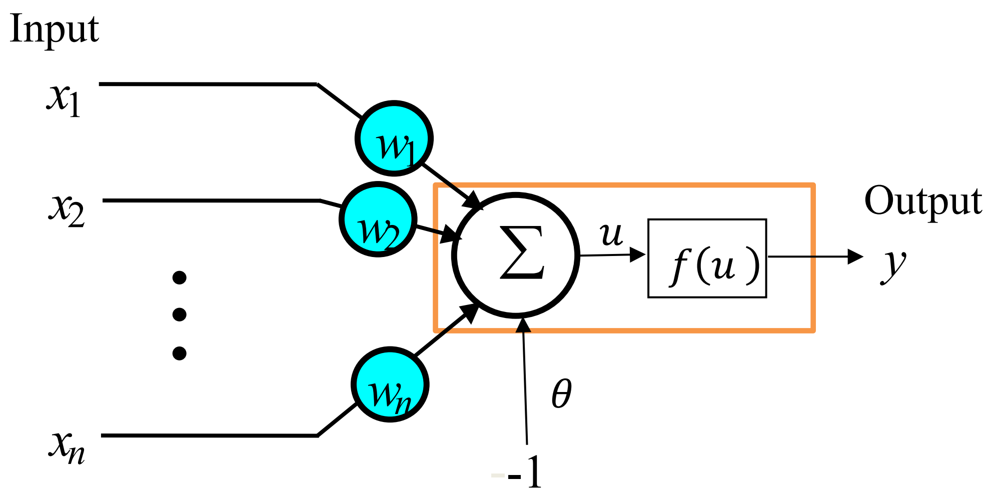
Chapter 1

Introduction to Neural Networks

Neural networks and deep learning

Artificial Neuron



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

Artificial Neuron

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

Basic elements of an artificial neuron:

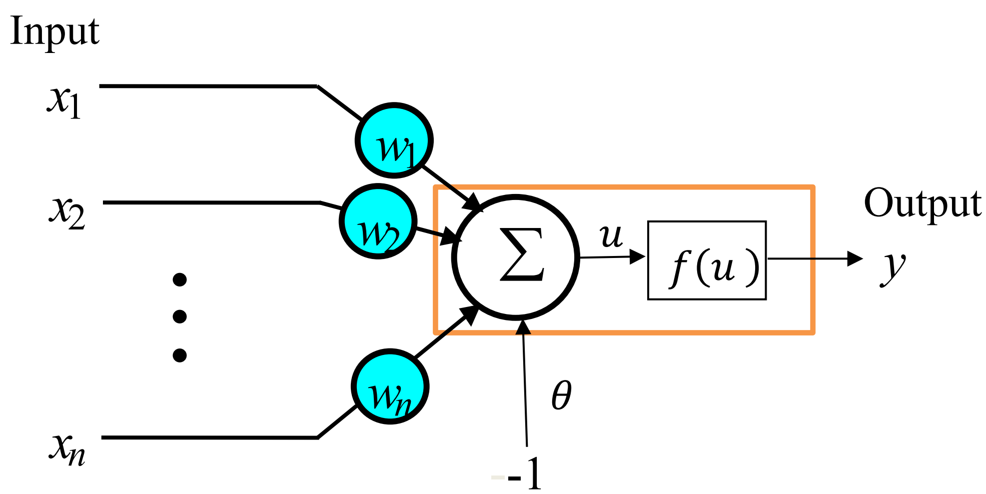
• A set of **input** signals: the input is a vector $\boldsymbol{x} = (x_1 \quad x_2 \quad \cdots \quad x_n)^T$ where n is the number (or the dimension) of input signals. Inputs are also referred to as **features**.

• Inputs are connected to the neuron via synaptic connections whose strengths are represented by their **weights**.

• The weight vector $\boldsymbol{w} = (w_1 \quad w_2 \quad \cdots \quad w_n)^T$ where w_i is the synaptic weight connecting i th input to the neuron.

Notation: vectors are denoted in <u>bold</u> and written as horizontally with a <u>transpose</u> (^{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

$$u = w_1 x_1 + w_2 x_2 + \cdots w_n x_n - \theta = \sum_{i=1}^{n} w_i x_i - \theta$$

By using vector notations:

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

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.

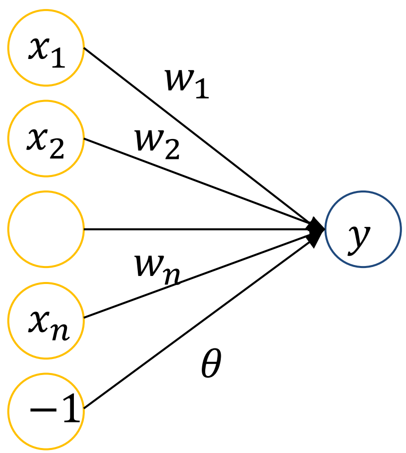
For some neurons, the **output** y is equal to the activation of the neuron.

$$y = f(u)$$

Note that activation is not generally equal to the output.

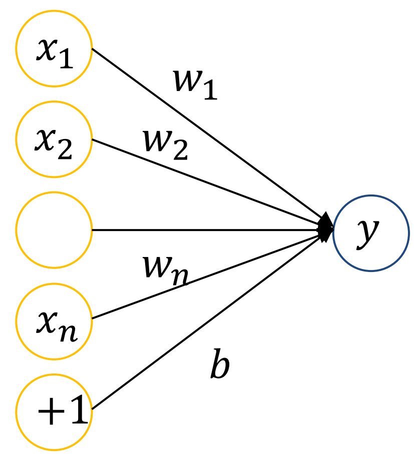
Bias vs. Threshold

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

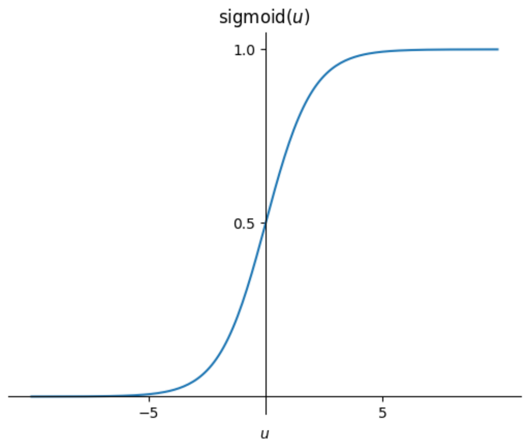
$$u = \boldsymbol{w}^T \boldsymbol{x} - \theta$$
$$y = f(u)$$

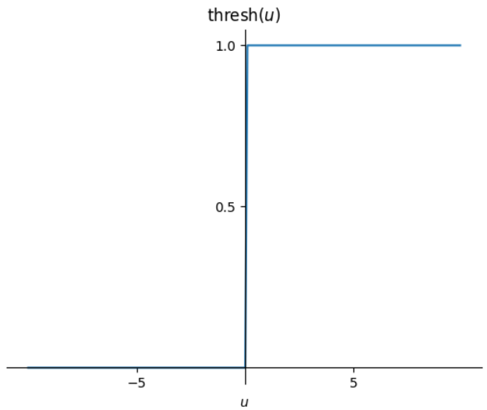


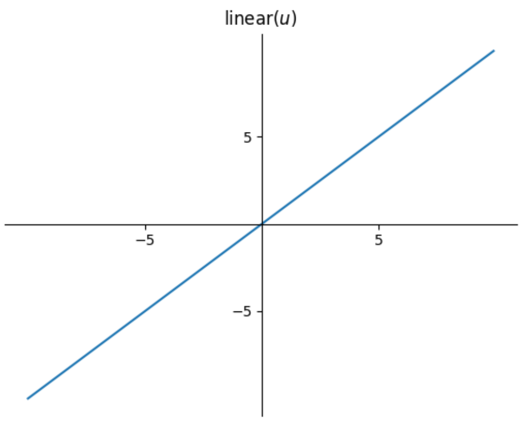
Bias

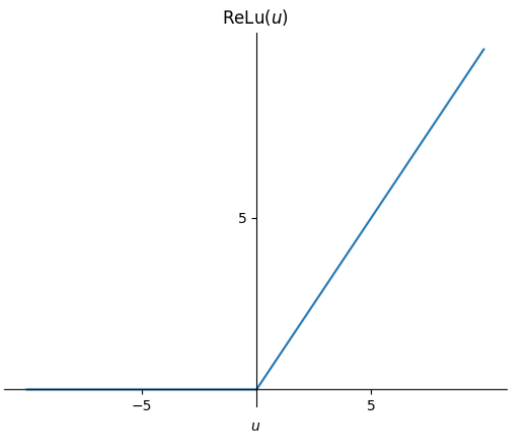
$$u = \boldsymbol{w}^T \boldsymbol{x} + b$$
$$y = f(u)$$

Activation functions









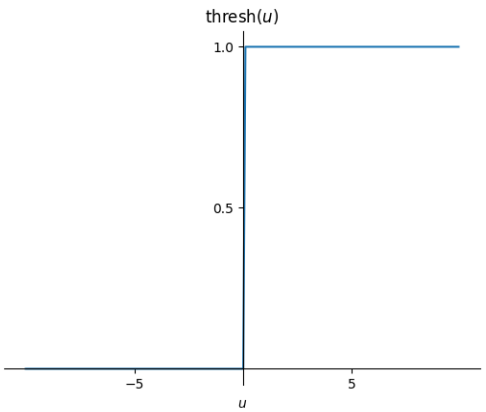
Activation functions

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

$$f(u) = theshold(u) = 1(u > 0)$$

where $1(\cdot)$ is the indictor function defined by

$$1(x) = \begin{cases} 1, & x \text{ is True} \\ 0, & x \text{ is False} \end{cases}$$



Activation functions

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

$$f(u) = linear(u) = u$$

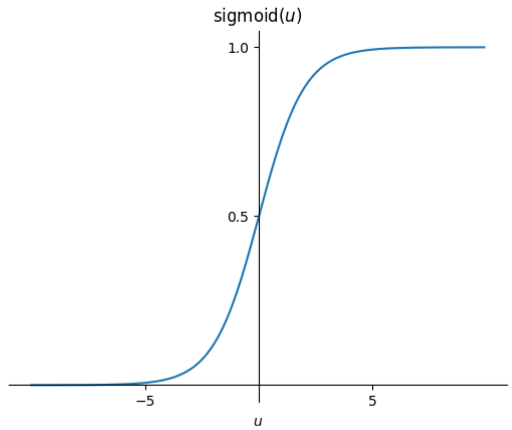
The ***ReLU*** **(rectified-linear unit)** activation function can be written as

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

Sigmoid activation function

The sigmoidal is known as the **logistic function** or simply **sigmoid function**

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



In general, the ***sigmoid*** activation function can be written as

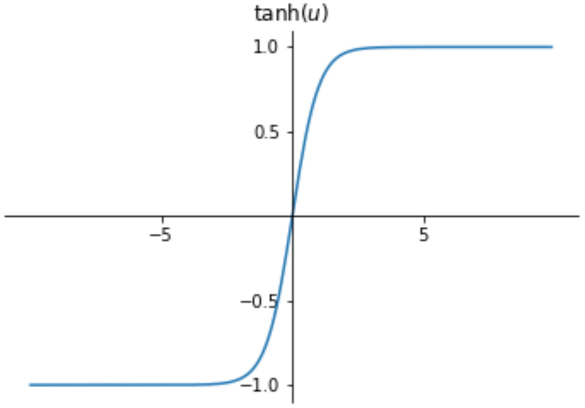
$$f(u) = \frac{a}{1 + e^{-bu}}$$

a is the gain (amplitude) and b is the slope.

But often, $a = 1.0$ and $b = 1.0$,

Tanh activation function

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

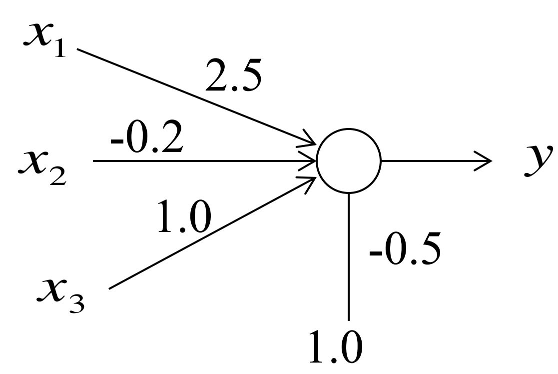


Tanh activation function has the same shape as sigmoidal and spans from -1 and +1. It is also known as **bipolar sigmoidal**.

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

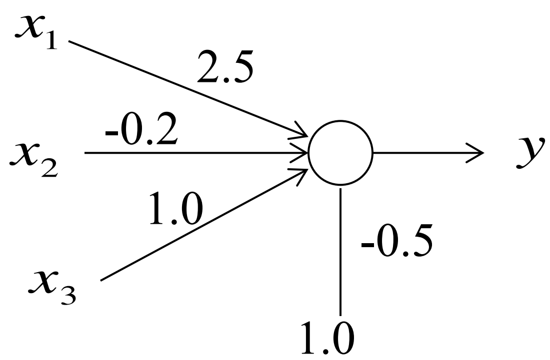
Example 1

The artificial neuron in the figure receives 3-dimensional inputs $\boldsymbol{x} = (x_1 \quad x_2 \quad x_3)^T$ 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



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

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

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

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

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

Tensorflow 2.2

Tensorflow is about processing of **tensors**. Tensor is a multidimensional array.

Rank refers to the number of dimensions and **shape** gives the sizes of each dimension of the tensor.

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 Program

Tensorflow program involves two steps:

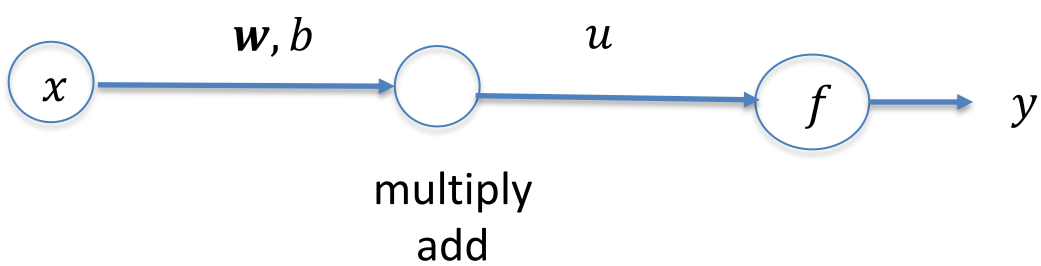
- Building the computational graph
- Evaluating the computational graph

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

Computational graph of a neuron

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



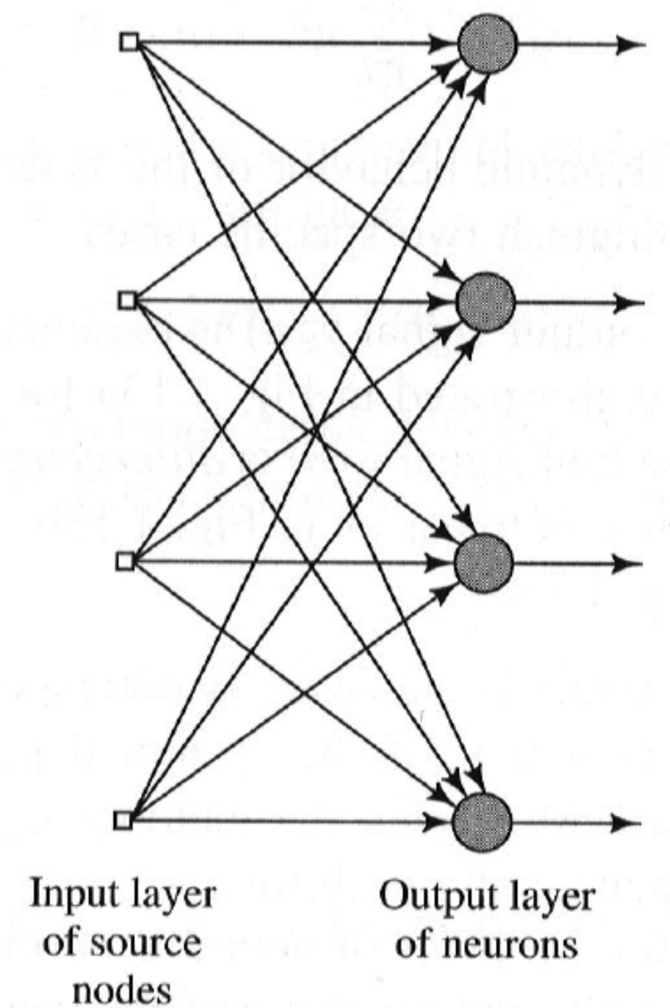
Tensorflow Implementation of Example 1

```
import tensorflow as tf

# a class for neuron
class Neuron():
  # initiate a neuron class with weights and biases (initiate the object)
  def __init__(self):
    self.w = tf.Variable([2.5, -0.2, 1.0], tf.float32)
    self.b = tf.Variable(-0.5, tf.float32)

  # evaluate the neuron (implement a function)
  def __call__(self, x):
    u = tf.tensordot(self.w, x, axes=1) + self.b
    y = 0.8/(1+tf.exp(-1.2*u))

  return u, y

# create a neuron
neuron = Neuron()

# evaluate
u, y = neuron([0.8, 2.0, -0.5])

# print: u = 0.600, y = 0.538
```

ANN Architectures

Single – layer of neurons

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



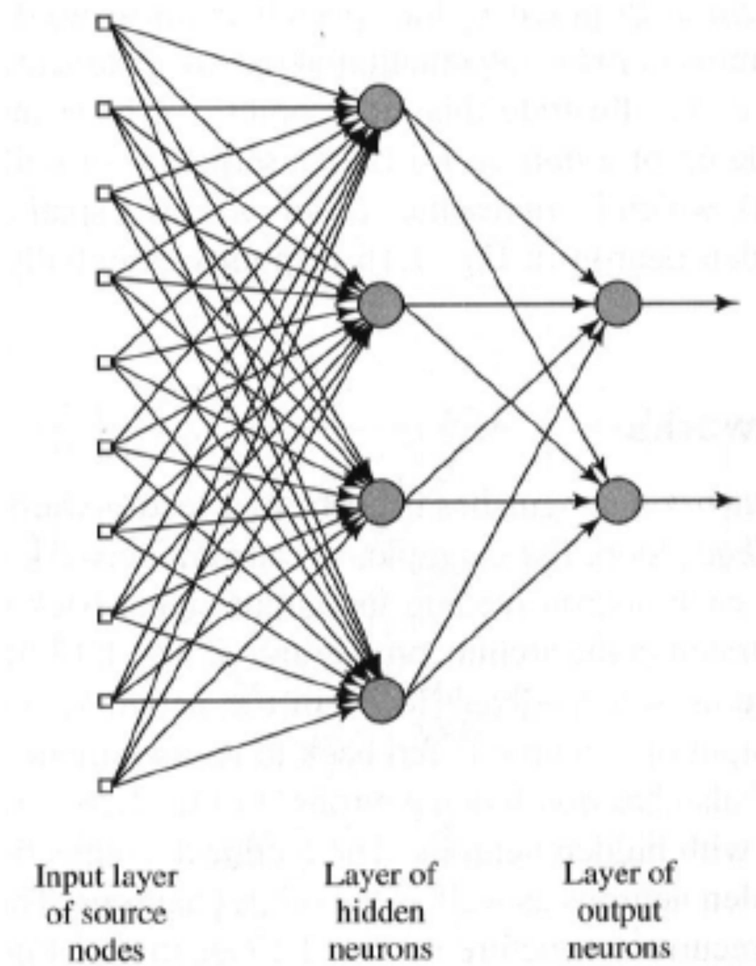
ANN Architectures

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*.

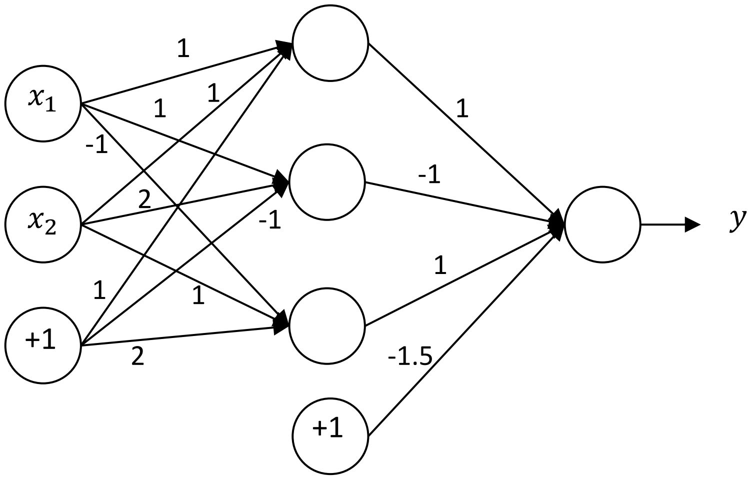
• Multilayer neural networks can handle *more complicated* and *larger scale problems* than single-layer networks.

• However, training multilayer network may be *more difficult* and *time-consuming*.



Three-layer network

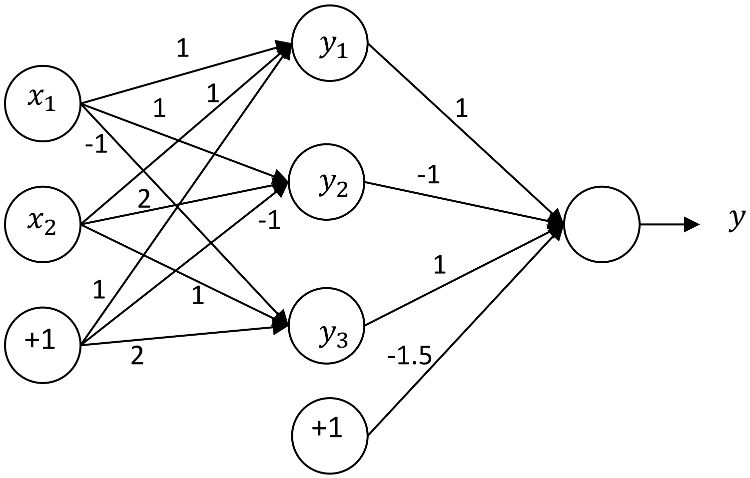
Example 2



Three-layer neural network receives 2-dimensional inputs $(x_1, x_2) \in \boldsymbol{R}^2$ and has one output neuron and three hidden neurons. All the neurons have <u>unit step activation functions</u>. 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



Synaptic input:

$$u_1 = x_1 + x_2 + 1$$

Output $y_1 = 1(u_1 > 0)$

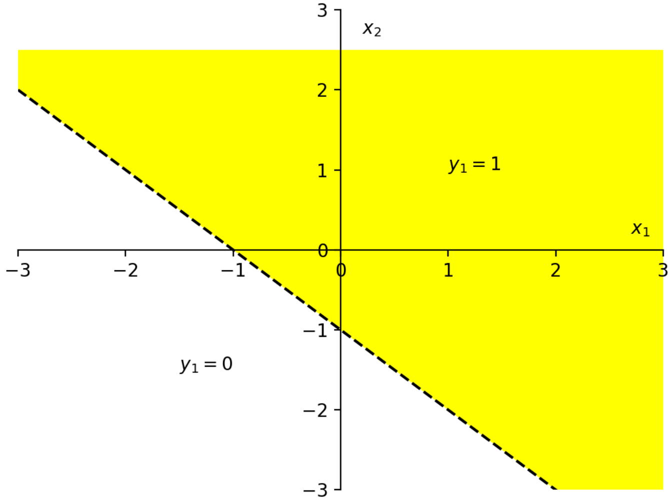
$$u_2 = x_1 + 2x_2 - 1$$
$$y_2 = 1(u_2 > 0)$$

$$u_3 = -x_1 + x_2 + 2$$
$$y_3 = 1(u_3 > 0)$$

$$u = y_1 - y_2 + y_3 - 1.5$$
$$y = 1(u > 0)$$

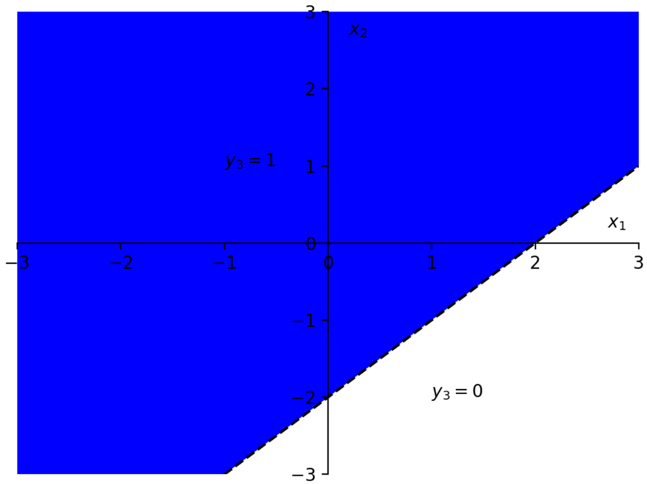
$$y_1 = f(u_1) = y_1 = 1(u_1 > 0)$$

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

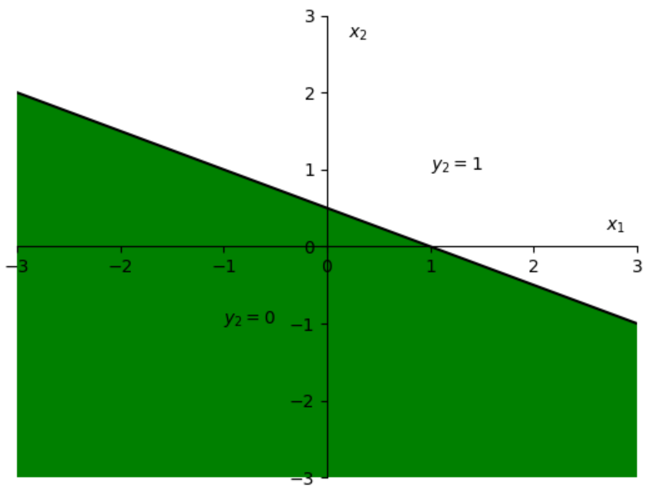


The boundary line is obtained by setting $u_1 = 0$; and for one side of the boundary, $y = 1$ and on other side $y_1 = 0$.

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



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



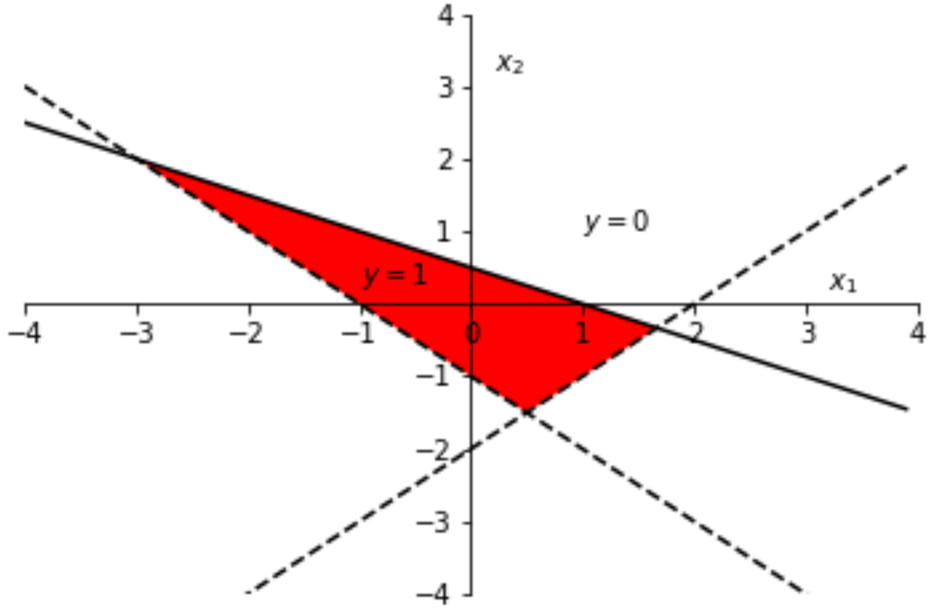
Output layer neuron:

$$u = y_1 - y_2 + y_3 - 1.5$$
$$y = 1(u > 0)$$

Note that $y_1, y_2, y_3 \in \{0, 1\}$

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

$$Y = Y_1 \overline{Y_2} Y_3$$



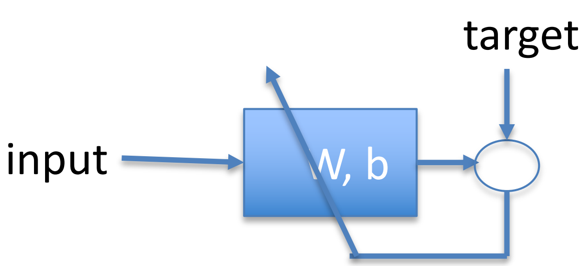
$$x = (0.0, 0.0) \ \rightarrow y = 1$$
$$x = (2.0, 2.0) \ \rightarrow y = 0$$
$$x = (-1.0, 1.0) \ \rightarrow y = 1$$

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

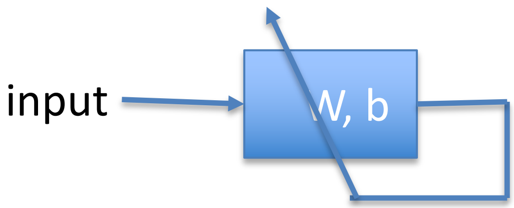
Training (or learning) of neural networks

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

Supervised learning of neurons

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

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

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

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 $\Delta\boldsymbol{w}$ and biases Δb are determined according **a learning algorithm**, and then the parameters are updated.

Initialize $\boldsymbol{w}$, b

Iterate until convergence:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w}$$
$$b \leftarrow b + \Delta b$$

Gradient descent learning

The grading descent procedure states that the value of $\boldsymbol{w}$ (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:

$$\Delta \boldsymbol{w} \propto -\frac{\partial J(\boldsymbol{w},b)}{\partial \boldsymbol{w}}$$

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

Where $\frac{\partial J(\boldsymbol{w},b)}{\partial \boldsymbol{w}}$ 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 **gradient descent equations** for learning the weights is given by

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

Gradient descent learning

The **gradient descent equations** for the weights is given by

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

Similarly, for bais

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

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

Gradient descent learning is given by

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

Gradient descent learning of neurons

Initialize weight $\boldsymbol{w}$ and bias b
Iterate until convergence:

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

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 decrease in the cost function J

Note: Will drop the arguments in J.

Training data for supervised learning

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

A set of P training patterns: $\{(\boldsymbol{x}_p, d_p)\}_{p=1}^{P}$
Or $\{(\boldsymbol{x}_1, d_1), (\boldsymbol{x}_2, d_2), \cdots (\boldsymbol{x}_P, d_P)\}$

$\boldsymbol{x}_p$ is the input (features) and d_p is the target (desired output) of p th training pattern. There are P patterns in the dataset.

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

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

Stochastic Gradient Descent (SGD) learning

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

Set learning factor α
Initialize $(\boldsymbol{w}, b)$
Iterate until convergence:
    for each pattern $(\boldsymbol{x}_p, d_p)$:
        $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} J_p$
        $b \leftarrow b - \alpha \nabla_b J_p$

- In each **epoch** or cycle of iteration, the learning takes place individually over every pattern
- The cost J_p is computed from the output and the target of the p th training pattern.

(Batch) Gradient Descent

Inputs are presented as a batch in a **data matrix** $\boldsymbol{X}$ and a **target vector** $\boldsymbol{d}$. 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 $\{(\boldsymbol{x}_p, d_p)\}_{p=1}^{P}$.

Data matrix:

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

Target vector:

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

(Batch) Gradient descent learning

Given a set of training patterns: $(\boldsymbol{X}, \boldsymbol{d})$
Set learning factor α
Initialize $(\boldsymbol{w}, b)$
Iterate until convergence:

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

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

The cost J is computed using <u>all</u> the training patterns.
That is, using $(\boldsymbol{X}, \boldsymbol{d})$
In each epoch, the weights are updated once considering all the input patterns.

Summary

- Analogy between biological and artificial neurons

- Transfer function of artificial neuron:
$$u = \boldsymbol{w}^T \boldsymbol{x} + b$$
$$y = f(u)$$

- 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:
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} J$$
$$b \leftarrow b - \alpha \nabla_{b} J$$

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