

Introduction to Artificial Neural Networks and Machine Learning

(Revised slides from HOU-PLH31)



I. Hatzilygeroudis

Dept of Computer Engineering & Informatics, University of Patras

Artificial Neural Networks-ANN

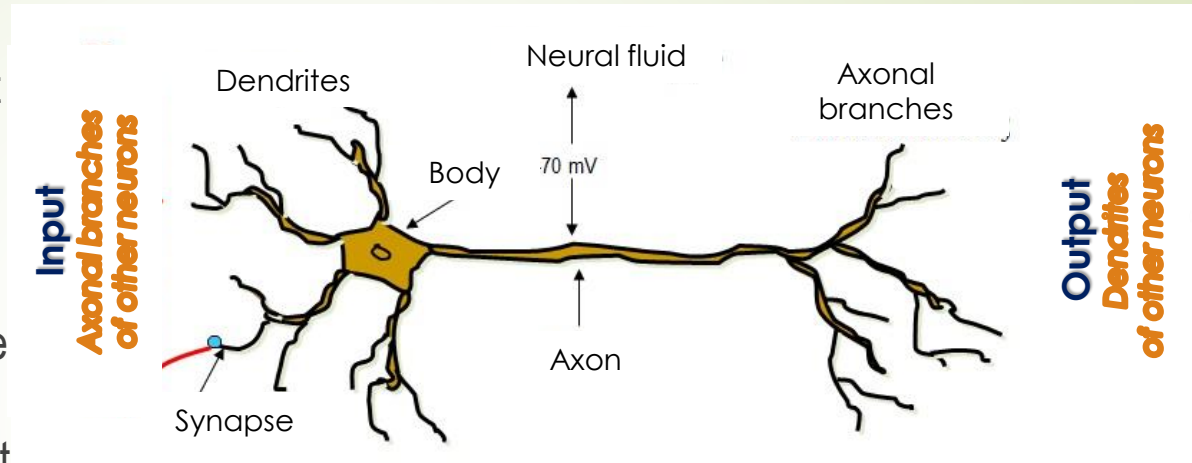
- ❑ ANN: arose from the need to make computational models of the human brain (biological neural networks)
- ❑ Problem: we still don't know (exactly) how the human brain works!
- ❑ 1950: simplified mathematical models of the brain.
- ❑ The first ANNs: computer simulation of these models (solving elementary problems)

Artificial Neural Networks-ANN

- ❑ The brain consists of a huge number of interconnected **neurons**, i.e. nerve cells.
- ❑ Every neuron
- ❑ receives stimuli (inputs) from other cells through connections that affect its state and, depending on the state it is in,
- ❑ it sends impulses (outputs) to affect in turn the state of other neurons.
- ❑ Each connection between two neurons is characterized by a **power value (synaptic potential)** which indicates how strong the interaction between them is.

Biological neuron

- ❑ **Dendrites**, which are the input lines of stimuli (biological signals)
- ❑ **Body**, in which the accumulation of stimuli and the determination of the excitation of the axon, which is the output line of the neuron.
- ❑ **Synapse**, which is the point of connection between two neurons.

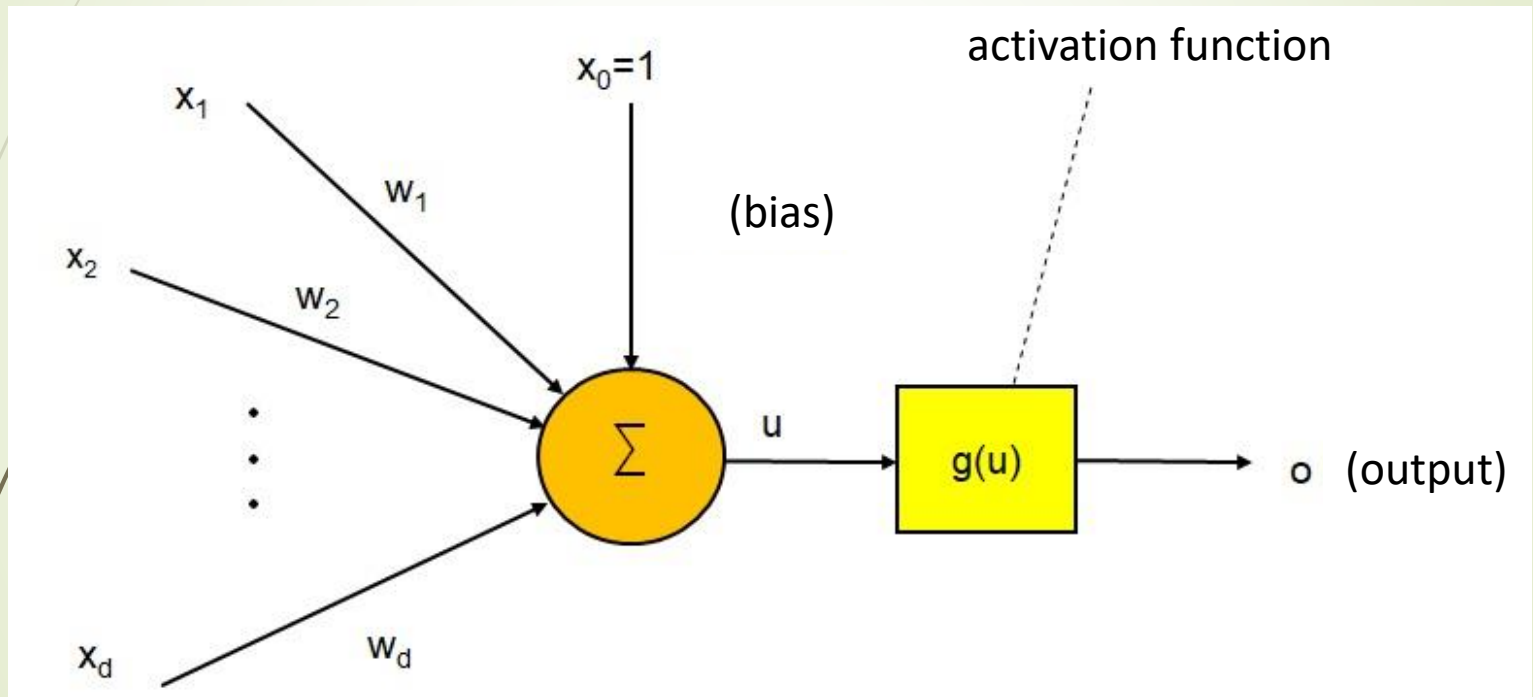


It has been observed that the signal exiting the axon of one neuron and entering the dendrite of the other neuron **is modulated** by a rate related to the strength of the synapse called the **synaptic potential**.

Synaptic Potential

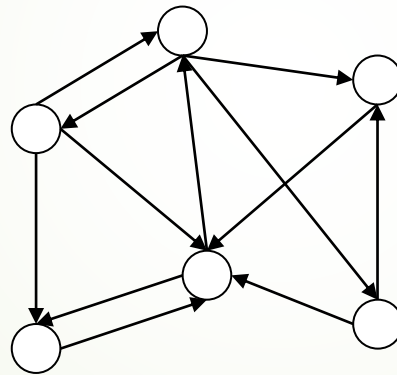
- The synaptic potential can amplify (positive) or suppress (negative) the output signal.
- Our knowledge is 'stored' in the values of synaptic potentials.
- Learning in biological systems is the change of synaptic potentials.**
- The more a synapse is used, the more its potential is enhanced.

Artificial Neuron



Artificial Neural Networks-ANN

- ❑ Architectural structures (**networks**) consisting of a number of interconnected **processing units** (artificial neurons).
- ❑ Each connection between two modules is characterized by a **weight** value.



- ❑ Each processing unit is characterized by **inputs** and **outputs**. Locally implements a **simple calculation** based on the inputs it receives and transmits the result (output) to other processing units it is connected to.



Artificial Neural Networks-ANN

- ❑ The **weight values** constitute the **knowledge** stored in the ANN and define its **functionality**.
- ❑ Usually a ANN develops an overall functionality through a form of **training** (learning)..

Capabilities of ANNs

- ❑ Basic abilities of the human brain
 - ✓ Learning by example
 - ✓ Ability to Generalize
 - ✓ Stores experiences (distributed storage)
 - ✓ Self-organization
 - ✓ Tolerance of noise and incomplete information
 - ✓ Damage tolerance
- ❑ The brain's capabilities are complementary to conventional computers
- ❑ Artificial Neural Networks also have (to some extent) the above capabilities

Machine Learning

❑ Training an ANN:

- ✓ Determining the weights of its links so as to perform a desired function which is described via examples

❑ Generalization Ability:

- ✓ The objective of the training process: to find appropriate weight values so that the ANN 'gives correct answers' for the examples that are 'similar' to the ones with which it was trained

- ❑ TNNs have proven to be a successful technology for developing systems with good generalization ability using a set of representative training examples.



Machine Learning Methods

- Supervised learning
- Unsupervised learning
- Reinforcement learning
- Semi-supervised learning

Supervised Learning

- ❑ The elements of the set of examples are pairs of the form: (input, desired output) ($X = \{(x^i, t^i)\}, i=1, \dots, N$).
- ❑ Each x^i is of the form $\langle v_{i1}, v_{i2}, \dots, v_{in} \rangle$ where $v_{ij} \ j = 1, n$ are values corresponding to features/properties of the problem represented by the data set.
- ❑ Qualitatively we can think of each pair as: (question/ x^i , correct answer/ t^i).
- ❑ The learning system implements **input-output correlations**
- ❑ When some data x^i appears as input, we want the ANN to provide the output with the corresponding desired value t^i .

Supervised Learning

- ❑ The term supervised learning derives from the analogue of 'supervisor':
 - ✓ supervises the learning process, asking questions and providing the correct answers at the same time.
- ❑ Suitable for two major categories of problems:
 - ✓ classification
 - t : class label
 - ✓ regression or function approximation
 - t : number (value)

Unsupervised Learning

- ❑ The training examples do not include the desired output but only the input data ($X=\{x^i\}$, $i=1,\dots,N$).
- ❑ The goal is to extract some basic structural properties of the training data (eg finding clusters).
- ❑ Problem Categories:
 - ✓ **Clustering:** dividing the training data into groups so that data in the same group are sufficiently 'similar' to each other and sufficiently 'different' from those of the other groups.
 - ✓ **Dimensionality reduction:** projection of the data into a smaller dimensional space in which the relative distances between the data in the original multidimensional space are preserved as much as possible
 - ✓ If the **dimension** of the projection space is **two** then the visualization of the original multidimensional data is possible.
 - ✓ **Topographic data map**

Reinforcement Learning

- ❑ In the learning system, the desired output is not provided for each input, but only the value of a quantity called the reinforcement signal ($X=\{x^i, r^i\}$, $i=1,\dots,N$).
- ❑ The reinforcement signal r indicates whether the system provided a response in the correct or incorrect direction but does not provide details of what the correct response is.
- ❑ Applications in robotics, games.

Artificial Neuron

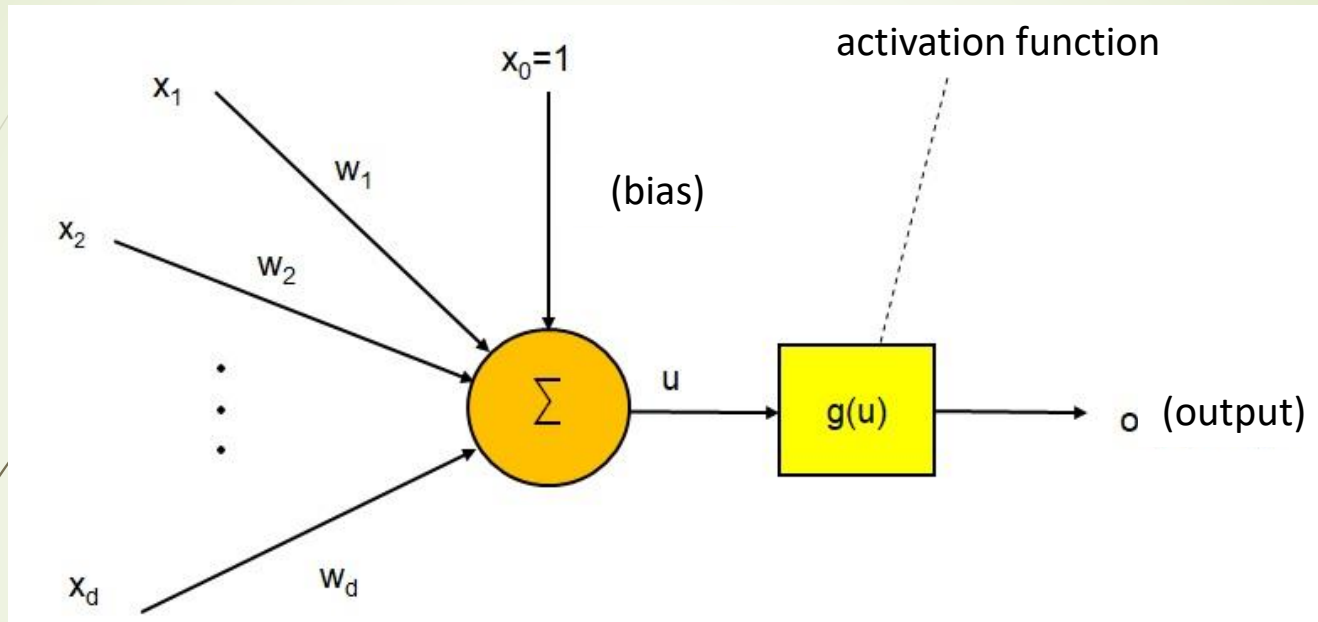
(Revised slides from HOU-PLH31)



I. Hatzilygeroudis

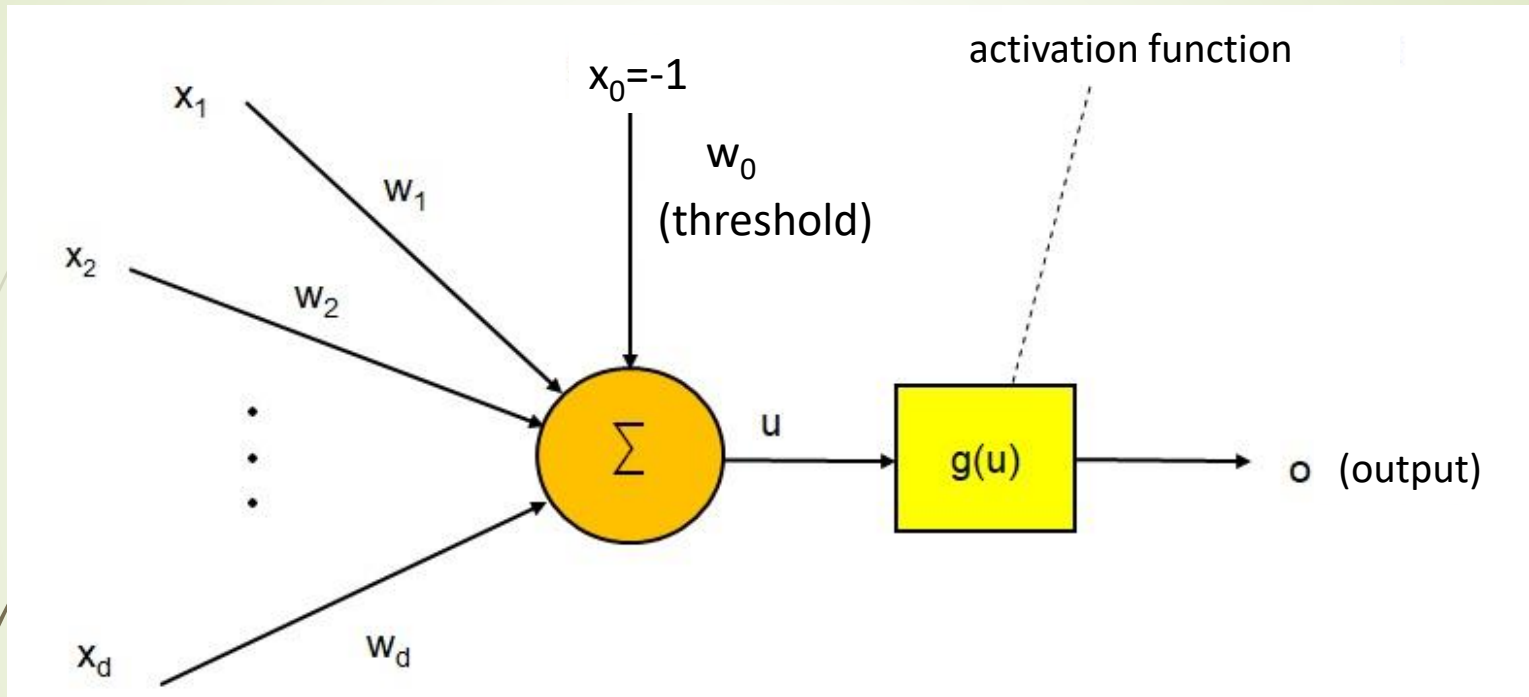
Dept of Computer Engineering & Informatics, University of
Patras

Artificial Neuron-Computational Model



- d inputs,
- Input signal x_i ($i=1, \dots, d$)
- Input weights w_i , ($i=1, \dots, d$)
- bias $w_0 \rightarrow$ weight value of the fixed input (input value = 1)

Artificial Neuron-Alternative Model



- d inputs,
- Input signal x_i ($i=1, \dots, d$)
- Input weights w_i , ($i=1, \dots, d$)
- threshold $w_0 \rightarrow$ weight value of the fixed input (input value = -1)

Artificial Neuron

Computation in two stages:

- ✓ calculation of total input (activation):

$$u(\mathbf{x}) = \sum_{i=1}^d w_i x_i + w_0$$

- ✓ Calculation of neuron output $o(\mathbf{x})$ by passing the total input $u(\mathbf{x})$ through an **activation function** $g(\cdot)$

$$o(\mathbf{x}) = g(u)$$

Neuron of internal product

$$u(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

Artificial Neuron

Alternative notation:

✓ Weights vector: $w = (w_1, w_2, \dots, w_d)^T$

✓ Extended weights vector:

$$w_e = (w_0, w_1, w_2, \dots, w_d)^T$$

✓ Extended input vector:

$$x_e = (1, x_1, x_2, \dots, x_d)^T$$

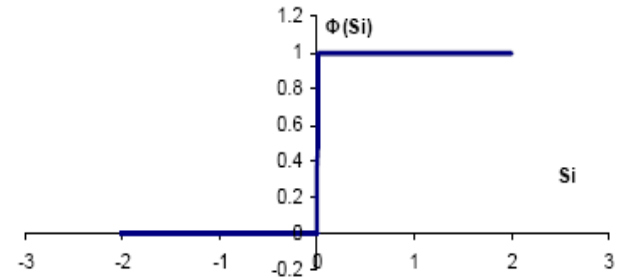
$$u(x) = w_e^T x_e \quad \Leftrightarrow \quad u(x) = w^T x + w_0$$

Activation Functions

Step or Threshold function:

- ✓ The activation function in the biological neuron
- ✓ It is characterized by two values **a** and **b**.
- ✓ If **$x < 0$** then **$g(x) = a$** and if **$x > 0$** then **$g(x) = b$** . Usually the values $a = 0$ and $b = 1$ or $a = -1$ and $b = 1$ are used.
- ✓ **The step function has the disadvantage that its derivative is zero**

Since learning in ANNs is the change of weight values and change is related to the derivative, the step function is not convenient as the activation function of neurons in ANNs.



Activation Functions

Sigmoid functions

- ✓ They have a sigma form
- ✓ They are continuous and derivative approximations of step function.
- ✓ At the limit where the slope becomes too large, the sigmoid becomes the step function.
- ✓ **Two basic types:**

1) Logistic:

$$\sigma(\mathbf{x}) = 1 / (1 + \exp(-a\mathbf{x}))$$

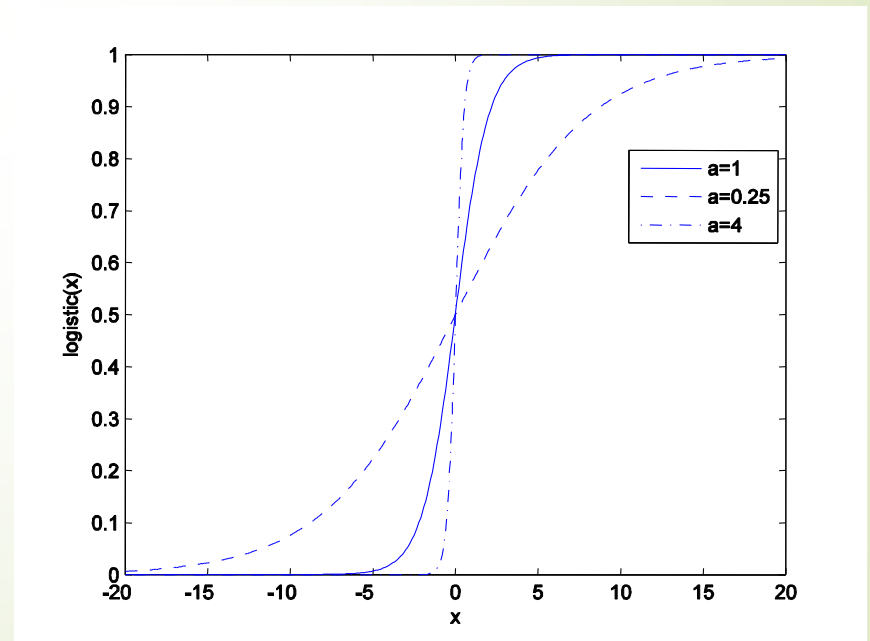
(a: slope, usually a=1)

give values in (0,1)

$$\sigma'(\mathbf{x}) = \sigma(\mathbf{x})(1 - \sigma(\mathbf{x})) \quad (\text{for } a=1)$$

$$\sigma''(\mathbf{x}) = \sigma(\mathbf{x})(1 - \sigma(\mathbf{x}))(1 - 2\sigma(\mathbf{x}))$$

We can calculate the derivative $\sigma'(x)$ with only $\sigma(x)$ given, without needing the value of x .



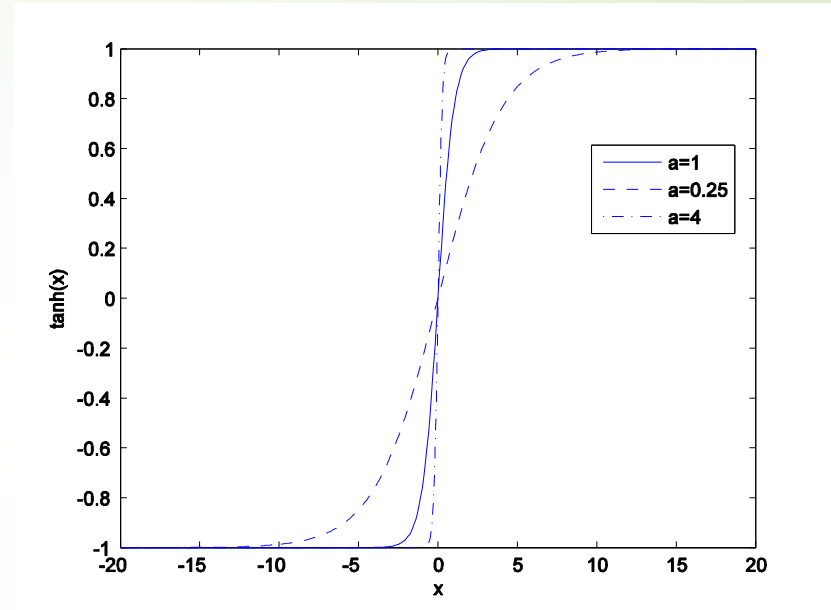
Activation Functions

2) Hyperbolic tangent:

$$\tanh(x) = \frac{e^{ax} - e^{-ax}}{e^{ax} + e^{-ax}}$$

(a: slope, usually a=1)
gives values in (-1,1)

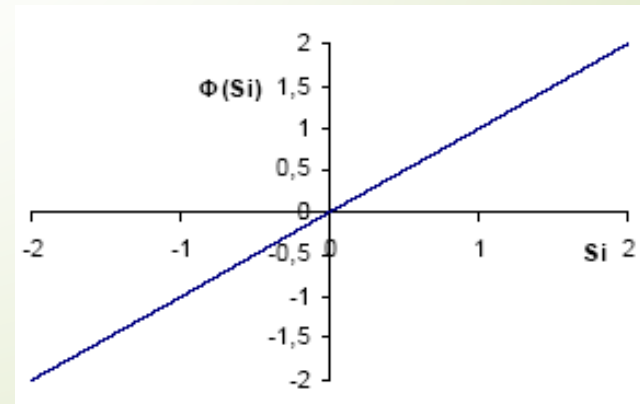
$$\tanh'(x) = 1 - \tanh^2(x) \text{ (for } a=1\text{)}$$



Linear function

$$g(x) = x, \quad g'(x) = 1$$

gives values in \mathbb{R}



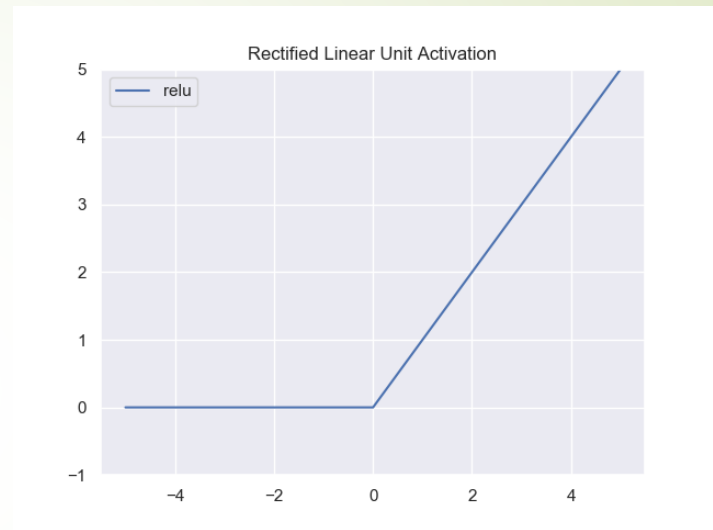
Activation Functions

Rectified linear unit (Relu)

$$g(x) = \max(0, x)$$

□ Properties:

- ✓ Nonlinear
- ✓ $(g(-1) + g(1) = 1, g(-1+1) = 0)$
- ✓ Less activations-
computationally favorable
- ✓ Ideal for deep networks



Disadvantages:

- ✓ It is not upper bounded.
- ✓ For negative parameters, the local derivative is 0, so they do not respond during training (dying Relu problem).

Perceptron

- ❑ Perceptron is the simplest form of Neural Network, which is used to classify linearly separable patterns, following the McCulloch – Pitts model.
- ❑ Uses the **sign function** as an activation function:

$$\text{sgn}(u) = \begin{cases} +1 & \text{if } u \geq 0 \\ -1 & \text{if } u < 0 \end{cases}$$

Training Perceptron

1. Initialization

We initialize the weights and threshold with values in the range $[-0.5, 0.5]$

2. Activation-Output computation

$$y(n) = g(u(n)) \quad u(n) = \sum_{i=0}^m w_i(n) x_i(n) \quad \text{or} \quad u(n) = \mathbf{W}(n)^T \times \mathbf{X}(n)$$

3. Weights adaptation

$$w_i(n+1) = w_i(n) + \Delta w_i(n) \quad \Delta w_i(n) = \eta x_i(n) e(n) \quad e(n) = d(n) - y(n)$$

$$w_i(n+1) = w_i(n) + \eta [d(n) - y(n)] x_i(n)$$

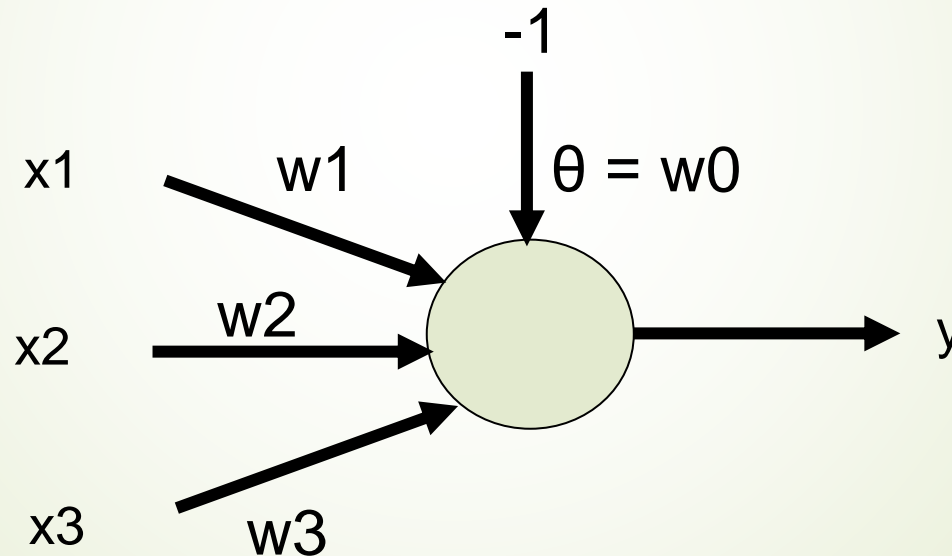
4. Testing-Iteration (from step 2)

$$e(n) < \varepsilon \quad (\varepsilon = 10^{-1} - 10^{-4})$$

Training Perceptron-Example

We have the following three-input neuron. We use the Perceptron algorithm to solve the following simple pattern classification problem:

$$X1 = [1, -1, 1]^T \rightarrow d1 = 0 \text{ και } X2 = [1, 1, -1]^T \rightarrow d2 = 1$$



Consider weights w_1, w_2, w_3 having as initial values: $[0.5, -1, -0.5]$ and threshold $\theta = w_0 = -0.5$. Also, $\eta = 1$.

Παράδειγμα εκπαίδευσης Perceptron

- We assume that we are at step k , set as input the vector $X1$ and calculate the total input

$$u(k+1) = \mathbf{w}^T(k) * X1$$

where $w = [-0.5, 0.5, -1, -0.5]$ and $X1 = [-1, 1, -1, 1]$

We use extended vectors, where the first values correspond to the threshold input. So

$$u(k+1) = \begin{bmatrix} -0.5 \\ 0.5 \\ -1 \\ -0.5 \end{bmatrix} * [-1, 1, -1, 1] = 1.5, \text{ and } y(k+1) = \text{sgn}(1.5) = +1$$

- Update the weights, after we have calculated the error

$$e = d1 - y(k+1) = 0 - 1 = -1$$

Παράδειγμα εκπαίδευσης Perceptron

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) + \eta \cdot e \cdot X_1 \\ &= [-0.5, 0.5, -1, -0.5] + 1(-1)[-1, 1, -1, 1] = [0.5, -0.5, 0, -1.5] \end{aligned}$$

- We proceed to the next step $k+2$, setting as input the vector X_2 and calculate the output

$$u(k+2) = \mathbf{w}^T(k+1) \cdot X_2 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0 \\ -1.5 \end{bmatrix} \cdot [-1, 1, 1, -1] = 0.5 > 0$$

$$\text{and } y(k+2) = \text{sgn}(0.5) = +1$$

- Update the weights, after we have calculated the error

$$e = d_2 - y(k+2) = 1 - 1 = 0$$

Since $e=0$ there is no need to update the weights.

Παράδειγμα εκπαίδευσης Perceptron

- We proceed to the next step $k+3$, setting as input the vector X_1 and calculate the output

$$u(k+3) = \mathbf{w}^T(k+2) \cdot X_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0 \\ -1.5 \end{bmatrix} \cdot [-1, 1, -1, 1] = -2.5$$

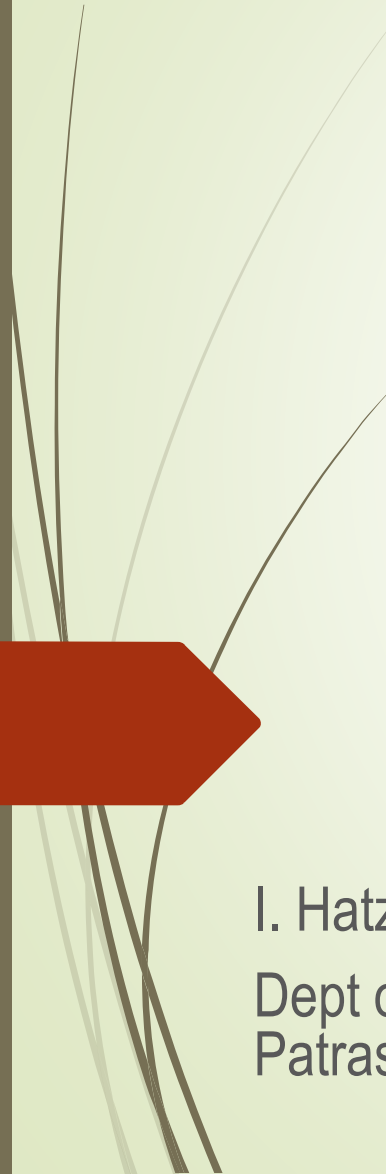
$$\text{and } y(k+3) = \text{sgn}(-2.5) = -1$$

- Update the weights, after we have calculated the error

$$e = d_1 - y(k+3) = 0 - (-1) = 1$$

$$\begin{aligned} \mathbf{w}(k+3) &= \mathbf{w}(k+2) + \eta \cdot e \cdot X_1 \\ &= [0.5, -0.5, 0, -1.5] + 1 \cdot 1[-1, 1, -1, 1] = [-0.5, 0.5, -1, -0.5] \end{aligned}$$

and so on ...



ANN Training based on minimization of squared training error

(Revised slides from HOU-PLH31)



I. Hatzilygeroudis

Dept of Computer Engineering & Informatics, University of
Patras

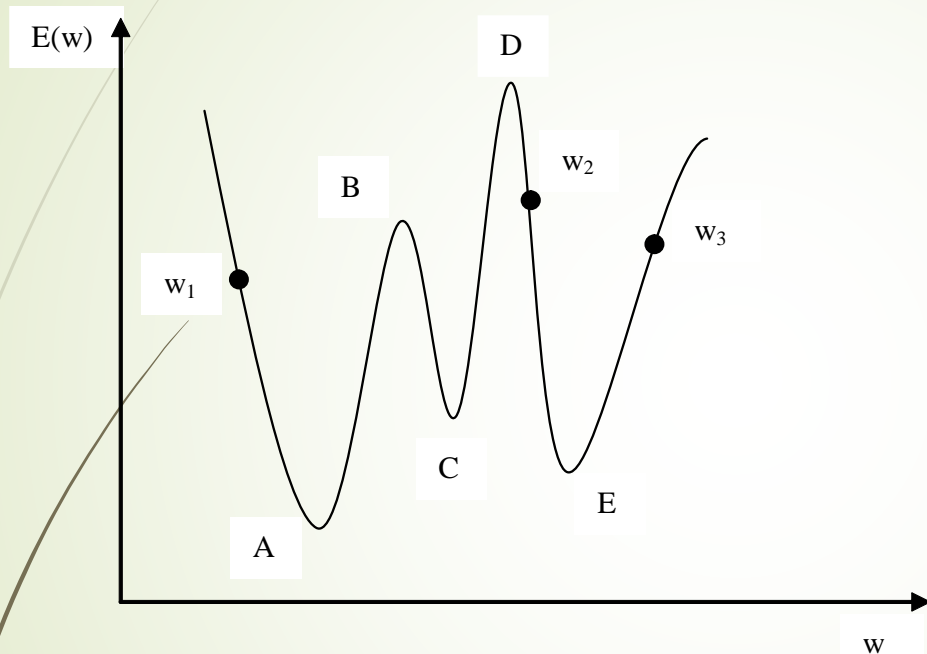
Minimization of error function

- ❑ ANN training: can be formulated as a problem of **minimizing an error function $E(w)$ with respect to the vector $w=(w_1, \dots, w_L)$** of ANN parameters (weights and biases).
- ❑ Usually what is needed is the calculation of the **partial derivatives $\partial E / \partial w_i$** of the error with respect to the ANN parameters
- ❑ Many efficient **numerical minimization methods** are based on partial derivatives
- ❑ Most popular method for ANNs: **gradient descent**
- ❑ It is also the simplest

Minimization of error function

- ❑ Let $E(w)$ be an error function that we want to minimize with respect to w :
 - ✓ to find a **point** w^* of **minimum value**, at which the function $E(w^*)$ becomes minimal.
- ❑ The **extrema** of a function satisfy the condition that $\partial E / \partial w_i = 0$ for each $i=1, \dots, L$.
- ❑ A function can have more than one minima called **local minima**.
- ❑ The best (the one with the smallest value) of the local minima is called the **global minimum**.

Minimization of error function



Three local minima:
(A, C, E)
Global minimum: A

- **Analytical finding** of minima: solution of the system of equations $\partial E / \partial w_i = 0$, $i=1, \dots, L$. Only possible when $E(w)$ is quadratic.
- We resort to **numerical analysis methods** (iterative)

Minimization of error function

Iterative methods:

- start from an initial value (usually random) $w^{(0)}$.
- at each iteration t the vector of weights is modified by $\Delta w(t)$:

$$w(t+1) = w(t) + \Delta w(t)$$

so that the function value decreases:

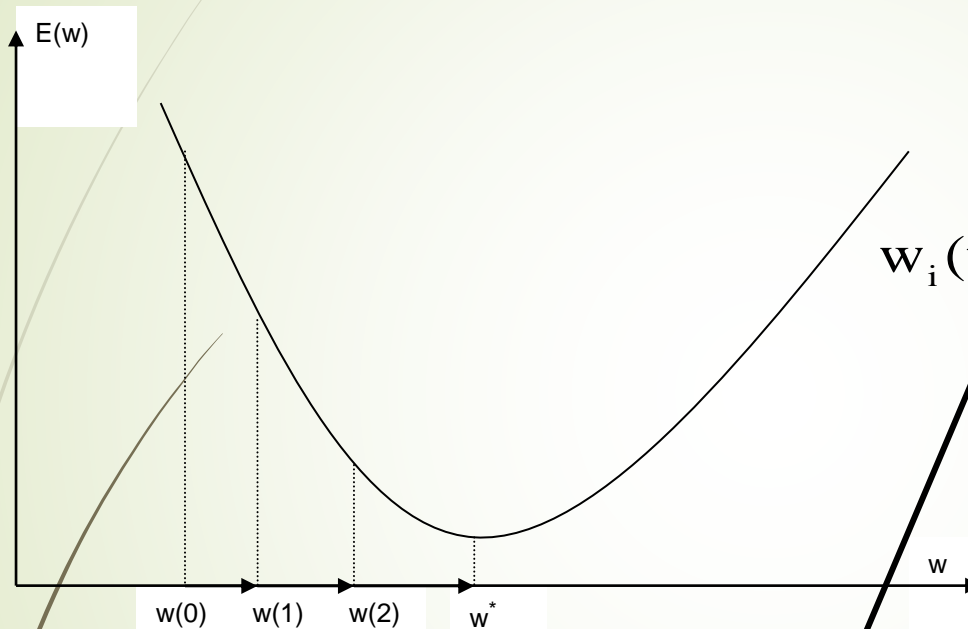
$$E(w(t+1)) \leq E(w(t)).$$

- Optimization algorithms differ in how they calculate the change $\Delta w(t)$.
- Information related to the slope of the function is usually used.
- The iterative process converges to a local minimum w^* of function $E(w)$.

Minimization of error function

- ❑ The methods implement local minimization.
- ❑ If the function $E(w)$ has many local minima, the minimum the method will arrive at depends on the initial value of the vector $w^{(0)}$ (which is usually chosen randomly).
- ❑ There is a possibility of 'locking' into unwanted (high-valued) local minima of the error function.
- ❑ A simple solution: multiple runs from different initial values. We keep the best of the solutions we find.

Gradient descent (GD)



$$w_i(t+1) = w_i(t) - \eta \frac{\partial E}{\partial w_i}, \quad i=1, \dots, L$$

- ❑ We start from an initial value of the weights $w_i(0)$ (usually random).
- ❑ At each iteration t :
 - ✓ Calculate the slope and **updating w_i**
 - ✓ Check for termination of the method
 - ✓ If positive, terminate it, else $t:=t+1$ and continue.

Learning rate

η : it is called **descent step**

- ✓ In the case of ANN training, it is called the **learning rate**.
- ✓ Determines whether to move in the decreasing direction of the function with small or large steps.
- ✓ A small learning rate implies a smooth descent to the local minimum, but more iterations are required.
- ✓ A large learning rate implies a faster descent (bigger steps, fewer iterations), but also an increased probability of **oscillations** around the minimum point.

Training of single neuron based on error minimization

- Set of training examples $D = \{(x^n, t^n)\}$, $n = 1, \dots, N$
- $x^n = (x_{n1}, \dots, x_{nd})^T$ and t^n number
- Training single neuron with weights $w = (w_0, w_1, \dots, w_d)^T$ and activation function $g(u)$.
- For input x^n : $u(x^n; w) = \sum_i w_i x_i + w_0$, $o(x^n; w) = g(u(x^n; w))$
- In the case that for some vector of weights the training is perfect:
- $o(x^n; w) = t^n$ for each $n = 1, \dots, N$
- that is, the output of the neuron for input x^n will be equal to the desired t^n .

Training of single neuron based on error minimization

- ✓ Hence, we can define the **quadratic training error function**:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (t^n - o(\mathbf{x}^n; \mathbf{w}))^2 \iff E(\mathbf{w}) = \sum_{n=1}^N E^n(\mathbf{w}), E^n(\mathbf{w}) = \frac{1}{2} (t^n - o(\mathbf{x}^n; \mathbf{w}))^2$$

- ✓ As a sum of squares, it has as a lower bound the value zero, which results when we have perfect training.
- ✓ The most important class of ANN training methods for supervised learning results from **updating the vector of weights \mathbf{w} aiming to minimize the squared error $E(\mathbf{w})$** .
- ✓ Most widely used minimization method: **gradient descent**

✓ .

Partial Derivative of training error

$$E(\mathbf{w}) = \sum_{n=1}^N E^n(\mathbf{w}), \quad E^n(\mathbf{w}) = \frac{1}{2} (t^n - o(\mathbf{x}^n; \mathbf{w}))^2 \quad \frac{\partial E}{\partial w_i} = ?$$

$$\frac{\partial E}{\partial w_i} = \sum_{n=1}^N \frac{\partial E^n}{\partial w_i} \quad \frac{\partial E^n}{\partial w_i} = -(t^n - o(\mathbf{x}^n; \mathbf{w})) \frac{\partial o(\mathbf{x}^n; \mathbf{w})}{\partial w_i}$$

$$\frac{\partial o(\mathbf{x}^n; \mathbf{w})}{\partial w_i} = \frac{\partial g(u)}{\partial u} \frac{\partial u(\mathbf{x}^n; \mathbf{w})}{\partial w_i} = g'(u) x_{ni}, \quad i=0, \dots, d, \quad x_{n0} = 1$$

$$\frac{\partial E^n}{\partial w_i} = -(t^n - o(\mathbf{x}^n; \mathbf{w})) g'(u(\mathbf{x}^n; \mathbf{w})) x_{ni}, \quad i=0, \dots, d, \quad x_{n0} = 1$$

$$\frac{\partial E}{\partial w_i} = - \sum_{n=1}^N (t^n - o(\mathbf{x}^n; \mathbf{w})) g'(u(\mathbf{x}^n; \mathbf{w})) x_{ni}, \quad i=0, \dots, d, \quad x_{n0} = 1$$

Partial Derivative of training error

- Computation of the partial derivative corresponding to the error for a training example (x^n, t^n) :
 - ✓ application of x^n as input to the neuron and computation of the total input $u(x^n; w)$ and the output $o(x^n; w)$
 - ✓ calculation of the **error**: $\delta^n = (t^n - o(x^n; w))$
 - ✓ calculation of partial derivatives with respect to w_i

$$\frac{\partial E^n}{\partial w_i} = -(t^n - o(x^n; w))g'(u(x^n; w))x_{ni}, \quad i=0, \dots, d, \quad x_{n0} = 1$$

Single neuron training with gradient descent (batch update)

1. Initialization: We set $k=0$, initial values of weights $w(0)$ and set the value of the learning rate η .
2. At each iteration k , let $w(t)$ be the vector of weights.
 - We initialize: $\frac{\partial E}{\partial w_i} = 0$, $i=0, \dots, L$
 - For $n=1, \dots, N$:
 - ✓ apply x^n as input to the neuron and compute the total input $u(x^n; w)$ and the output $o(x^n; w)$
 - ✓ calculation of the error: $\delta^n = (t^n - o(x^n; w))$.
$$\frac{\partial E}{\partial w_i} := \frac{\partial E}{\partial w_i} - \delta^n g'(u(x^n; w)) x_{ni}, \quad i=0, \dots, d, \quad x_{n0} = 1$$
 - We update the values of the weights: $w_i(t+1) = w_i(t) - \eta \frac{\partial E}{\partial w_i}$, $i=1, \dots, L$
3. We check for termination of the method. If positive, process is terminated.
4. $k:=k+1$, go to step 2.

Single neuron training with gradient descent (batch update)

- ❑ Batch update: the weights are updated once at the end of each epoch based on the partial derivative of the total error, i.e. summing the partial derivatives of the individual errors.
- ❑ The iteration counter t counts the epochs. An epoch is considered the passage of all examples of the training set.
- ❑ Batch update corresponds to the mathematically rigorous implementation of the gradient descent method to minimize the error $E(w)$:

$$w_i(t+1) = w_i(t) - \eta \frac{\partial E}{\partial w_i}, \quad i=1, \dots, L$$

- ❑ At each epoch t the error $E(w)$ should decrease (if the learning rate is sufficiently small)


Single neuron training with gradient descent (sequential update)

- ❑ The function $E(w)$ that we want to minimize has the following useful property: it is expressed as the **sum of the individual errors $E^n(w)$** .
- ❑ An alternative approach to minimizing $E(w)$:
 - ✓ At each iteration τ (i.e. after passing each example) we apply the gradient descent update rule to minimize **some of the individual errors $E^n(w)$** :

$$w_i(\tau+1) = w_i(\tau) + n(t^n - o(x^n; w))g'(u(x^n; w))x_{ni}, \quad i=0, \dots, d, \quad x_{n0} = 1$$

Single neuron training with gradient descent (sequential update)

- ❑ It turns out that if all terms $E^n(w)$ are chosen equally often, then the final result of the method is the minimization of the total error $E(w)$
- ❑ that is, operating at each step in the direction of reducing one term, we achieve in the end the reduction of the sum of the terms.
- ❑ This fact should not be taken as something obvious since at each step changing the weights to reduce the term $E^n(w)$ does not necessarily reduce the total error $E(w)$ because there may be other terms $E^m(w)$ which increase with the change of weights.



Single neuron training with gradient descent (sequential update)

- ❑ The above procedure is called stochastic gradient descent or on-line gradient descent or sequential gradient descent.
- ❑ We will call it the gradient descent method with **sequential updating** of the weights.
- ❑ While in batch updating we have one update of the weights per epoch (training cycle), in serial updating we have N updates.

Training of linear neuron

The linear neuron has an activation function $g(u)=u$, hence $g'(u)=1$.

- **Batch update:** $w_i(k+1)=w_i(k)+n \sum_{n=1}^N (t^n - o(x^n; w)) x_{ni}$, $i=0, \dots, d$, $x_{n0} = 1$
- **Sequential update:** $w_i(\kappa+1)=w_i(\kappa)+n(t^n - o(x^n; w))x_{ni}$, $i=0, \dots, d$, $x_{n0} = 1$
- Av $\delta^n = t^n - o(x^n; w)$: $w_i(\kappa+1) = w_i(\kappa) + n \delta^n x_{ni}$
(delta rule)