Temas Selectos de Aprendizaje Automático

Sesion II

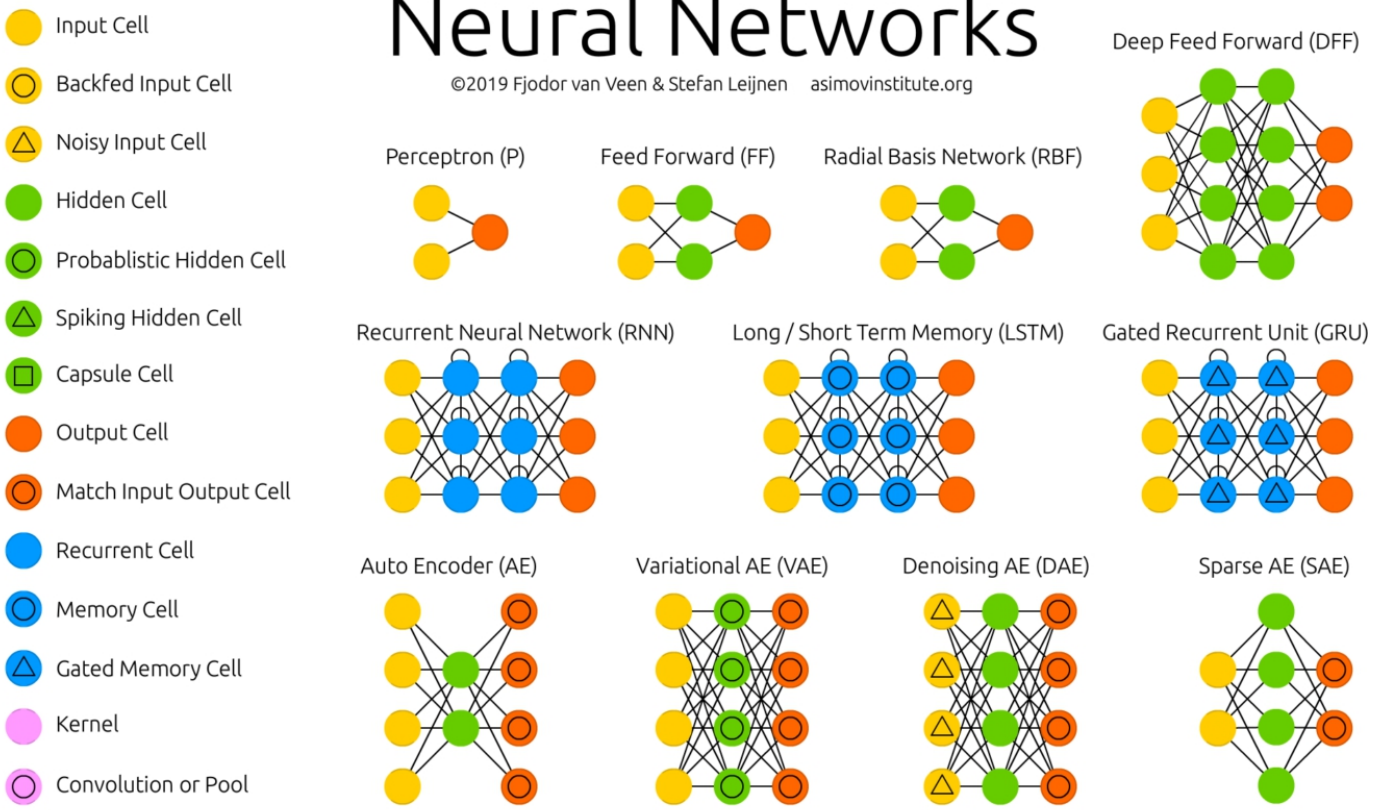
PhD. Edwyn Javier Aldana Bobadilla

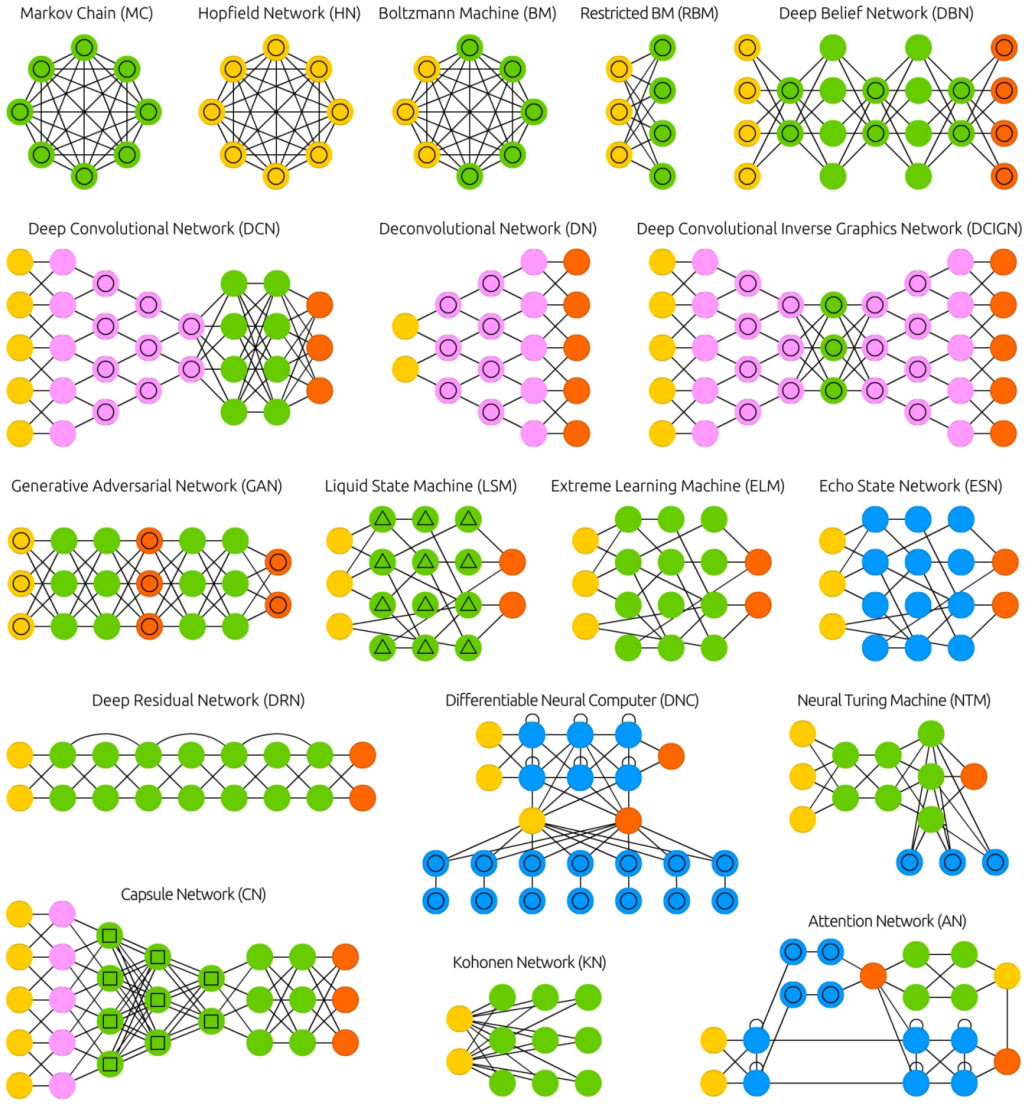
Agenda (5)

Topics	Date

• **Neural Network Introduction**
  • Fundamentals
• **Deep Learning**
  • Fundamentals
• **Convolution Neural Networks**
  • Convolution operation
  • Architecture
  • Application
• **Recurrent Neural Networks**
  • Fundamentals
	July 18, 17:00-22:00

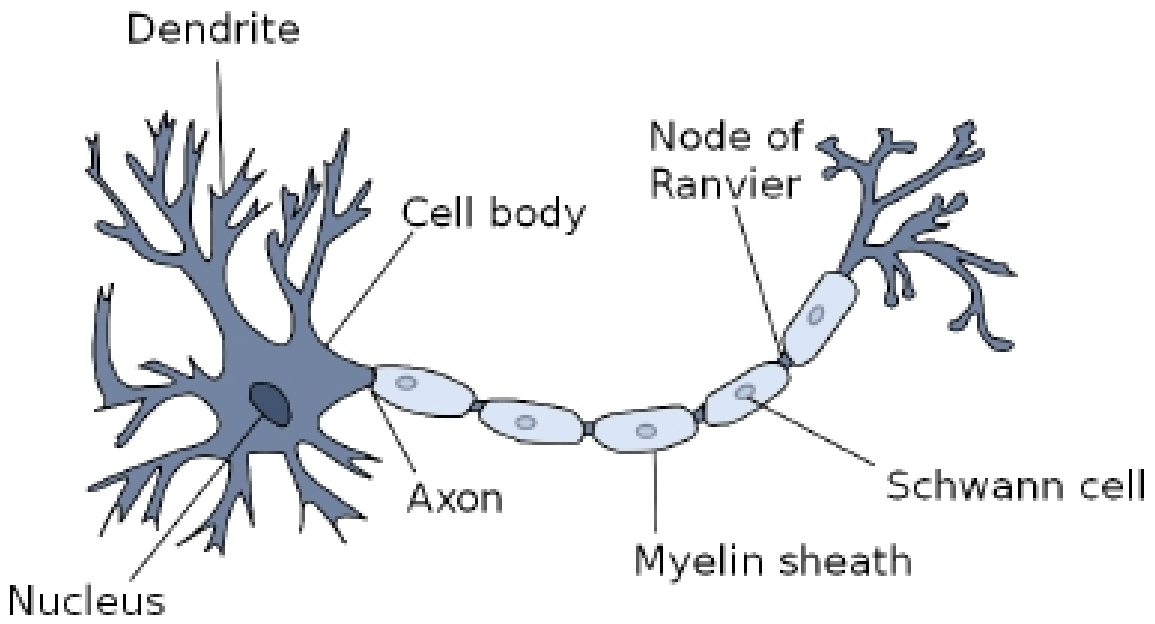
Neural Networks: The foundations of Deep Learning





What is a neuron?

Typically a neuron is defined as an information processing unit.



What is a neuron?

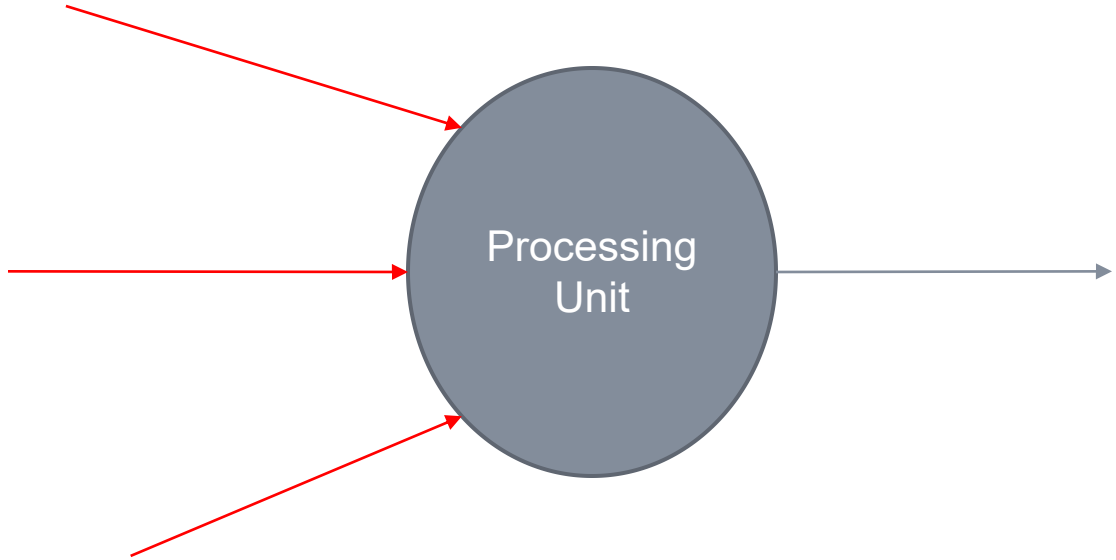
From biology, a neuron processes electrochemical input signals which come from others neurons.

The result of such processing is another electrochemical signal which is called output.

This signal is transmitted to other neuron through the axon.

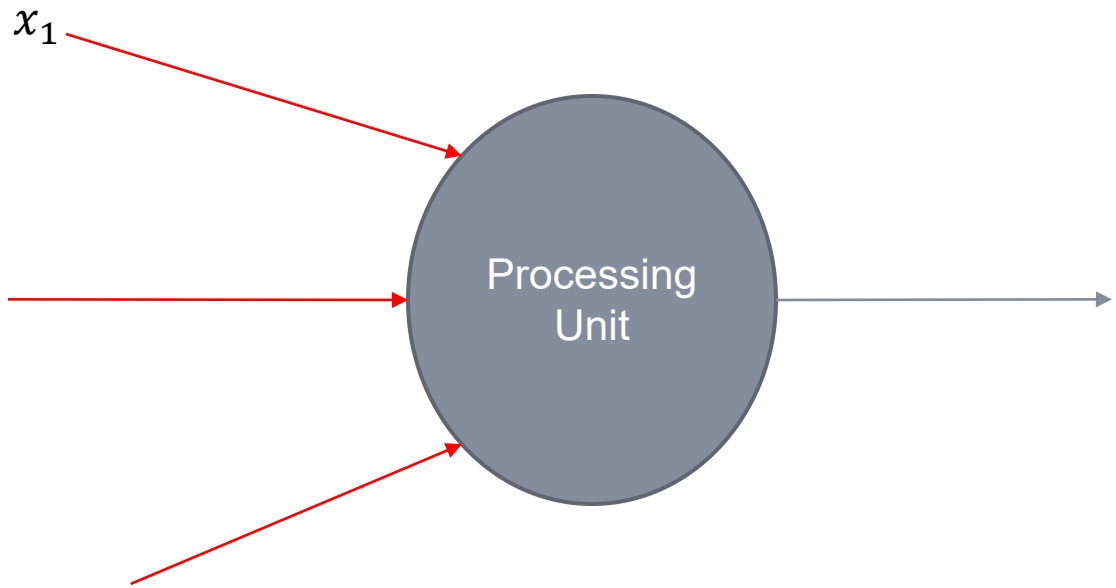
Artificial Neuron

Inspired by biological neurons



Artificial Neuron

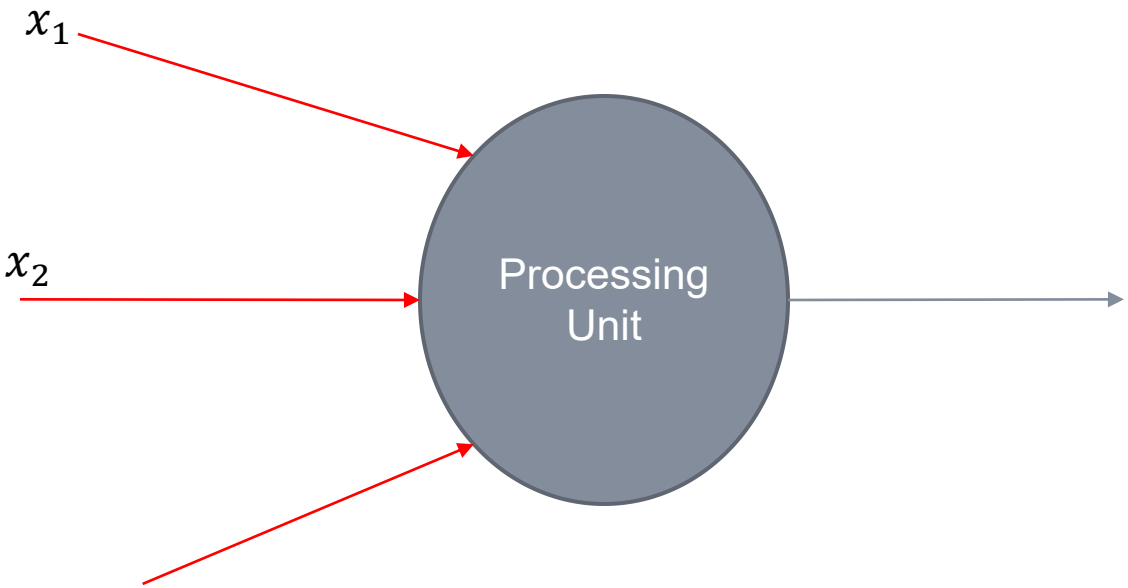
Inspired by biological neurons



The input of a neuron is a vector $\vec{x}$

Artificial Neuron

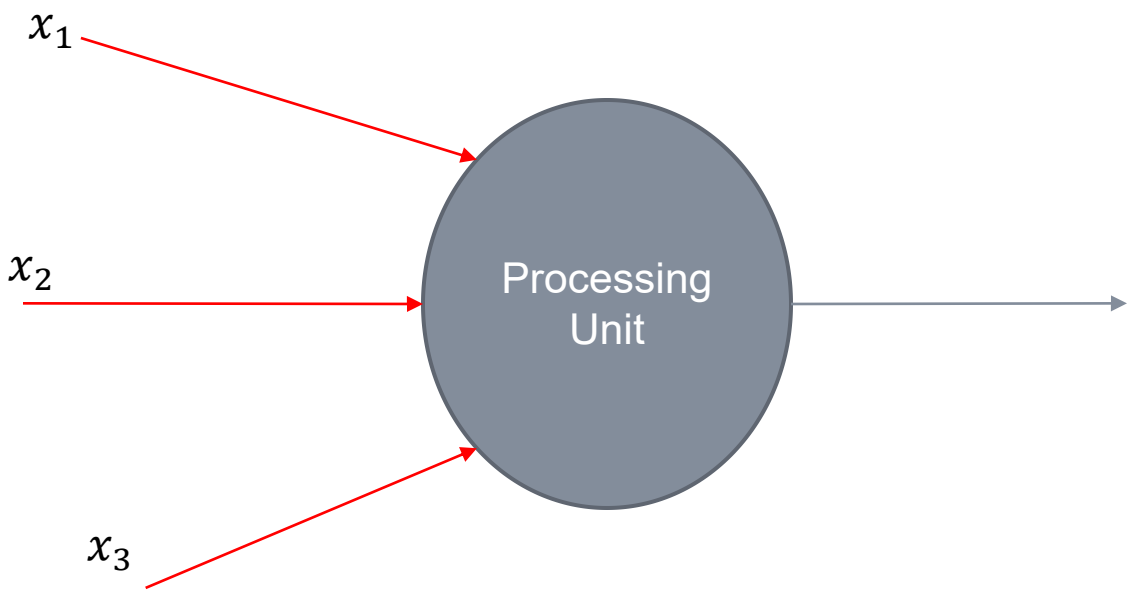
Inspired by biological neurons



The input of a neuron is a vector $\vec{x}$

Artificial Neuron

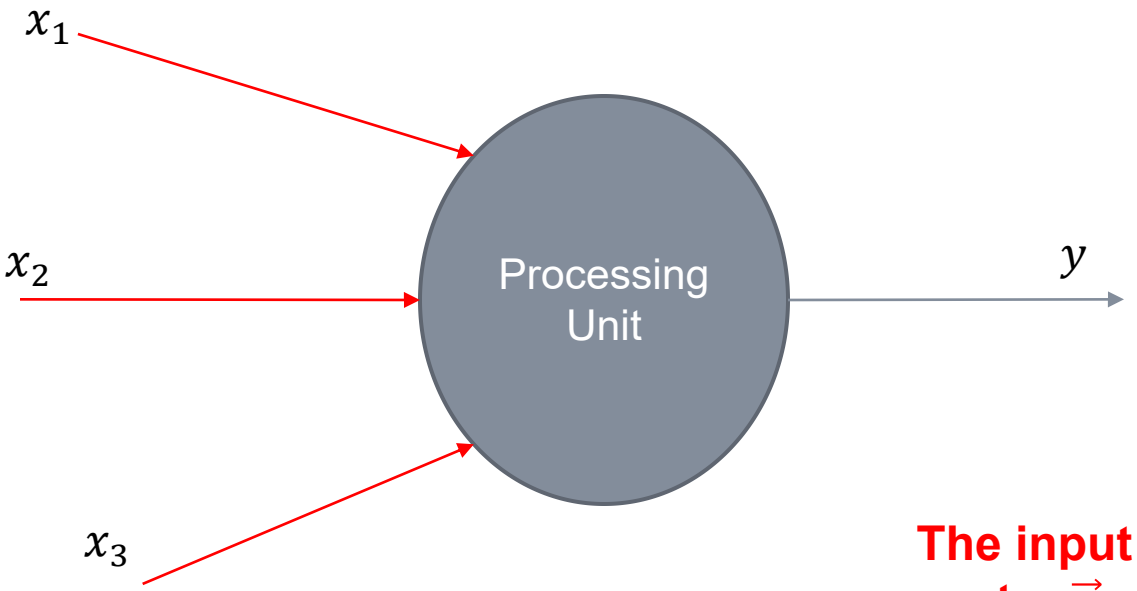
Inspired by biological neurons.



The input of a neuron is a vector $\vec{x}$

Artificial Neuron

Inspired by biological neurons.

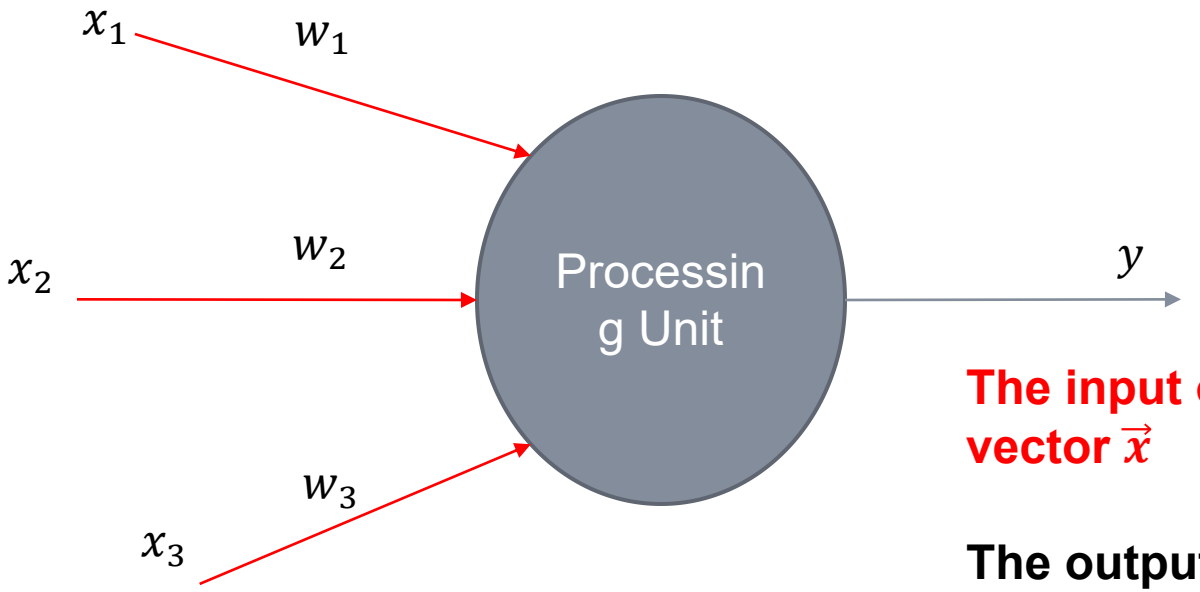


The input of a neuron is a vector $\vec{x}$

The output of the neuron is a scalar value

Artificial Neuron

Inspired by biological neurons.



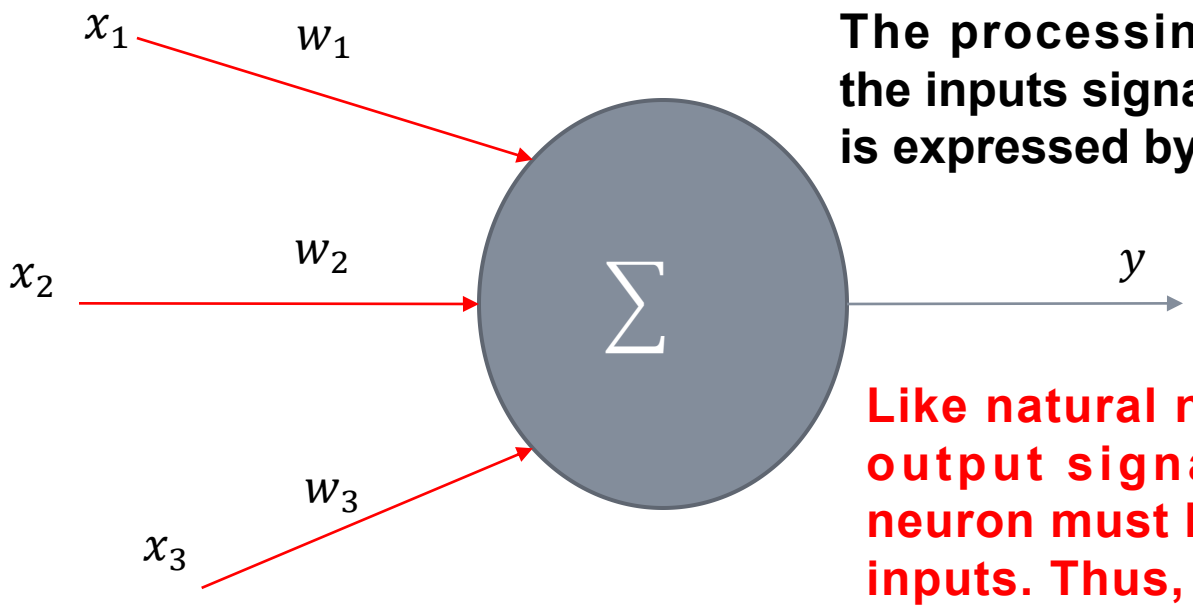
The input of a neuron is a vector $\vec{x}$

The output of the neuron is a scalar value

The importance of a component of $\vec{x}$ is weighted with a value w_i

Artificial Neuron

Inspired by biological neurons.



The input of a neuron is a vector $\vec{x}$

The output of the neuron is a scalar value

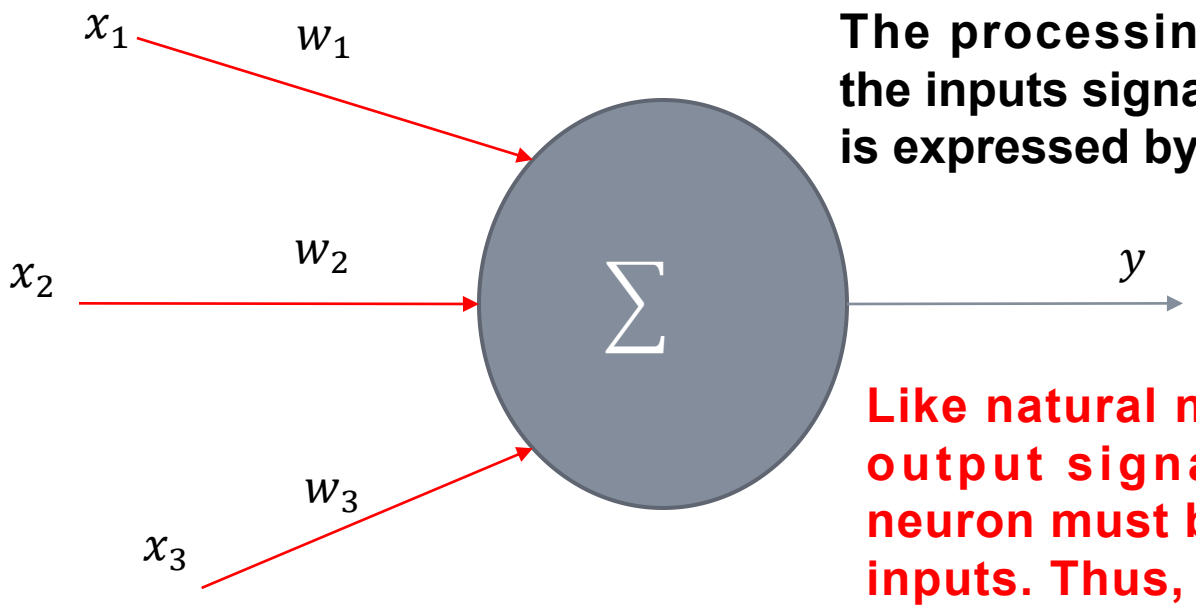
The importance of a component of $\vec{x}$ is weighted with a value w_i

The processing unit summarized the inputs signals (mathematically it is expressed by a sum).

Like natural neurons, the value of output signal determine if the neuron must be activated given its inputs. Thus, the neurons have an activation threshold.

Artificial Neuron

Inspired by biological neurons.



The input of a neuron is a vector $\vec{x}$

The output of the neuron is a scalar value

The importance of a component of $\vec{x}$ is weighted with a value w_i

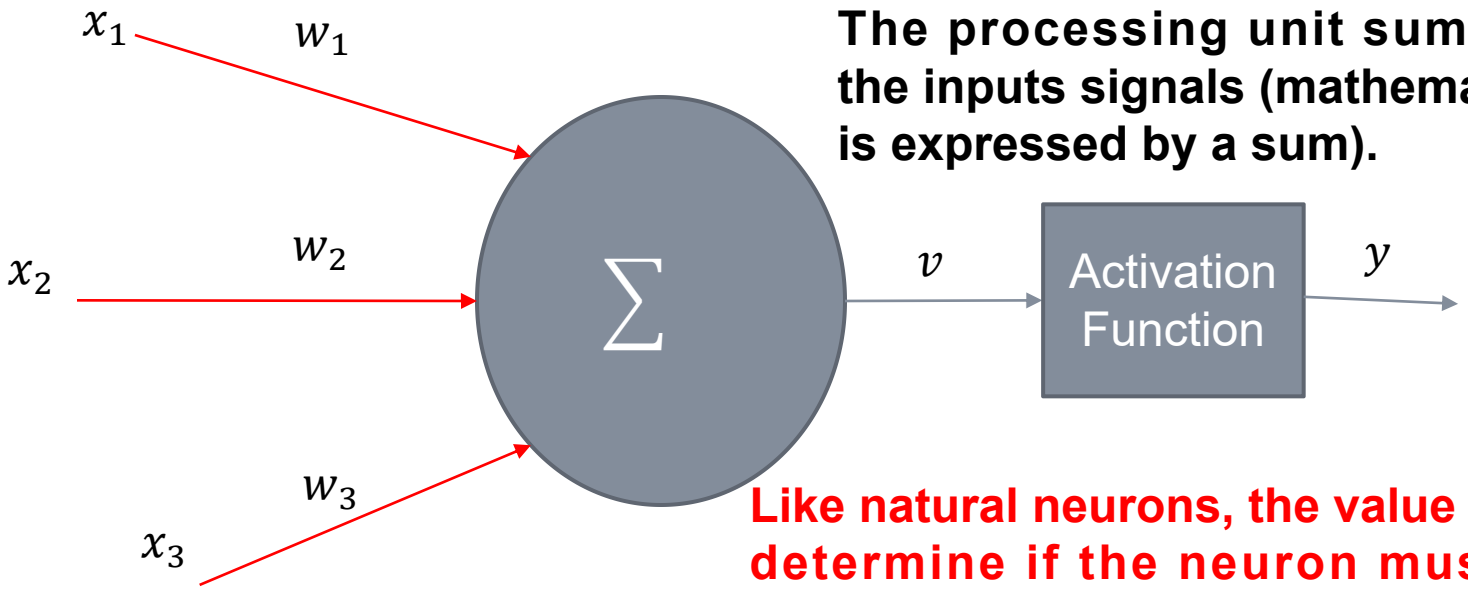
The processing unit summarized the inputs signals (mathematically it is expressed by a sum).

Like natural neurons, the value of output signal determine if the neuron must be activated given its inputs. Thus, the neurons have an activation threshold.

We can emulate such threshold with mathematical functions known as activation functions

Artificial Neuron

Inspired by biological neurons.



The input of a neuron is a vector $\vec{x}$

The output of the neuron is a scalar value

The importance of a component of $\vec{x}$ is weighted with a value w_i

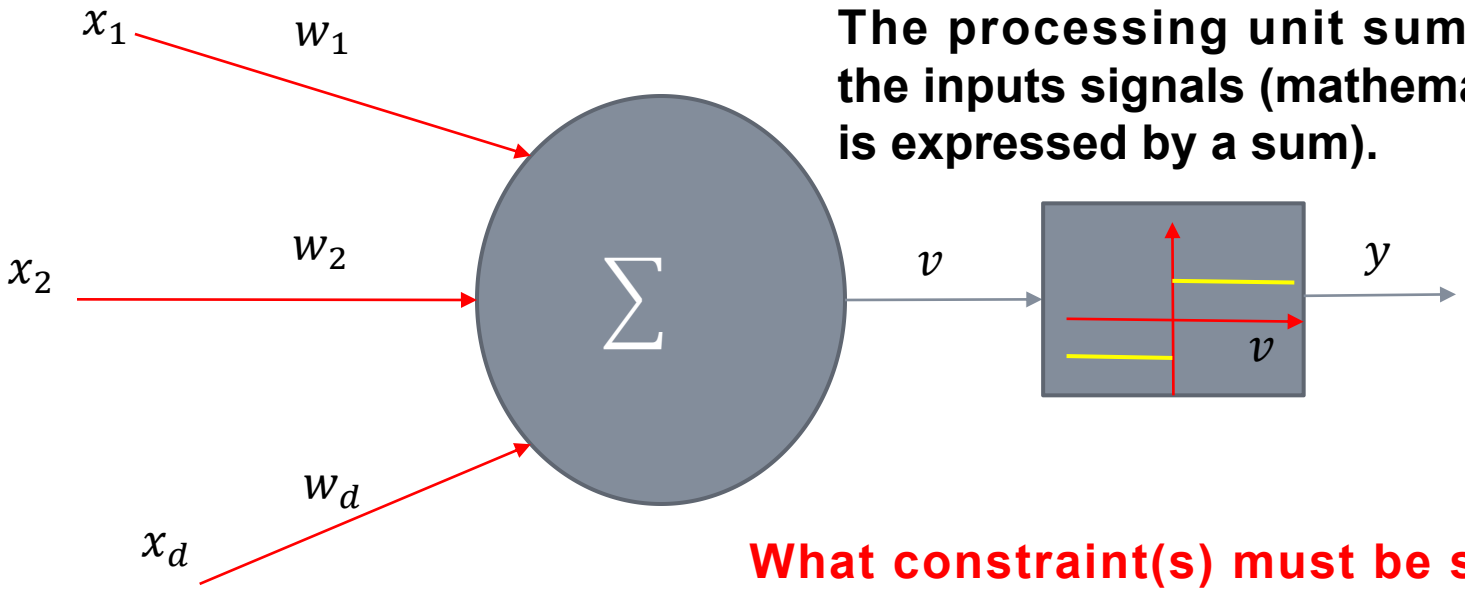
The processing unit summarized the inputs signals (mathematically it is expressed by a sum).

Like natural neurons, the value of output signal determine if the neuron must be activated given its inputs. Thus, the neurons have an activation threshold.

We can emulate such threshold with mathematical functions known as activation

Artificial Neuron

Inspired by biological neurons.



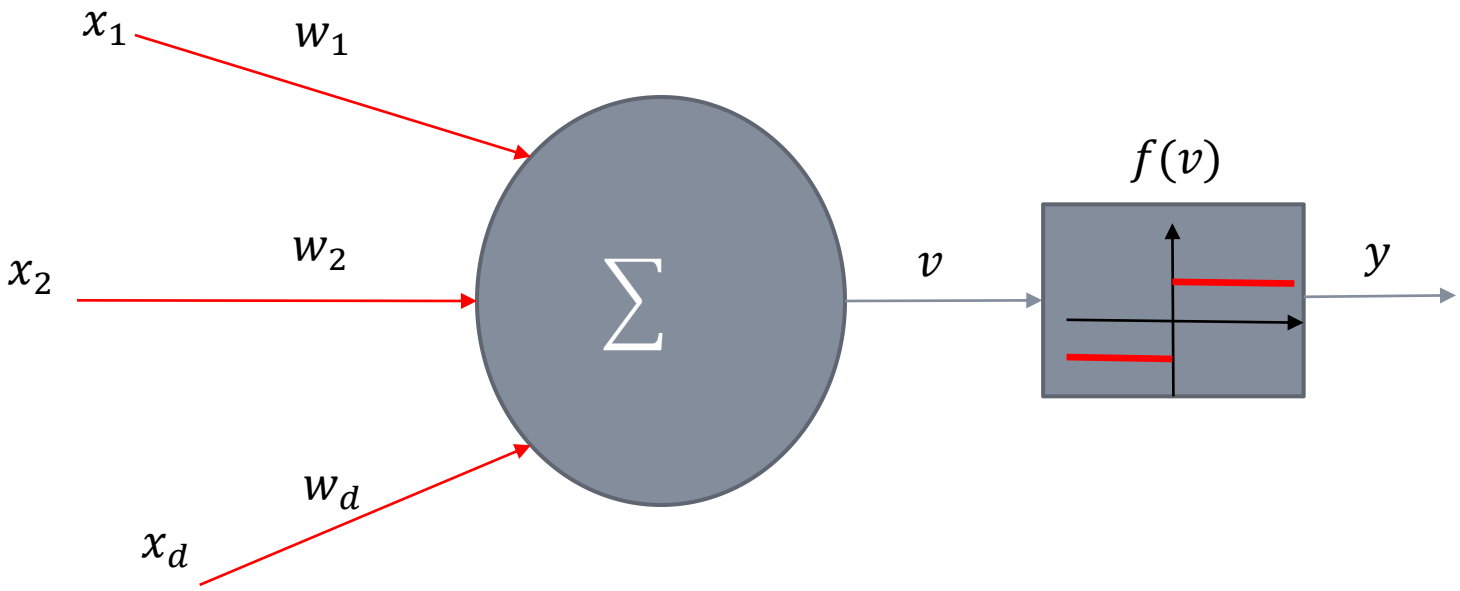
The input of a neuron is a vector $\vec{x}$

The output of the neuron is a scalar value

The importance of a component of $\vec{x}$ is weighted with a value w_i

The processing unit summarized the inputs signals (mathematically it is expressed by a sum).

What constraint(s) must be satisfied by the activation function?

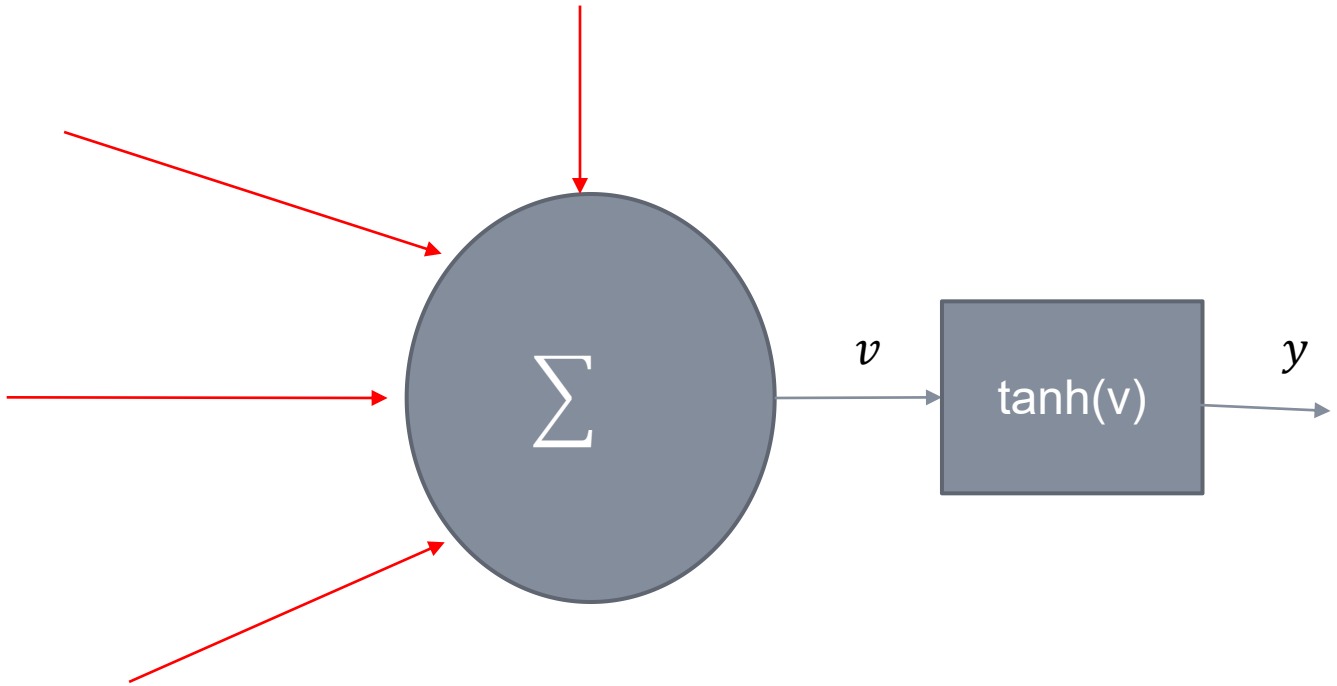

x_1
w_1
x_2
w_2
w_d
x_d
$\sum$
v
$f(v)$
y

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

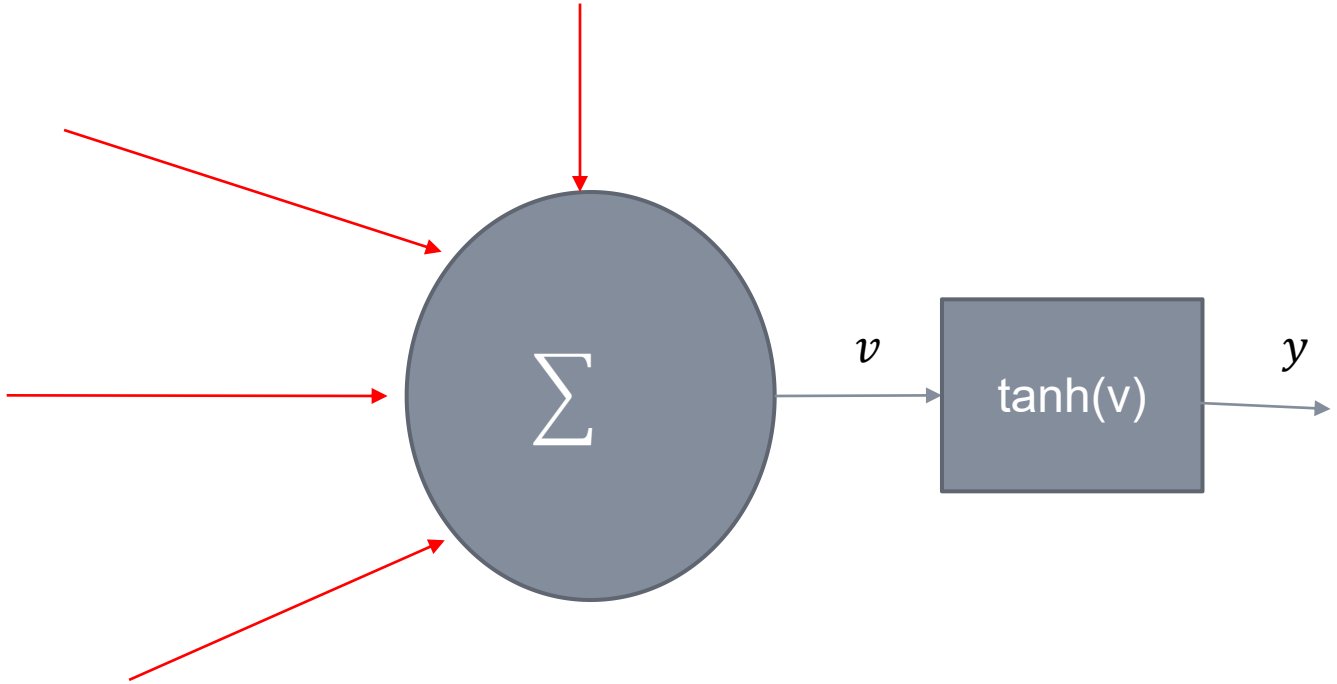
Dataset



Artificial Neuron

Training a neuron (perceptron).

Step 1: A random vector W is generated



X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

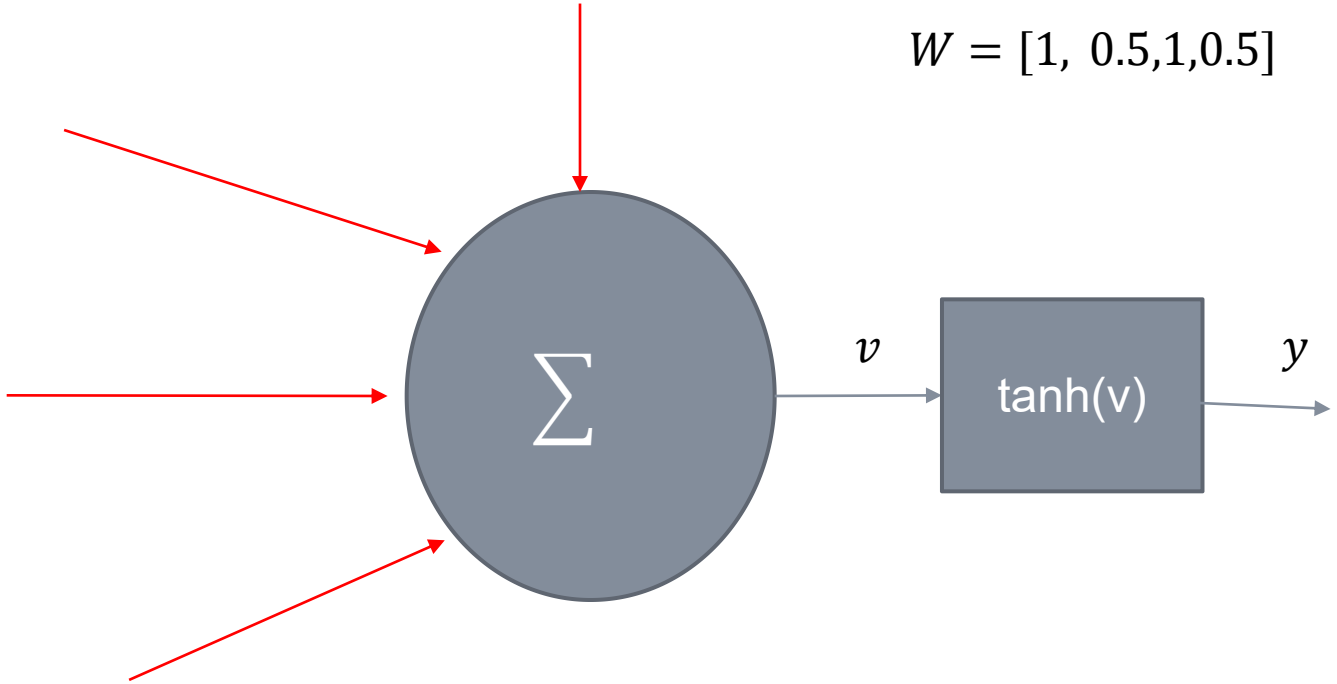
Dataset

Artificial Neuron

Training a neuron (perceptron).

Step 1: A random vector W is generated

$$W = [1,\ 0.5,1,0.5]$$



X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

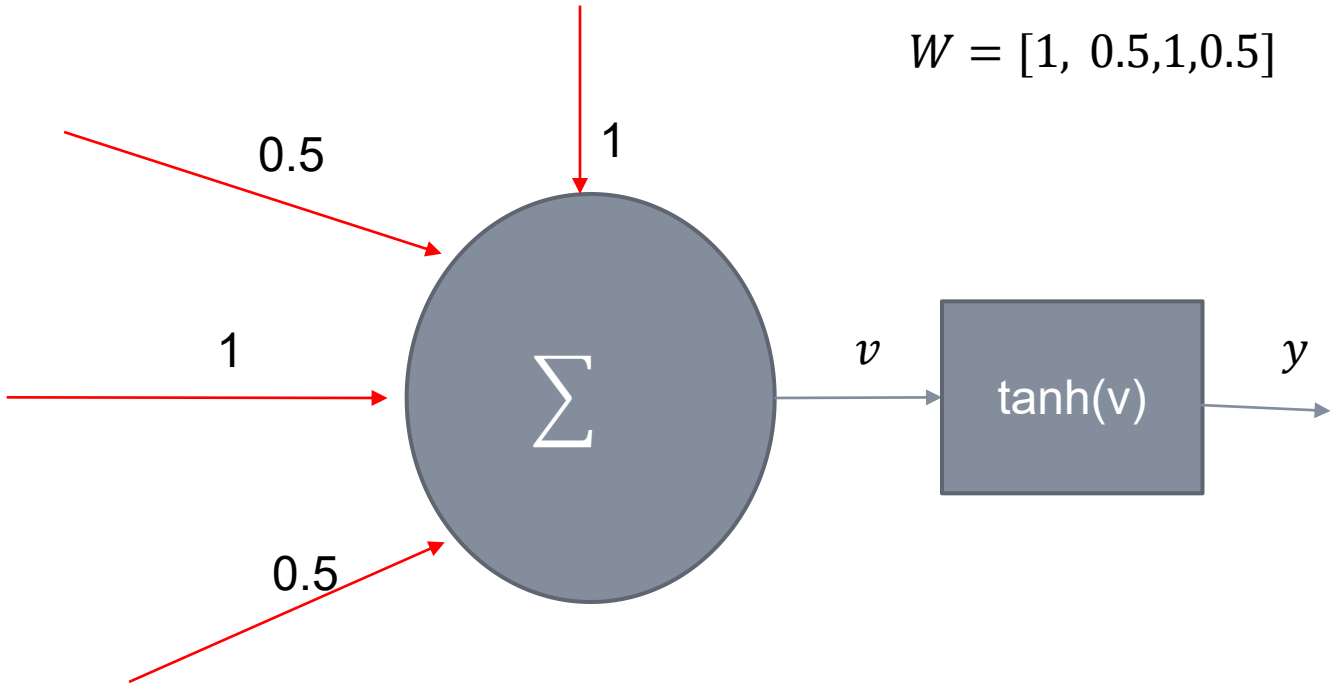
Dataset

Artificial Neuron

Training a neuron (perceptron).

Step 1: A random vector W is generated

$$W = [1,\ 0.5,1,0.5]$$



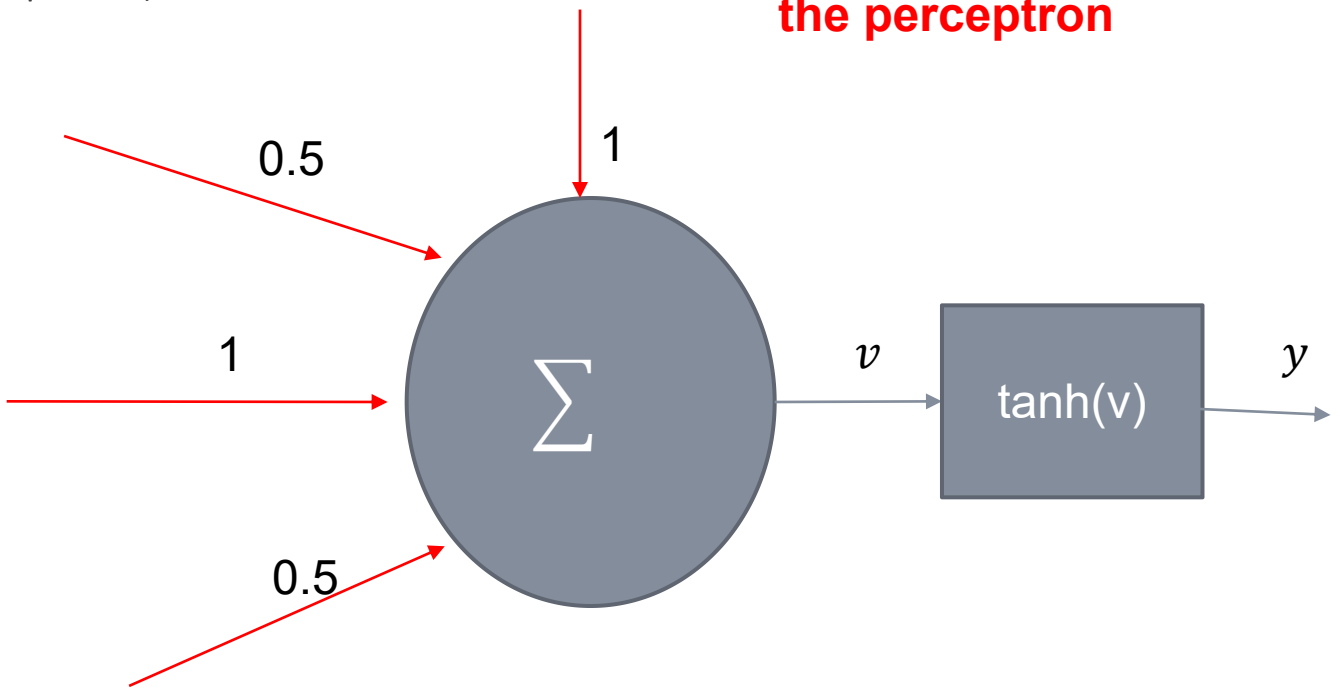
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

Artificial Neuron

Training a neuron (perceptron).

Step 2: The first pattern is presented to the perceptron



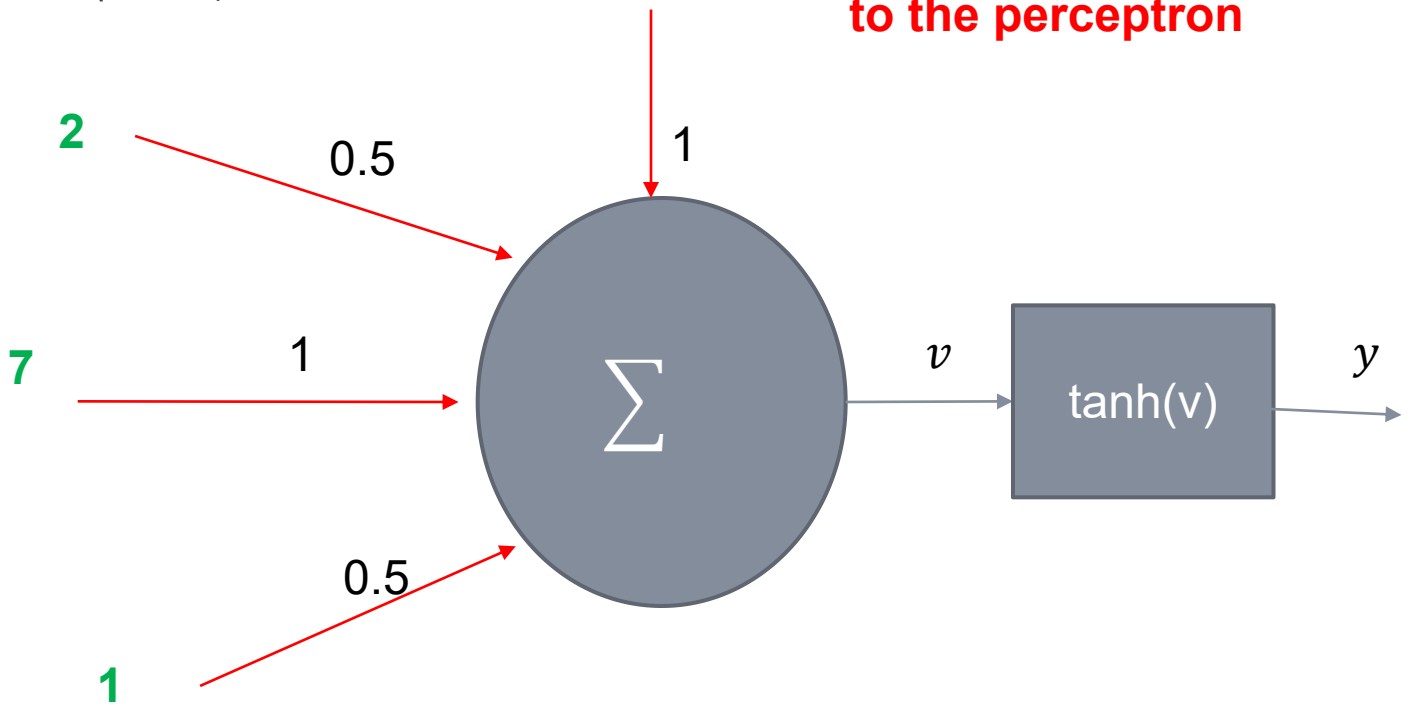
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

Artificial Neuron

Training a neuron (perceptron).

Step 2: The first pattern is presented to the perceptron



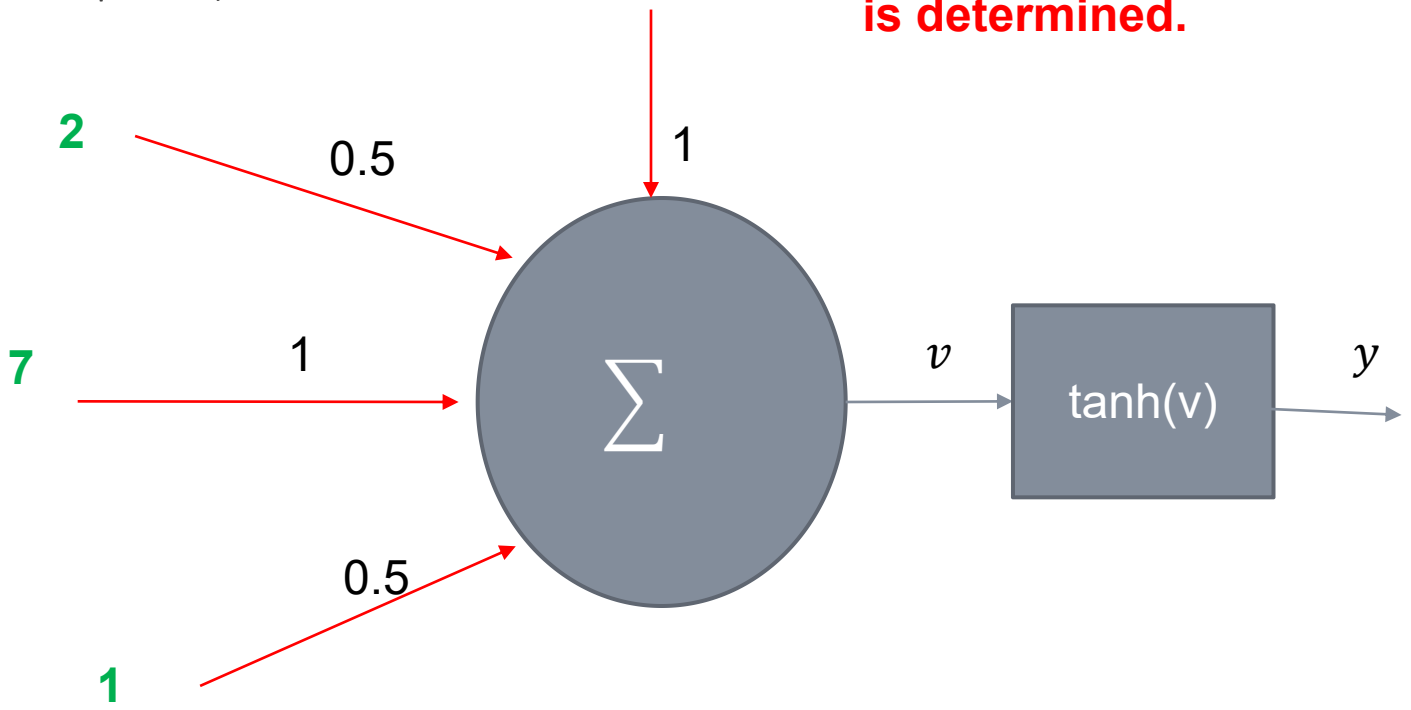
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

Artificial Neuron

Training a neuron (perceptron).

Step 3: The output of the perceptron is determined.



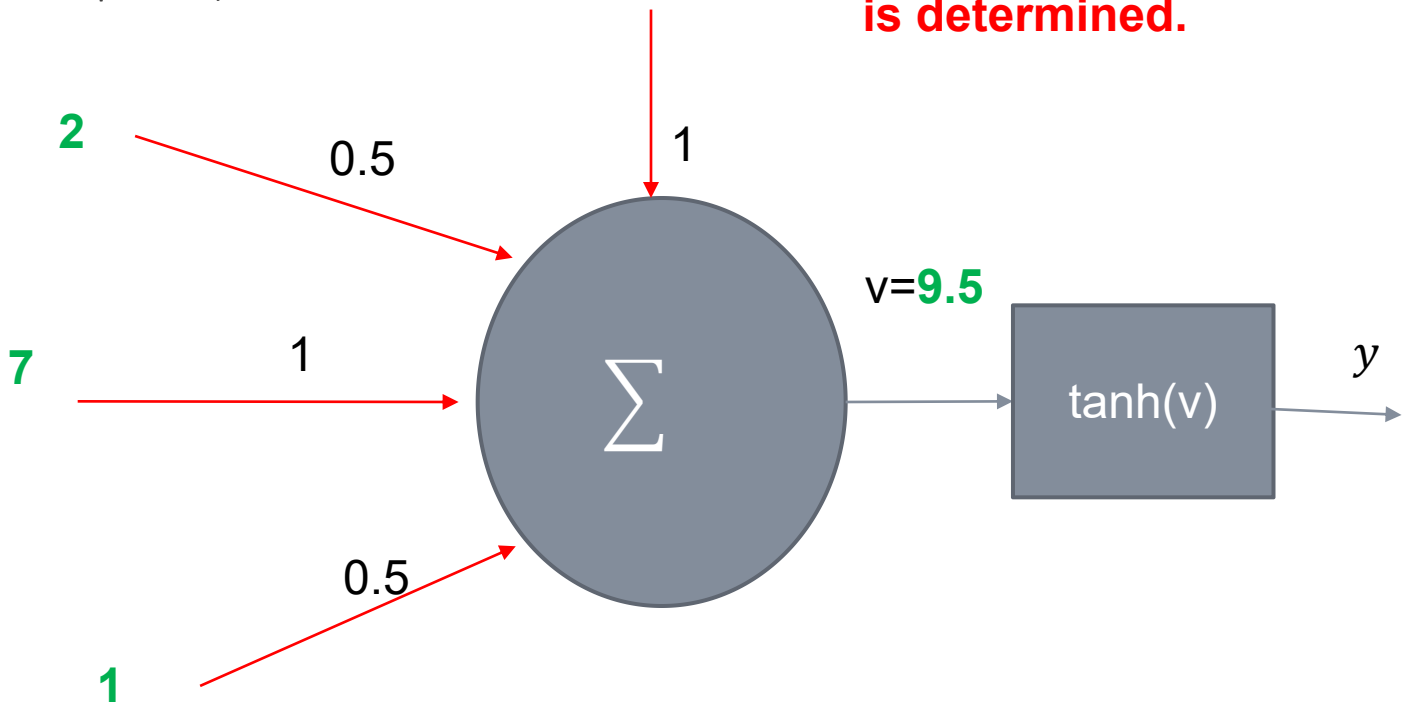
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

Artificial Neuron

Training a neuron (perceptron).

Step 3: The output of the perceptron is determined.



X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

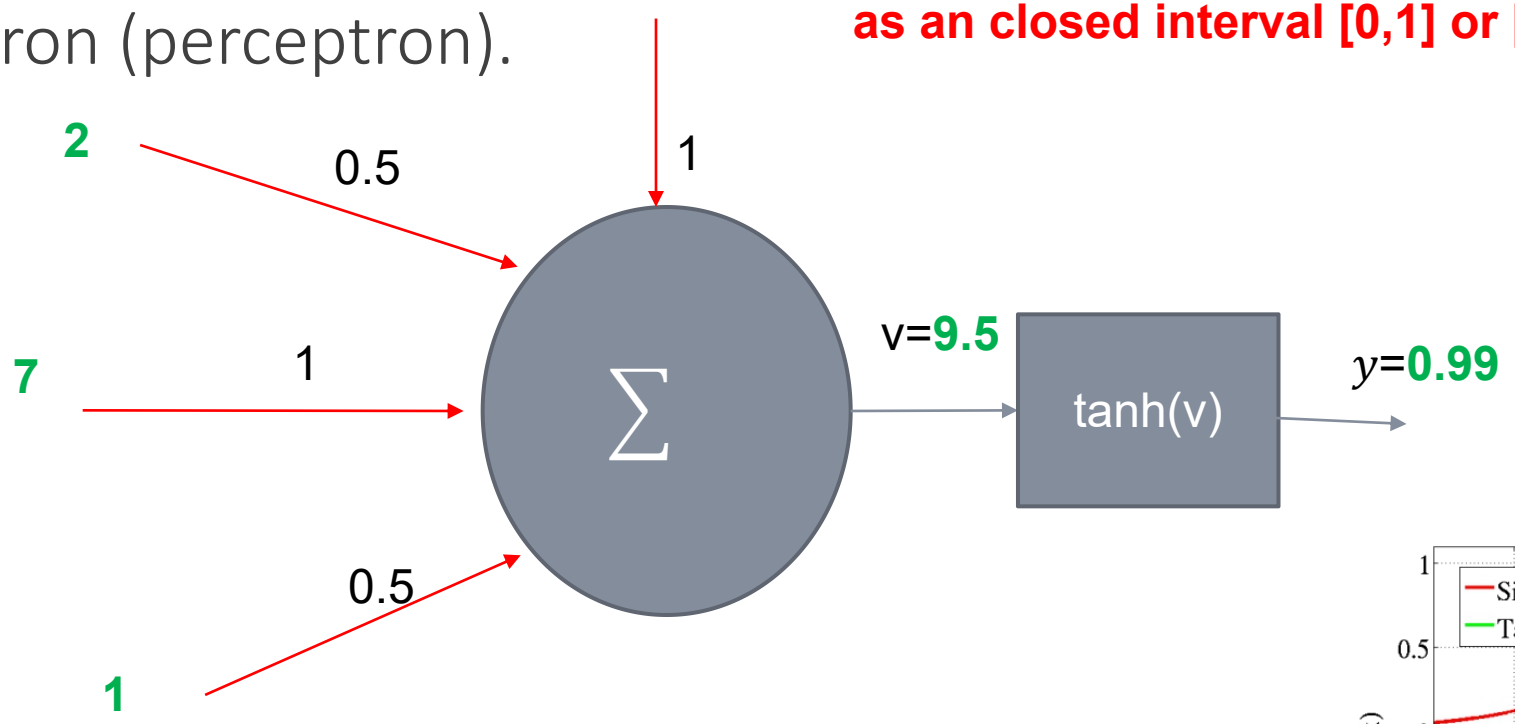
Artificial Neuron

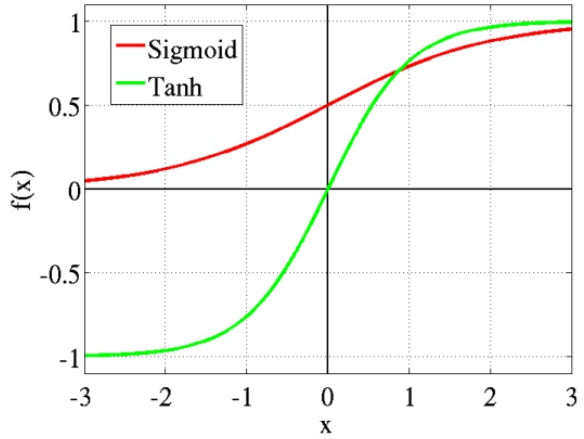
Training a neuron (perceptron).

Step 4: The value of y is calculated based on activation function. Recall that such function allows to normalize the output as an closed interval [0,1] or [-1, 1].

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset





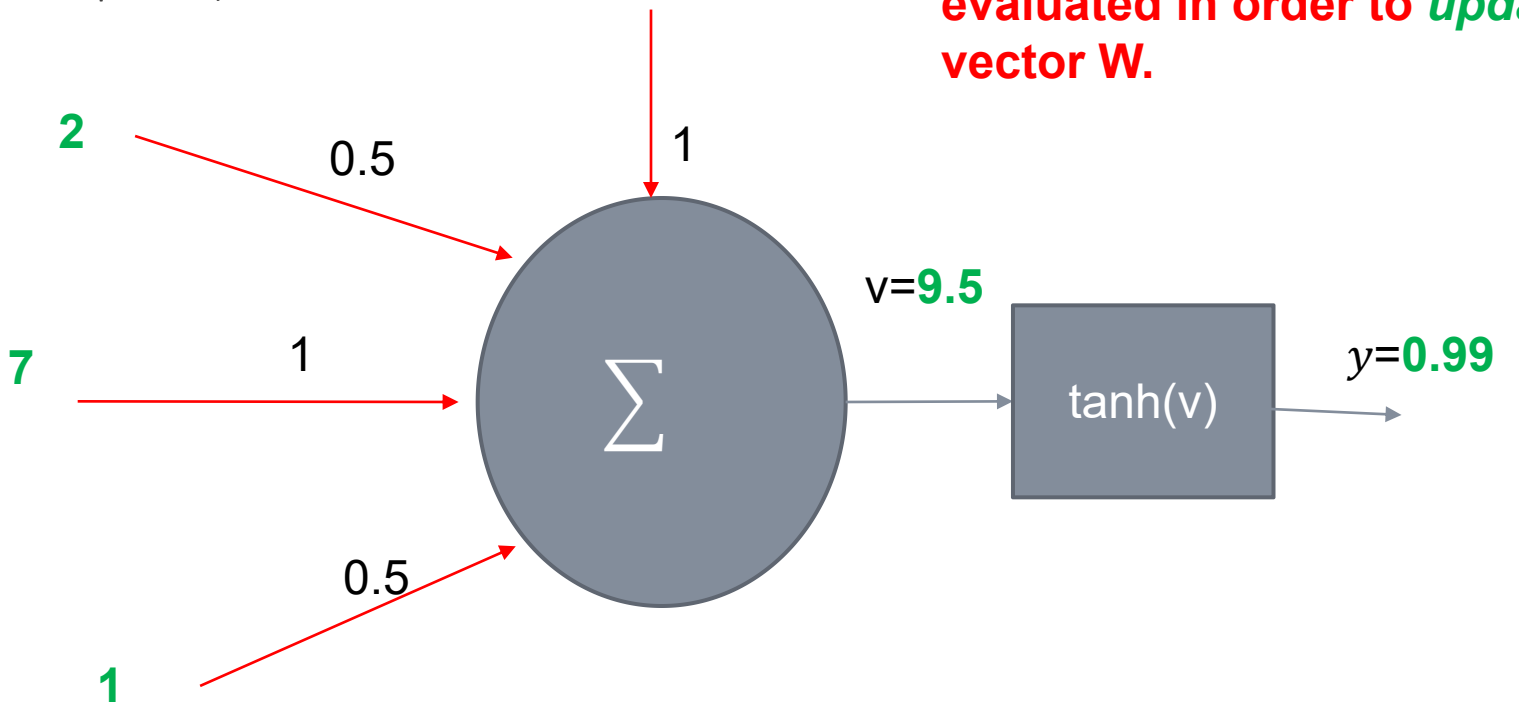
Artificial Neuron

Training a neuron (perceptron).

Step 5: The value of y is evaluated in order to *update* the vector W.

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

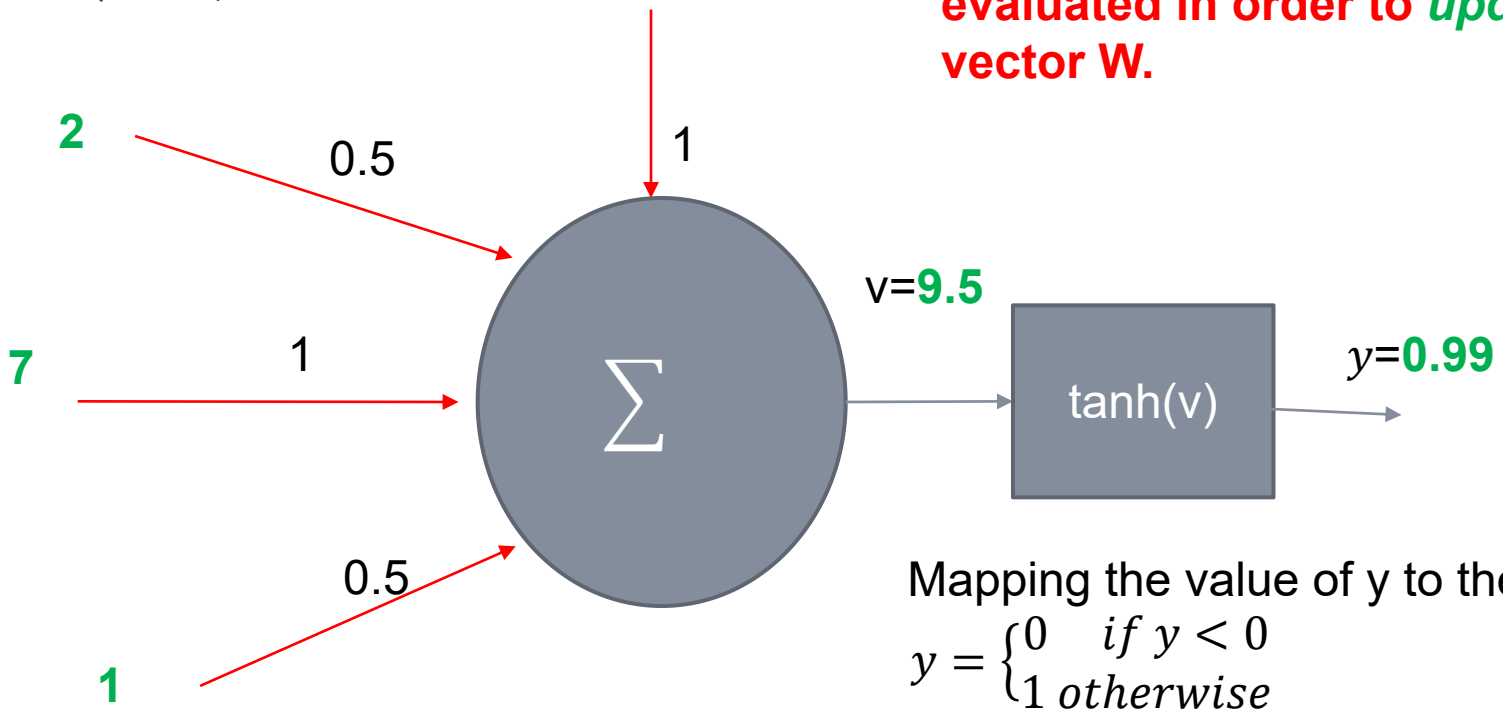


Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The value of y is evaluated in order to *update* the vector W.

Mapping the value of y to the class label.

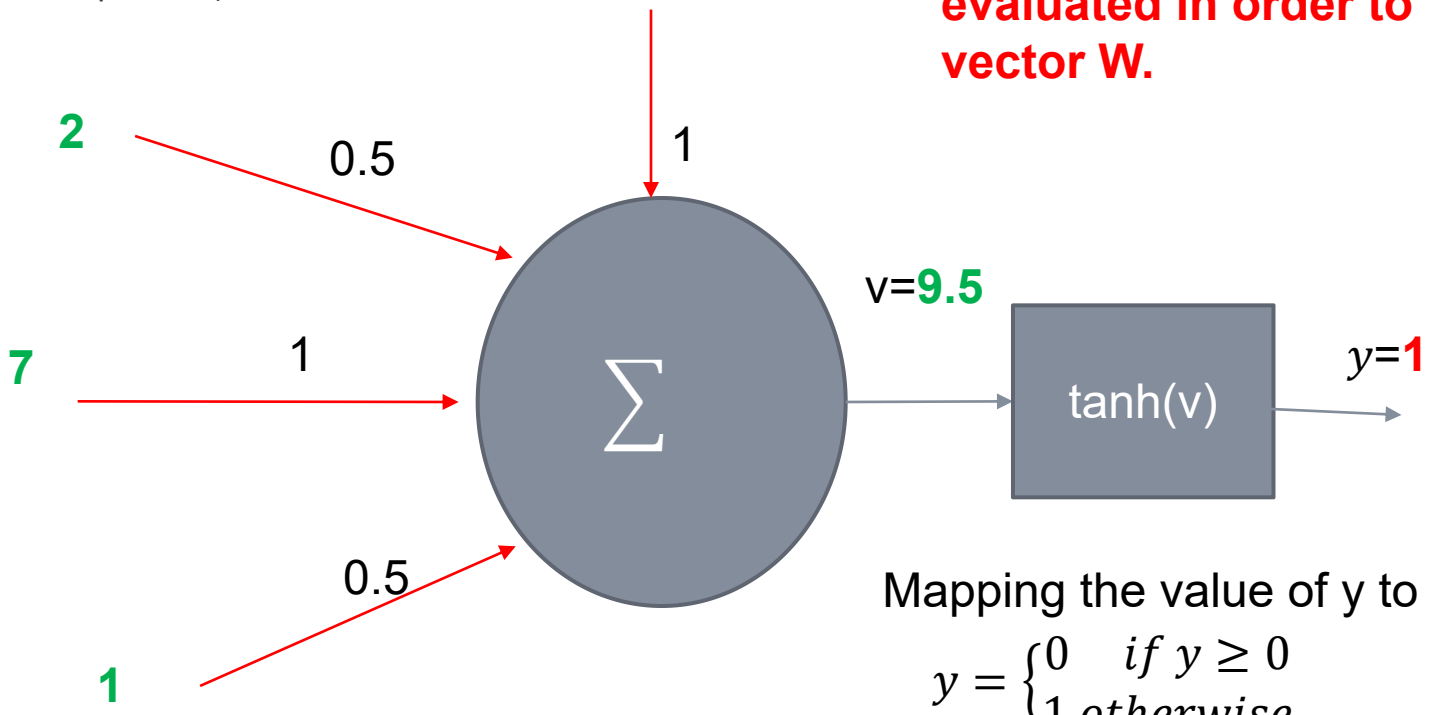
$$y = \begin{cases} 0 & if\ y < 0 \\ 1 & otherwise \end{cases}$$

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The value of y is evaluated in order to *update* the vector W.

Mapping the value of y to the class label.

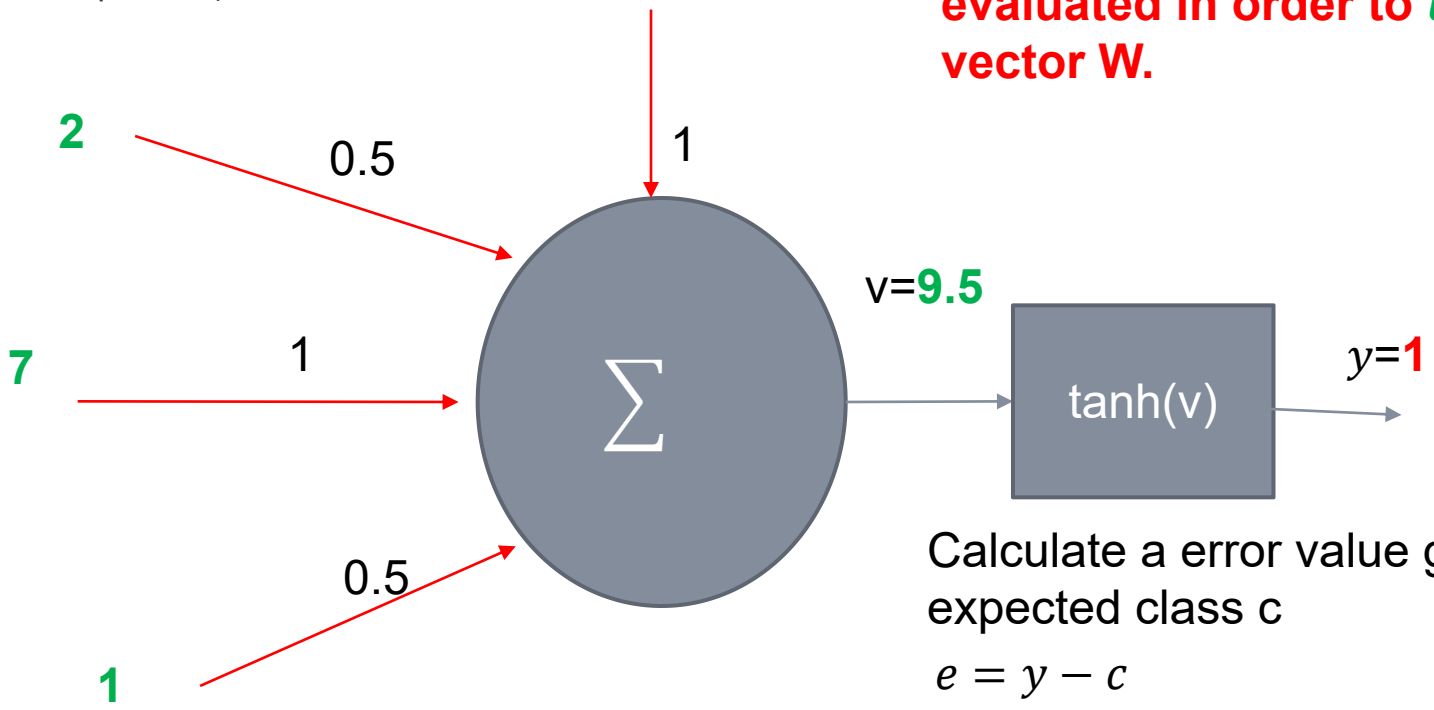
$$y = \begin{cases} 0 & if\ y \geq 0 \\ 1 & otherwise \end{cases}$$

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The value of y is evaluated in order to *update* the vector W.

Calculate a error value given the expected class c

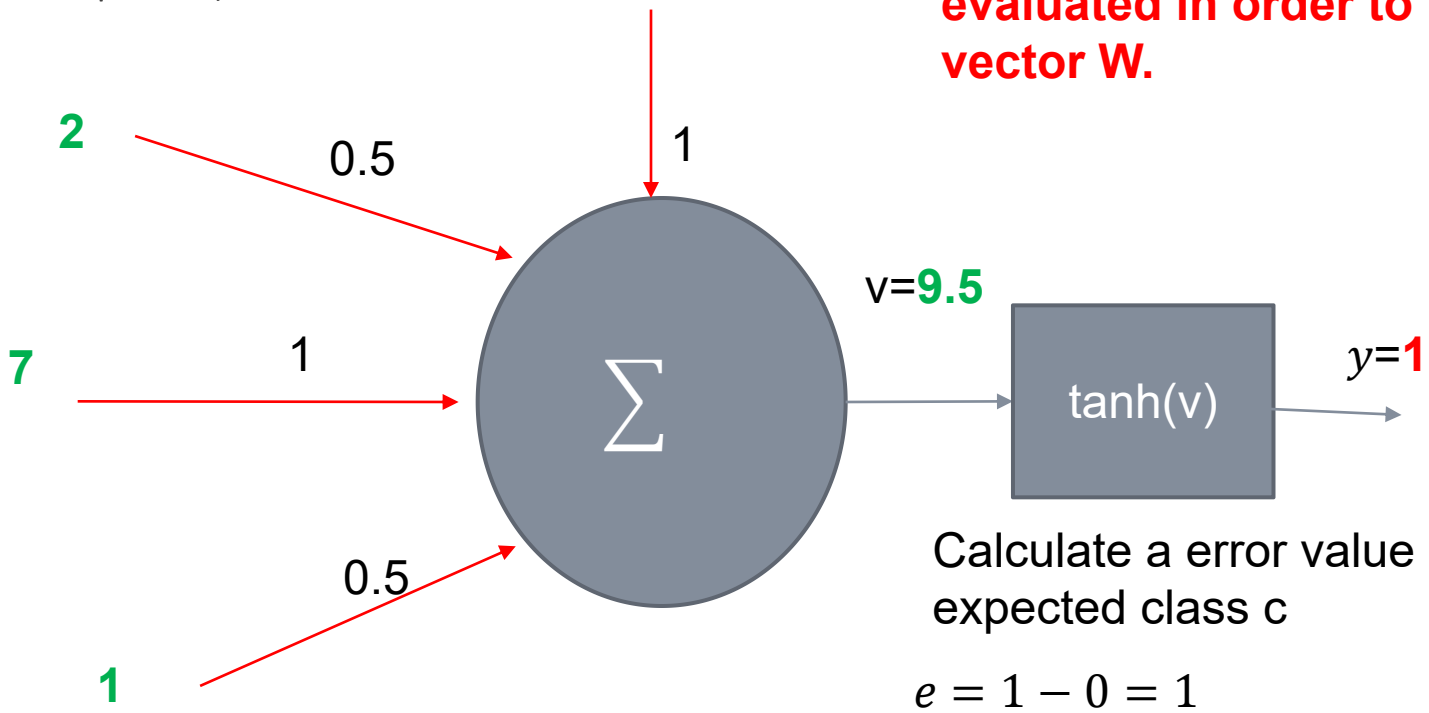
$e = y - c$

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The value of y is evaluated in order to *update* the vector W.

Calculate a error value given the expected class c

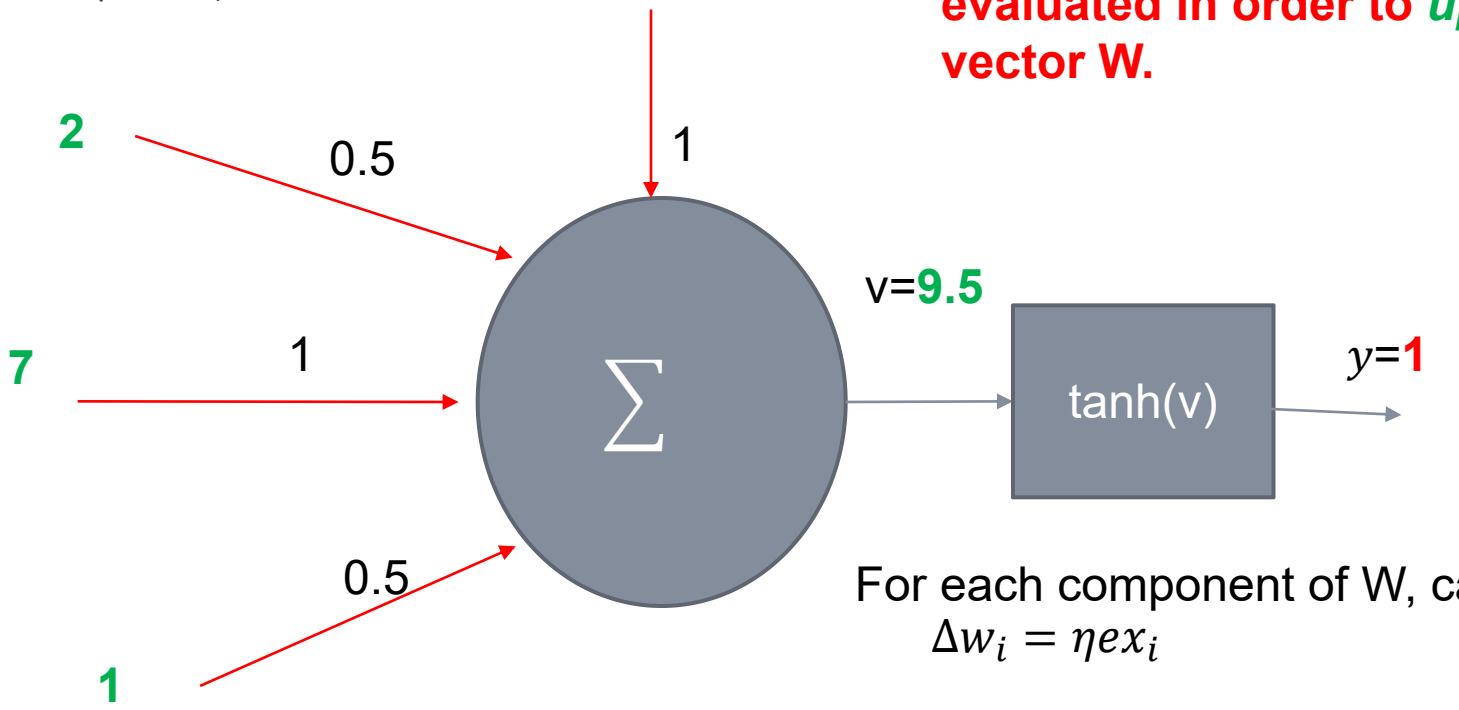
$$e = 1 - 0 = 1$$

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The value of y is evaluated in order to *update* the vector W.

For each component of W, calculate a variation Δw_i

$$\Delta w_i = \eta e x_i$$

Where η is a perturbation value, usually it is a small positive value. Assume η=0.01

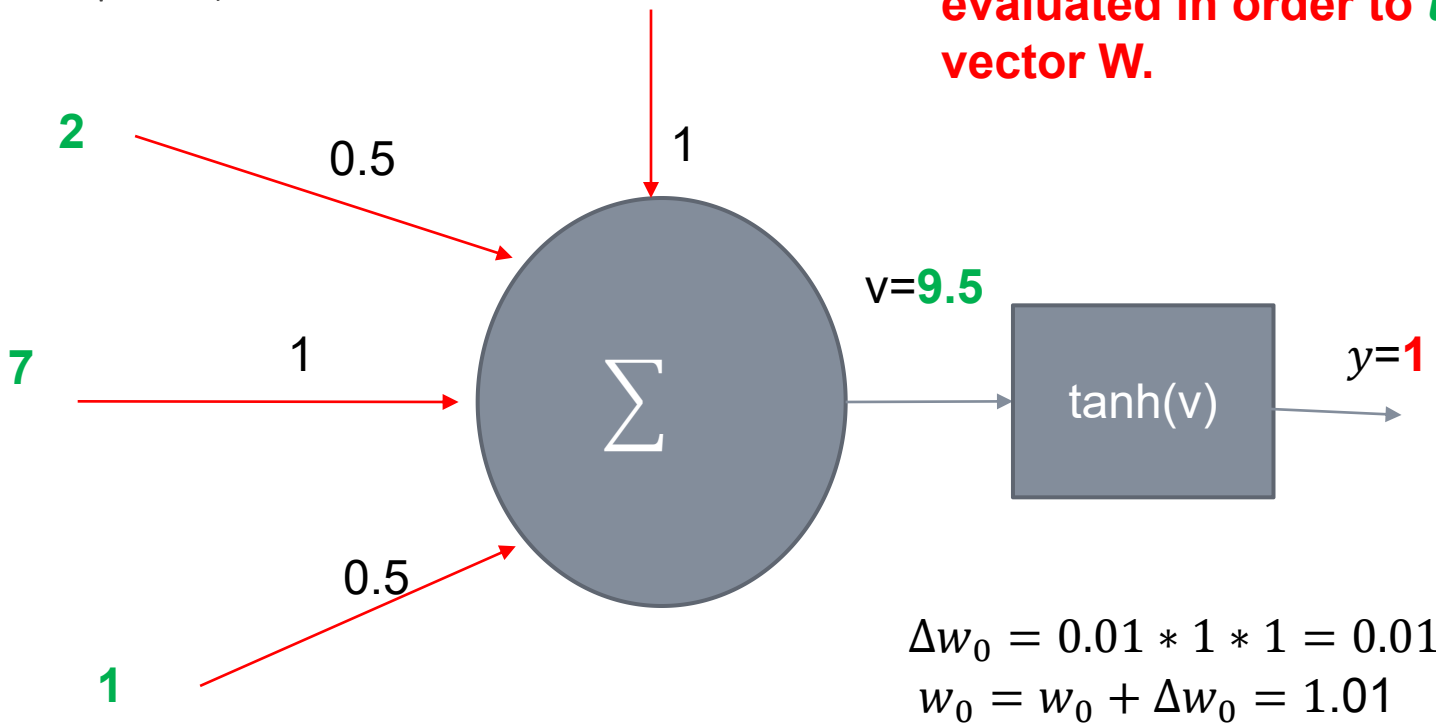
Update $w_i = w_i + \Delta w_i$

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The value of y is evaluated in order to *update* the vector W.

$$\Delta w_0 = 0.01 * 1 * 1 = 0.011$$
$$w_0 = w_0 + \Delta w_0 = 1.01$$

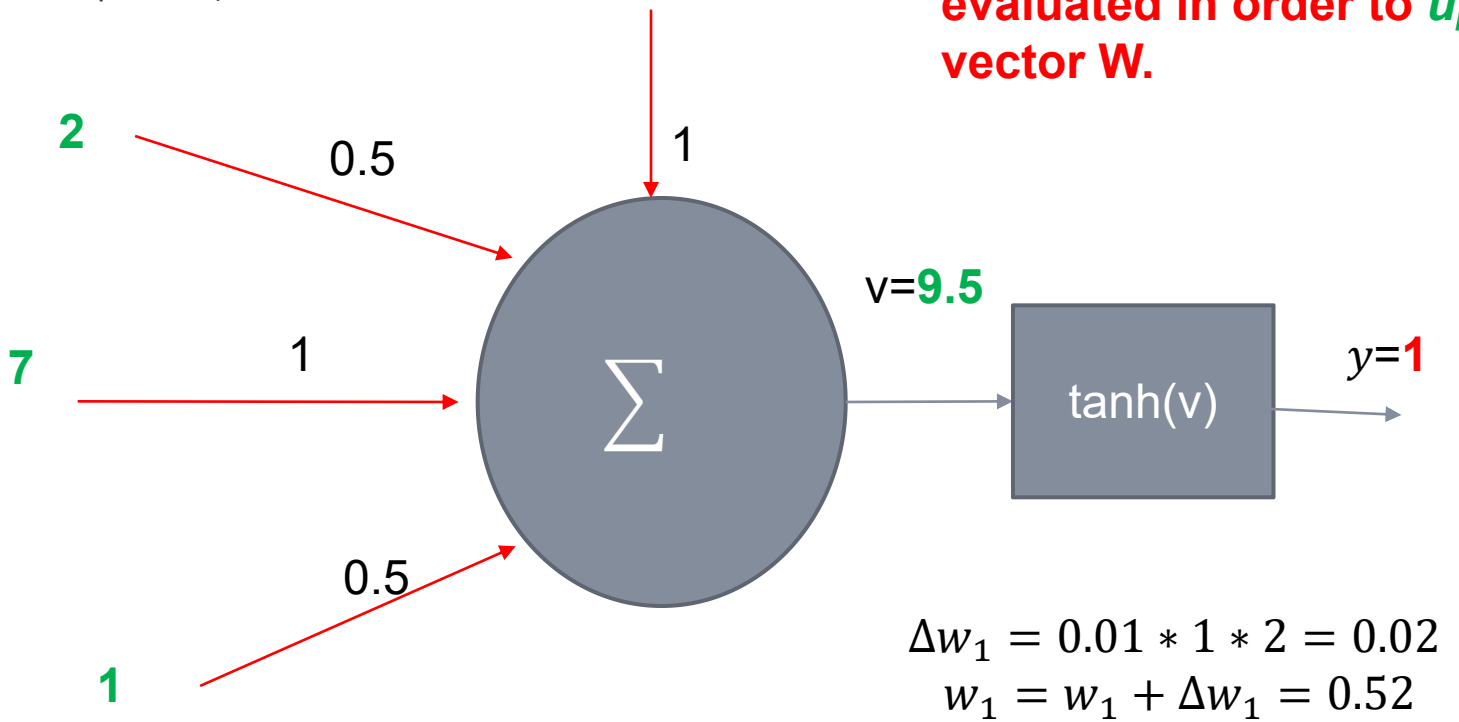
$$W = [1.01$$

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The value of y is evaluated in order to *update* the vector W.

$$\Delta w_1 = 0.01 * 1 * 2 = 0.02$$
$$w_1 = w_1 + \Delta w_1 = 0.52$$

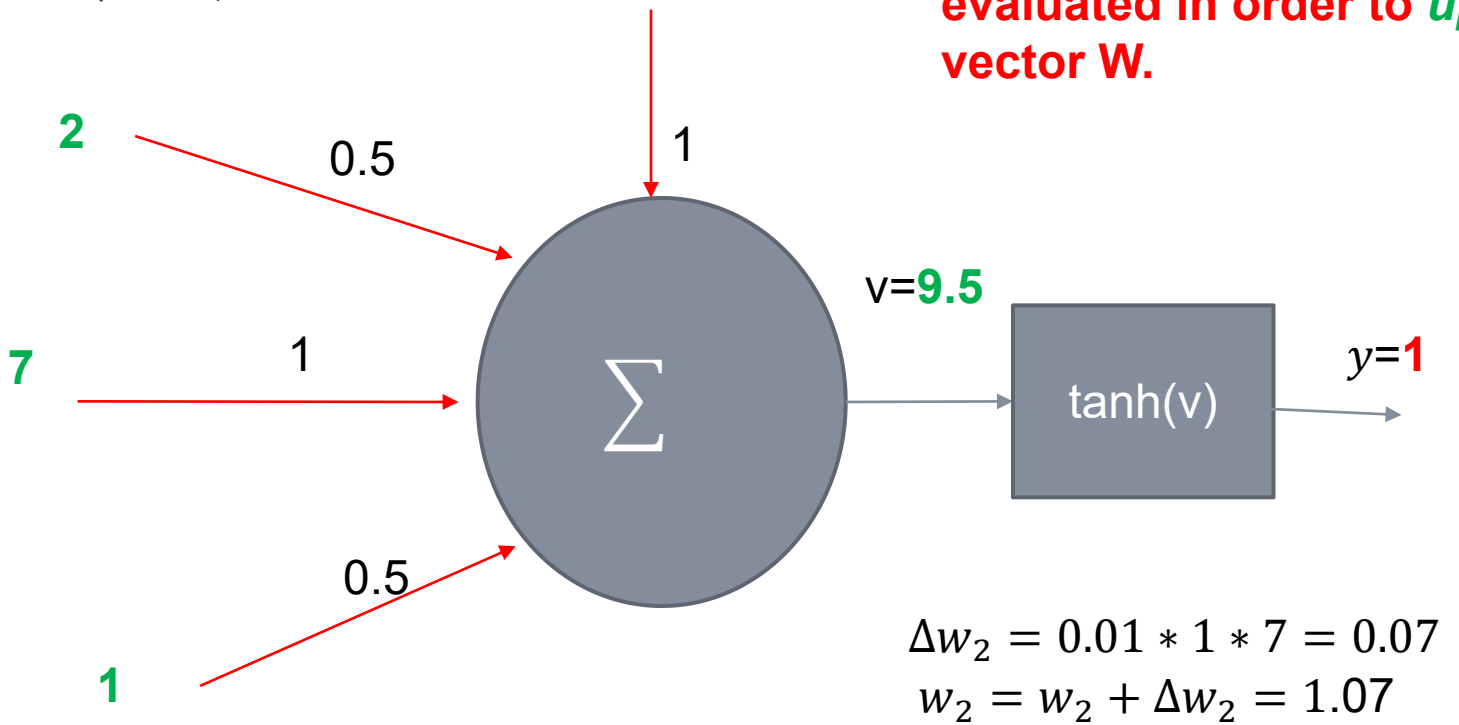
$$W = [1.01, 0.52$$

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The value of y is evaluated in order to *update* the vector W.

$$\Delta w_2 = 0.01 * 1 * 7 = 0.07$$
$$w_2 = w_2 + \Delta w_2 = 1.07$$

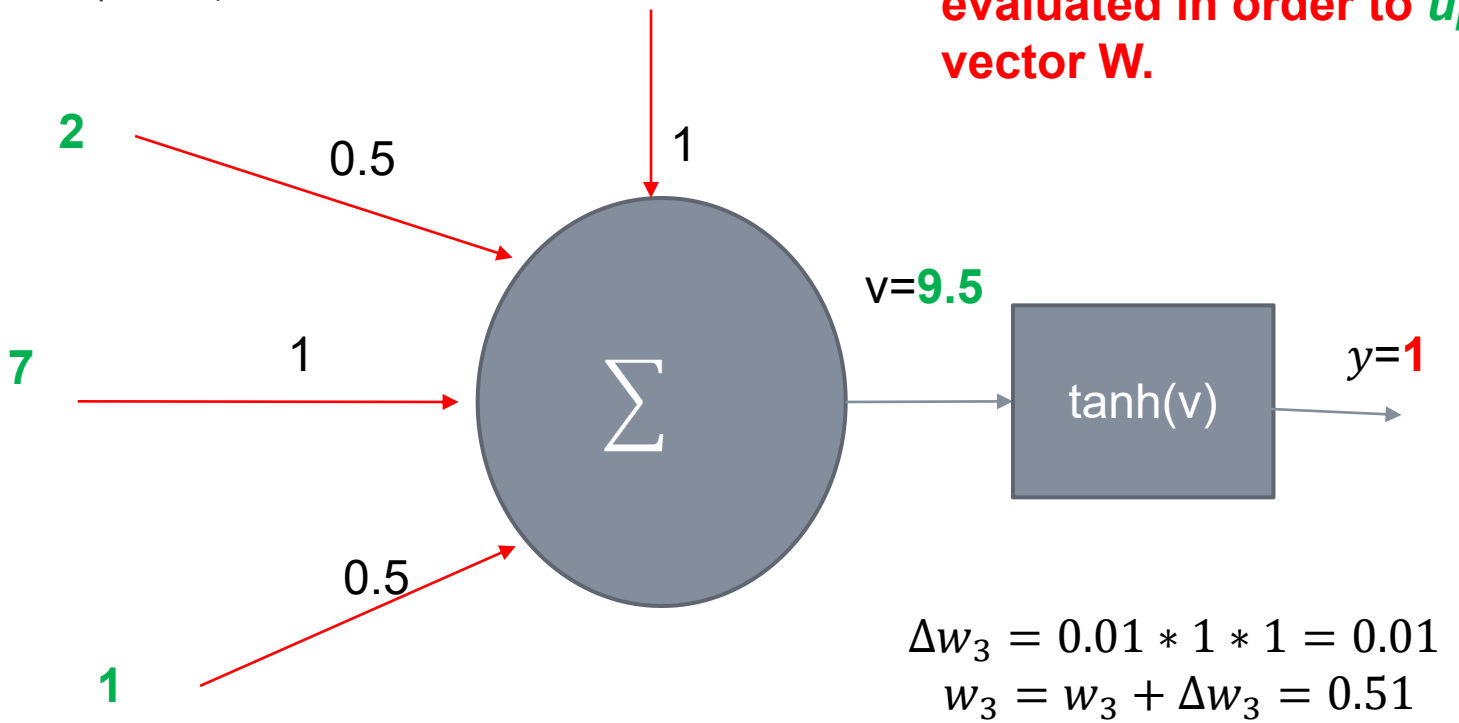
$$W = [1.01,\ 0.52, 1.07,$$

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The value of y is evaluated in order to *update* the vector W.

$$\Delta w_3 = 0.01 * 1 * 1 = 0.01$$
$$w_3 = w_3 + \Delta w_3 = 0.51$$

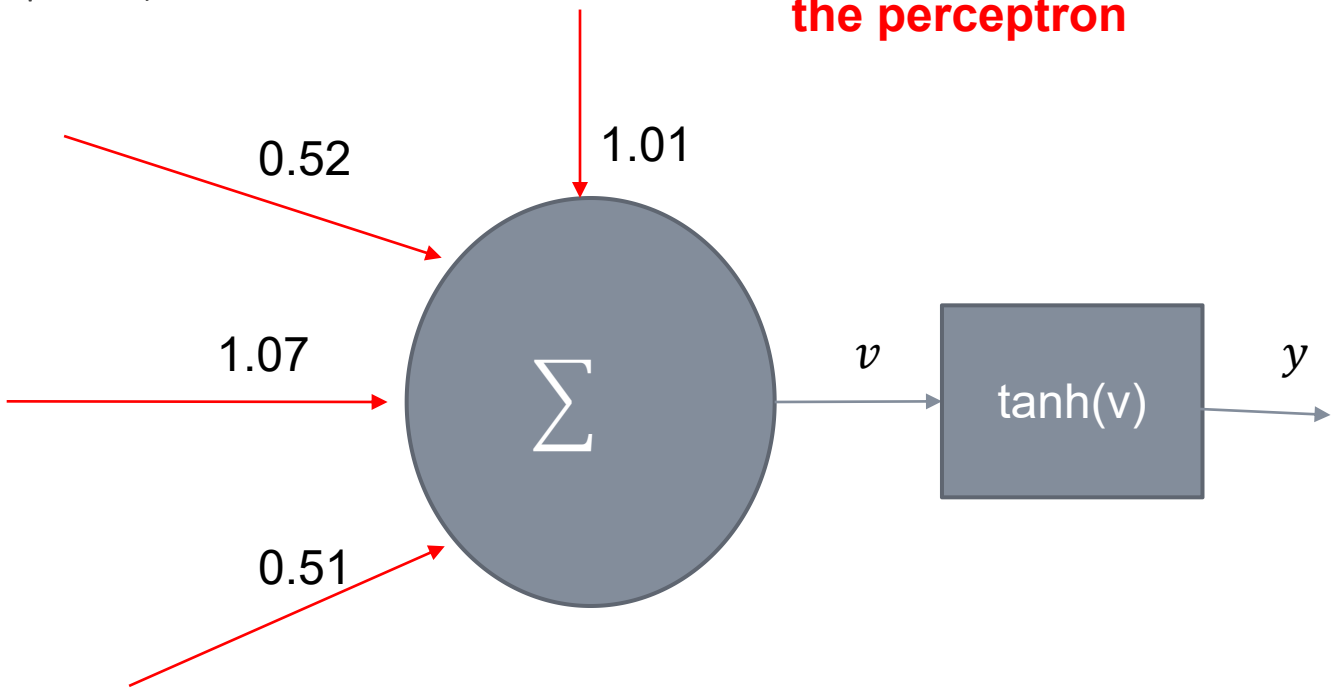
Repeat steps 2-5 again with **other pattern**

$$W = [1.01,\ 0.52, 1.07, 0.51]$$

Artificial Neuron

Training a neuron (perceptron).

Step 2: The first pattern is presented to the perceptron



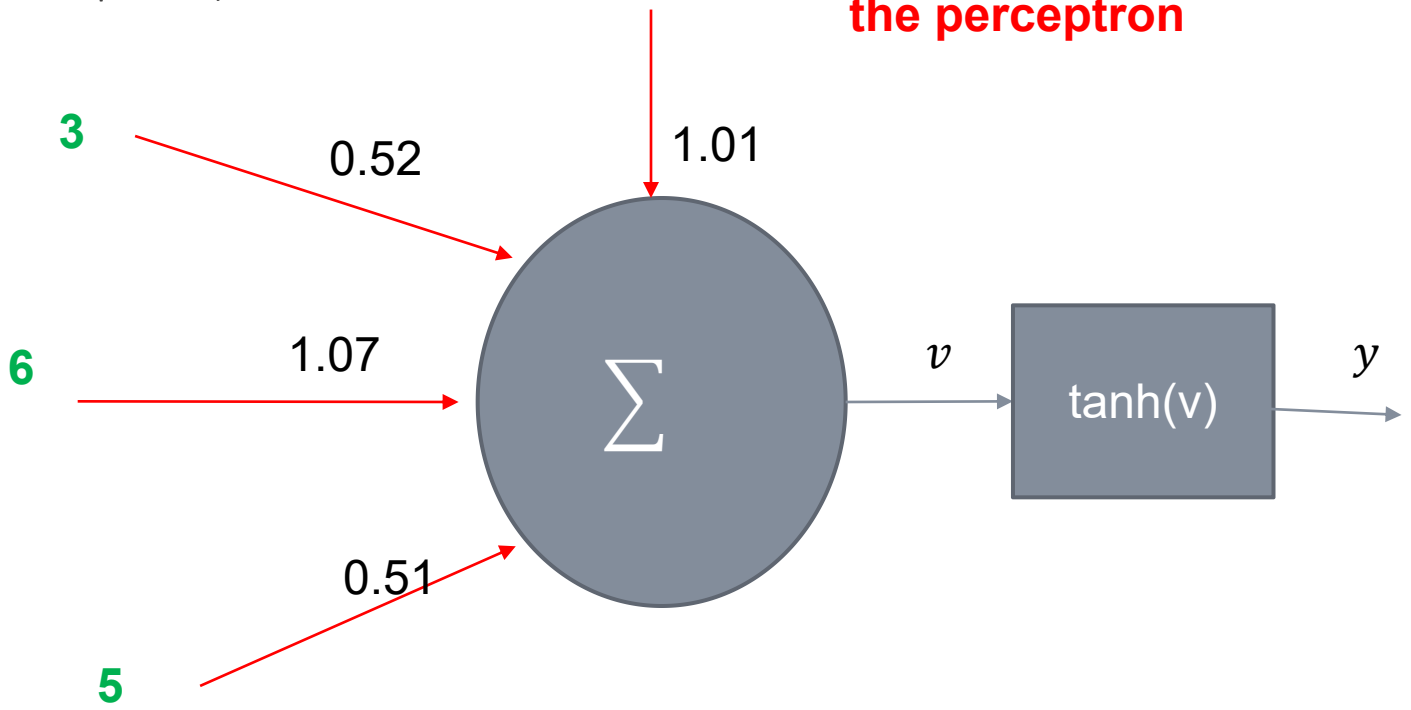
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

Artificial Neuron

Training a neuron (perceptron).

Step 2: The first pattern is presented to the perceptron



X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

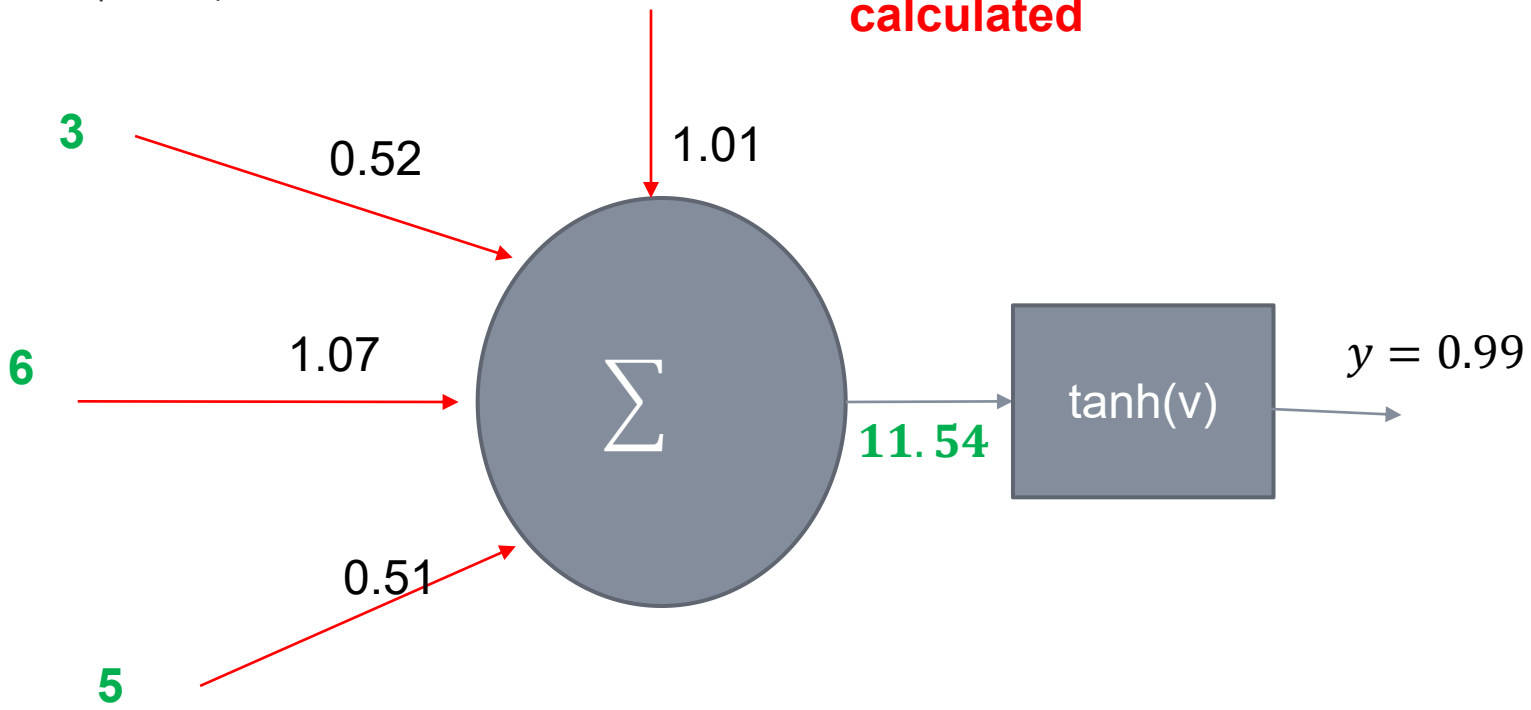
Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

Step 3: The output of the perceptron is calculated



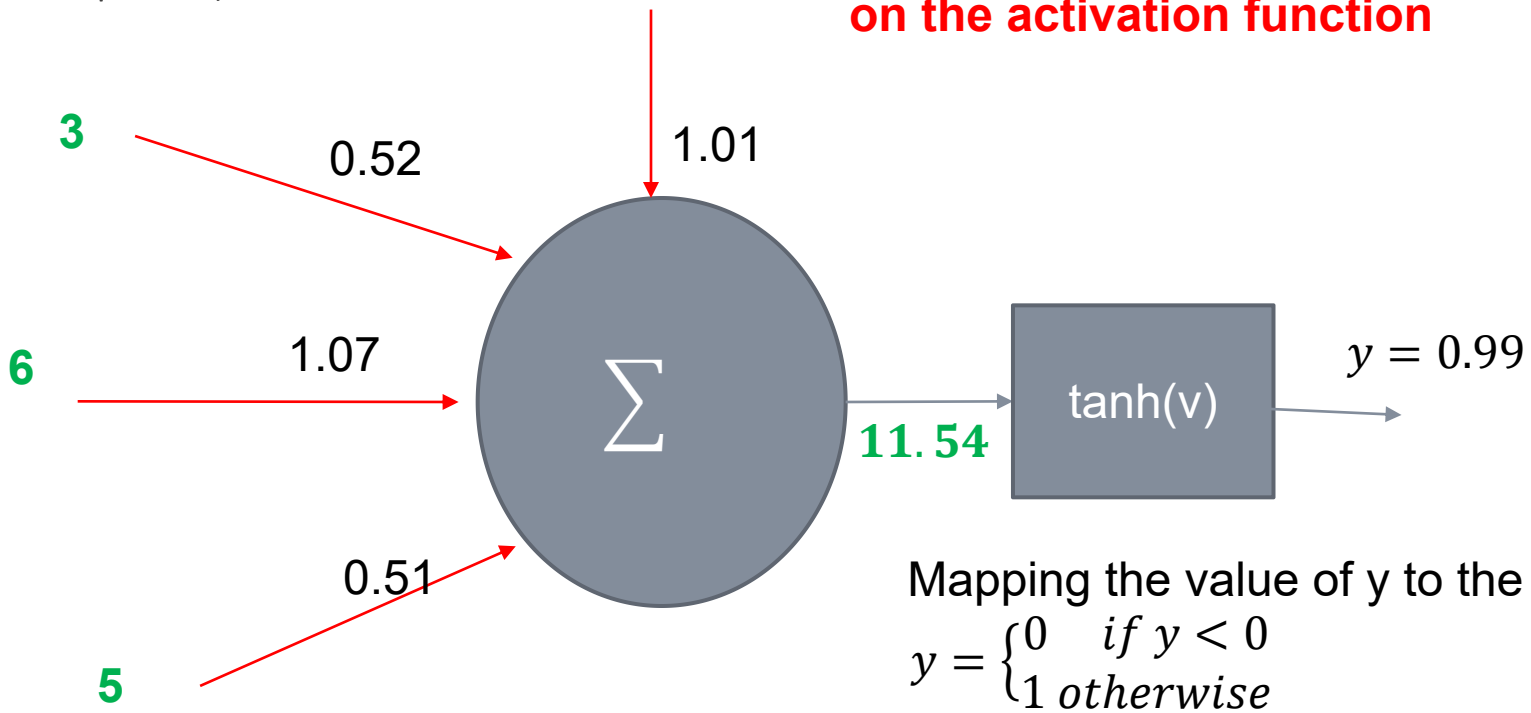
Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

Step 4: The final output of is calculated based on the activation function



Mapping the value of y to the class label.

$$y = \begin{cases} 0 & if\ y < 0 \\ 1 & otherwise \end{cases}$$

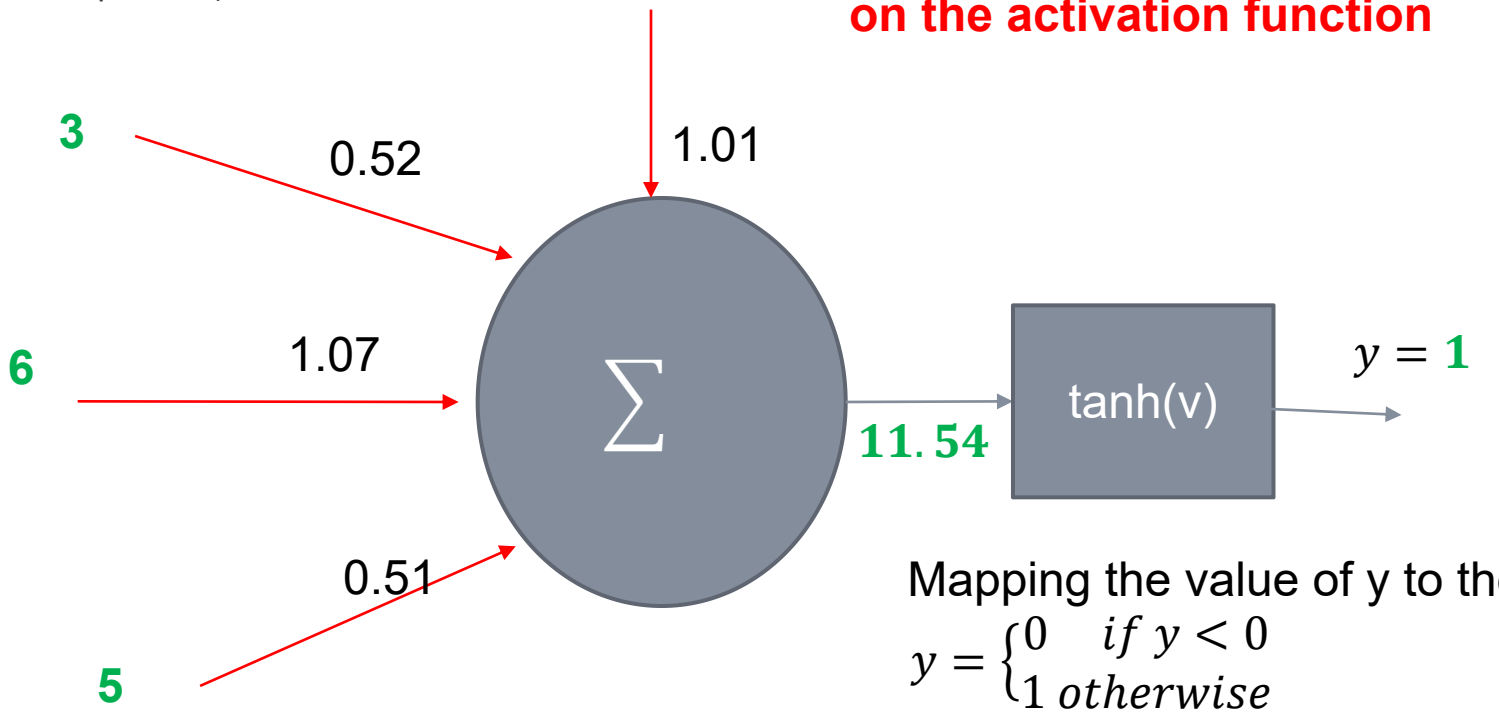
Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

Step 4: The final output of is calculated based on the activation function



Mapping the value of y to the class label.

$$y = \begin{cases} 0 & if\ y < 0 \\ 1 & otherwise \end{cases}$$

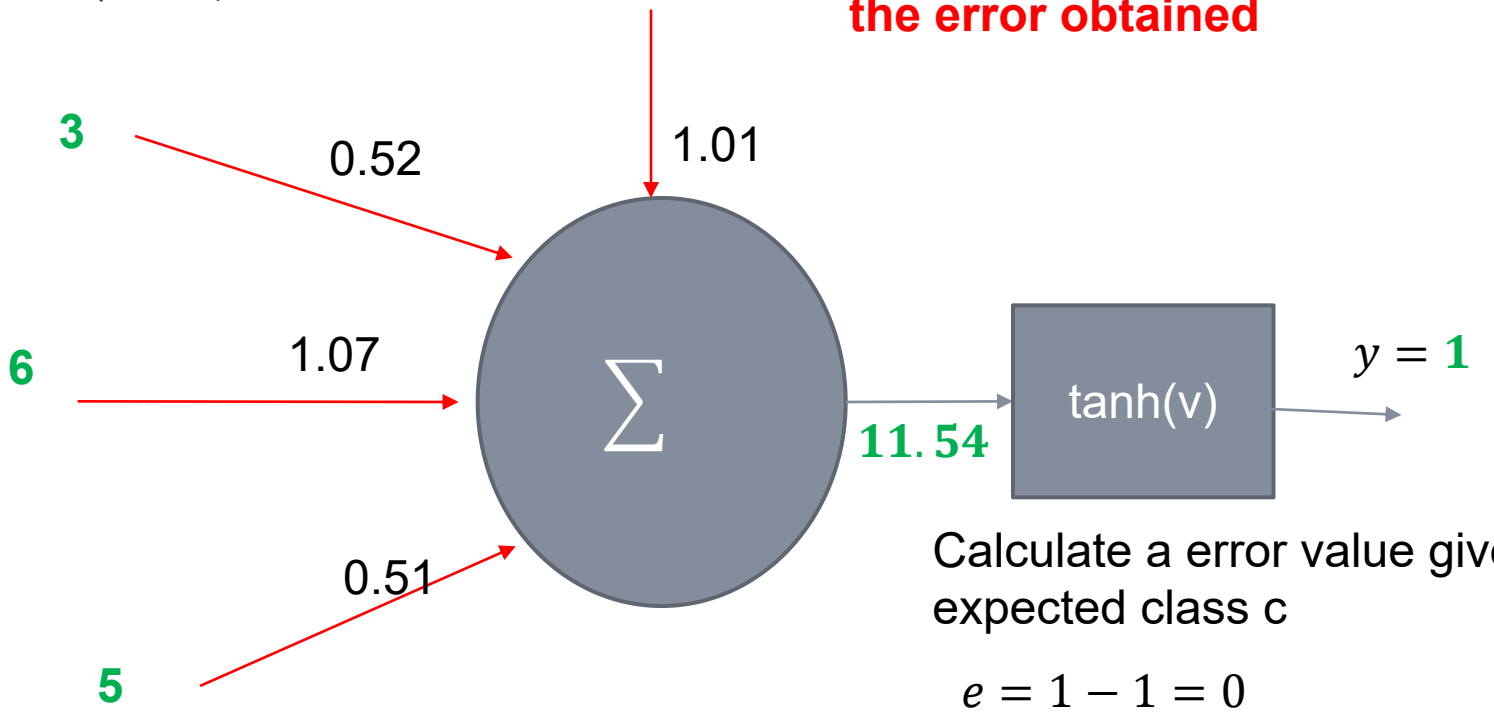
Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset

Step 5: The values of W are updated based on the error obtained



Calculate a error value given the expected class c

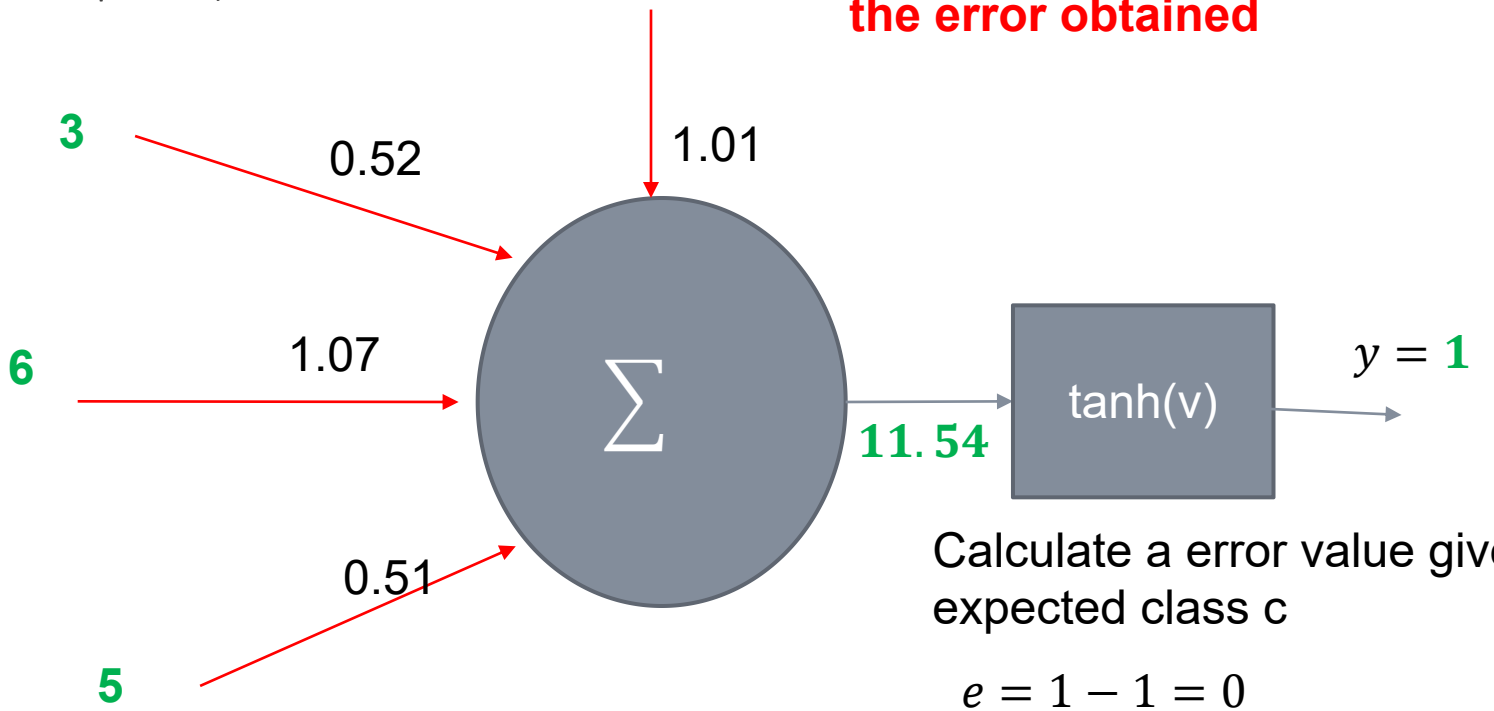
$$e = 1 - 1 = 0$$

Artificial Neuron

Training a neuron (perceptron).

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



Step 5: The values of W are updated based on the error obtained

Calculate a error value given the expected class c

$$e = 1 - 1 = 0$$

For each component of W, calculate a variation Δw_i

$$\Delta w_i = \eta e x_i$$

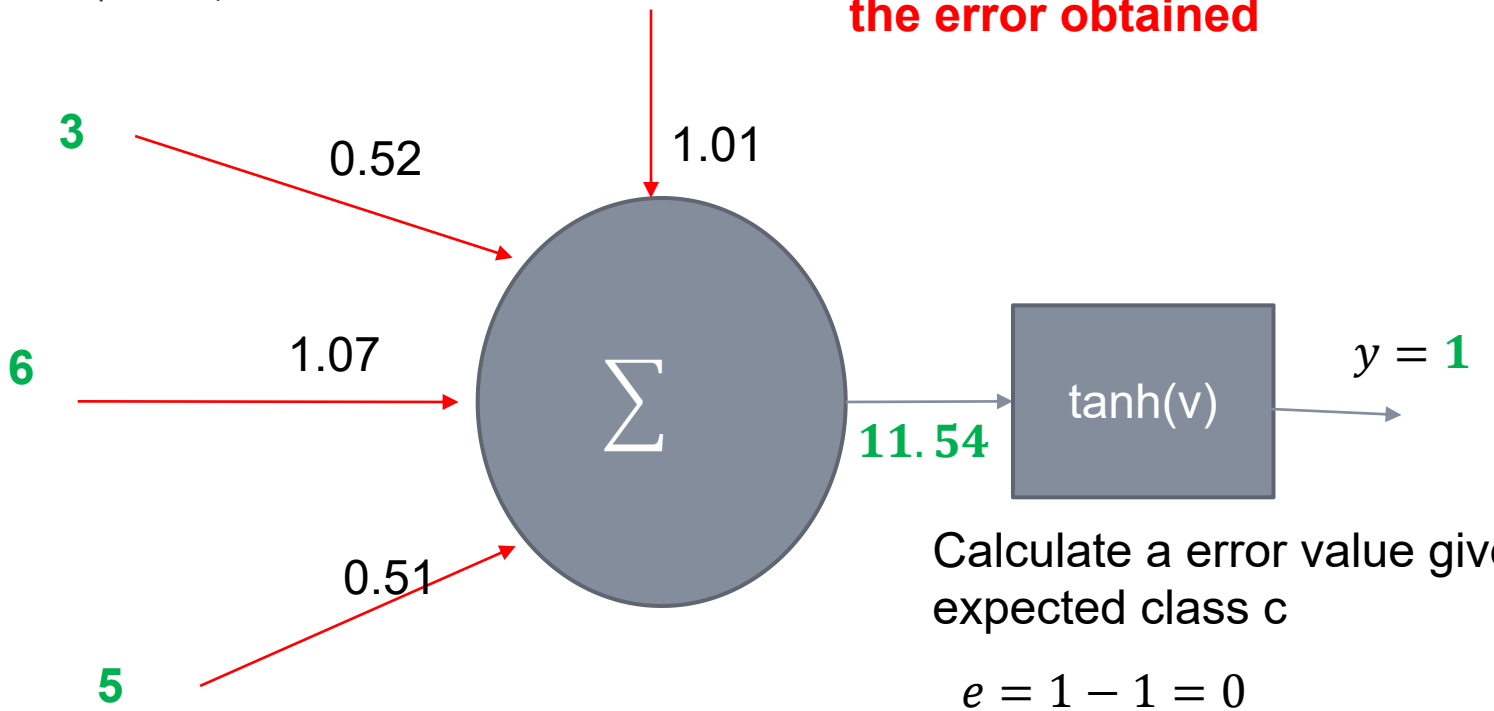
Artificial Neuron

Training a neuron (perceptron).

Step 5: The values of W are updated based on the error obtained

X1	X2	X3	c
2	7	1	0
3	**6**	**5**	1
5	4	7	0
6	8	5	0
7	9	7	1
8	0	8	1

Dataset



In this case W remains the same

Calculate a error value given the expected class c

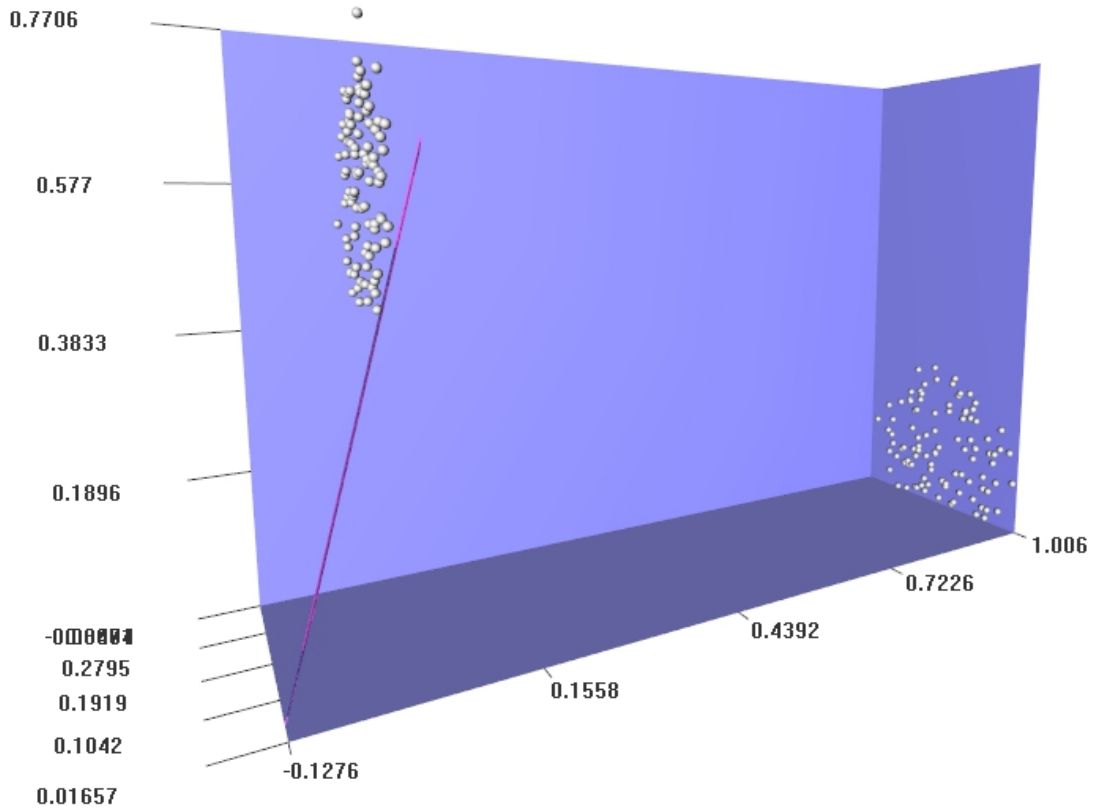
$$e = 1 - 1 = 0$$

For each component of W, calculate a variation Δw_i

$$\Delta w_i = \eta e x_i$$

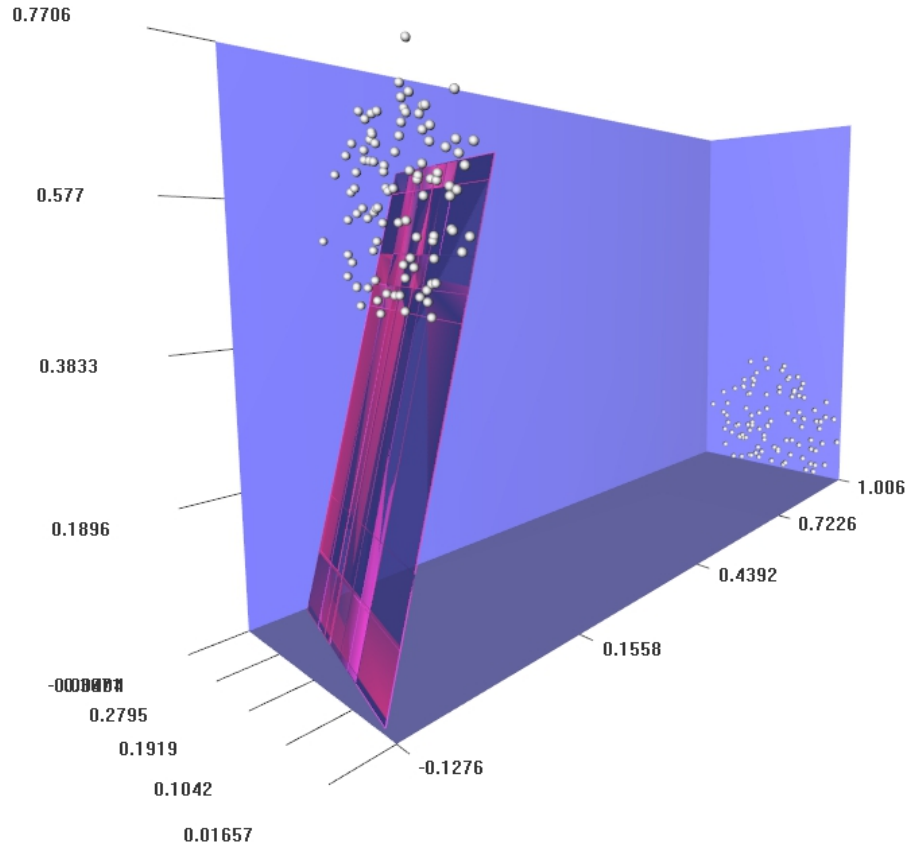
Artificial Neuron

Example of classification based on perceptron.



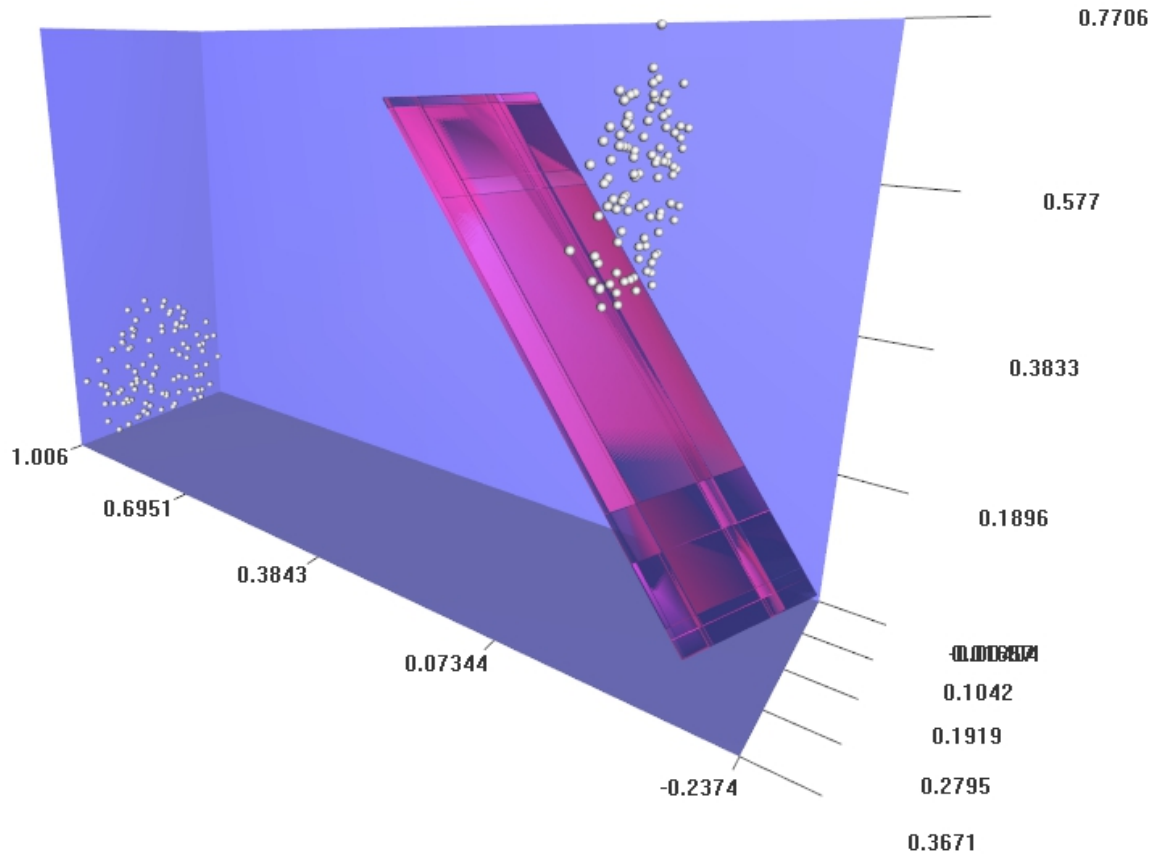
Artificial Neuron

Example of classification based on perceptron.



Artificial Neuron

Example of classification based on perceptron.



Artificial Neuron

- Although the perceptron rule finds a successful weight vector when the training ***n*** examples are linearly separable, it can fail to converge if the examples are not linearly separable.
- A second training rule, called the ***delta rule,*** is designed to overcome this difficulty.
- If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.

Artificial Neuron (The Delta Rule)

The key idea behind the delta rule is to use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

This rule is important because gradient descent provides the basis for the BACKPROPAGATION algorithm, which can learn networks with many interconnected neurons.

Artificial Neuron (The Delta Rule)

The key idea behind the delta rule is to use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

This rule is important because gradient descent provides the basis for the BACKPROPAGATION algorithm, which can learn networks with many interconnected neurons.

The delta training rule is best understood by considering the task of training an perceptron; that is, a linear unit for which the output o is given by:

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

Artificial Neuron (The Delta Rule)

The key idea behind the delta rule is to use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

We can derive a weight learning rule for linear units, specifying a measure for the ***training error*** of the weight vector, relative to the training examples.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

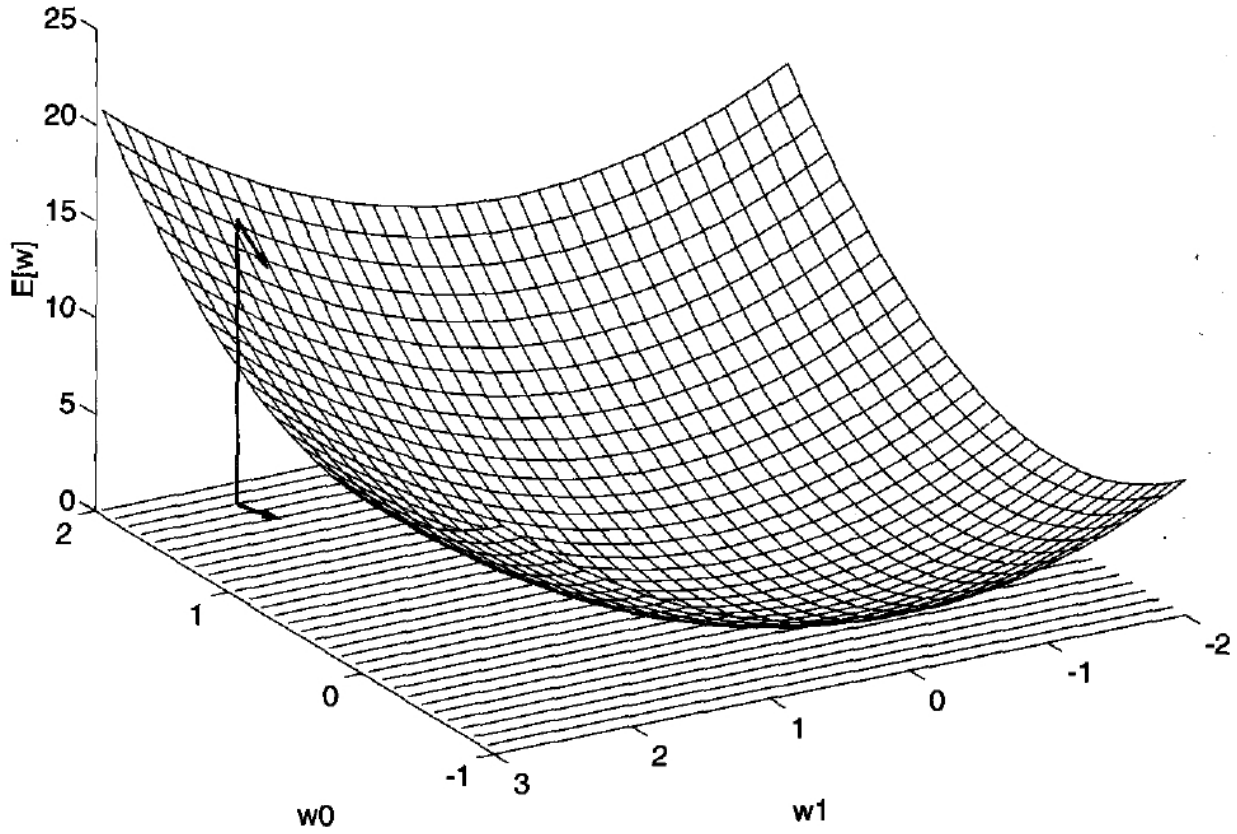
Where t_d and o_d are the ***target output*** and the ***output*** of the perceptron unit for a training example **d** in a dataset **D**.

Remember that:

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

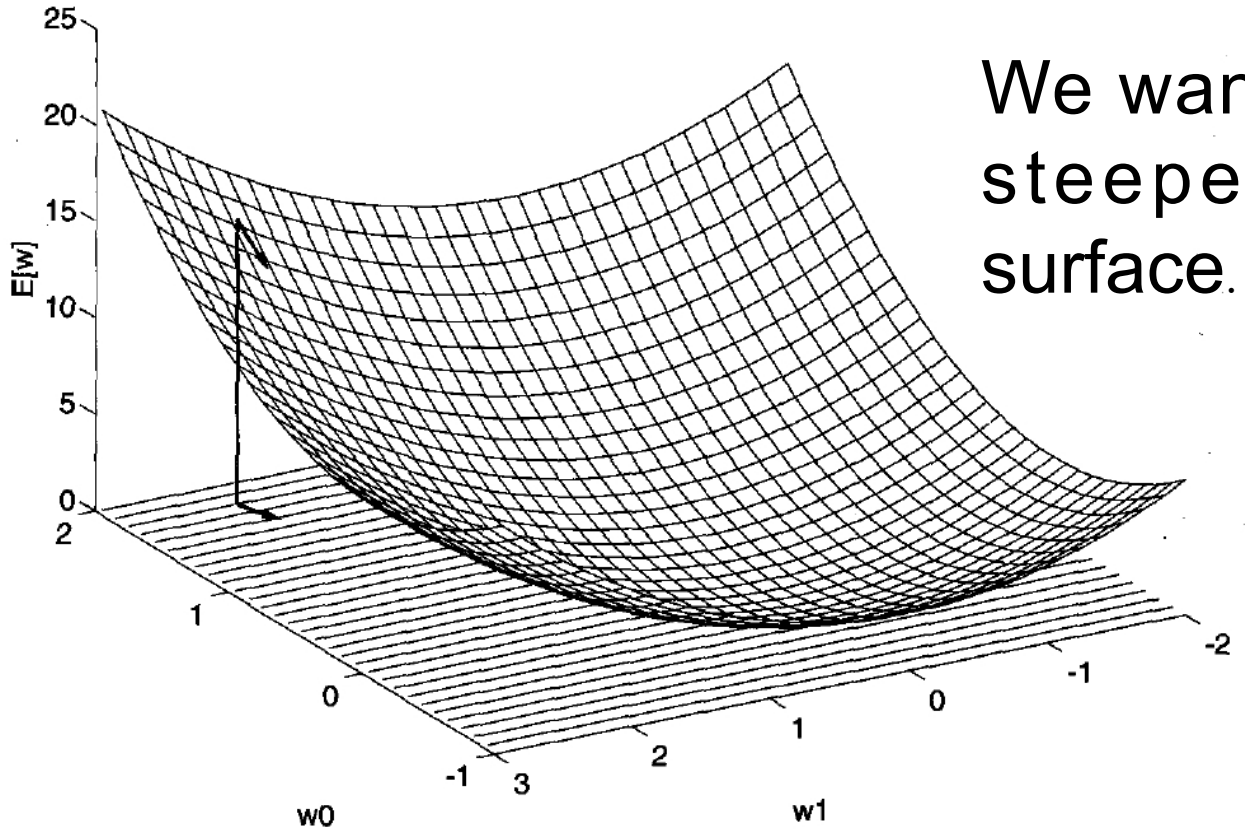
Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method



Artificial Neuron (The Delta Rule)

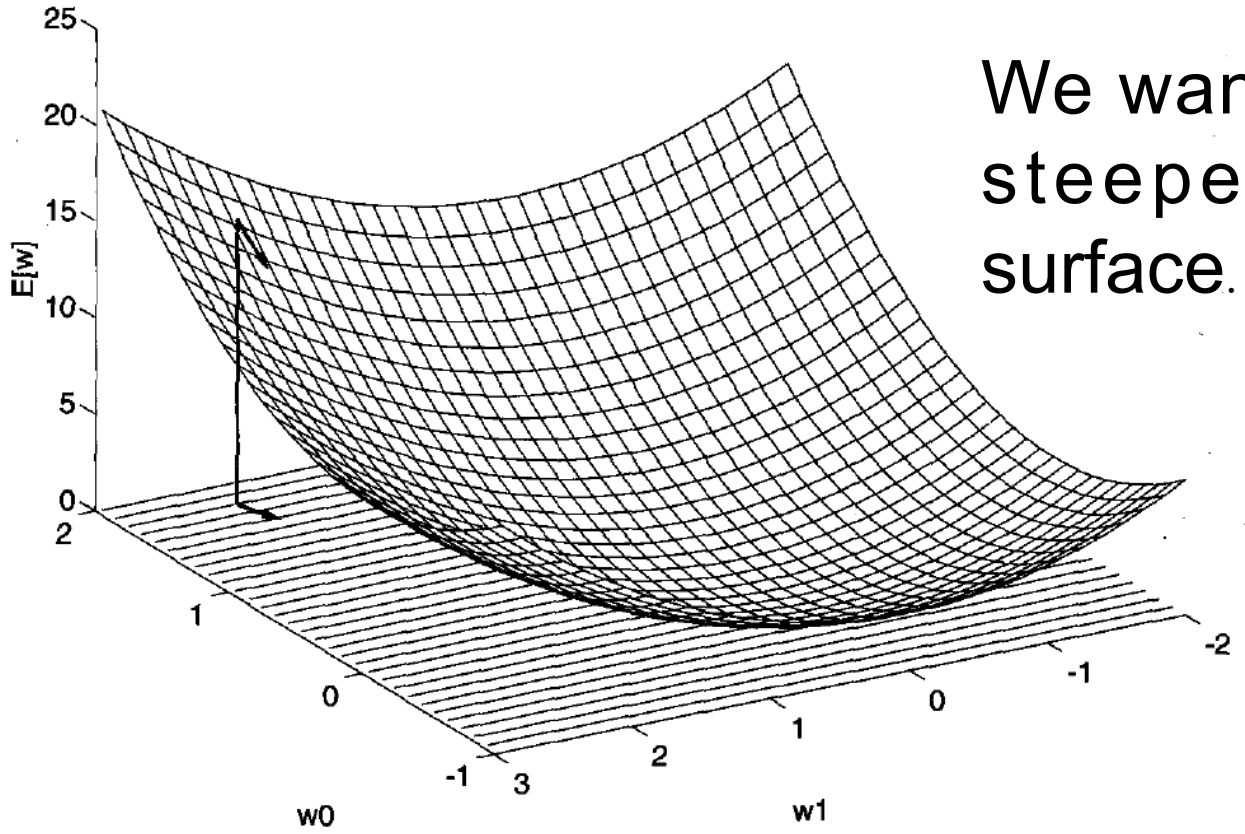
Visualizing the gradient descendent method



We want to calculate the direction of steepest descent along the error surface.

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method

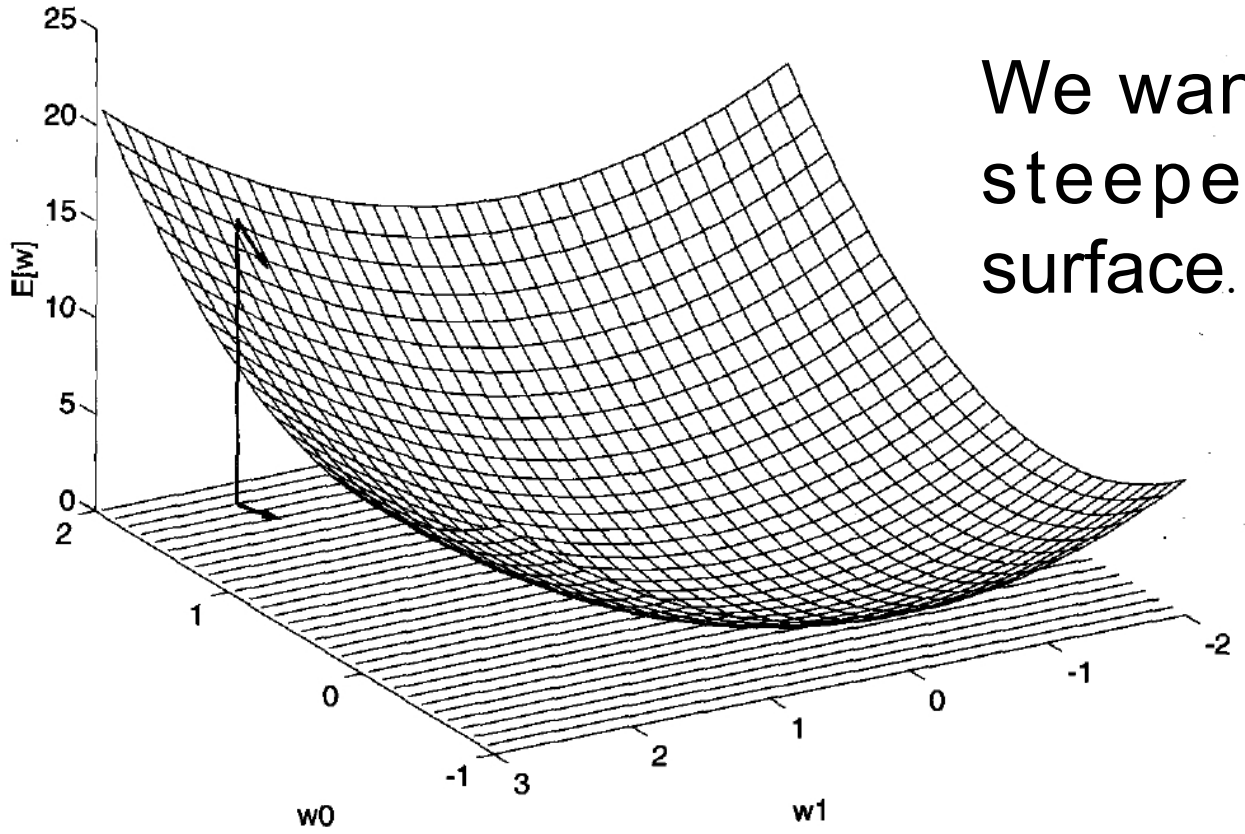


We want to calculate the direction of steepest descent along the error surface.

$$\nabla E(\vec{w})$$

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method

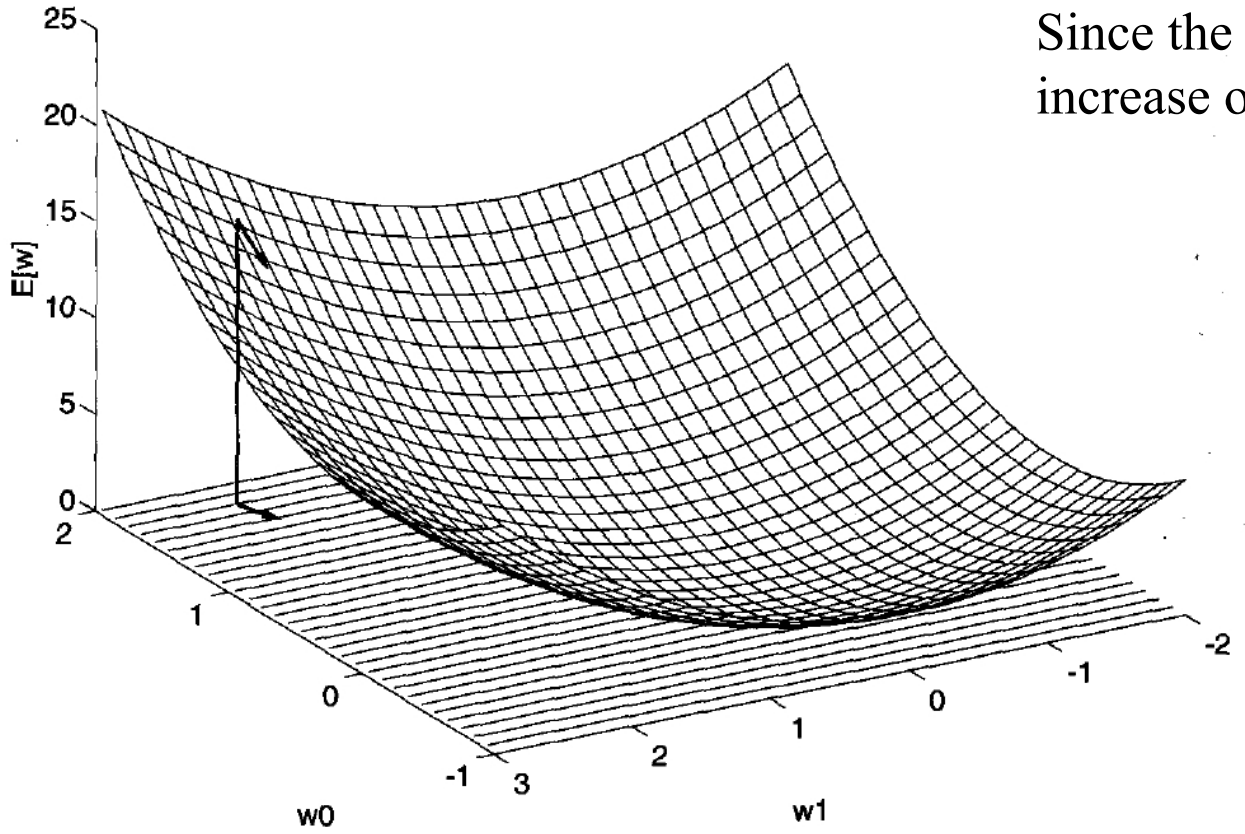


We want to calculate the direction of steepest descent along the error surface.

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n} \right]$$

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method

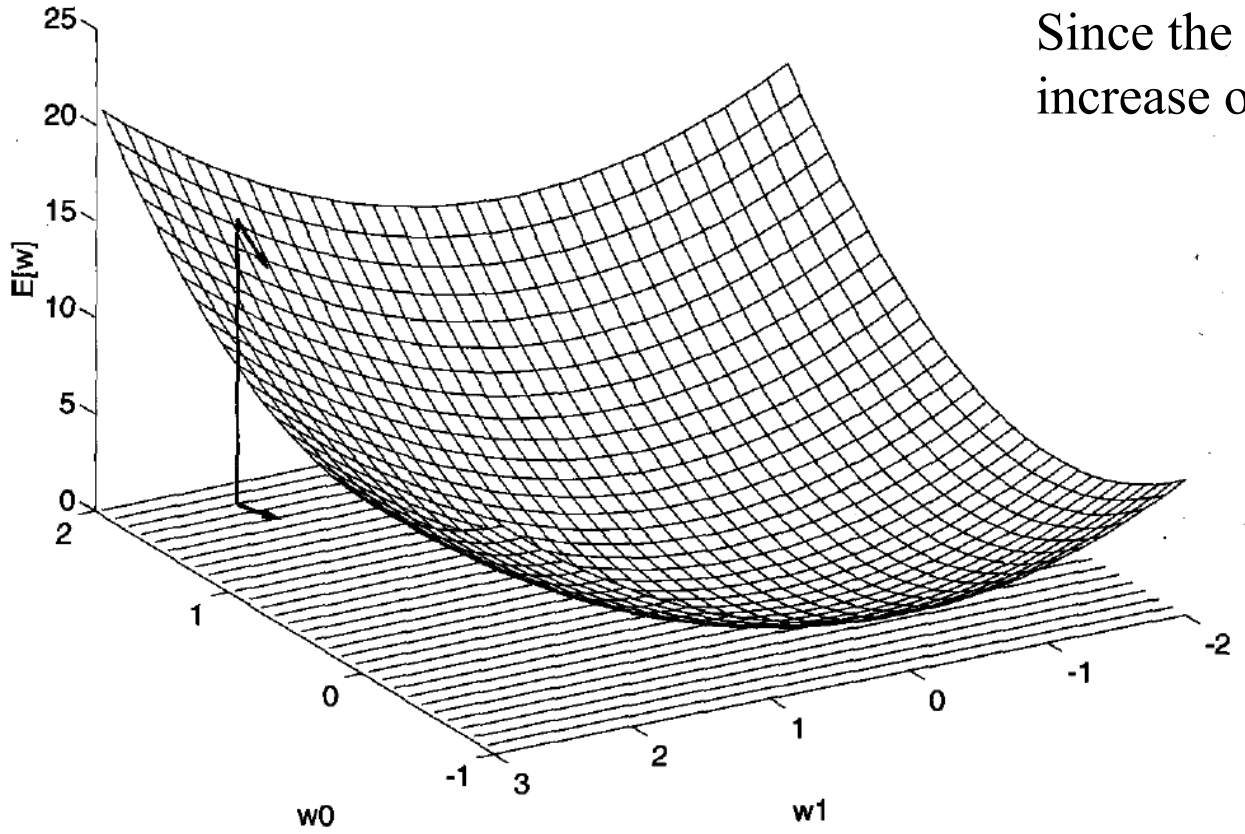


Since the gradient specifies the direction of steepest increase of ***E***, the training rule for gradient descent is:

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

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method



Since the gradient specifies the direction of steepest increase of ***E***, the training rule for gradient descent is:

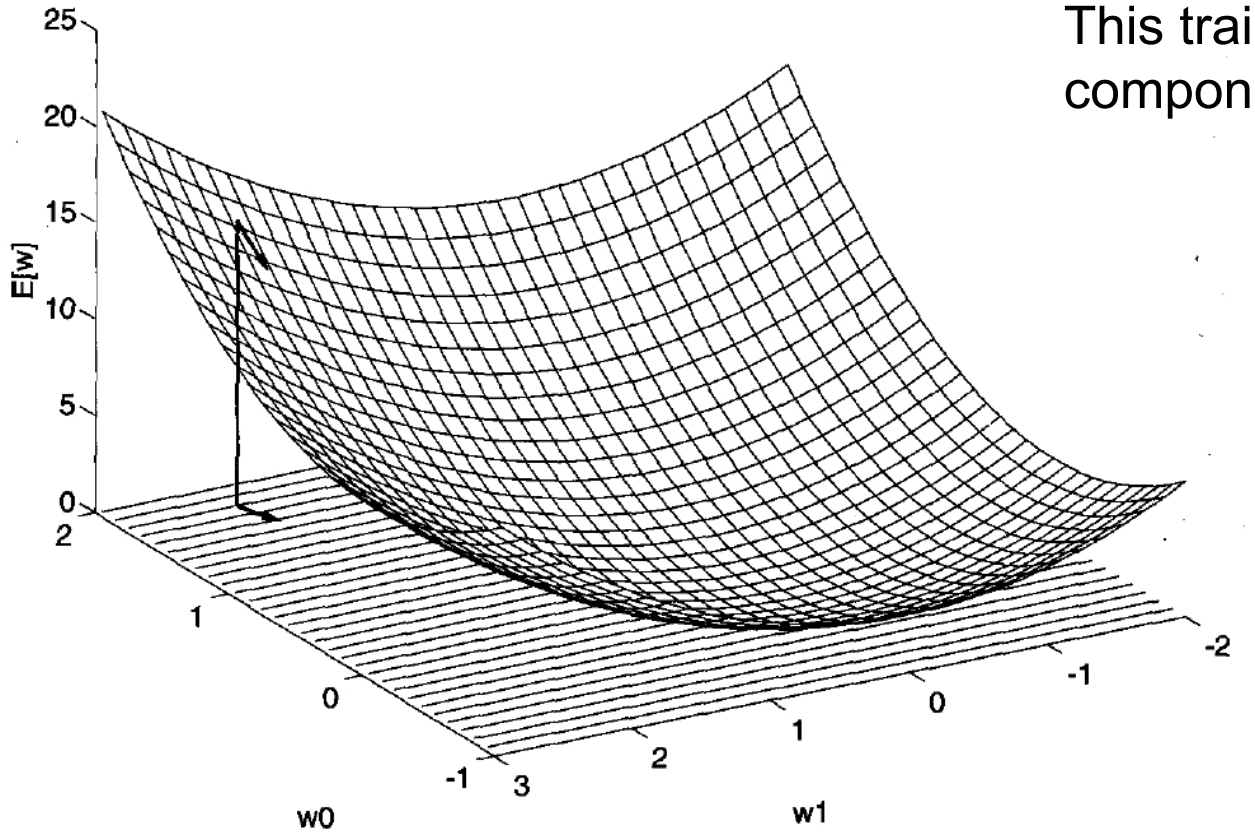
$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

where

$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method



This training rule can also be written in its component form:

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

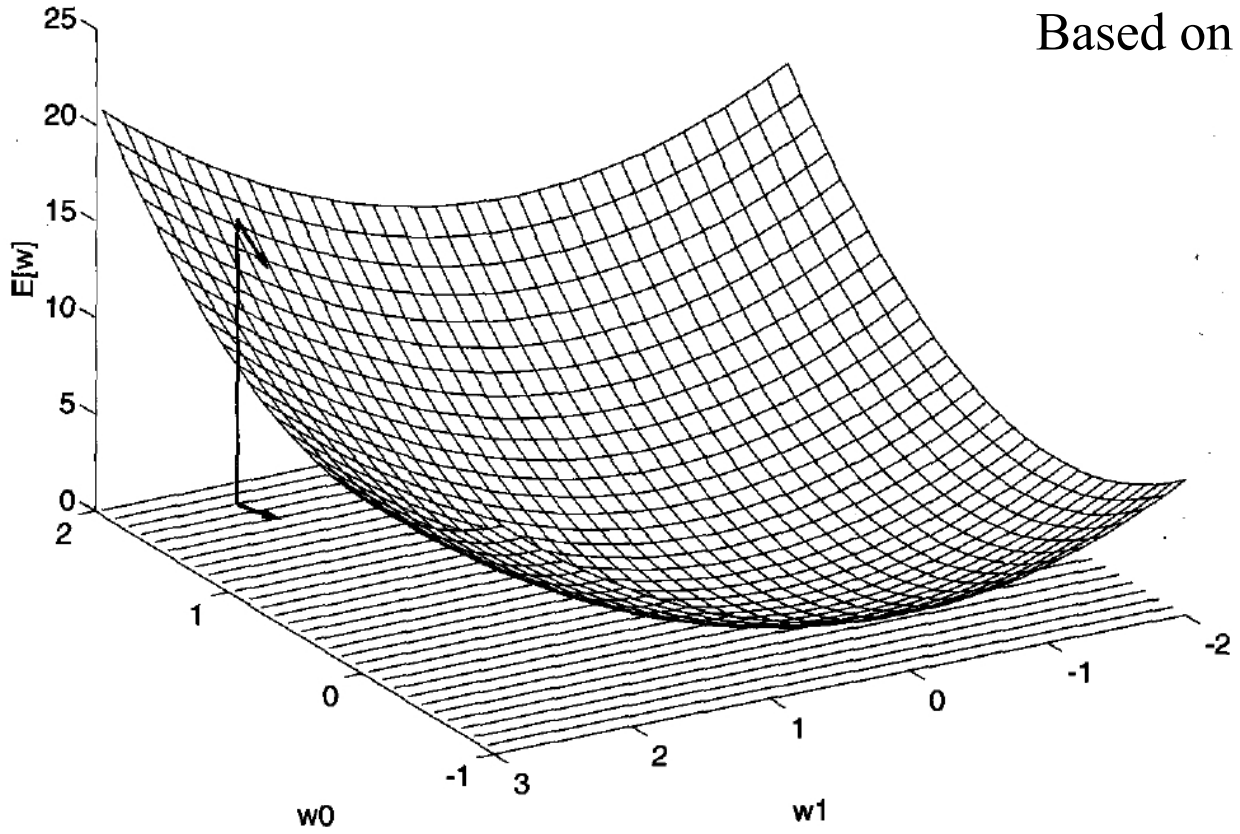
where

$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$

What it means the negative sign?

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method



Based on the above the component representation is:

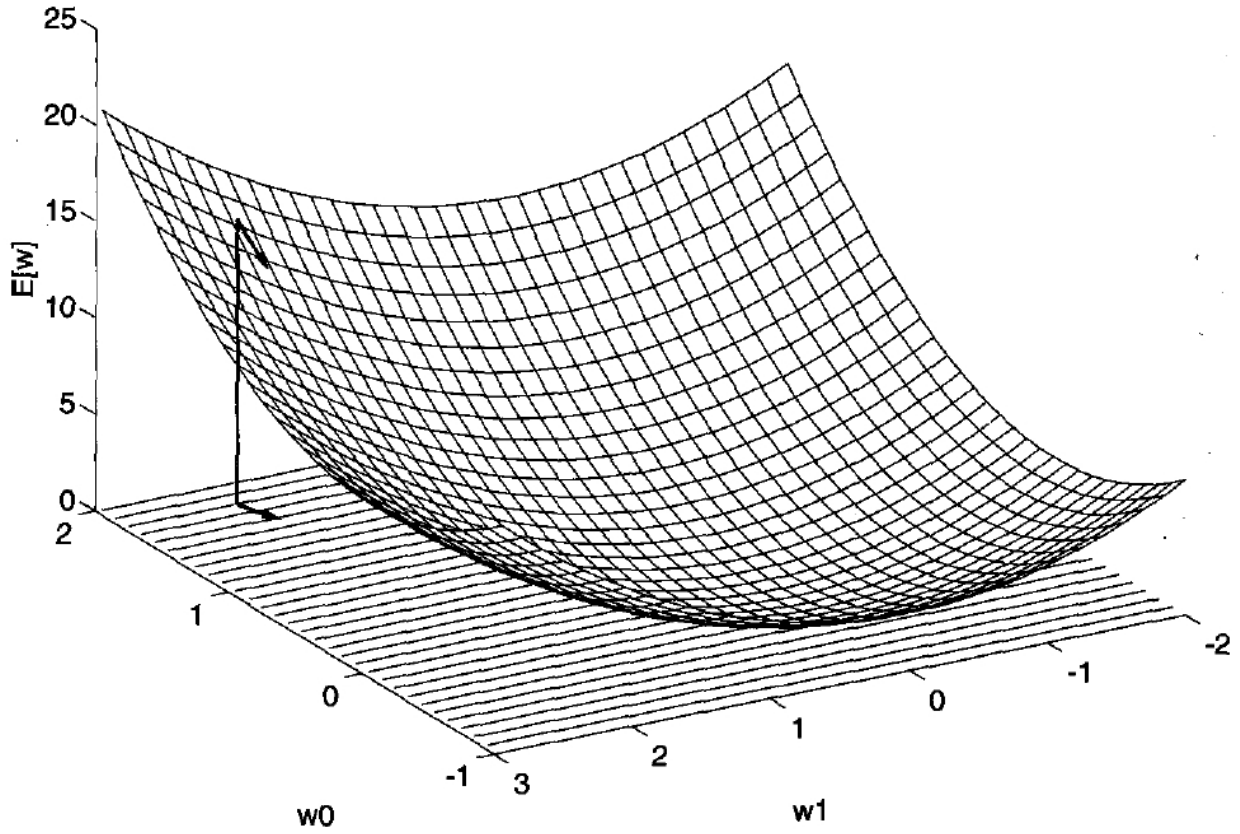
$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method

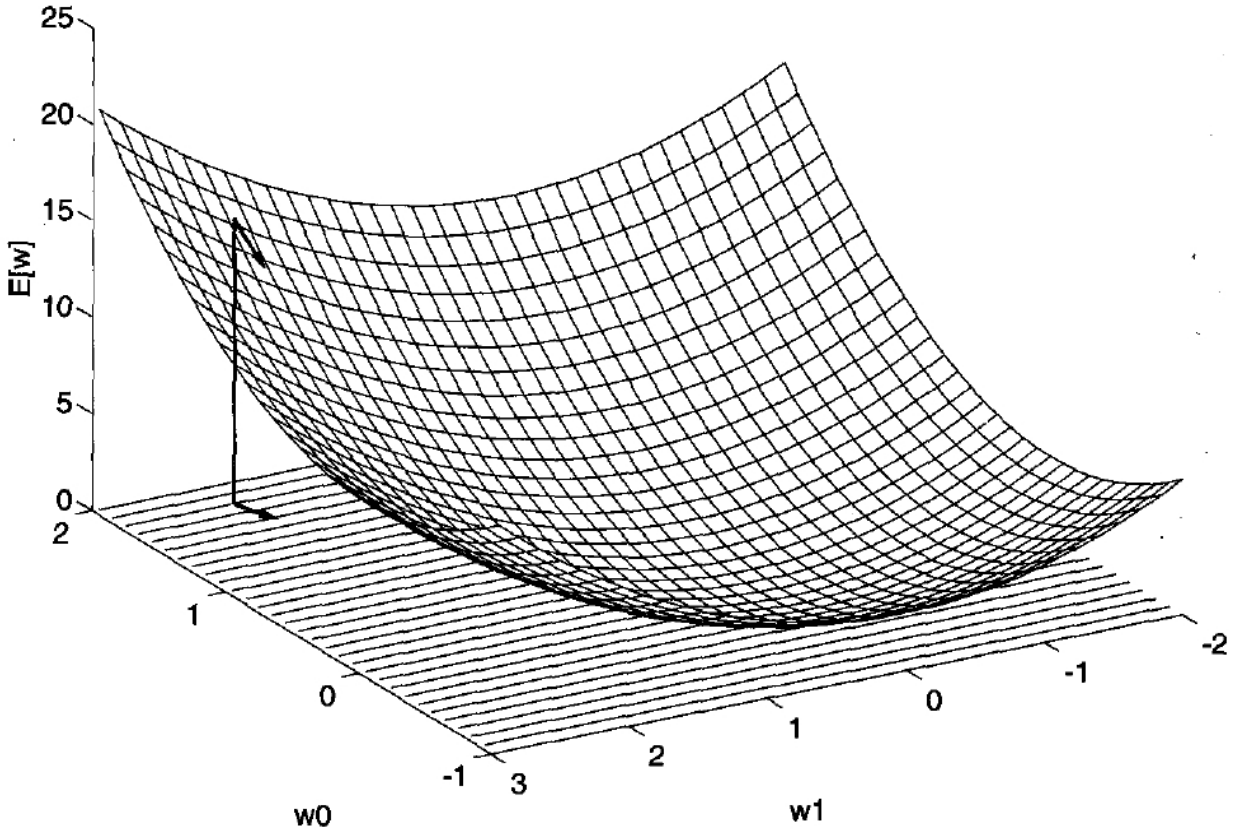


$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method



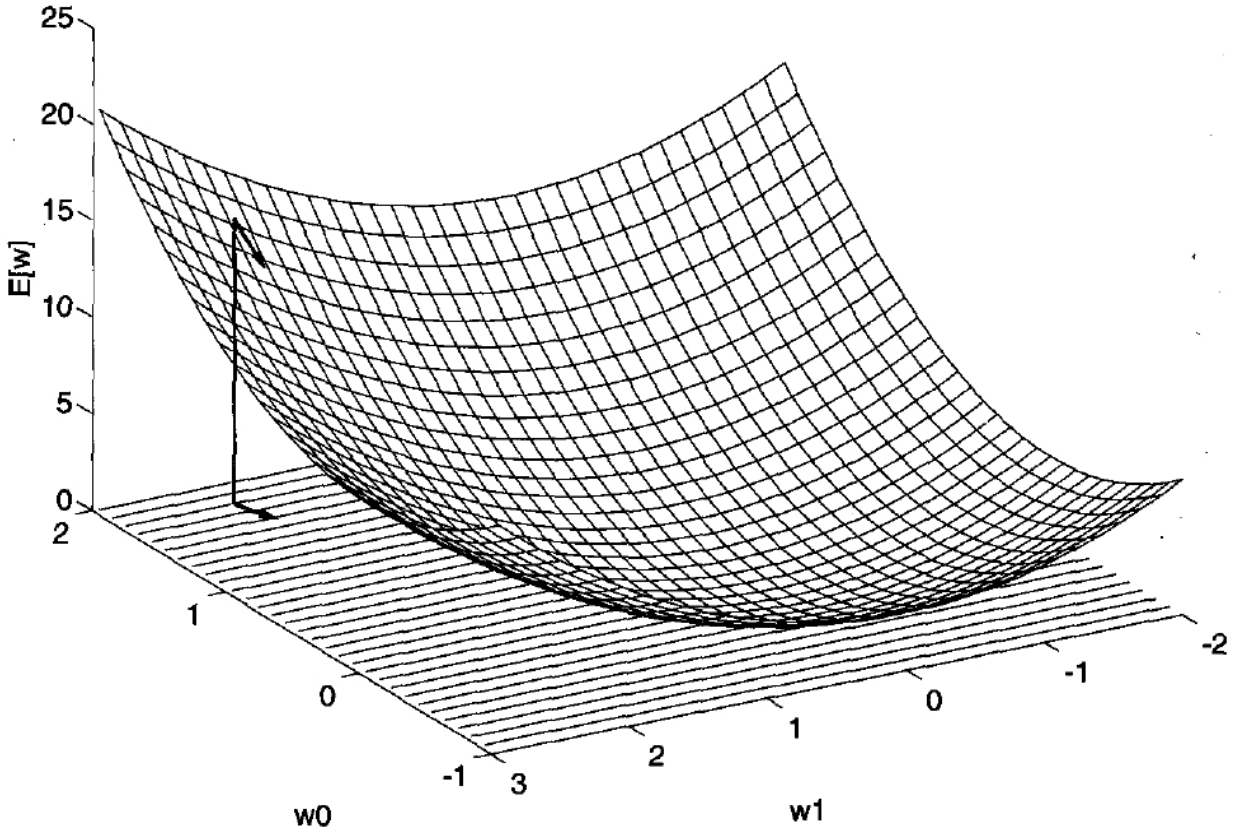
$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method



$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

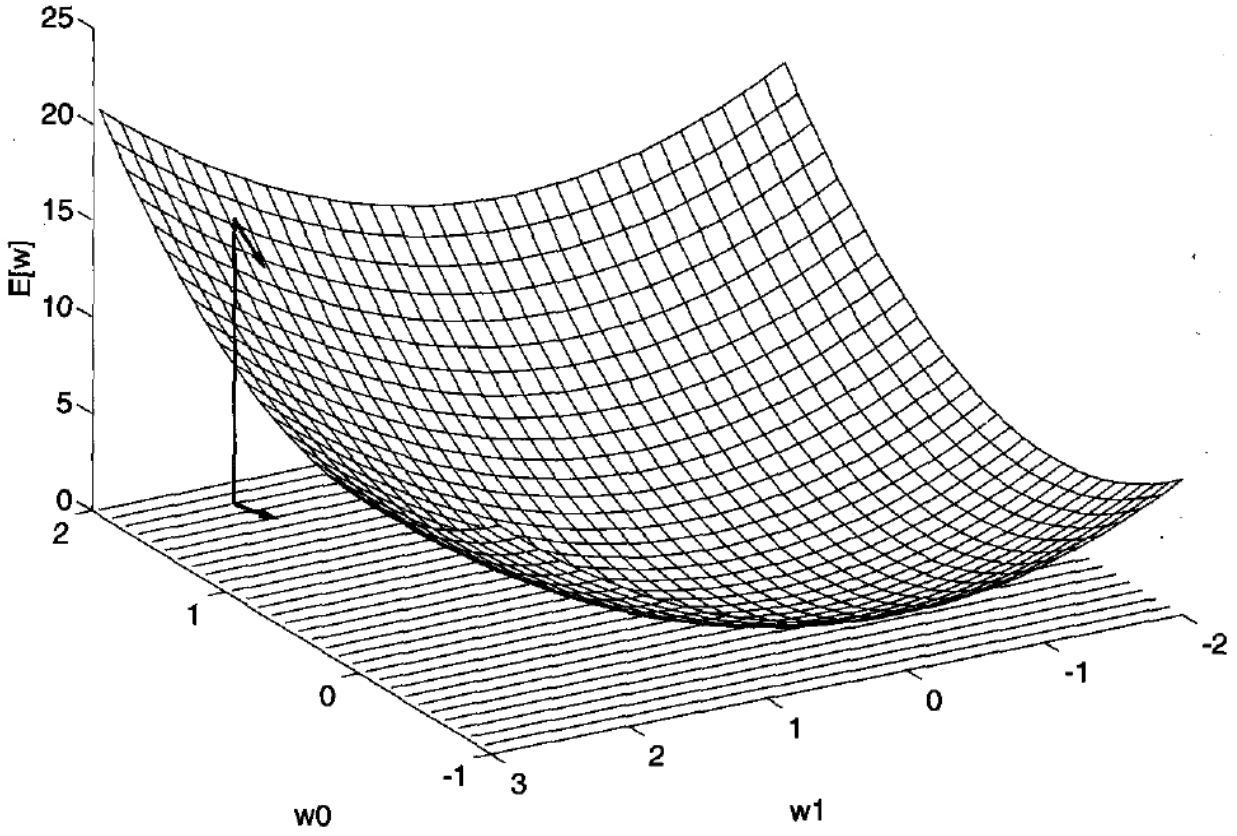
$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method



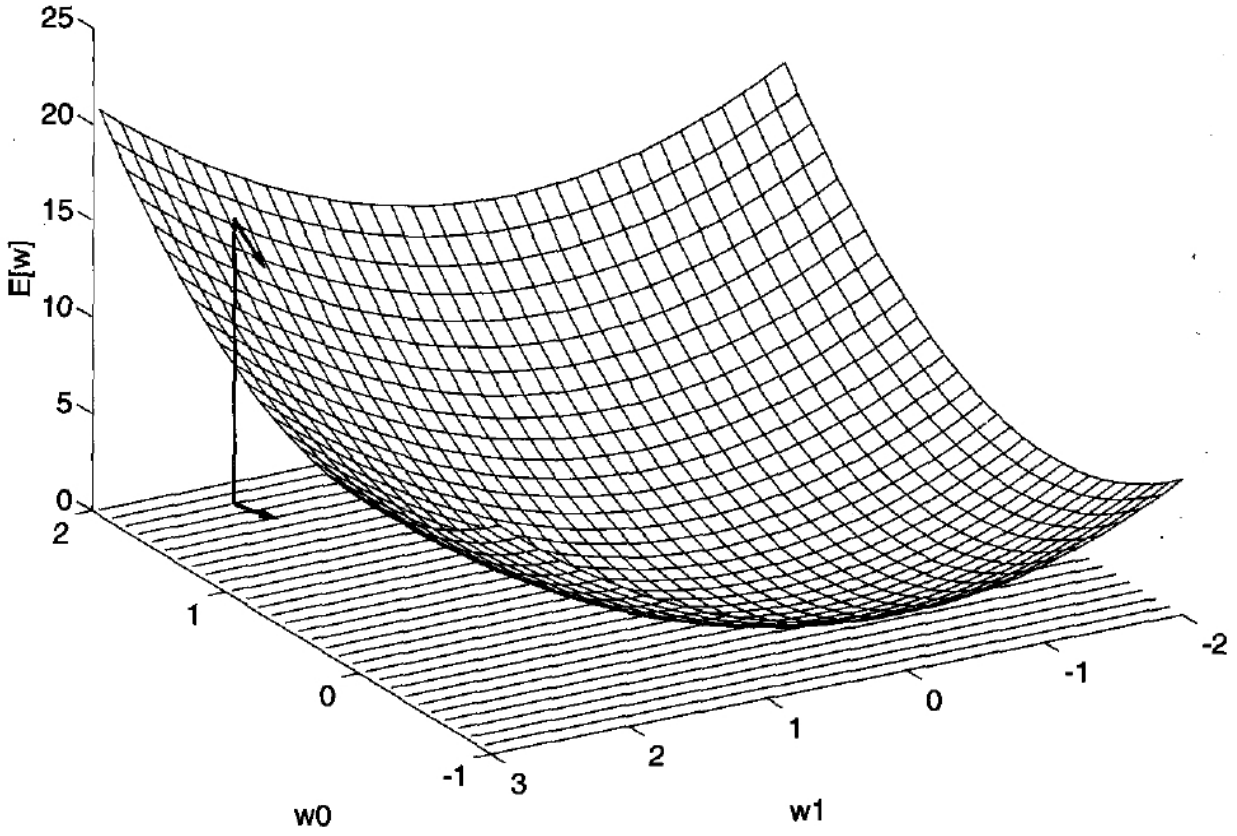
$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method



$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

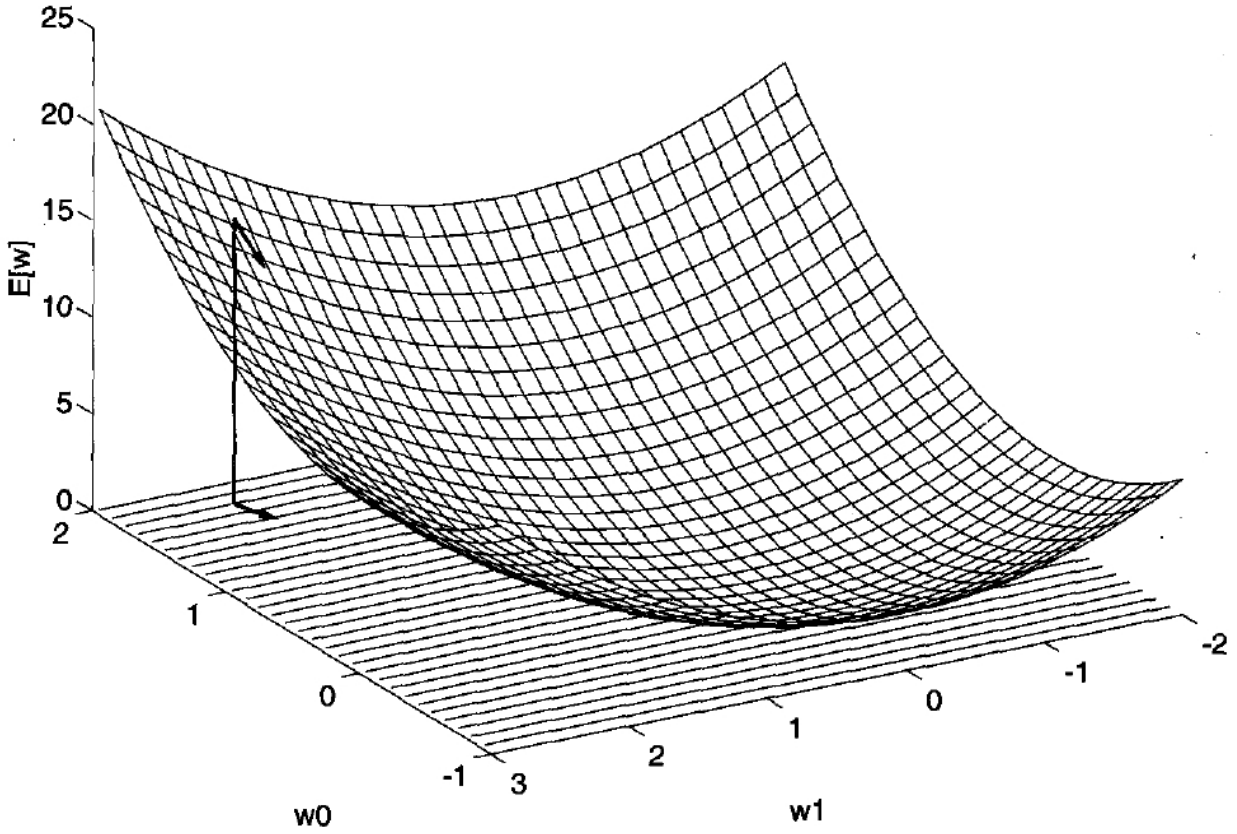
$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d})$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

Artificial Neuron (The Delta Rule)

Visualizing the gradient descendent method



Thus:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$$

Exercises

Exercise 1

Review the script in which a perceptron implementation is shown.

Execute the script changing etha parameter.

Change the input dataset including a non-linear separable problem.

Exercise 2

Review the script using a perceptron implementation based on sklearn API

Cinvestav
Tamaulipas

Multilayer Neural Networks

Multilayer networks

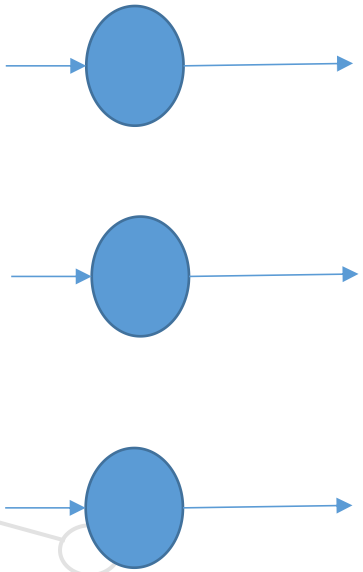
- Assume a problem that requires to learn about a set of patterns classified into **three classes**. How to solve this learning problem through artificial neurons?

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2

Multilayer networks

- Assume a problem that requires to learn about a set of patterns classified into **three classes**. How to solve this learning problem through artificial neurons?

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2

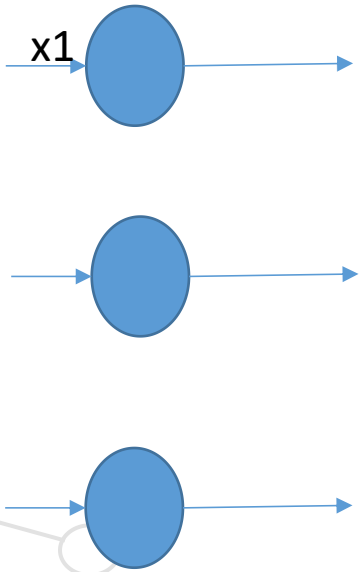


For each feature, a neuron is defined.

Multilayer networks

- Assume a problem that requires to learn about a set of patterns classified into **three classes**. How to solve this learning problem through artificial neurons?

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2

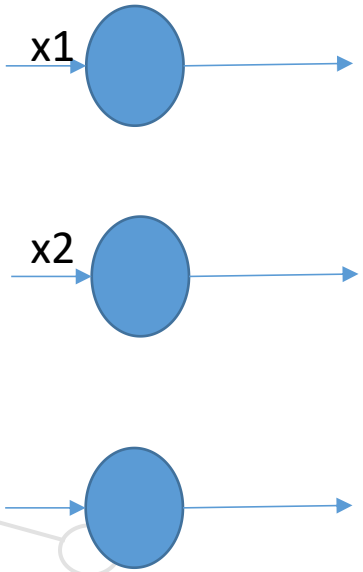


For each feature, a neuron is defined.

Multilayer networks

- Assume a problem that requires to learn about a set of patterns classified into **three classes**. How to solve this learning problem through artificial neurons?

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2

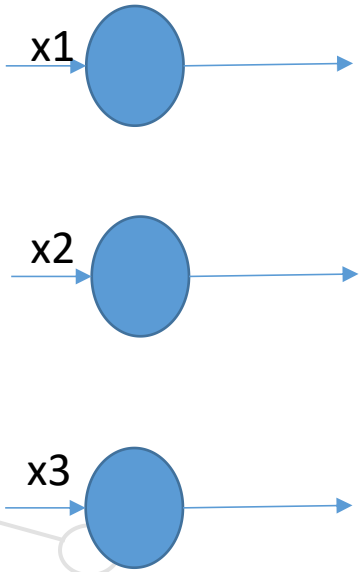


For each feature, a neuron is defined.

Multilayer networks

- Assume a problem that requires to learn about a set of patterns classified into **three classes**. How to solve this learning problem through artificial neurons?

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



For each feature, a neuron is defined.

Multilayer networks

- Assume a problem that requires to learn about a set of patterns classified into **three classes**. How to solve this learning problem through artificial neurons?

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2

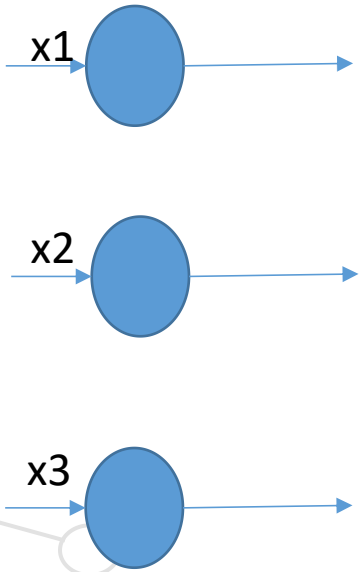
A set of neurons that encodes the class label of the input pattern is created.

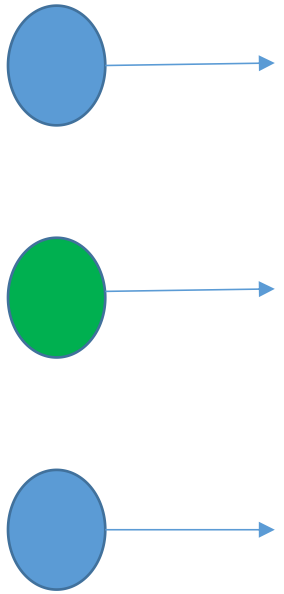
For each feature, a neuron is defined.

Multilayer networks

- Assume a problem that requires to learn about a set of patterns classified into **three classes**. How to solve this learning problem through artificial neurons?

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2





A set of neurons that represent the class label of the input pattern is created.

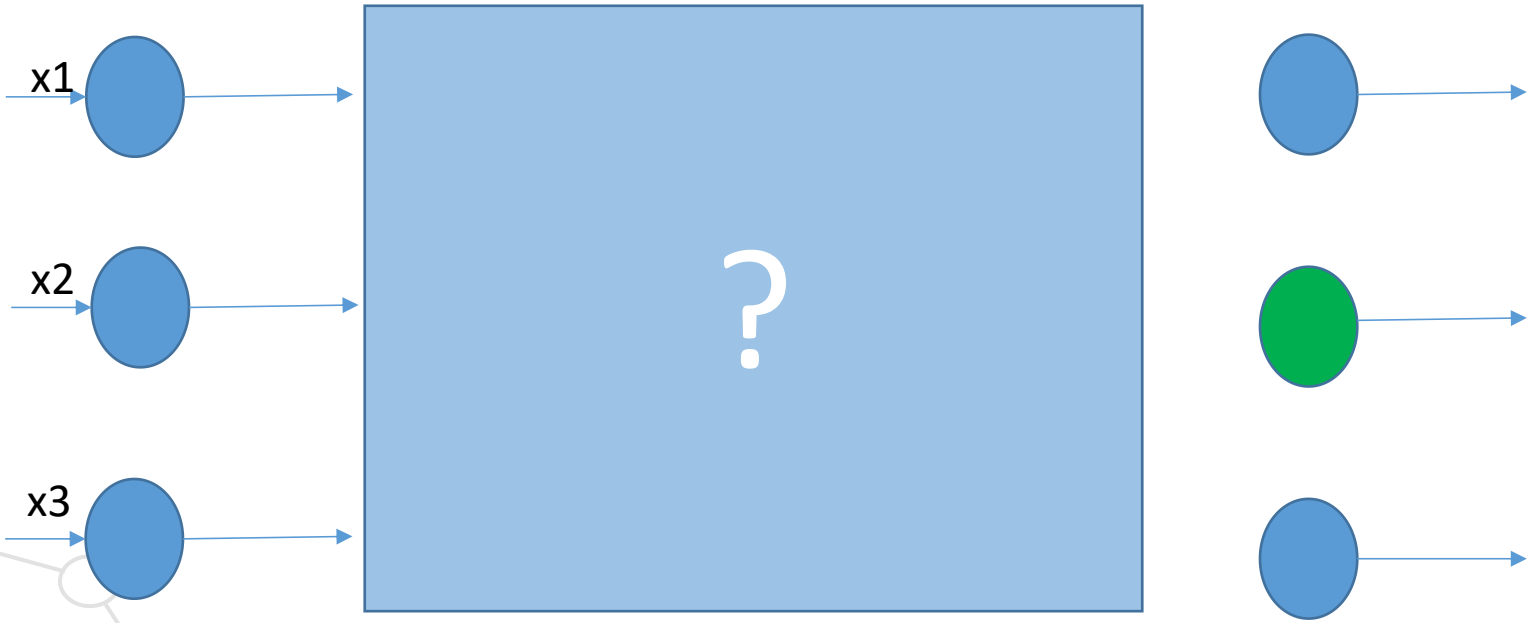
Note that only one neuron will be activated

For each feature, a neuron is defined.

Multilayer networks

- The mystery box represent a set of neurons and interconnections that propagate the signals from input neurons to output neurons.

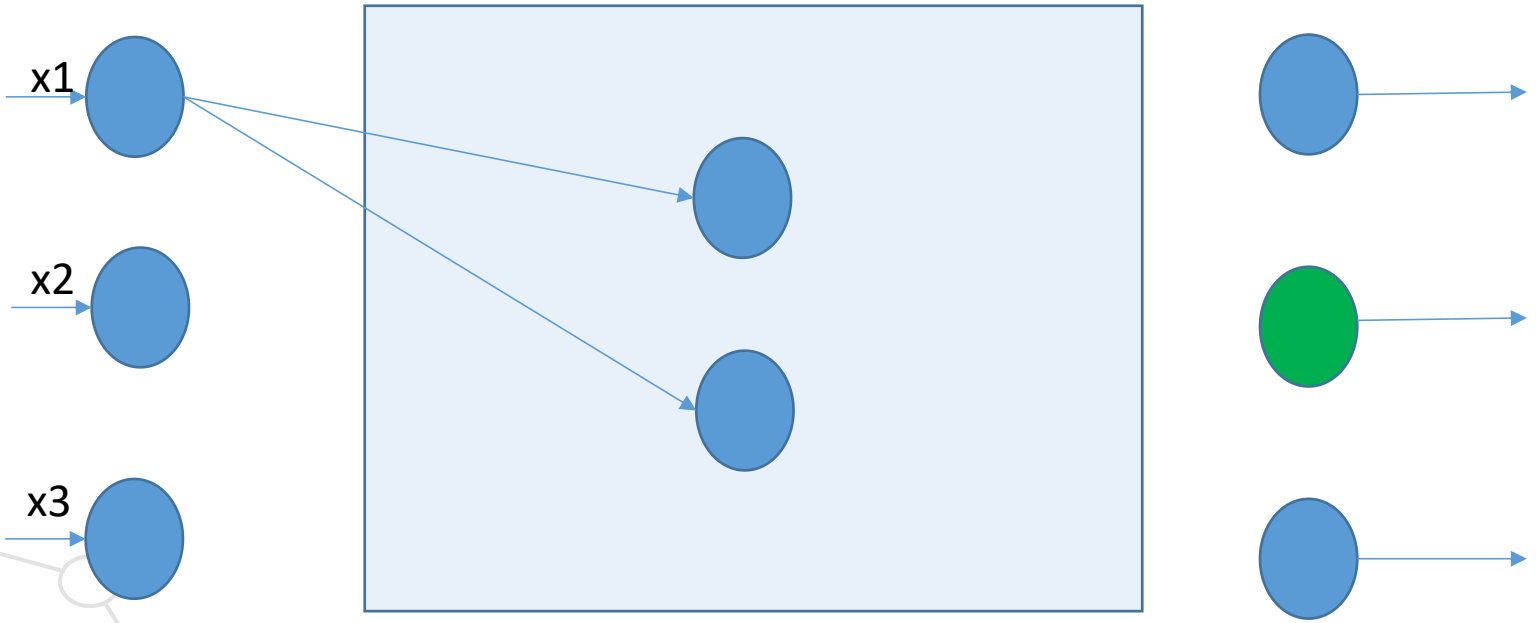
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks

- The mystery box represent a set of neurons and interconnections that propagate the signals from input neurons to output neurons.

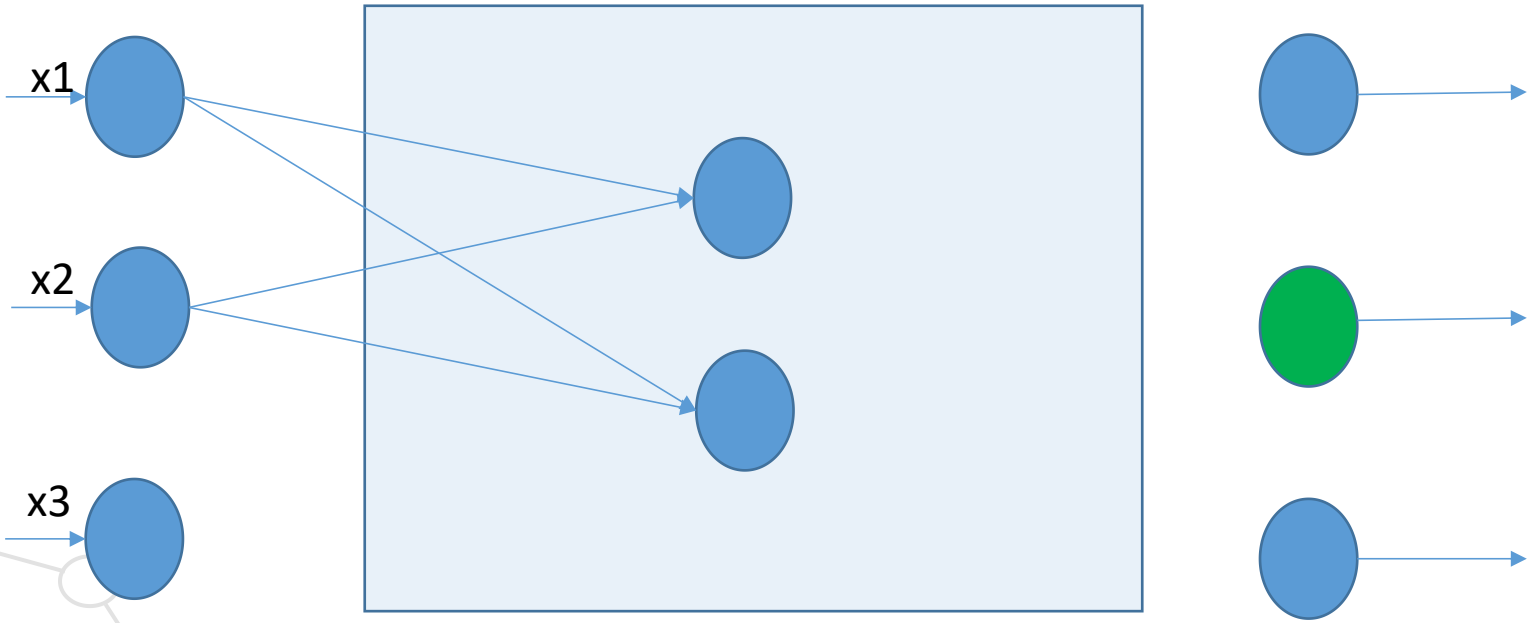
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks

- The mystery box represent a set of neurons and interconnections that propagate the signals from input neurons to output neurons.

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



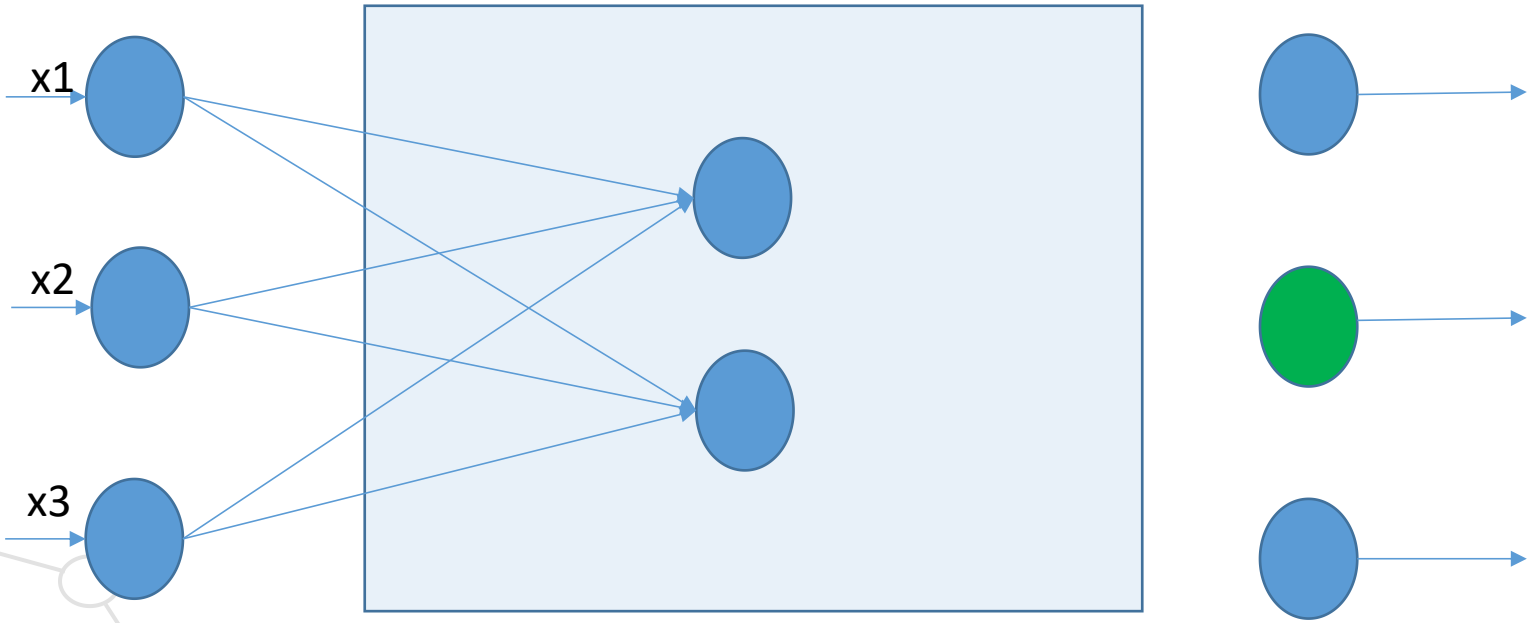
Input Neurons

Output Neurons

Multilayer networks

- The mystery box represent a set of neurons and interconnections that propagate the signals from input neurons to output neurons.

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



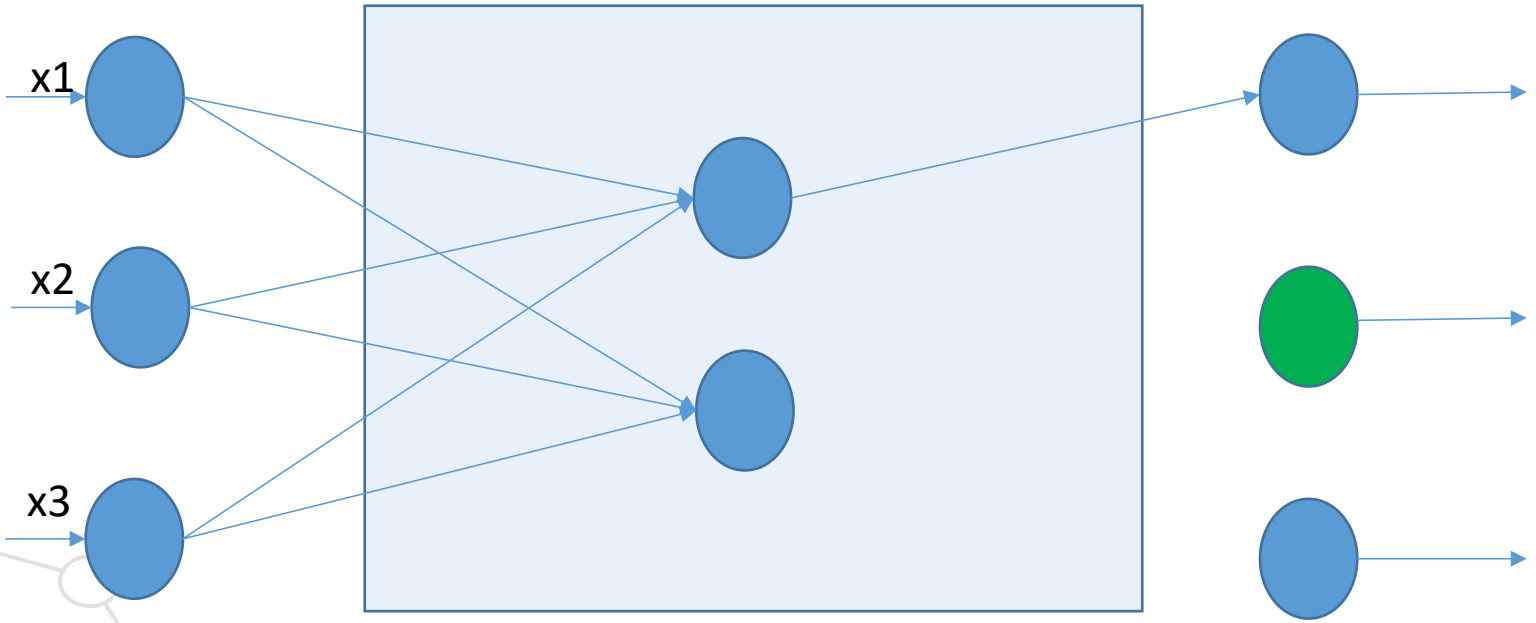
Input Neurons

Output Neurons

Multilayer networks

- The mystery box represent a set of neurons and interconnections that propagate the signals from input neurons to output neurons.

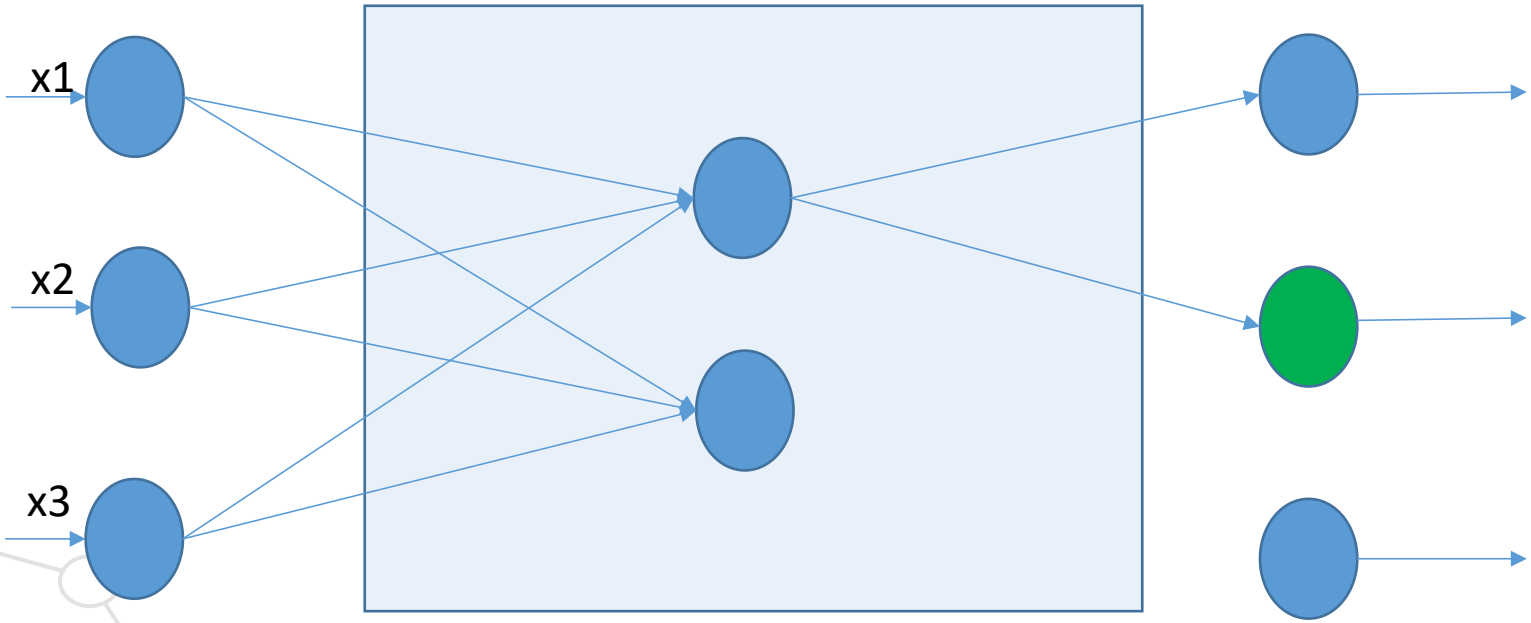
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks

- The mystery box represent a set of neurons and interconnections that propagate the signals from input neurons to output neurons.

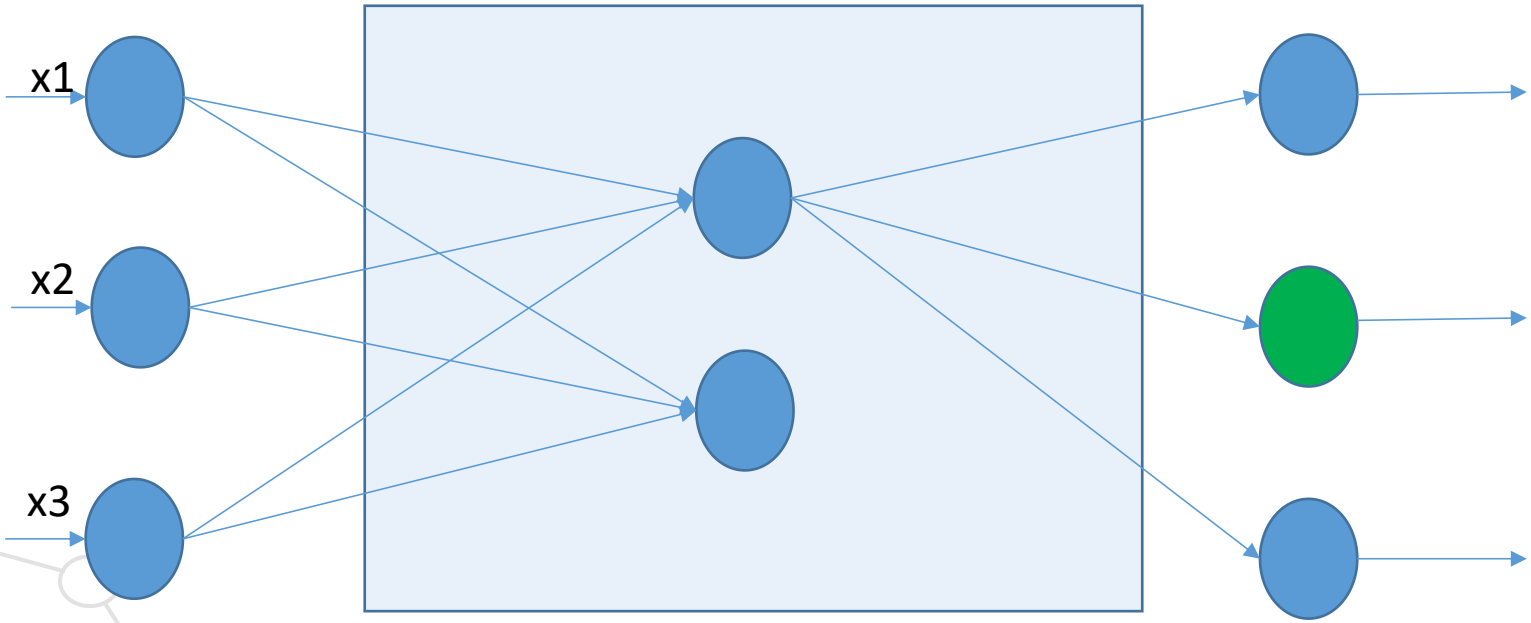
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks

- The mystery box represent a set of neurons and interconnections that propagate the signals from input neurons to output neurons.

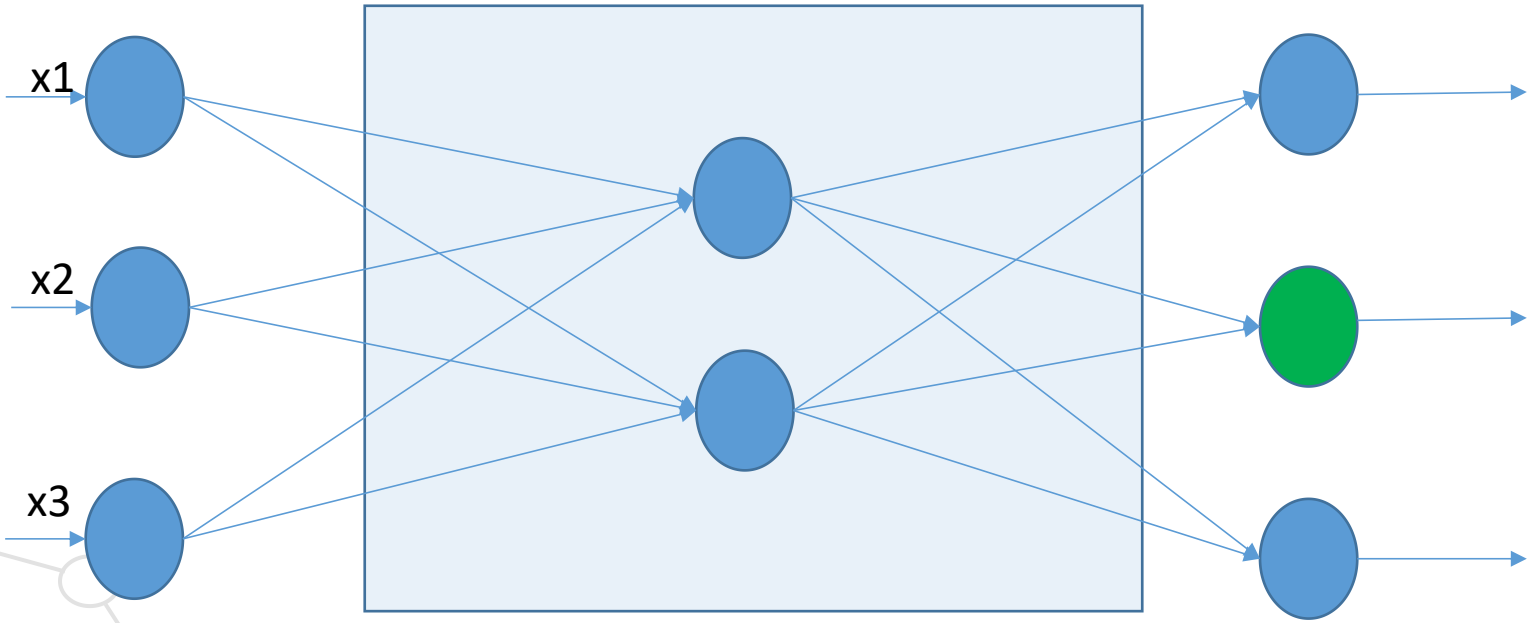
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks

- The mystery box represent a set of neurons and interconnections that propagate the signals from input neurons to output neurons.

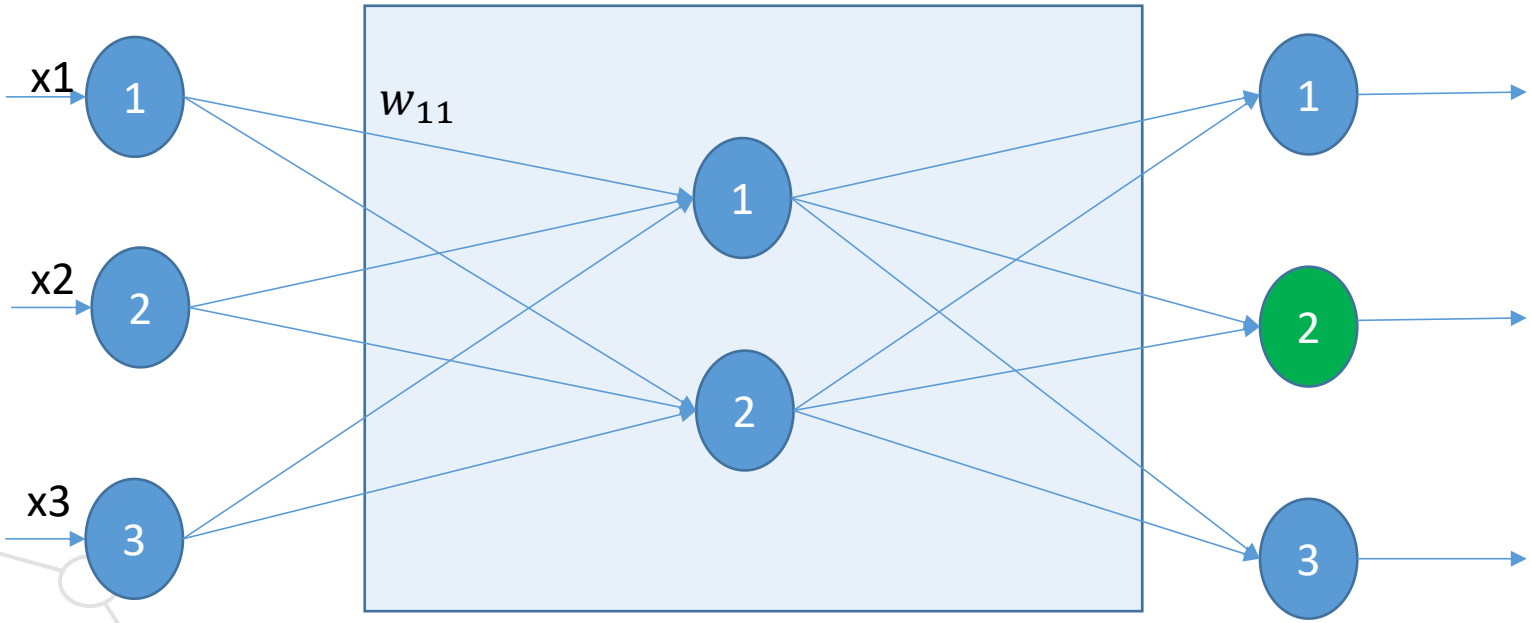
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks

- A set of weights must be defined.

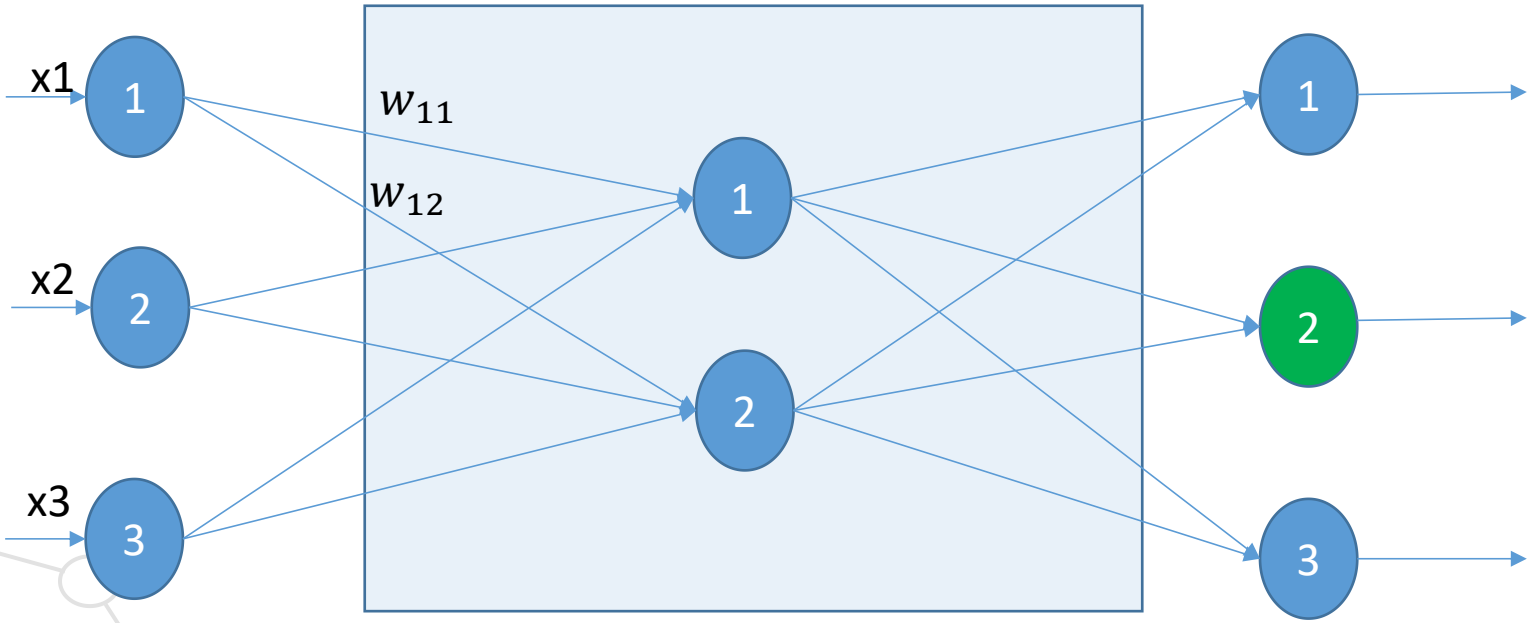
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks

- A set of weights must be defined.

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



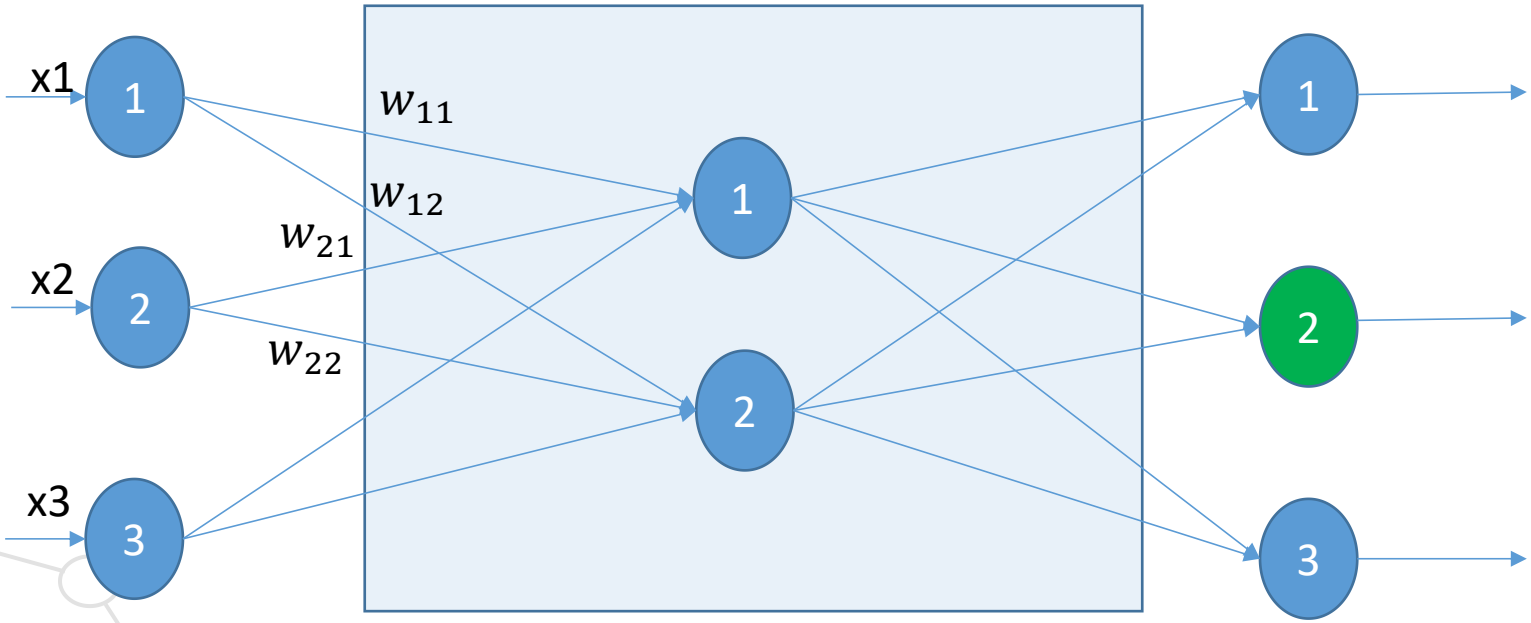
Input Neurons

Output Neurons

Multilayer networks

- A set of weights must be defined.

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



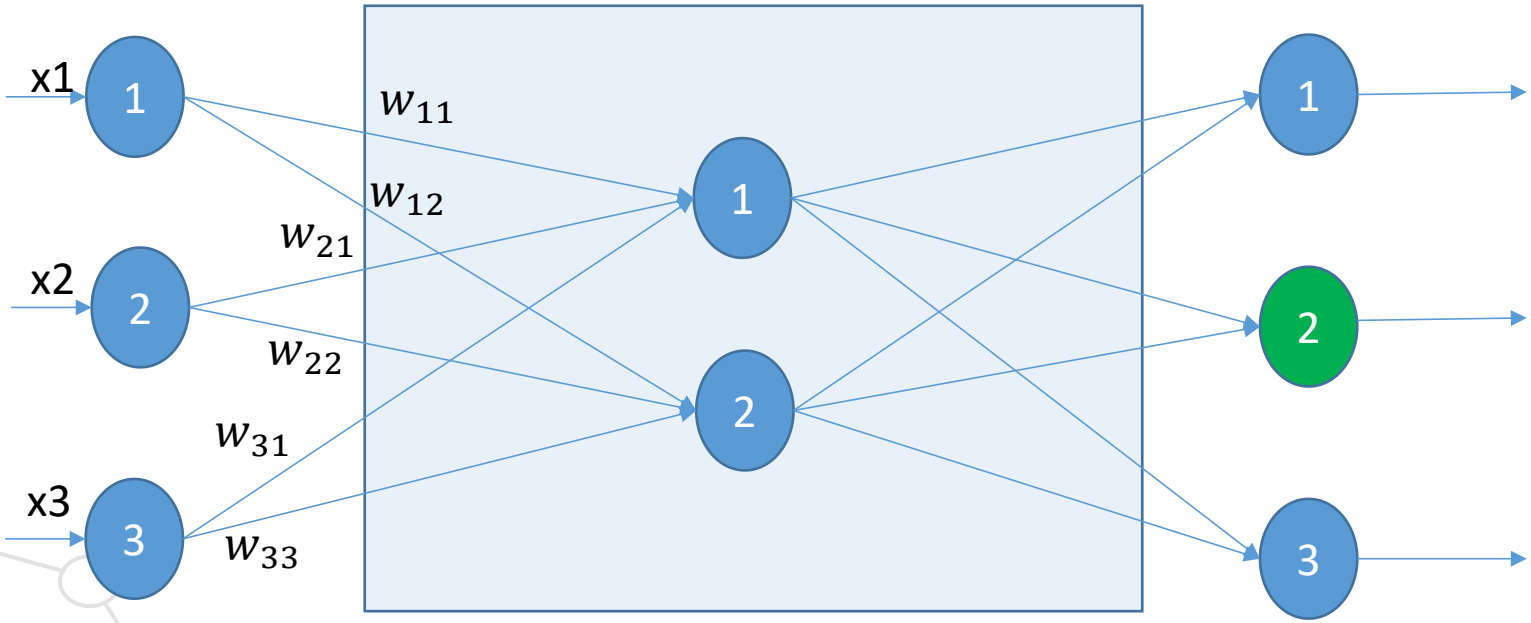
Input Neurons

Output Neurons

Multilayer networks

- A set of weights must be defined.

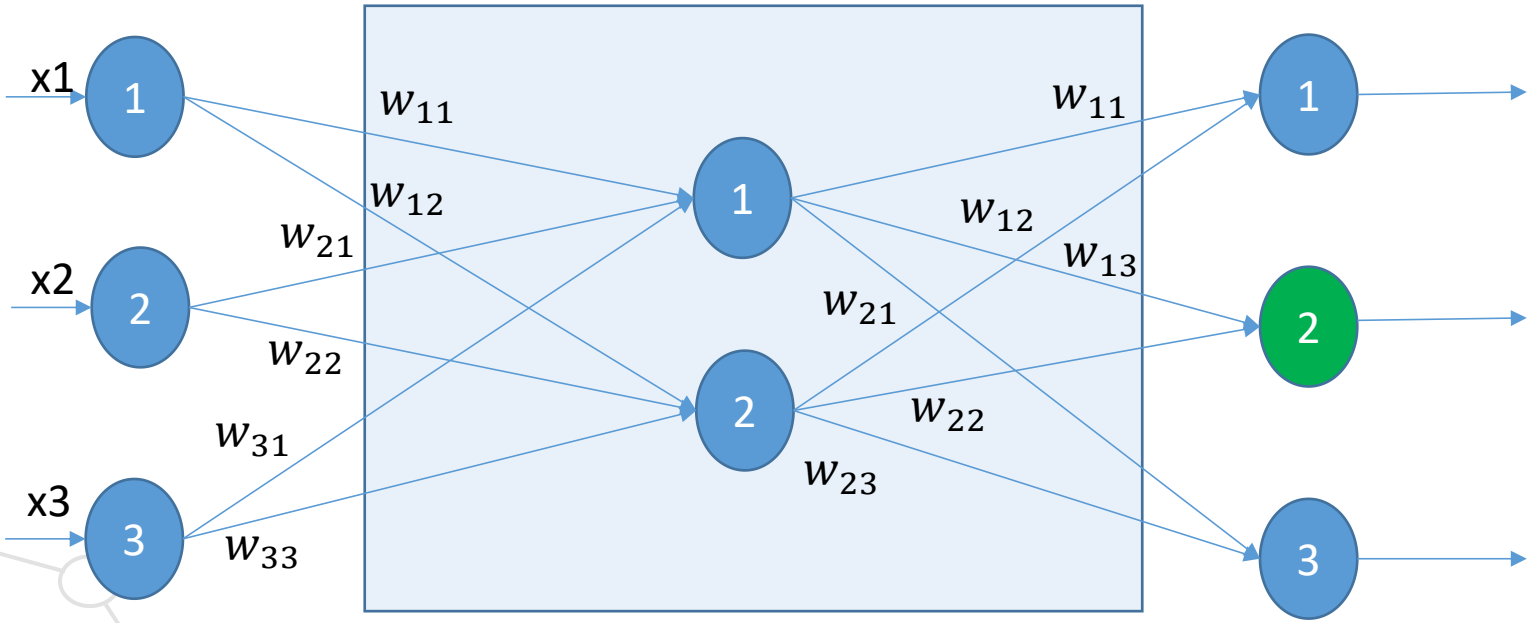
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks

- A set of weights must be defined.

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



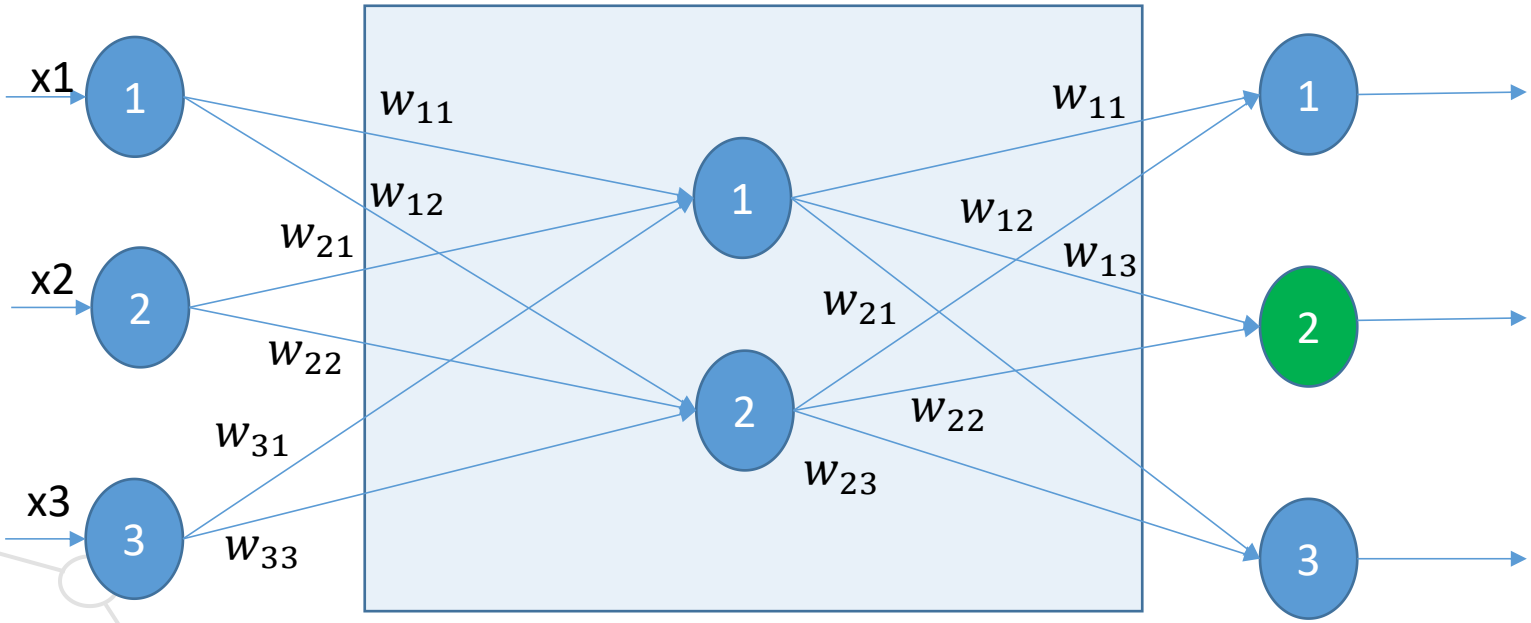
Input Neurons

Output Neurons

Multilayer networks

- A set of weights must be defined.
- Like perceptron, for the output of each neuron, an **activation function** is applied.

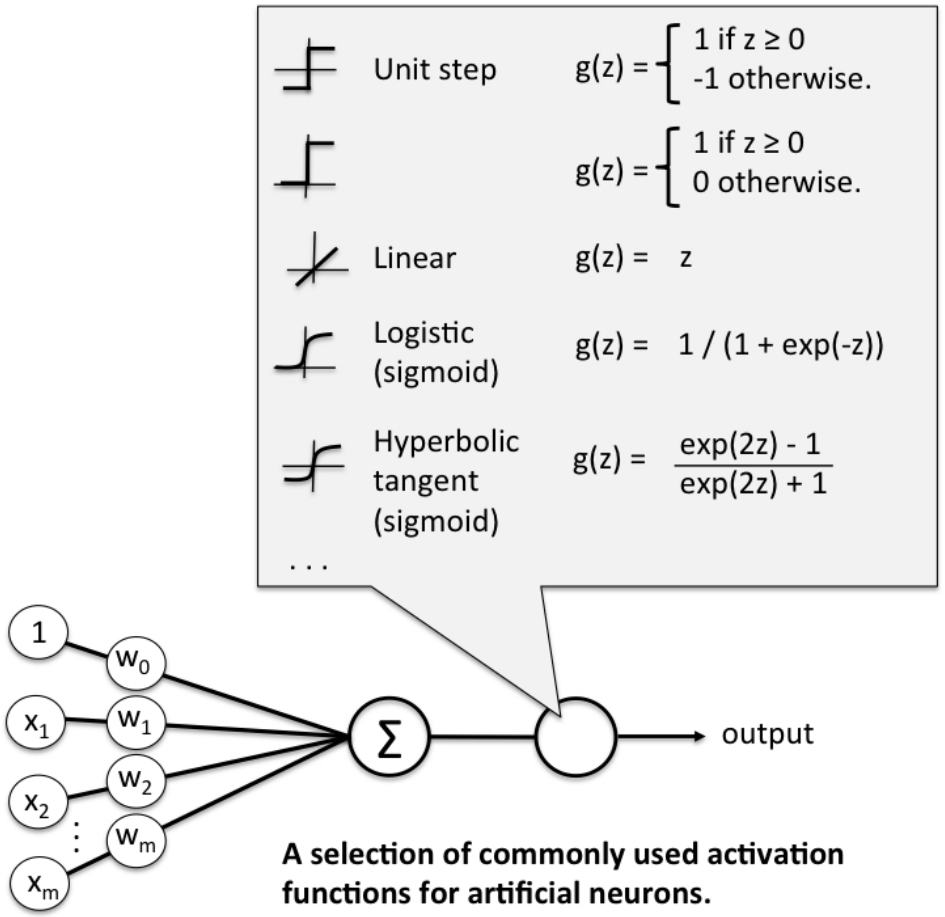
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Input Neurons

Output Neurons

Activation Functions



A selection of commonly used activation functions for artificial neurons.

Typically these functions are within **Hidden layers of a Neural Network Model**.

For output layers we should use a **Softmax function** for a classification problem to compute the probabilites for the classes.

For a regression problem it should simply use a **linear function**.

Softmax Function

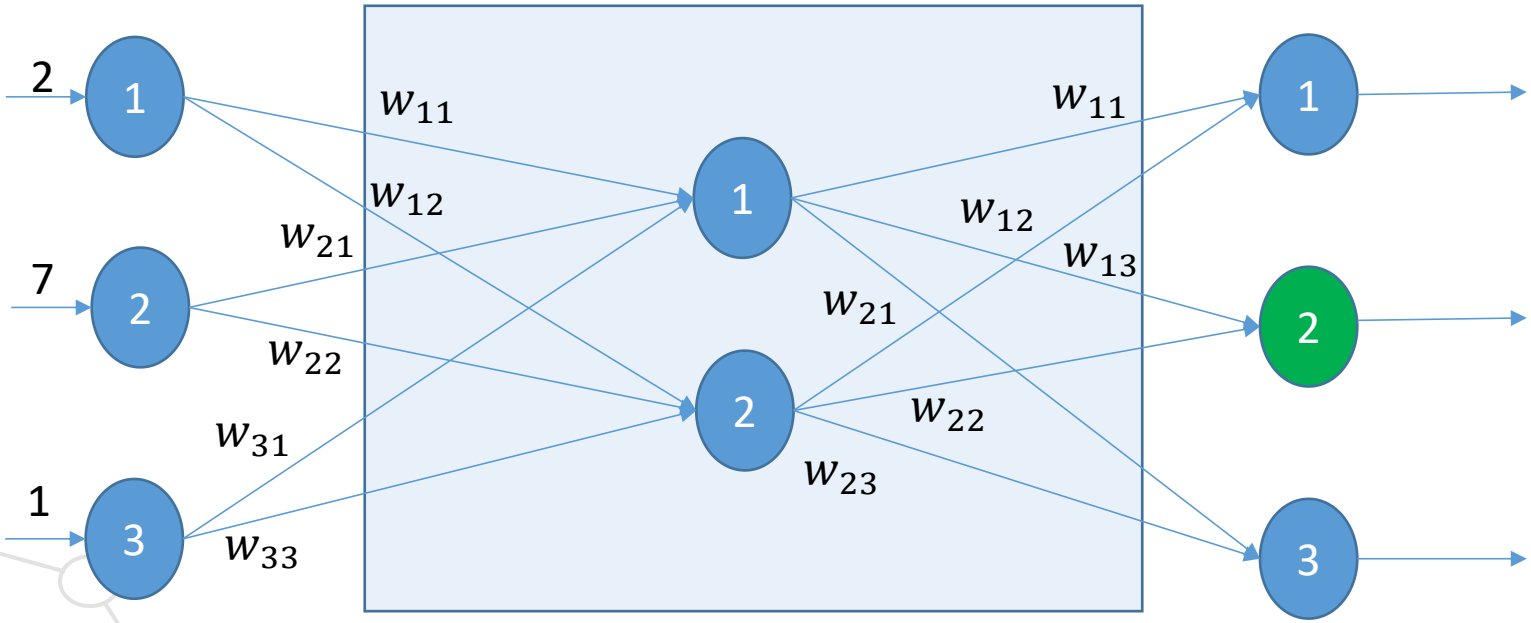
The softmax function converts its inputs, known as logit or logit scores, to be between 0 and 1, and also normalizes the outputs so that they all sum up to 1. In other words, the softmax function turns your logits into probabilities. Mathematically, the softmax function is defined as follows:

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$

Multilayer networks-Learning Process

- Step 1: Propagate the input forward through the network

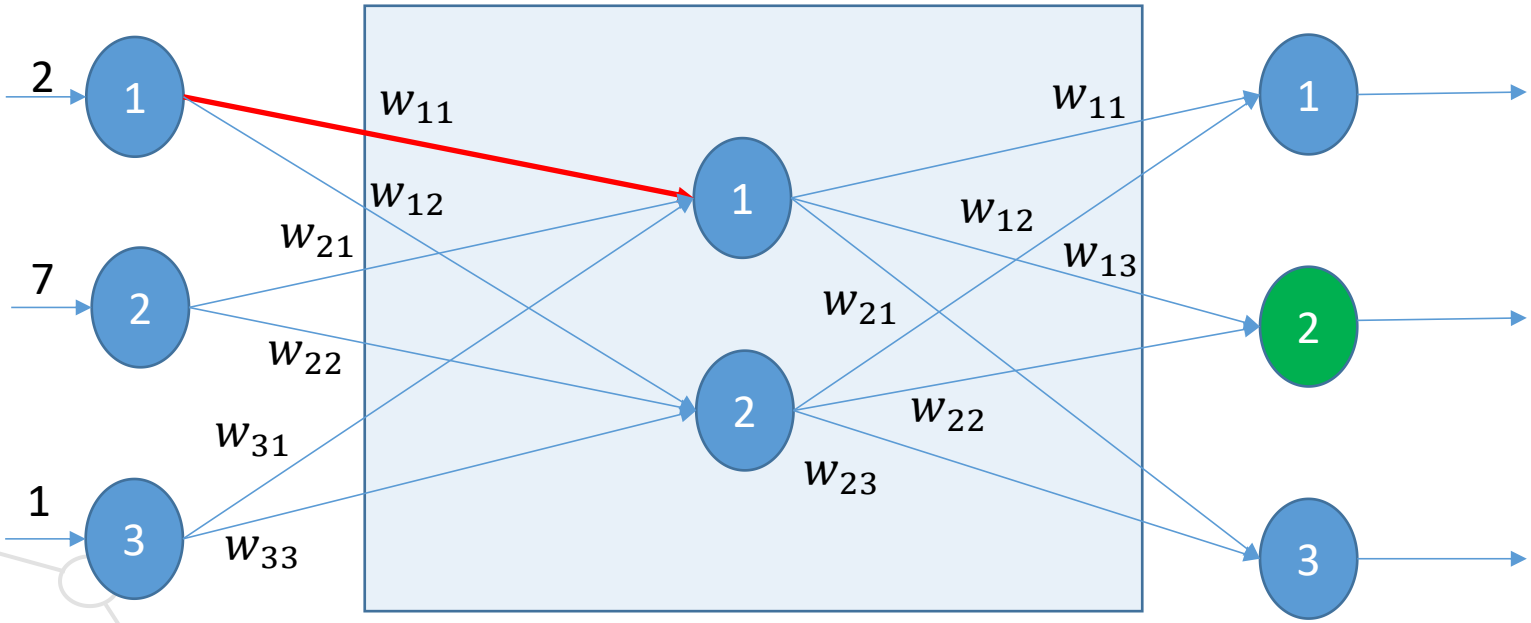
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 1: Propagate the input forward through the network

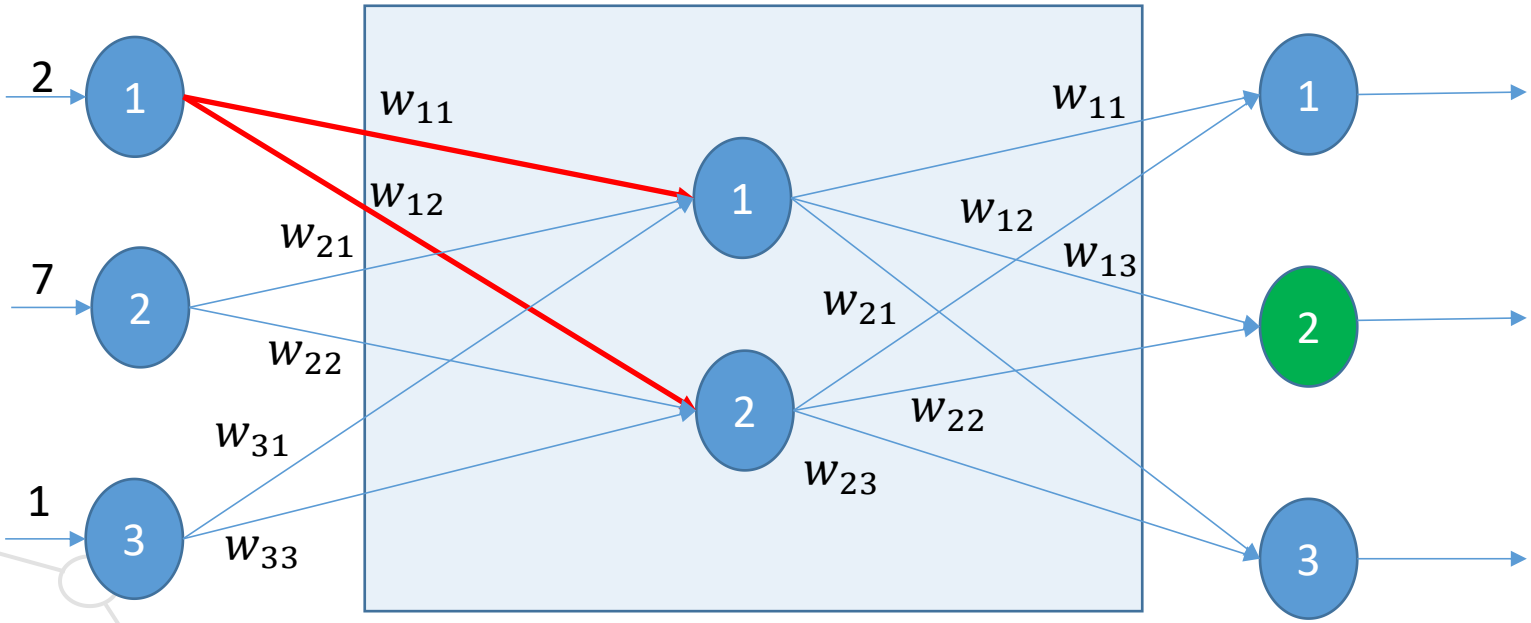
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 1: Propagate the input forward through the network

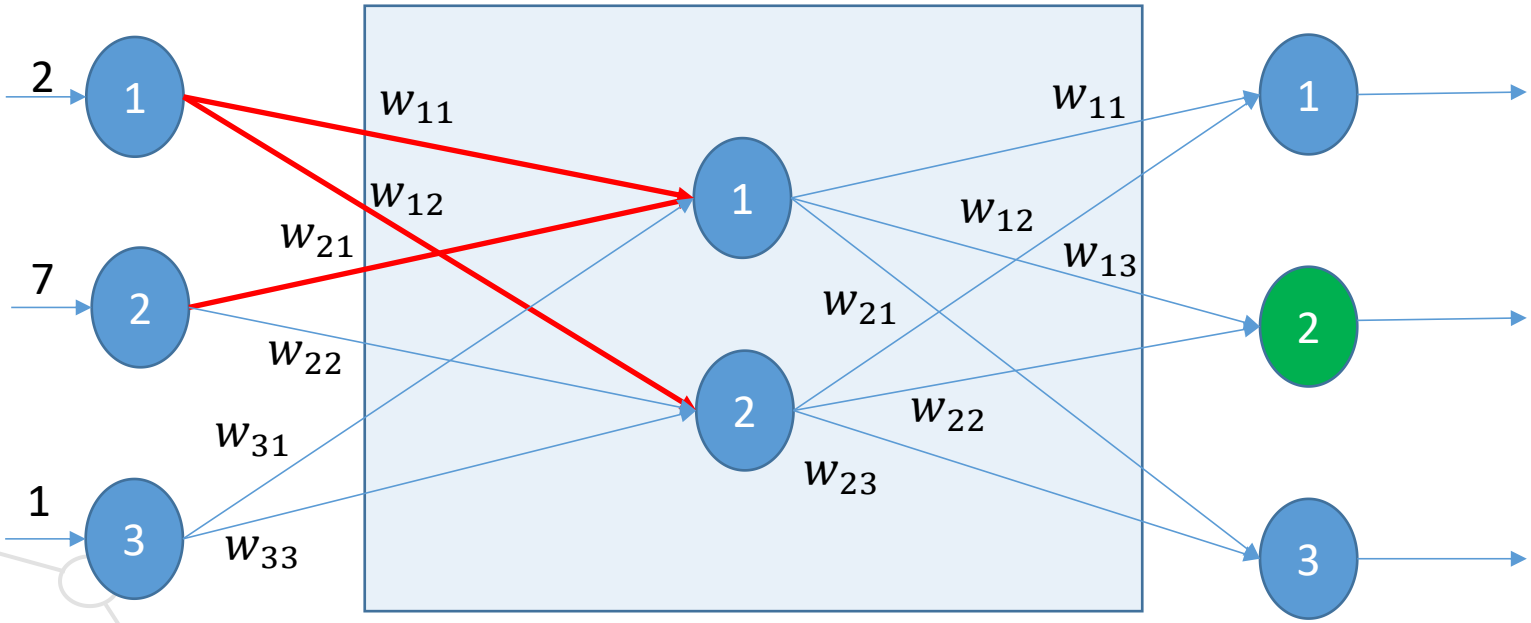
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 1: Propagate the input forward through the network

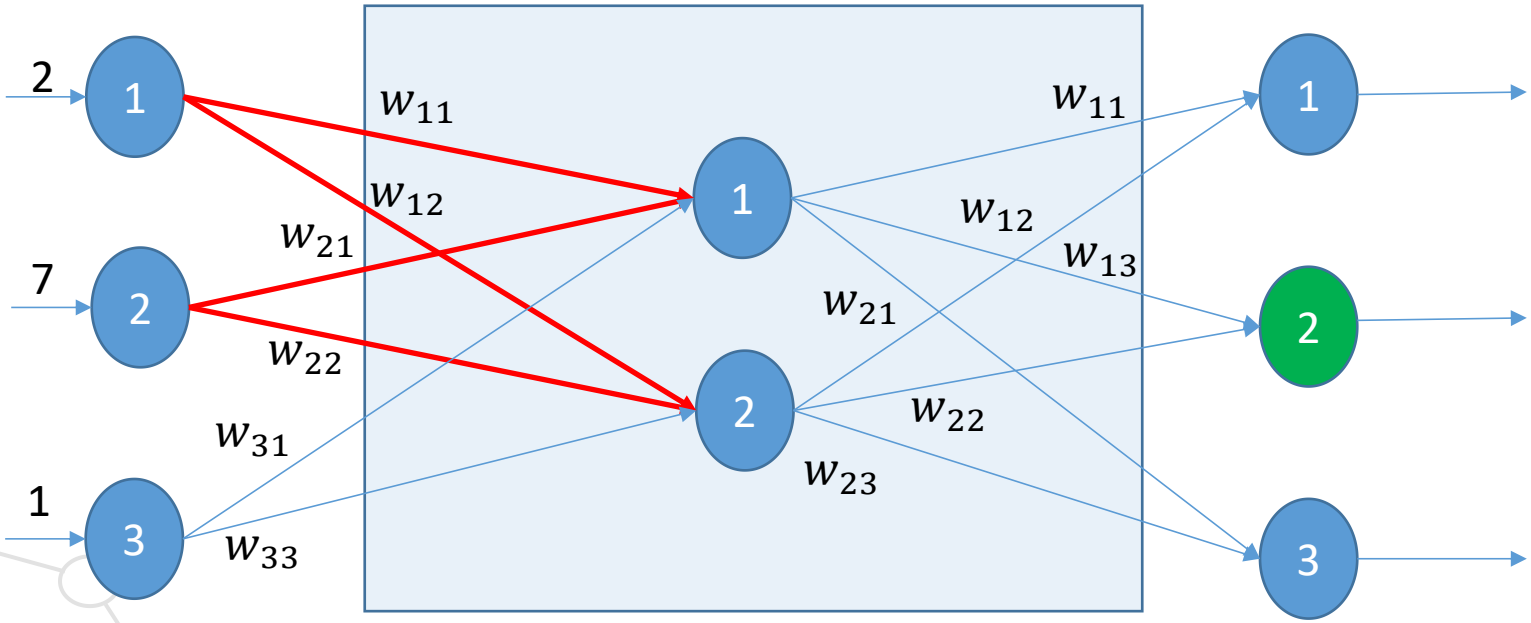
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 1: Propagate the input forward through the network

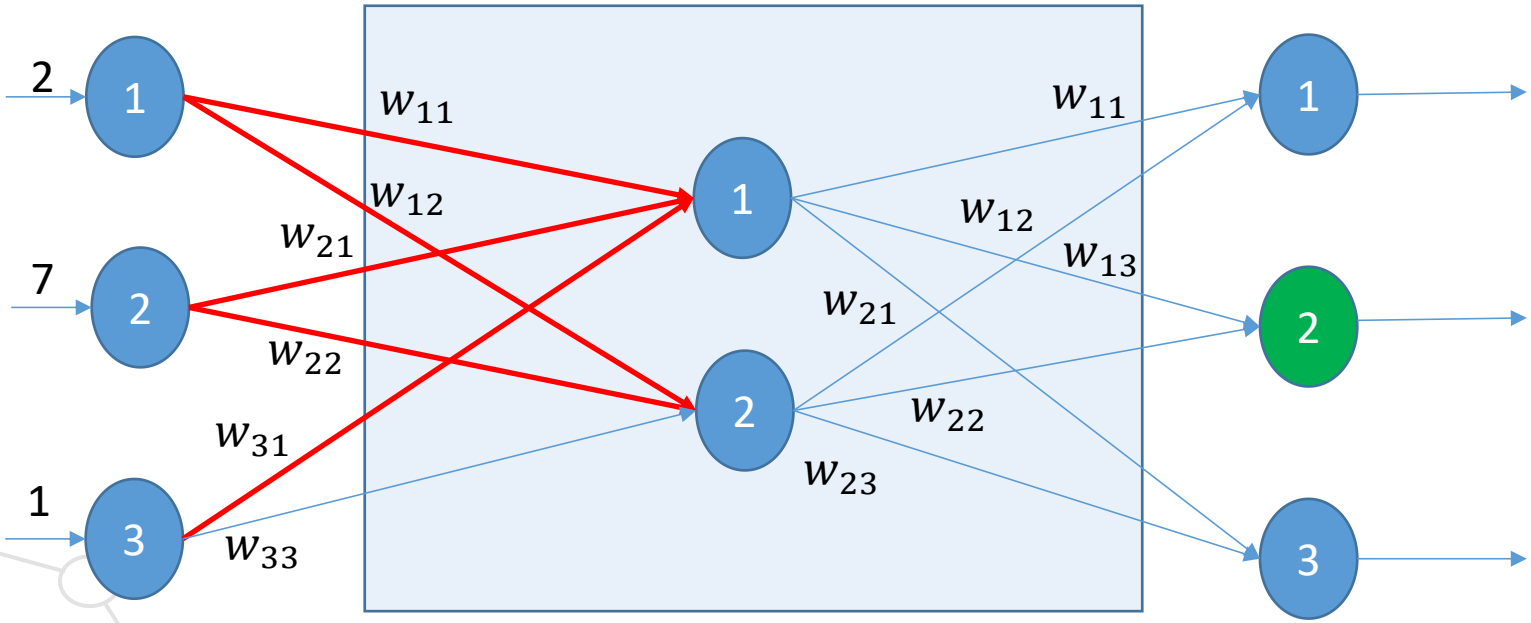
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 1: Propagate the input forward through the network

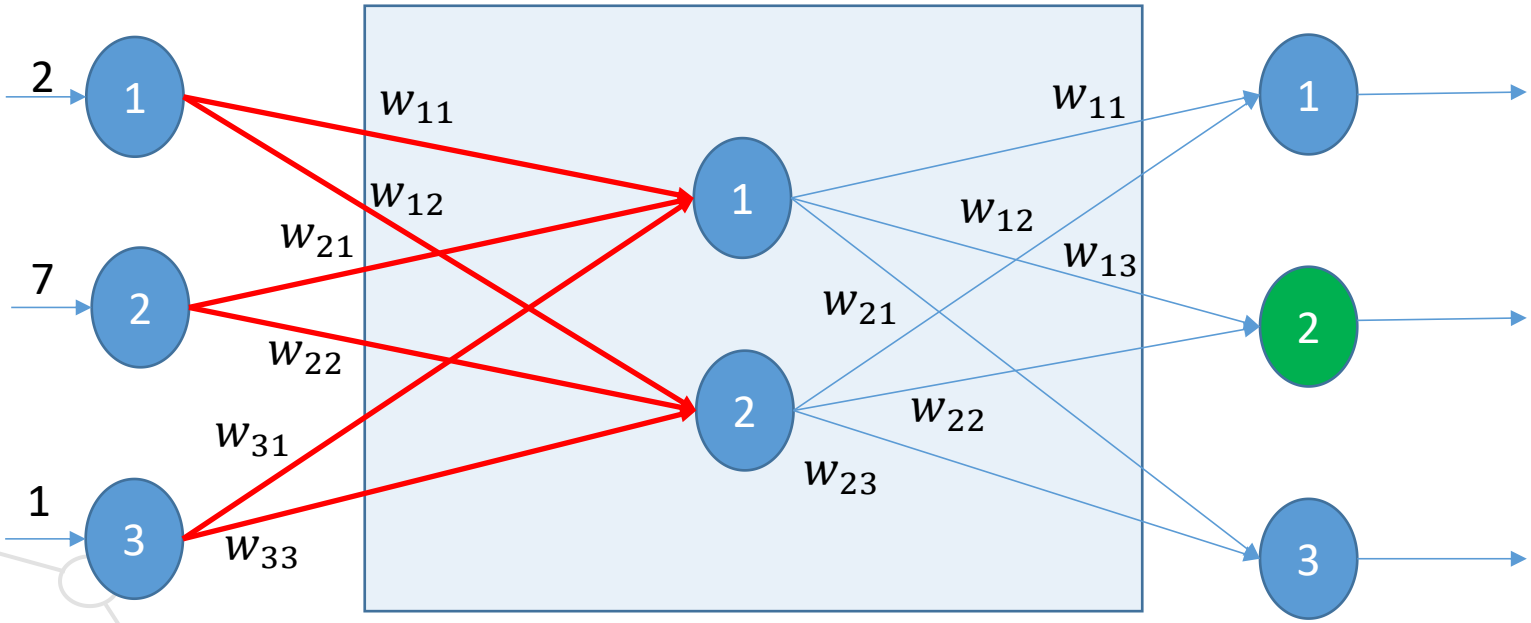
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 1: Propagate the input forward through the network

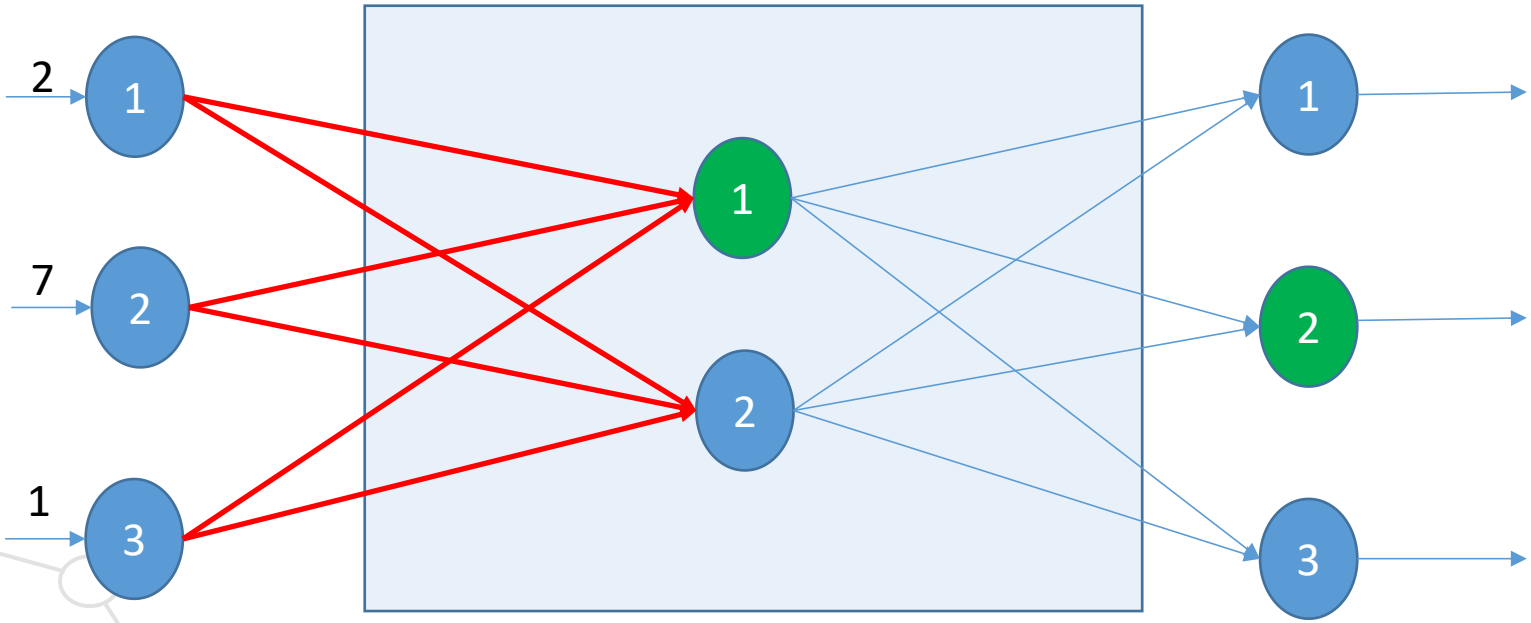
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 1: Propagate the input forward through the network

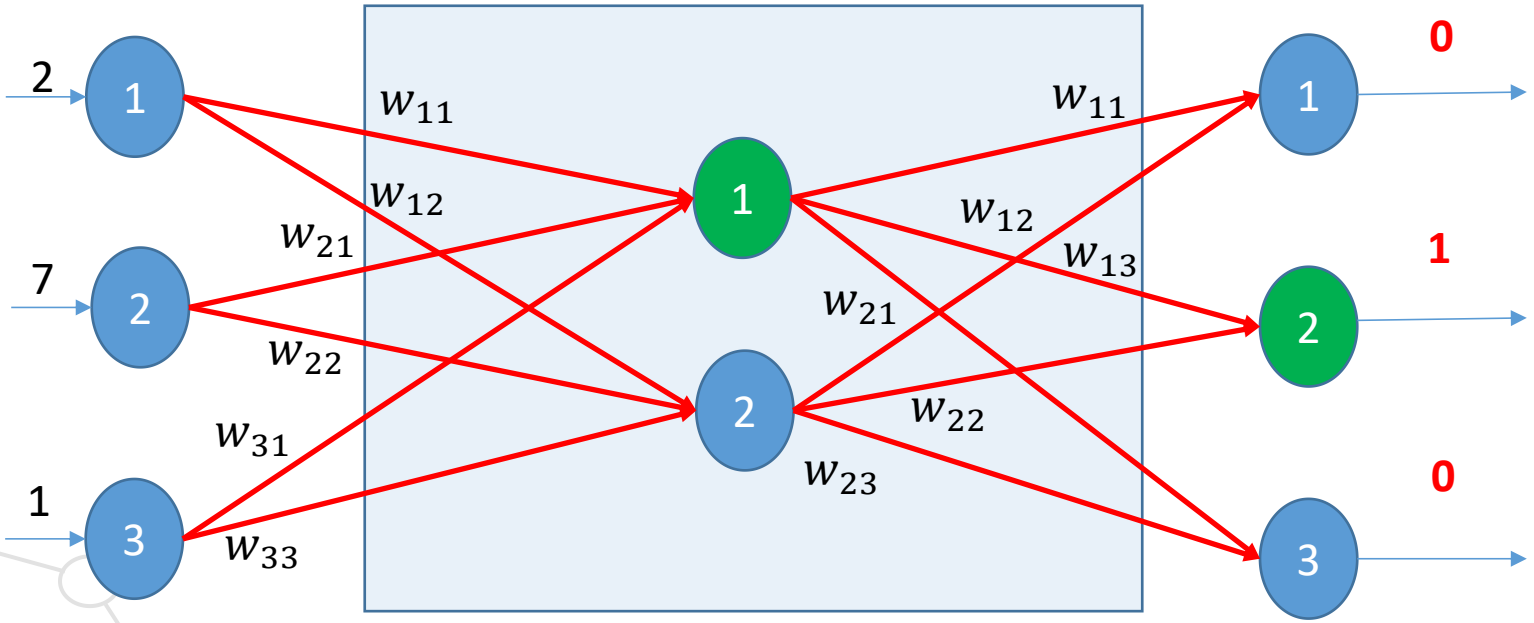
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 1: Propagate the input forward through the network

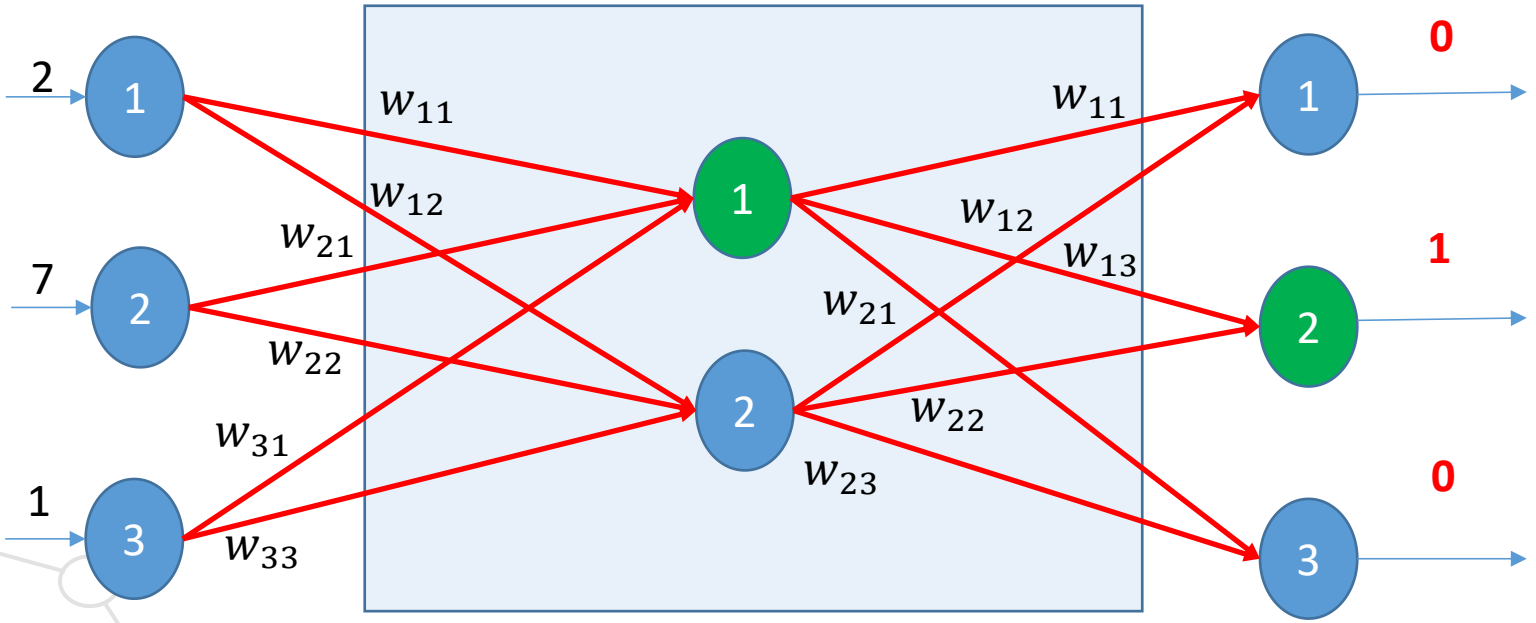
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 2: Determine the output values of the network

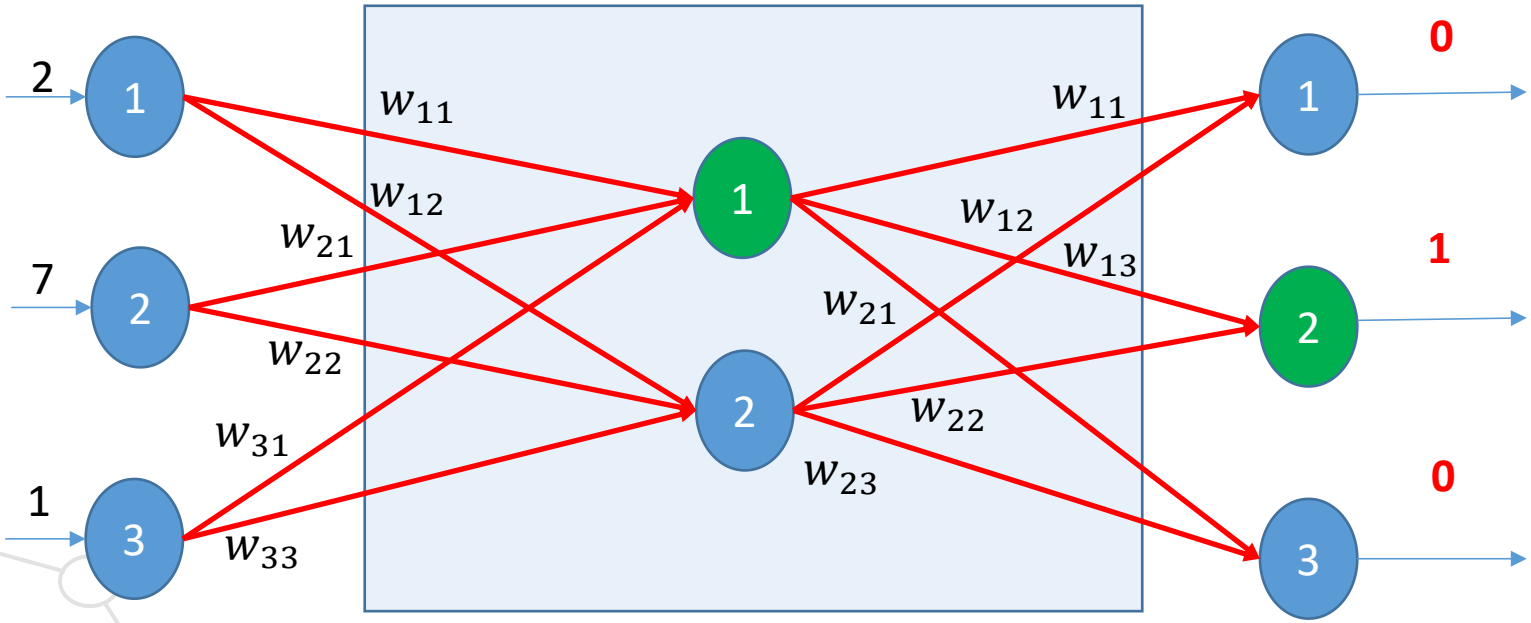
X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Multilayer networks-Learning Process

- Step 3: Learning Process (propagate the **error** backward)

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Input Neurons

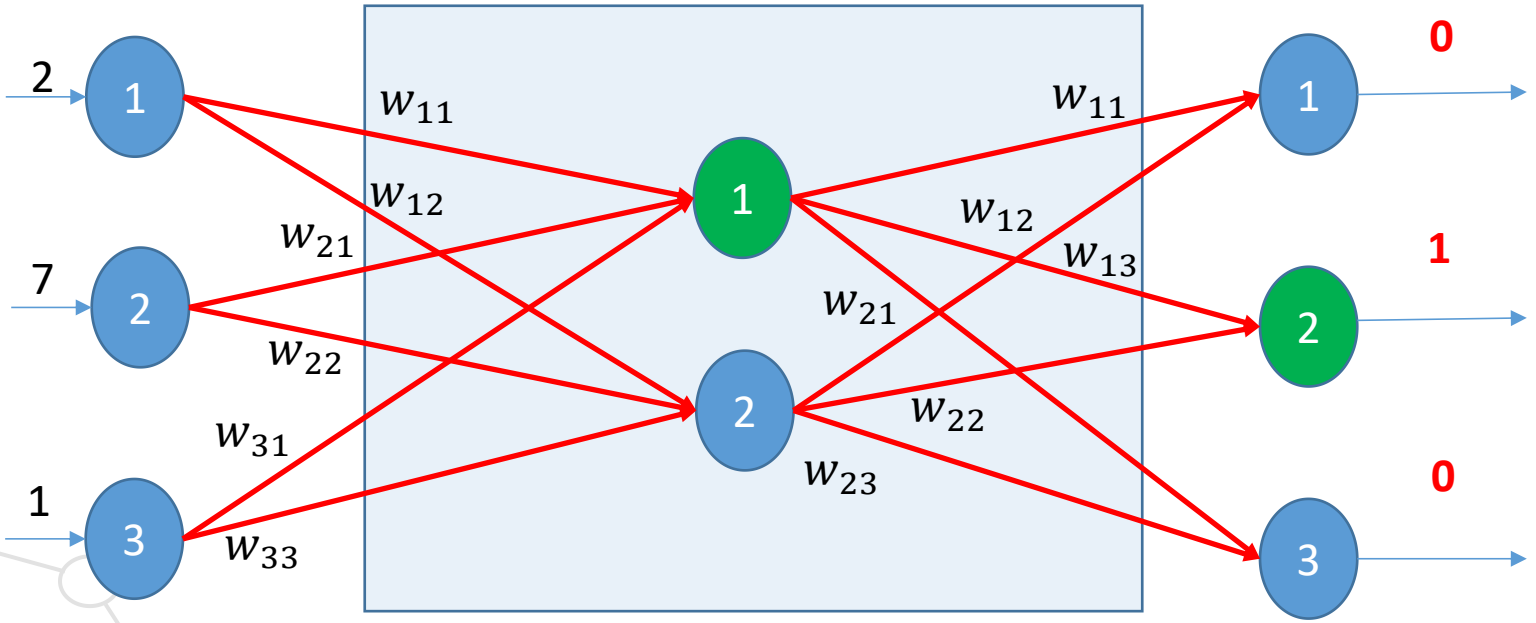
Output Neurons

Multilayer networks-Learning Process

- Like single perceptron, we want to update the weigths in this form:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



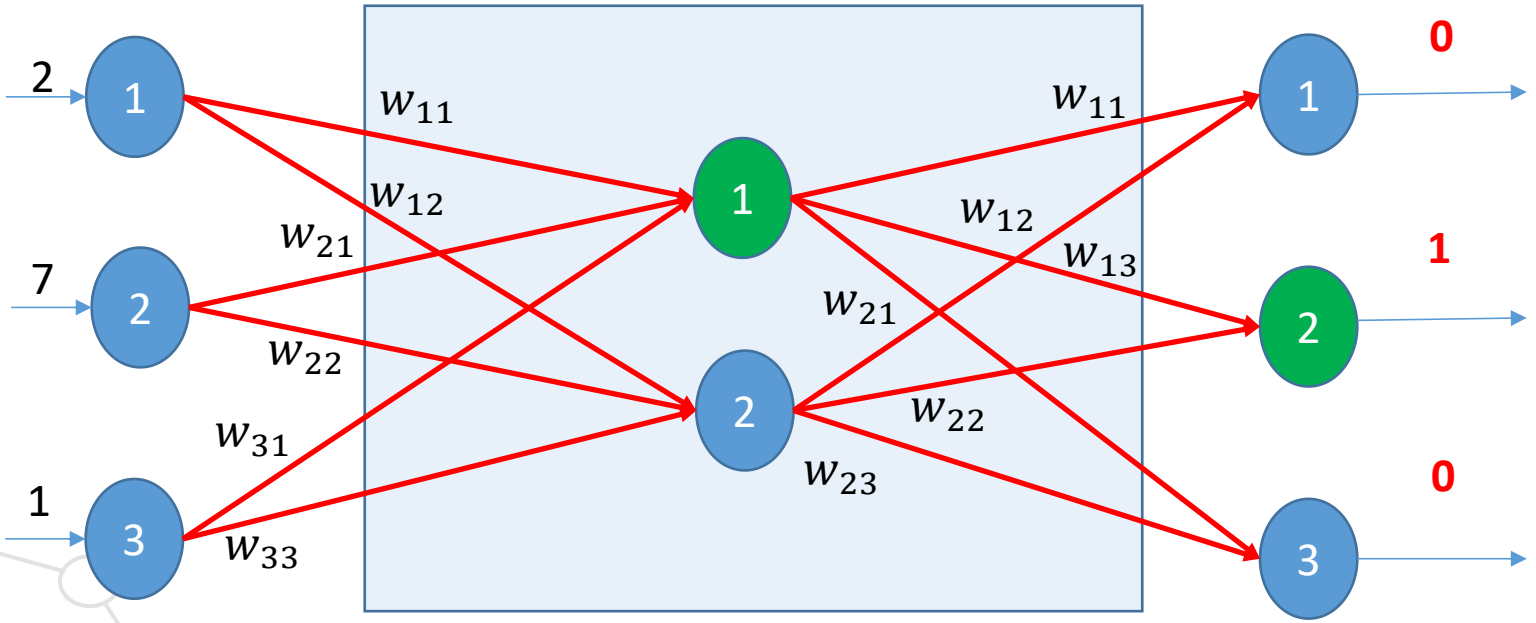
Multilayer networks-Learning Process

- Like single perceptron, we want to update the weigths in this form:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

BACKPROPAGATION METHOD

X1	X2	X3	c
2	7	1	0
3	6	5	1
5	4	7	2
6	8	5	0
7	9	7	1
8	0	8	2



Design the neural network architecture

Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

Design the neural network architecture

Assume a problem with the following dataset:

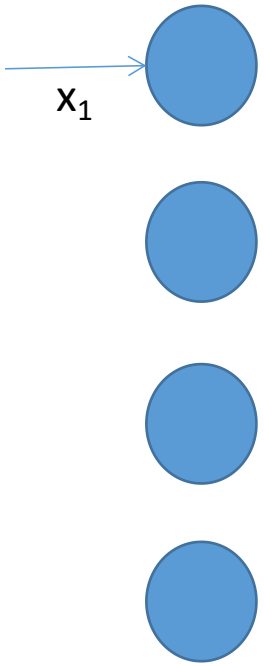
x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4

Design the neural network architecture

Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

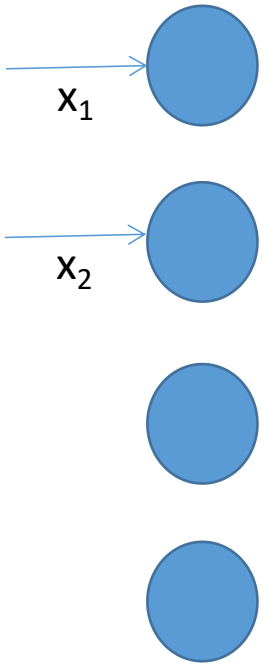


We have three classes and feature vectors in R^4

Design the neural network architecture

Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

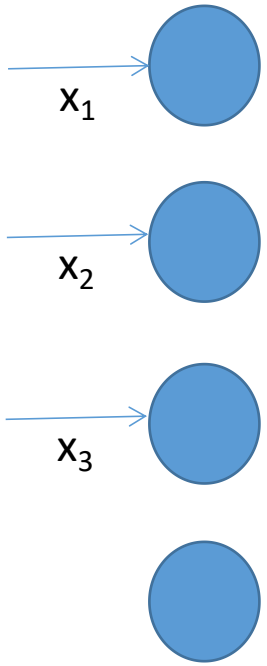


We have three classes and feature vectors in R^4

Design the neural network architecture

Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1



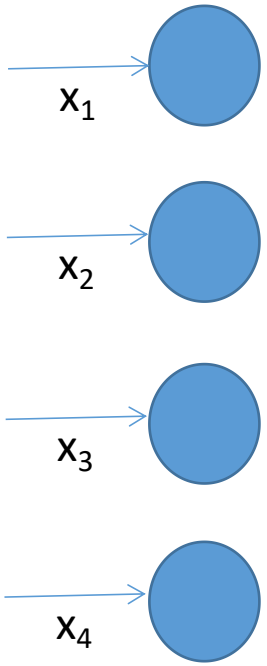
We have three classes and feature vectors in R^4

Design the neural network architecture

Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

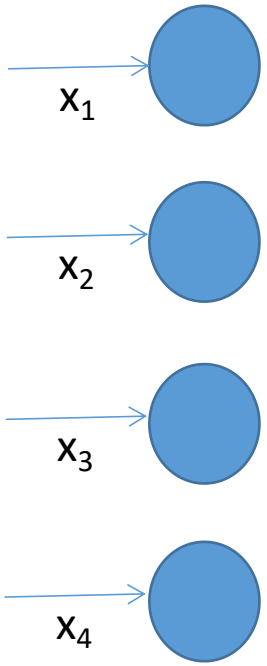
How many outputs?

Design the neural network architecture

Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

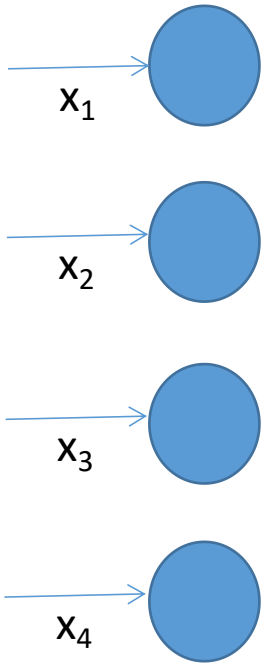
Option 1: As many as the number of classes

Design the neural network architecture

Assume a problem with the following dataset:

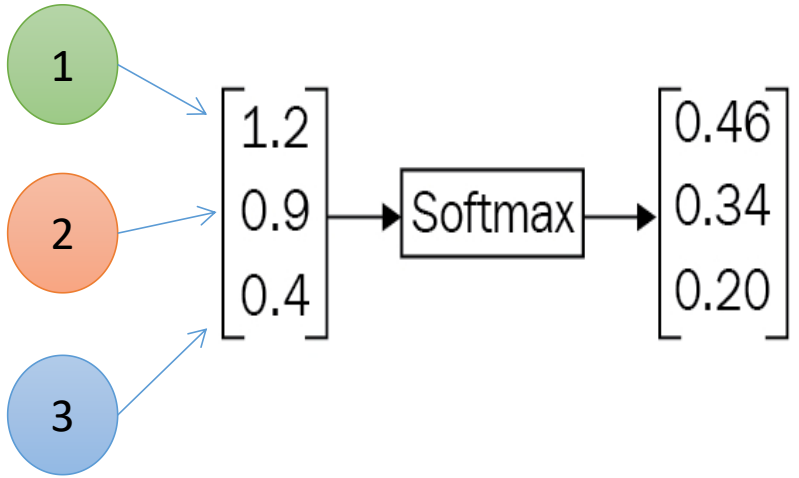
x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

Option 1: As many as the number of classes

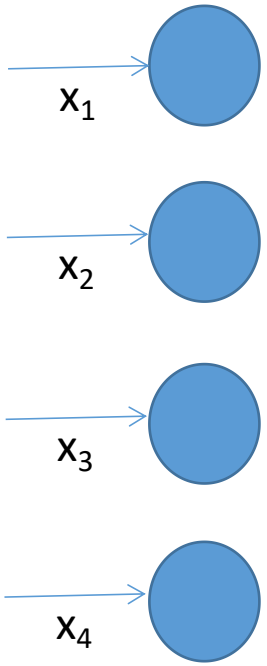


Design the neural network architecture

Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

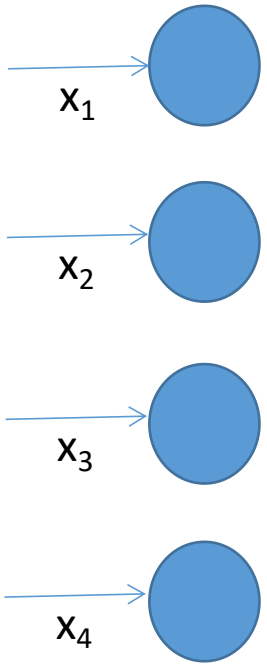
Option 2:
Use binary neurons and encode the class labels with them

Design the neural network architecture

Assume a problem with the following dataset:

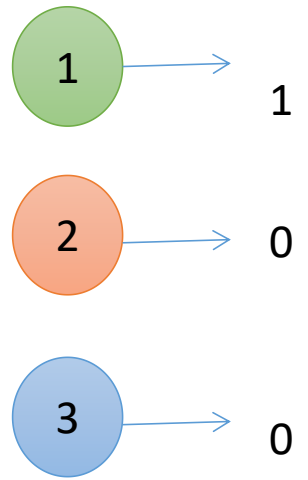
x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

Option 2:
Encode the class labels based on binary output of the neurons



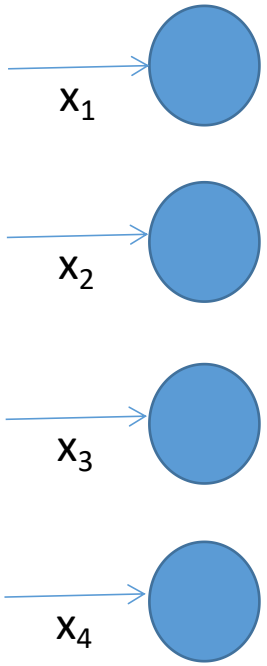
Class 1

Design the neural network architecture

Assume a problem with the following dataset:

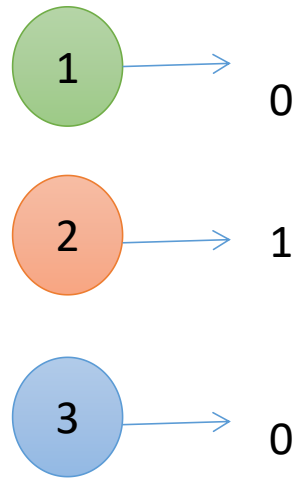
x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

Option 2:
Encode the class labels based on binary output of the neurons



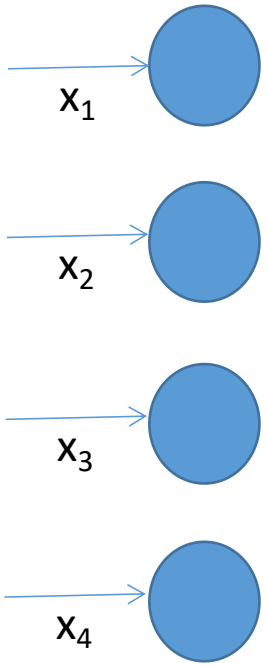
Class 2

Design the neural network architecture

Assume a problem with the following dataset:

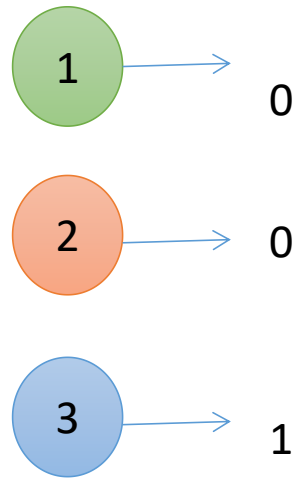
x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

Option 2:
Encode the class labels based on binary output of the neurons



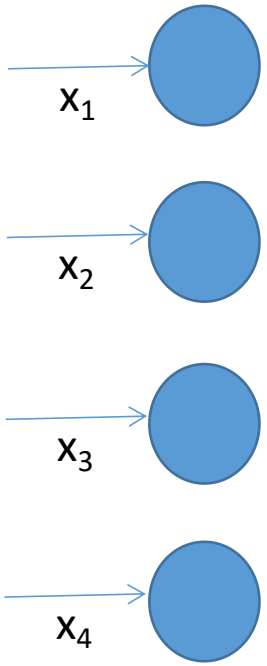
Class 3

Design the neural network architecture

Assume a problem with the following dataset:

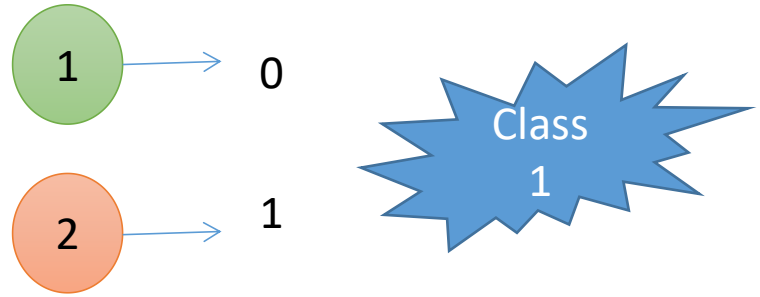
x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

Option 3: Code the class as binary number based on the outputs values

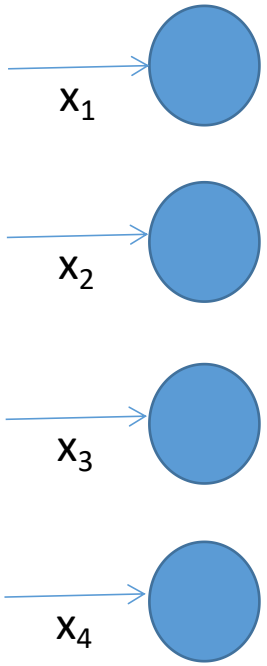


Design the neural network architecture

Assume a problem with the following dataset:

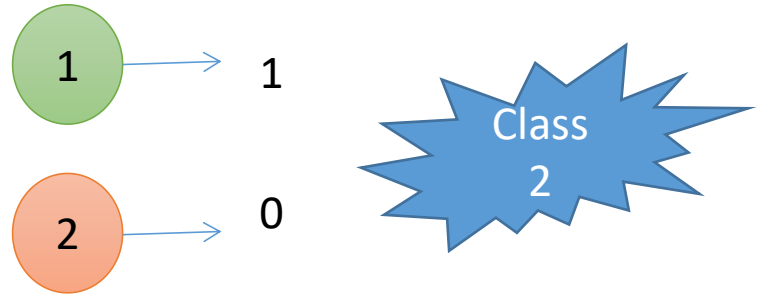
x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

Option 3: Code the class as binary number based on the outputs values

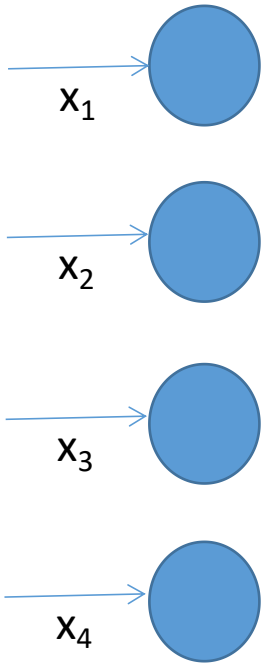


Design the neural network architecture

Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons

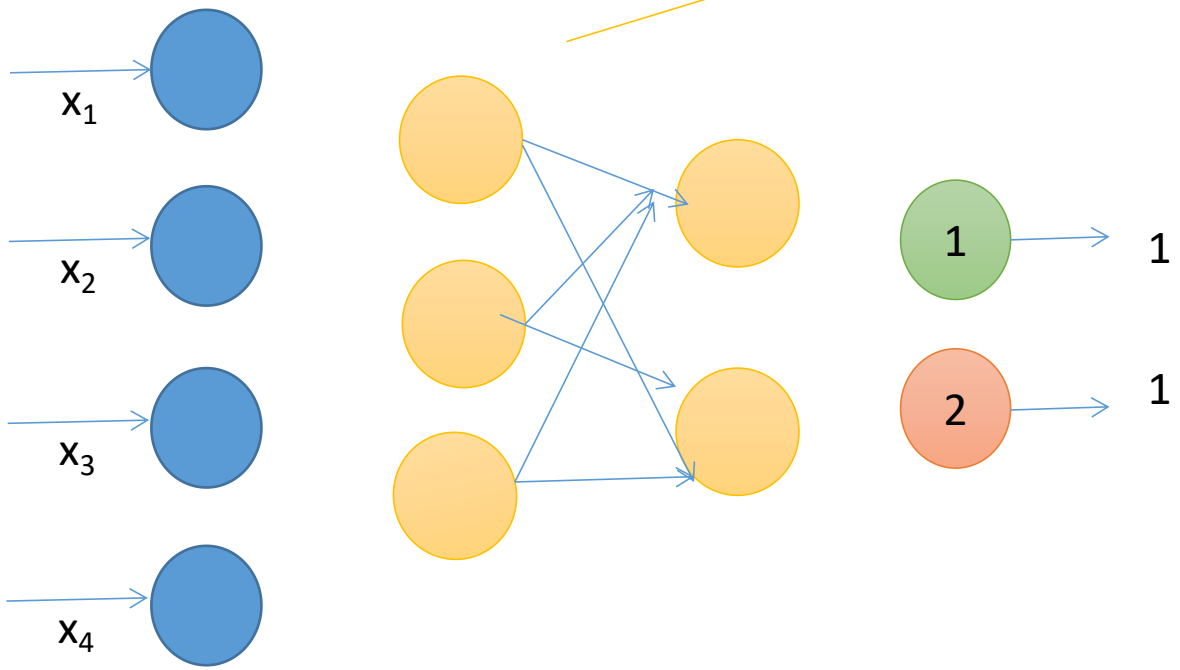
Option 3: Code the class as binary number based on the outputs values

Design the neural network architecture

Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



The number of hidden neurons is a parameter

Input neurons

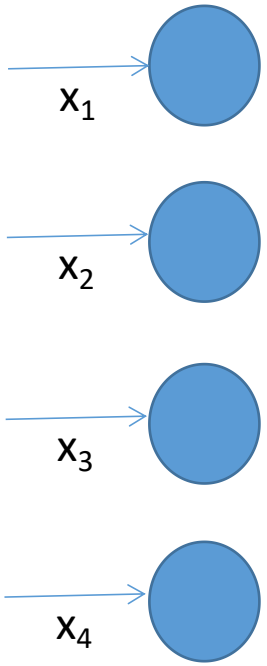
Output neurons

Design the neural network architecture

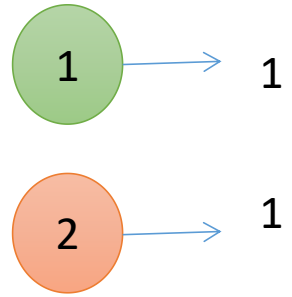
Assume a problem with the following dataset:

x1	x2	x3	x4	y
3	6	12	5	3
5	5	1	7	2
1	8	4	3	1
4	9	5	9	3
6	10	7	4	1

We have three classes and feature vectors in R^4



Input neurons



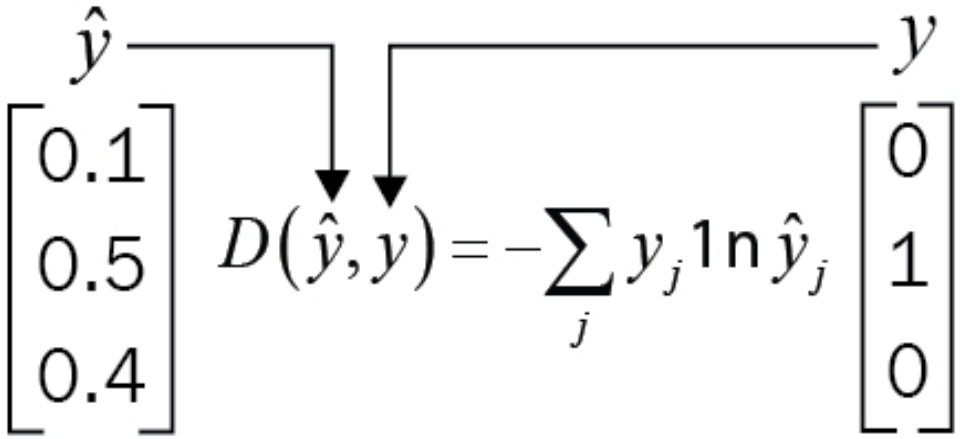
Output neurons

Loss Function

- The MLPN use different type of error functions in order to update the weights.
- What loss function to use? Depends on the problem
- For classification problems (binary and multiclass), you can use **Cross-Entropy**
- For regression problems, we can use **Mean Squared Error**.

Loss Function- Cross Entropy

- Cross entropy measures the similarity distance between two probability vectors by means of **cross-entropy**.
- In neural networks, we can use this measure to determine the loss (error) of a neural model. This measure is defined as:



Model Prediction

Actual Values

Exercises

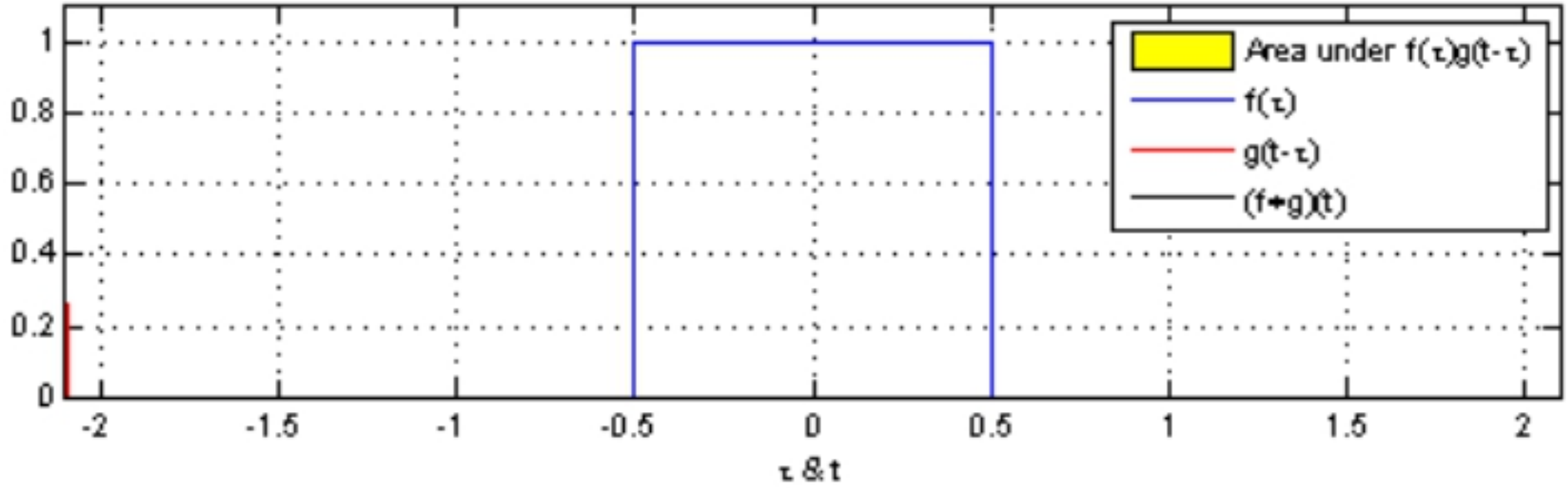
1. Review script of loss functions
2. Review scripts of Multilayer Percentron Networks

Convolution

Introduction

Convolution

In mathematics convolution is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other.



Source: Wikipedia

Introduction

CONVOLUTION

There are several possible notations to indicate the convolution of two (multi-dimensional) signals to produce an output signal. The most common are:⊖

$$c = a \otimes b = a * b$$

We shall use the first form, $c = a \otimes b$, with the following formal definitions.

Introduction

In 2D continuous space:

$$c(x,y) = a(x,y) \otimes b(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} a(\chi,\zeta) b(x-\chi, y-\zeta) d\chi d\zeta$$

In 2D discrete space:

$$c[m,n] = a[m,n] \otimes b[m,n] = \sum_{j=-\infty}^{+\infty} \sum_{k=-\infty}^{+\infty} a[j,k] b[m-j, n-k]$$

Introduction

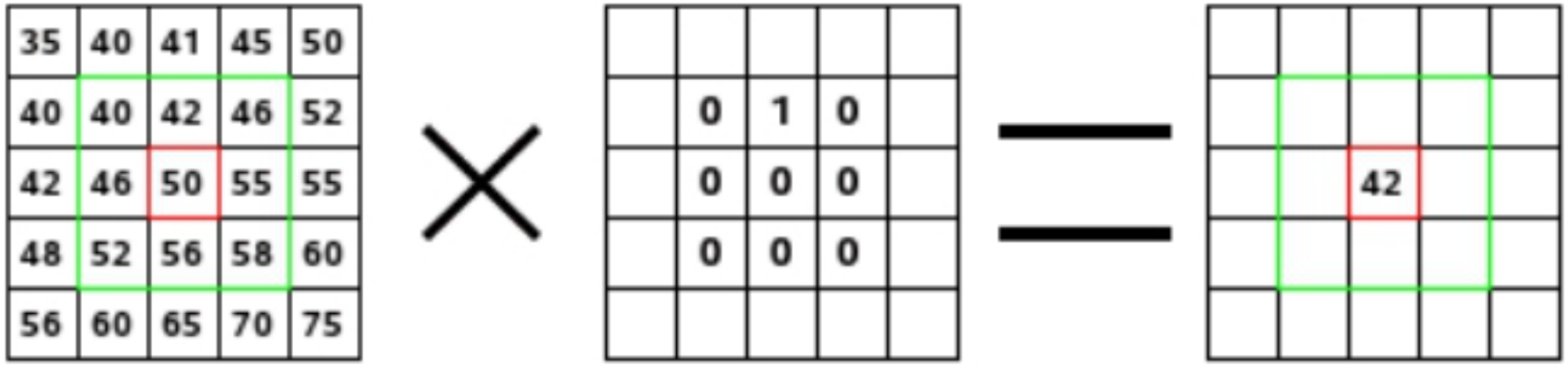
What We See

Considering only one channel

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

What Computers See

Introduction



Introduction

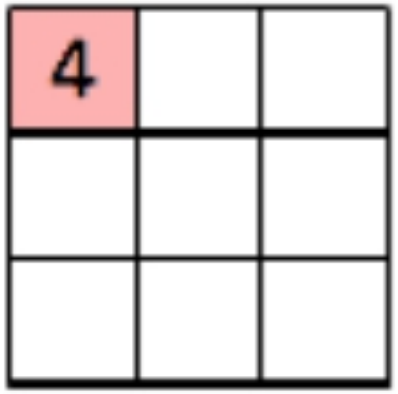
	0	1	0	
	1	-4	1	
	0	1	0	

Introduction

	0	1	0	
	1	-4	1	
	0	1	0	

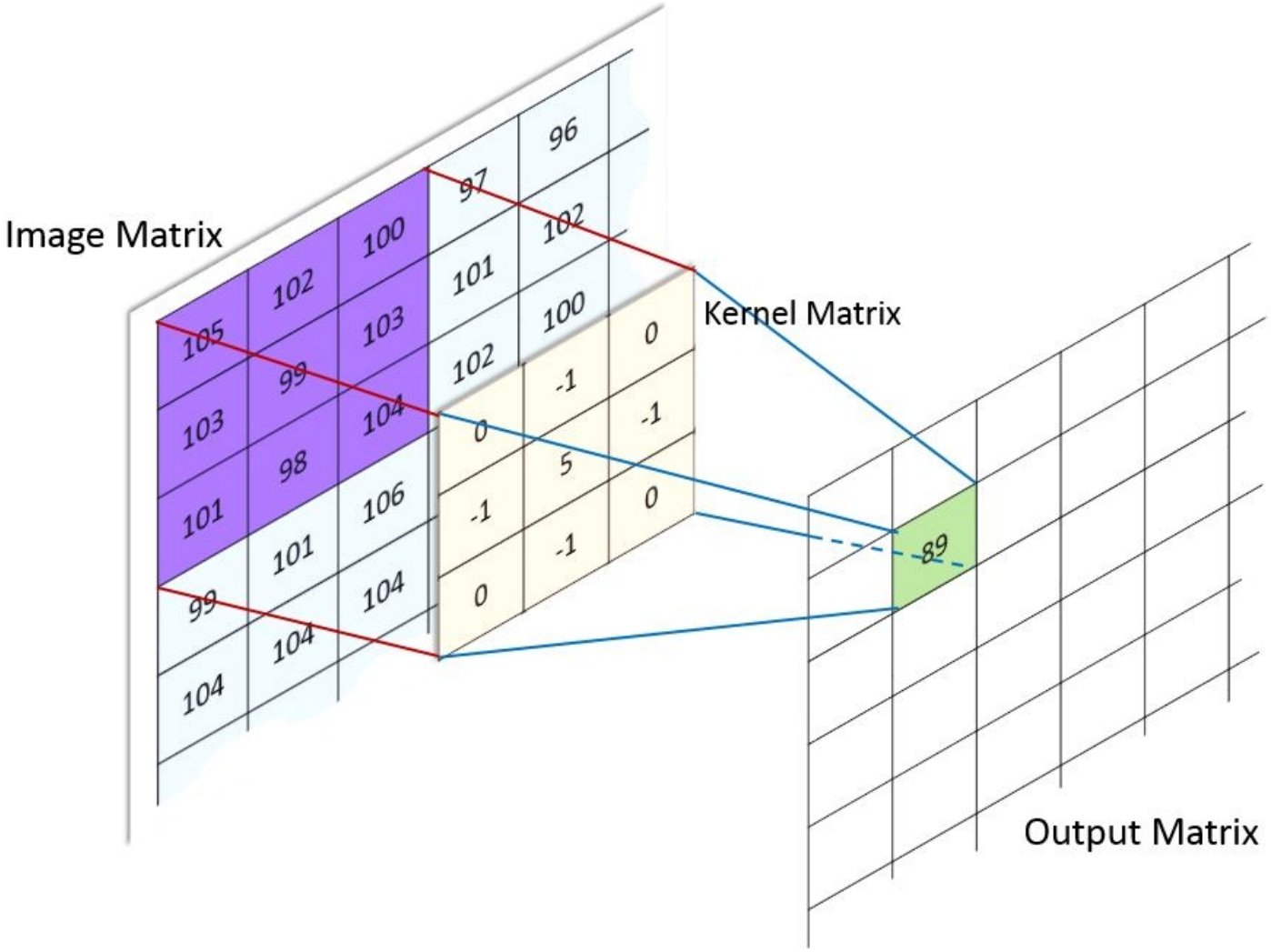
Introduction

Image

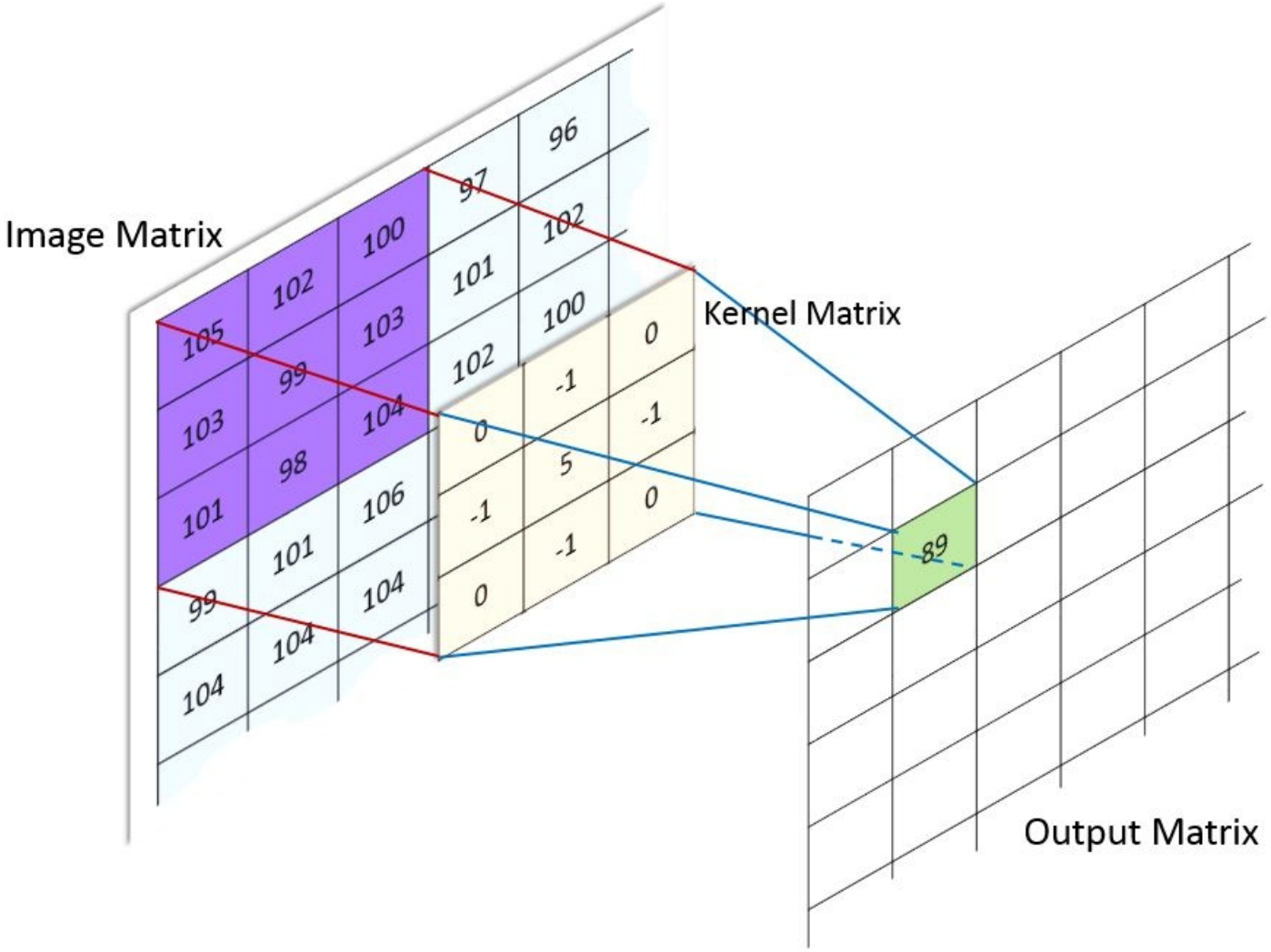


Convolved Feature

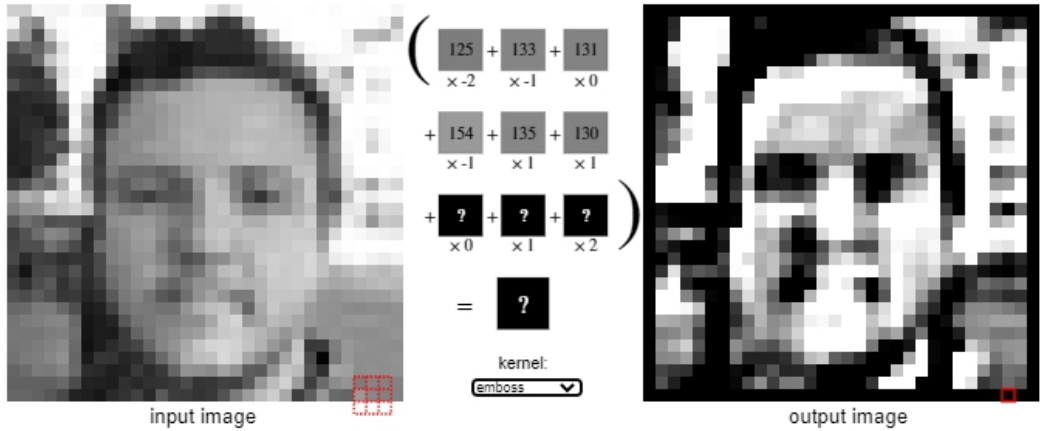
Introduction

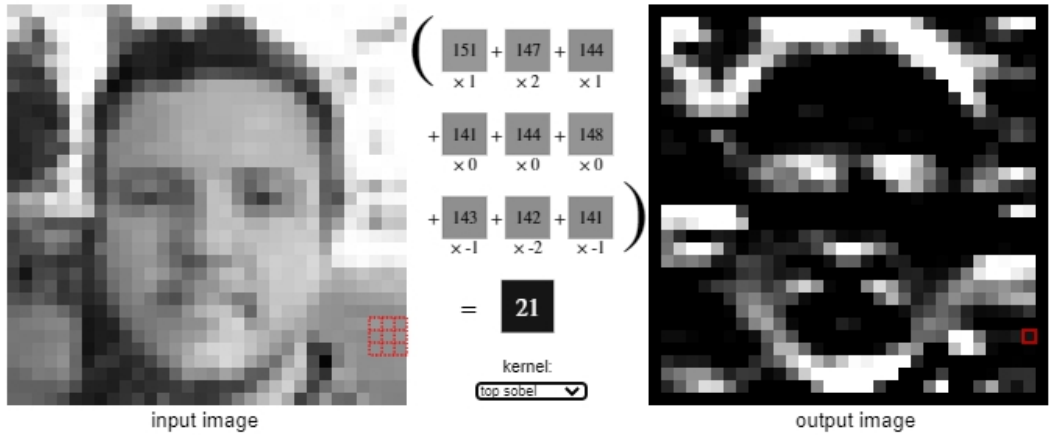


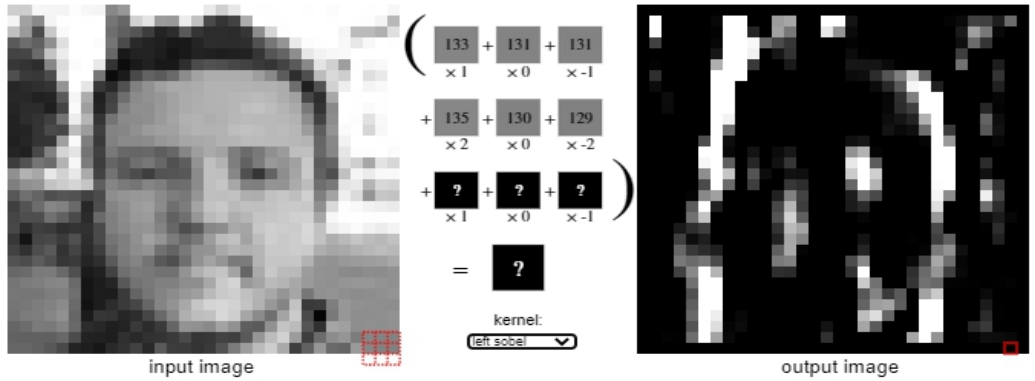
Introduction



Kernel Effects





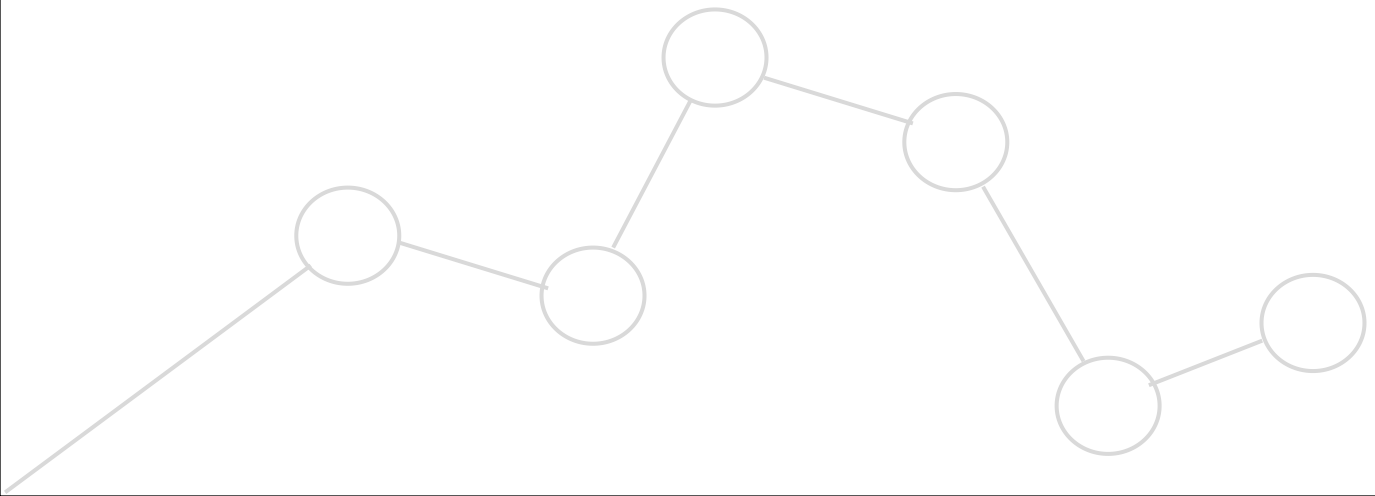


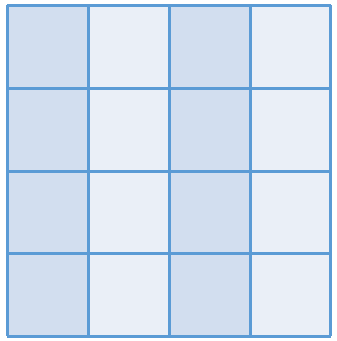
https://setosa.io/ev/image-kernels/

Cinvestav
Tamaulipas

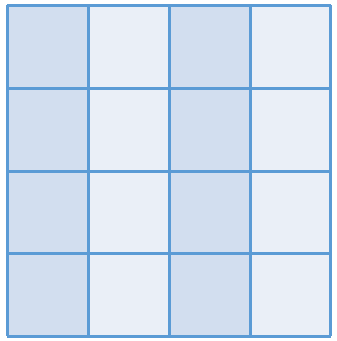
Deep Learning

- An introduction to Convolutional Neural Networks

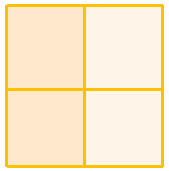


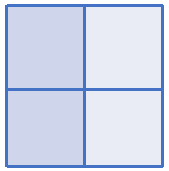


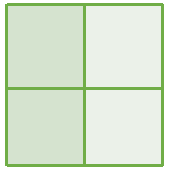
4x4
One channel

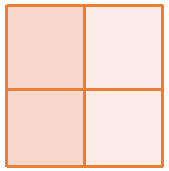


4x4
One channel

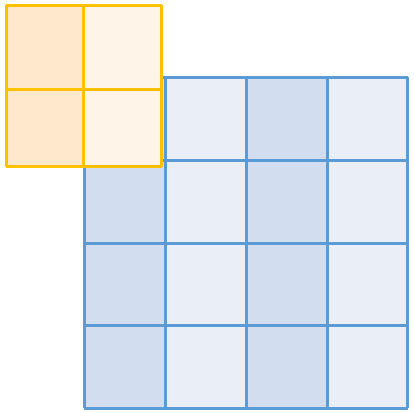




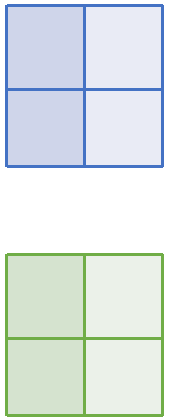


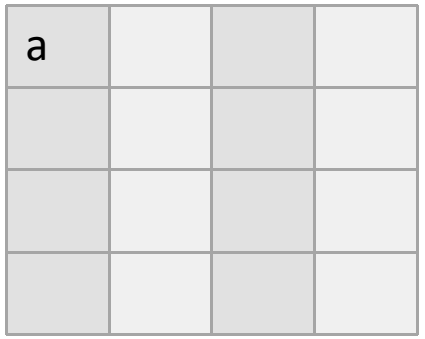


Convolution Stage

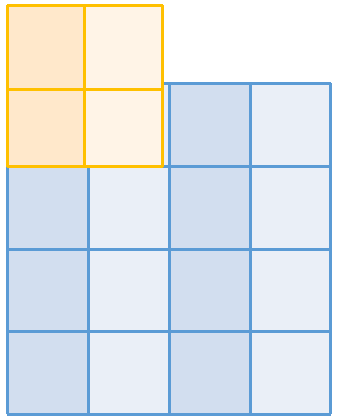


4x4
One channel

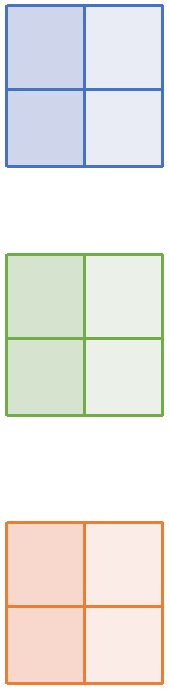


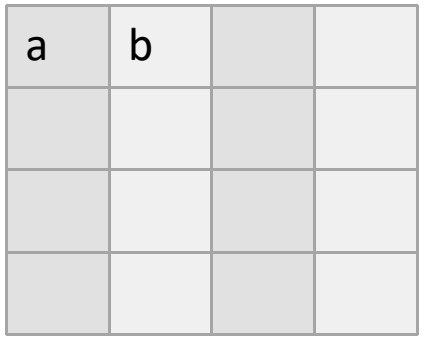


Convolution Stage

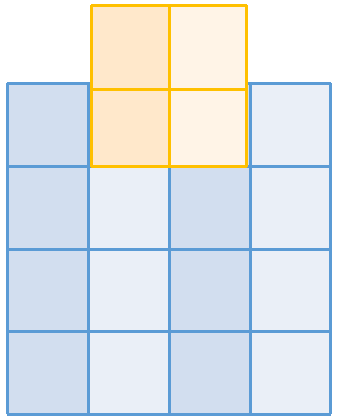


4x4
One channel

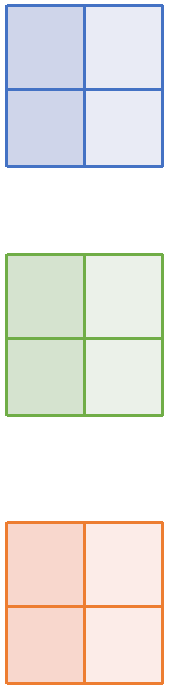




Convolution Stage

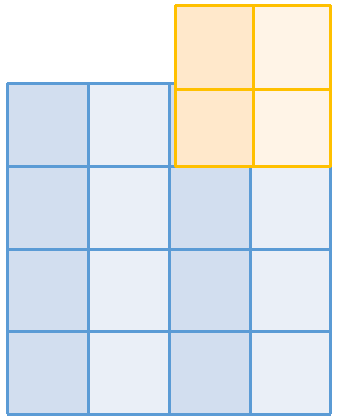


4x4
One channel

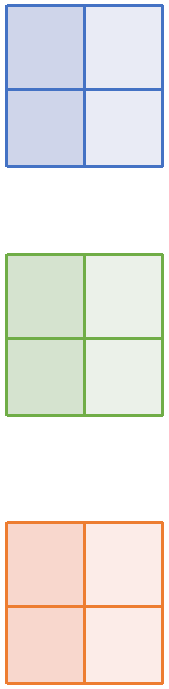


a	b	c	

Convolution Stage

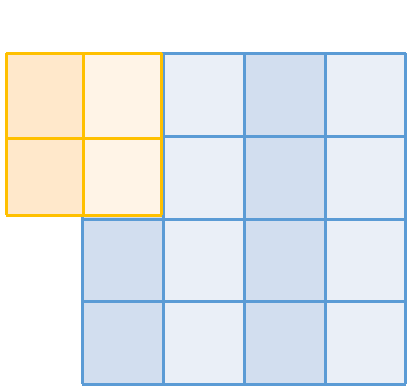


4x4
One channel

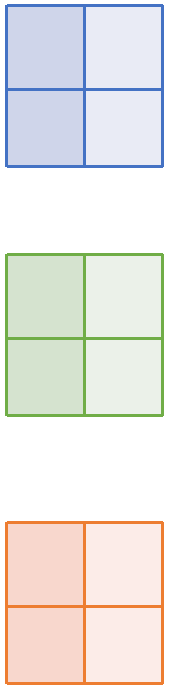


a	b	c	d

Convolution Stage

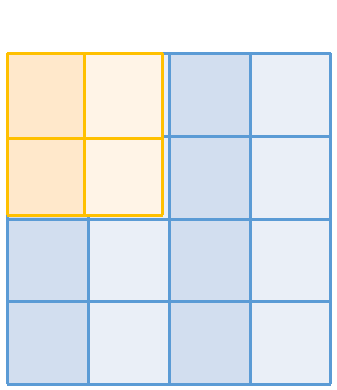


4x4
One channel

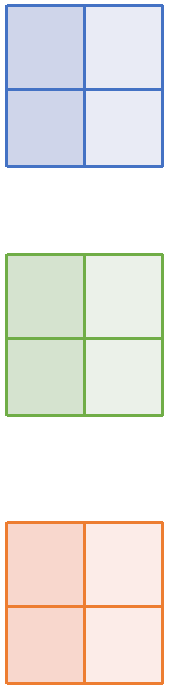


a	b	c	d
e			

Convolution Stage

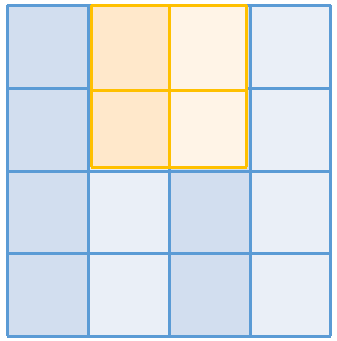


4x4
One channel

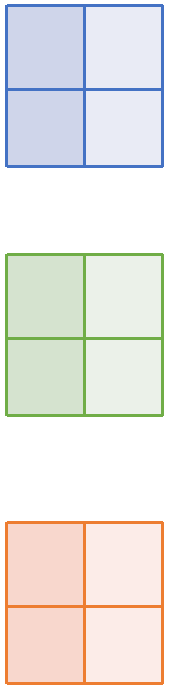


a	b	c	d
e	f		

Convolution Stage

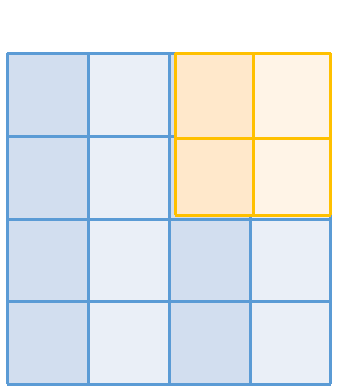


4x4
One channel

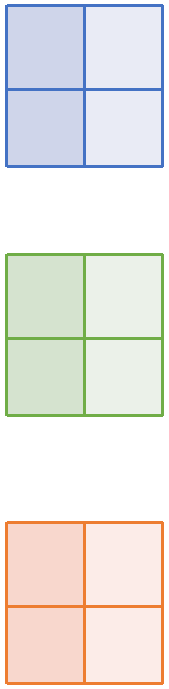


a	b	c	d
e	f	g	

Convolution Stage

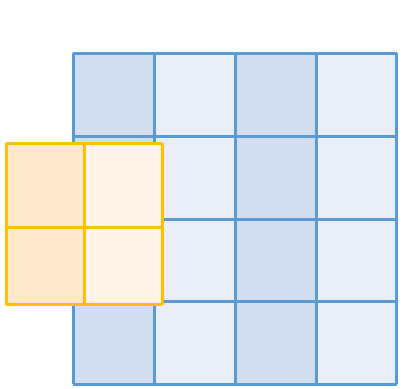


4x4
One channel

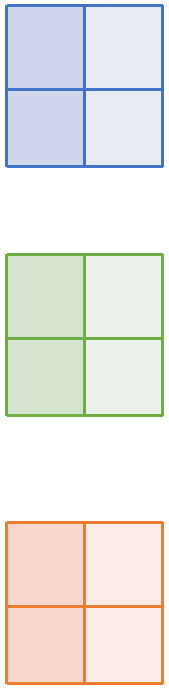


a	b	c	d
e	f	g	h

Convolution Stage

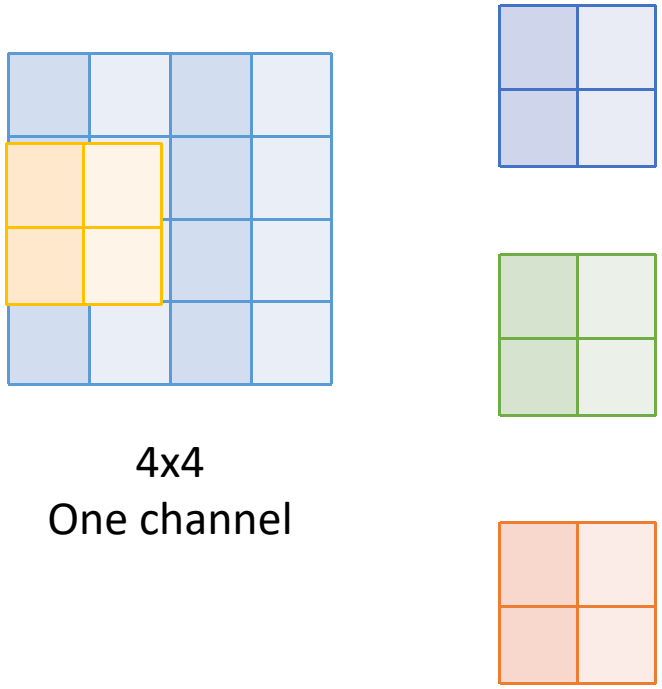


4x4
One channel



a	b	c	d
e	f	g	h
i			

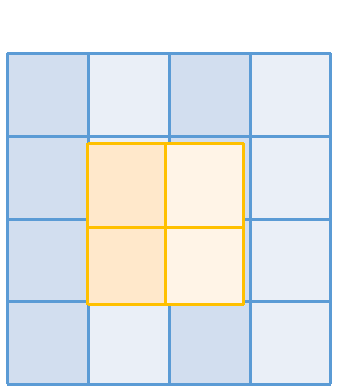
Convolution Stage



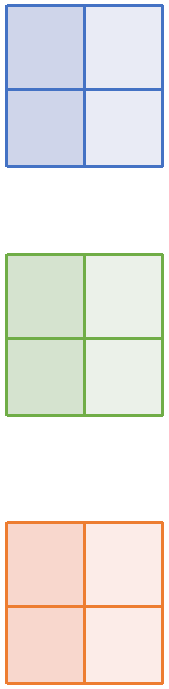
4x4
One channel

a	b	c	d
e	f	g	h
i	j		

Convolution Stage

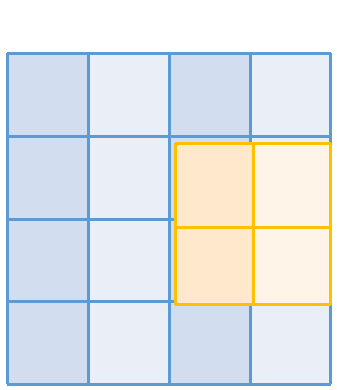


4x4
One channel

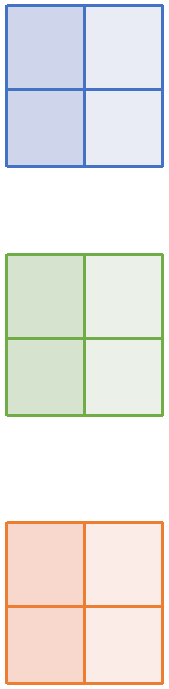


a	b	c	d
e	f	g	h
i	j	k	

Convolution Stage

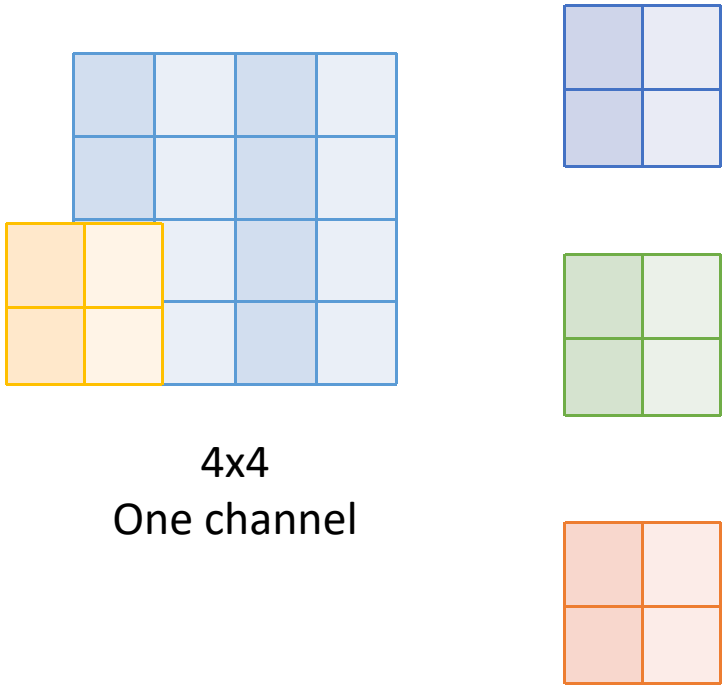


4x4
One channel



a	b	c	d
e	f	g	h
i	j	k	l

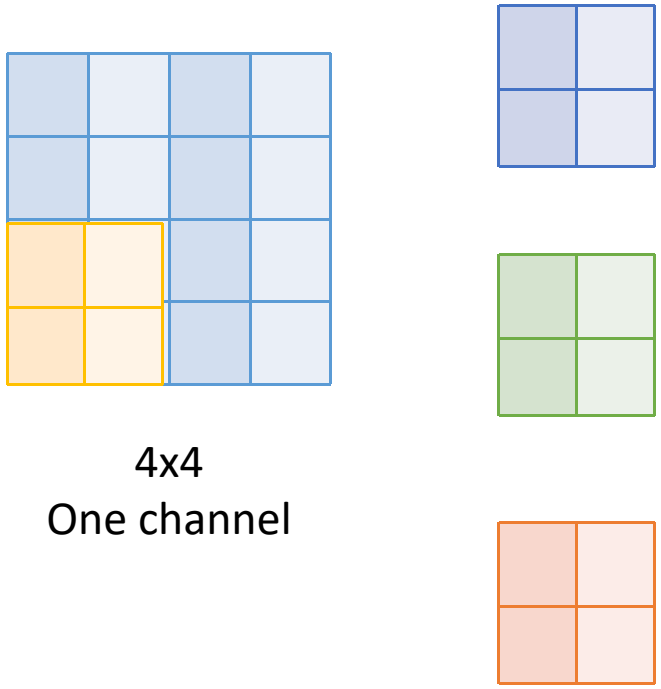
Convolution Stage



4x4
One channel

a	b	c	d
e	f	g	h
i	j	k	l
m			

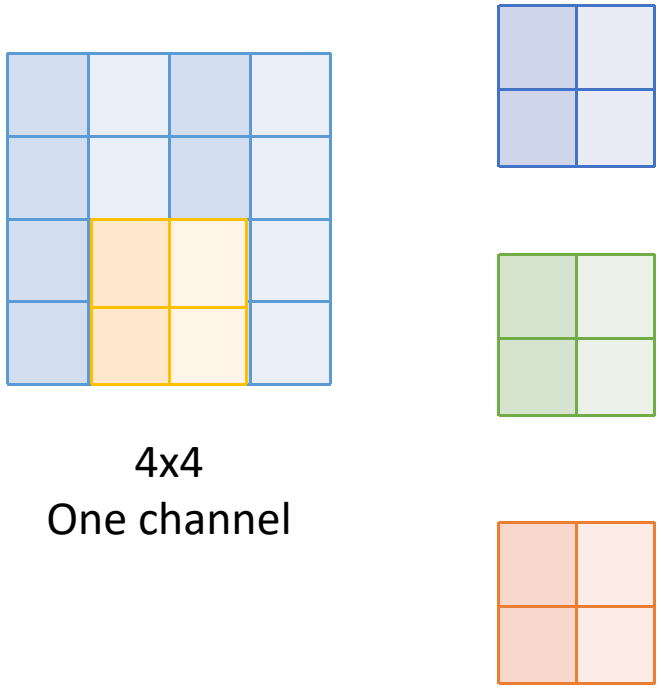
Convolution Stage



4x4
One channel

a	b	c	d
e	f	g	h
i	j	k	l
m	n		

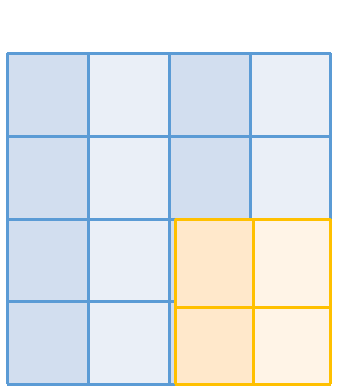
Convolution Stage



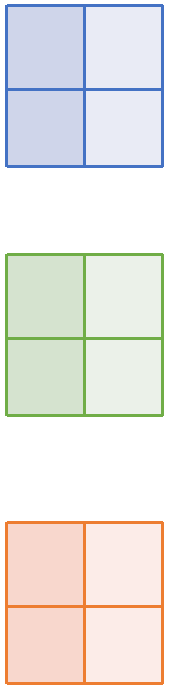
4x4
One channel

a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	

Convolution Stage

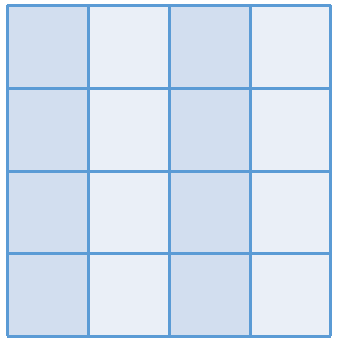


4x4
One channel

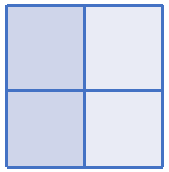


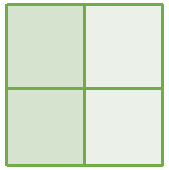
a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	o

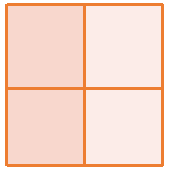
Convolution Stage



4x4
One channel

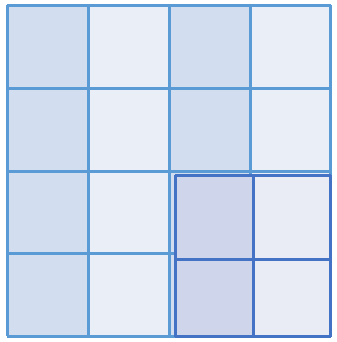




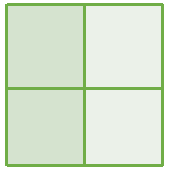


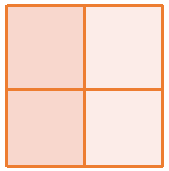
a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	o

Convolution Stage



4x4
One channel

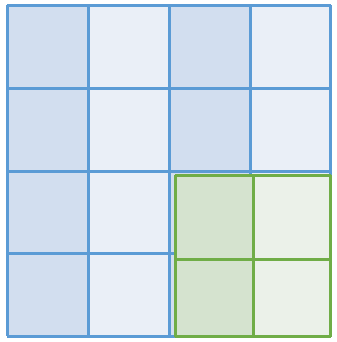




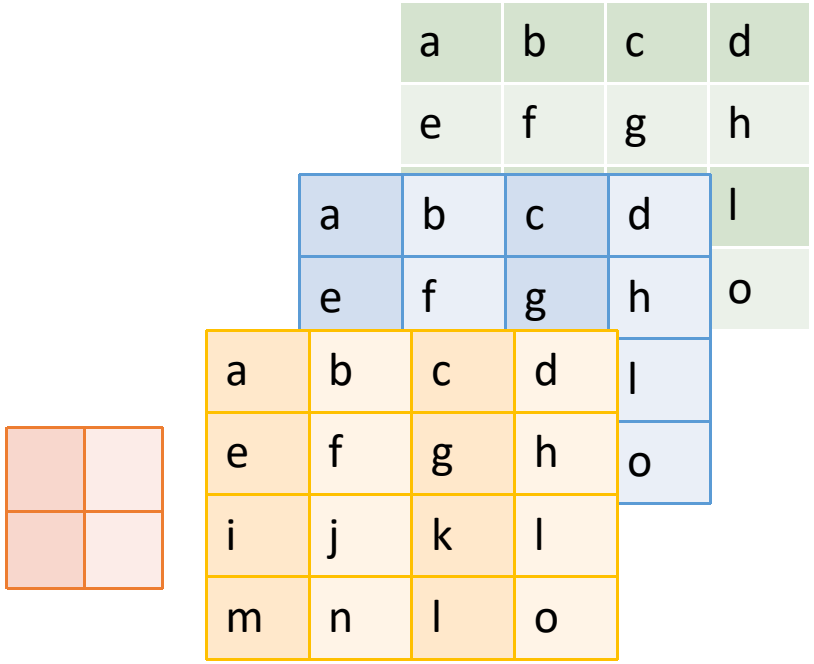
a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	o

a	b	c	d
e	f	g	h
			l
			o

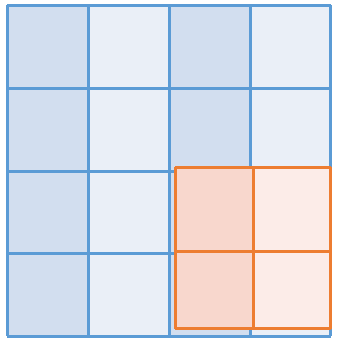
Convolution Stage



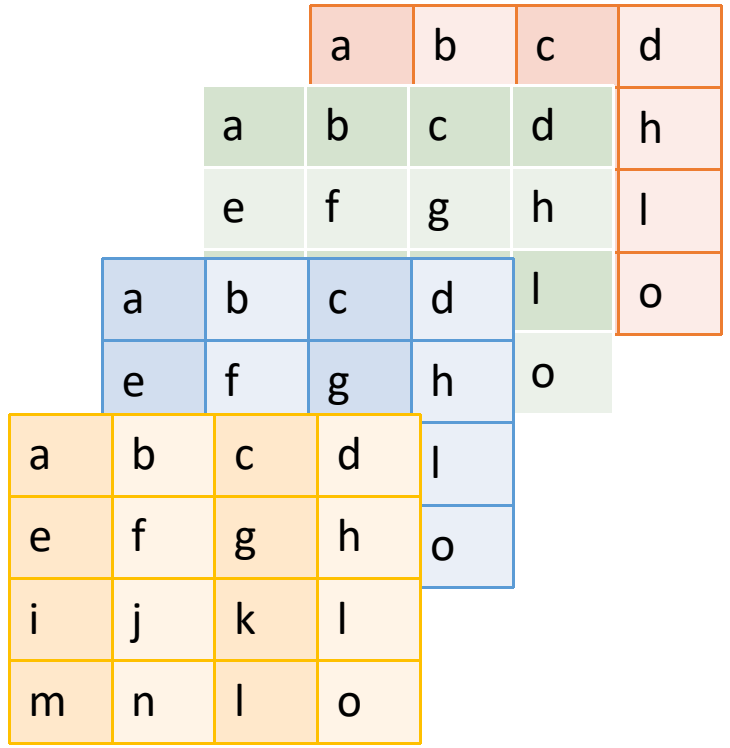
4x4
One channel



Convolution Stage

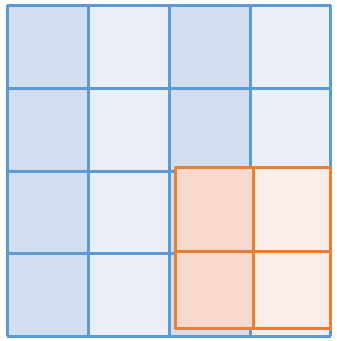


4x4
One channel

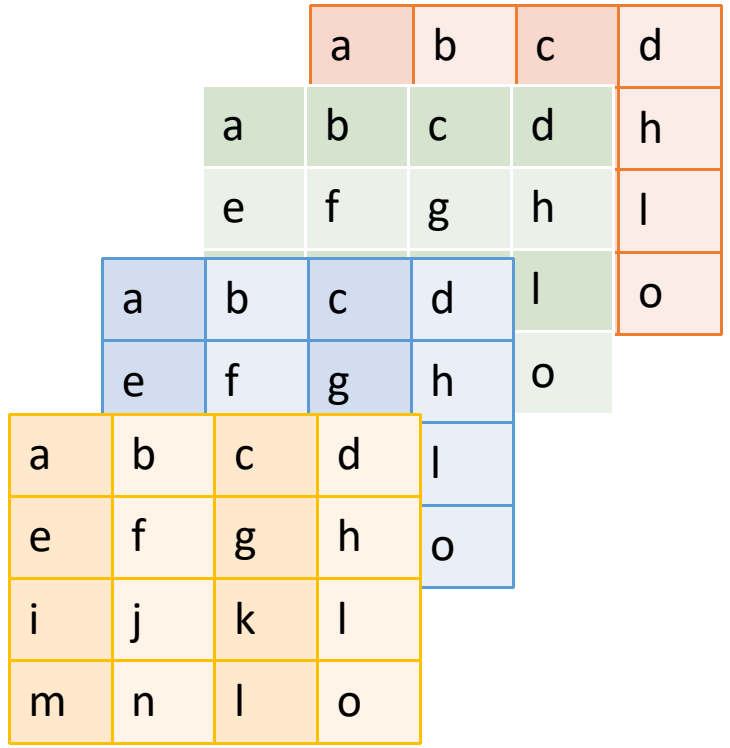


4X4x4

Convolution Stage

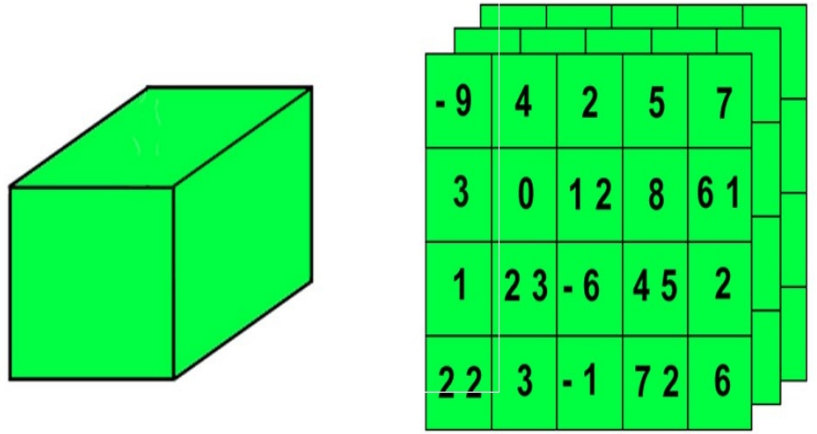


4x4
One channel

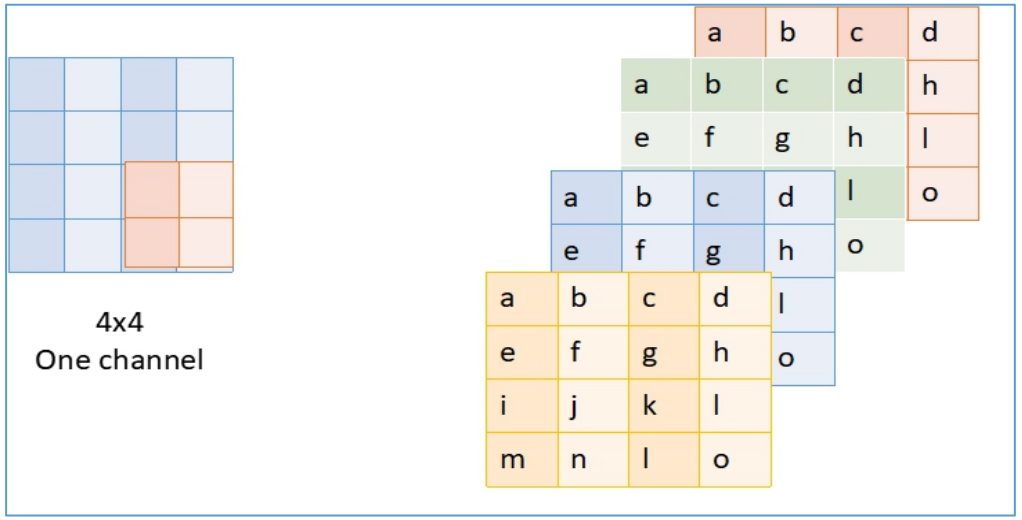


4X4x4

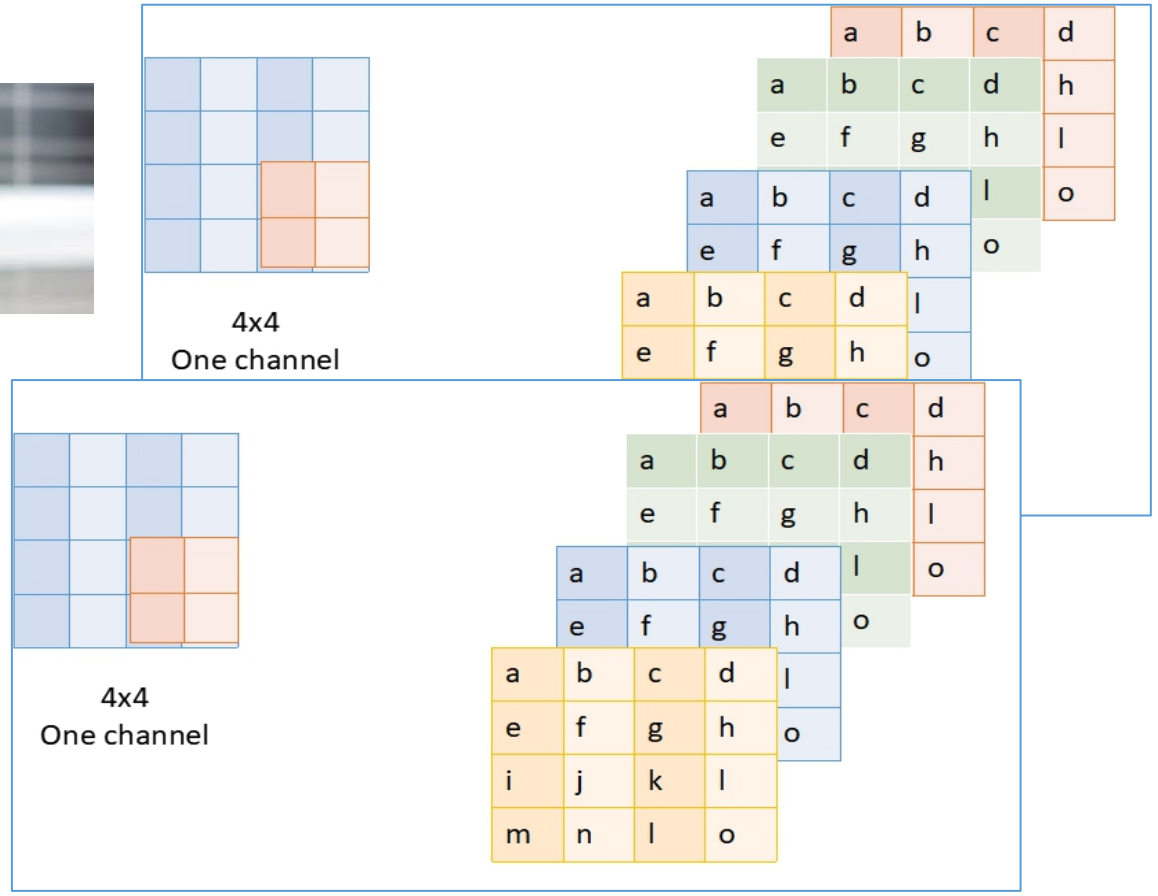
3D TENSOR/
CUBE



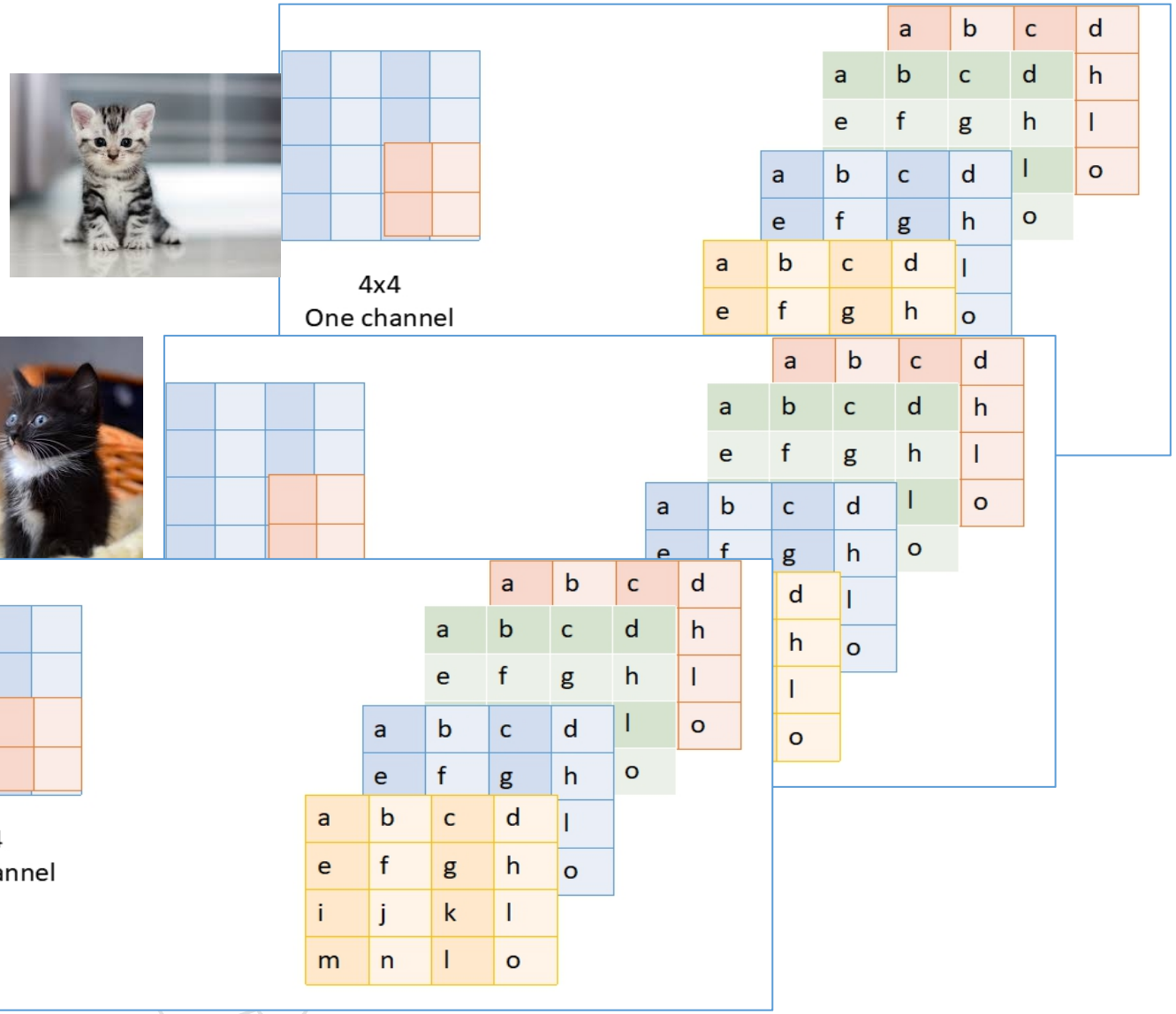
Convolution Stage



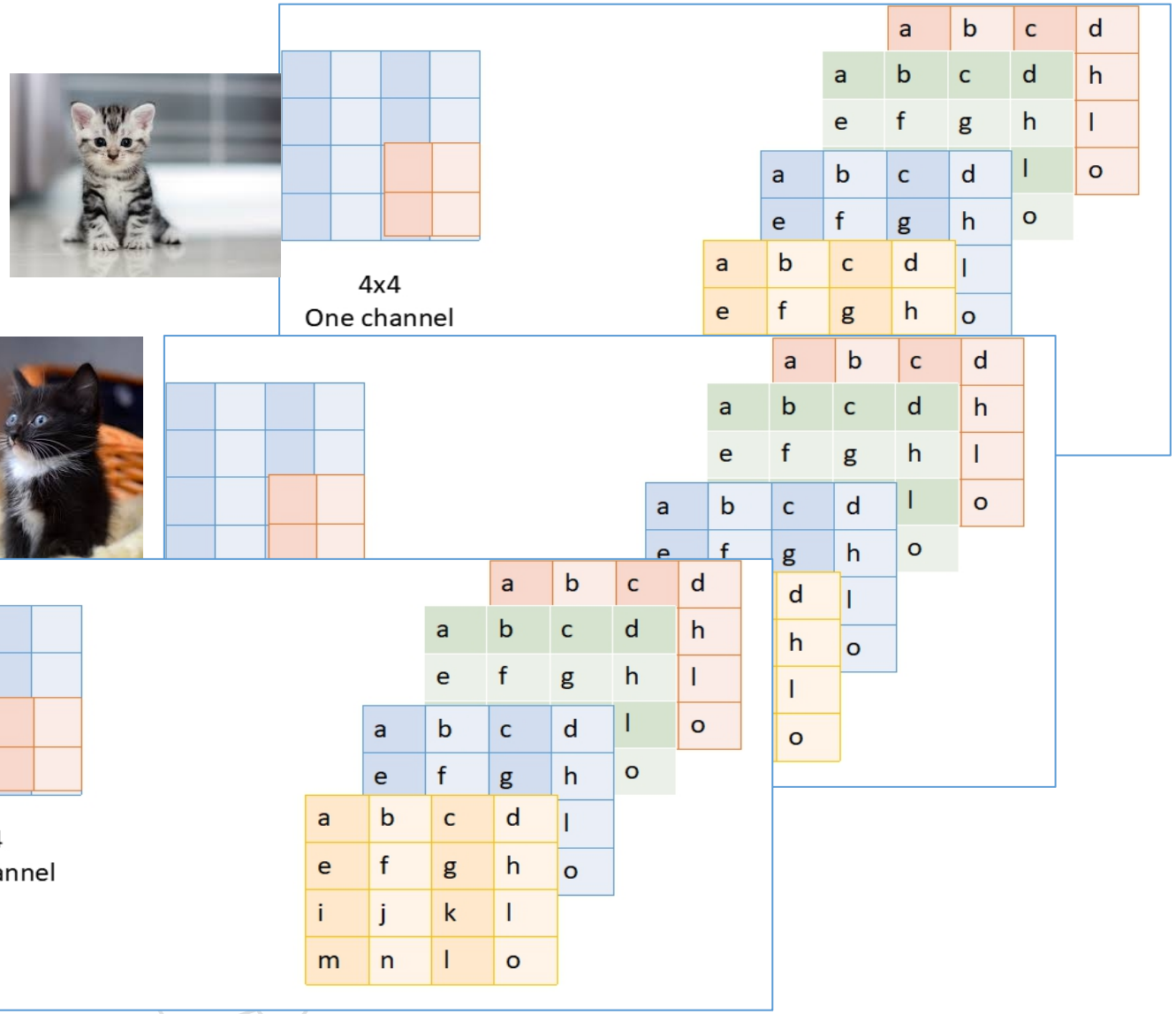
Convolution Stage

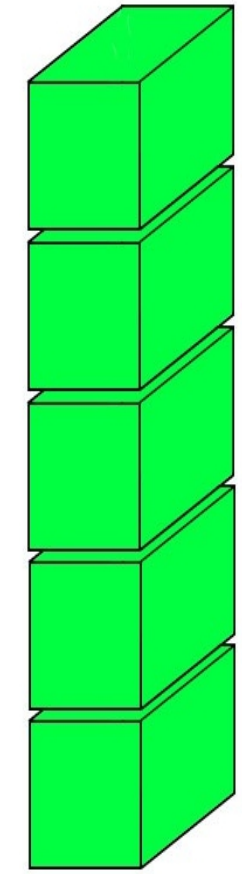


Convolution Stage



Convolution Stage

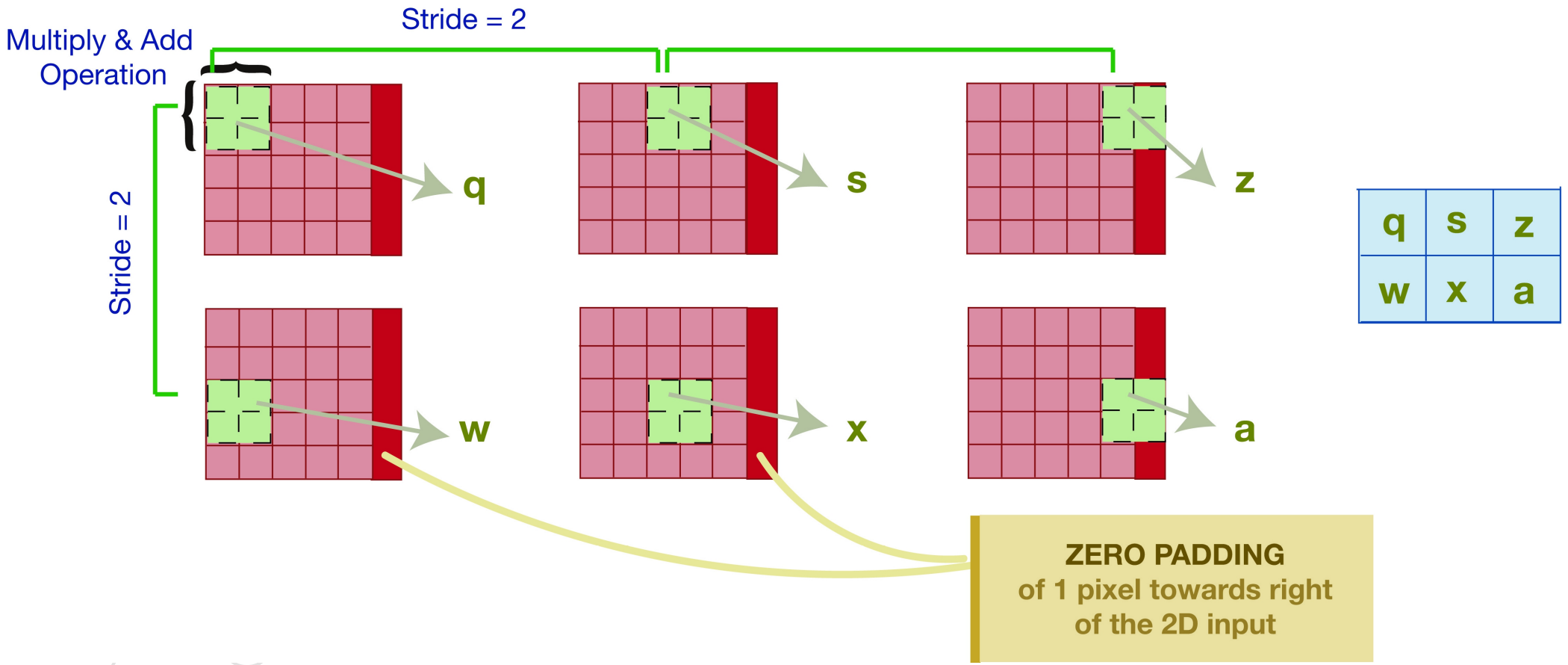




4D TENSOR

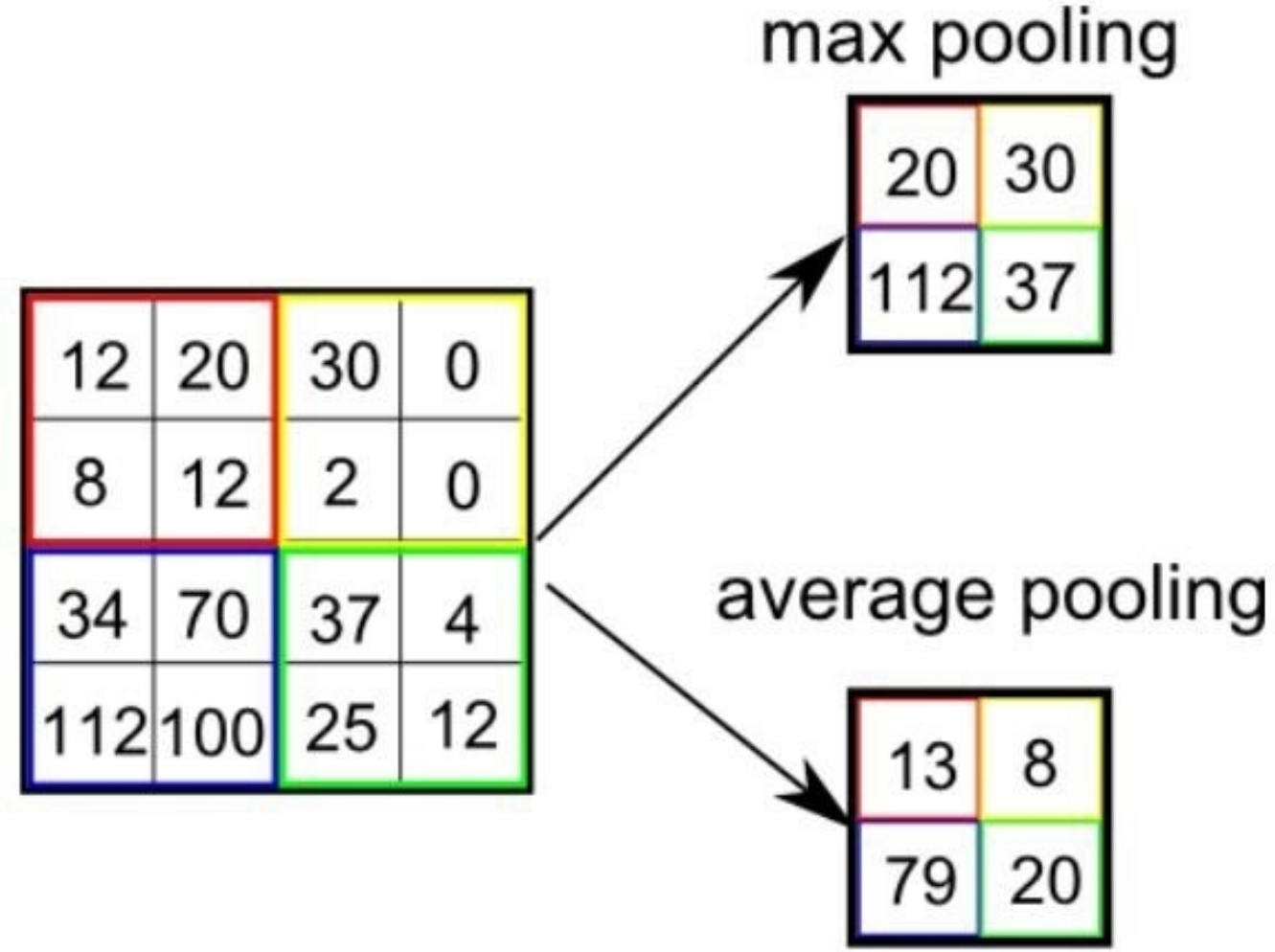
VECTOR OF CUBES

Stride and Padding

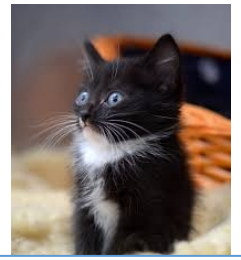


q	s	z
w	x	a

Pooling Stage



Pooling Stage

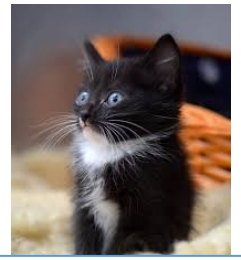


4x4
One channel

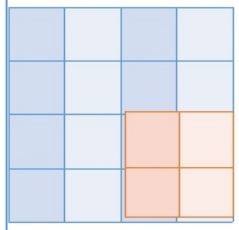
4x4
One channel

a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	o

Pooling Stage

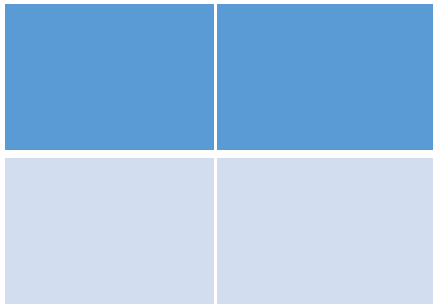


4x4
One channel

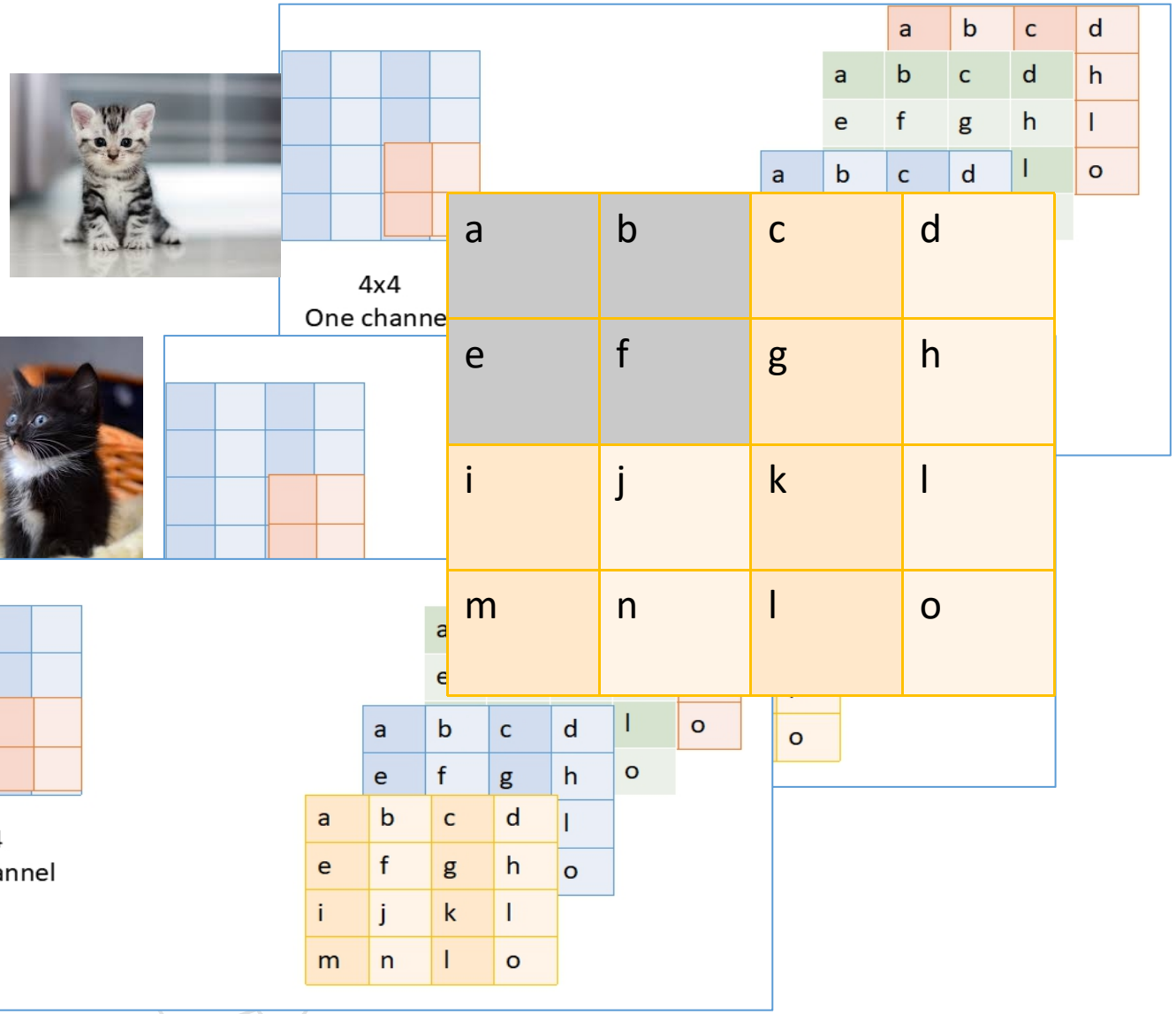


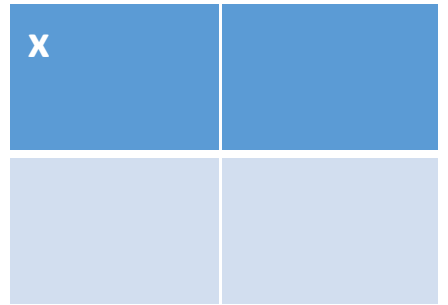
4x4
One channe

a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	o

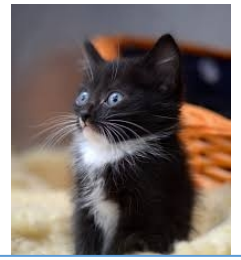


Pooling Stage





Pooling Stage



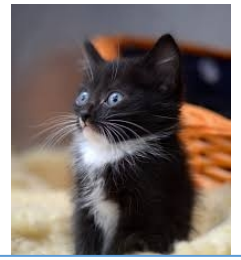
4x4
One channel

4x4
One channe

a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	o

x	y

Pooling Stage

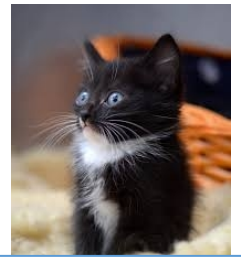


4x4
One channel

a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	o

x	y
z	

Pooling Stage



4x4
One channel

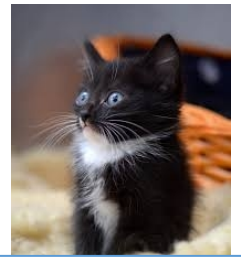
a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	o

x	y
z	w

Pooling Stage

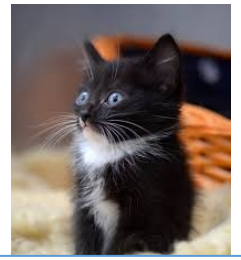
x	y
z	w

4x4
One channel



4x4
One channel

Pooling Stage



4x4
One channel

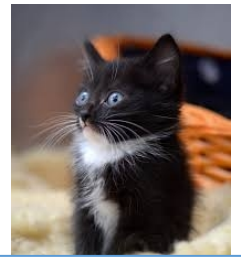
4x4
One channel

a	b	c	d
e	f	g	h
i	j	k	l
m	n	l	o

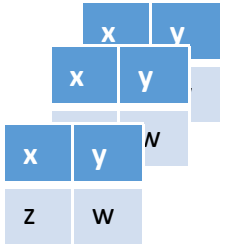
x	y
z	w

Pooling Stage

4x4
One channel

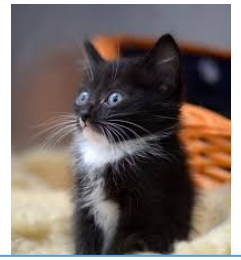


4x4
One channel

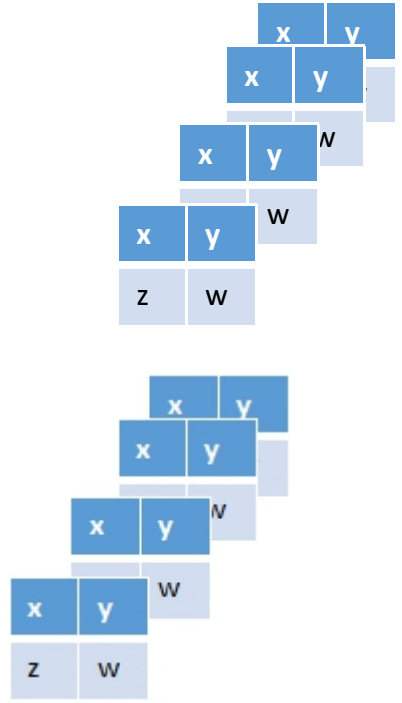


Pooling Stage

4x4
One channel

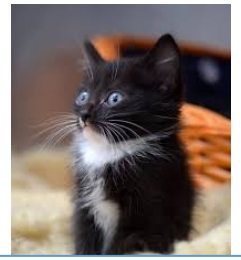


4x4
One channel



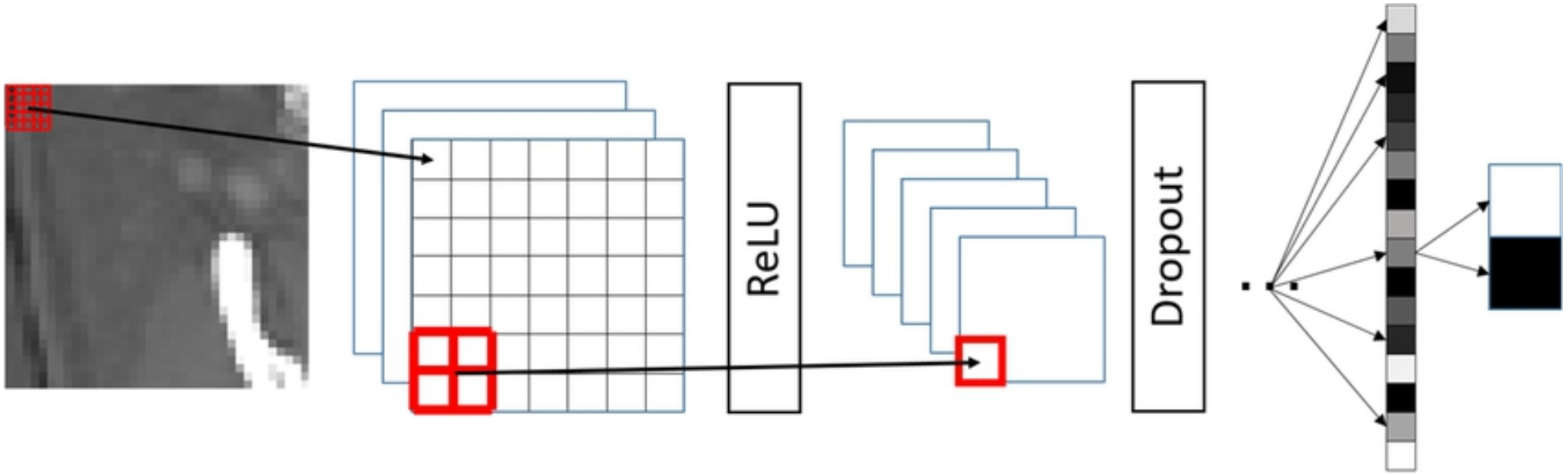
Pooling Stage

4x4
One channel

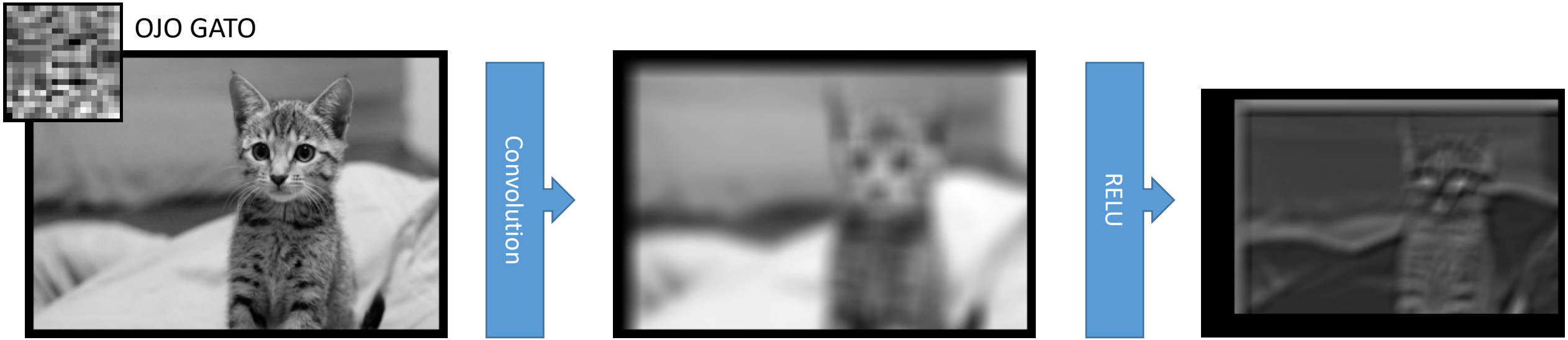


4x4
One channel

Summary Stages



Convolution and Relu



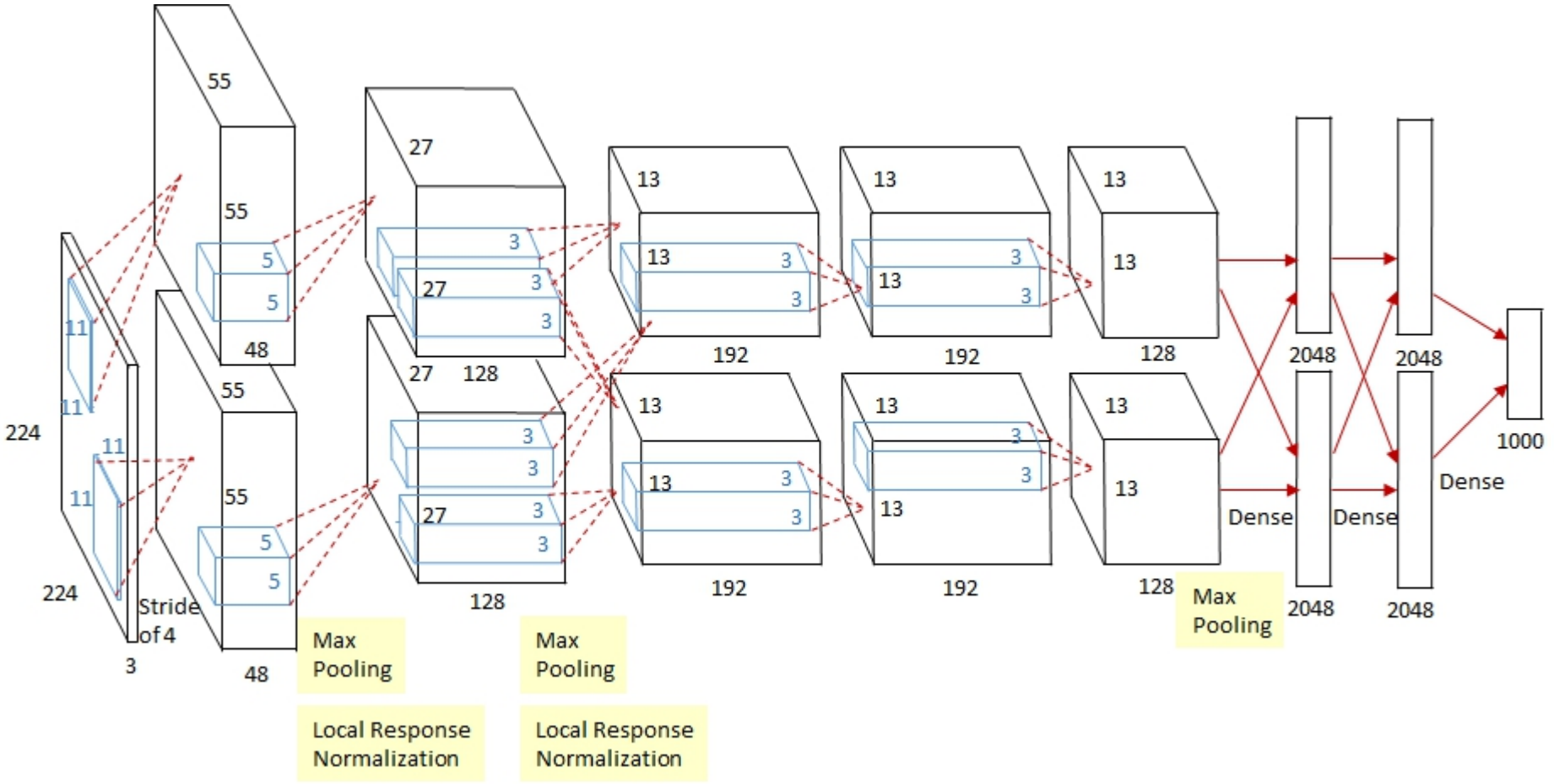
Exercise 1

- Download the script *Deep_Convolution_Layer_Relu*
- Review the effects of ReLU function after a convolution is applied.
- Note that this operation highligths several properties of the image

Exercise 2

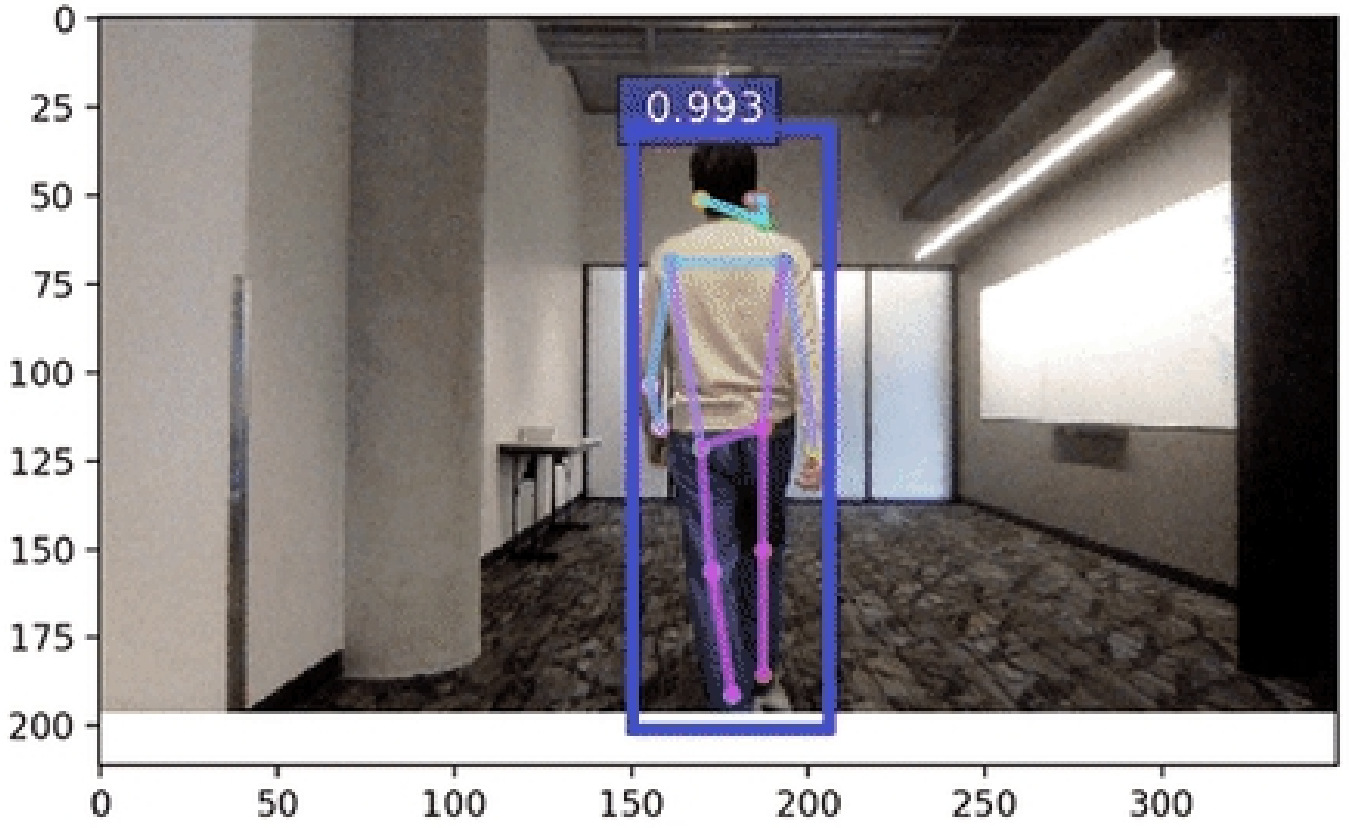
- Download the script *DeepLearning_ConvolutionNetwork*
- Review the process to create a convolution network using the dataset included in file miml_dataset.zip

AlexNet Architecture



Exercise 3

- Convolution Netwo Application
- Pose Estimation



Cinvestav
Tamaulipas

Recurrent Neural Networks

Dr. Edwyn Javier Aldana Bobadilla

Recurrent Neural Networks

- Recurrent Neural Networks or RNN are a very important variant of neural networks heavily used in Natural Language Processing.
- In traditional neural network, an input is processed through a number of layers and an output is produced, with an assumption that two successive inputs are independent of each other.
- This assumption is not true in many real-life scenarios: price of a stock at a given time, the n^{th} word in a sequence of words.
- The above examples are common application of RNNs.

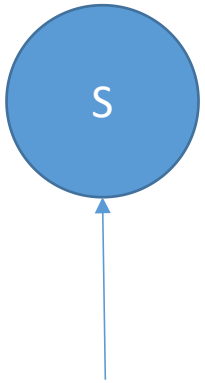
Recurrent Neural Networks

- RNNs are recurrent because they carry out the same task for each element of a sequence.
- It is said that a RNN has a memory which captures information about what has been calculated previously.
-

RNN Architecture

$$s = state$$

RNN Architecture

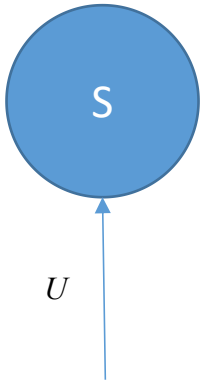


x

$$s = state$$

$$x = input$$

RNN Architecture



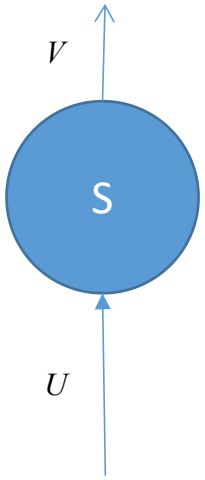
x

$s = state$

$x = input$

$U = \text{weights\_value}$

RNN Architecture



x

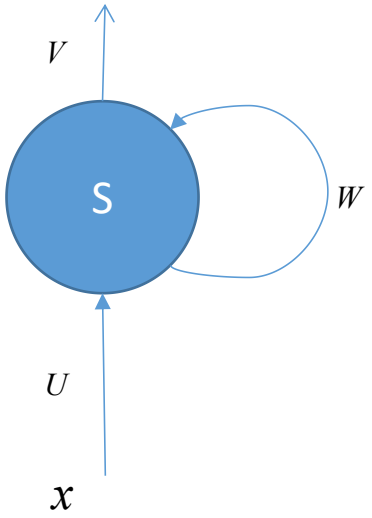
$s = state$

$x = input$

$U = \text{weights vector}$

$V = \text{output value}$

RNN Architecture



$s = state$

$x = input$

$U = \text{weights vectors}$

$V, W = \text{output values}$

RNN Architecture

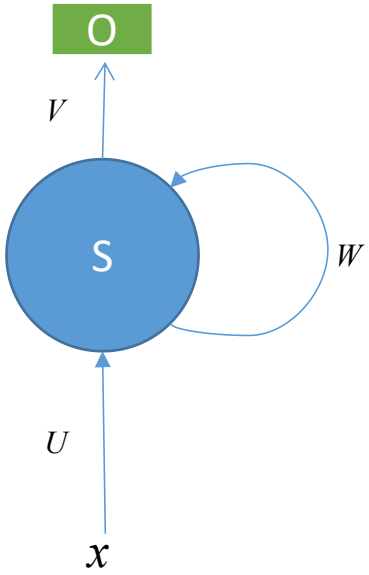
$s = state$

$x = input$

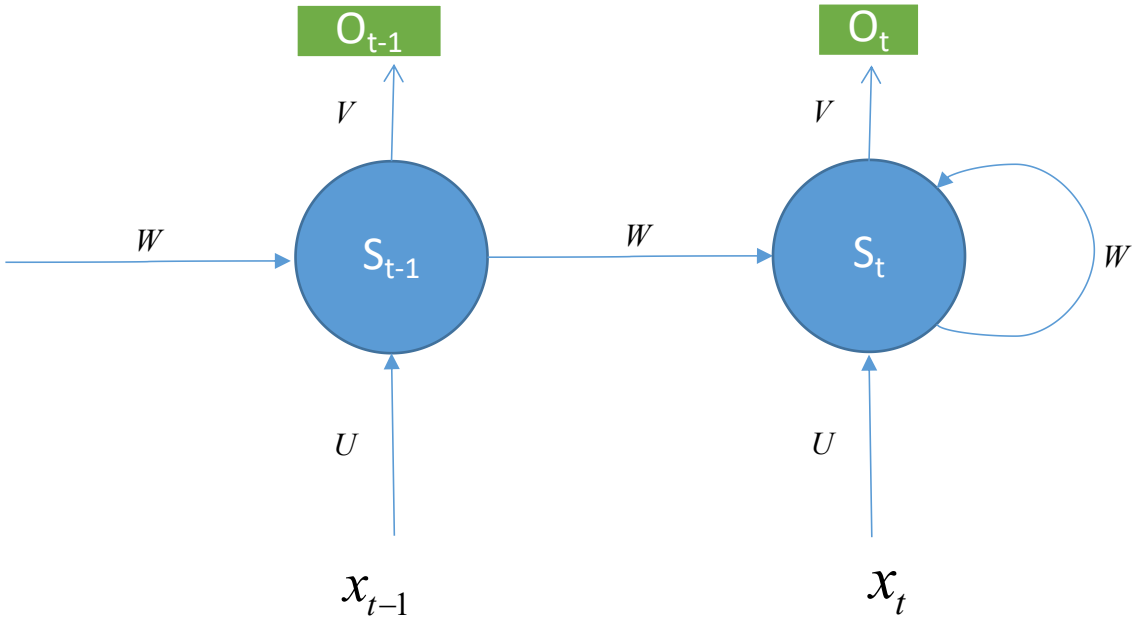
$U = \text{weights vectors}$

$V, W = \text{output values}$

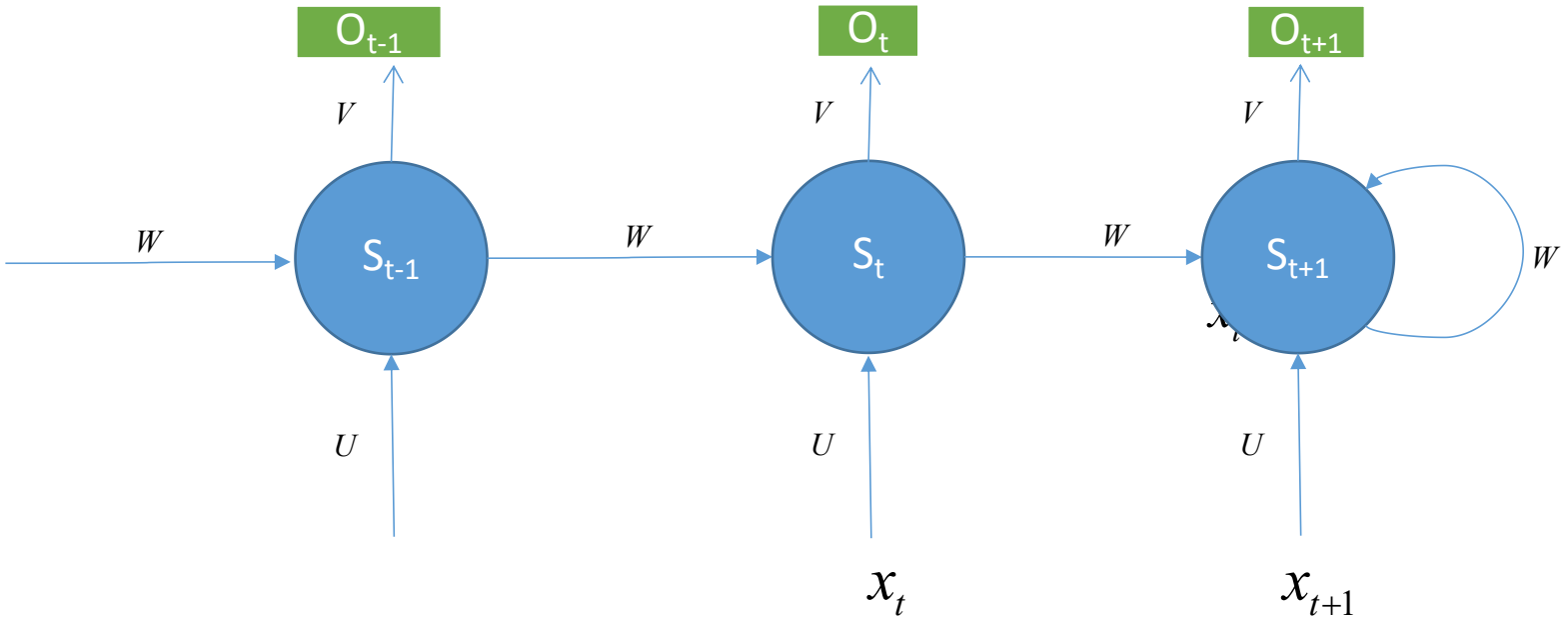
$\text{O} = \text{Final Output}$



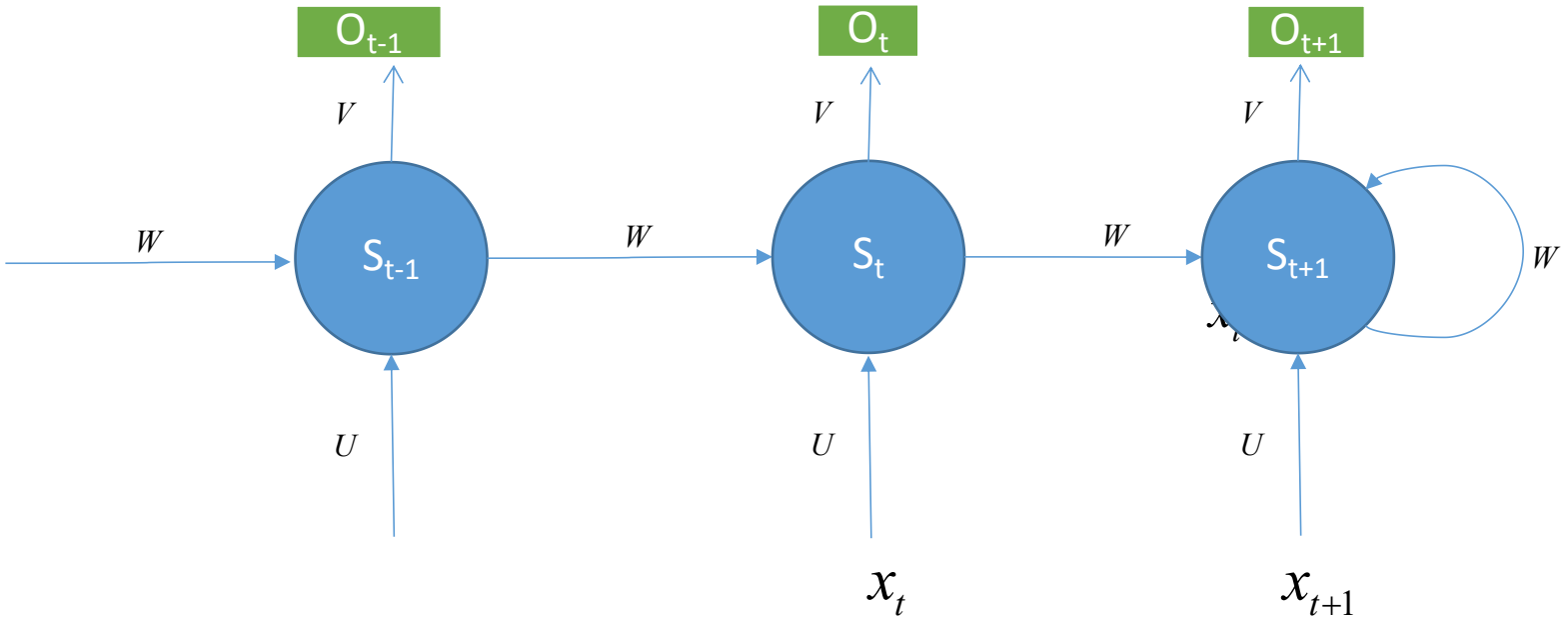
Unfolding a RNN



Unfolding a RNN

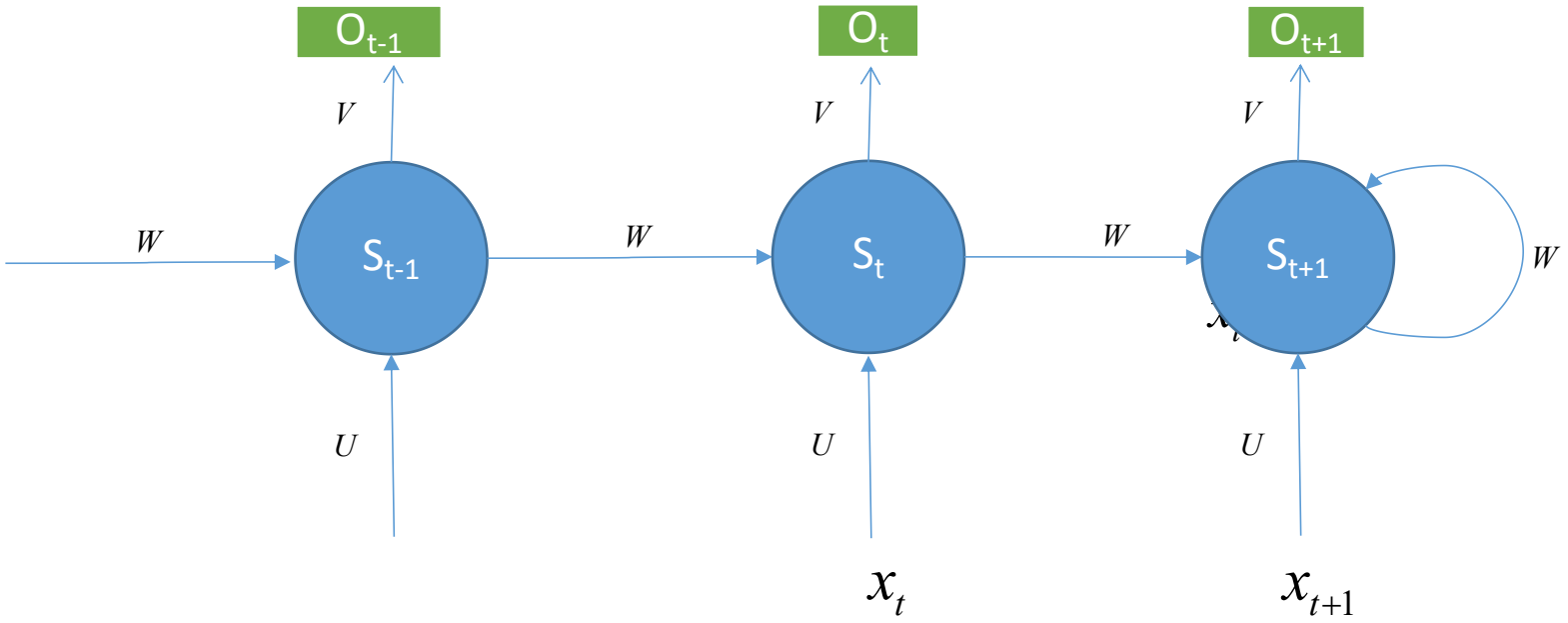


Example of a RNN



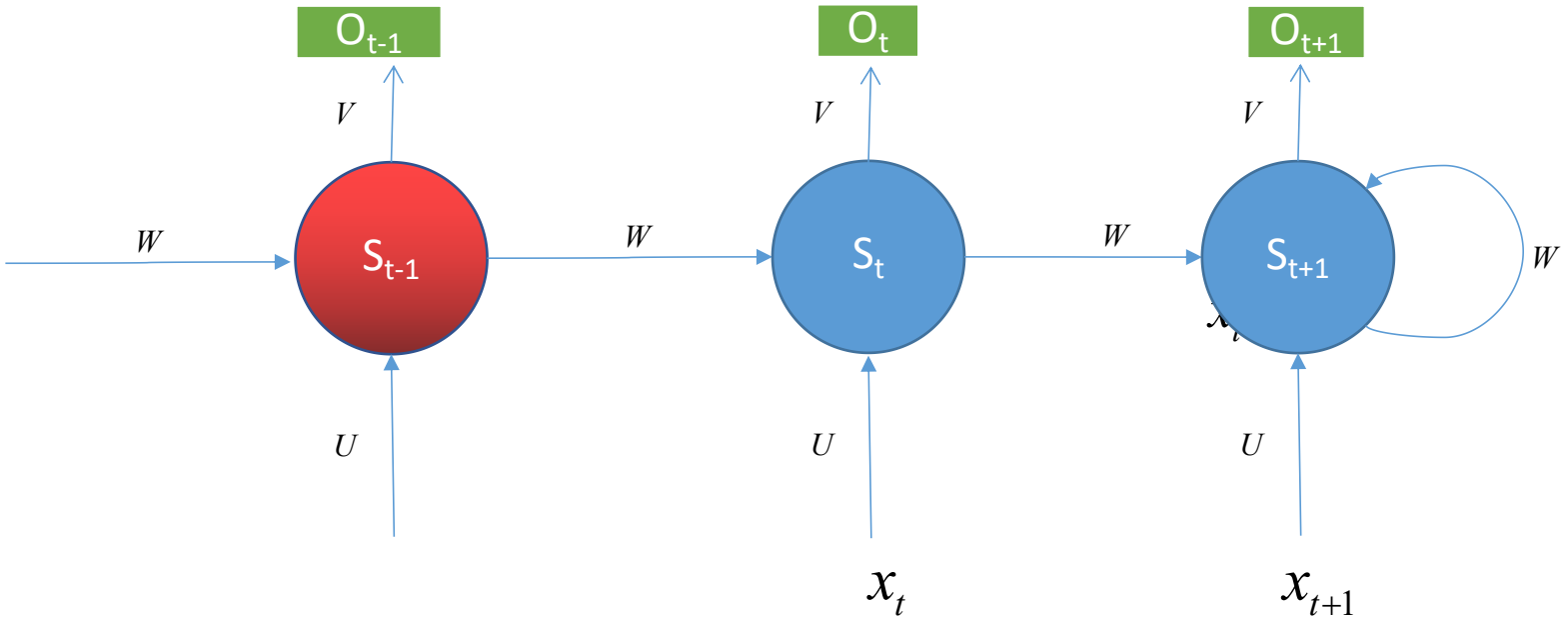
1000	0100	0001

Example of a RNN



1000	0100	0001
Un	viejo	hidalgo

Example of a RNN

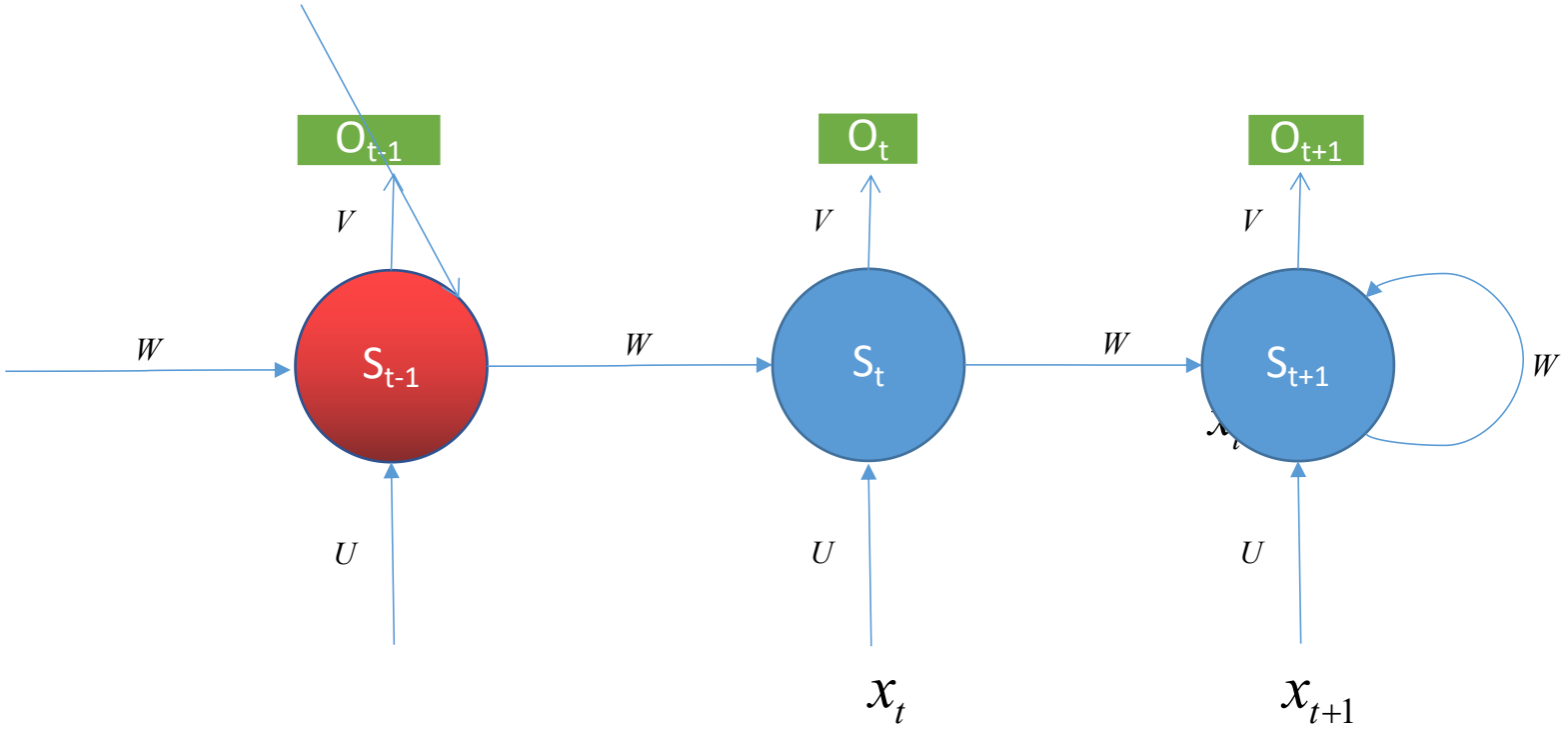


1000	0100	0001
Un	viejo	hidalgo

Example of a RNN

$$s_{t-1} = f(U_{t-1}x_{t-1} + W_{t-1})$$

The function f is commonly tanh or ReLU

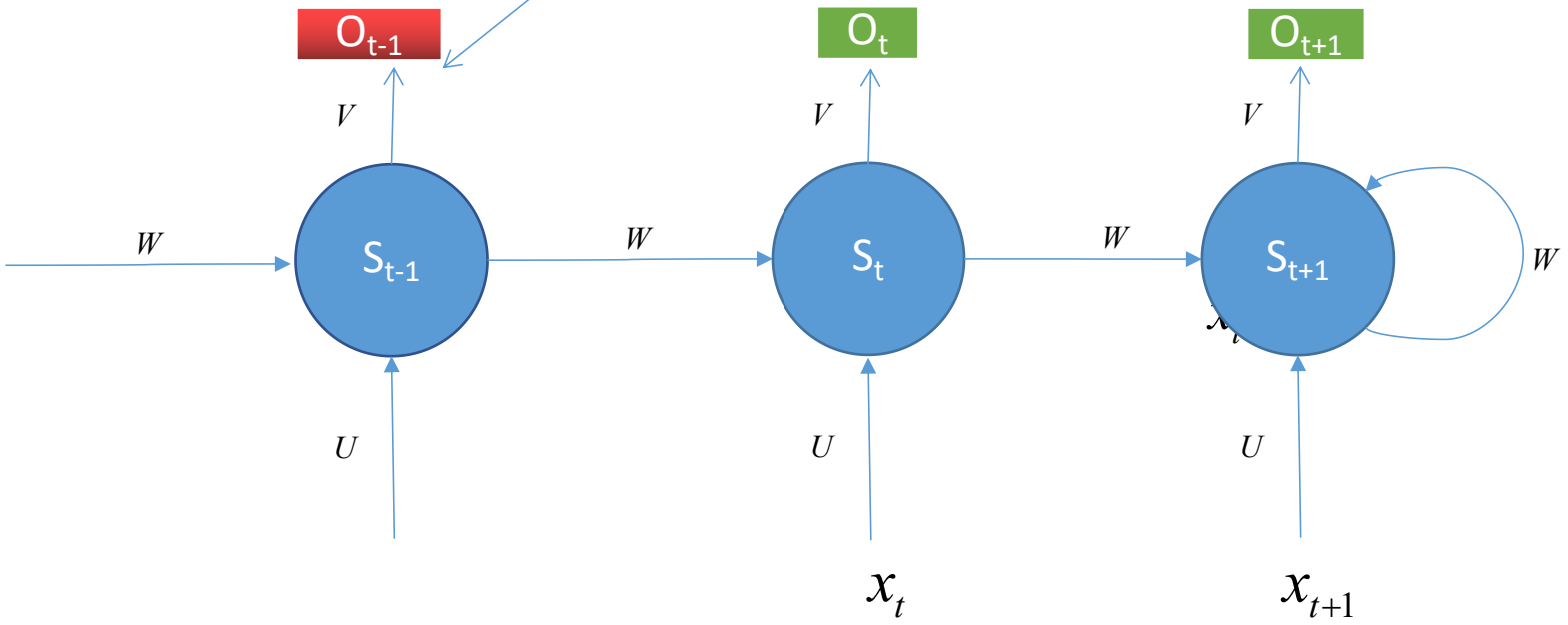


1000	0100	0001
Un	viejo	hidalgo

Example of a RNN

$$o_{t-1} = soft\max(V_{t-1})$$

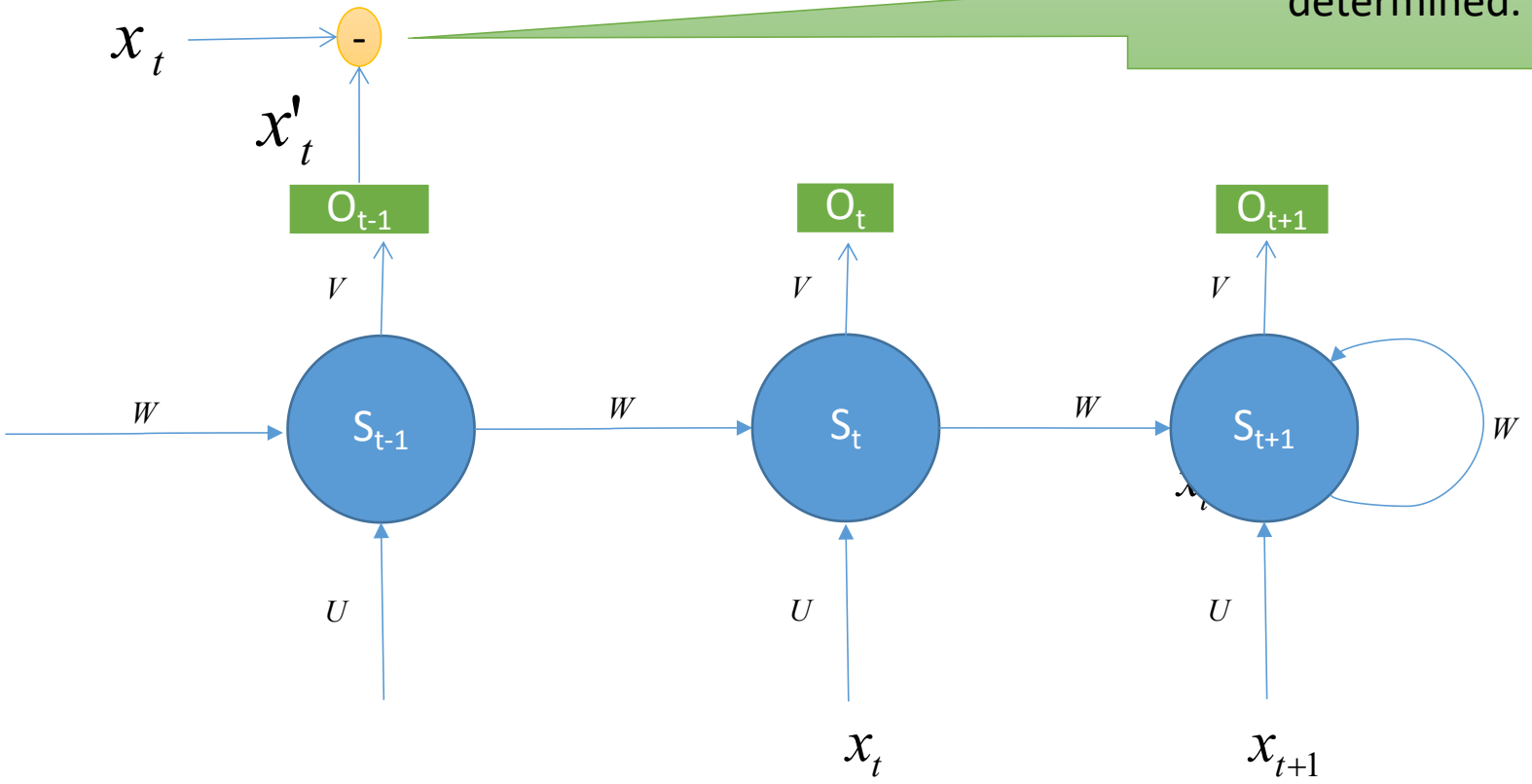
The softmax function allows us to find the probability of the next word given the actual word ($p(x_{t+1}|x_t)$)



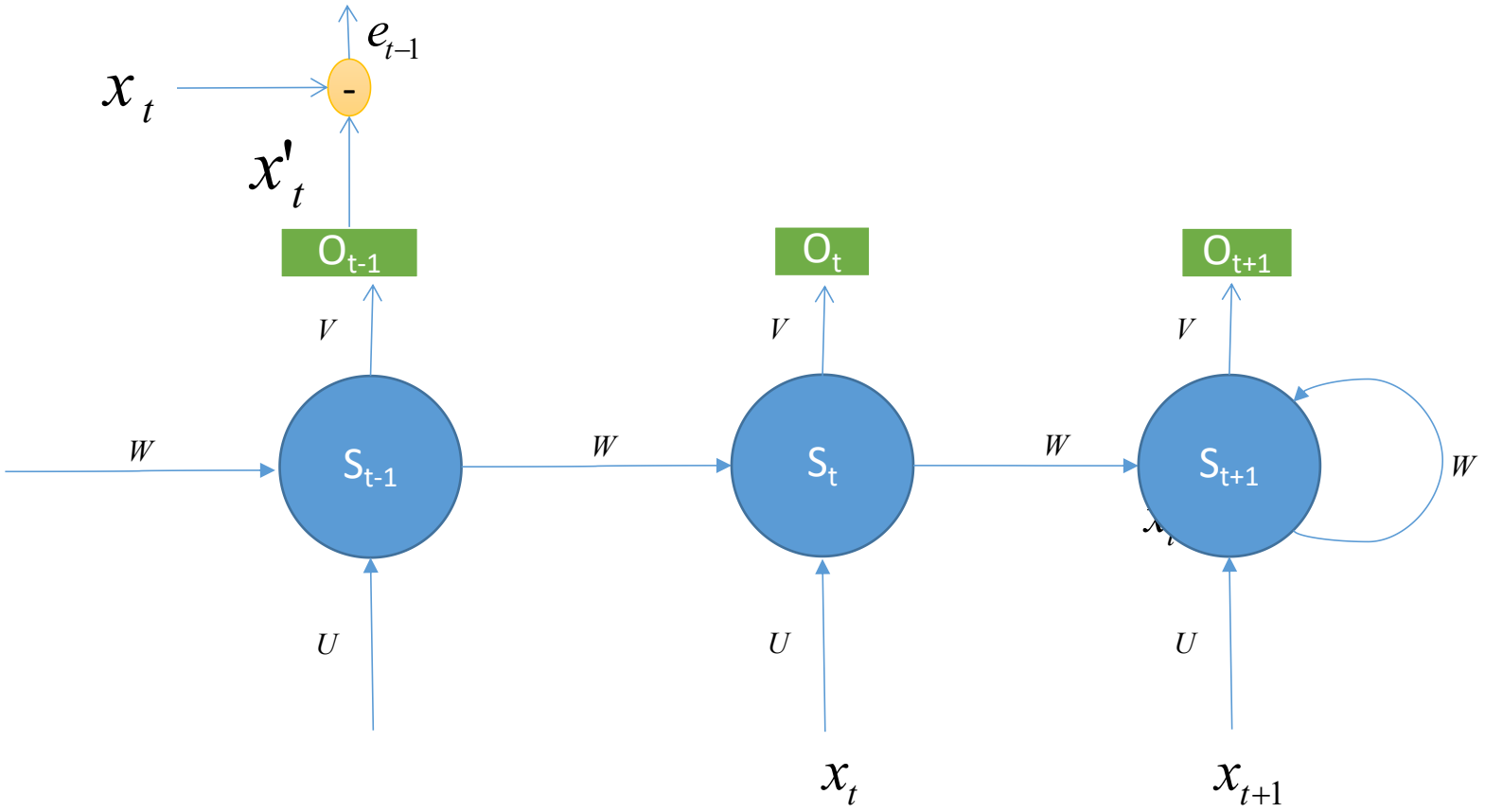
1000	0100	0001
Un	viejo	hidalgo

Training Process

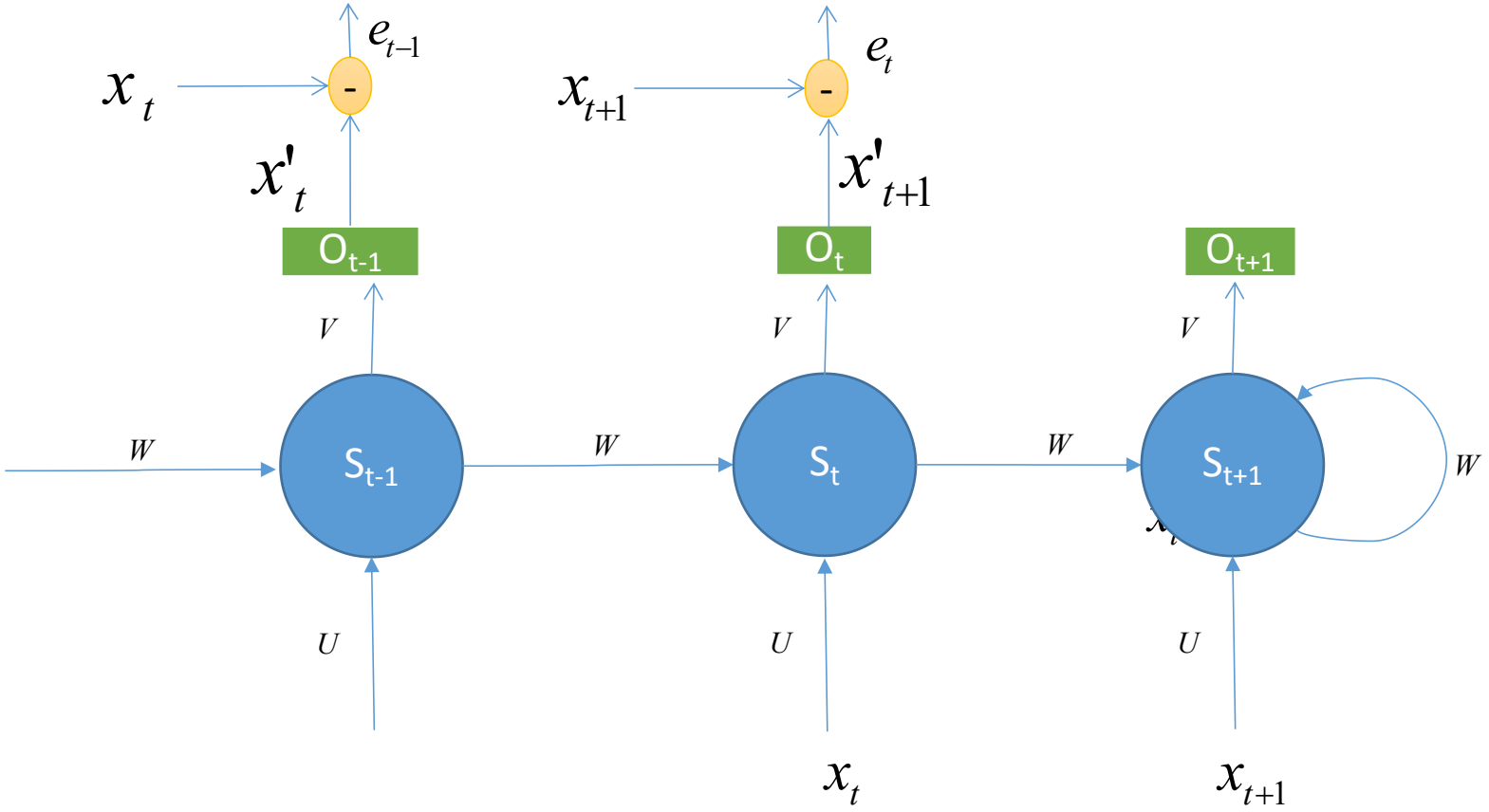
An **error function** can be determined.



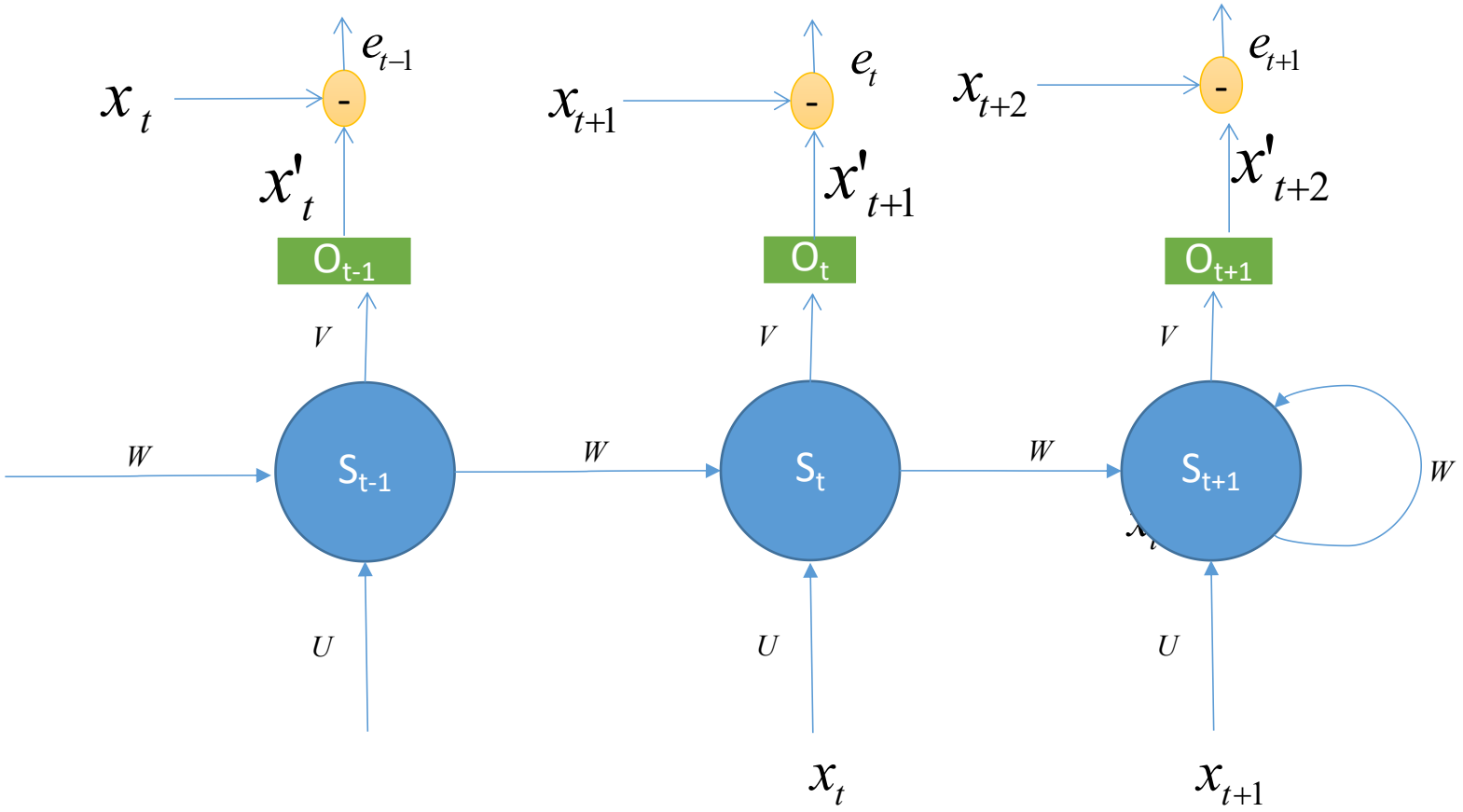
Training Process



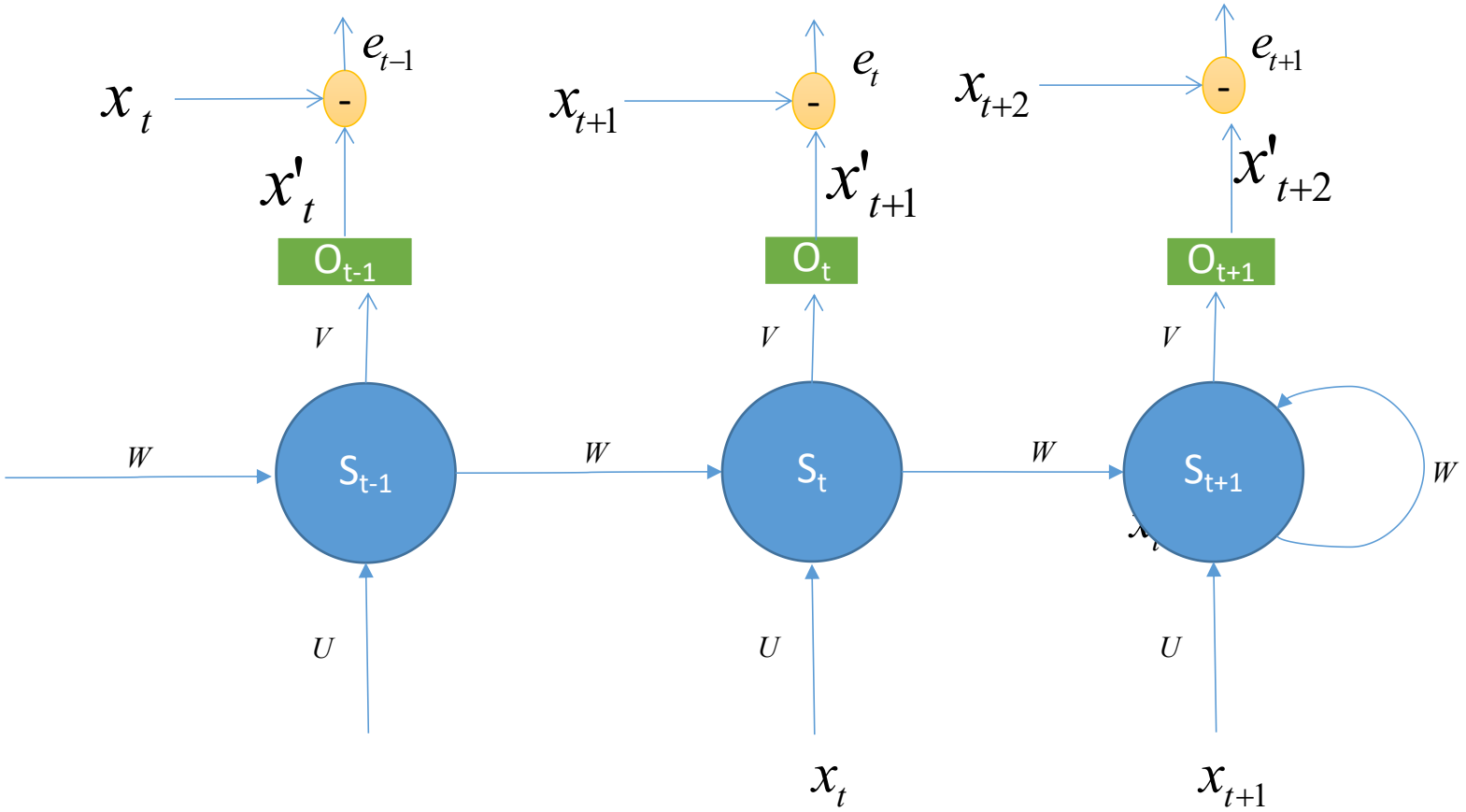
Training Process



Training Process

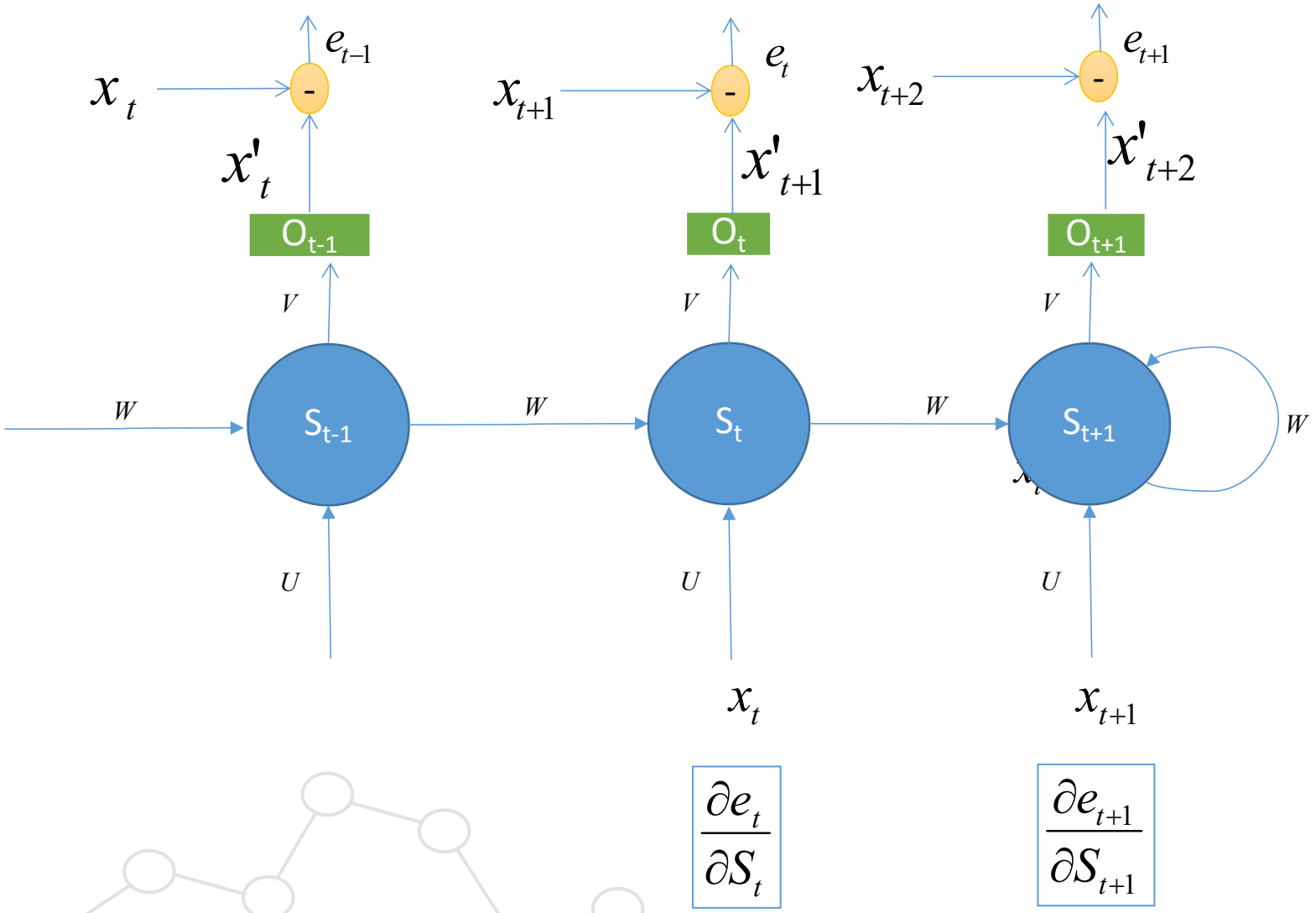


Training Process



$$\frac{\partial e_{t+1}}{\partial S_{t+1}}$$

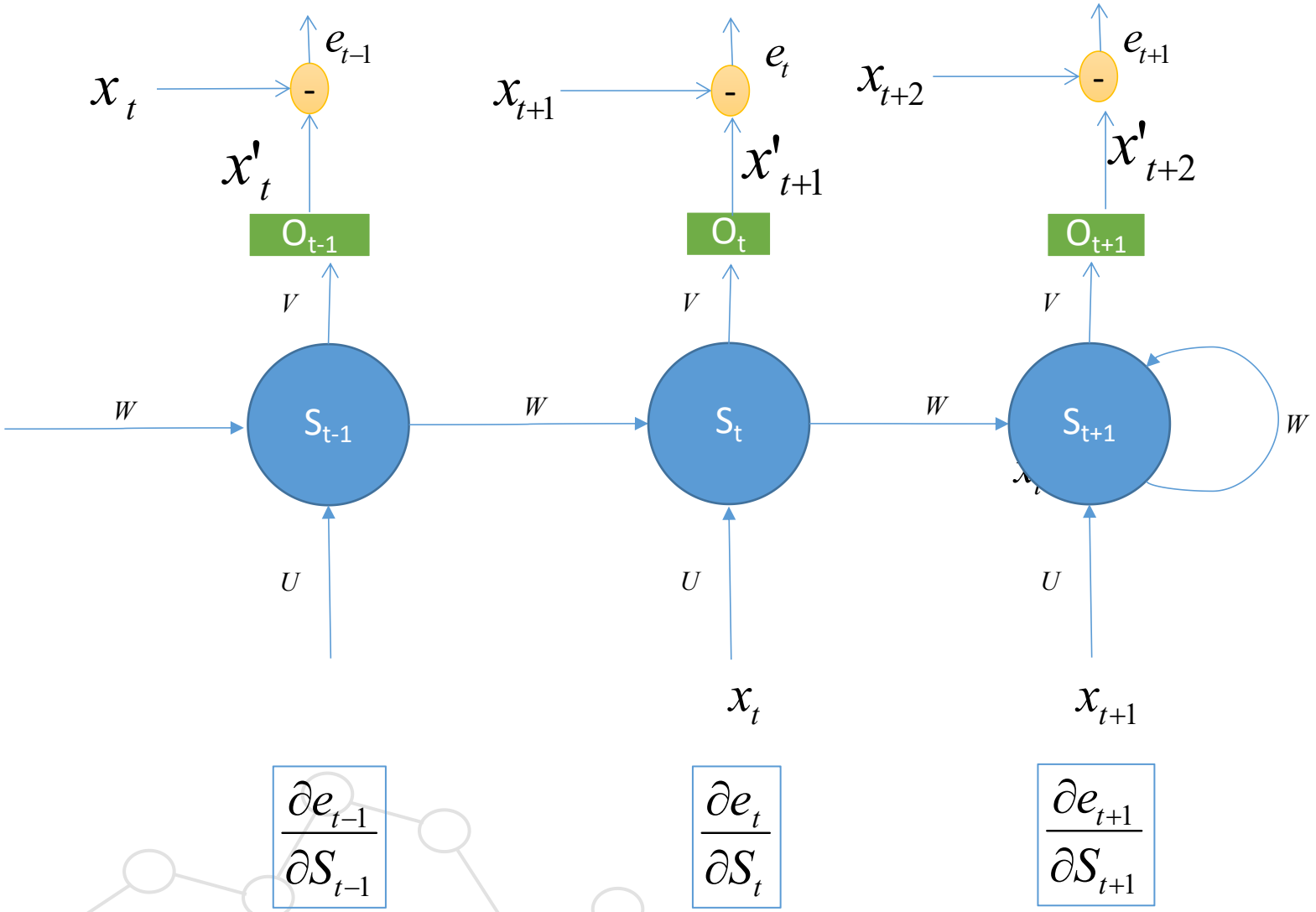
Training Process



$$\frac{\partial e_t}{\partial S_t}$$

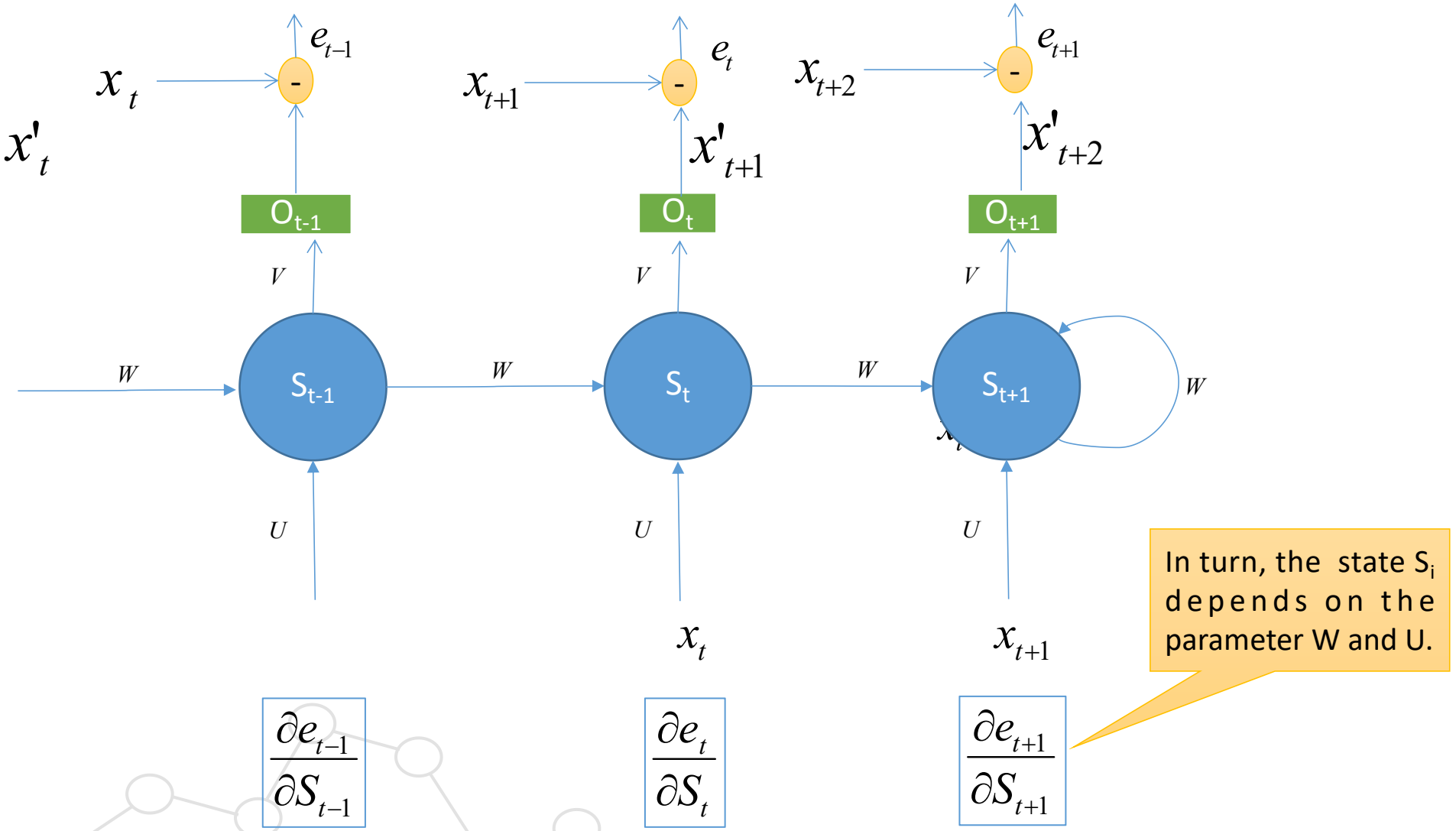
$$\frac{\partial e_{t+1}}{\partial S_{t+1}}$$

Training Process



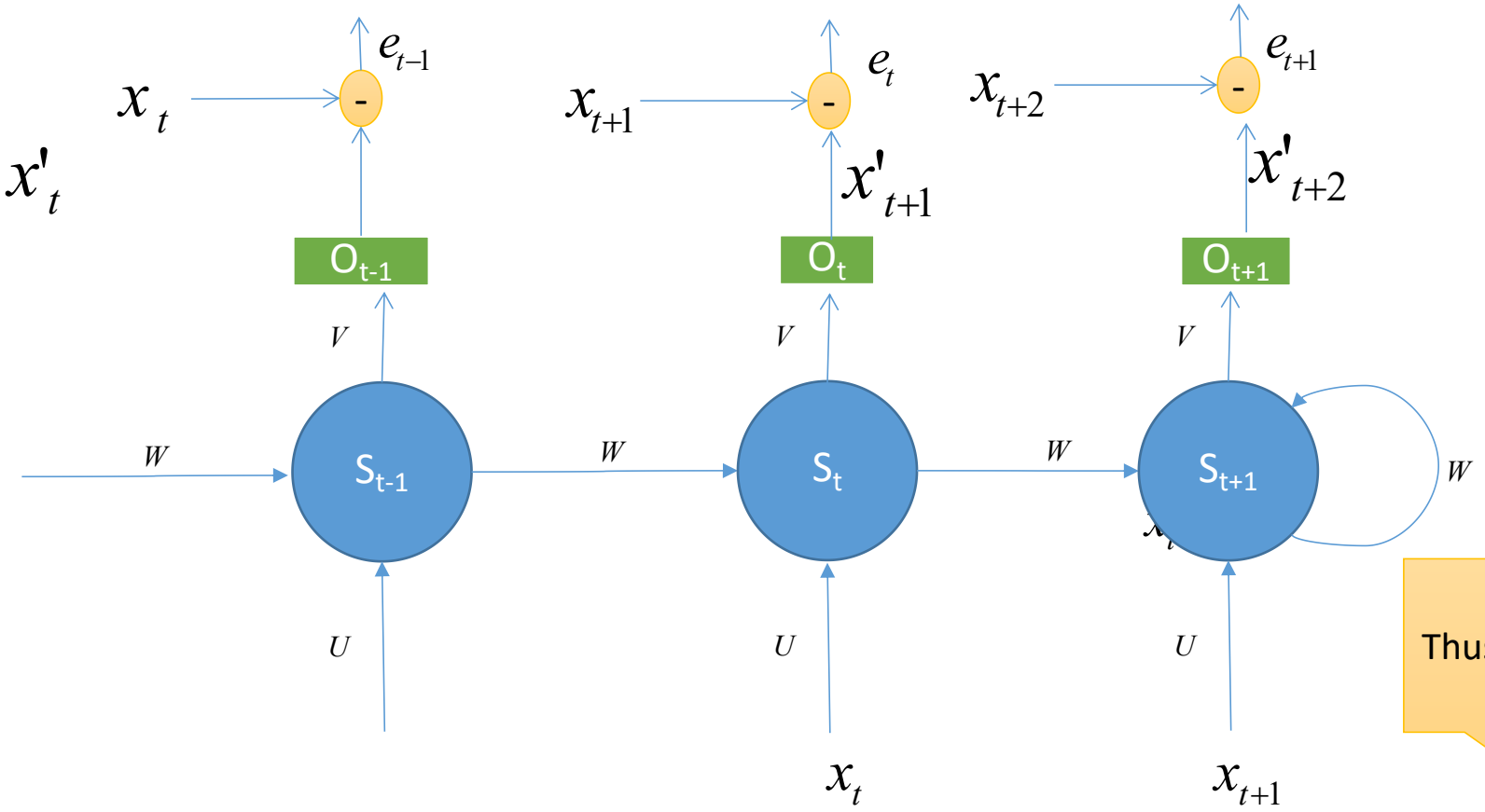
$$\frac{\partial e_{t-1}}{\partial S_{t-1}} \qquad \frac{\partial e_t}{\partial S_t} \qquad \frac{\partial e_{t+1}}{\partial S_{t+1}}$$

Training Process



In turn, the state S_i depends on the parameter W and U.

Training Process

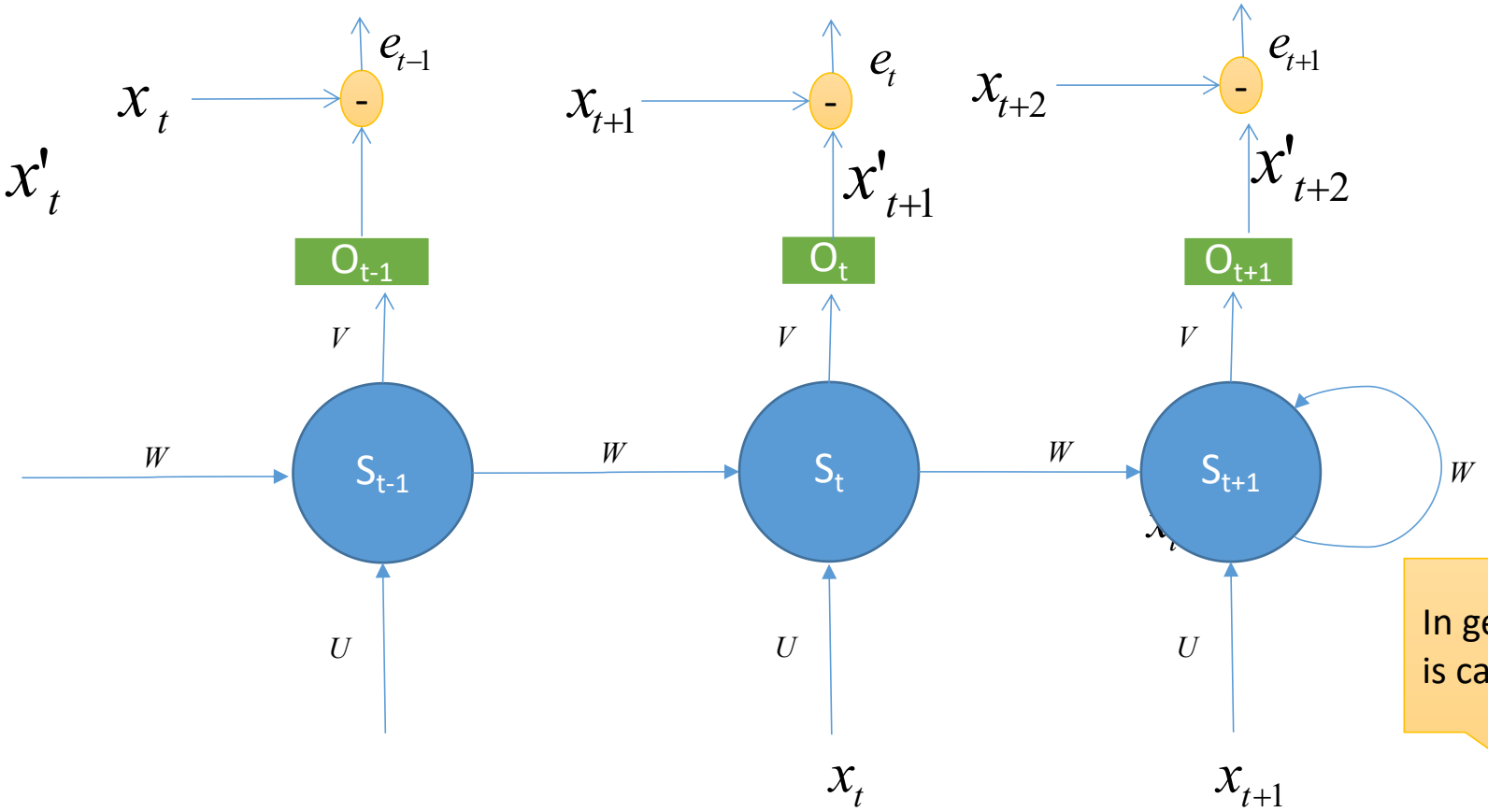


Thus,

$$\frac{\partial e_{t-1}}{\partial S_{t-1}} \qquad \frac{\partial e_t}{\partial S_t} \qquad \frac{\partial e_{t+1}}{\partial S_{t+1}}$$

$$\frac{\partial e_t}{\partial [W,U]} = \frac{\partial e_t}{\partial \theta}$$

Training Process



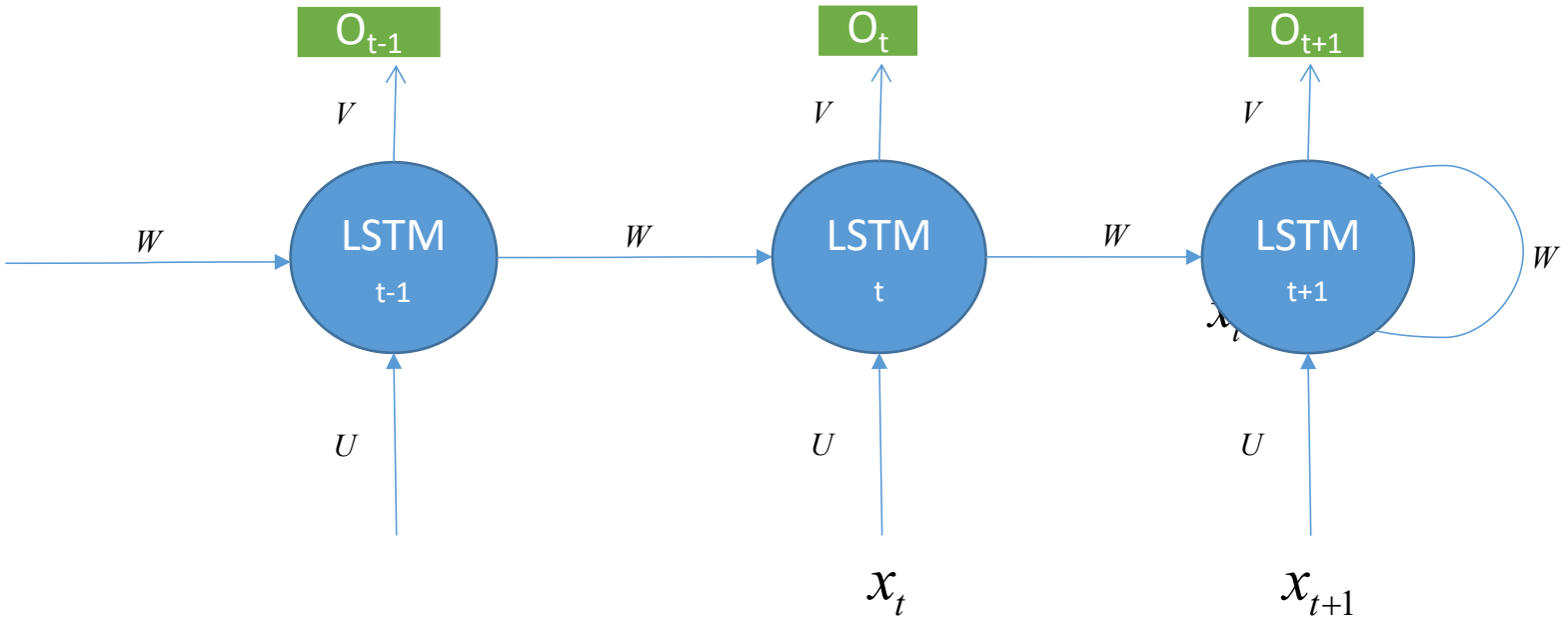
In general, the error is calculated as:

$$\frac{\partial e_{t-1}}{\partial S_{t-1}} \qquad \frac{\partial e_t}{\partial S_t} \qquad \frac{\partial e_{t+1}}{\partial S_{t+1}}$$

$$\frac{\partial E}{\partial \theta} = \sum_{1 \le t \le T} \frac{\partial e_t}{\partial \theta}$$

Backpropagation through time BPTT

Long-Short Term Memory



Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero

Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Size of n-gram=3

Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Size of n-gram=3

5	451	321	8

Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Size of n-gram=3

5	451	321	8
451	321	8	321

Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Size of n-gram=3

5	451	321	8
451	321	8	321
321	8	321	8

Labeled dataset

Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Size of n-gram=3

5	451	321	8
451	321	8	321
321	8	321	8

N= number of sequences

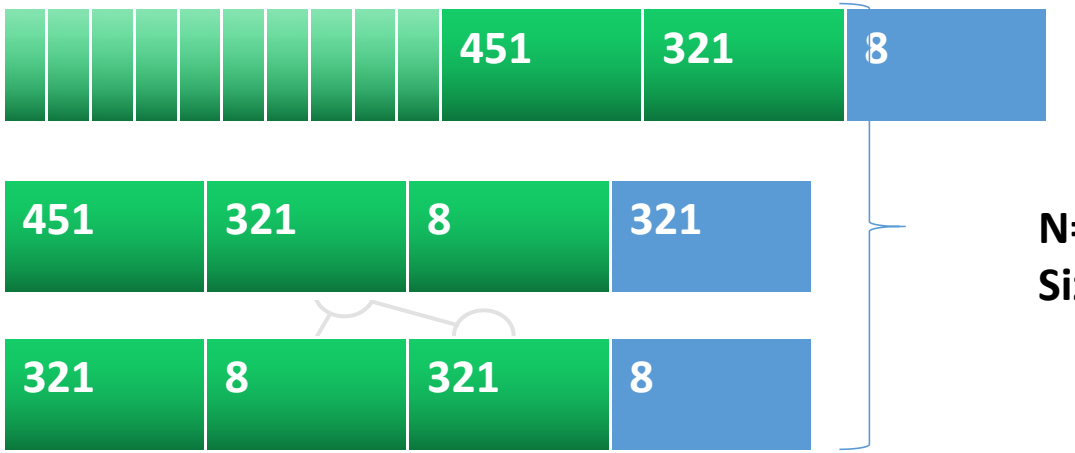
Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Size of n-gram=3



N= number of sequences
Size of window=10

Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Size of n-gram=3

451	321	8	321
321	8	321	8

N= number of sequences
Size of window=10

Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Size of n-gram=3

451	321	8	321
321	8	321	8

N= number of sequences
Size of window=10

Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

Indices

Size of n-gram=3

321	8	321	8

N= number of sequences
Size of window=10

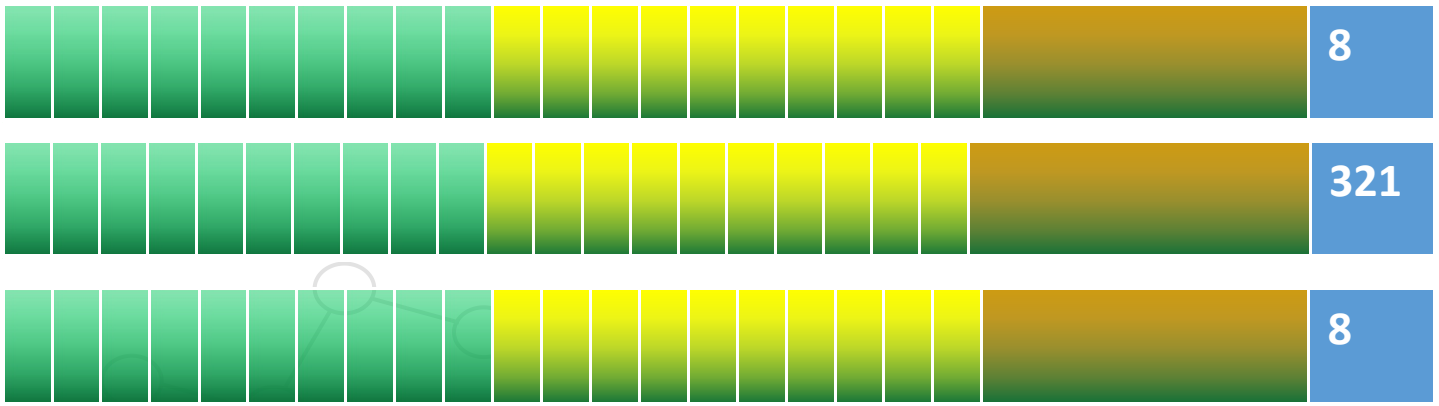
Recurrent Neural Networks-Example

Un viejo hidalgo de los de lanza en astillero ...

Un	viejo	hidalgo	de	los	de	lanza	en	astillero	...
5	451	321	8	321	8	212	10	85	...

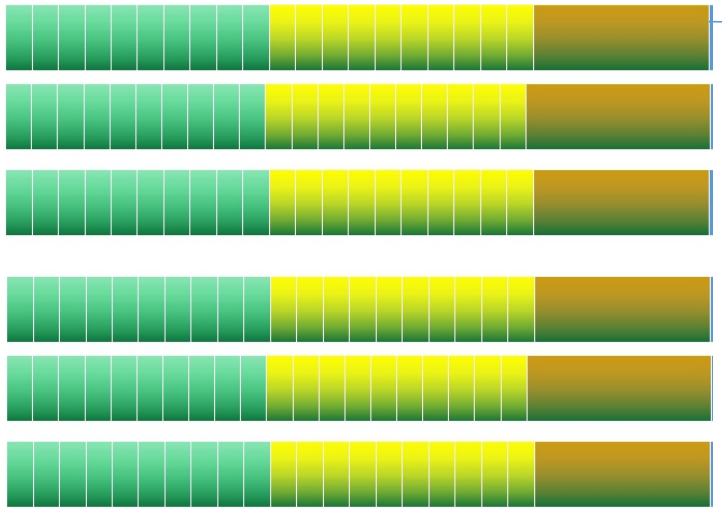
Indices

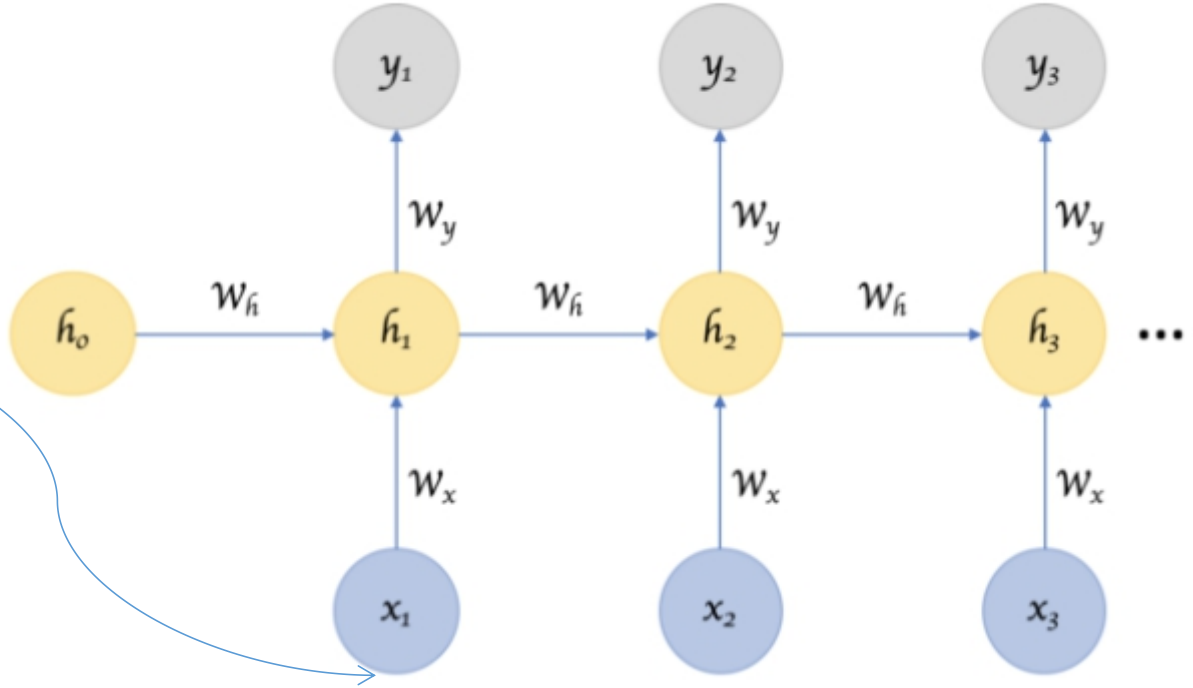
Size of n-gram=3



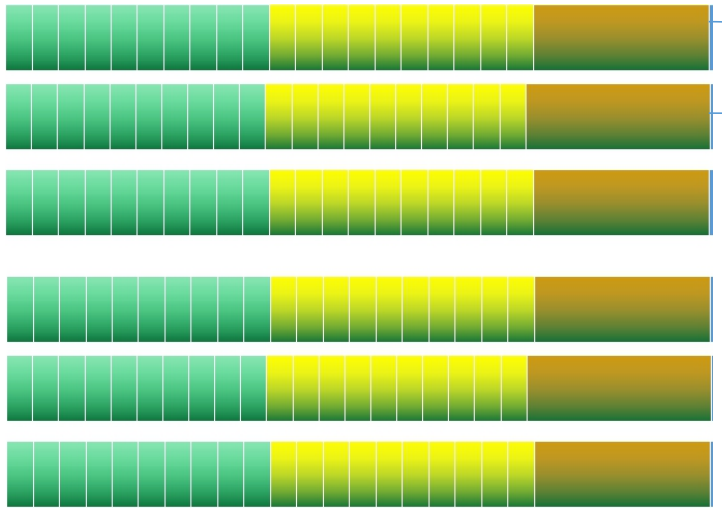
N= number of sequences
Size of window=10

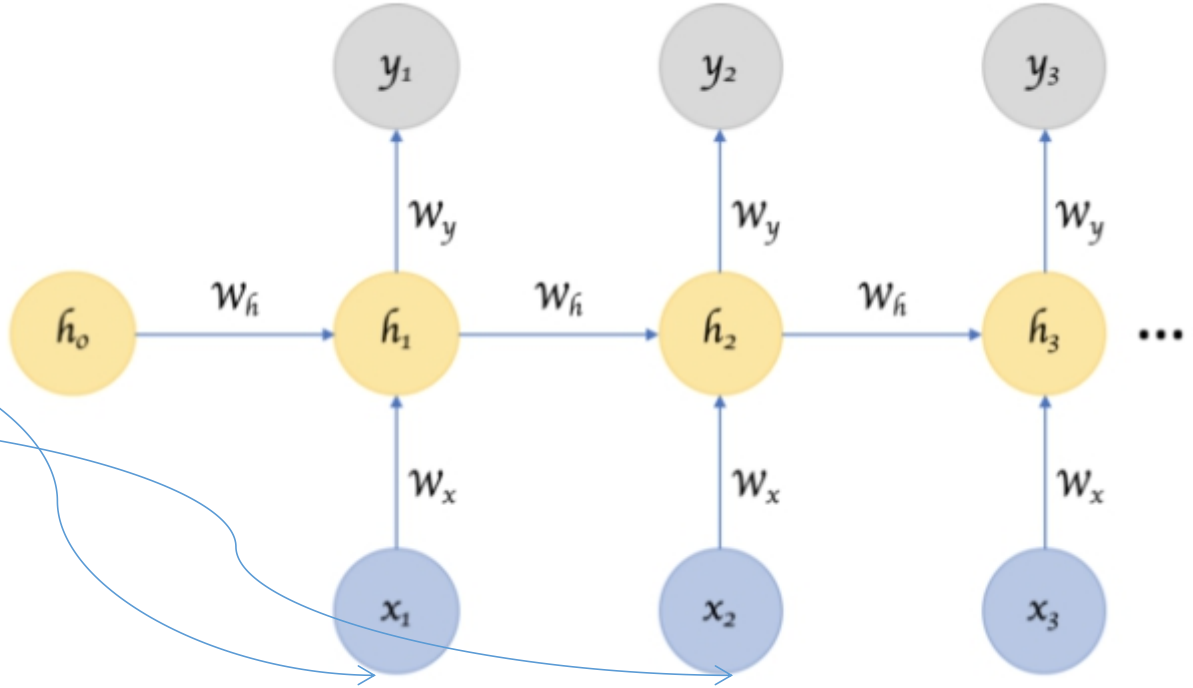
Recurrent Neural Networks-Example



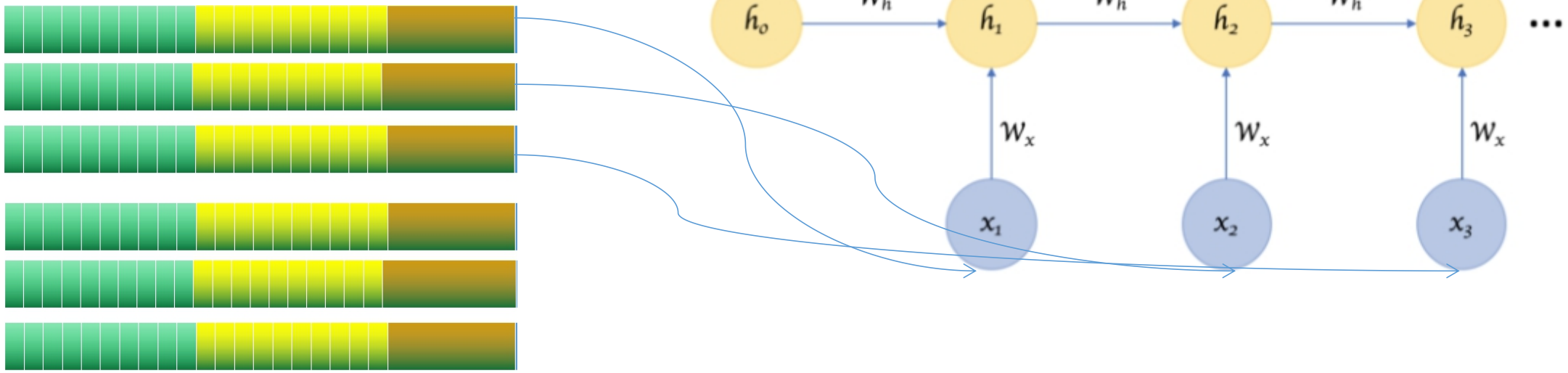


Recurrent Neural Networks-Example





Recurrent Neural Networks-Example



Cinvestav

Muchas Gracias