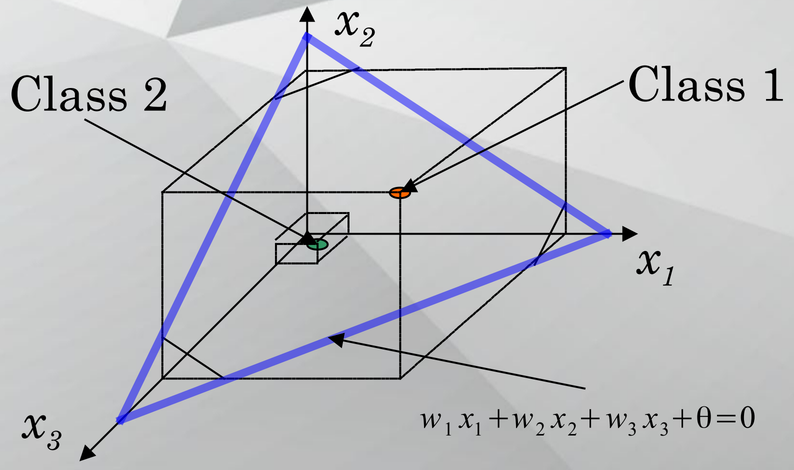
From the transfer function we get

$$y = \sum_{i=1}^{n} w_i x_i \quad - \text{ equation of a } \textbf{hyperplane}.$$

If we compose the transfer function with the threshold function (the activation) we **linear separate** the space $\mathbf{R}^n$ with this hyperplane in two regions.



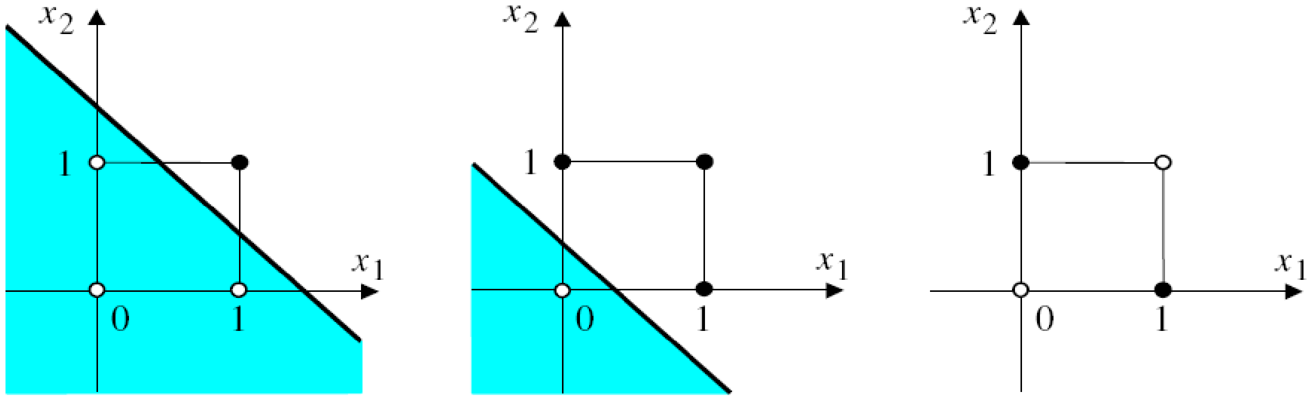
Binary classification with *m= 2* input attributes



Binary classification with *m= 3* input attributes

Perceptron – limits

Non-linear separable data can not be classified.



(a) AND $(x_1 \cap x_2)$ (b) OR $(x_1 \cup x_2)$ (c) Exclusive-OR $(x_1 \oplus x_2)$

A perceptron can learn And and OR operations,
but it can not learn XOR operation (it is not linear separable)

Solutions
- Neuron with continuous threshold
- More neurons

Neuron training

Perceptron's rule → perceptron's algorithm

1. Start by some random weights
2. Determine the quality of the model create for these weights for a **single input data**
3. Re-compute the weights based on the model's quality
4. Repeat (from step 2) until a maximum quality is obtained

Delta's rule → algorithm of gradient descent

1. Start by some random weights
2. Determine the quality of the model create for these weights for **all input data**
3. Re-compute the weights based on the model's quality
4. Repeat (from step 2) until a maximum quality is obtained

Similar to perceptron's rule, but the model's quality is established based on all data (all training data).

The perceptron – learning

Aim:

To identify the optimal weights $(w_1, w_2, \ldots, w_m)$ by minimising the errors.

Training data set of n data

$$\{((x_1^d, x_2^d, \ldots, x_m^d), t^d) : d = 1, 2, \ldots, n\}$$

with m – the number of attributes.

The error is the difference between the real output y and the output o computed by the perceptron for the input $(x_1, x_2, \ldots, x_m)$.

Perceptron's algorithm:

Based on error minimisation associated to an instance of train data

Modify the weights based on error associated to an instance of train data

Perceptron's learning alg.

function training_perceptron$\left(\theta, \eta, m, n, \{((x_1^d, x_2^d, \ldots, x_m^d), t^d) : d=1,2,\ldots,n\}\right)$

w_i = *random(-1, 1)*, where *i=1, 2, ..., m* # initialise random the perceptron's weights

do repeat

for *d = 1* **to** *n* **do**

activate the neuron and determine the output $o^d = \varphi\left(\sum_{i=1}^{n} w_i * x_i^d\right)$

compute the error $e_d = t^d - o^d$

determine the weight modification $\Delta w_i = \eta e_d x_i^d$

modify the weights $w_i = w_i + \Delta w_i$

until there are no incorrect classified examples

return $(w_1, w_2, \ldots, w_m)$

end function

Solving *logic AND* problem

AND	0 (False)	1 (True)
0 (False)	0	0
1 (True)	0	1

threshold

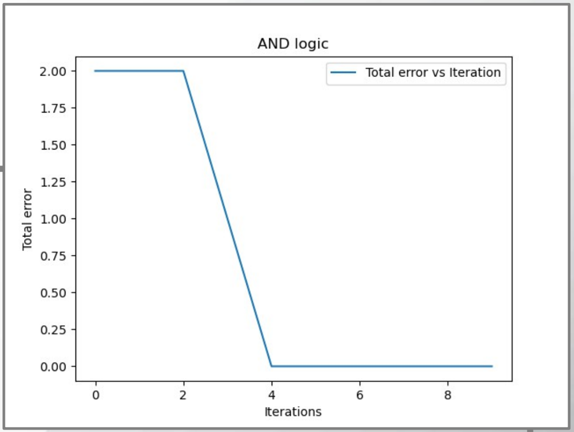
$\theta = 0.2$

learning rate

$\eta = 0.1$

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	–1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	–1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	–1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Perceptron use




```python
from numpy import random, dot, array
import matplotlib as mpl

def threshold(x):
    if x < 0.2 :
        return 0
    return 1

class Perceptron:
    def __init__(self, noInputs, activationFunction, learningRate):
        self.noInputs = noInputs
        self.weights = random.rand(self.noInputs)
        self.activationFunction = activationFunction
        self.output = 0
        self.learningRate = learningRate
        self.errorVariation = []
    def fire(self, inputs):
        self.output = self.activationFunction(
                inner(array(inputs), self.weights))
        return self.output

    def trainPerceptronRule(self, inputs, output):
        totalError = 0
        for i in range(len(inputs)):
            error = output[i] - self.fire(inputs[i])
            delta = self.learningRate*error*array(inputs[i])
            self.weights += delta
            totalError += error**2
        self.errorVariation.append(totalError)
```

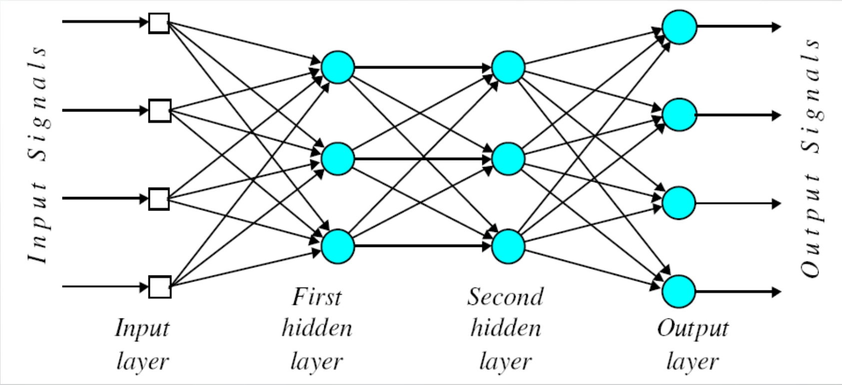
```python
def test1():
    # AND logic
    p = Perceptron(2, threshold, 0.1)
    x = [[1,1],[1,0],[0,1],[0,0]]
    t = [1,0,0,0]
    noIterations = 10
    for i in range(noIterations):
        p.trainPerceptronRule(x,t)
    print(p.weights)
    for j in range(len(x)):
        print(x[j], p.fire(x[j]))

    mpl.pyplot.plot(range(noIterations),
         p.errorVariation, label = 'Total error
                                     vs Iteration')
    mpl.pyplot.xlabel('Iterations')
    mpl.pyplot.ylabel('Total error')
    mpl.pyplot.legend()
    mpl.pyplot.title('AND logic')
    mpl.pyplot.show()
```

Artificial Neural Networks

Learning Objectives

By the end of this section you should be able to answer there questions:

1. What is a *feed forward artificial neural network*?
2. How is organized an ANN?
3. What activation functions are used in ANNs?
4. How passes the signal trough an ANN?
5. How to train an ANN?



A structure similar to a biological neural network

- 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts
- the original ANN was built with electrical circuits
- A set of nodes (units, neurons, processing elements) located in a graph with more layers

In a *feed forward network* the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) to the output nodes.

– is the most simple type of ANN.

Artificial Neural Networks

Nodes

- Have inputs and outputs
- Perform a simple computing through an activation function
- Connected by weighted links
  - Links between nodes give the network structure
  - Links influence the performed computations

Layers

- Input layer

  Contains m nodes (m – the number of attributes of a data)
- Output layer

  Contains r nodes (r – the number of outputs)
- Intermediate layers

  Different structures

  Different sizes

Example of a feed forward network structure:

6:4:10:2 → input layer with 6 units (artificial neurons), two hidden layers with 4 and 10 units respectively and an output layer with 2 units.

this structure can be used for a problem with 6 attributes and 2 outputs

each node from the first layer has one input and one output, each unit from the first hidden layer has 6 weights and one output, each unit from the second hidden layer has 4 weights and one output, each node from the output layer has 10 weights and one output.

Artificial Neural Networks

Learning process

A training data set of n data

$$\{((x_1^d, x_2^d, \dots, x_m^d), (t_1^d, t_2^d, \dots, t_r^d)) : d = 1, 2, \dots, n\}$$

where m – number of attributes and r – number of outputs

Form an ANN with m input nodes, r output nodes and an internal structure

- Some hidden layers, each layer having a given number of nodes
- With weighted connections between every 2 nodes of consecutive layers

Determine the optimal weights by minimising the error

- Difference between the real output y and the output computed by the network

Problems solved by an ANN

- Problem data can be represented by pairs (attribute-value)
- Objective function can be:
    - Single or multi-criteria
    - Discrete or continuous (real values)
- Training data can be noised
- A large training time

Neuron processing

Information is transmitted to the neuron → compute the weighted sum of inputs

$$net = \sum_{i=1}^{n} w_i * x_i$$

Neuron processes the information → by using an activation function

$$o = f(net)$$

- Constant function
- Step function
- Slope function
- Linear function
- Sigmoid function
- Gaussian function

Activation function

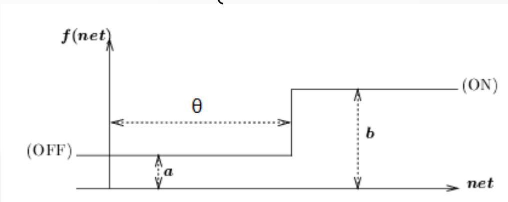
Constant function

$$f(net) = const.$$

Step function

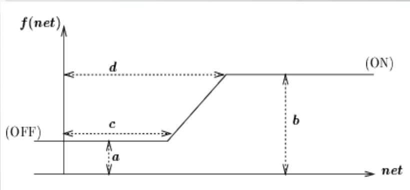
θ - threshold

$$f(net) = \begin{cases} a, & \text{if } net < \theta \\ b, & \text{if } net \geq \theta \end{cases}$$



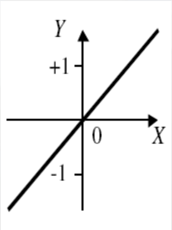
Slope function

$$f(net) = \begin{cases} a, & \text{if } net \leq c \\ b, & \text{if } net \geq d \\ a + \dfrac{(net-c)(b-a)}{d-c}, & \text{otherwise} \end{cases}$$



Linear function

$$f(net) = a * net + b$$

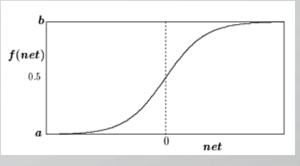


Sigmoid function

$$f(net) = z + \frac{1}{1 + \exp(-x \cdot net + y)}$$

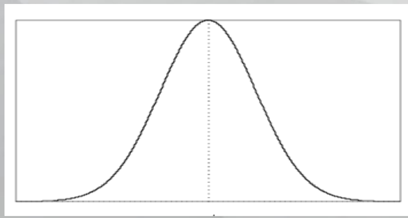
$$f(net) = \tanh(x \cdot net - y) + z$$

$$\text{where} \quad \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$



Gaussian function

$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2} \left(\frac{net - \mu}{\sigma} \right)^2 \right]$$



Attention! Also some other functions are in use – we will see this later!

Gradient descent

- based on the error associated to the entire set of train data
- modify the weights in the direction of the steepest slope of error reduction ***E(w)*** for the entire set of train data

$$E(\mathbf{w}) = \frac{1}{2}\sum_{d=1}^{n}(t^d - o^d)^2$$

- How to determine the steepest slope? → derive *E* based on *w* (establish the gradient of error *E*)

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_m}\right)$$

- Error's gradient is computed based on the neuron's activation function (that function must be differentiable → continuous)
  - linear function $f(net) = \sum_{i=1}^{m} w_i x_i^d$
  - sigmoid function $f(net) = \frac{1}{1+e^{-\mathbf{wx}}} = \frac{1}{1+e^{-\sum_{i=1}^{m} w_i x_i^d}}$
- How are modified the weights ? $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$, where $i = 1,2,\ldots,m$

Compute the error's gradient

- linear function

$$f(net) = \sum_{i=1}^{m} w_i x_i^d$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial \frac{1}{2}\sum_{d=1}^{n}\left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2}\sum_{d=1}^{n}\frac{\partial\left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2}\sum_{d=1}^{n} 2\left(t^d - o^d\right)\frac{\partial\left(t^d - \mathbf{w}\mathbf{x}^d\right)}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \sum_{d=1}^{n}\left(t^d - o^d\right)\frac{\partial\left(t^d - w_1 x_1^d - w_2 x_2^d - \ldots - w_m x_m^d\right)}{\partial w_i} = \sum_{d=1}^{n}\left(t^d - o^d\right)\left(-x_i^d\right)$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} = \eta\sum_{d=1}^{n}\left(t^d - o^d\right)x_i^d$$

- sigmoid function

$$f(net) = \frac{1}{1+e^{-\mathbf{w}\mathbf{x}}} = \frac{1}{1+e^{-\sum_{i=1}^{m} w_i x_i^d}} \qquad y = s(z) = \frac{1}{1+e^{-z}} \Rightarrow \frac{\partial s(z)}{\partial z} = s(z)\left(1-s(z)\right)$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial \frac{1}{2}\sum_{d=1}^{n}\left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2}\sum_{d=1}^{n}\frac{\partial\left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2}\sum_{d=1}^{n} 2\left(t^d - o^d\right)\frac{\partial\left(t^d - sig(\mathbf{w}\mathbf{x}^d)\right)}{\partial w_i} = \sum_{d=1}^{n}\left(t^d - o^d\right)\left(1-o^d\right)o^d\left(-x_i^d\right)$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} = \eta\sum_{d=1}^{n}\left(t^d - o^d\right)\left(1-o^d\right)o^d x_i^d$$

Simple and stochastic GDA

Simple GDA	Stochastic GDA
Initialisation of network weights $w_i = random(a,b)$, where $i = 1, 2, ..., m$ **While** not stop condition $\Delta w_i = 0$, where $i = 1, 2, ..., m$ **For each** train example $(\mathbf{x}_d, t_d)$, where $d = 1, 2, ..., n$ Activate the neuron and determine the output o_d Linear activation ➔ $o_d = \mathbf{w}\,\mathbf{x}_d$ Sigmoid activation ➔ $o_d = sig(\mathbf{w}\,\mathbf{x}_d)$ **For each** weight w_i, where $i = 1, 2, ..., m$ Determine the weight modification $\Delta w_i = \Delta w_i - \eta \frac{\partial E}{\partial w_i}$ where η - learning rate **For each** weight w_i, where $i = 1, 2, ..., m$ Modify the weights w_i $w_i = w_i + \Delta w_i$ **EndWhile**	Initialisation of network weights $w_i = random(a,b)$, where $i = 1, 2, ..., m$ **While** not stop condition $\Delta w_i = 0$, unde $i = 1, 2, ..., m$ **For** a random sample subset from the training data set $(\mathbf{x}_{d_i}, t_{d_i})$, where $d_i \in \{1, 2, ..., n\}$ Activate the neuron and determine the output o_{d_i} Linear activation ➔ $o_{d_i} = \mathbf{w}\,\mathbf{x}_{d_i}$ Sigmoid activation ➔ $o_{d_i} = sig(\mathbf{w}\,\mathbf{x}_{d_i})$ **For each** weight w_i, where $i = 1, 2, ..., m$ Determine the weight modification $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$ where η - learning rate **For each** weight w_i, where $i = 1, 2, ..., m$ Modify the weights w_i $w_i = w_i + \Delta w_i$ **EndWhile**

Neuron learning

Differences	Perceptron's algorithm	Gradient descent algorithm (delta rule)
What does o^d represent?	$o^d=\text{sign}(\mathbf{wx}^d)$	$o^d=\mathbf{wx}^d$ or $o^d=\text{sig}(\mathbf{wx}^d)$
Convergence	After a finite # of steps (until the perfect separation)	Asymptotic (to minimum error)
Solved problems	With linear separable data	Any data (linear separable or non-linear)
Neuron's output	Discrete and with threshold	Continue and without threshold

THANK YOU !