Formal Computational Systems Report

An Investigation into a CTRNN Node

Mike Blow

04.12.03

Abstract

This report aims to provide an introduction to Continuous Time Recurrent Neural Network (CTRNN) nodes. It starts with a short overview of the node and its use in a neural network, and then analyses the node in an attempt to understand its behaviour.

What is a CTRNN Node?

To answer this fundamental question we should first briefly look at the idea of neural networks. A network is a computer architecture consisting of several interconnected processors running in parallel. One of these processors (or 'nodes') can be thought of as an artificial neuron, and when several nodes are connected together (in a way reminiscent of the connections in the human brain) they create a neural network. Each node produces an output that is some function of its summed inputs. Various types of node and network topographies have been created with the ability to adapt and learn.

A Continuous Time Recurrent Neural Network (CTRNN) uses nodes that include a time-dependent element or 'learning rate'. This is a crucial as it allows the network to be more than just reactionary and to create complex, oscillatory and even chaotic behaviour **[1]**. A CTRNN Node also has plausible biological analogies where y can be thought of as the membrane potential and $\sigma(x)$ is associated with its firing frequency.

The equation for a CTRNN Node is:

$$\frac{dx}{dt} = \frac{1}{\tau}(-x + w\sigma(x+\theta) + I)$$

where:
x is the output of the node
τ is 'tau', the learning rate
w is the weight of the connection from j
I is the external input
And **σ** is 'sigma' the sigmoid activation function:

$$\frac{1}{1+e^{-(x+\theta)}}$$

where:
θ is the bias

It can be useful to think of the equation as four sections:

[learning rate (τ)] [current output(x)] [connections (w σ (x+ θ))] [external input (I)]

Method

In this report I intend to investigate a CTRNN node in order understand what each variable does in an intuitive way. I will start by making the equation as simple as possible and then add terms so the effect of each can be easily seen.

In real neurons time is continuous, but to simulate them on a computer we have to discretize time. In order to do this we integrate the output of the neuron over small time steps. The smaller the steps, the better the approximation to continuous time. In practise the timestep is chosen small enough to capture the time-variant behaviour we are interested in. There is also an important relationship between 'h', the timestep, and 'τ', the leaning rate, as will be seen later.

Step 1: Basic behaviour

It can be seen from **(1)** that we can easily simplify the equation by making **τ = 1 and I = 0**. If we also assume to begin with that the node has no connection from itself or any other node the **w σ(y+ θ)** term disappears, and we are left with:

$$\frac{dx}{dt} = h * -x$$

Fig. 1 shows the output of this equation over time from starting values of 1, 0 and –1.

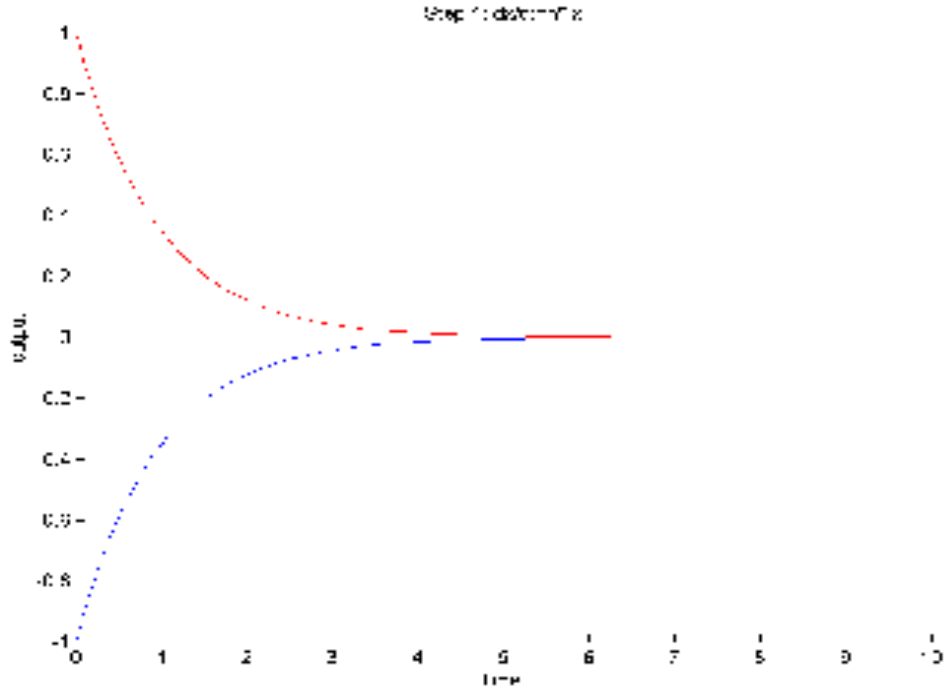


fig. 1 basic behaviour

This behaviour is easy to understand if we work through a timestep:

Say **x(t0)=10** and **h**, the timestep, = **0.1**
then the change in x with respect to time **(dx/dt) = h.-x(t)**
which in our case = **0.1*-10 = -1**
so **x(t+1) = 10-1 = 9**

for the next timestep, **(dx/dt) = h.-x(t+1)**
=0.1*9 = 0.9
x(t+2) = 9-0.9 = 8.1
etc.

and it can be seen that dx/dt reduces as x reduces giving the results shown. Obviously when x=0, **dx/dt = 0.1 * 0 = 0** so there is no change.

Step 2: τ, the learning rate

The next step is to add τ into the equation. *Fig. 2* shows the effect as τ is increased, and it can be seen that it affects the decay time of the system (note: the decay time is also the time taken to rise up to a steady state: see the '–1' example in *fig. 1*). If τ is small, the decay time is very quick and the system soon reaches its steady state. If τ is large the node will take much longer to respond. In practise this results in varying behaviours: a small τ gives a reactionary node that quickly responds to changes in its input, and large τ endows the node with a form of 'memory', as its current state is affected by its previous state.

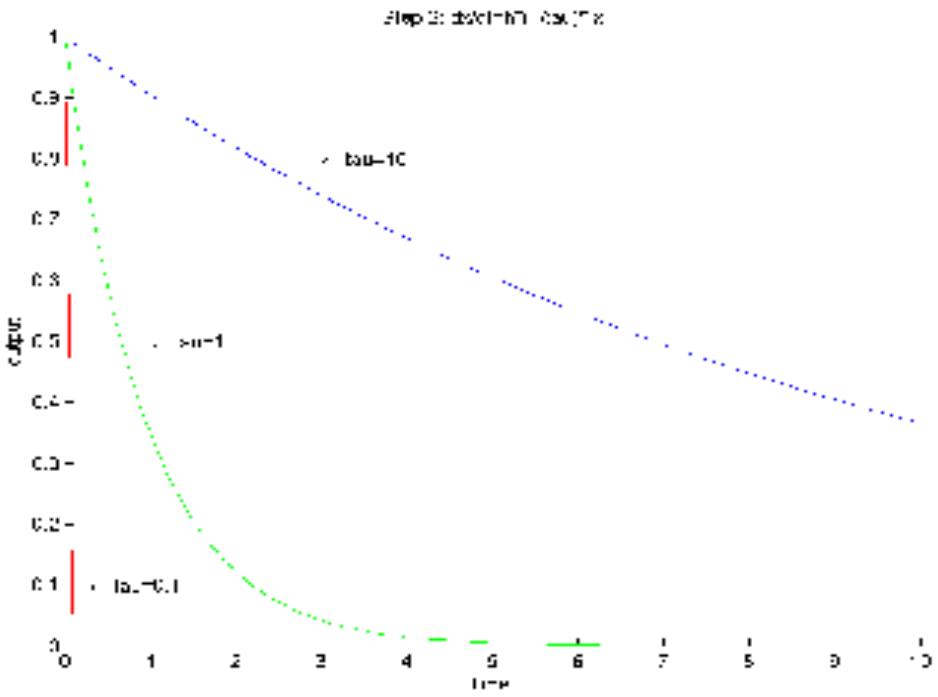


fig. 2 effect of tau

It is also worth investigating the relationship between τ and the timestep 'h'. H is the interval over which we approximate the behaviour of the system and has a marked effect on the output. Fig 2a shows three timestep values where **h <= τ**:

When **h = 0.01* τ** the output is a smooth decay curve.

At **h = 0.5* τ** the curve is still there but two detrimental effects are noticeable: the curve is no longer smooth, and it is less accurate. This is because as we increase h we increase the distance over which we are integrating (i.e. approximating the curve) with a consequent increase in error.

At **h = τ** we no longer have a curve: the output jumps straight from its initial value to its final value in one timestep.

Fig. 2b shows the effect of 3 values of h for **h > τ**. This is an interesting case because it leads to instability in the equation, where a tiny change in h can lead to totally different behaviour:

At **h = 1.9* τ** the output will oscillate but gradually die down to the steady state value.

h = 2* τ is a special case: here the system is at an unstable equilibrium point and in theory will oscillate at the same level forever. In real life unstable points like this will never persist as the slightest change in value will send the output away from the point; in this case, a negative perturbation will result in a decay to 0 and a positive one in increasing oscillations. This is also known as a 'bifurcation point'.

At just over 2, **h = 2.1* τ**, the oscillations start to increase in amplitude and the equation becomes unstable.

The important points are these: **to avoid instability in the output, h must be less than or equal to τ**, and **for accurate approximations of the continuous-time output of the system, h should be a small fraction of τ**. Thus the choice of 'h' is crucial.

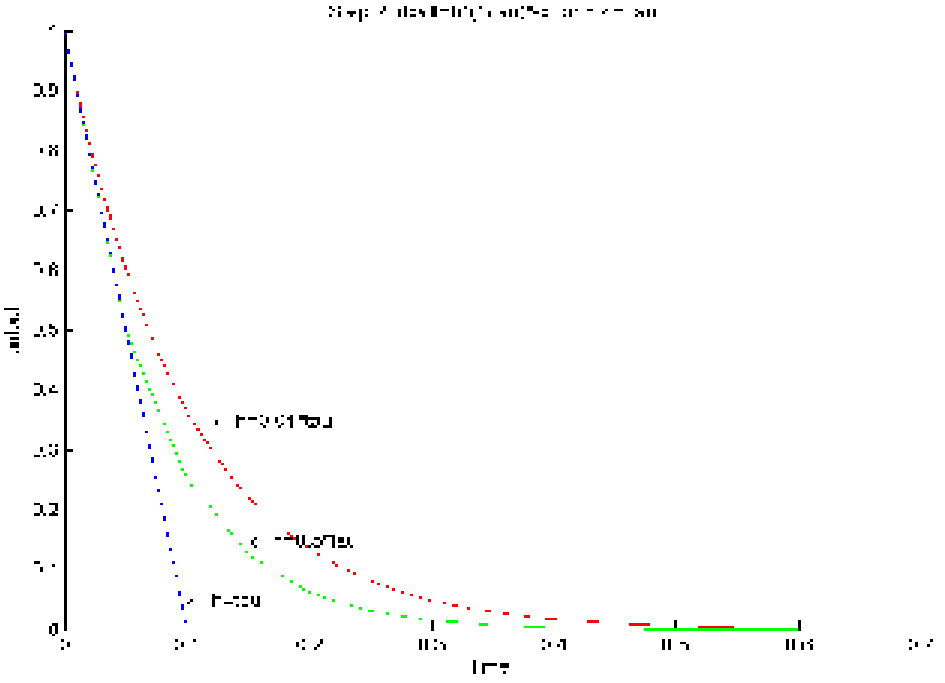


fig. 2a h <= tau

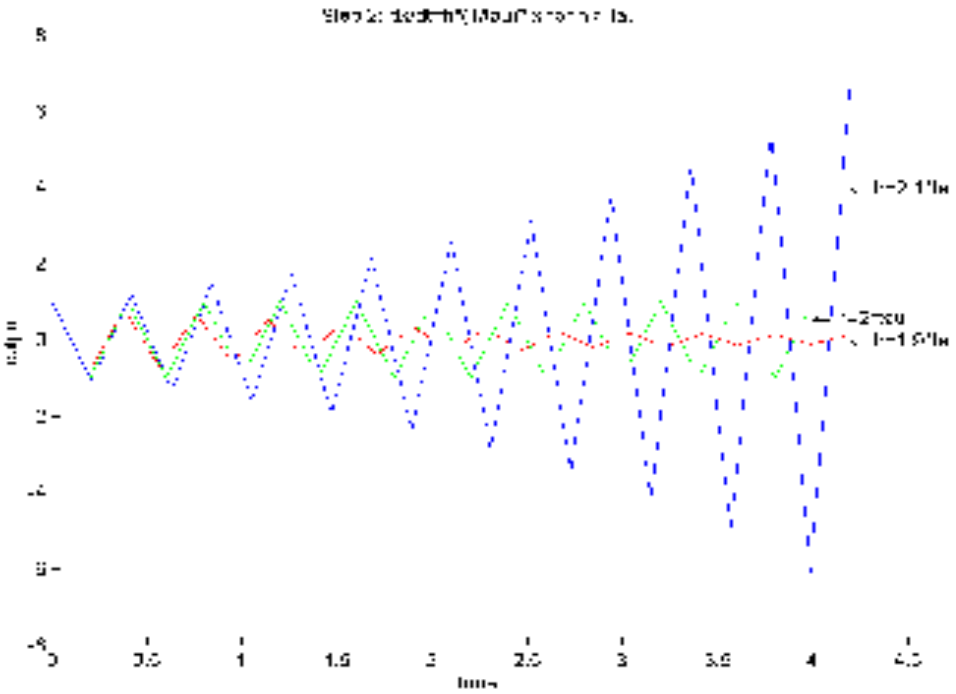


fig. 2b h > tau

Step 3: External input

In real life these networks will most likely have some form of input, for instance from a sensor on a robot, and this is represented in the equation by 'I'. *Fig. 3* shows the effect of three values of I on the behaviour of the system. *Fig. 3a* shows the effect of changing I from 5 to 0 at time = 10 for two different values of τ.

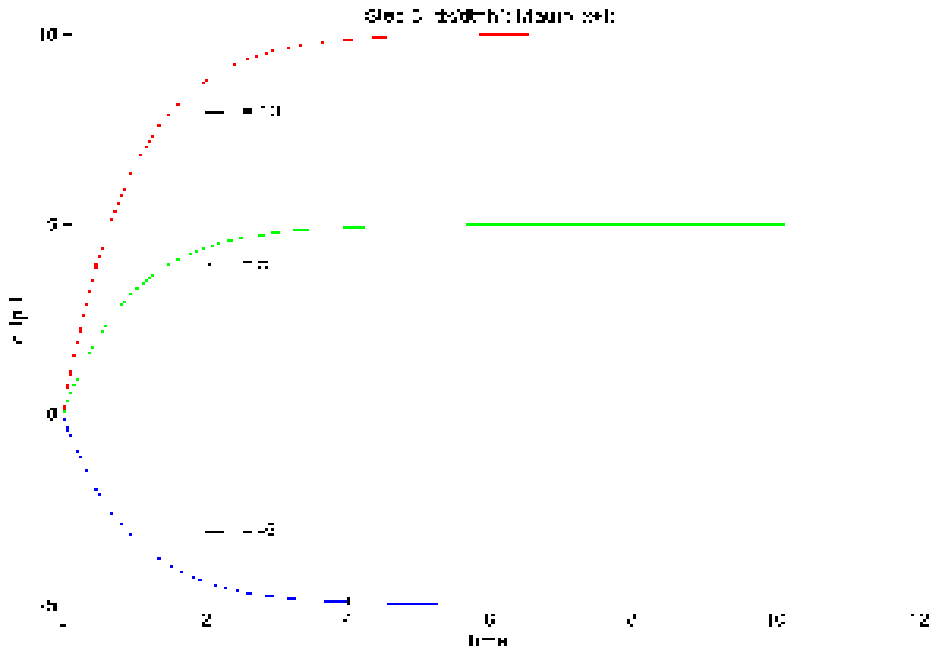


fig. 3 output tends to external input

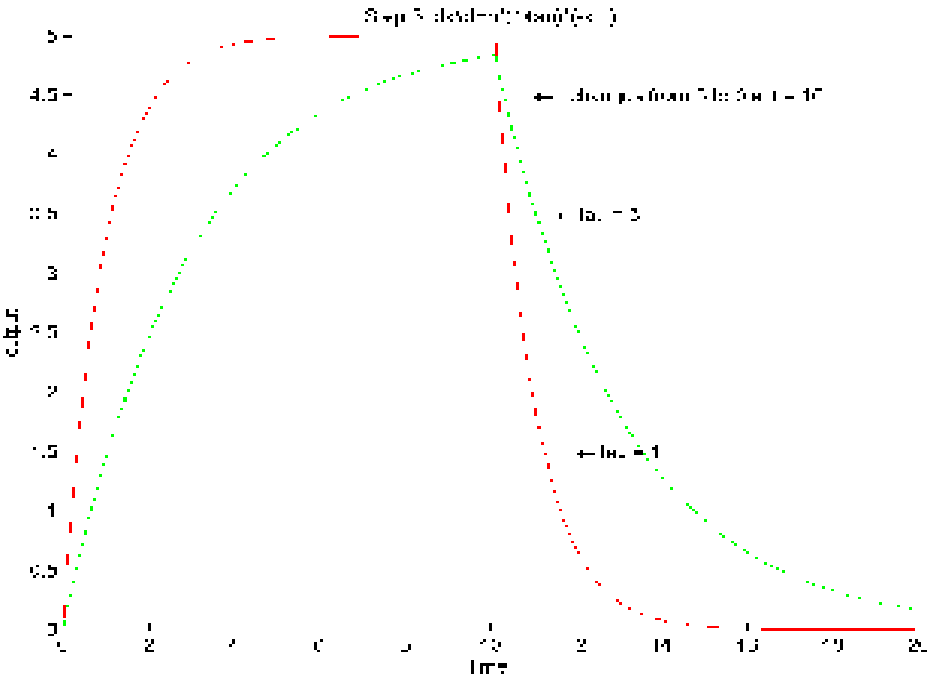


fig. 3a output follows changes in external input

The output decays to I. So we can say that without any connections **the node's output decays to I with time constant τ.**

Step 4: Adding a single self-connection

The simplest practical way in which one of these nodes can be used in a network is with a single self-connection (*Fig. 4*). This is represented in the node equation by the term:

$$w * \sigma(x + \theta) = w * \frac{1}{1 + e^{-(x+\theta)}}$$

where w is the weight of the connection and θ is the bias.

An explanation of the sigmoid function

The σ represents a sigmoid function (*fig. 4a*) which outputs values in the range 0 to 1. Negative input results in values < 0.5 and positive input in values > 0.5. The other important effect of the sigmoid is that the output saturates towards the extremes: over the range –4 to +4 the output varies with the input but above or below these values the output will stay effectively constant at 0 or 1 respectively. This means there is a finite range of **(y + θ)** over which σ will change, and gives us an idea of the useful limits of θ for our practical experiments (for instance +/- 2, in **[2]**).

The weight term

The weight term is a multiplication factor and determines how much effect the value at the connection has on the final value of the node. Multiple connections are represented by multiple instances of the connection term as given above, and each has an individual weight governing its effect.

Fig. 4b shows the effect of various weights on the output of the node. In the absence of bias the value of the connection term reduces to

weight * sigmoid of the output

This will equate to 0.5 when x is at 0 (from fig. 4a, and because of (1+e^-x)) and so will create a change in x. In the absence of external input or bias if x is then positive the output of the node tends to the weight value because as x gets larger the value of the sigmoid term tends towards 1. When the weight value is reached there is no further change because the output and the connection term cancel out, for instance for a weight of 10:

dx/dt = -x + (w* σ(y)) = -10 + (10 * 1) = 0

If x is negative the value of the sigmoid term tends to 0, reducing the effect of the weight and explaining the asymmetry of graph 4b.

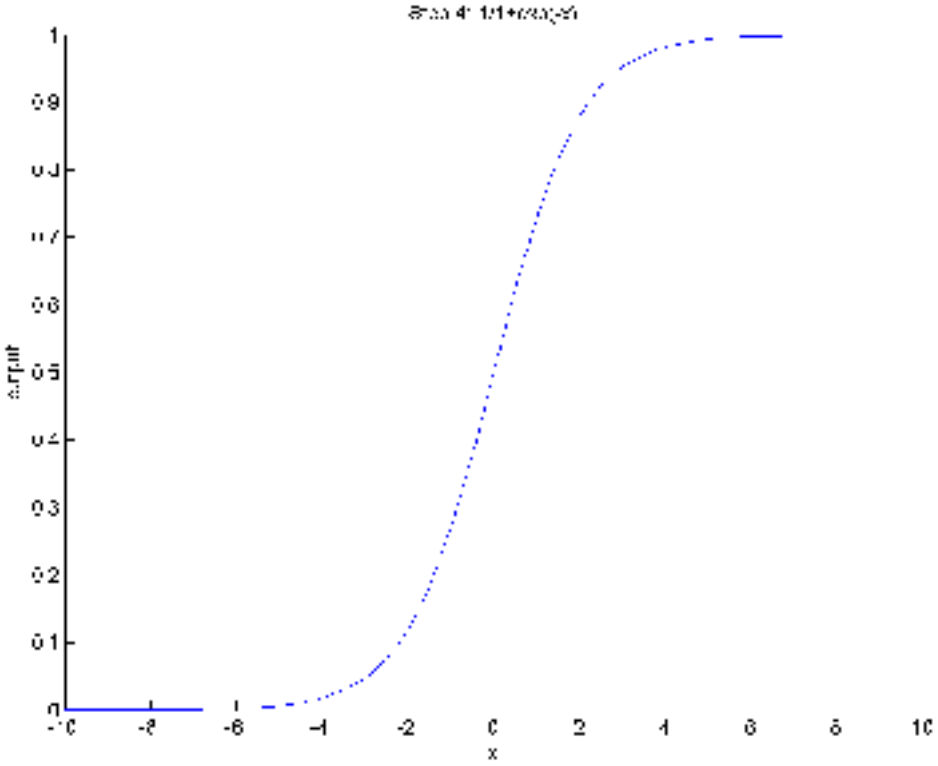


fig. 4a the sigmoid function

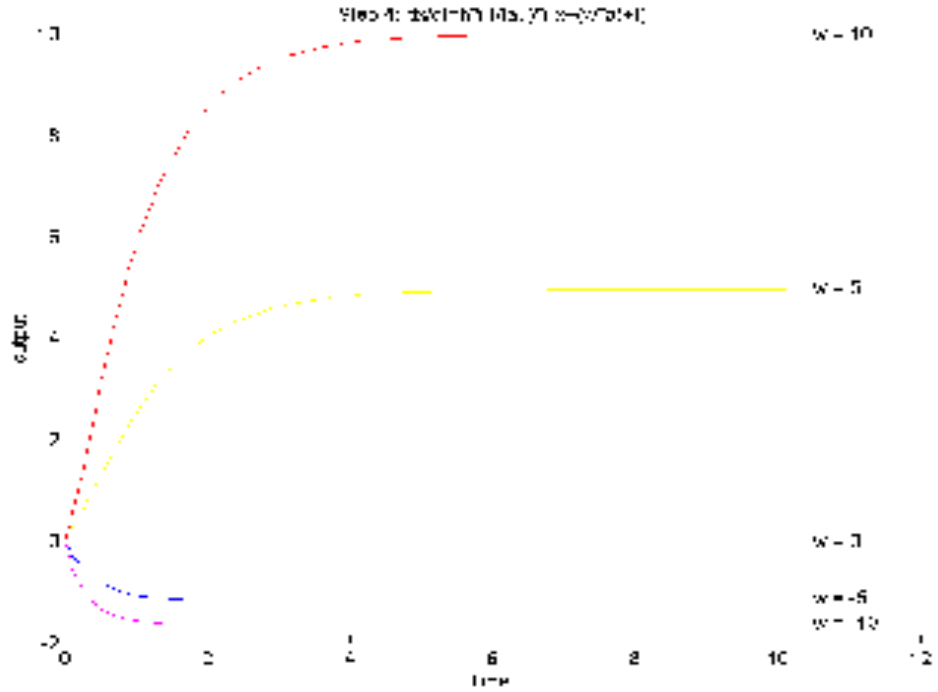


fig. 4b the effect of weights

The bias term

The bias term 'θ' affects the output of the sigmoid function. The effect of varying positive bias is shown in *fig. 4c* and varying negative bias in *fig. 4d* (note in *fig. 4c* the top line is **θ = 10** and in 4d the bottom line is **θ = -10**). In each of these graphs **w = 1** and **I = 0**, and a line at **θ = 0** is included for reference. What is clear from these graphs is that the more positive or negative the bias gets, the more the output 'saturates' at the weight value or zero respectively. It is easy to see why by looking at the sigmoid curve; if the input is roughly > 4 or < -4 the output of the sigmoid will be 1 or 0 and will hardly change with increasing input. Thus the output of the activation function will tend towards the weight value or 0.

Since the output of the sigmoid function depends on **(x + θ)** we can use θ to bias the output of the sigmoid into a particular section of the response curve (usually the 'interesting' section where the output changes with input). Thus the output of the sigmoid as x changes will vary with θ.

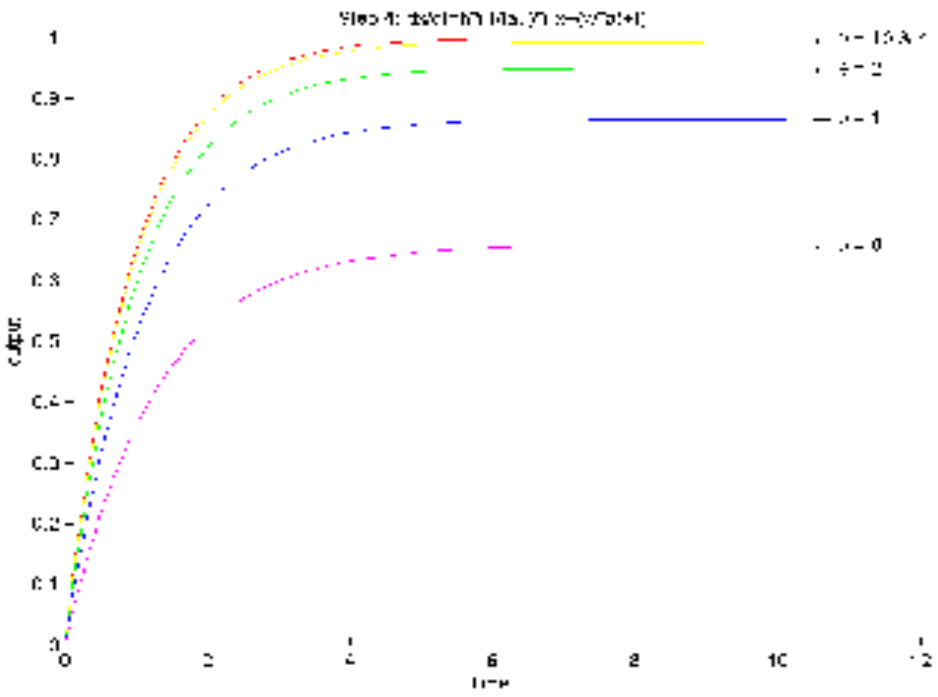


fig. 4c the effect of increasingly positive bias

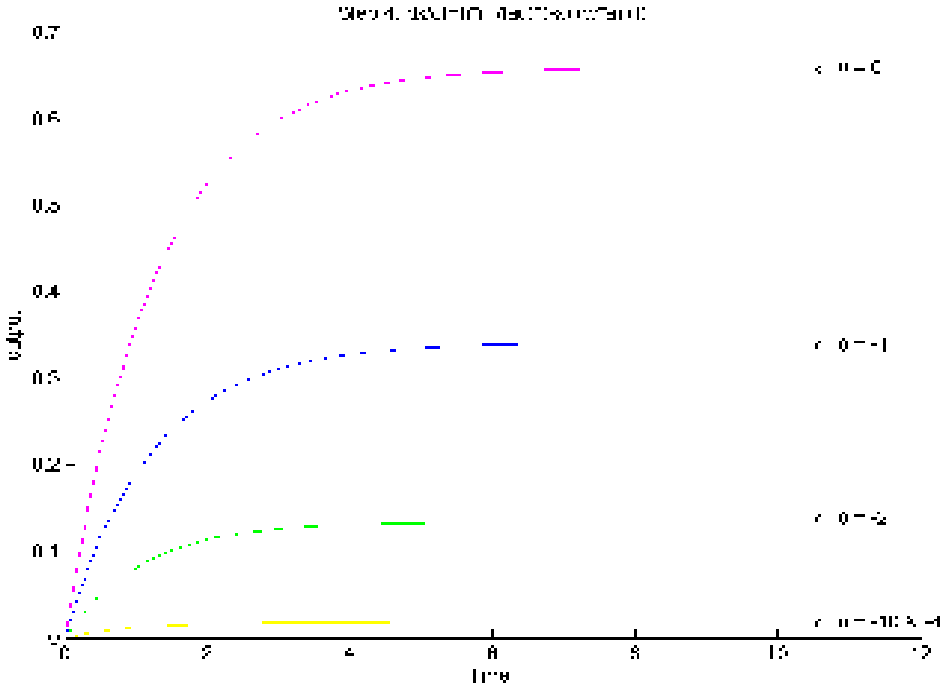


fig. 4d the effect of increasingly negative bias

Step 5: Putting it all together - varying I and w for different θ

Although it is impossible to show all the possible combinations of parameters here there are two graphs, which will prove useful to our understanding of the node's behaviour. The first (*fig. 5*) shows the output for the positive biases in *fig. 4c* where **w = 5** and **I changes from 0 to 10 at t = 5**. The second (*fig. 5a*) shows the output for the negative biases (*fig. 4d*) where **w = -5** and **I changes from 0 to -10 at t = 5**. Note that in the first case the output is driven towards w + I, and in the second towards I, because of the sigmoid term tending towards 1 or 0 respectively.

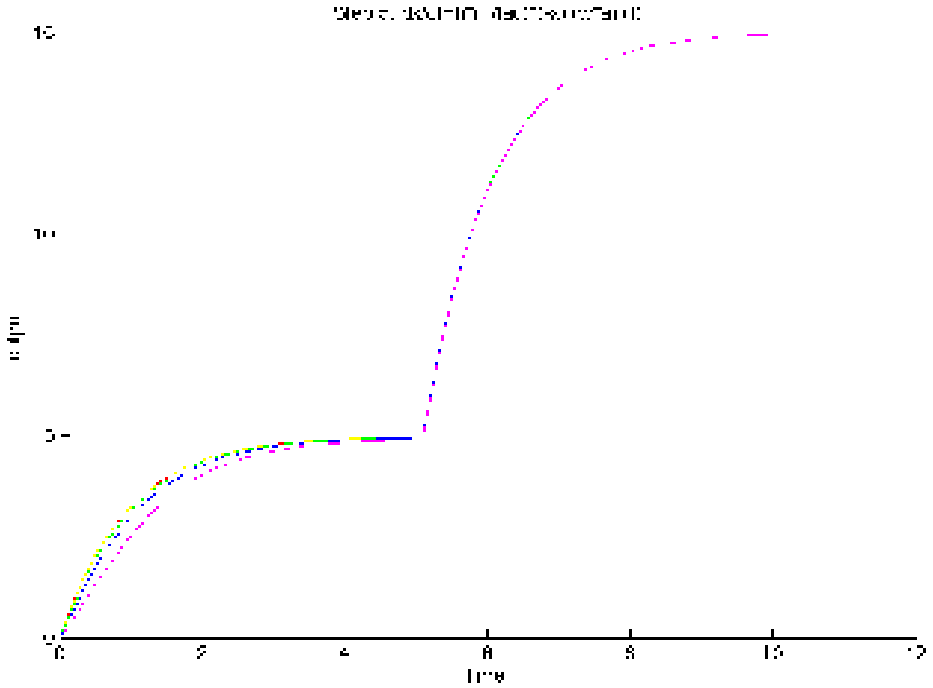


fig. 5 w = 5, I changes from 0 to 10 at t = 5, positive biases

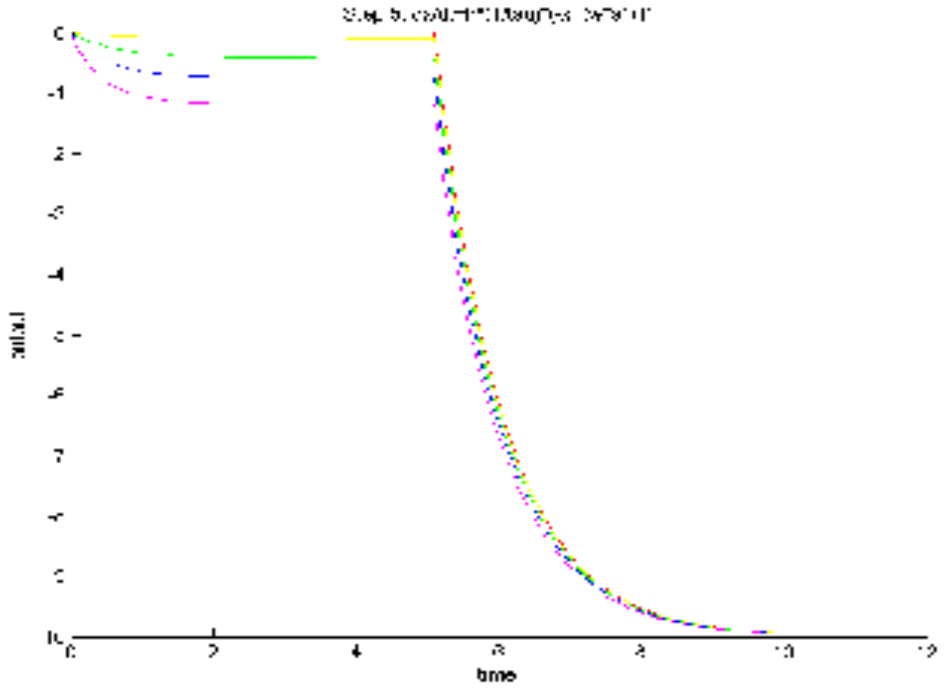
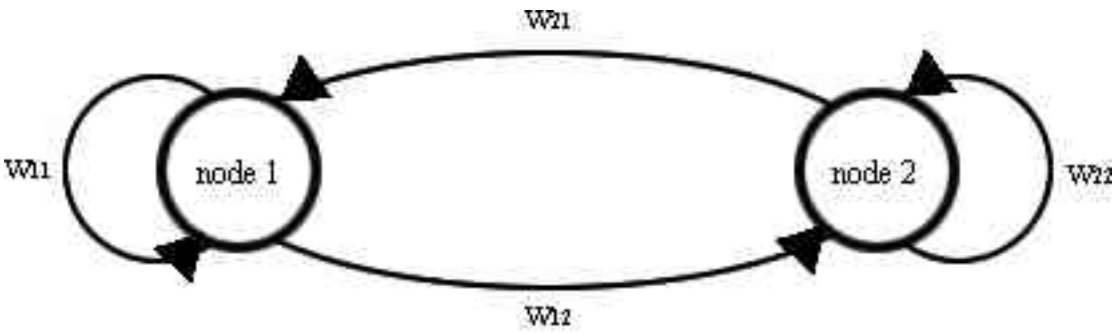


fig. 5a w = -5, I changes from 0 to -10 at t = 5, negative biases

Step 6: A 2-node network

When 2 or more CTRNN nodes are connected their behaviour becomes more complex and dynamic as they interact. Although it is beyond the scope of this report to analyse the examples in detail, *figs. 6a – 6e* show some of the range of behaviours that can occur from the network shown in *fig. 6*. Each graph shows the output of each node against time (top) and the output of node one on the x axis plotted against the output of node 2 on y (phase space plot, bottom). The network was seeded with random values for starting **x** (in the range -10 to +10), **tau** (0.1 – 10), **weights** (- 20 to +20) and **biases** (- 4 to + 4) but note that for all these examples **I remained 0**. This means the behaviour of the nodes was 'self-generated' and not influenced by external input. In most instances the output tends to a steady state but *figs. 6d and e* show some oscillatory behaviours.

It is this time-variant and dynamic behaviour which has made CTRNNs an active topic of research for robot control in recent years **[2]**.



$$\frac{dx_1}{dt} = \frac{1}{\tau_1}\left(-x_1 + w_{11}\,\sigma(x_1+\theta_1) + w_{21}\,\sigma(x_2+\theta_2) + I_1\right)$$

$$\frac{dx_2}{dt} = \frac{1}{\tau_2}\left(-x_2 + w_{22}\,\sigma(x_2+\theta_2) + w_{12}\,\sigma(x_1+\theta_1) + I_2\right)$$

fig. 6 2-node circuit and equations

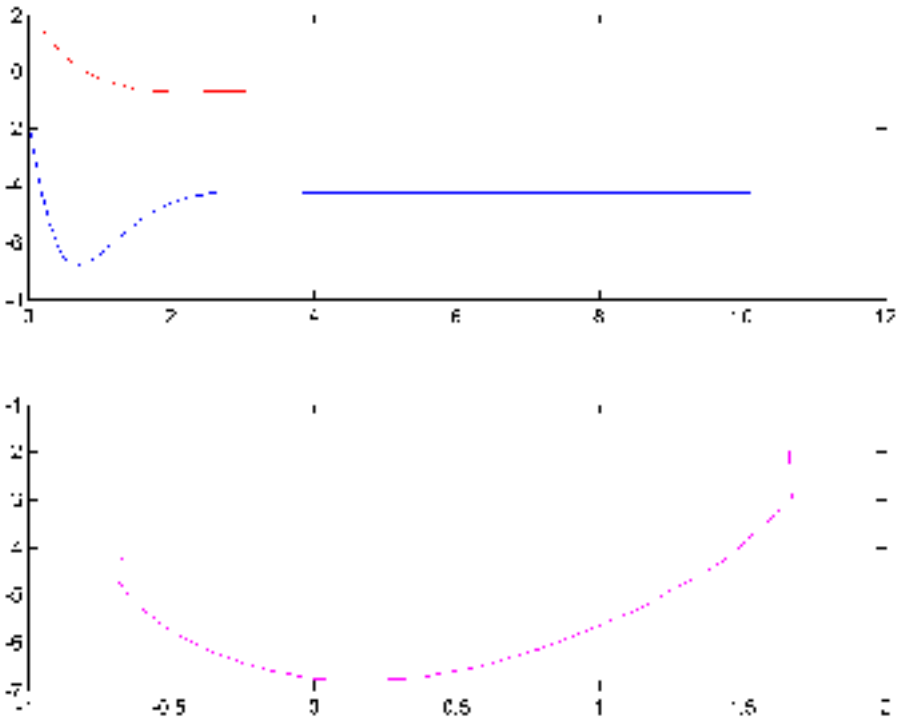


fig. 6a

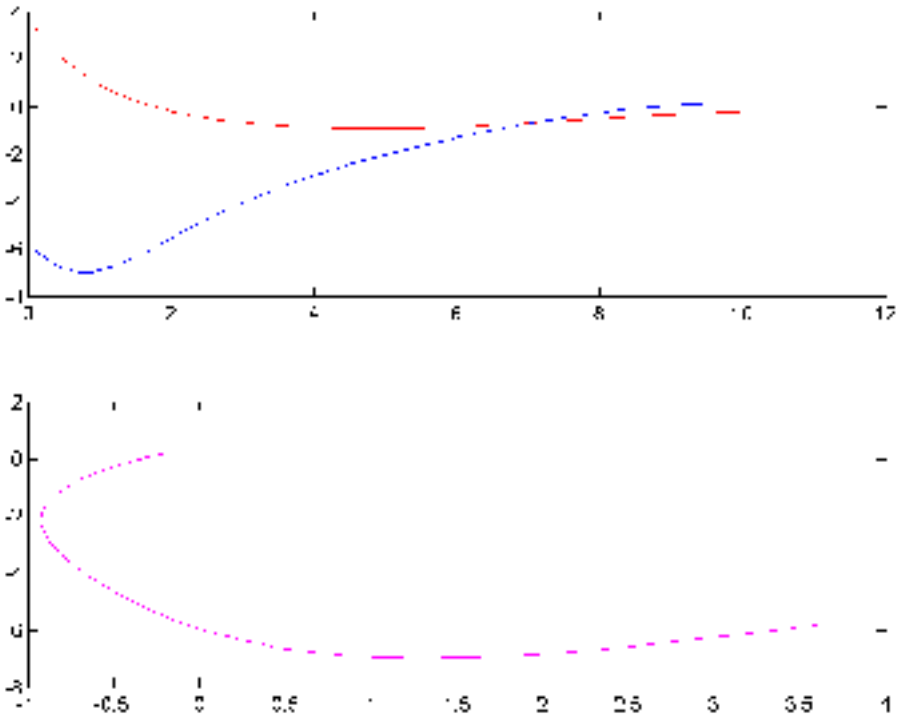


fig. 6b

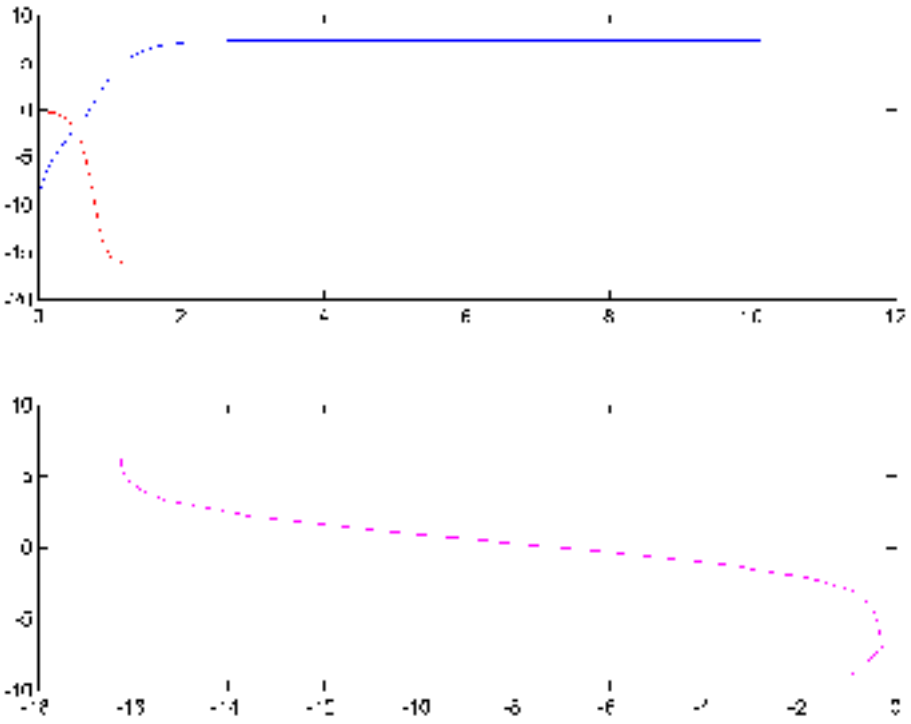


fig. 6c

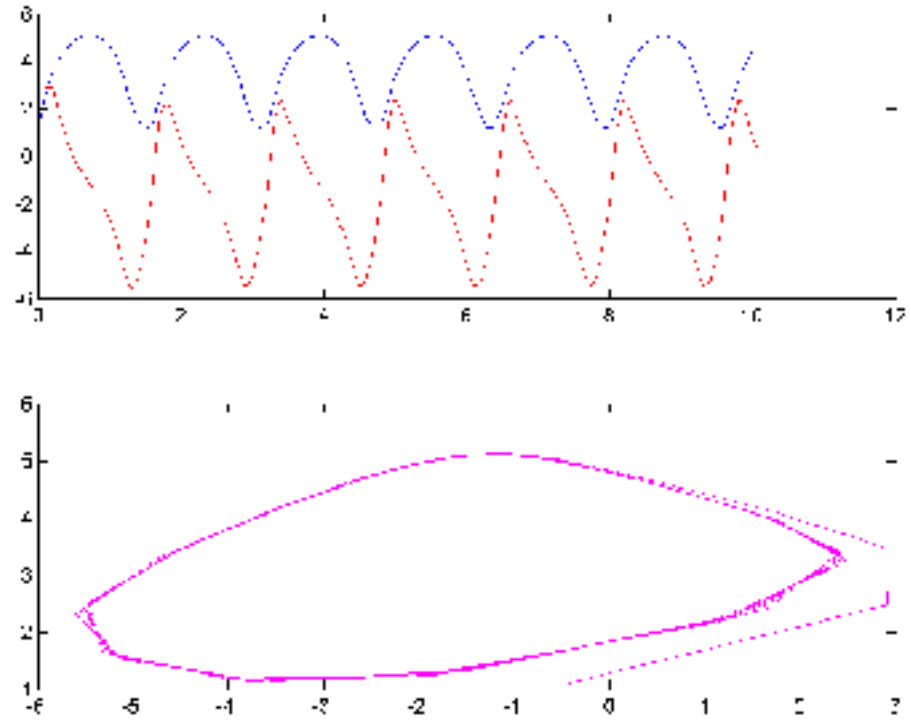


fig. 6d

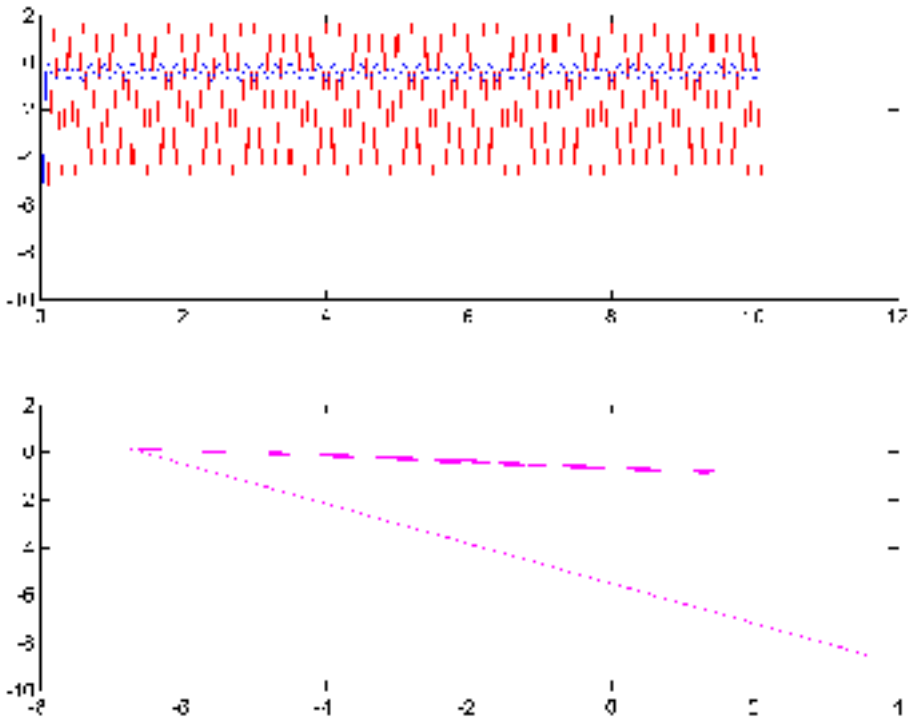


fig. 6e

Step 7: Dynamical systems analysis

In order to predict what the behaviour of a dynamical system will be we need to analyse it and determine two things:

1) the equilibrium points, i.e. the points at which dx/dt = 0, and
2) the stability of these points, i.e. what will happen to x if we perturb it away from the equilibrium point slightly. For a stable point dx/dt will return to 0 and x will return to the equilibrium point. For an unstable point dx/dt will get larger with time and x will never return to the equilibrium point (see *fig. 2b* for an example). There is also a special case 'half-stable' point at which periodic behaviour can occur.

A simple example

To analyse the equation in step 3,

dx/dt = h * (1/ τ) * (-x + I)

assuming **τ = 1** and **h = 1** is not unreasonable and simplifies the equation further:

dx/dt = -x + I

at a fixed point **dx/dt = 0**, so **–x + I = 0**

and solving for x,

x = I

so there is a fixed point at I, which is apparent from looking at the graphs in step 3.

To find the stability for a discrete dynamical system first we define

x(n+1) = f(x(n))

the fixed points are where **x(n) = x(n+1)** (i.e. change in x from n to n+1 = 0, another way of saying dx/dt=0)

so the fixed points can be found by solving **a = f(a)**. This can be solved graphically by plotting **y = f(a)** and **y = a** on the same axes.

In our case **dx/dt = -x + I**

And from **x(n+1) = x(n) + h * dx/dt**, remembering **a = f(x(n))** and **h = 1:**

y = a + h * (–a + I)

y = a – a + I

y = I

If these graphs are plotted with 'a' on the horizontal axis and 'y' on the vertical, the plot of y = a is a linear graph and the plot of y = I is a horizontal line at I (*fig. 7*). The fixed points of the equation are where the two lines cross, in this case always at x = I, so this equation has one fixed point at I.

The stability of the point is given by the gradient of **f(a)** (i.e. **f'(a)**) at the fixed point:

If the gradient is > 1 the point is **unstable**
If the gradient is > 1 the point is **stable**
If the gradient is = 1 the point is **half-stable** (inconclusive)

In this case **f(a)** at the fixed point is a flat line, so the gradient is 0, meaning that the point is stable. If x is perturbed away from this point it will return to it. More complex cases can be solved by finding the absolute value of the derivative of **f(a)** at the fixed point (written as | **f '(a(0))** |).

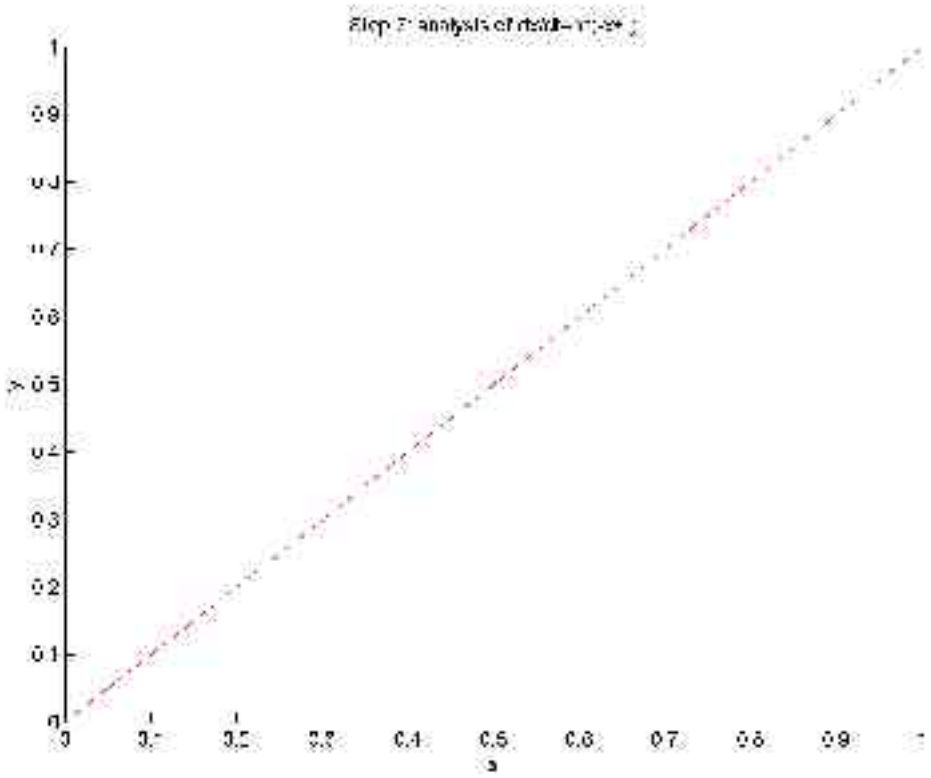


fig. 7 finding the fixed points of x by plotting y = a and y = f(a)

Analysis of the CTRNN equation

The CTRNN equation, for one node with a single self-connection, is:

$$\frac{dx}{dt} = \frac{1}{\tau}\left(-x + w\sigma(x+\theta) + I\right)$$

from the simple example we know that the fixed points occur when **dx/dt = 0**. If we again assume **τ = 1** the equation becomes:

dx/dt = -x + (w * σ(x + θ)) + I

so at a fixed point

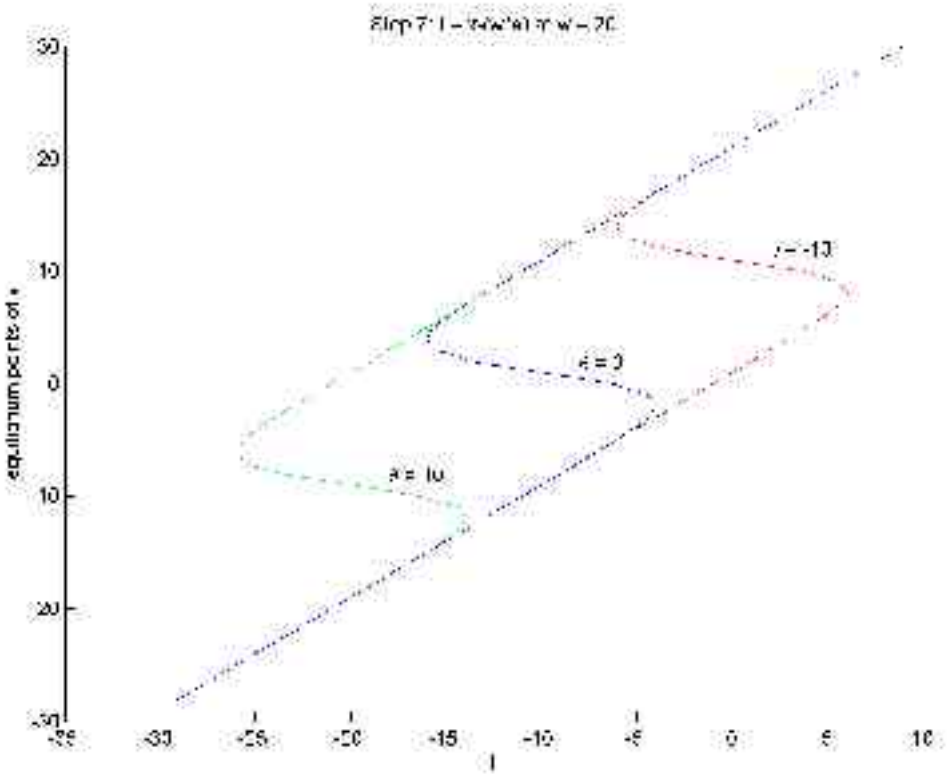
-x + (w * σ(x + θ)) + I = 0

and

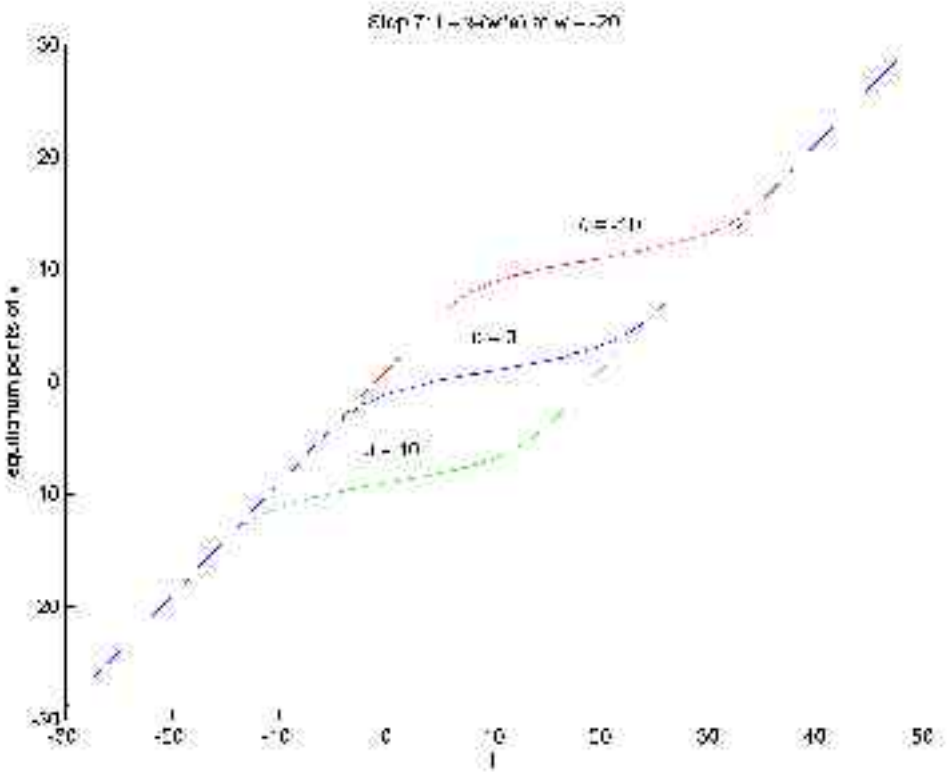
I = x - (w * σ(x + θ))

This equation cannot be solved for x but can be plotted to give the fixed points of x for varying I, given values for w and θ. *Fig. 7a* shows the fixed points of x for biases of –10, 0 and +10 when **w = 20** and *fig. 7b* when **w = -20**. On both graphs the lines are solid for stable equilibrium points and dotted for unstable ones (see discussion

below). We can see from the graph for example that for **θ = 0** at **I = -10** there are two stable equilibrium points at approximately **I = -8** and **I = 12** surrounding an unstable one at **I = 2**. We can also see that a change in bias shifts the response horizontally and vertically by –θ**.**



******** *fig. 7a equilibrium points at w = 20*



**** *fig. 7b equilibrium points at w = -20*

The stability of the fixed points can be found, as before, by using the derivative of the function, although in this case the point is stable if f'(x) < 0 and unstable if f'(x) > 0. So:

f(x) (or dx/dt) **= -x + w* σ(x + θ) + I**

f'(x) = -1 + w * σ(x + θ) * (1 - σ(x + θ))

(from the derivative of σ(x) = σ(x) * (1 - σ(x))

from our previous example:

x = -8

f'(x) = -1 + 20 * (σ(-8)) * (1 - σ(-8))

= -1 + (20 * 0.0003 * (1 – 0.0003))

= -0.99 : STABLE

x = 2

f'(x) = -1 + 20 * (σ(2)) * (1 - σ(2))

= -1 + (20 * 0.881 * (1 – 0.881))

= 1.097 : UNSTABLE

x = 12

f'(x) = -1 + 20 * (σ(12)) * (1 - σ(12))

= -1 + (20 * 1 * (1 – 1))

= -1 : STABLE

Onset of instability

From graphs 7a and 7b above it is clear that at some point an unstable equilibrium point appears as w increases from –20 to 20. To find this point we need to find the value of w where f'(x) is zero. *Fig. 7c* shows the output of the sigmoid derivative **σ(x + θ) * (1 - σ(x + θ))** (note the shift due to θ which accounts for the shift in responses seen in *figs. 7a and b*). We can see that the largest this term will ever be is 0.25. So:

f'(x) = 0 = -1 + (w * 0.25)

w * 0.25 = 1

therefore **w = 4**

So the unstable equilibrium point in the CTRNN behaviour appears at **w = 4**. Below this weight value the equation has one stable equilibrium point. Above it the equation has three, with the size of the central unstable area (*see fig. 7a*) increasing with w.

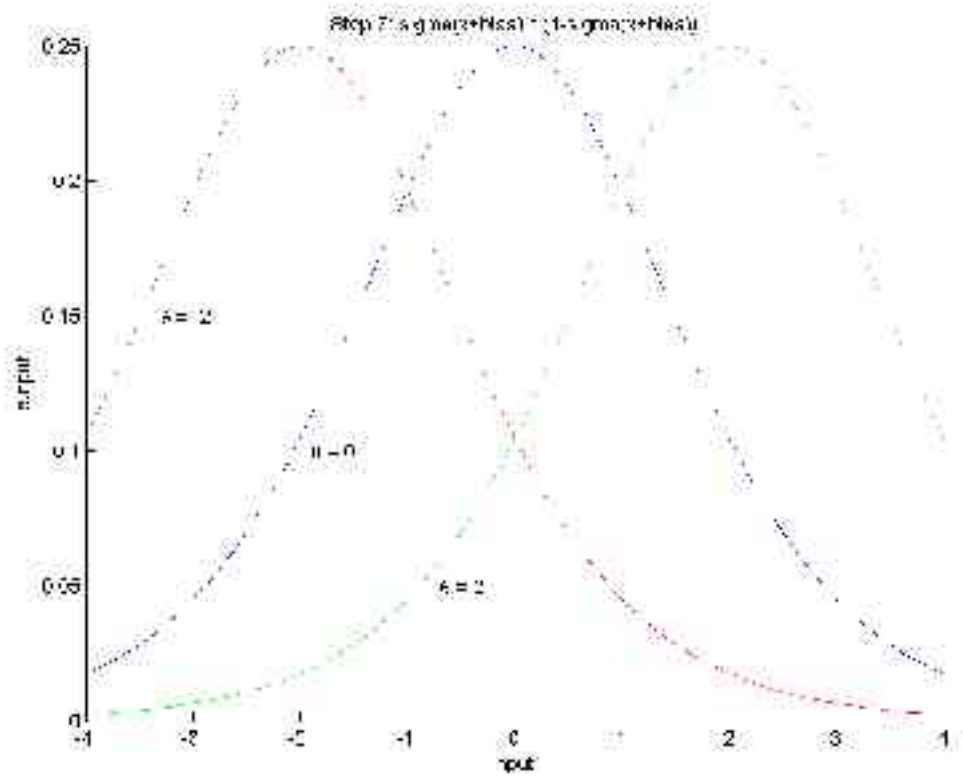


*fig. 7c output of the sigmoid derivative σ(x + θ) * (1 - σ(x + θ))*

Conclusion

This report has looked in some detail at the behaviour of a CTRNN node. We have altered each of the parameters affecting the node and gained an intuitive understanding of their effect. We have also seen various examples of the dynamic behaviour of a simple 2–node network plotted against time and in phase space. Finally we have applied some dynamical systems analysis techniques to examine the fixed points and stability of a single node under various circumstances.

References

[1] Beer, R.D. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behaviour* 3(4):469-509.
[2] E. Tuci, I. Harvey and M. Quinn. (2002) Evolving integrated controllers for autonomous learning robots using dynamic neural networks. In B. Hallam, D. Floreano, J. Hallem, G. Hayes and J-A Meyer, editors, *From Animals to Animats VII* Cambridge, MA, 4-9 August 2002. MIT press.
[3] Non-linear Dynamics and Chaos p 44-48, Steven H. Strogatz, Addison Wesley