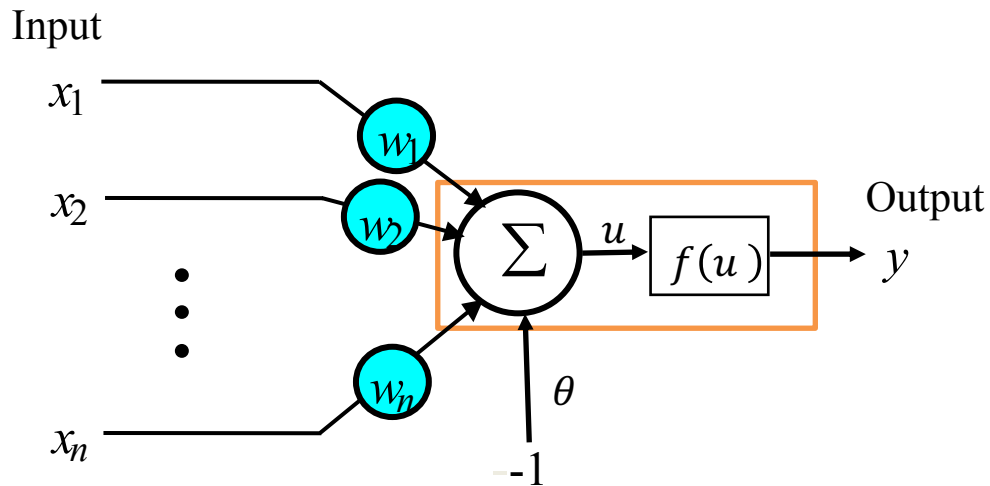


Chapter 1

Introduction to Neural Networks

Neural networks and deep learning

Artificial Neuron



Input vector $\mathbf{x} = (x_1 \ x_2 \ \cdots \ x_n)^T$
weight vector $\mathbf{w} = (w_1 \ w_2 \ \cdots \ w_n)^T$
 n is the number of inputs.

Artificial Neuron

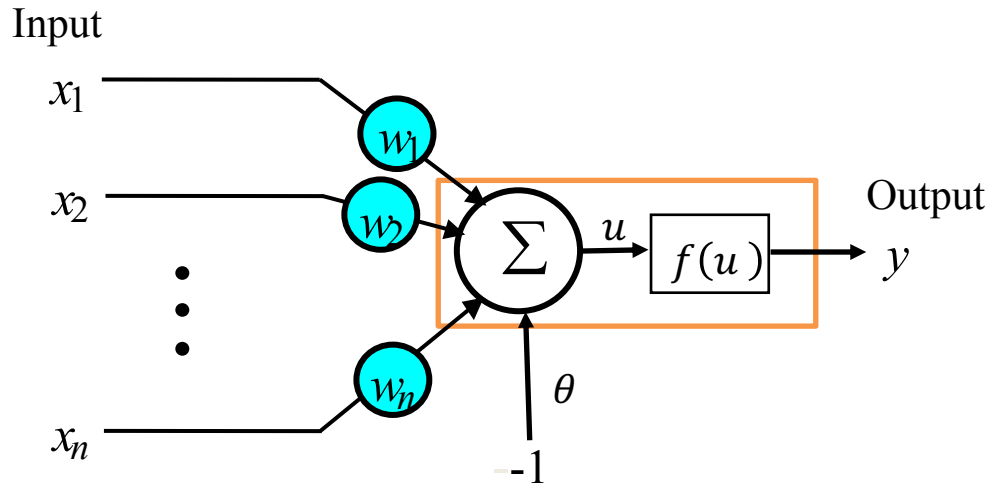
An **artificial neuron** is the basic unit of neural networks.

Basic elements of an artificial neuron:

- A set of **input** signals: the input is a vector $\mathbf{x} = (x_1 \ x_2 \ \cdots \ x_n)^T$ where n is the number (or the dimension) of input signals. Inputs are also referred to as **features**.
- Inputs are connected to the neuron via synaptic connections whose strengths are represented by their **weights**.
- The weight vector $\mathbf{w} = (w_1 \ w_2 \ \cdots \ w_n)^T$ where w_i is the synaptic weight connecting i th input to the neuron.

Notation: vectors are denoted in bold and written as horizontally with a transpose (T).

Artificial Neuron



The total **synaptic input** u to the neuron is given by the sum of the products of the inputs and their corresponding connecting weights minus the **threshold** of the neuron.

The total synaptic input to a neuron, u is given by

$$u = w_1x_1 + w_2x_2 + \cdots w_nx_n - \theta = \sum_{i=1}^n w_ix_i - \theta$$

By using vector notations:

$$u = \mathbf{x}^T \mathbf{w} - \theta$$

where θ is the threshold of the neuron.

Artificial Neuron

The **activation function** f relates synaptic input to the activation of the neuron.

$f(u)$ denotes the **activation** of the neuron.

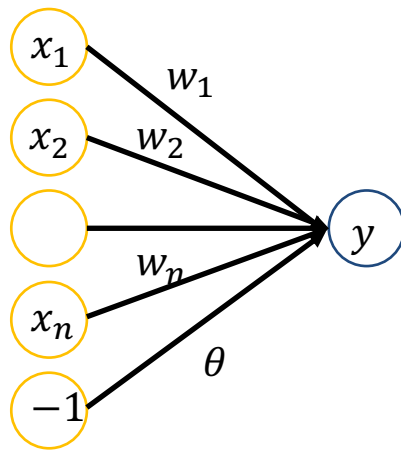
For some neurons, the **output** y is equal to the activation of the neuron.

$$y = f(u)$$

Note that activation is not generally equal to the output.

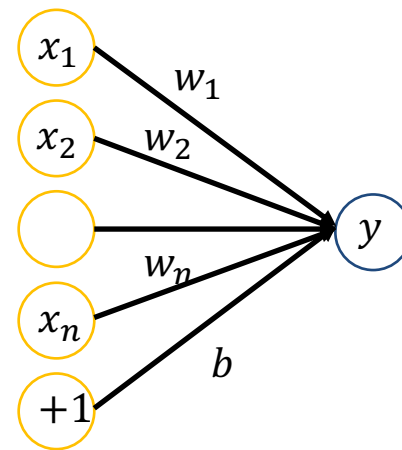
Bias vs. Threshold

The threshold is often considered as a weight with an input of -1 . Often the threshold is represented as a **bias** that receives constant $+1$ input.



Threshold

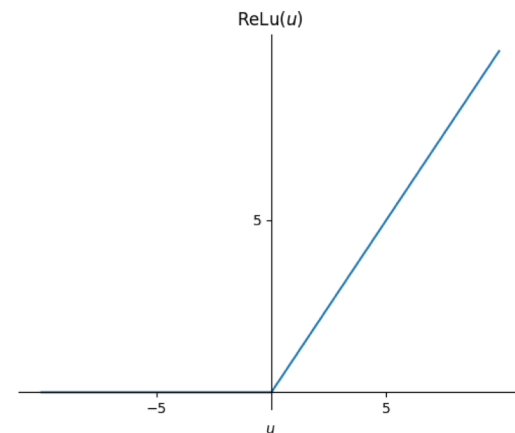
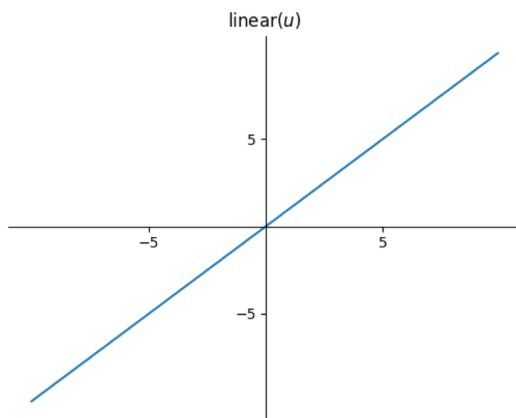
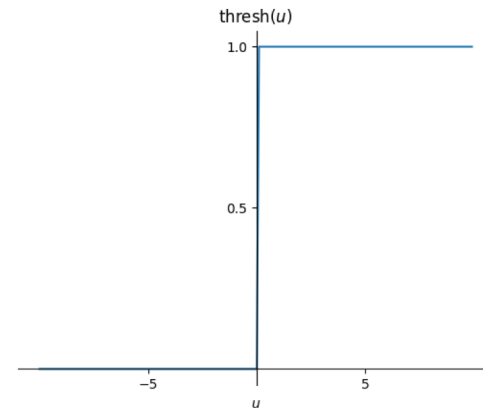
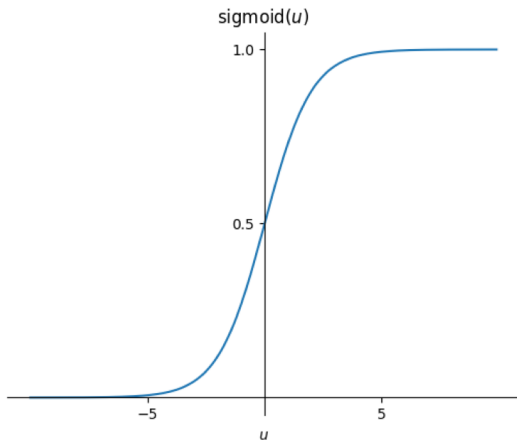
$$u = \mathbf{x}^T \mathbf{w} - \theta$$
$$y = f(u)$$



Bias

$$u = \mathbf{x}^T \mathbf{w} + b$$
$$y = f(u)$$

Activation functions



Activation functions

For ***threshold (unit step) activation function***, the activation is given by

$$f(u) = \text{threshold}(u) = \begin{cases} 1.0, & u > 0 \\ 0, & u \leq 0 \end{cases}$$

A neuron with ***linear activation function*** can be written as

$$f(u) = \text{linear}(u) = u$$

The ***ReLU activation function*** can be written as

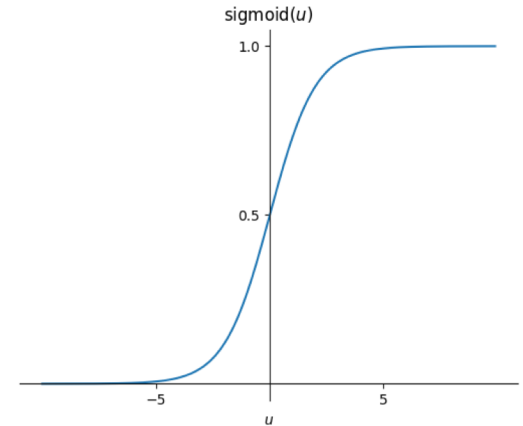
$$f(u) = \text{relu}(u) = \max\{0, u\}$$

Sigmoid activation function

For *sigmoid* activation function, the activation:

$$f(u) = \frac{a}{1 + e^{-bu}}$$

a is the gain and b is the slope.

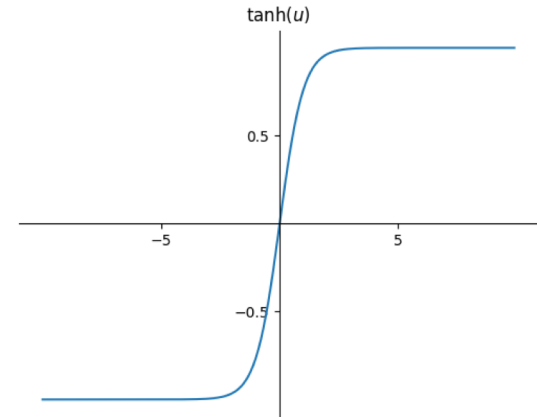


When $a = 1.0$ and $b = 1.0$, the sigmoidal is known as the **logistic function** or simply **sigmoid function**

$$f(u) = \text{sigmoid}(u) = \frac{1}{1 + e^{-u}}$$

Tanh activation function

$$f(u) = \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$



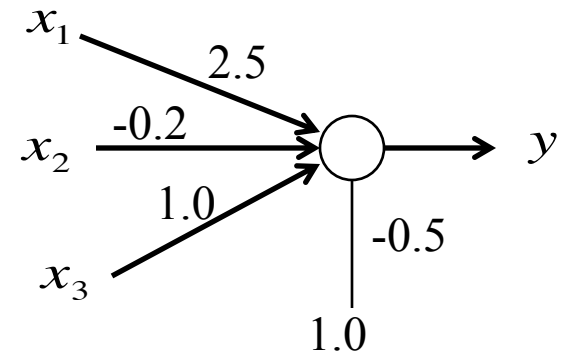
Tanh activation function has the same shape as sigmoidal and spans from -1 and +1. It is also known as **bipolar sigmoidal**.

Sigmoidal is the most pervasive and biologically plausible activation function. Since sigmoid function is *differentiable*, it leads to mathematically attractive neuronal models.

Example 1

The artificial neuron in the figure receives 3-dimensional inputs $\mathbf{x} = (x_1 \ x_2 \ x_3)^T$ and has an activation function given by

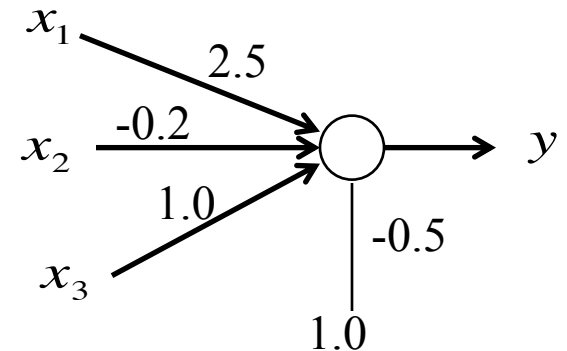
$$f(u) = \frac{0.8}{1 + e^{-1.2u}}.$$



Find the synaptic input and the output of the neuron for inputs: $\begin{pmatrix} 0.8 \\ 2.0 \\ -0.5 \end{pmatrix}$ and $\begin{pmatrix} -0.4 \\ 1.5 \\ 1.0 \end{pmatrix}$.

Example 1

$$\mathbf{w} = \begin{pmatrix} 2.5 \\ -0.2 \\ 1.0 \end{pmatrix}, \quad b = -0.5$$



Consider $\mathbf{x} = \begin{pmatrix} 0.8 \\ 2.0 \\ -0.5 \end{pmatrix}$

$$\text{Synaptic input } u = \mathbf{x}^T \mathbf{w} + b = (0.8 \quad 2.0 \quad -0.5) \begin{pmatrix} 2.5 \\ -0.2 \\ 1.0 \end{pmatrix} - 0.5 = 0.6$$

$$\text{Output} = y = f(u) = \frac{0.8}{1 + e^{-1.2u}} = \frac{0.8}{1 + e^{-1.2 \times 0.6}} = 0.538$$

Similarly, for $\mathbf{x} = \begin{pmatrix} -0.4 \\ 1.5 \\ 1.0 \end{pmatrix}$, $u = -0.8$ and output $y = f(u) = 0.222$

Tensorflow

Tensorflow is about processing of **tensors**.

Tensor is a multidimensional array. **Rank** refers to the number of dimensions and **shape** gives the size of each dimension

3. # a rank 0 tensor; a scalar with shape [],

[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]

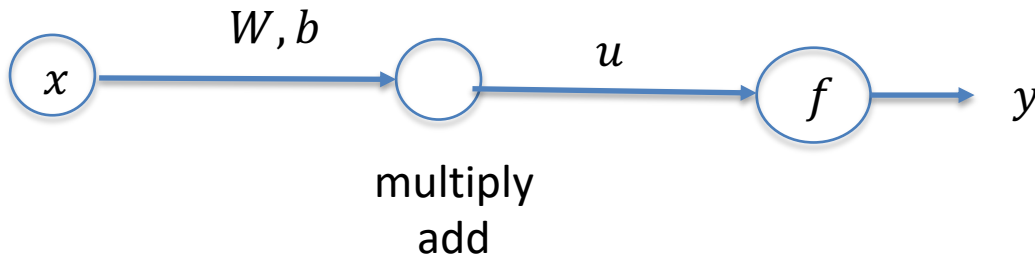
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]

[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]

Tensorflow executes a program in two phases:

- Building the computational graph
- Evaluating the computational graph

Computational graph of a neuron



$$u = \mathbf{x}^T \mathbf{w} + b$$
$$y = f(u)$$

A **computational graph** is a series of tensorflow operations arranged into a graph.

- Nodes of the graph represent tensorflow operations
- Edges represent values (tensors) that follow through the graph

Tensorflow Implementation of Example 1

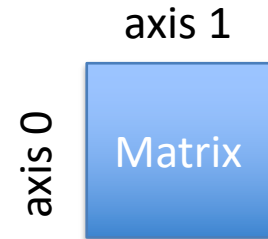
```
import tensorflow as tf

# building the computational graph
w = tf.Variable([2.5, -0.2, 1.0], tf.float32)
b = tf.Variable(-0.5, tf.float32)

x = tf.placeholder(tf.float32)
u = tf.tensordot(w, x, axes=1) + b
y = 0.8/(1+tf.exp(-1.2*u))

# Evaluating the computational graph
sess = tf.Session()
sess.run(tf.global_variables_initializer())

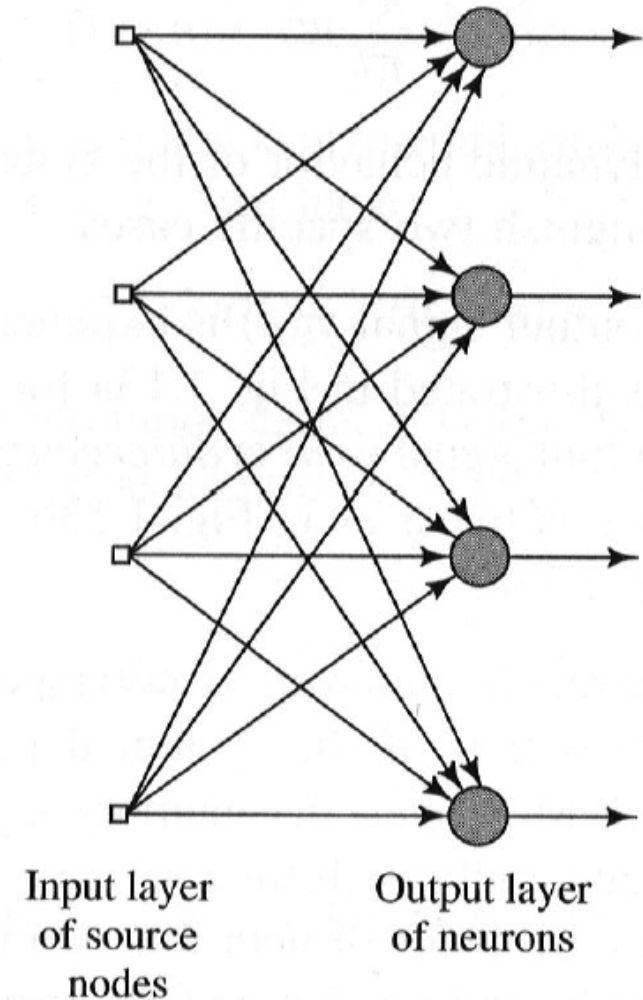
U, Y = sess.run([u, y], {x: [0.8, 2.0, -0.5]})
print(U, Y)
```



ANN Architectures

Single – layer networks

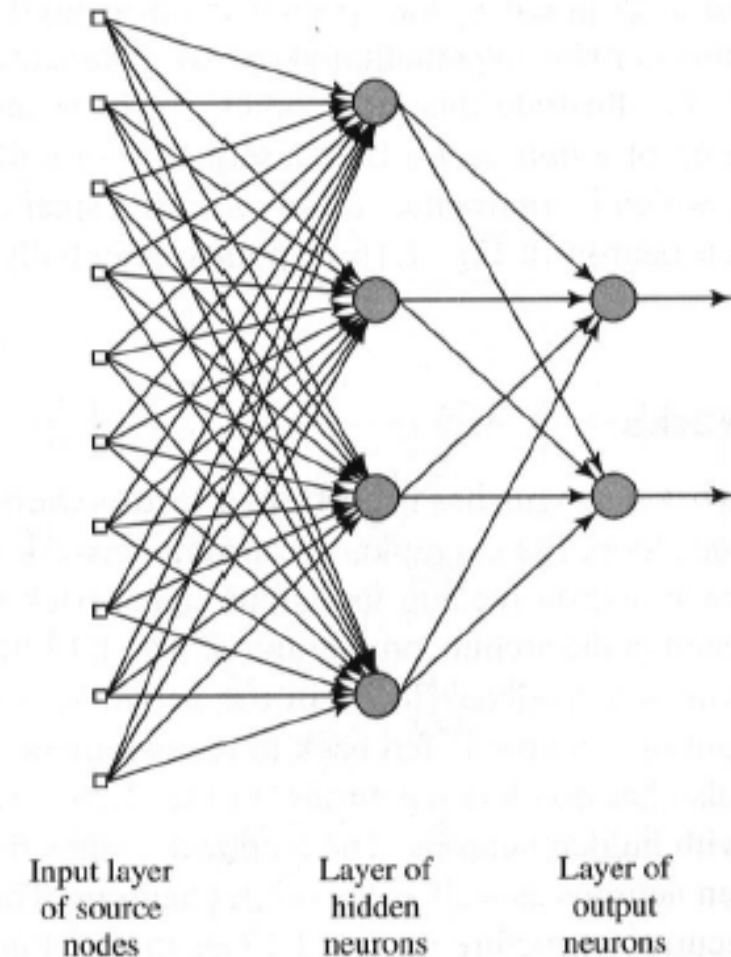
- Comprised of an input layer of source units that inject into an output layer of neurons.



ANN Architectures

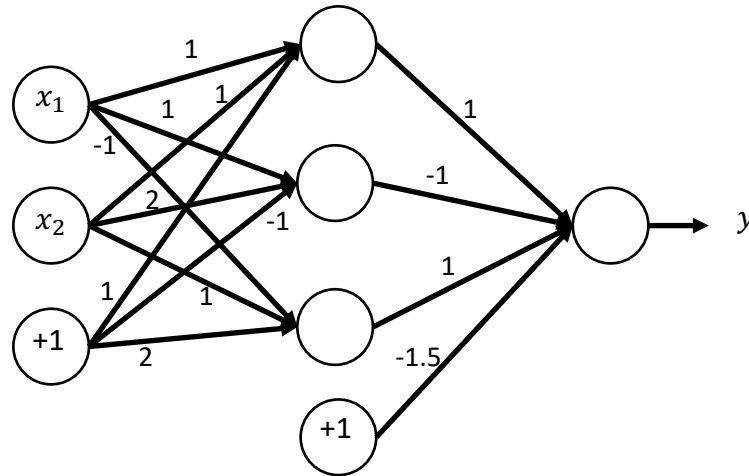
Multilayer Feedforward Networks:

- Comprised of more than one layer of neurons. Layers between input source nodes and output layer is referred to as *hidden layers*.
- Multilayer neural networks can handle *more complicated and larger scale problems* than single-layer networks.
- However, training multilayer network may be *more difficult and time-consuming*.



Three-layer network

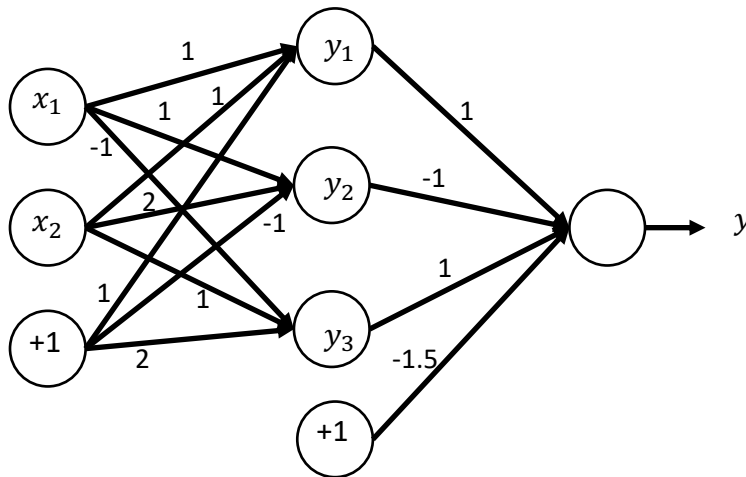
Example 2



Three-layer neural network receives 2-dimensional inputs $(x_1, x_2) \in \mathbf{R}^2$ and has one output neuron and three hidden neurons. All the neurons have unit step activation functions. The weights of the connections are given in the figure. Find the space of inputs for which the output $y = 1.0$.

Find the output for inputs $(0.0, 0.0)$, $(2.0, 2.0)$, and $(-1.0, 1.0)$

Example 2



Activation function:

$$f(u) = \begin{cases} 1.0 & \text{for } u > 0 \\ 0.0 & \text{for } u \leq 0 \end{cases}$$

Synaptic input:

$$u_1 = x_1 + x_2 + 1$$

Output $y_1 = f(u_1)$

$$u_2 = x_1 + 2x_2 - 1$$

$$y_2 = f(u_2)$$

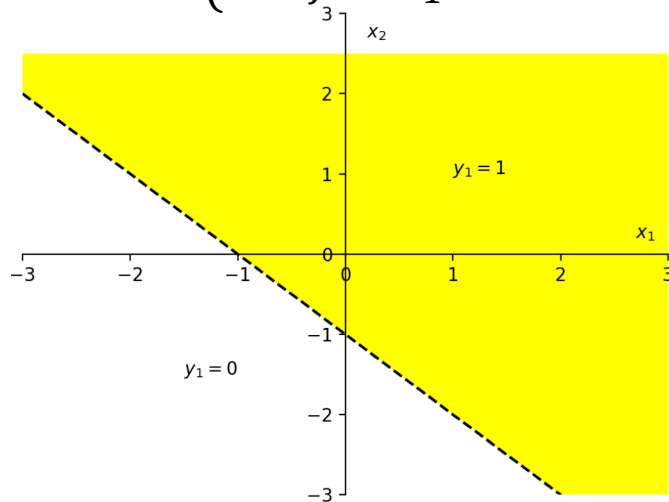
$$u_3 = -x_1 + x_2 + 2$$

$$y_3 = f(u_3)$$

$$u = y_1 - y_2 + y_3 - 1.5$$

$$y = f(u)$$

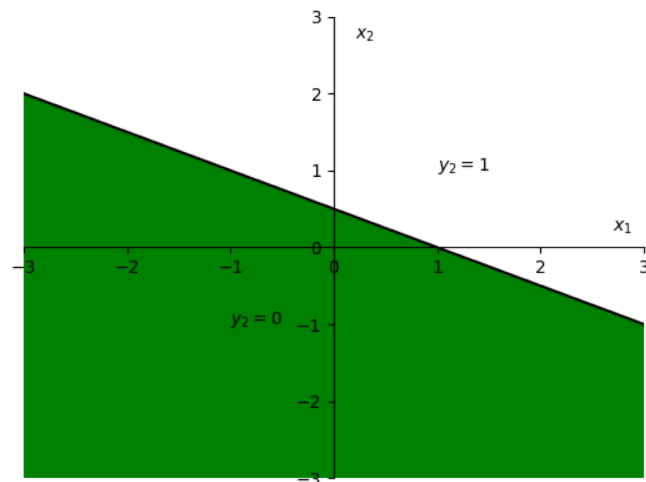
$$y_1 = f(u_1) = \begin{cases} 1.0 & \text{for } u_1 > 0 \\ 0.0 & \text{for } u_1 \leq 0 \end{cases}$$



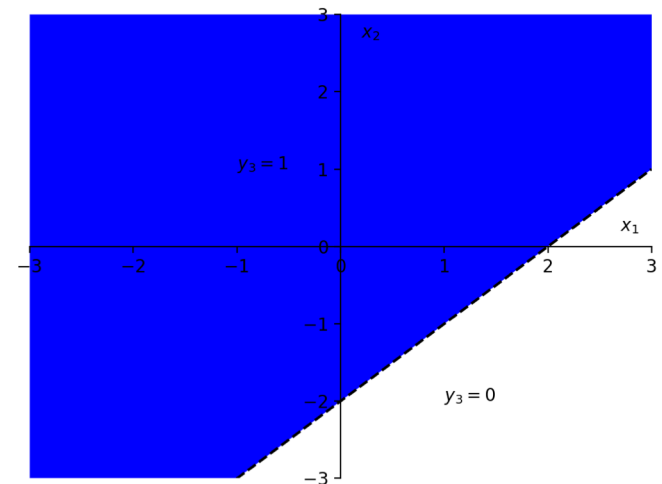
$$u_1 = x_1 + x_2 + 1 = 0 \rightarrow x_2 = -x_1 - 1$$

The boundary line is obtained by setting $u_1 = 0$; and for one side of the boundary, $y = 1$ and on other side $y_1 = 0$.

$$u_2 = 2x_2 + x_1 - 1 = 0 \rightarrow x_2 = -0.5x_1 + 0.5$$



$$u_3 = x_2 - x_1 + 2 = 0 \rightarrow x_2 = x_1 - 2$$

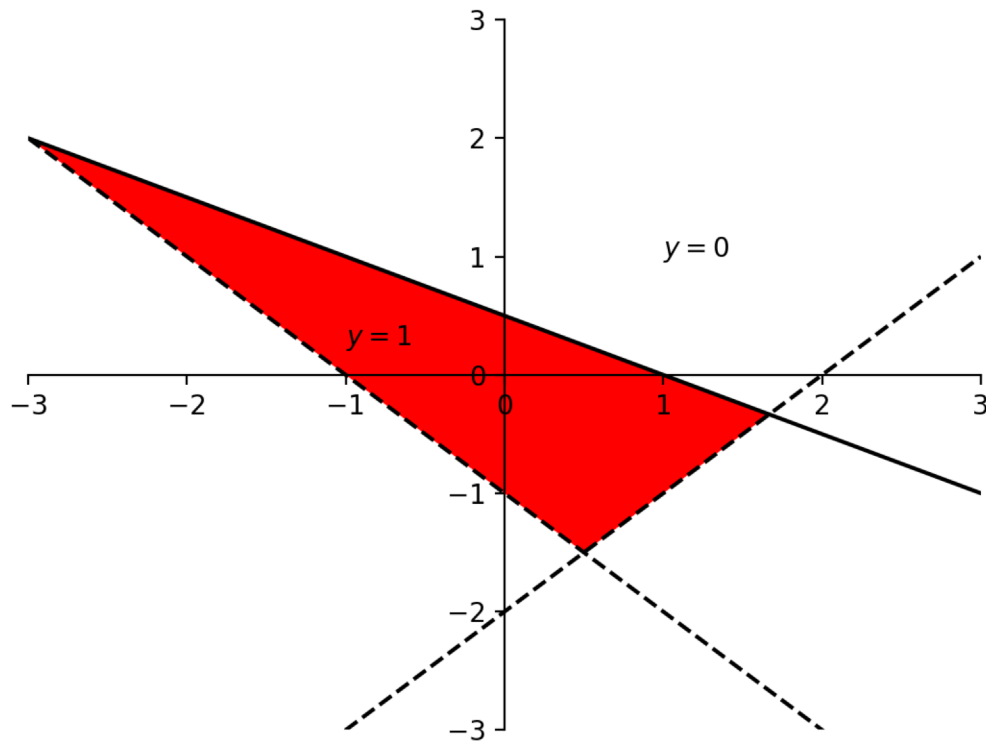


Output layer neuron:

$$u = y_1 - y_2 + y_3 - 1.5$$
$$y = f(u)$$

y_1	y_2	y_3	u	y
0	0	0	-1.5	0
0	0	1	-0.5	0
0	1	0	-2.5	0
0	1	1	-1.5	0
1	0	0	-0.5	0
1	0	1	0.5	1
1	1	0	-1.5	0
1	1	1	-0.5	0

$$Y = Y_1 \bar{Y}_2 Y_3$$



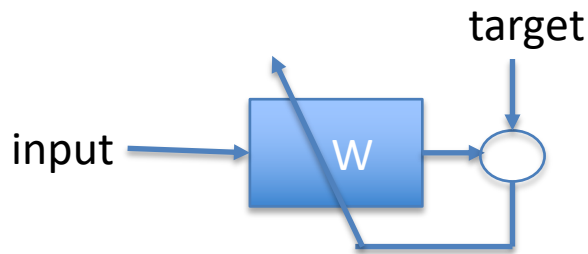
$$\begin{aligned}x &= (0.0, 0.0) \rightarrow y = 1 \\x &= (2.0, 2.0) \rightarrow y = 0 \\x &= (-1.0, 1.0) \rightarrow y = 1\end{aligned}$$

Note that networks of discrete perceptrons (neurons with threshold activation functions) implement Boolean functions.

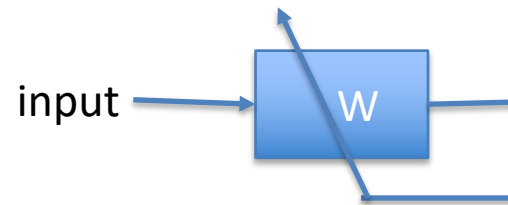
Training (or learning) of neural networks

Neural networks attain their operating characteristics through **learning** (or **training**). During training, the weights or the strengths of connections are gradually adjusted to their desirable values.

Training may be either **supervised** or **unsupervised**.



Supervised Learning



Unsupervised Learning

Supervised and unsupervised learning

Supervised Learning:

For each training input pattern, the network is presented with the correct **targets** (the desired output).

Unsupervised Learning:

For each training input pattern, the network adjusts weights *without knowing* the correct target.

In unsupervised training, the network **self-organizes** to classify similar input patterns into clusters.

Learning of neurons

Learning of a neuron or neural network is usually performed in order to minimize a **cost function** (a **loss function** or **error function**).

The cost function $J(\mathbf{w}, b)$ is of an artificial neuron is typically a multi-dimensional function that depends on weight vector \mathbf{w} and the bias b . The neuron learning attempts to find the weight vector \mathbf{w}^* and bias b^* that minimize the error function:

$$\mathbf{w}^*, b^* = \arg \min_{\mathbf{w}, b} J(\mathbf{w}, b)$$

Supervised learning of neurons

Given a set of training patterns, the parameters (weights and biases) of neurons, minimizing the cost function, are learned using an iterative procedure.

In each iteration, changes of weights Δw and biases Δb are determined according **a learning algorithm**, and then the parameters are updated.

$$w \leftarrow w + \Delta w$$

$$b \leftarrow b + \Delta b$$

Supervised learning of neurons

For supervised learning, the set of training patterns consists of training pairs of inputs and corresponding targets.

A set of P training patterns: $\{(\mathbf{x}_p, d_p)\}_{p=1}^P$

Or $\{(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \dots (\mathbf{x}_P, d_P)\}$

\mathbf{x}_p is the input (features) and d_p is the target (desired output) of p th training pattern.

The input is n -dimensional, $\mathbf{x}_p \in \mathbf{R}^n$ and written as

$$\mathbf{x}_p = (x_{p1}, x_{p2}, \dots x_{pn})^T$$

Supervised learning of neurons

Given a training dataset $\{(x_p, d_p)\}_{p=1}^P$

Initialize weight \mathbf{w} and bias b

Iterate until convergence:

Find $\Delta\mathbf{w}$ and Δb

$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$

$b \leftarrow b + \Delta b$

Convergence is achieved by observing one of the following:

1. No changes in weights and biases
2. No difference between the outputs and targets
3. No improvement in the cost function

Gradient descent learning

The gradient descent procedure states that the value of \mathbf{w} (and b) is updated during learning by searching in the direction of and proportional the **negative gradient** of the cost function. That is, the change of the weight vector:

$$\Delta \mathbf{w} \propto - \frac{\partial J(\mathbf{w}, b)}{\partial \mathbf{w}}$$
$$\Delta \mathbf{w} = -\alpha \frac{\partial J(\mathbf{w}, b)}{\partial \mathbf{w}}$$

Where $\frac{\partial J(\mathbf{w}, b)}{\partial \mathbf{w}}$ is the gradient (partial derivative) of cost with respect to weight and α is *learning factor or learning rate*. $\alpha \in (0.0, 1.0]$.

The learning equation is given by: $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial \mathbf{w}}$$

Similarly,

$$b \leftarrow b - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial b}$$

Gradient descent learning

The **gradient descent equations** for the weights and bias are given by

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial \mathbf{w}} \\ b &\leftarrow b - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial b}\end{aligned}$$

Notation: $\nabla_{\mathbf{w}} J = \frac{\partial J(\mathbf{w}, b)}{\partial \mathbf{w}}$ and $\nabla_b J = \frac{\partial J(\mathbf{w}, b)}{\partial b}$.

Gradient descent learning is given by

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J \\ b &\leftarrow b - \alpha \nabla_b J\end{aligned}$$

where α is the learning factor (rate).

Stochastic Gradient Descent (SGD) learning

Given a set of training patterns $\{(\mathbf{x}_p, d_p)\}_{p=1}^P$

Set learning factor α

Initialize (\mathbf{w}, b)

Iterate until convergence:

for each pattern (\mathbf{x}_p, d_p) :

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J_p$$

$$b \leftarrow b - \alpha \nabla_b J_p$$

- The cost J_p is computed from the output and the target of the p th training pattern.
- A cycle of iteration that goes over all the input patterns is referred to as an **epoch**

(Batch) Gradient Descent

Inputs are presented as a batch in a **data matrix** \mathbf{X} and a **target vector** \mathbf{d} .
The input data points are written as rows in the data matrix and the targets are written into a single vector in the target vector.

Given a input dataset $\{(\mathbf{x}_p, d_p)\}_{p=1}^P$.

Data matrix:

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_P^T \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{Pn} \end{pmatrix}$$

Target vector:

$$\mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_P \end{pmatrix}$$

(Batch) Gradient descent learning

Given a set of training patterns: (\mathbf{X}, \mathbf{d})

Set learning factor α

Initialize (\mathbf{w}, b)

Iterate until convergence:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J$$

$$b \leftarrow b - \alpha \nabla_b J$$

The cost J is computed using all the training patterns.

That is, using (\mathbf{X}, \mathbf{d})

Summary

- Analogy between biological and artificial neurons

- Transfer function of artificial neuron:

$$u = \mathbf{x}^T \mathbf{w} + b$$
$$y = f(u)$$

- Types of activation functions: sigmoid, threshold, linear, ReLU, and tanh.

- Given inputs, to find the outputs for simple feedforward networks

- Supervised and unsupervised learning

- Gradient descent learning:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J$$
$$b \leftarrow b - \alpha \nabla_b J$$

- Stochastic gradient descent (SGD) and batch gradient descent (GD)