

# Module: Machine Intelligence & Deep Learning

MA van Wyk

2019

This year, for the first time, a module on modern machine learning is presented. The objective of this module is to expose the student to the intersection of Artificial Intelligence, Computational Intelligence and Machine Learning. Of the many interesting theories including Probabilistic Graphical Models, Evolutionary Computing, Complex Networks etc. within this vast discipline we will only find time to address Artificial Neural Networks and Deep Learning. Our approach here is one starting with the most basic elements/components namely the Artificial Neuron for which we will discuss a popular model and how a neuron “learns”. With this as foundation we will move on to networks of artificial neurons which are called Artificial Neural Networks. This will lead to architectures and topologies associated with Deep Learning. In closing software tools and applications for Deep Learning Neural Networks will be reviewed and some real life applications will be studied.

## The Biological Neuron, Biological Neural Networks

A neuron is a unit of the nervous systems that produces different responses to different stimuli – therefore, from an abstract perspective, it is the most basic information processing device of any living organism. Neurons were discovered round about the same time independently by Spanish neuroscientist [Santiago Ramón y Cajal](#) and British physiologist **Charles Sherrington** at the end of the 19<sup>th</sup> century.

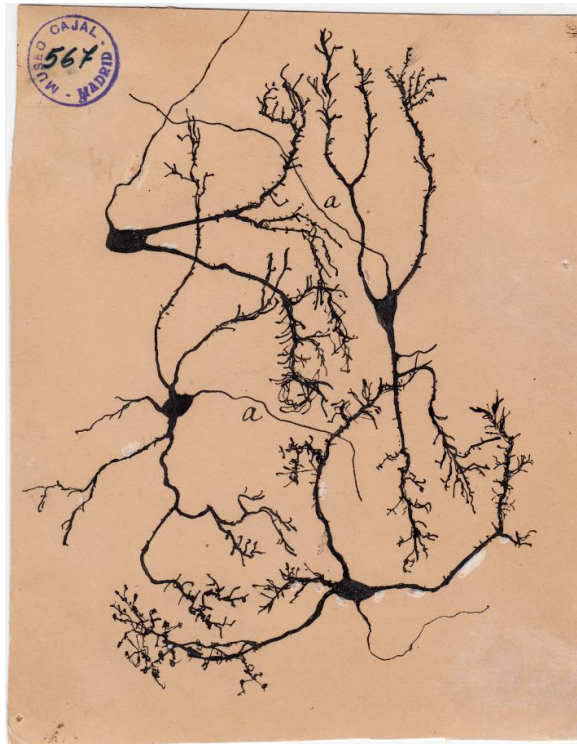


Figure 1. Drawing of dyed brain tissue viewed under a microscope.  
(Source: [University of Minnesota](#))

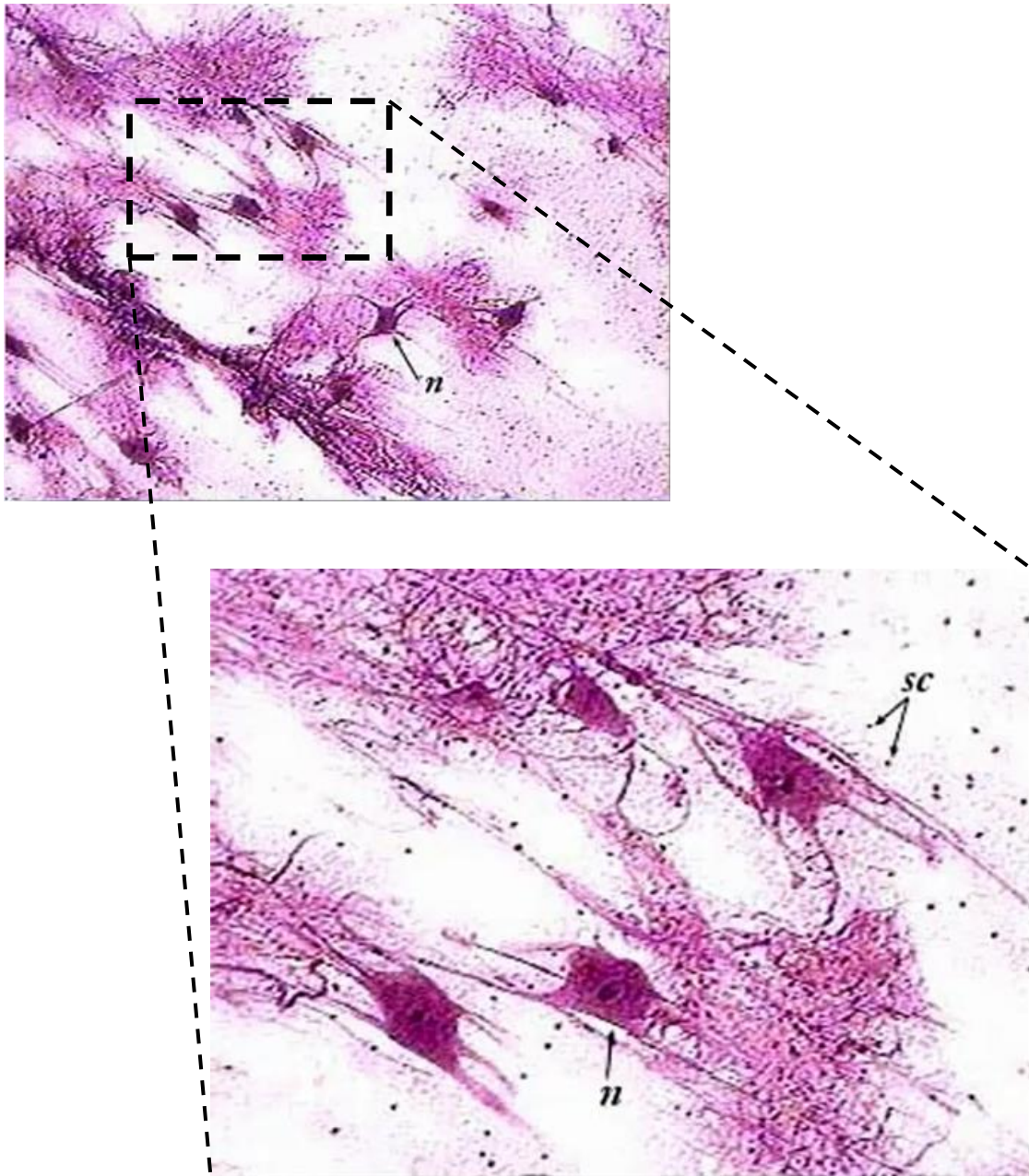


Figure 2. Photo of biological neurons, smeared onto glass, coloured and viewed through a microscope. (Source: [www.austincc.edu](http://www.austincc.edu))

The study of brain tissue has led biologists to identify the interactions between neurons in the brain as depicted in Figures 2 and 3. In short, a neuron is excited through its dendrites (inputs), the neural signals interact in neuron's nucleus generating a non-linear response which in turn excites other neurons through its axon (output). An interesting discussion of how new neural interconnections are created during learning and memorisation can be found in the online documentary [NOVA: Memory Hackers](#).

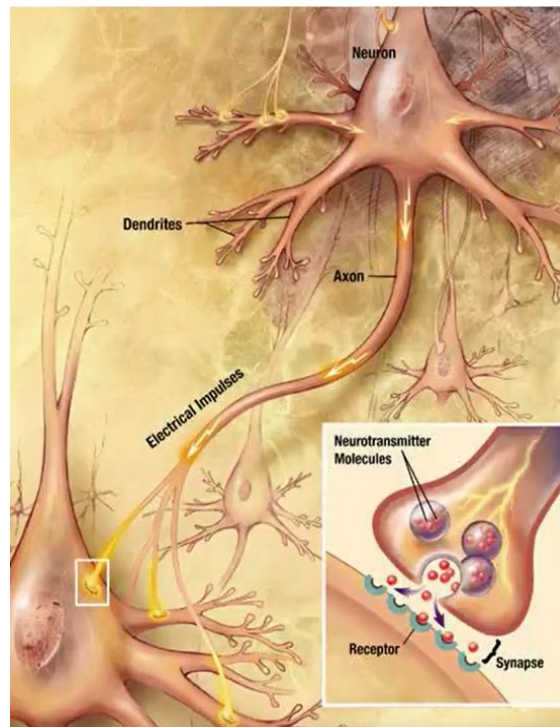


Figure 3. Neurons and their synaptic interactions. (Source: Wikipedia)

## The Artificial Neuron and Artificial Neural Networks

In 1943 **Warren McCulloch** (neuroscientist) and **Walter Pitts** (logician) proposed one of the first models of a neuron. The key characteristic of this model was that it determined a weighted sum of input signals and compared this to a given threshold. When the sum was less than the threshold then the neuron produced no response but when the sum exceeded the threshold then the neuron produced an output. However, this model could not learn and had to be designed. They went on to show that, in principle, networks of these neurons could compute any arithmetic or logical function.

Then in the 1950s **Frank Rosenblatt** and several other researchers developed a class of neural networks called perceptrons with the underlying principle of operation of a perceptron being that of the McCulloch-Pitts neuron. In 1969 **Marvin Minsky** and **Seymour Papert** published a book entitled *Perceptrons: An Introduction to Computational Geometry* in which they presented the first thorough mathematical analysis of perceptron networks. Their findings showed that a two-layer perceptron (i.e. an input layer followed by an output layer) were severely restrictive and that certain primitive logic functions (e.g. Boolean XOR function) could not be modelled by such networks. This resulted in research into neural networks to almost completely halt until, in the 1980s, when these restrictions were overcome resulting in a renewed interest in neural networks. The answer was simple: add more layers! In the case of the XOR logic function creating a perceptron network consisting of an input layer feeding into a hidden layer which in turn feeds the output layers produces a network that has the ability to solve the [XOR function](#) or [XOR problem](#) if the correct weights are chosen. (Haykin, 1994, Section 6.6)



Bet you did not know that Wits is [Seymour Papert's alma mater](#)!

Biological and artificial neuron models proposed in the mid 1900s led to the generally accepted model of an artificial neuron shown in Figure 4. Here  $\mathbf{x} := (x_1 \cdots x_p)^T$  or  $\mathbf{x} := (1 \ x_1 \cdots x_p)^T$  is the vector of inputs sometimes including a constant as first component called the bias. The vector  $\mathbf{w} := (w_1 \cdots w_p)^T$  or  $\mathbf{w} := (w_0 \ w_1 \cdots w_p)^T$  is the vector of weights associated with the input(s) and  $\varphi(\bullet)$  is the activation function of the neuron. The equation describing this computational model is

$$y = \varphi(v) = \varphi(\mathbf{w}^T \mathbf{x}). \quad (1)$$

We recognise the product  $\mathbf{w}^T \mathbf{x}$  as the inner product (also called dot product and scalar product) of the vectors  $\mathbf{w}$  and  $\mathbf{x}$ . Sometimes we might want to explicitly indicate the bias' contribution. In such instances we will write  $\theta := w_0 \cdot 1 \equiv w_0$  for the simple reason that  $\theta$  is so commonly used in the literature. Equation then (1) becomes

$$y = \varphi(\mathbf{w}^T \mathbf{x} + \theta) \quad (\text{bias absorbed}) \quad (2)$$

with the signal and weight vectors now not containing the bias component. Note:  $\mathbf{w}^T \mathbf{x} = \mathbf{x}^T \mathbf{w}$ .

Several reasons can be given as motivation for including a bias term which include choosing a non-zero threshold level for discontinuous activation functions as well as using biases to set the operating points of neurons. By incorporating the bias as an additional input component and assigning to it a weight enables us to obtain a more economical presentation of equations.

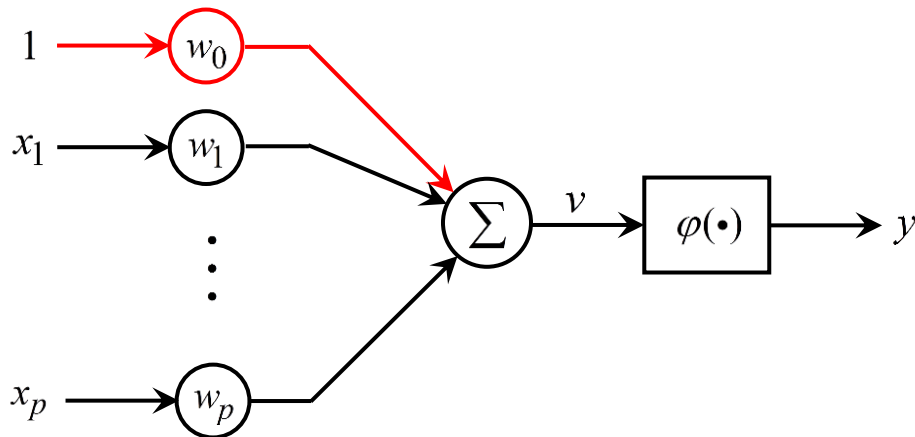


Figure 4. Computational model of an artificial neuron.



Knowing only the choice of activation function and input weight vector for a given neuron, how does one determine whether the neuron is operating in the linear regime or non-linear regime?

An Artificial Neural Networks usually consists of large numbers of densely connected neurons. Therefore we need to significantly simplify the graphical representation of a neuron (see Figure 5) in order not to obscure the important bigger picture focussing of the interconnections.



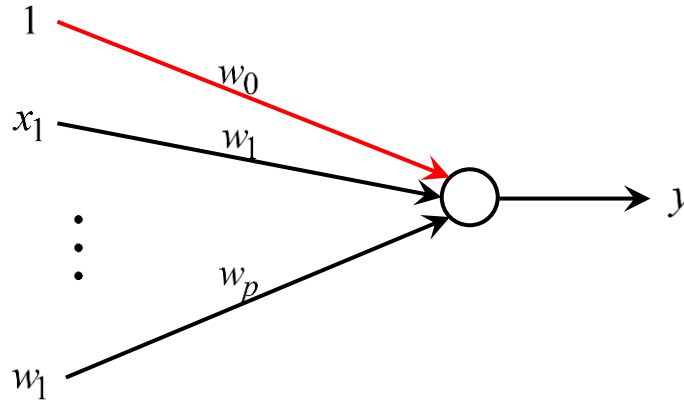


Figure 5. The signal-flow graph of a neuron.

## Geometric Interpretation of a Neuron

From a geometric point of view the linear discriminator  $f(\mathbf{w}, \mathbf{x}) := \mathbf{w}^T \mathbf{x} + \theta$  can be interpreted in two ways. Firstly, by considering both the weight  $\mathbf{w}$  and the bias as given and fixed  $f_{\mathbf{w}, \theta}(\mathbf{x}) := \mathbf{w}^T \mathbf{x} + \theta$  can be visualised in the  $p$ -dimensional space of input patterns, called the pattern space. Since this function's value can be negative ( $-$ ),  $0$  or positive ( $+$ ), the function  $\text{sgn}(\mathbf{w}^T \mathbf{x} + \theta)$  discriminates ("polarises") the pattern space into three distinct regions: there is a region in which  $\text{sgn}(\mathbf{w}^T \mathbf{x} + \theta) = -1$ , the region in which  $\text{sgn}(\mathbf{w}^T \mathbf{x} + \theta) = \mathbf{w}^T \mathbf{x} + \theta = 0$  termed the decision boundary and the region in which  $\text{sgn}(\mathbf{w}^T \mathbf{x} + \theta) = +1$ . For simplicity we will identify these three regions using the tags " $\text{sgn} = -1$ ", " $\text{sgn} = 0$ " and " $\text{sgn} = +1$ " respectively as shown below. These three regions show which inputs will be labelled  $-1$ ,  $0$  and  $+1$  for a given weight vector.



Can you prove the following three facts?

- The above three regions, each, is a connected region of  $\mathbf{R}^p$ .
- Only the region associated with  $\mathbf{w}^T \mathbf{x} + \theta = 0$  (i.e. tagged " $\text{sgn} = 0$ ") is a manifold. This manifold separates the other two regions and is called the decision boundary.
- These three regions are disjoint (i.e. they do not overlap).

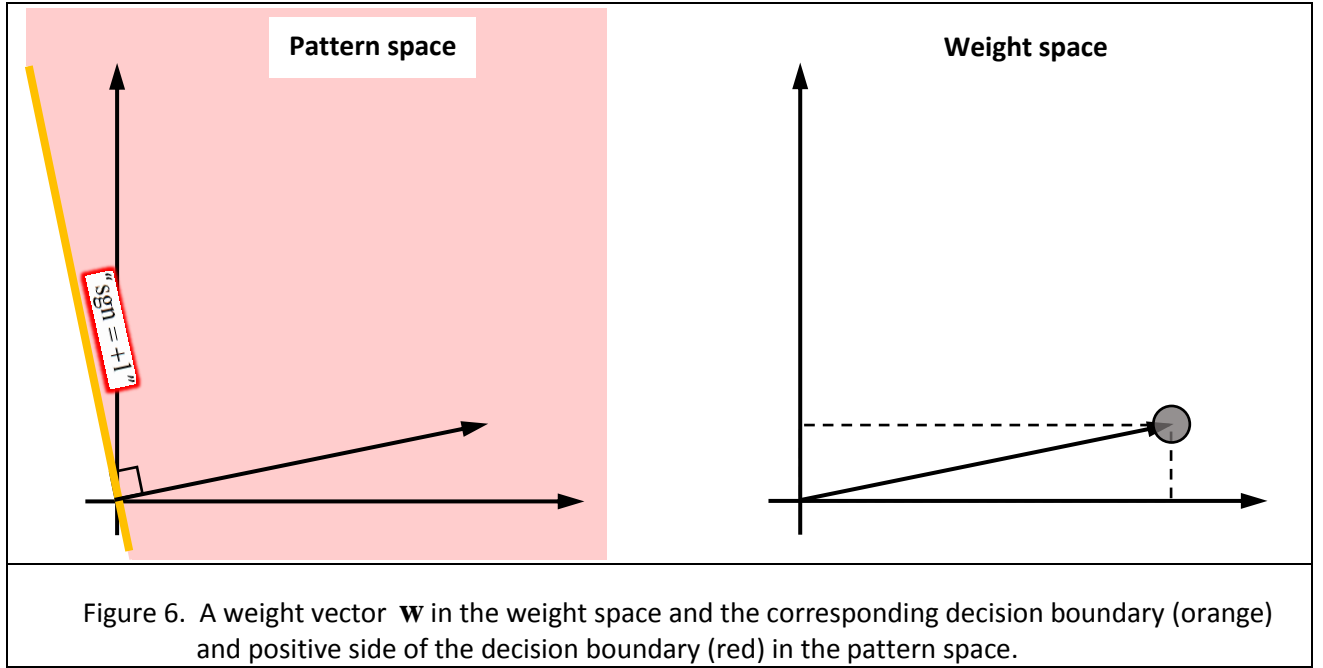


What mathematical operation is needed to reverse the decision polarity for a given linear discriminator?

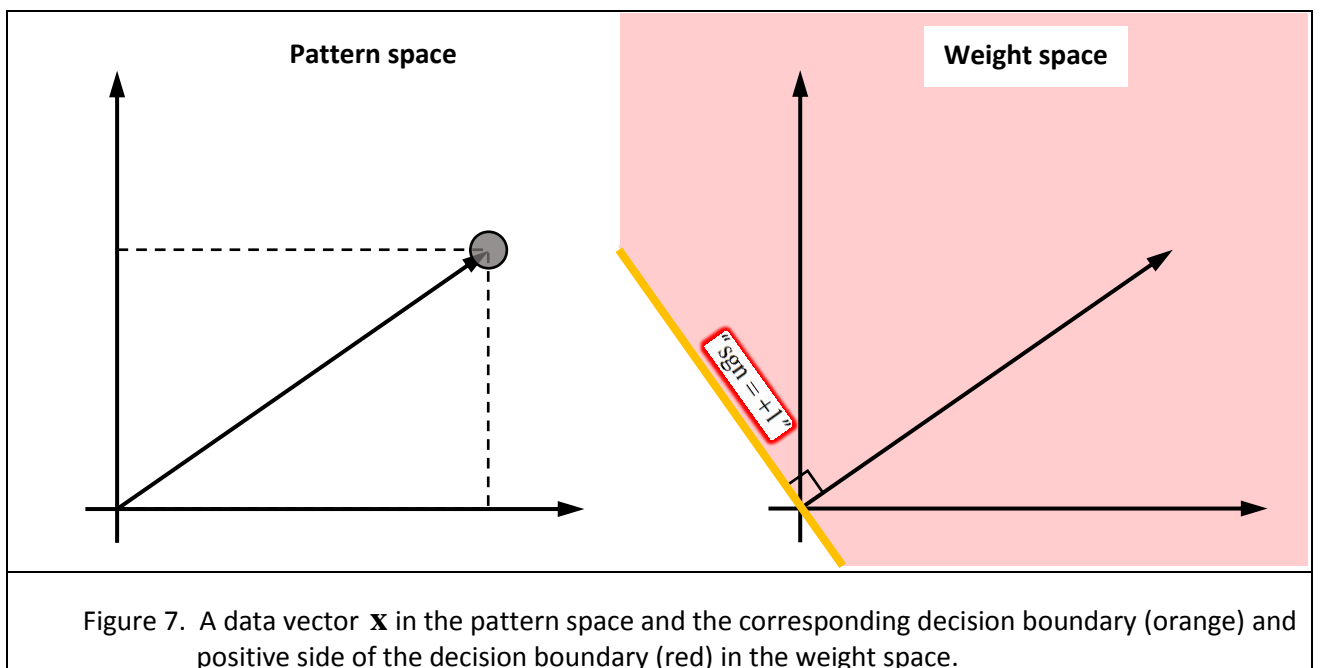


A manifold  $M$  is a connected region in  $\mathbf{R}^p$  which has the property that it can always be transformed into a vector subspace of  $\mathbf{R}^p$  by simply subtracting from each vector in  $M$  a fixed vector in  $M$ . Conversely, adding to each vector in a subspace  $V$  of  $\mathbf{R}^p$  any fixed vector from the set  $V^c$ , produces a manifold. In short *a manifold is a translated vector space*.

Clearly the decision boundary is orthogonal to the given weight vector  $\mathbf{w}$  with the positive side of it corresponding to the side to which vector  $\mathbf{w}$  is pointing as shown below in Figure 6.



Secondly, by considering the pattern vector  $\mathbf{x}$  as given and fixed,  $f_{\mathbf{x}}(\mathbf{w}) := \mathbf{x}^T \mathbf{w} + \theta$  can be visualised in the  $p$ -dimensional space of weights, called the weight space. Since the inner product is symmetric in  $\mathbf{w}$  and  $\mathbf{x}$  (i.e.  $\mathbf{w}^T \mathbf{x} = \mathbf{x}^T \mathbf{w}$ ) the same geometric properties described above for the pattern space also immediately hold for the weight space as shown below in Figure 7. As in pattern space, here too, we will identify these three regions using the tags "sgn = -1", "sgn = 0" and "sgn = +1" respectively in accordance with the sign of  $f_{\mathbf{x}}(\mathbf{w})$ . Here these three regions show which weights will result in the given input vector  $\mathbf{x}$  to be labelled -1, 0 and +1. For this reason it will be convenient to interpret the neuron learning process in weight space where the equation  $\mathbf{w}^T \mathbf{x} + \theta = 0$  describes the separating decision boundary just as in pattern space.



The mere fact that

- the decision boundary separates two region and
- the decision boundary is a translated linear space ( vector) space, that is, not curved

implies that a single neuron can only correctly discriminate a set of input patterns that is linearly separable. A set of data vectors (input patterns) is said to be linearly separable if a decision boundary separating that region of vectors labelled “ $\text{sgn} = -1$ ” from the region of vectors labelled “ $\text{sgn} = +1$ ” is a manifold consisting of vectors exclusively labelled “ $\text{sgn} = 0$ ” as in Figure 8(a).

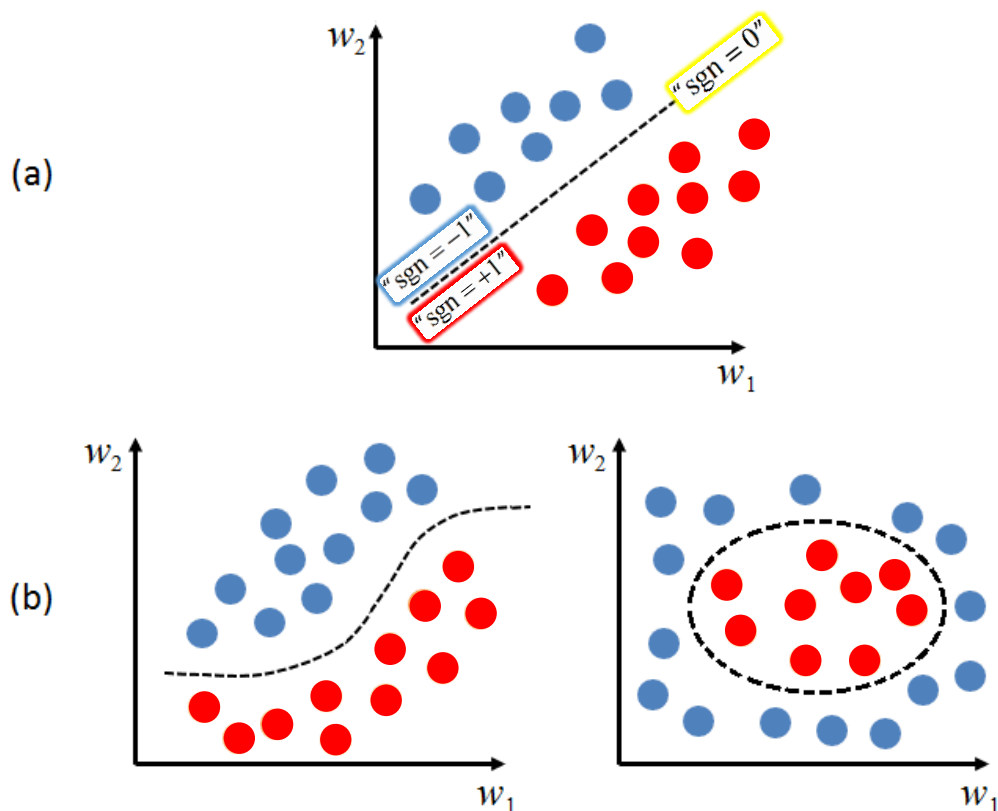


Figure 8. The data set in (a) is linearly separable and the data sets in (b) are non-linearly separable but not linearly separable. (Source: [John Sullivan](#))



- With the regions “ $\text{sgn} = +1$ ” and “ $\text{sgn} = -1$ ” as assigned in Figure 8(a), what is the direction of the normal vector relative to the decision boundary?
- What transformations will render the sets of data vectors, in Figure 8(b), linearly separable?

We summarise the above discussion in the form of a (mathematical) lemma:

**Lemma:** Suppose  $\mathbf{x}^*$  is an arbitrary but fixed point in pattern space and similarly let  $\mathbf{w}^*$  be an arbitrary but fixed point in weight space. Then the equation  $\mathbf{w}^T \mathbf{x} \big|_{\mathbf{x}=\mathbf{x}^*} = 0$  relates the point

$\mathbf{x} = \mathbf{x}^*$  in the pattern space to a manifold (a straight line when case  $p = 2$ ) in weight space with the side of the decision boundary in which  $\mathbf{x}^T \mathbf{w}^* > 0$  given by side toward which  $\mathbf{w}^*$  is pointing . Similarly the equation  $\mathbf{x}^T \mathbf{w}|_{\mathbf{w}=\mathbf{w}^*} = 0$  relates the point  $\mathbf{w} = \mathbf{w}^*$  in the weight space to a manifold (a straight line when case  $p = 2$ ) in pattern space with the side of the decision boundary in which  $\mathbf{w}^T \mathbf{x}^* > 0$  given by side toward which  $\mathbf{x}^*$  is pointing .



I am sure you can do the proof, right?

The reason for the presence of a bias, feeding into a neuron, is still not clear. The following example will serve as a motivation for the presence of the bias input.

### Example

We will consider the two-input Boolean AND function  $o = x_1 \text{ AND } x_2$  with the truth table below in Table 2 to be modelled with a single neuron. In order to aid with the interpretation, from a neuron perspective, we cast the truth table into an input-output table shown below in Table 3. Since the function being modelled by the neuron is a Boolean logic function it makes sense to choose the neuron's activation function to be the unit step function  $u(v)$ .

Table 2. Truth table for the Boolean AND function.

$o :$

	$x_1$		
		0	1
$x_2$	0	0	0
	1	0	1

Table 3. Input-Output table for the Boolean AND function  
(e.g.  $\mathbf{x}_4 := (1, 1)^T$  and  $d_4 = 1$ ).

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
$x_1$	0	1	0	1
$x_2$	0	0	1	1
$o$	0	0	0	1
	$d_1$	$d_2$	$d_3$	$d_4$

For this example the input to the activation function of the neuron is

$$v = \mathbf{w}^T \mathbf{x} = w_1 x_1 + w_2 x_2$$

where the input vector or input pattern  $\mathbf{x} \in \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ . We wish to locate the points  $\mathbf{x} \in \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$  in the pattern space and then plot the associated decision boundary, in the weight space, with each (manifold) described by an equation of the form



$$\mathbf{w}^T \mathbf{x} = w_1 x_1 + w_2 x_2 = 0.$$

which is recognised as the equation for a straight line in  $\mathbf{R}^2$ .

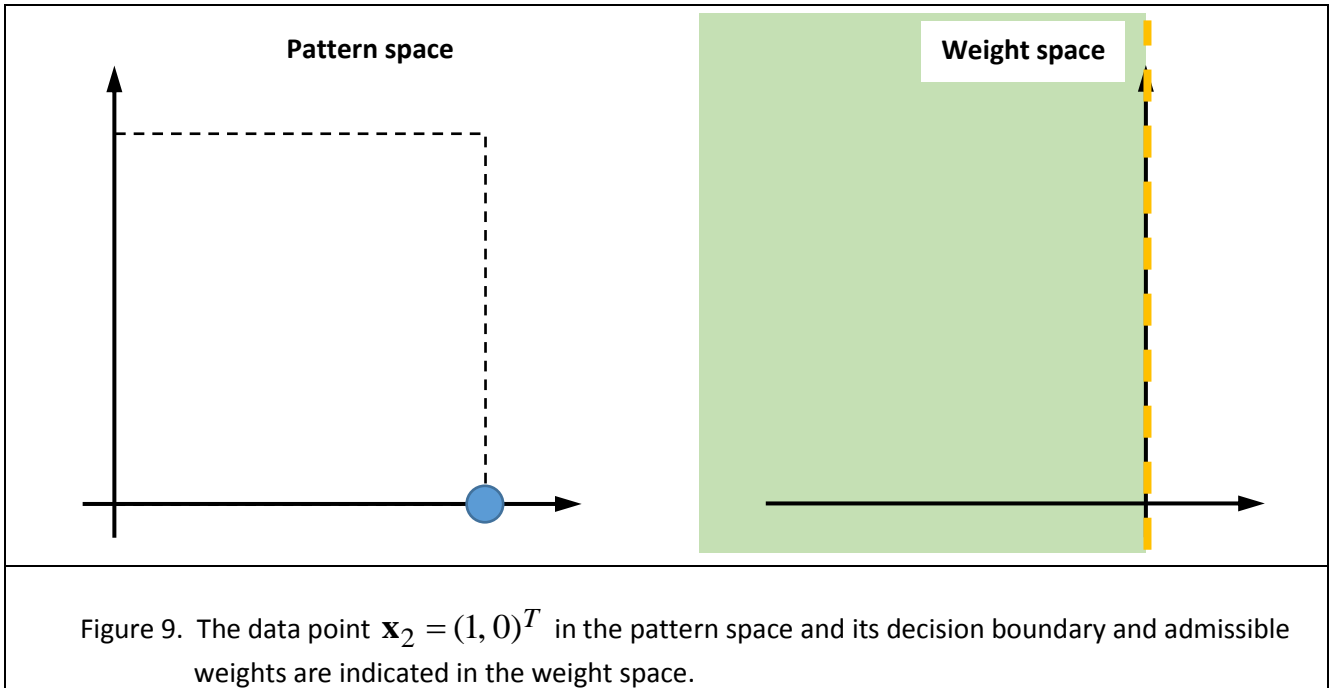
For  $\mathbf{x} = \mathbf{x}_2 = (1, 0)^T$  the decision boundary, in weight space, is described by

$$\mathbf{w}^T \mathbf{x}_2 = w_1 \times 1 + w_2 \times 0 = 0 \Rightarrow \{w_1 = 0, w_2 \in \mathbf{R}\}$$

which is a straight line coinciding with  $w_2$ -axis in weight space as depicted by the dashed orange line in Figure 9. The desired output  $d_2 = 0$  when input pattern  $\mathbf{x}_2$  is applied to the neuron's inputs, requires the weights to satisfy

$$\mathbf{w}^T \mathbf{x}_2 = w_1 \times 1 + w_2 \times 0 < 0 \Rightarrow \{w_1 < 0, w_2 \in \mathbf{R}\}.$$

The set of all admissible weights, i.e. those that will ensure that the correct output is generated for the input  $\mathbf{x}_2$ , is the green shaded region in Figure 9.



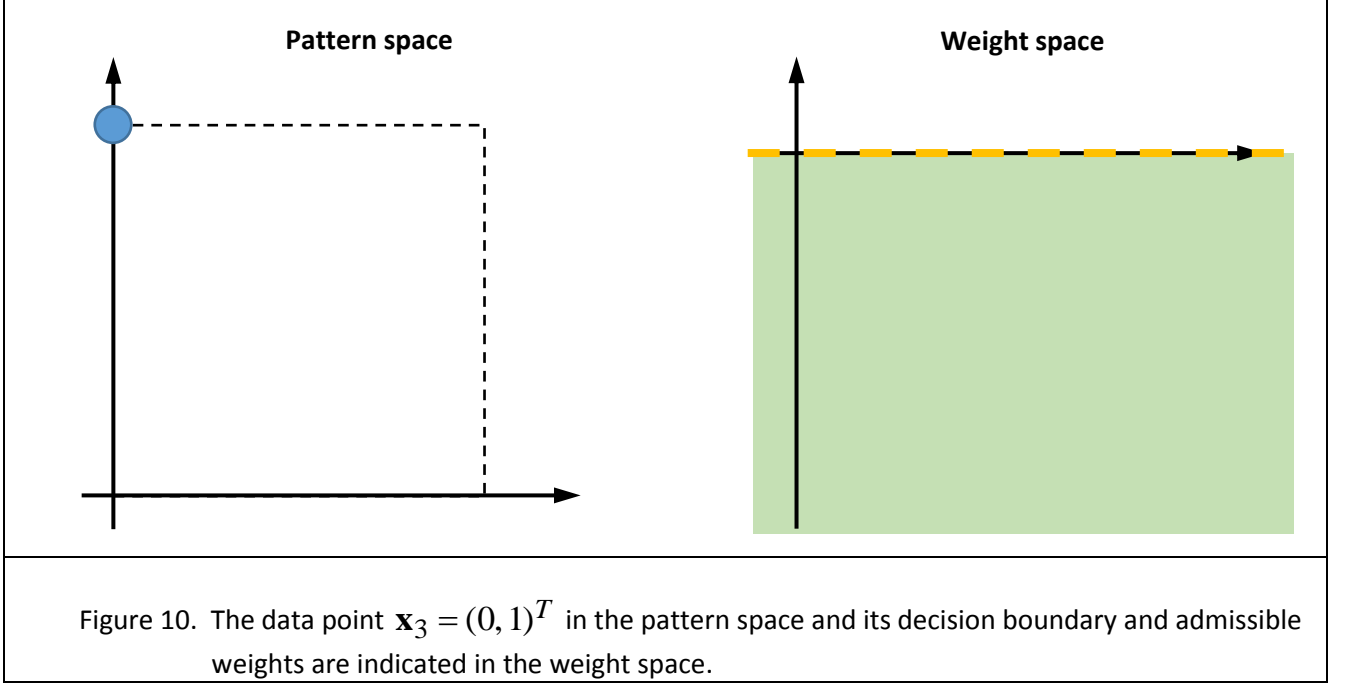
For  $\mathbf{x} = \mathbf{x}_3 = (0, 1)^T$  the decision boundary, in weight space, is described by

$$\mathbf{w}^T \mathbf{x}_3 = w_1 \times 0 + w_2 \times 1 = 0 \Rightarrow \{w_1 \in \mathbf{R}, w_2 = 0\}$$

which is a straight line coinciding with  $w_1$ -axis in weight space as depicted by the dashed below in Figure 10. The desired output  $d_3 = 0$  when input pattern  $\mathbf{x}_3$  is applied to the neuron's inputs, requires the weights to satisfy

$$\mathbf{w}^T \mathbf{x}_3 = w_1 \times 0 + w_2 \times 1 < 0 \Rightarrow \{w_1 \in \mathbf{R}, w_2 < 0\}.$$

The set of all admissible weights that will ensure that the correct output is generated for the input  $\mathbf{x}_3$ , is the green shaded region in Figure 10.



For  $\mathbf{x} = \mathbf{x}_1 = (0, 0)^T$  the decision boundary, in weight space, is described by

$$\mathbf{w}^T \mathbf{x}_1 = w_1 \times 0 + w_2 \times 0 = 0 \Rightarrow \{w_1 \in \mathbf{R}, w_2 \in \mathbf{R}\}$$

which is the whole 2 dimensional plane. The desired output  $d_1 = 0$  when input pattern  $\mathbf{x}_1$  is applied to the neuron's inputs, requires the weights to satisfy

$$\mathbf{w}^T \mathbf{x}_1 = w_1 \times 0 + w_2 \times 0 < 0 \Rightarrow (w_1, w_2) \in \emptyset.$$

This is an artefact of the step function used as activation function. The set of all admissible weights that will ensure that the correct output is generated for the input  $\mathbf{x}_1$ , is the empty set in the weight space in Figure 11.

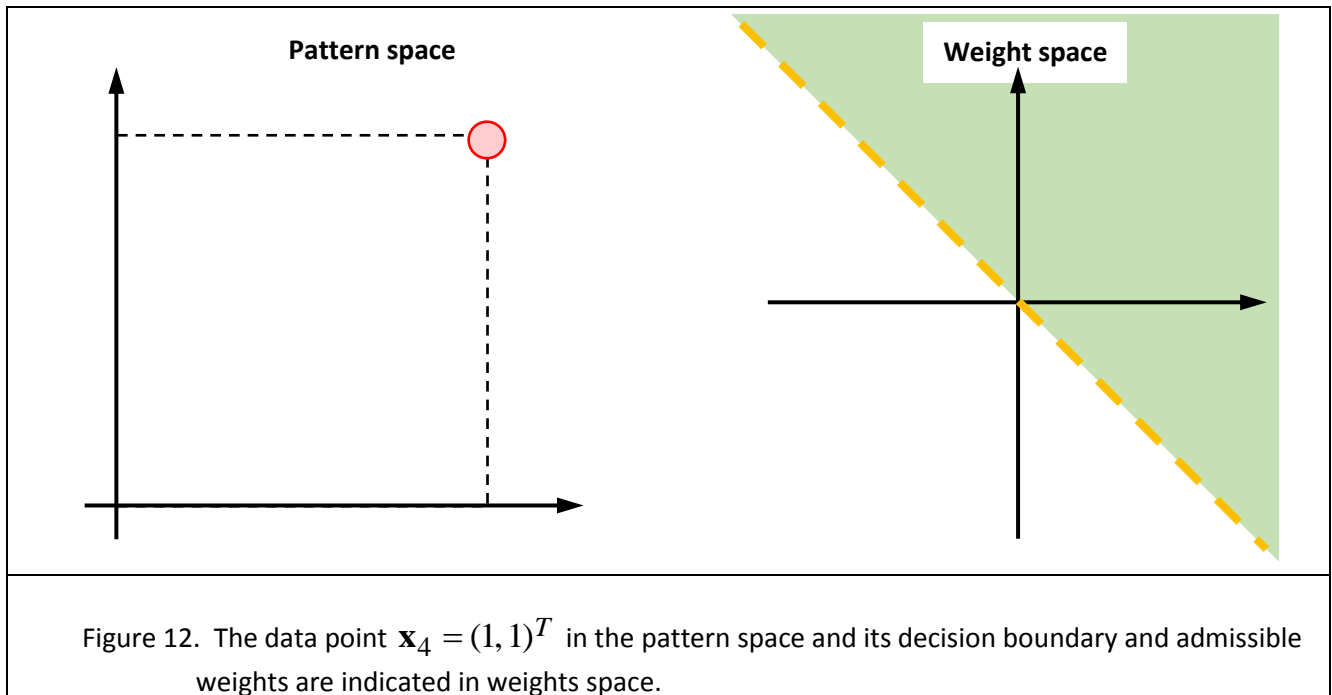
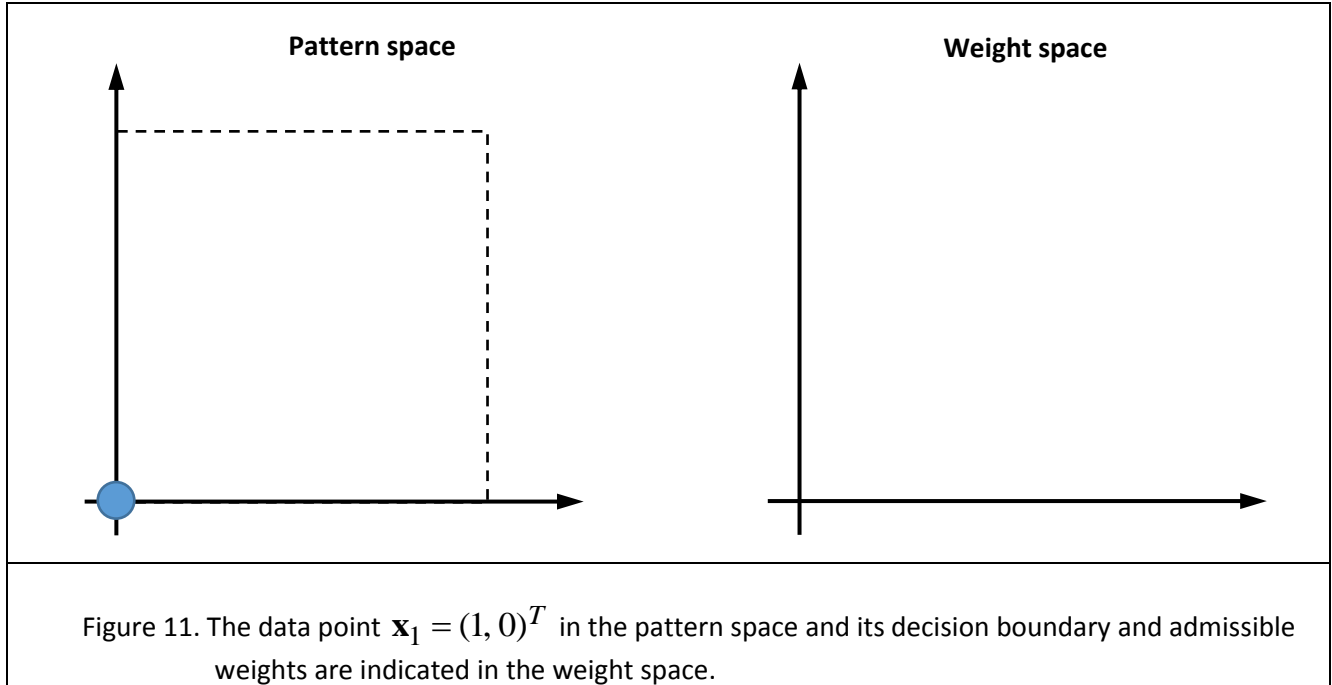
Finally for  $\mathbf{x} = \mathbf{x}_4 = (1, 1)^T$  the decision boundary, in weight space, is described by

$$\mathbf{w}^T \mathbf{x}_4 = w_1 \times 1 + w_2 \times 1 = 0 \Rightarrow w_2 = -w_1$$

which is a diagonal straight line with slope of  $-1$  and passing through the origin in weight space as depicted by the dashed orange line in Figure 12. The desired output  $d_4 = 1$  when input pattern  $\mathbf{x}_1$  is applied to the neuron's inputs, requires the weights to satisfy

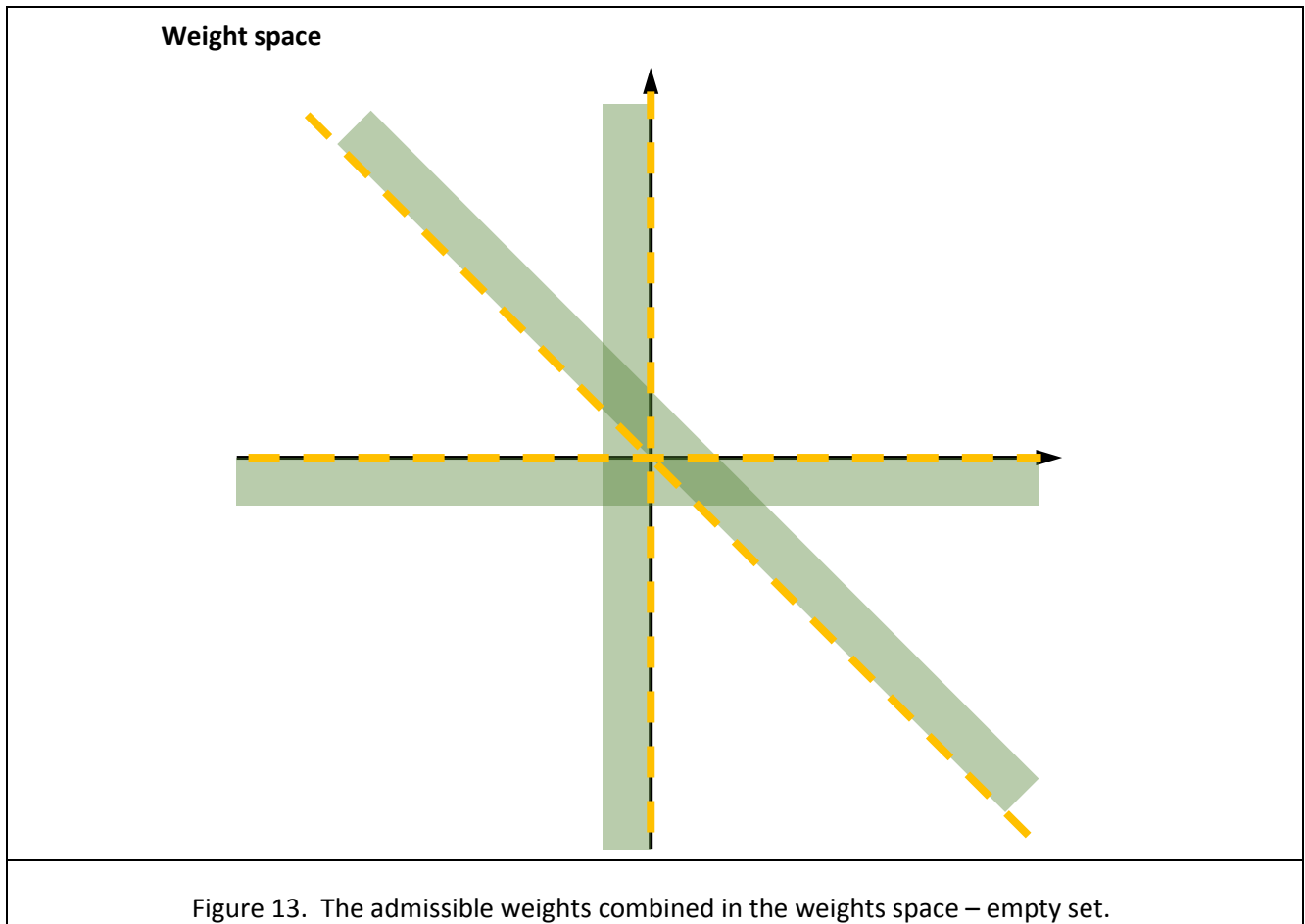
$$\mathbf{w}^T \mathbf{x}_4 = w_1 \times 1 + w_2 \times 1 > 0 \Rightarrow w_2 > -w_1$$

The set of all admissible weights that will ensure that the correct output is generated for the input  $\mathbf{x}_3$ , is the green shaded region in Figure 12.



Now, the set of all weight vectors for which the neuron will model the two-input AND gate is the intersection of the sets of admissible weights within the green regions of Figures 9, 10, 11 and 12.

Because Figure 11 has no green region and because the others only overlap pairwise this intersection is empty, that is, there is no weight vector that enables the neuron to model the two-input AND gate.



By introducing a bias the equation describing the input to the activation function of the neuron (i.e. the dot product) is given by

$$v = \mathbf{w}^T \mathbf{x} + \theta = w_1 x_1 + w_2 x_2 + \theta$$

For the choice  $\theta = -\frac{3}{2}$  and weight vector  $\mathbf{w} = (1, 1)^T$  the neuron now successfully models the AND gate.



- Repeat this example and sketch the weight spaces for the case when a bias of  $\theta = -\frac{3}{2}$  is fed into the neuron.
- Determine the full range of bias values and corresponding sets of admissible weights for which the neuron in part a will be able to model the AND gate successfully.

## Different Neuron Activation Functions Used

Table 1. List of frequently used neural [activation functions](#).

Descriptive name	Activation function $\varphi(v)$	Mnemonic
Identity function	$v$	purelin
Bipolar threshold function	$\text{sgn } v$	hardlims
Unipolar threshold function	$\max(\text{sgn } v, 0) \stackrel{\text{a.e.}}{=} \frac{1+\text{sgn } v}{2} \stackrel{\text{a.e.}}{=} u(v)$	hardlim
Unipolar saturating linear function	$\max(\min(v, 1), 0)$	satlin
Bipolar saturating linear function	$\max(\min(v, 1), -1)$	Satlins
Rectifier function	$\max(v, 0)$	relu, poslin
Sigmoid function	$\frac{1}{1+e^{-v}}$	logsig
Hyperbolic tangent function	$\frac{e^v - e^{-v}}{e^v + e^{-v}}$	tansig
Gaussian function	$e^{-v^2}$	gauss
Distance function	$\ v\ $	
Maximum function	$\max_j v_j$	maxout
Normalised ranking function	$y_k = \frac{v_k}{\sum_j v_j}, \quad \forall k$	softmax
Competitive function	$y_k = \delta_{k, \arg \max_j v_j}$	compet

The obvious question arises namely, on what grounds does one choose the activation function? A few of the considerations to be considered are:

- Is uniform-approximation-ability essential?  
For function approximation (LT1) it would yield greater flexibility to choose an activation function that would ensure Uniform Approximation ability. On the other hand for classification (LT3) this is not essential.
- Is continuity or differentiability important?  
For the approximation of continuous (resp. differentiable) functions (LT1) continuity (resp. differentiability) the activation function is required to be continuous (resp. differentiable). Even if function approximation (LT1) of a function with discontinuities is considered, using learning rules such as gradient descent (LR3) or back propagation (LT4) would require the activation functions at least to be piecewise differentiable.
- Is learning rate of convergence important? While learning the rate of convergence of weights toward a solution varies depending on the choice of activation function.
- Is it required of the neuron (or neural network) to be able to operate linearly as well as non-linearly?

Some activation functions have both linear and non-linear regions; for example, both the sigmoid and hyperbolic tangent functions have regions of almost linearity and non-linearity and could therefore operate in both regimes.

## Biological Learning Process

Much literature exists on the biological process of learning. However, time does not allow to digress into this fascinating body of research. Some of the recent results about learning and memory are discussed in the documentary [NOVA: Memory Hackers](#).

## Artificial Learning Process

Artificial learning is a process by which the free parameters of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the context in which the parameter changes take place. (Haykin, 1994)

## Learning Schemes

### LS1. Supervised learning

With this approach exemplars of input stimuli and corresponding output responses, collectively an input-output training set, are assumed given which are then presented to the learning system (neuron or neural network) in order to influence its learning of the task being considered.

### LS2. Unsupervised learning

Here the objective is for the learning system to discover associations

### LS3. Reinforcement learning

This approach is based on encouragement of constructive actions and discouragement of nonconstructive actions invented and/or performed by the learning system.

## Learning Rules

### LR1. Perceptron Learning

Consider a perceptron, that is, a neuron with activation function is *hardlim*. This supervised learning algorithm, presents an input pattern to the neuron. If the neuron's output is correct

$$\mathbf{w}_k = \mathbf{w}_{k-1} + (d_l - y(\mathbf{x}_l))\mathbf{x}_l \quad (\text{bias absorbed})$$



- a. Choosing  $\mathbf{w}_0^T = (1, 0, 0)$  and  $\mu = \frac{3}{2}$ , enumerate analytically the sequence of weight vectors produced while attempting in an attempt to learn the Boolean NAND function



respectively by manual calculation. Plot the final decision boundary. ~~(The hints directly above applies here.)~~

- b. Repeat these this exercise for the Boolean NOR function.

## LR2. Error Correction Learning

Consider a perceptron, that is, a neuron with activation function is  $\varphi(v) := \text{sgn } v$ . This supervised learning algorithm, presents an input pattern to the neuron. If the neuron's output is correct (compared to the required output value) then the neuron's weight vector is left unchanged. However, if the output is incorrect, then the weight vector is updated by flipping it to the opposite side of the decision boundary in weight space thereby correcting the error.

We will now derive this learning rule. Recall that the decision boundary is described by the equation

$$\mathbf{w}^T \mathbf{x} = 0. \quad (\text{bias absorbed}) \quad (3)$$

Since we are working in the weight space the input pattern vector  $\mathbf{x}$  is assumed to be given and fixed while  $\mathbf{w}$  is assumed to be our variable. Obviously the zero vector, i.e.  $\mathbf{w} = \mathbf{0}$  is one of the vectors satisfying (3) but there are infinitely many such vectors namely all vectors orthogonal to  $\mathbf{x}$ . From this we conclude that the decision boundary (with the weight vector as variable) is described by all vectors  $\mathbf{w}$  orthogonal to the given and fixed input pattern vector  $\mathbf{x}$ . The vector  $\mathbf{x}$  is the so called normal to the plane described by the equation  $\mathbf{w}^T \mathbf{x} = 0$ . It is custom to work with the unit normal associated with a plane and for this reason we introduce the unit normal and write it as

$$\hat{\mathbf{x}} := \frac{\mathbf{x}}{\|\mathbf{x}\|}, \quad \|\mathbf{x}\| := \sqrt{\mathbf{x}^T \mathbf{x}}.$$



Prove that  $\|\hat{\mathbf{x}}\| = 1$ .

We can now express the equation for the decision boundary as

$$\mathbf{w}^T \mathbf{x} = \mathbf{w}^T \|\mathbf{x}\| \hat{\mathbf{x}} = \|\mathbf{x}\| \mathbf{w}^T \hat{\mathbf{x}} = 0 \quad \Rightarrow \quad \mathbf{w}^T \hat{\mathbf{x}} = 0$$

Now, suppose the weight vector was initialised to the value  $\mathbf{w}_0$  (typically chosen randomly). Suppose it is desired that this given pattern vector  $\mathbf{x}$  be classified by the neuron as  $d(\mathbf{x}) = +1$  but that  $\mathbf{w}_0^T \mathbf{x} < 0$  so that

$$y(\mathbf{x}) = \varphi(\mathbf{w}_0^T \mathbf{x}) = -1.$$

Recall from first year mathematics that

$$c_0 := \mathbf{w}_0^T \hat{\mathbf{x}} = \|\mathbf{w}_0\| \|\hat{\mathbf{x}}\| \cos \gamma = \|\mathbf{w}_0\| \cos \gamma$$

and consequently  $\mathbf{w}_0^T \mathbf{x} < 0$  means that  $\mathbf{w}_0^T \hat{\mathbf{x}} < 0$  which implies that  $\cos \gamma < 0$  and this means that  $\frac{\pi}{2} < \gamma < \frac{3\pi}{2}$ . In other words the vector  $\mathbf{w}_0$  has a component in the direction of  $-\hat{\mathbf{x}}$ . This means that  $\mathbf{w}_0$  can be express in the form  $\mathbf{w}_0 = c_0 \hat{\mathbf{x}} + \mathbf{r}$  for some vector  $\mathbf{r}_0$  orthogonal to  $\hat{\mathbf{x}}$ .



A little thought reveals that  $c_0$  is the projection of  $\mathbf{w}_0$  onto the unit normal  $\hat{\mathbf{x}}$  of the decision boundary and therefore  $|c_0|$  the distance from the tip of  $\mathbf{w}_0$  to the decision boundary.

Clearly the input pattern  $\mathbf{x}$  is incorrectly mapped by the neuron and hence we need to correct this error by updating the weight vector, changing it from  $\mathbf{w}_0$  to a new value  $\mathbf{w}_1$  such that

$$y(\mathbf{x}) = \varphi(\mathbf{w}_1^T \mathbf{x}) = +1 \equiv d(\mathbf{x}).$$

To achieve this all we need to is to add the appropriate increment  $\delta \mathbf{w}$  to  $\mathbf{w}_0$  which is what we turn to now. To flip the weight vector to the other side of the decision boundary we need to add the appropriate multiple of  $\hat{\mathbf{x}}$  to  $\mathbf{w}_0$ . By the above reasoning  $\delta \mathbf{w} = \delta c \hat{\mathbf{x}}$  with  $\delta c > -c_0 > 0$  will suffice, giving  $\mathbf{w}_1 := \mathbf{w}_0 + \delta \mathbf{w}$ . Lets verify this by direct calculation:

$$c_1 := \mathbf{w}_1^T \hat{\mathbf{x}} = (\delta \mathbf{w} + \mathbf{w}_0)^T \hat{\mathbf{x}} = \delta \mathbf{w}^T \hat{\mathbf{x}} + \mathbf{w}_0^T \hat{\mathbf{x}} = \delta c + c_0 > 0$$

from which follows that  $\mathbf{w}_1^T \mathbf{x} = \|\mathbf{x}\| \mathbf{w}_1^T \hat{\mathbf{x}} = \|\mathbf{x}\| (\delta c + c_0) > 0$  as required. The error has now been corrected and the neuron maps the given input pattern  $\mathbf{x}$  to the correct value

$$y(\mathbf{x}) = \varphi(\mathbf{w}_1^T \mathbf{x}) = +1 \equiv d(\mathbf{x}).$$

A similar reason applies to the case when the desired output is  $d(\mathbf{x}) = -1$  but the neuron maps the input pattern incorrectly to  $y(\mathbf{x}) = \varphi(\mathbf{w}_0^T \mathbf{x}) = +1$ . In this case we just need to change the sign of  $\delta \mathbf{w}$ . We are now in a position to write down the error correcting learning rule.

**Summary of the algorithm:** Suppose a single neuron with a symmetric hard-limiter as activation function is required to learn the linearly separable set of perhaps noise-contaminated input-output pairs  $\{(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \dots, (\mathbf{x}_N, d_N)\}$  with the vector input  $\mathbf{x}_k \in \mathbf{R}^3$  and scalar output  $d_k \in \{-1, 1\}$  for each  $k = 1, 2, 3, \dots, N = 100$ .

**Step 1:** Choose an arbitrary initial weight vector  $\mathbf{w}_0$ .

**Step 2:** For each  $k$ , from 1 to  $N$ , in any order (lets agree to do this in chronological order) present the  $k^{\text{th}}$  input vector  $\mathbf{x}_k$  to the neuron and compare the actual output  $y(\mathbf{x}_k)$  to the desired output  $d_k$ . If they have the same value then the  $k^{\text{th}}$  input-output pair  $(\mathbf{x}_k, d_k)$  is already assimilated and there is no need to change the weight vector, that is  $\mathbf{w}_k = \mathbf{w}_{k-1}$  and repeat Step 2 for the next value of  $k$ .

**Step 3:** However, if  $y(\mathbf{x}_k) \neq d_k$  the weights have to be updated in order to assimilate the input-output pair  $(\mathbf{x}_k, d_k)$ . The weight update that will ensure that this input-output pair is learned is

$$\mathbf{w}_k = \mathbf{w}_{k-1} - \mu \left( \mathbf{w}_{k-1}^T \hat{\mathbf{x}}_k \right) \hat{\mathbf{x}}_k \quad \text{with} \quad \hat{\mathbf{x}}_k := \frac{\mathbf{x}_k}{\|\mathbf{x}_k\|}, \quad 1 < \mu < 2$$

(with the bias absorbed). Select the next value of  $k$  and return to Step 2.



- Is it possible that while learning a new input-output pair the neuron might forget a previously assimilated pair? If so who does one handle this unfortunate even?
- What will happen if one attempts to learn an input-output set that is not linearly separable?



Prove (no iterating the error correcting learning rule) that it is impossible to solve either of the Boolean AND and OR functions with a 2-input perceptron with no bias, i.e.  $\mathbf{w}, \mathbf{x} \in \mathbf{R}^2$  and  $\theta \equiv 0$ . Hint: a.) Overlay all input patterns onto the pattern space and consider an arbitrary decision boundary of the form  $\mathbf{w}^T \mathbf{x} = 0$  in the pattern space b.) Use levels +1 and -1 as opposed to 1 and 0.) What is causing the difficulty? Recommend small changes to the problem so that two weights would be sufficient.



- Choosing  $\mathbf{w}_0^T = (1, 0 \ 0)$  and  $\mu = \frac{3}{2}$ , enumerate analytically the sequence of weight vectors produced while attempting in an attempt to learn the Boolean AND function respectively by manual calculation. Plot the final decision boundary. (The hints directly above applies here.)
- Repeat these this exercise for the Boolean OR function.

### LR3. Gradient Descent Learning

TBD

### LR3. Hebbian Learning

TBD

### LR4. Competitive Learning

TBD

### LR5. Boltzmann Learning

TBD

### Credit Assignment Learning

TBD

## Learning Tasks

Neurons and neural networks learning tasks are (Haykin, 1994):

### LT1. Approximation

Suppose a given set of (noisy) input-output pairs  $\{(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \dots, (\mathbf{x}_N, d_N)\}$  are generated by the non-linear functional relation  $d = g(\mathbf{x})$ . Supervised learning is then expected to produce a neural network approximation  $\hat{g}(\mathbf{x})$  to  $g(\mathbf{x})$  within a specified accuracy.

*Universal Approximation Theory* (Hornik, Stinchcombe, White, 1989)

For any given real continuous function  $g: U \rightarrow \mathbf{R}$  defined on a compact subset  $U \subset \mathbf{R}^n$  and arbitrary  $\varepsilon > 0$  there a neural network approximator  $\hat{g}: U \rightarrow \mathbf{R}$  such that

$$\sup_{\mathbf{x} \in U} |\hat{g}(\mathbf{x}) - g(\mathbf{x})| < \varepsilon.$$



Then, given the an arbitrary input  $\mathbf{x}$ , the neural network is expected to produce the approximation  $\hat{g}(\mathbf{x})$  to within the required accuracy  $\varepsilon > 0$ .

### LT2. Association

*Auto-association* – requires a neural network to store (that is, learn or memorise) a given set  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$  of patterns (vectors) using unsupervised learning. Later, upon being presented with a distorted (i.e. noisy or even partial) version of any of the memorised patterns, the neural network is expected to retrieve or recall the respective memorised pattern.

*Hetero-association* – requires a neural network to store (learn or memorise) associations between one set  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$  of (input) patterns and another totally different set  $\{\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_N\}$  of (output) patterns, generally of different dimension. Later, upon being presented with a distorted (i.e. noisy or even partial) version of any of the input patterns, the neural network is expected to retrieve or recall the respective associated output pattern.

Pattern matching and pattern completion are instances of association.



What is the difference between a mapping and an association in the context of neural networks?

### LT3. Classification





Given a set of input-category pairs  $\{(\mathbf{x}_1, c_1), (\mathbf{x}_2, c_2), \dots, (\mathbf{x}_N, c_N)\}$  generated from a fixed number of classes, the neural network is required to deduce the decision boundaries for the different classes using supervised learning and then, when presented with an input pattern  $\mathbf{x}$ , the neural network is expected to correctly categorise it. In the case when the classes are unknown unsupervised learning can be used to cluster input patterns in order to discover the implicit classes.

#### **LT4. Data Mining**

The objective is the discovery of hidden patterns buried in large amounts of data. This intersects with the fields of Big Data and Complex Networks.

#### **LT4. Prediction and Control**

The neural network is expected to be able to predict the next event given the history of some process. If the future event is related to past events according to some mathematical relationship then prediction is simply a special case of LT1. In the case of a general causal relationship the neural network can discover these causal relationships using unsupervised learning.

Icon	Meaning
	Drawing your attention to relevant facts.
	Challenge to help broaden your views and understanding.
	Indicates a tutorial problem to be attempted during a tutorial session.
	Homework problems to be attempted in your own time.