Chapter 3

The artificial neuron: Perceptrons

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of the ANN that we call a Perceptron
- carry out simple hand simulations of Perceptrons
- manipulate the equations defining the behaviour of Perceptrons with step activations
- explain how Perceptrons can be thought of as modelling lines in the plan
- explain how simple Perceptrons (such as those implementing a **NOT**, **AND**, **NAND** and **OR** gates) can be designed
- discuss the limitations and possible applications of Perceptrons
- understand the behaviour of single threshold units with feedback
- understand the use of a single layer of Perceptrons to divide a plane into parts
- define the terms: **threshold units**, **step units, step activation**, **extended truth table**, **clamped**, **threshold**, **bipolar activation** and **recurrent**.

Essential reading

Rojas, Chapter 3.

3.1 Introduction: Single units

In this chapter we will walk you slowly through some very simple ANNs consisting of just one simple unit. However, as we will see, even a single unit can be very powerful (although later we will see that one unit may often not be powerful enough). Following other authors we use the term 'Perceptron' for a single unit, giving its details when necessary. Historically, a Perceptron was a particular type of unit but we prefer to follow the common usage.

Units with just this simple step activation function are surprisingly powerful when combined as we shall soon see, but for now let us see what a lone unit can do!

One of the main problems that we will repeatedly come across during our studies is that of knowing what a given ANN actually does. Another problem is the 'inverse' of this – being able to build a network to do what we want it to do. We shall see that in a few cases this is no problem but in the majority of circumstances both of these are very difficult.

3.2 Units with binary inputs and step activation

3.2.1 One or two inputs

To make our calculations easier we will for the time being restrict ourselves to looking at a single unit whose inputs are all binary and whose activation function is the threshold (*T(<0,0,1)*) given above, so that the outputs are also either *0* or *1*. We call units with threshold activations **threshold units**. You may also see them called **step units**, as a step is another way of describing a threshold.

A simple example is shown in Figure 3.1 below.

Inputs	Weights	Parameters	Form	Value
0	*bias*	Learning rate	η	**??**
?a	?	Net	Σ	??
		Activation	T(<0,0,1)	??

Figure 3.1: A simple unit with no bias.

Notice that we have set the 'input' at the bias to zero. This means that the bias has no effect on the calculations as whatever its value the result of multiplying by zero will be zero. We say 'no bias' when the bias has no effect. Also notice that there is just one input, *?a*.

Although this is a very simple network, it allows us to introduce the concept of an 'extended' truth table. An **extended truth table** is a truth table that has entries that evaluate to *0* or *1* but these entries could be variables or even expressions. Because all the inputs are binary, we can use an extended truth table to show how inputs map to outputs.

Here is the extended truth table for the single input unit:

?a	*Net Σ*	*Activation* *a = A(Σ) = T (0,0,1)(Σ)*
0	*0*	*1*
1	*w*	*if w < 0 then 0 else 1*

Figure 3.2: An extended truth table.

Notice that we have used the 'value' of *net* (ie *1* times the weight when the input is *1*). Also notice that we have spelt out the form of activation function.

Although not very exciting, this truth table shows us that, by making *w* positive, we can make the output always *1* and by making *w* negative we can make the unit output the opposite to its input. This possibility provides us with a way of building a **NOT gate**, to use the Boolean name for a device whose binary output is opposite to its binary input.

That was too easy, so let us now turn to two inputs as in the following diagram, Figure 3.3.

Inputs	Weights	Parameters	Form	Value
0	*bias*	Learning rate	h	**??**
?a	?	Net		??
?b	?	Activation	T(<0, 0, 1)	??

Figure 3.3: A two-input threshold unit.

Again drawing an extended truth table:

?a	*?b*	*net S*	*Activation a = A(net)*
0	*0*	*0*	*1*
0	*1*	w_b	*if $w_b < 0$ then 0 else 1*
1	*0*	w_a	*if $w_a < 0$ then 0 else 1*
1	*1*	$w_a + w_b$	*if $w_a + w_b < 0$ then 0 else 1*

Figure 3.4: An extended truth table for the unit of Figure 3.3.

As you can see things are beginning to get complicated!

Exercise 2

a. Rewrite the table in Figure 3.4 using values *(1, 1), (1, –1), (–1, 1)* and *(–1, –1)* for (w_a, w_b).

b. Rewrite the table in Figure 3.4 using values *(0, 1)* and *(1, 0)* for (w_a, w_b).

c. Do you recognise any of the resulting truth tables?

Comment: One thing that is worth noting here is that if a weight is zero, then the node's behaviour is independent of the corresponding input.

3.2.2 Three or more inputs

It is important that you have some facility with working out the outputs, given inputs and weights (examination questions often ask you to do this), so let us look at some more examples. The diagram below represents a unit with three or more inputs. The 'dotted' input '…' represents 0 or more edges, allowing for an unspecified number of extra inputs.

Inputs	**Weights**	**Parameters**	**Form**	**Value**
1	*bias*	*Learning rate*	η	*0.15*
?a	*?*	*Net*	Σ	*??*
?b	*?*	*Activation*	*T(<0,0,1)*	*??*
…	…	…	…	…
?N	*?*			

Figure 3.5: An N input threshold unit.

We use capital *N* for the number of inputs. When we introduced the notation of the Figures, we said that it is often useful to add to the external inputs a special 'input' which is fixed at the value 1. We called the weight associated with this input the ***bias*** of the unit and now we are including it in the discussion. Previously we set the input to *0* but now it is **clamped**, that is fixed at *1*. Notice that the bias does not count in the *N* inputs – one often finds it thought of as input *0* but, unfortunately, it is also called input *N+1* by some authors.

Although dealing with such units will in general require a computer to keep track of *net* and *activation*, we will look to see if we say something about the output if the values of all but one of the weights are the same. Let us call the value of the common weight *w.*

In this case the value of *net* is the value of *bias* plus *w* times the sum of the other inputs. We can write this as:

$$net = bias + w\Sigma_I^N ?_i$$

Here we have used $?_i$ to denote the ith input – ? reminding us that it is an input.

The equation shows that *bias* indeed acts as a bias by giving the rest of the sum a head start.

To make life a little more complicated the bias, or at least an equivalent of it, has again for historical reasons, been given another name, threshold, or rather bias is 'minus threshold'.

Suppose that we put this sum into the step activation function. We get:

$$a = [if\ (bias + w\Sigma_1^N\ ?_i) < 0\ then\ 0\ else\ 1]$$

This is, of course the same as saying:

$$a = [if\ w\Sigma_1^N\ ?_i < -bias\ then\ 0\ else\ 1]$$

Now we can see that *–bias* is acting as a (variable) threshold that the rest of the sum must equal or exceed before the output can become 1. When you read around the subject you will see **threshold** being mentioned, and you now know that this is just minus the bias, which in turn is the name of a weight connected to an input that is always 1.

Before we move on, let us look at what we can make with units of the type where

$$net = bias + w\Sigma_1^N\ ?_i$$

To see what we can do let us simplify the equation a little by using the symbol *M* to represent the number of inputs with value 1. Also if we let *bias* be any real number, rather than just a whole number, then we can write our equation as:

$$net = bias + wM$$

We have been able to do this because the inputs are either *0* or *1* and all the weights are the same.

Remember that the activation is *1* if *net* is greater than or equal to zero, so we can write the condition for our unit to have activation *1* as:

$$net = bias + wM \geq 0$$

or as

$$M \geq -bias/w \text{ and } bias \geq -wM$$

These formulae give us a means of designing units which, in order to output a 1 on firing:

a. require all inputs to be a 1 (that is an **AND gate**) by setting $w = 1$ and $bias = –N$ (the number of inputs)
b. require at least one input to be 1, by setting $w = 1$ and $bias = –1$ (this is an **inclusive OR gate**)
c. require at least a certain number of inputs to be 1, again setting $w = 1$ and *bias = – the number of inputs that we want to be on*. This represents a sort of 'voting' circuit
d. require at most *M* inputs to be 1 by setting $w = –1$ and $bias = M$
e. require at least one of the inputs not to be 1 by setting $w = –1$ and $bias = N – 1$ (this is the **NAND gate**).

You can see that just by adjusting the weights (including the *bias*) we can design a number of useful units. In fact we know from Boolean algebra that by combining a number of NAND gates (e. above) we can make any Boolean function that we want.

This last result might suggest that there is nothing else to do – but that would be an incorrect conclusion. All that it implies is that we can build an ANN to compute any 'computable function' so that they correspond to universal computing devices. Finding the required weights is another, much harder, question. We will see how weight can be found by the use of learning algorithms that train the networks.

3.2.3 A unit as a line in the plane

Let us now turn to the limitations of single units of this type, where we no longer insist that the weights are the same. Weights, inputs and bias are now arbitrary real numbers. We are going to do this by giving another way of looking at the calculation implied by the equation:

$$a = [if\ (bias + \Sigma_1^N w_i ?_i) < 0\ then\ 0\ else\ 1]$$

Note that $w_i ?_i$ above stands for weight w_i times input $?_i$. The argument, though not the diagrams of lines in the plane below, works with any number of inputs – that is values for N, but to make our diagrams easy to imagine and draw we will take it to be just 2. Instead of *?a* and *?b* we shall use the letters *x* and *y* – for reasons that you will soon see. We will use *v* and *w* for the weights corresponding to *x* and *y* respectively.

With this notation our equation becomes:

$$a = if\ (bias + vx + wy) < 0\ then\ 0\ else\ 1$$

From co-ordinate geometry we may know that the inequality $(bias + vx + wy) < 0$ represents a 'half plane' determined by the line $(bias + vx + wy) = 0$ and the sign of *bias*. For an easy 'trick' to find out which side of the line corresponds to an activation of 1, just substitute $x = 0$ and $y = 0$ into the equation. This gives the activation at the origin which turns out to be the same as the bias. So if the bias is < 0, the origin has activation of zero and if the bias is positive the activation at the origin is 1. (This link with co-ordinate geometry is the reason we are using *x* and *y* notation here, as you have probably realised.)

Exercise 3

On a single piece of graph paper plot the line: $(bias + vx + wy) = 0$ for the values of *bias*, *v* and *w* given in the following table:

bias	*v*	*w*
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Figure 3.6: Simple parameters for showing parameters of two-input threshold units.

For each line, mark which half plane represents the *activation* being 1 – you might like to use different colours for the lines to help keep tabs on which half plane belongs to which line.

Of course you could use Excel or another package to draw the lines for you – but you cannot do this in an examination so it is important to practise doing it by hand too.

We have included a simple line drawing Excel spreadsheet on the CD-ROM, which also has the solution to this exercise. Take a look at this spreadsheet now.

Exercise 4

In much of this section we have used the activation *A(net) = if net < 0 then 0 else 1*.

How would the results differ if we used: *A(net) = if net ≤ 0 then 0 else 1* instead?

(You should find this a seemingly small but very significant difference.)

3.2.4 Drawing a line in a plane

How about the inverse of this: given a straight line graph, can we build a unit that separates the plane along the line?

Suppose that we have a line given by $y = mx + c$. We can see that this can be written as $mx – y + c = 0$ so that setting $bias = c$, $v = m$ and $w = –1$ will provide the required weights. Not all straight lines, however, can be written as $y = mx + c$: for example, a vertical line cannot be so written. However, it can be written in the form represented by units. The vertical line which goes through $x = c$ and can be written as $c – x + 0y = 0$, that is $bias = c$, $v = –1$ and $w = 0$.

We have now seen that any straight line in the plane can be represented by a unit and that any two-input (plus bias) unit represents a straight line. If we have more inputs then we must work in 'higher dimensions' with such units representing and being represented by 'hyperplanes'. You will read about these in the reading associated with this topic which is given at the end of this section.

There are a few 'loose ends' we need to tie up to complete our discussions on single units.

Firstly, we note that if you multiply the equation of a straight line by any non-zero number it still represents the same line – so strictly speaking a line is represented by a family of units rather than a unique one. For example, the line $y = mx + c$ is exactly the same line as $7y = 7mx + 7c$. The unit with $bias = c$, $v = m$ and $w = –1$ represents the same line as that with $bias = 7c$, $v = 7m$ and $w = –7$.

Next, we may want the activation to be one 'above the line' or to be one 'below the line'. The same line is involved, so the same family of units have to be used. However if you change the sign of *bias* (by, for example, multiplying the equation of the line by –1) you change the side of the line with *activation = 1*.

Finally, consider the truth table in Figure 3.7.

This truth table is that of the 'exclusive or' function **XOR**; that is, the Boolean function of two variables that is true when one or other, but not both, of its inputs are 1.

x	*y*	*Activation*
0	0	0

0	1	1
1	0	1
1	1	0

Figure 3.7: Truth table of an exclusive OR unit (had one existed).

If you mark these four points on a graph and try to find a straight line which separates the zeros from the ones, you will fail – there is no such line. This means that there is no single unit that can do this separation and so no single unit can implement this truth table.

This is not a contradiction with our earlier statement that a network can compute anything, for here is an XOR made with units:

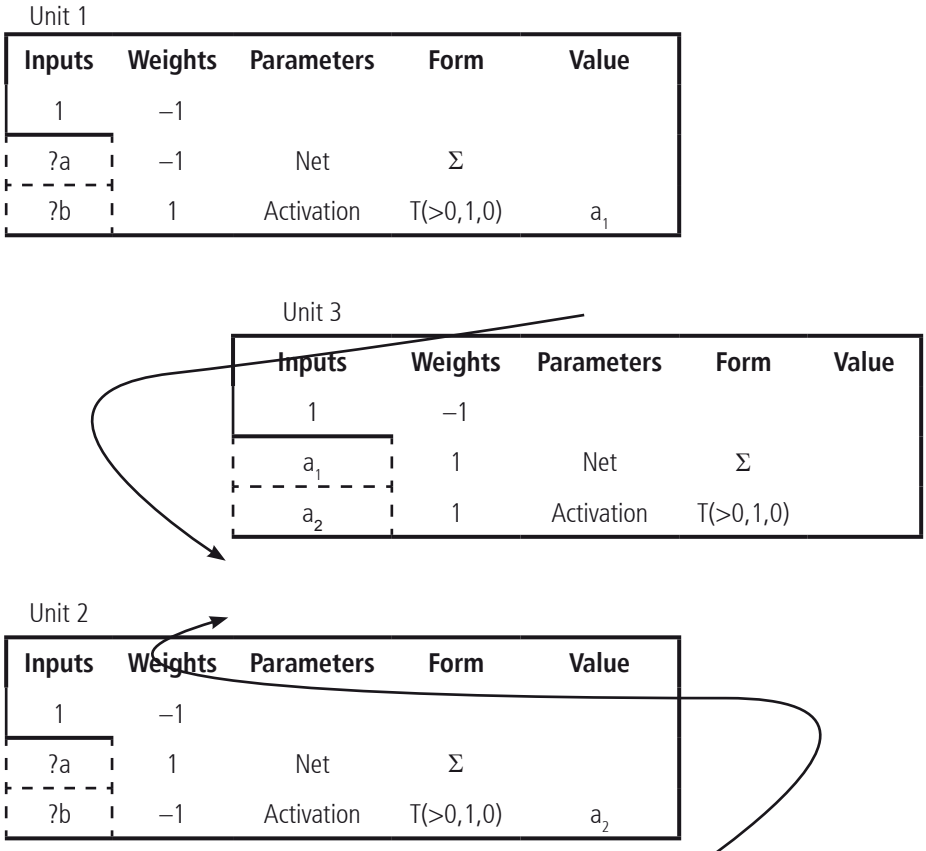

Figure 3.8: XOR made from units.

It was the fact that 'simple' Boolean functions such as XOR cannot be modelled by a single unit, mentioned in Minsky and Papert's book on Perceptrons (Minsky and Papert, 1969) that seems to have persuaded many in the 1970s not to develop neural networks further. They appear to have been mistaken.

Reading

Take a look at the first two chapters of Rojas. Skim any sections with deep mathematical derivations (there are just a few) but try to get a sense of what he is trying to say.

Now read pages 55–76 of Rojas, noting the following points as you do so:

- r3.1.1 makes an analogy between the way that the eye works and classical Perceptrons.
- r3.1.2 reports the limitations of this model.

- r3.2 covers much of what we have said in this chapter – using different notation and extending the materials – but the maths may be more difficult to follow.
- r3.3 extends this work to higher dimensions than two. Also note the idea of weight space is just a matter of treating different variables as dependent or independent.
- You can omit r3.3.3 and r3.3.4 at this stage but you may wish to read them later.
- r3.4 (omitting the maths) is an interesting account of how Perceptrons might be used to model vision.
- r3.4.1 shows how the process of detecting if a pixel belongs to an edge in a picture can be modelled using a Perceptron unit. A set of such units could be used to find all edge pixels in parallel.
- Read the account of the Laplacian operator in r3.4.2, although you do not need to know this.
- r3.4.3 and 4 continues the explanation of how a retina might be modelled.
- r3.5 summarises the history of these topics and is worth a quick read.

Exercise 5

Should we be surprised at proposition 6 of Rojas?

Learning activity

The CD-ROM that accompanies this guide contains a spreadsheet called **lines** that should help you understand and get a feel for the equivalence between Perceptrons and lines in the plane. Use the spreadsheet until you feel confident about designing Perceptrons that implement given lines.

3.3 A single unit with feedback

Up to now our units have had no feedback – that is there were no connections (direct or indirect) from a unit's output back to any of its inputs. Units and networks which have some sort of feedback are called **recurrent**. They are very useful in the right circumstances, but here we will just take a brief look at some complications they introduce.

First, let us look at units with step activation.

Consider a unit with just one input, *?a* say, whose output is connected to its input.

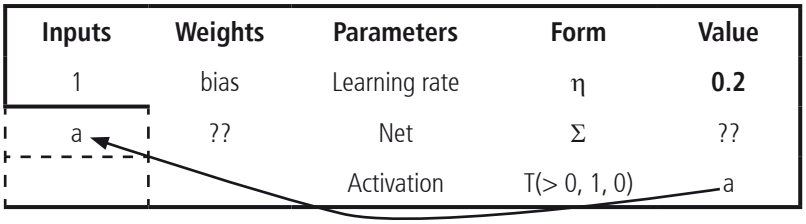

Figure 3.9: A single unit with feedback.

Notice that we are using *T(> 0,1,0)* as threshold. That is *a = if(net > 0 then 1 else 0).*

Our first complication is that the only way to influence this unit is by choice of weight, as its input is determined already by whatever its output happens to be.

To understand the behaviour of this unit we can draw a table of some possible bias, weight and input values.

bias	*?a*	weight	net	activation
–1	0	–1		
–1	0	0		
–1	0	1		
–1	1	–1		
0	0	–1		
0	0	0		
0	0	1		
0	1	–1		
1	0	–1		
1	0	0		
1	0	1		
1	1	–1		

Figure 3.10: How net and activation depend on bias, input and weight.

Exercise 6

Fill in the table above.

Note that sometimes the second and the last columns are not the same. But they should be the same, if the activation is supposed to be fed back to the input. Those rows where the entries are the same are stable, while those that have different entries under *?a* and *activation* are unstable. To make sense of these rows, we need to introduce the concept of time and of a delay between an input being set and an output being determined.

We have to introduce a concept of time and time steps so that the input may follow the output. If we do this, we might expect three possible types of behaviour: a) nothing changes – we have already seen this; b) the input changes in a predictable way; or c) the input changes in an unpredictable way.

In fact only a) and b) are possible with this setup.

Exercise 7

Using experimentation or algebra (or both), try to write down conditions for each of the possible behaviours.

Exercise 8

Repeat Exercises 6 and 7 above with bipolar and sigmoid activations. Are the results any different?

This is as far as we will go with recurrent networks for now. Later we will see just how recurrent networks can achieve useful performance.

3.4 Single layers of units

The section title is a little deceptive because a single layer of units is no more than a number of independent units – possibly sharing some inputs. A single layer of units, instead of producing just one activation as output, produces as many activations as there are units.

Nothing new is needed to understand what these achieve **but** something does emerge from the fact that they are working on the same inputs. Suppose that we have two two-input threshold units and thus two lines in a plane as shown in Figure 3.11.

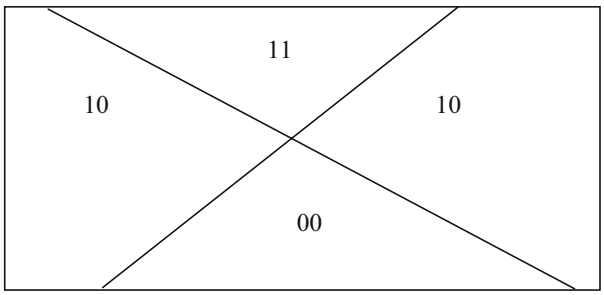

Figure 3.11: Two units dividing the plane into four classes.

Taken together the activations can be thought of as a binary number. If we take the line with a positive slope in the figure as belonging to the unit representing the most significant bit and the line with negative slope representing the least significant bit, then together the two units separate the plane into four regions labelled 00, 01, 10 and 11.

We now have the potential to separate all two digit binary numbers.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of the ANN that we call a Perceptron
- carry out simple hand simulations of Perceptrons
- manipulate the equations defining the behaviour of Perceptrons with step activations
- explain how Perceptrons can be thought of as modelling lines in the plan
- explain how simple Perceptrons (such as those implementing a **NOT, AND, NAND and OR** gates) can be designed
- discuss the limitations and possible applications of Perceptrons
- understand the behaviour of single threshold units with feedback
- understand the use of a single layer of Perceptrons to divide a plane into parts
- define the terms: **threshold units**, **step units**, **step activation**, **extended truth table**, **clamped**, **threshold**, **bipolar activation** and **recurrent**.