Chapter 6
Neural Circuits

A brief Introduction

This chapter on Neural Circuits is all about how the study of biological neurons inside our brains and linked to our senses and to our motor functions (arms, legs, heart etc.,) can inform how we create digital computer software and hardware artefacts. I have in mind physical artefacts such as legged robots which draw on parallels with real biological creatures, or image processing algorithms which draw on understanding of how the biological eye functions. There are many sorts of eyes, from the mammalian to the fly-eye which have different structures and purposes. There are many sorts of legged creatures (2, 4, 6, 8, lots) of legs and most of these biological creatures have been used to design physical robots.

Computers have been always linked to a study of the real world and in fact a simulation of real-world phenomena. Perhaps the earliest was the Greek Antikythera, discovered in the 20th century, but believed to be over 2000 years old. It was a mechanical device used to predict the positions of the sun, moon and planets, predict eclipses and (in retrospect) to determine the dates of ancient Olympic games.

Computers have been always linked with mathematics, and in particular solutions of 'Ordinary Differential Equations' (ODEs) which predict how things change with time, from the trajectories of wartime projectiles to the changes of stock market values, the beating of the heart and the synchronized flashing of fireflies.

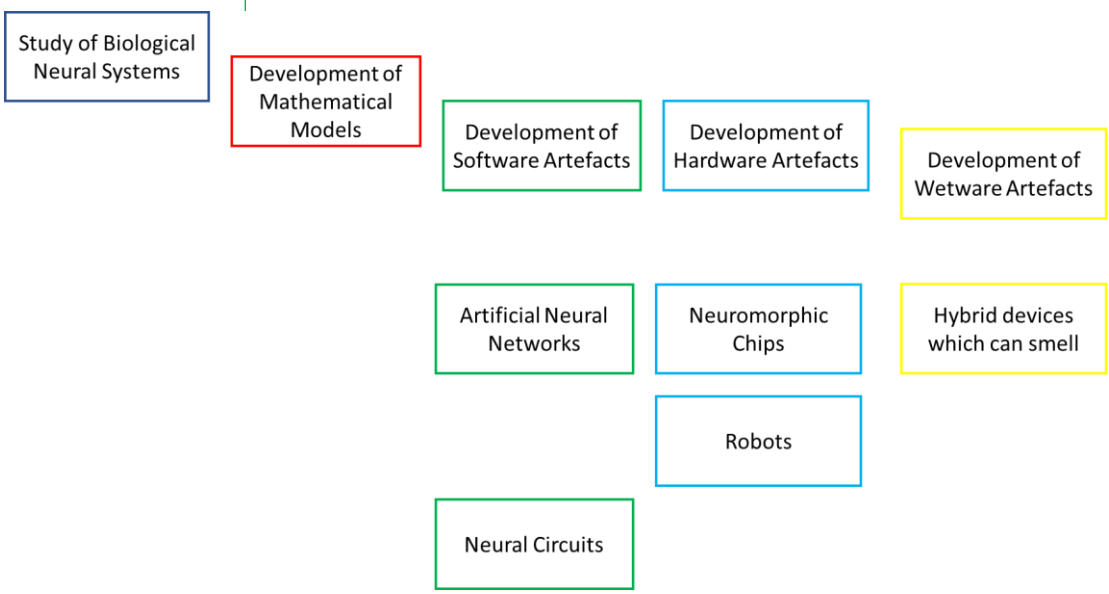
Computers have always been linked with Electronic Engineering; before our digital age, in the era of analogue computers machines contained circuits; electronic adders, multipliers and integrators (which solved ODEs) all linked

together with wires, where voltages represented variables, and controlled power stations, automotive control and more.

Computers have always been linked with cognitive psychology which attempts to understand how the mind works (even though no-one has proved that the mind is located in the brain). Of particular interest to me is how language comprehension and composition (story-writing) works.

Computers have always been linked with medicine, from the early stethoscopes through the development of digital radiography to contemporary functional Nuclear Magnetic Resonance imaging (f-MRI).

You get the idea. Computing is more than spreadsheets, websites, databases, games and cyber; computing has fundamental roots in the nature of man's enquiry into reality. So how does all of this fit into our studies, and in particular the material presented in this and the following chapter? Perhaps a diagram may help.



Reading from top left and moving across, the whole process of scientific discovery, modelling and simulation and finally application development is depicted. We start (top left) with

a study of biological neural systems; this means observing sections of brain material, or *in vivo* experiments on animals or individual neural cells. Various Nobel Laureates such as Golgi and Cajal (1904) and Hubel and Weisel (1981) showed that neural systems are not just goo, they comprise individual elements called *neurons*, and these worked by processing electrical signals. This is truly fundamental; our brains work by electronics, and they have objects (neurons) which we can study!

Mathematical models of single neurons, and populations of neurons were developed by a huge number of research groups. These continue to be developed as new results come in from biological research. Mathematical models are important, since they can lead to the engineering of technological artefacts (software and hardware). These models can be crafted to simplify the complex nature of wet biology. Models can abstract out the annoying detail and give us a tractable (understandable) approach. Broadly speaking there are two flavours of models. The easiest one to understand sees the neuron as a 'leaky integrator' which responds to input values with a smoothly varying output response (Fig.1). The second model is perhaps closer to the biological and models the 'spikes' emitted by real neurons (Fig.2). This is currently the subject of active research.

Once the scientist and mathematicians have developed their mathematical models, the *engineers* can take over, and develop artefacts, and make money.

Let's think about *hardware* artefacts, known today as 'Neuromorphic Chips'. These are CPU chips which have a totally different architecture from the sequential, synchronously clocked processor or the parallel processing chips, both which separate processor from memory. Intel's Loihi-2 neuromorphic chip has 1 million neurons which can make 120 million connexions with other neurons, and of course these connexions can change strength, and finally it uses spiking neural architecture. There are a few applications of this technology.

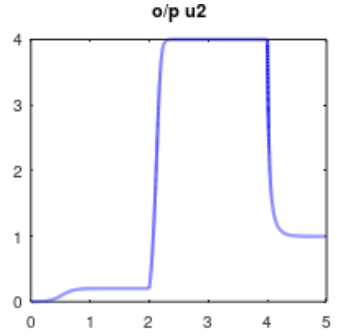


Figure 1 Leaky Integrator neuron. Reponse changes smoothly with time.

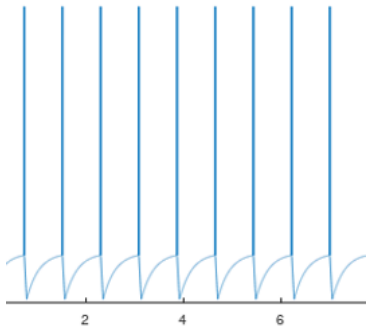


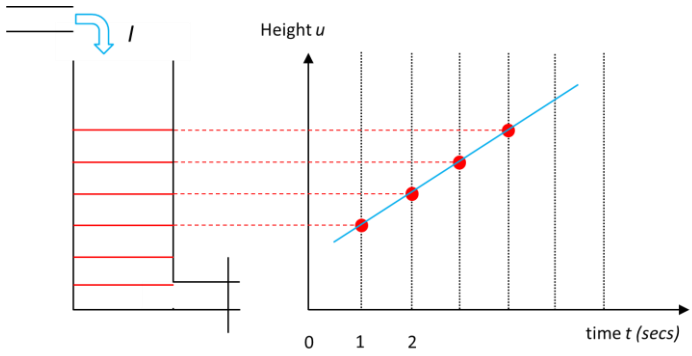
Figure 2 Spiking neuron model. Information is encoded as frequency of spikes.

Perhaps the most exciting development in this arena is the creating of new wetware artefacts. Here we have come full circle, and biological neurons are grown onto a silicon substrate to obtain some desired processing. Applications to odour detection have been made.

The Leaky Integrator Neuron

There are two important concepts you need to grasp. When a neuron receives an input, its internal state *changes*. Eventually this change stops, the state converges to an *equilibrium value* which is the final 'output' of the neuron. Let's take a physical example, a leaky water bucket.

Look at the diagram below where water is entering the bucket at a constant rate *I* and the leak is plugged.



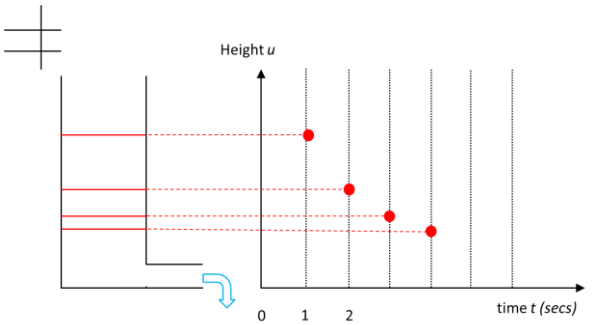
The height of the water is shown increasing with time. Since the water is coming in at a constant rate, then in each second the additional height is the same, so the height u rises on a straight line. We can write this mathematically like this, where the symbol Δu means change in height and the symbol Δu means change in time

$$\frac{\Delta u}{\Delta t} = I \qquad (1)$$

The left side of this expression is the rate of change of height due to the water coming in. The left side tells us 'the height of water is changing'. The right side *I* is the water flow. This tells us '*how* the water height is changing'. A larger *I* means

that more water is coming in per second, so the rate of change of height (the left side) will be larger.

Now suppose the bucket is nearly full so we turn off the water supply, and open the tap (leak) at the bottom. Here's how the water level will change



Of course, the water level falls, but not in a straight line. When the bucket is full, there is more water pushing down on the water in the pipe, so the flow through the pipe is larger. When the bucket is almost empty, there is little water pushing down on the water in the pipe, so the flow is reduced. A good model for the water rate of flow is proportional to the height, with a negative sign to show it's coming out,

$$-u$$

So the rate of change becomes

$$\frac{\Delta u}{\Delta t} = -u \qquad (2)$$

So what happens when we have water coming in *and* leaking out at the same time. The *net* water coming in is just

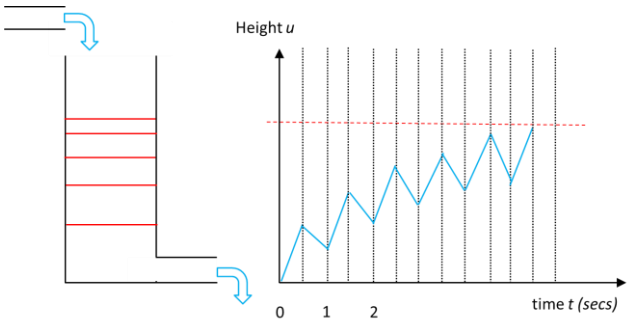
$$I - u$$

so the rate of change of water height is just

$$\frac{\Delta u}{\Delta t} = I - u \qquad (3)$$

A diagram may help. Imagine we let the water come in for a short time (with the leak closed) then stop the water coming

in and open the leak and repeat this rather strange process. The water height might look something like this.



The amount of water that enters in each half-second is the same, so the rise in height is the same. But since the water height increases, the drop in height due to the leak gets larger and larger. So the height follows a curve which becomes flatter and flatter until is horizontal. You can see this from expression (3),

$$\frac{\Delta u}{\Delta t} = 0 \quad when \quad I = u$$

This is the *equilibrium* state of the bucket, the water height is proportional to the flow into the bucket[1].

There are a couple more points we must deal with before this model is complete. First, in the above discussion we have been thinking in discrete time steps Δt whereas in reality the height change is *continuous* with time. When we come to actually solving neural equations by software, we shall use continuous time. This needs a change in notation so we replace Δt with a new symbol *dt.* Think of it like this, we make the change infinitesimally small,

$$\lim_{\Delta t \to 0} \Delta t = dt$$

so our expression (3) becomes

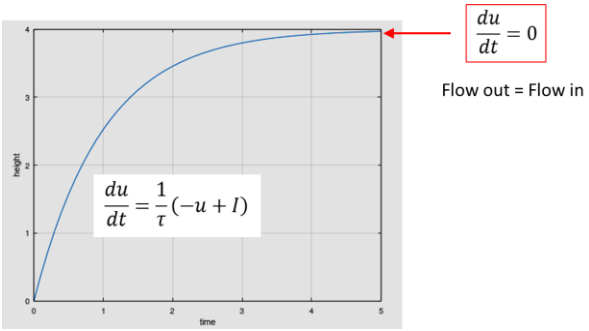
[1] The astute reader will note that we have chosen not to introduce various constants in order to keep the maths clean.

$$\frac{du}{dt} = I - u$$

The second point concerns the size of the cross-sectional area of the bucket, glance over at Fig.3. If water flows in are the same for both buckets, then the one on the left will fill faster than the one on the right. It is more responsive, we say it has a smaller 'time constant' τ (pronounced 'tau'). This is incorporated into the expression like this,

$$\frac{du}{dt} = \frac{1}{\tau}(-u + I) \qquad (4)$$

This is the final expression for the leaky bucket and is also the expression for all the leaky integrator neuros we shall use in this chapter. Let's summarise this as a graph.



Expression (4) is called a 'Ordinary Differential Equation' (ODE). We shall need one of these for each neuron in any circuit we construct. ODEs have application throughout science and engineering, and other disciplines such as business dynamics. Wherever something is changing, then and ODE solution springs to mind.

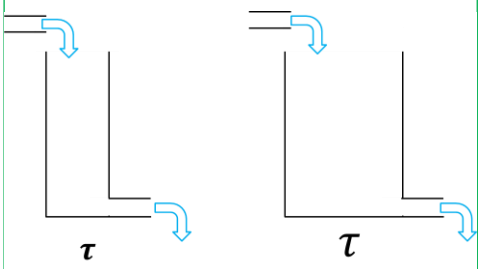


Figure 3 Buckets of different areas have different response times 'tau'

Some Basic Neural Circuits

Single Neuron with a Single Input

The neuron in Fig.4 has an input pulse of height I and it outputs its state $u_1$which changes with time. We know how to model this neuron, it's just our leaky integrator,

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 + I) \qquad (5)$$

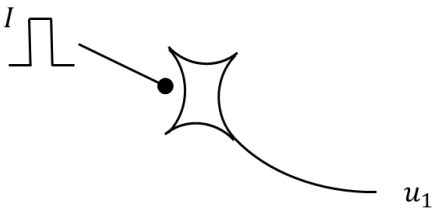


Figure 4 Single Neuron with a single input

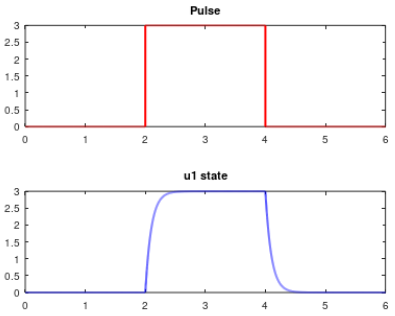


Figure 5 Single neuron response to input pulse of height 3

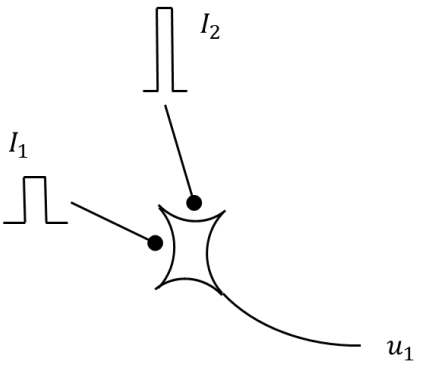


Figure 6 Neural circuit to add two variables

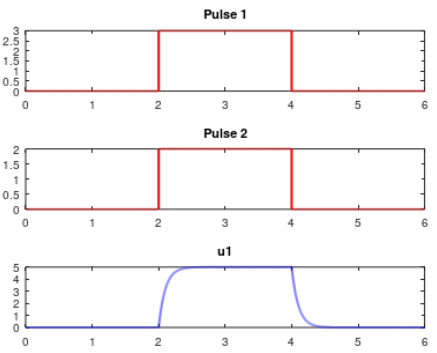


Figure 7 Adding two variables 3 and 2

with equilibrium solution

$$u_1(t \to \infty) = I$$

where we have made explicit that this happens as time approaches infinity. Fig.5 shows a typical solution for a pulse which rises to height *I=3* then falls to zero. The neuron state steadily rises to 3, then falls gracefully to zero, in both cases it reaches the input value. You could think of this neural circuit as assigning a value to a variable.

Neural Addition

The circuit shown in Fig.6 is able to add two inputs. The black circle is called a *synapse* and is where an input comes into a neuron. The output tail from the neuron is called its *axon.*

We can write the expression for the neuron's behaviour as

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 + I_1 + I_2) \qquad (6)$$

since we have two inputs! The equilibrium solution is obtained by setting $\frac{du_1}{dt} = 0$ which gives us

$$-u_1 + I_1 + I_2$$

i.e.,

$$u_1 = I_1 + I_2$$

so we have added the inputs. This addition process for variables of values 3 and 2 is shown in Fig.7

Neural Subtraction

So far we have assumed that the synapses (black) circles can be though of making the neuron's state increase with time (like water coming in). These biological synapses are called *excitatory*. There are also synapses which behave in the opposite way, these *inhibitory* synapses have a negative effect on the neuron's state increase, just like sucking water out with a pump. These can be used to perform subtraction as shown in Fig.8.

Again, it's straightforward to write down the ODE for this circuit, we have

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 + I_1 - I_2) \qquad (7)$$

with equilibrium solution

$$u_1 = I_1 - I_2$$

Neural Multiplication

Recent research in biology has shown that individual neurons are able to carry out multiplication. The arrangement of the inputs is slightly different. One input makes synaptic contact with the neuron as usual, but the second input makes contact with the first, just before it reaches the neuron. This is called a *shunting* effect. In terms of water pipes, you can think of one pipe increasing or decreasing the flow of water in the main pipe.

The ODE is again quite straightforward

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 + I_1 I_2) \qquad (8)$$

with equilibrium solution

$$u_1 = I_1 I_2$$

Applications to Autonomous Robots

Robot with steering

Here we have an autonomous robot which is driven by two rear wheels which rotate with the same speed, and turns by rotating the front driving wheel. The object is to build a neural circuit so the robot will turn towards the light. Taking inspiration from biological critters, we use two sensor neurons and two motor neurons. The problem is how to wire these up to obtain the desired behaviour.

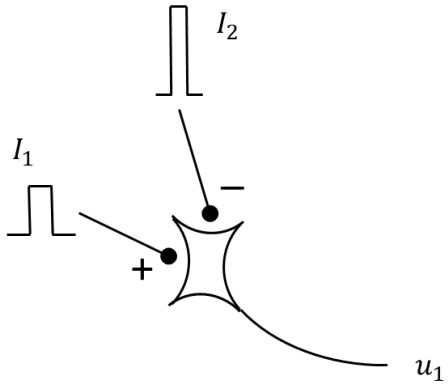


Figure 8 Neural Subtraction using inhibition on input-2

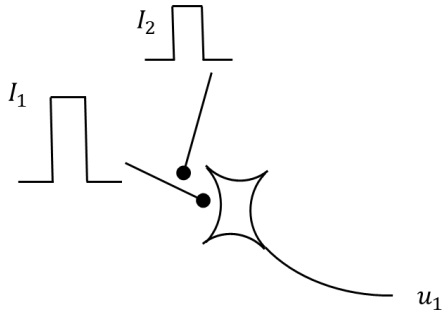


Figure 9 Neural multiplication using 'shunting'

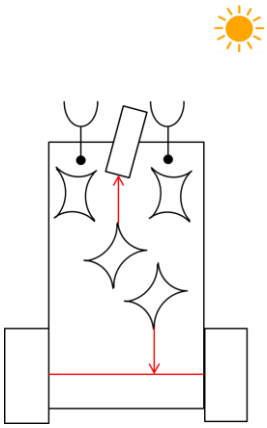
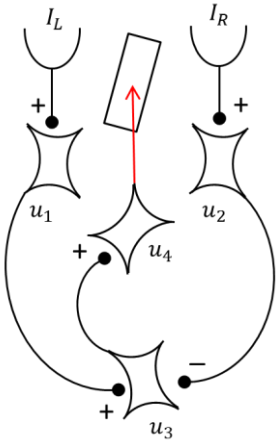
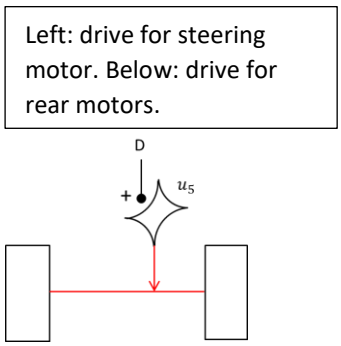


Figure 10 Robot with steering showing 2 sensory neurons and 2 motor neurons

It's easy to understand that the sensor (eye) nearest to the light will receive more light and therefore provide greater input to its associated neuron. Also if the light is in front at the centre then both these neurons will receive the same excitation, any difference in their outputs will indicate the light is to one side and the front wheel should turn. So, we need to calculate the difference and use this to drive the front motor like this.





For the input sensory neurons, we have

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 + I_L)$$

$$\frac{du_2}{dt} = \frac{1}{\tau}(-u_2 + I_R)$$

then find the difference

$$\frac{du_3}{dt} = \frac{1}{\tau}(-u_3 + u_1 - u_2)$$

then drive the steering motor neuron

$$\frac{du_4}{dt} = \frac{1}{\tau}(-u_4 + u_3)$$

and drive the rear wheel neuron

$$\frac{du_5}{dt} = \frac{1}{\tau}(-u_5 + D)$$

where D is a constant value to produce a steady forward speed. This circuit is probably over-complex you could effect a solution with just two neurons in total, but if nature had to solve this problem, then that's a probable solution. Working down these ODEs solving each for its equilibrium state we find

$$u_1 = I_L \qquad u_2 = I_R$$
$$u_3 = u_1 - u_2 \qquad = I_L - I_R$$
$$u_4 = u_3 = I_L - I_R$$

so indeed the motor drive u_4 is the difference between the sensor inputs, and can be positive and negative. Finally we must make the wheel turn in the correct direction. The solution of the motor driving the rear wheels is simply $u_5 = D$.

Braitenberg Vehicles

These are differential-drive robots introduced by Valentino Braitenberg as experiments in 'Synthetic Psychology'. Again we have two sensory and two motor neurons, and movement towards the light is achieved by cross-coupling left and right sensory and motor neurons, with exciting synapses, Fig. 11. The ODEs are straightforward

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 + I_L)$$
$$\frac{du_2}{dt} = \frac{1}{\tau}(-u_2 + I_R)$$

and for the motor neurons

$$\frac{du_3}{dt} = \frac{1}{\tau}(-u_3 + u_2)$$
$$\frac{du_4}{dt} = \frac{1}{\tau}(-u_4 + u_1)$$

with equilibrium solutions $u_3 = I_R$ and $u_4 = I_L$ which drive the wheels to make the robot turn correctly

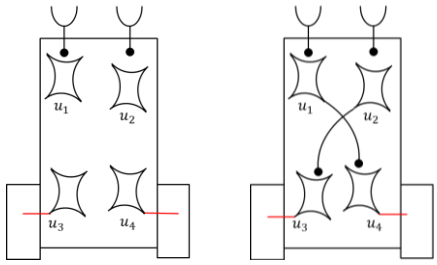
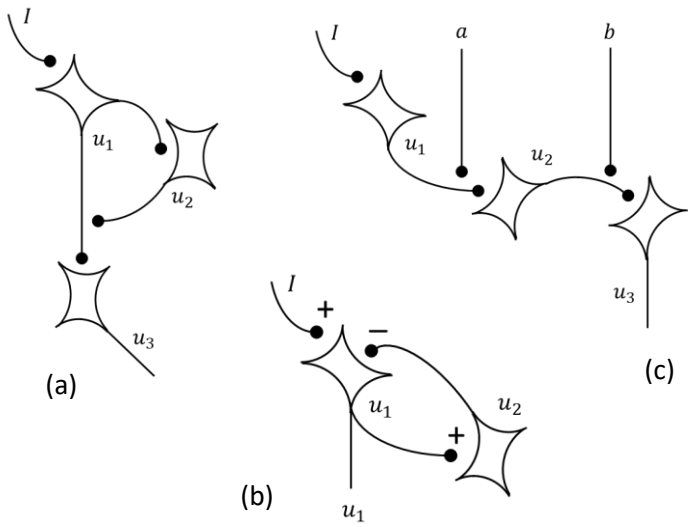


Figure 11 Braitenberg Vehicle Problem (left) and Solution (right).

A Jump Aside – Puzzles

Work out what these circuits do: output(s) in terms of input(s). Solutions from the author



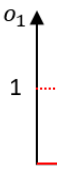


Figure 13 T

Enhancing the Leaky Integrator Model

So far the neuron has successfully performed some *linear* operations such as summing the input signals. But there is a better model of a neuron which has an additional processing stage following the summation. So, we can think of our enhanced neuron as having two parts. Fig.12 and we express this like this; the output of the i'th neuron is

$$o_i = f(u_i)$$

where the u_i is the internal state of the i'th neuron, just what we have been working with so far. This may not make too much sense (because it's abstract). So let's look at a concrete example of an output function

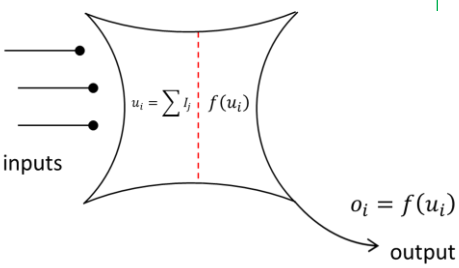


Figure 12 Enhanced neuron model with output function

Threshold Output Function

Here the output function takes the neuron state u_i and if it is above some threshold value θ which you can choose, then

the neuron output is 1, else it is zero. This function is shown in Fig.13. To understand this, let's take a worked example

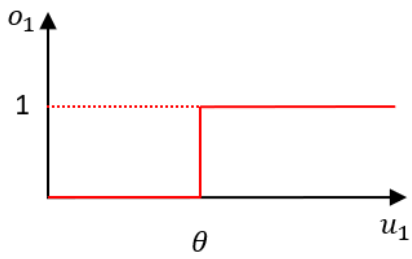
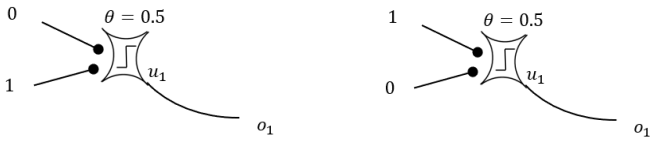
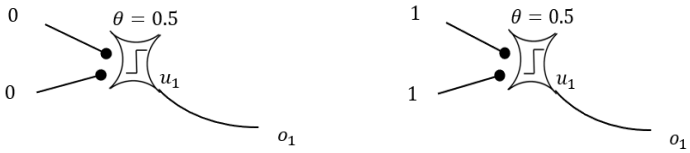


Figure 13 Threshold output function.

Let's assume that all the neuron levels are in the range between 0 and 1. So what do the following two circuits output when the threshold is set to 0.5?



Well in both cases the value of the neurons state is 1+0=1 and this is above the threshold value of 0.5, so we get an output of 1. Now think about the following two cases



In the first case the neuron state is 0+0=0 which is below the threshold, so the output is 0. In the second case the neuron state is 1+1=2 which is greater than the threshold, so the output is 1. Putting all this into a table we find

input 1	input 2	state	output
0	0	0	0
0	1	1	1
1	0	1	1
1	0	2	1

So this neuron functions as an OR-gate.

The NAND-Gate

All computer electronics (CPU and memory) can be built from a load of NAND-gates. If we can find a neural circuit for a NAND gate, then we can build an entire computer out of neurons. It's quite easy really.

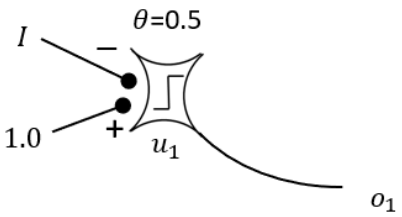


Figure 14 Neural NOT gate

First we create a neural AND gate. A bit of thought shows all we need is a two-input neuron with a threshold set to say 1.5 so that only an input (1,1) will raise the neuron's state above the threshold and output a 1. That's an AND gate. To get a NAND, we need to follow our AND by a NOT gate, so we must design a neural NOT gate or 'invertor'. Consider the single input neural circuit shown in Fig.14. If we design the neuron so its state follows this ODE

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 - I + 1)$$

then we find its equilibrium state is

$$u_1 = 1 - I$$

so if the input is 0 the output is 1 and if the input is 1 the output is zero. The threshold function will guarantee a nice clean output of exactly 1 or 0, the inverse of the input. We have a NOT gate. Therefore, we have a NAND gate.

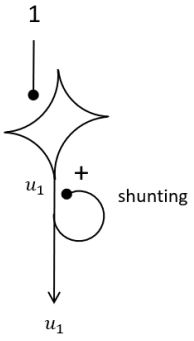
Since we have a NAND gate we have proved that all electronic circuits can be built from neurons.

Shunting feedback – A new type of neuron

Here we are going to invent a new type of *nonlinear* neuron. It demonstrates the power we have at creating new systems, but I must admit, there *is* biological support for our invention. We will *not* be using an output function in this case, the inherent nonlinearity will do this job for us.

The idea is shown in the diagram below. On the left we start with our simple linear neuron which has a constant input of 1.0, the usual ODE is shown. Now we take a leap of faith, and we multiply the output of this neuron, u_1 by itself, using the concept of shunting which we have seen earlier. This gives us a new ODE which has interesting solutions.

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 + 1)$$



$$\frac{du_1}{dt} = \frac{1}{\tau}u_1(-u_1 + 1)$$

You can see the inclusion of shunting; the bracket for the neuron on the left is multiplied by the value of the neuron's state u_1. The equilibrium solutions for our new neuron on the right are simply the solutions of this expression

$$u_1(-u_1 + 1) = 0$$

and there are two: $u_1 = 0$ and $u_1 = 1$. Logic levels again. But we are not complete since we must test the *stability* of these two solutions. That means setting $u_1$to a small value close to 0, and looking whether the circuit solution returns to 0, or diverges from it.

For $u_1 = 0.1$ we find that $du_1/dt = 0.09$, so the solution grows away from $u_1 = 0$. Unstable. Also for $u_1 = -0.1$ we find that $du_1/dt = -0.11$, so again the solution diverges from 0 (it becomes even more negative). So the solution 0 is *unstable* and the circuit will not converge to this solution.

Now for the other fixed point 1.0, we try dropping down to $u_1 = 0.9$ and here we find $du_1/dt = 0.09$ so the solution rises up back to 1.0, looking good. For $u_1 = 1.1$, just above 1.0 we find that $du_1/dt = -0.11$ so the solution returns back down to 1.0. Hence the solution 1.0 is *stable* and we expect the circuit to display this value at equilibrium.

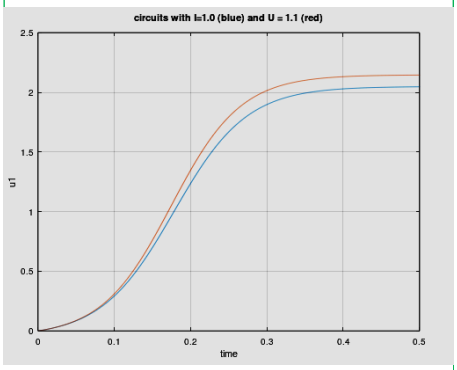


Figure 15 Response of the shunting circuit to to inputs: 1.0 (blue) and 1,1 (red)

Now let's develop this shunting circuit a little. First we need to add an input. The ODE now looks like this

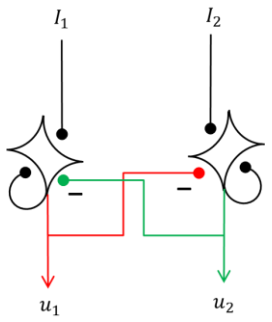
$$\frac{du_1}{dt} = \frac{1}{\tau} u_1(-u_1 + 1 + I_1)$$

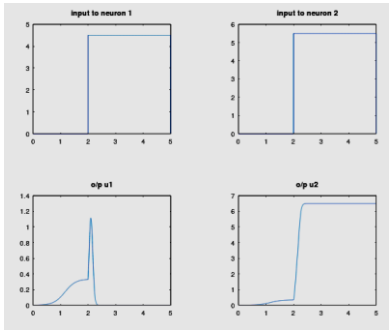
Let's see how this circuit responds to two different inputs, starting from a neural state u_1 close to zero. The results for two inputs, 1.0 and 1.1 are shown in Fig. 15. The results are unsurprising but important, the curve corresponding to the larger input always lies above the other curve and reaches a higher equilibrium value. This tells us that *the rate of increase* of the neural state is larger when the input is larger.

Application – Finding an array maximum

Here we shall use the above shunting circuit and develop an exciting application, a neural circuit which can find and output the maximum number in an array of numbers. The time taken to do this does not increase with the length of the array. This is an incredible result and cannot be achieved using procedural code.

Consider the circuit below with $I_1 = 4.5$ and $I_2 = 5.5$. The dynamics of the circuit is also shown





First the circuit; we have taken two of our shunting neurons and have coupled them together, through the colored lines. Note that the coupling is *inhibitory* and as we shall see soon it is *shunting inhibition.* We start with the input values, so what happens next? Both the values of u_1 and u_2 will

increase, but u_2 will rise faster than u_1. Since u_2 is larger its shunting inhibition (green line) on u_1 will be larger than the reciprocal inhibition of $u_1$on u_2 (red line). So u_2 (which was already rising faster) will rise even faster. This process continues until u_2 effectively kills (quenches) u_1. The circuit has selected the larger input!

Now let's have a look at the ODEs. To help out, a 'key' is provided in Fig.16 reminding us of the origin of each term. We have two 'symmetric' equations

$$\frac{du_1}{dt} = \frac{1}{\tau} u_1(-u_1 + 1 - ku_2 + I_1)$$

$$\frac{du_2}{dt} = \frac{1}{\tau} u_2(-u_2 + 1 - ku_1 + I_2)$$

If we assume that both neurons have identical initial values, $u_1(t=0) = u_1(t=0) = \epsilon$ where ϵ is a small number, then the only difference between them is the value of the neurons' inputs. This will establish their initial growth rates which, as discussed above will be different. The term ku_2, inhibition from neuron-2 into neuron 1 will be larger than the reciprocal term ku_1, inhibition from neuron-1 into neuron-2 if $u_2 > u_1$ which will be the case if $I_2 > I_1$, so neuron-1's growth will slow down faster than neuron-2's growth.

Eventually neuron-1 will be killed to zero, so the expression for neuron-2 will become

$$\frac{du_2}{dt} = \frac{1}{\tau} u_2(-u_2 + 1 + I_2)$$

and the equilibrium solution of this is simply

$$u_2 = 1 + I_2$$

in other words neuron-2 converges to a value equal to its input plus 1. So these two neurons form a 'winner takes all' circuit, which provides us with the largest input value (plus one).

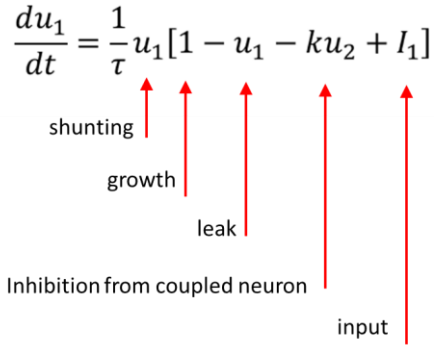
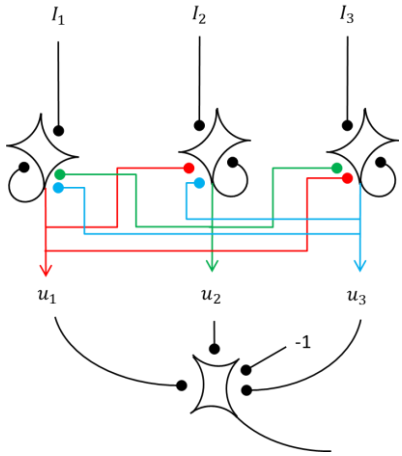
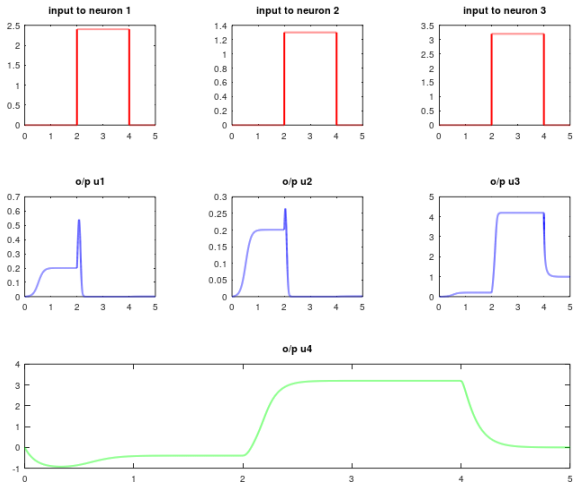


Figure 16 Meaning of individual terms

It's easy to generalize this to an array of N-elements. The circuit shown below for an array of 3 elements has one additional neuron to give us the maximum value in the array. It adds the outputs from the shunting inhibition layer (only one will be non-zero) and subtracts 1 to give us the maximum value in the input array.



The shunting layer will send the largest input value (plus 1) to the final neuron which subtracts 1 and outputs the maximum value. The diagram below shows the parallel neural processing for input values (2.4, 1.3, 3.2), the largest value 3.2 has been correctly output.



Sure there are some 'pathologies', e.g., if all inputs are the same value, then all outputs u_i are the same and are summed. It's fairly straightforward to show that for N neurons, the final output is

$$\left(\frac{I+1}{1+(N-1)k}\right)N-1$$

This is left as an exercise for the interested reader.

Procedural Programming with Neural Circuits

We have seen that computer hardware can be replaced with neural circuits, so we can build a computer out of *wetware*. Now comes my conjecture that procedural programming can be replaced by neural circuits. This means we have to show how these circuits can replace the three basic programming constructs: *sequence, selection* and *iteration.* Currently I have two out of the three, perhaps you can supply the missing link.

A Neural Selection Circuit

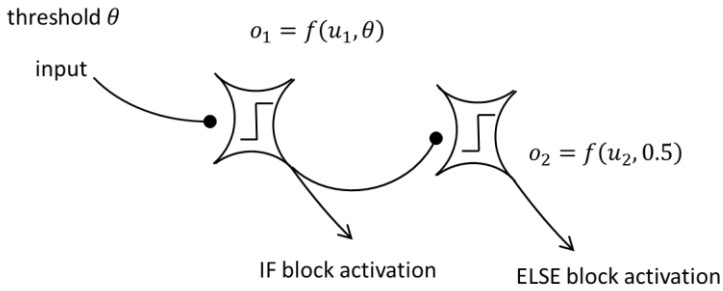
Here we shall see how to create the if-then-else construct. Take a simple example

```
if(input > threshold) {
  x = 6.0;
} else {
  x = 2.0;
}
```

There are two different things going on here. First we have the *logic* of selection or 'control flow' where the if statement has a condition whose outcome is *binary* , we either select the if-block or the else block. Then within each block we have a variable assignment. This is 'data flow'. We construct a neural circuit keeping these two dimensions of processing separate. Take the control flow. This is shown in the diagram below.

The input arrives at the first neuron which has a threshold set at **`threshold`**. So, the first neuron will output 1.0 when the

input is above the threshold, in other words it has generated the 'if' control signal, activating the 'if' block of code.



The equation for the first neuron state is simple

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 + input)$$

and the equation for the second neuron state is just our inverter

$$\frac{du_2}{dt} = \frac{1}{\tau}(-u_2 - o_1 + 1)$$

To complete the circuit, we need a third neuron to represent the value of the x-variable which is quite straightforward (Fig. 17).

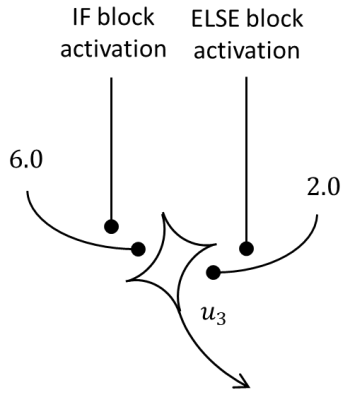


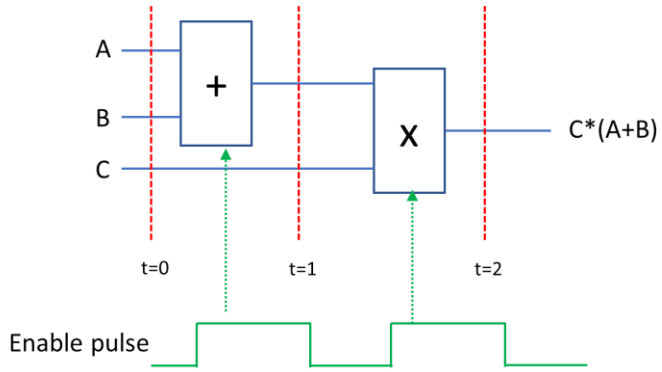
Figure 17 Assigning values to a variable based on if(condition)

Sequences of procedural statements

As neural circuit engineers we have considerable freedom in designing circuit which will produce sequences of pulses, based of course on biological neurons. Biology reveals to us several mechanisms of sequence generation; there are *neural oscillators* (which we shall meet in the next chapter), *spiking neurons* (which may be addressed in the future. But here we shall consider *chains of neuron delays*. Before we get into the details, let's have a look at the *concept* we are proposing, shown in the diagram below.

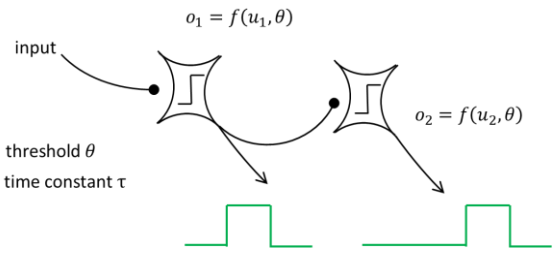
Let's consider a sequence of two maths operations; first we add two variables (A + B) and when sufficient time has

passed for this to complete, we multiply the result by a third variable C, so we end up with C x (A + B).



The two blue rectangles show processing blocks, an adder and a multiplier; we know how to do these using neural circuits, so imagine the circuits are inside the blocks. Now we know that neural processing takes time (remember 'tau') so the multiplication must be delayed until the addition is complete. That is the crux of the concept. The green enable pulse first enables ('switches on') the adder, and when addition is complete, it switches the adder off. Then, a little later it switches the multiplier on which does the second operation. So, we have a sequence.

The only question remaining is how to we produce the green pulse train? This is quite straightforward, we need a chain of threshold neurons, each will provide a delay and output a pulse like this



An example of the behaviour of a short chain is shown in Fig.18

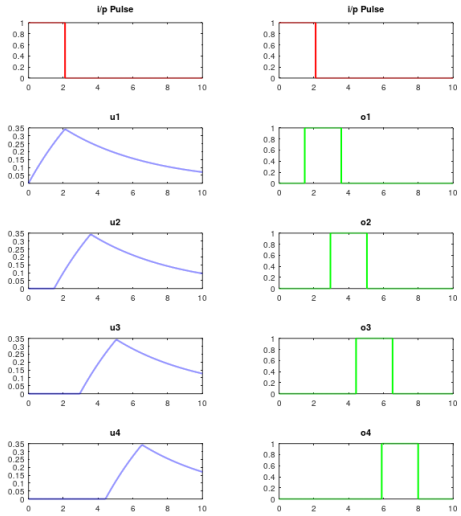


Figure 18 Pulse Sequence Generator: Red input to neural chain; blue neuron states; green thresholded outputs.

showing a clear series of pulses.

The whole process depends on choosing a correct time constant *and* a corresponding threshold. If we specify the threshold value τ and the length of the pulse we want T then it turns out the required threshold is

$$\theta = 1 - \frac{1}{2 - e^{-\frac{T}{\tau}}}$$

as shown in the appendix. The results shown in Fig.18 were produced for a pulse length of 2 secs and a time constant f 5 secs, with a computed threshold 0.255372.