Written by Peter Sarvari

Computer practical results for course Computational Neuroscience by Dr Claudia Clopath, Imperial College London

Integrate and Fire model

The IF model is given by the following equation:

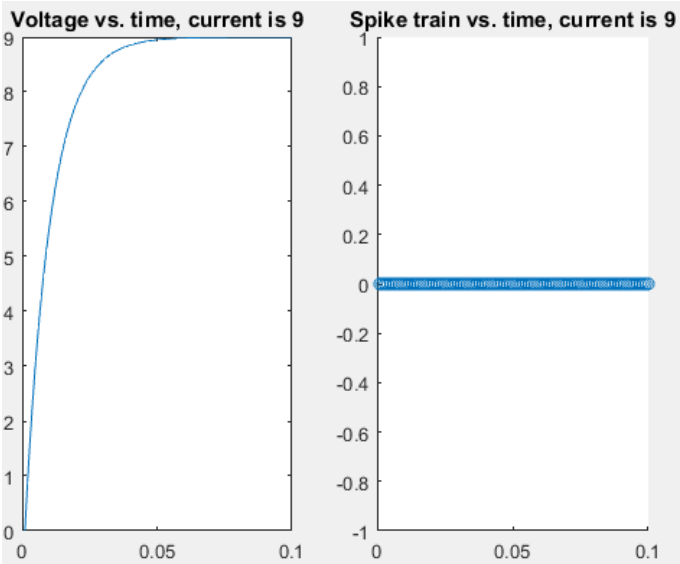
$$RC * \frac{dU}{dt} = -(U(t) - U_{rest}) + RI(t)$$

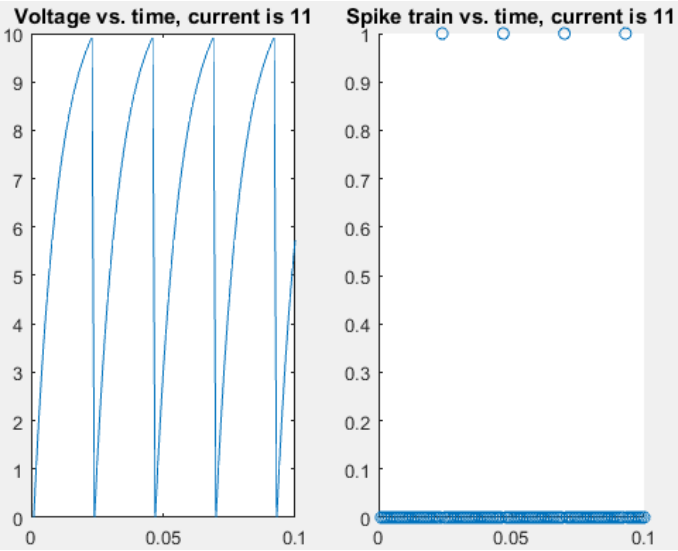
And has the following solution for constant current:

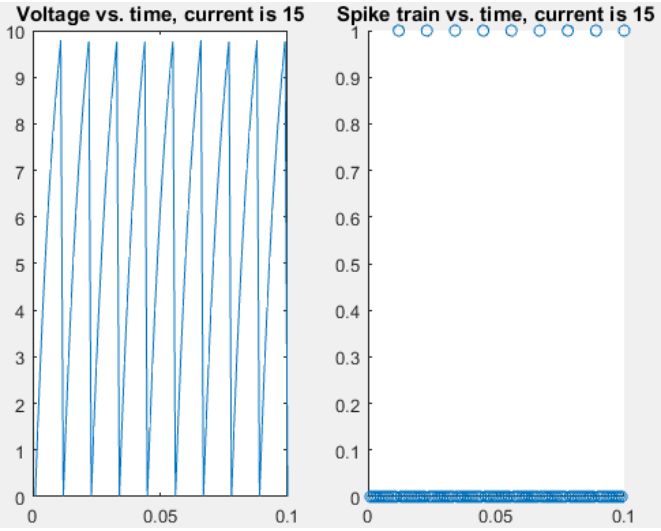
$$U(t) = U_{rest} + R * I_0 * \left(1 - e^{-\frac{(t-t_0)}{R*C}}\right)$$

When the threshold is reached, the neuron spikes and then its potential is reset to the resting value.

We simulate the equations via the Euler method for R=1, threshold = 10 and different current values (arbitrary units). The resting potential is zero.







I = 9 is less than the minimal current to reach the threshold (which is $\frac{10-0}{1} = 10$), hence no spiking behaviour is observed. After the threshold, the higher the current the more frequent the spiking is. In fact, the frequency as a function of constant I is given by the following gain function:

$$g(I) = \left[\tau * \ln\left(\frac{R * I}{R * I - (\theta - U_{rest})}\right)\right]^{-1}$$

Ring network

Paper: Ben-Yishai et al. Theory of orientation tuning in visual cortex, PNAS, 1995

The ring network consists of the following equations:

The i[th] neuron input current is written as:

$$I_i(\theta_0) = c * \left[(1 - \varepsilon) + \varepsilon * \cos\big(2 * (\theta_i - \theta_0)\big)\right]$$

Where θ_0 is the orientation of the visual stimulus and θ_i is the preferred direction of neuron i. Note that $(\theta_i - \theta_0)$ is multiplied by two inside the argument of cos, because if the orientation differs by 180 degrees, it is actually the same orientation. c is the image contrast, the bigger the stimulus is the bigger c is. ε controls how strongly the input is modulated, i.e. the bigger ε, the more selective the input is to the neurons whose preferred direction aligns with the object orientation.

Also we model the synaptic weights between two neurons as a function of their preferred direction:

$$w_{ab} = w(\theta_a, \theta_b) = -J_0 + J_1 * \cos\big(2(\theta_a - \theta_b)\big) \qquad (1)$$

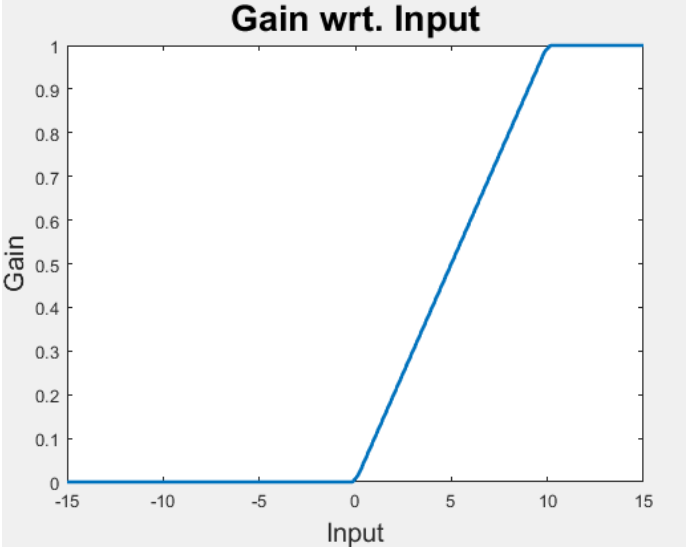
Now, we can write the rate equation with continuous labelling of neurons as follows:

$$\tau * \frac{dv(\theta_i)}{dt} = -v(\theta_i) + g\left[\int_{-\pi}^{\pi} w(\theta, \theta_i) * v(\theta) * d\theta + I_i(\theta_0)\right]$$

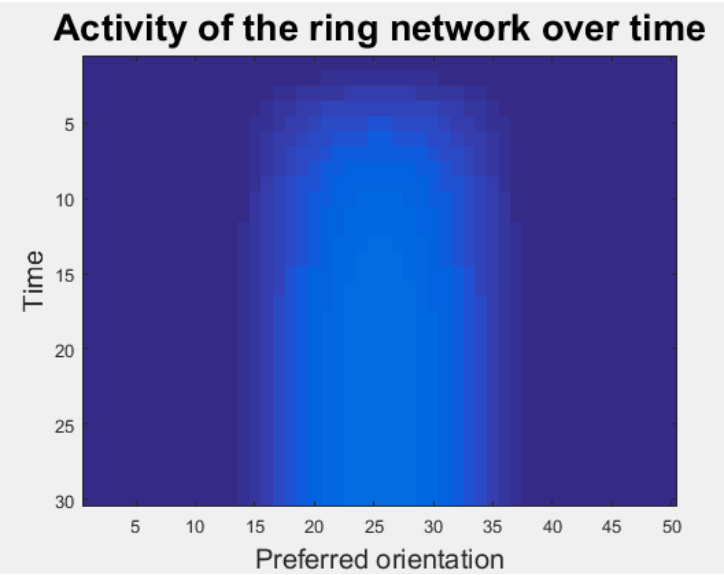
Where g is a sigmoid type function.

Results:

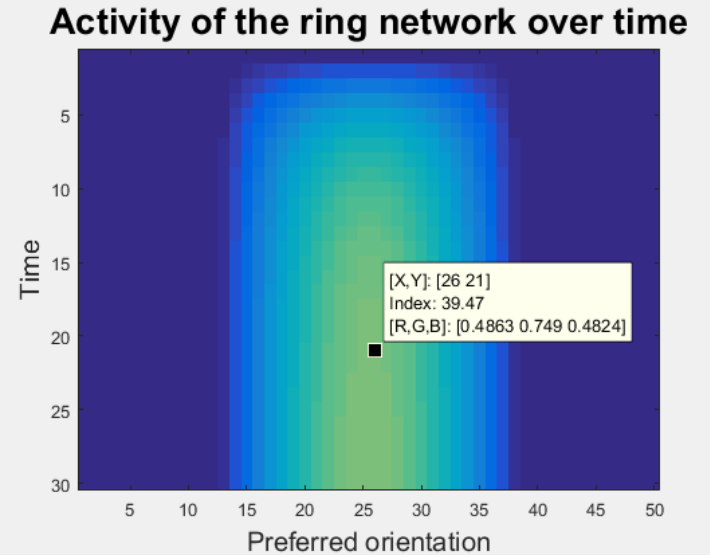
The saturation nonlinearity (function g) is chosen to be



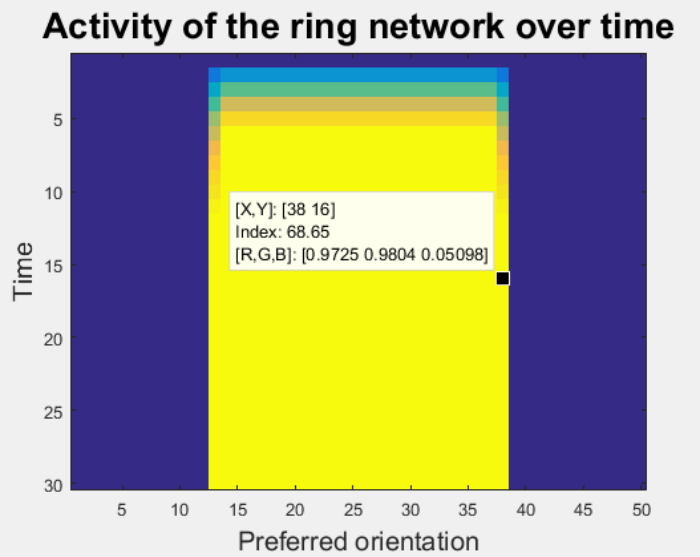
We implement the ring network with 50 neurons with preferred directions equally distributed from $-\pi$ to π. This can be seen on the x axis of the plots below. Assume no connectivity for now ($w_{ab} = 0$). Let's choose $\epsilon = 0.9$, which means that almost for half of the neurons the input will be negative, and positive for the other half. This means that if c is very high, just a bit more than half of the neurons (would be exactly half if ϵ was one) will have a very high activity, and the other half will have activity of zero. This can be seen on the third diagram, where c = 100. For c = 1.2 and c = 4, we see that neurons who have preferred direction aligned with the orientation of the object have the highest activity. However, as c increases (e.g. from 4 to 100), this specificity toward the orientation of the object is lost. It is due to the huge contrast (c=100), which makes the sigmoid function saturated even for small positive values of input current.



1The contrast (c) is set to 1.2

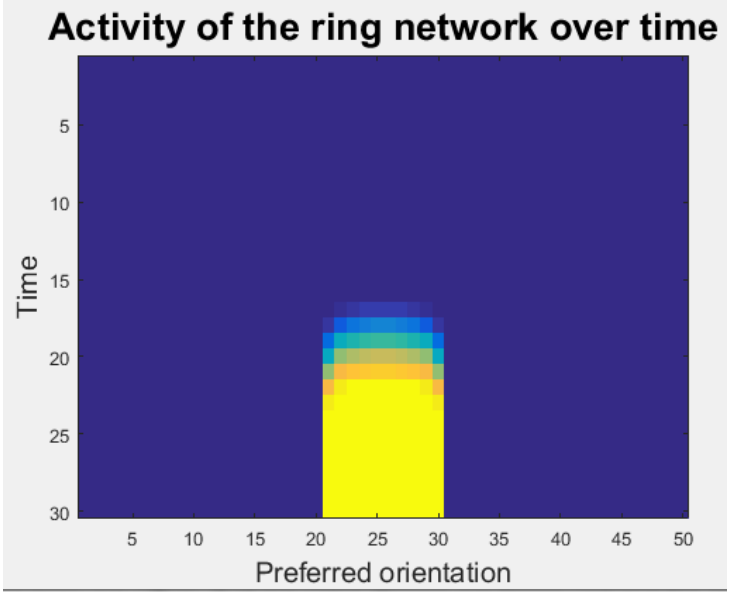


2The contrast (c) is set to 4

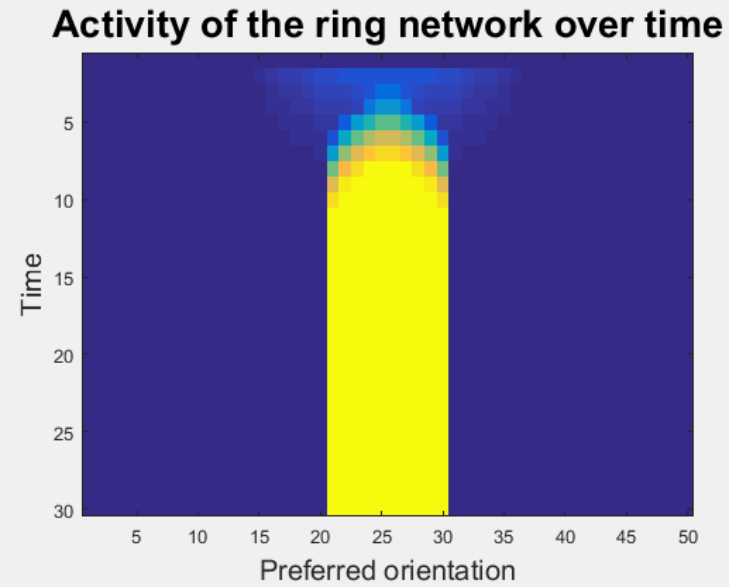


3The contrast (c) is set to 100

Now let's add connectivity! What we observe is the iceberg effect. By subtracting J_0 in (1), we only make those neurons fire whose incoming current is above a certain threshold, hence whose preferred direction are the closest to the orientation of the object. We decided to run for the following contrast values (in order) 10^{-7} and 70. We observe that the contrast value here does not matter, because with these parameters the sigmoid function is already maxed out by the incoming current from other neurons. Even as small contrast as 10^{-7} is eventually (with time) gets amplified by the network to max out the sigmoid. This is referred to as contrast invariance. Furthermore we run not only for $J_0 = 86$, but also for $J_0 = 150$. We see the iceberg effect at its best: the higher J_0 is, the more specific the active neurons are to the orientation of the object.

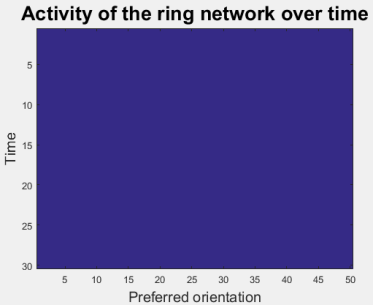


4The contrast is set to 10^-7; J0 = 86



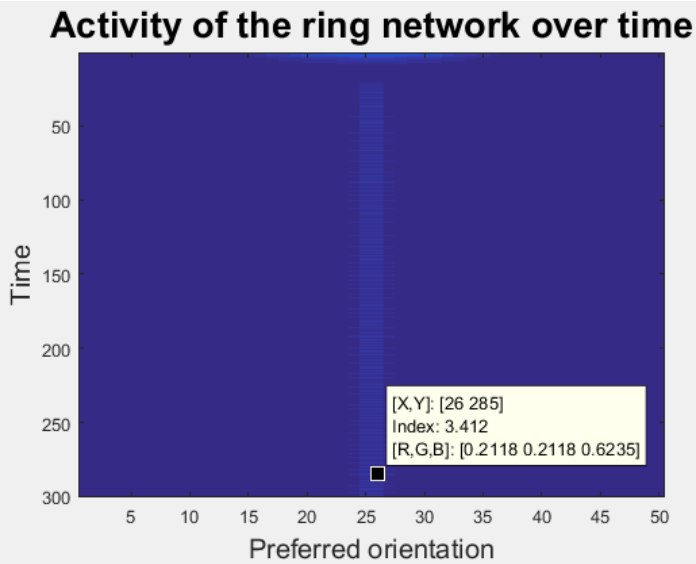
5The contrast is set to 70; J0 is 86

Now with J_0 = 150. Not surprisingly, for the tiny c and big J_0, we actually do not see any activity. This is because it is impossible to get positive current from (1) since J_0 is bigger than J_2 and the current from (1) is tiny because of the tiny contrast.



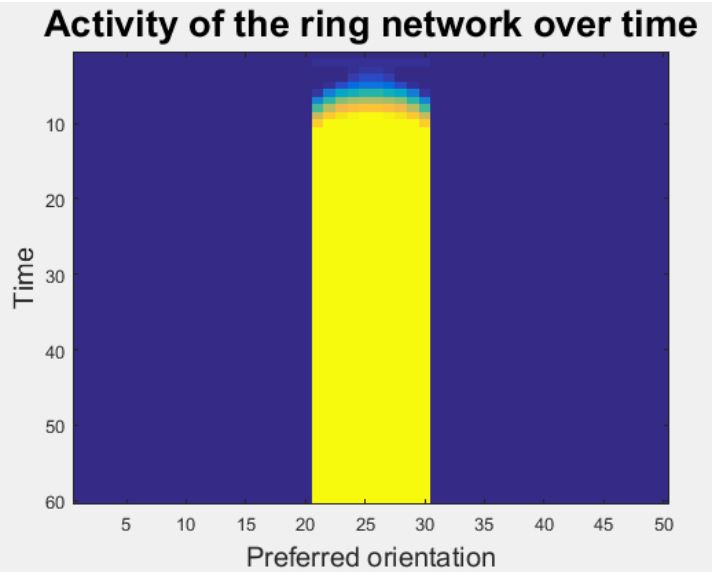
6The contrast is set to 10^-7, J0 is 150

Here we can see some small, but very specific activity.



7The contrast is set to 70 and J0 is set to 150

Next we will show sustained activity, i.e. neurons, whose preferred direction aligns with that of the orientation of the bar, keep firing even after removal of the stimulus. This makes sense if the weights are sufficiently strong, since neurons with preferred orientation similar to that of the bar are already firing and they give sufficient current to each other to keep each other firing even without the input. We simulate the network for 30 timesteps with input and then the input is removed for the remaining 30 timesteps.



8The contrast is set to 1.2 and J0 is 86

Bienenstock Cooper Munro (BCM) rule

The equations are:

$$\tau * \frac{d\boldsymbol{w}}{dt} = \boldsymbol{x} * y * (y - \theta)$$

$$\tau_\theta * \frac{d\theta}{dt} = \frac{y^2}{y_{target}} - \theta$$

Since the weights change much slower than the firing of the neurons (and the threshold changes even slower than the weights), we can replace y with its time average, <y>. Then in steady-state, we have:

$$< y >= \theta$$
$$\frac{< y >^2}{y_{target}} = \theta =< y >$$

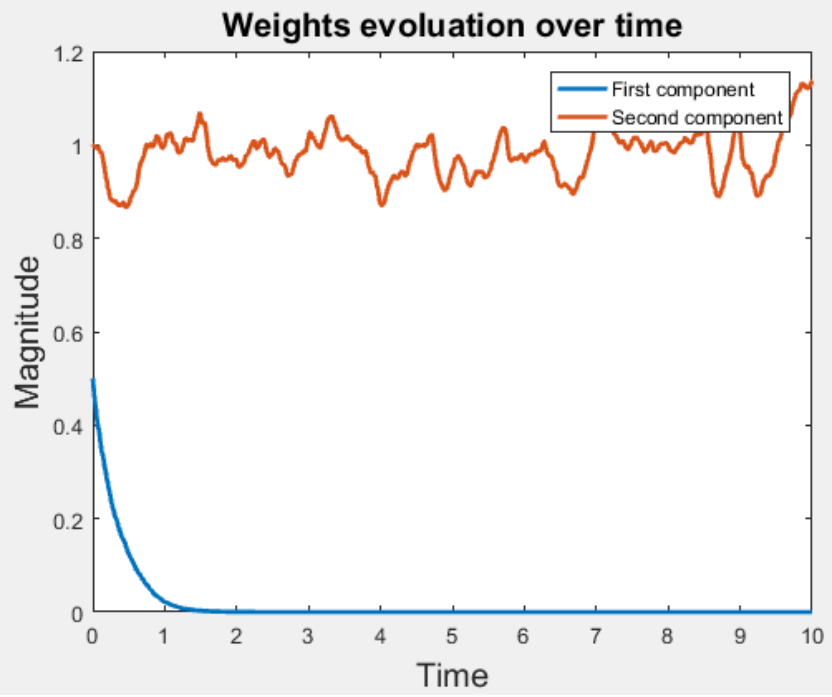
This implies that

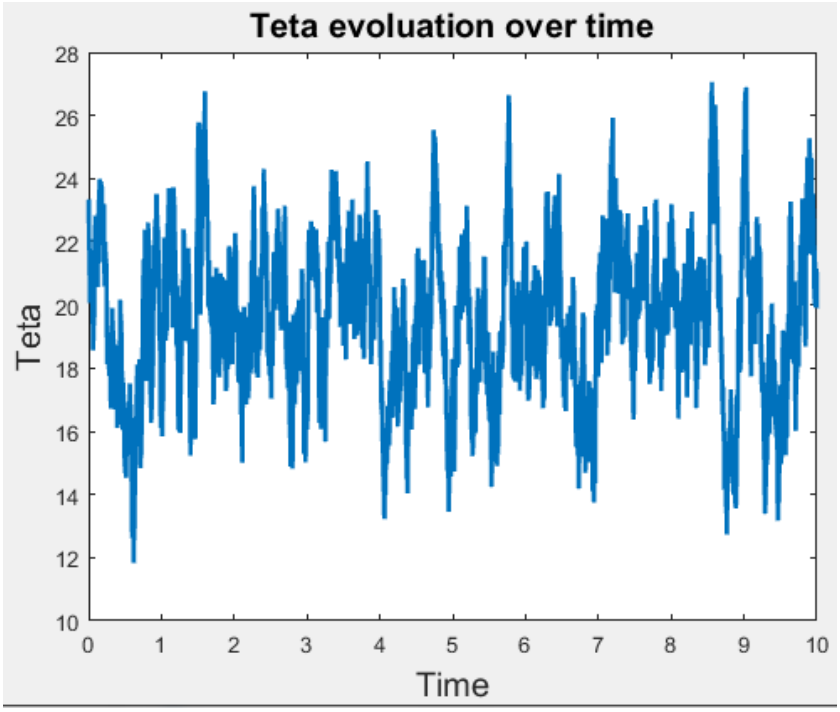
$$< y >= y_{target}$$

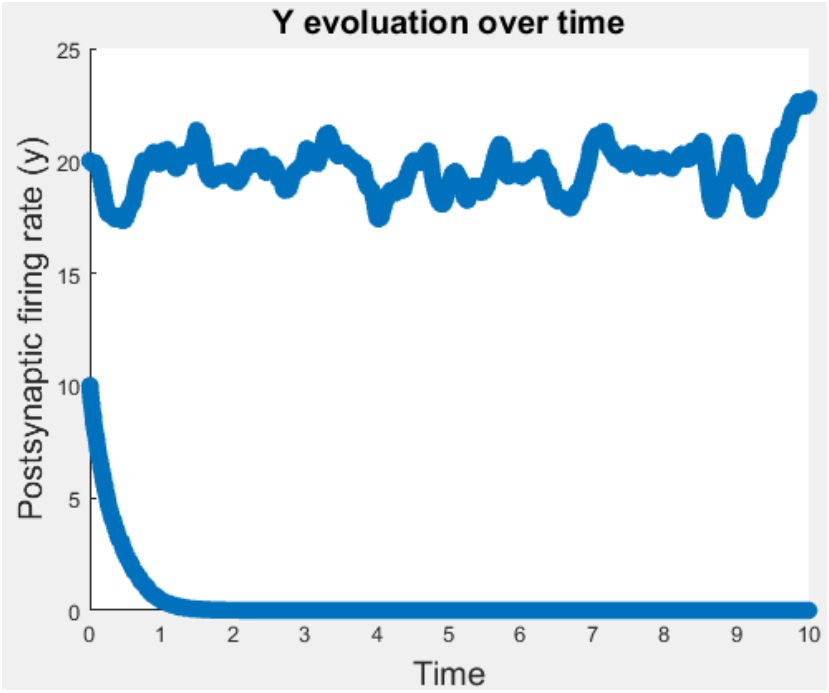
So with the second rule, we can set the long-term firing rate of y, all the weights but one will go to zero and one will be big, but not infinite, since $\theta = y$ in the long term.

Results:

Here we assume that we have two input neurons, x_1 and x_2 to the postsynaptic neuron, y. We further assume that either x_1 fires with frequency 20 (a.u.) or x_2 fires with frequency 20. The probability of x_1 firing is 0.5. Initial theta is chosen to be 23. We set y_{target} to 10. Both input neurons fire at the same intensity (20), but x_1 has initial weight of 0.5, whereas x_2 has initial weight of 1. It can be seen from the result of the BCM rule below, that the competition is won by x_2, which had the higher initial weight. The weight associated with x_1 goes to zero and so y stops firing when x_1 fires (which happens with a probability of 0.5). The average weight associated with x_2 stays one, so when x_2 fires, y fires at 20. So on average y fires at 10, which coincides with y_{target}.







Spike-time dependent plasticity (STDP)

If presynaptic neuron spikes before postsynaptic neuron then weights go up. They go up more, the shorter the time lag is. If postsynaptic neuron spikes before presynaptic neuron, then the weights go down. They go down more the shorter the time lag is. This is shown by the double exponential learning window, which can be derived by considering the following rules:

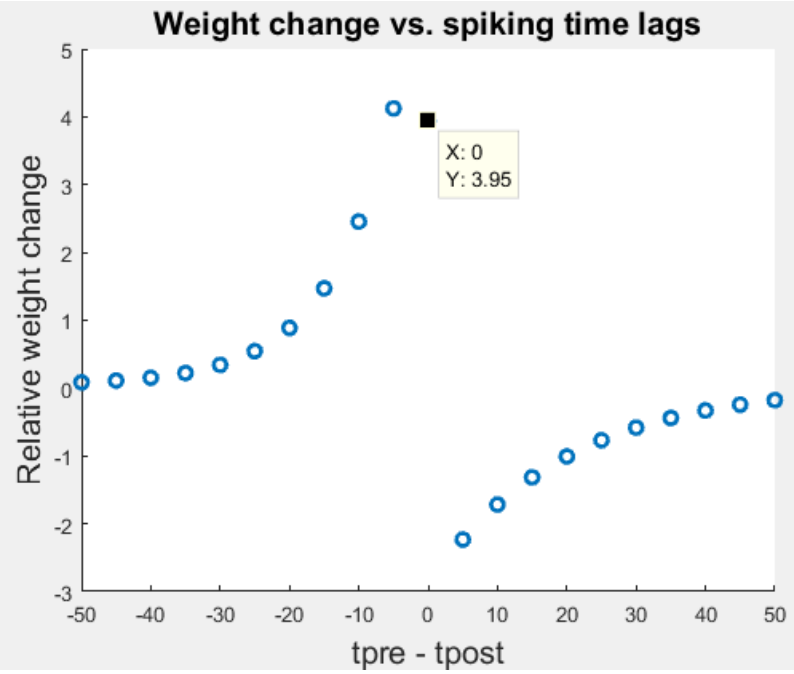
$$\tau_+ \frac{d}{dt} x^{pre} = -x^{pre} + \delta(t - t^{pre})$$
$$\tau_- \frac{d}{dt} y^{post} = -y^{post} + \delta(t - t^{post})$$

$$\frac{d}{dt} w = A_+ x^{pre} \delta(t - t^{post}) - A_- y^{post} \delta(t - t^{pre})$$

Results:

```
diffvec = (-50:5:50)
tpre = 1:1000:T;
```

Make as many experiments as length of diffvec. Tpre fires every second (1000 milisecond) and tpost fires at tpre+diffvec. T is chosen to be 60000, which means that we model 60 simultaneous firing with a certain lag. Simulate the system for long enough and plot the end values for the weights for the different experiments. This yields the double exponential learning window:



<u>Perceptron</u>

- Goal: learn associations between inputs x and target output y_t
- The actual output is binary: either 0 (no spike) or 1 (spike)
- We have different examples, $n = 1 \dots N$

Rule:

$$y^n = heaviside(\sum_i w_i * {x_i}^n - b)$$

Where b is the threshold/bias. The goal is that y^n becomes the target output, ${y_t}^n$. To achieve the goal, we need to reduce the error:

$$E = (\boldsymbol{y_t} - \boldsymbol{X} * \boldsymbol{w})' * (\boldsymbol{y_t} - \boldsymbol{X} * \boldsymbol{w})$$

Where $\boldsymbol{X}$ is the design matrix (dimensions: N*d, where d is the number of features for one example). Using the following from Petersen and Pedersen: The matrix cookbook:

$$\frac{\partial}{\partial \mathbf{s}}(\mathbf{x}-\mathbf{A}\mathbf{s})^T\mathbf{W}(\mathbf{x}-\mathbf{A}\mathbf{s}) = -2\mathbf{A}^T\mathbf{W}(\mathbf{x}-\mathbf{A}\mathbf{s}) \quad (84)$$

We can immediately see that

$$\frac{\delta E}{\delta \boldsymbol{w}} = -2 * \boldsymbol{X}' * (\boldsymbol{y_t} - \boldsymbol{X} * \boldsymbol{w})$$

We can easily see that the dimensions work out. Accordingly, the update for **w** in a gradient descent formulation is:

$$\boldsymbol{w} \rightarrow \boldsymbol{w} + \alpha * \boldsymbol{X}' * (\boldsymbol{y_t} - \boldsymbol{X} * \boldsymbol{w}) \quad (2)$$

Where α is the learning rate.

One perceptron is only able to correctly classify examples if they are linearly separable.

Results:

It is possible for a single perceptron to learn, for example, an OR gate, but for an XOR gate, we need two layers of perceptrons (nonlinear separation is needed). If the examples are generated randomly and the dimension is higher than the number of examples, than it is likely that the problem is linearly separable and the error will become zero after training of a single perceptron.

Reinforcement learning: Temporal difference (TD) learning

We would like to predict the total future reward expected from time t. We will do this using the previous stimuli and associated weights with them:

$$v(t) = \sum_{\tau=0}^{t} w(\tau) * u(t-\tau)$$

Goal is to minimize the error:

$$E = \left(\sum_{\tau=0}^{T-t} r(t+\tau) - v(t) \right)^2$$

Where T is the total length of the experiment. Define δ as

$$\delta(t) = \sum_{\tau=0}^{T-t} r(t+\tau) - v(t)$$

We can write that

$$\sum_{\tau=0}^{T-t} r(t+\tau) = r(t) + \sum_{\tau=0}^{T-t-1} r(t+1+\tau)$$

Note that we can estimate $\sum_{\tau=0}^{T-t-1} r(t+1+\tau)$ by $v(t+1)$ and then we have that

$$\delta(t) = r(t) + v(t+1) - v(t) \quad (3)$$

As previously, a gradient descent update can be formulated as:

$$w(\tau) \rightarrow w(\tau) + \alpha * \delta(t) * u(t - \tau)$$

Aside: note that if we have multiple stimuli and **u** is a vector, then we have an equation similar to that of a linear regression (equation (2)). The only difference is that the bracketed term, which is the prediction error, will be different and will reflect equation (3). For example, take the following problem (Problem Set 9 in Computational Neuroscience course, taught by Dr Claudia Clopath, Imperial College London):

The experiment consists of 100 identical trials. In each trial, a stimulus (e.g. a light, a bell, etc...) is presented at time $t_{cue} = 5s$ and kept active until the end of the trial. Following the cue, at time $t_{rew} = 20s$, a reward is delivered and the trial ended.

In our model, the stimulus is represented by a vector $\mathbf{x}(t) = \{x_1(t), x_2(t), ...\}$ that describes the presence (or the absence) of the cue at time t. In particular, we have that $x_i(t) = 1$ if the cue is on and its onset was i timesteps ago, $x_i(t) = 0$ otherwise. The weight vector $\mathbf{w}$ consists of $n = 15$ (cue duration) different weights, one for each component of $\mathbf{x}(t)$.

Actually, the weight vector has to consist of 16 elements, because we define a zeroth element of matrix x, which is one just when the cue turns on. We have a trial consisting of 20 timesteps and the vector **x** hence consists of 16 element. Let's define **X** as a matrix of the concatenated **x**s. Furthermore, let's define **V** as the vector of expected rewards, concatenating the v(t)s and extend it with one zero at the end. By the same way, refine **r** as the concatenation of the rewards. Then we can write:

$$\boldsymbol{V} = \boldsymbol{X} * \boldsymbol{w}$$

$$\boldsymbol{\delta} = \boldsymbol{r} + diff(\boldsymbol{V})$$

Where diff is the difference between elements as defined in MATLAB. And the update rule becomes (using equation (2) but replacing $(\boldsymbol{y_t} - \boldsymbol{X} * \boldsymbol{w})$ with $\boldsymbol{\delta}$):

$$\boldsymbol{w} \rightarrow \boldsymbol{w} + \alpha * \boldsymbol{X}' * \boldsymbol{\delta}$$

Furthermore, to make the problem more interesting, we will omit the reward in the 60th trial. In the results we how the error (δ) changes with time and the number of trials and give an explanation to what is observed.

Results:

It is very easy to understand what is happening if we set *alpha to one*. First there is an error at the end of the trial, because r(20) is one, but we do not expect any reward since all the weights are zero. This error at t = 20 in the first trial, updates the last weight (16th) to one. (Note that it only updates the 16th weight, because only the 16th element in x(20) was non-zero. This reason for this is that the cue happened 15 timesteps before (at timestep 5) and the first element of x is one if the cue just happened, the second is one if the cue happened one timestep before, etc.) In the next trial the error propagates and delta(19) will be one, because the estimated reward at V(20) is now one, since the last weight has been updated to 1, but V(19) is still zero, because the one before last weight that multiplies the only nonzero element in x(19) is zero. Hence the 15th weight is updated to one and in the next trial the error propagates to delta(18) and so on. Now, if at trial 60, the reward is skipped, the following will happen: delta(20) will be -1, and since alpha is one, the last weight will be updated to zero. At the next trial two weight updates will happen: first, the error propagates and now delta(19) will be -1, hence the 15th weight will be updated to zero. However, since now there is a

reward (r(20) is one again in the 61st trial) delta(20) is one and the 16th weight will be updated to be one again. Both of these updates continue to propagate: i.e. in the next trial (62nd trial) delta(18) will be -1, hence the 14th weight is updated to zero, but the 15th weight becomes 1 again. So essentially all weights become zero for one trial and then go back to one in the next. The trial in which a weight is zeroed out is dependent on which element of x(t) it multiplies. If it multiplies the nth element, then the weight will be zeroed out at the trial 60+(16-n) and will be reinstated to one at 61+(16-n).

Note that the same thing happens of alpha is less than one, only that the weights will not zero out completely (and will not go back to one straight after).