Neural Computation
Practical 8: Hebbian plasticity and alternative learning rules

Mark van Rossum
2018 version by Peggy Series, Matthias Hennig, and Theoklitos Amvrosiadis

November 20, 2018

1 Aims

- Model the process of Hebbian learning
- Explore alternative learning rules with additional constraints that are more biologically plausible

2 Theoretical Background

In his influential work in 1949, *The Organization of Behavior,* Donald Hebb postulated that "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased". Later, this became known with the catchy phrase "cells that fire together wire together". Later studies expanded and generalized this concept, to now state that in general the strength of the connection between two neurons depends on the correlation present in their activities. This principle is consistent with the experimental observations of long-term potentiation (**LTP**) and long-term depression (**LTD**)[1]. Note that other forms of non-Hebbian plasticity exist as well, that are not dependent on the correlations between the pre- and post-synaptic cell activities. An example of such a process is synaptic scaling[2], however we will not deal with such forms here.

What we will look at in this lab are phenomenological models (not based on detailed biophysical models of neurons, just modeling inputs and outputs essentially), and specifically ones that are based on neuronal firing rate, rather than spike-timing models.

2.1 Mathematical descriptions of plasticity

Consider a postsynaptic neuron with firing rate v, which receives synaptic connections from n presynaptic neurons, with firing rates $u_1, u_2, ...u_n$. Let the strength of these connections (also called weights) be given by $w_1, w_2, ...w_n$, respectively. The activities of the presynaptic neurons and the strength of the synapses can be written as vectors, $\boldsymbol{u}$, and $\boldsymbol{w}$, respectively.

The activity of the postsynaptic neuron then changes according to:

$\tau_r \frac{dv}{dt} = -v + \boldsymbol{w}.\boldsymbol{u}$

where τ_r is the time constant of the firing rate dynamics. When the firing rate changes much quicker than inputs arrive ($\tau_r \rightarrow 0$), reduces to:

$v = \boldsymbol{w}.\boldsymbol{u} = \sum_1^n w_i u_i$

[1]For a review, see *Malenka and Bear, 2004, https://doi.org/10.1016/j.neuron.2004.09.012*
[2]see *Abbott and Nelson, Nature Neuroscience, 2000,* for a general review.

which basically means that the activity of our postsynaptic neuron is given just by scaling each input by the strength of its connection with the neuron and then adding them all together. **This is the case we will consider here**.

Plasticity means changes in the connections between neurons, meaning that we need an equation for the dynamics of the weights. In the simplest form that Hebb's postulate would imply, this equation would be:

$\tau_w \frac{d\boldsymbol{w}}{dt} = \upsilon \boldsymbol{u}$

where τ_w is the time constant that shows how quickly the weights change. Sometimes you might also see $\eta = 1/\tau_w$, where η (the Greek letter "eta") is called the learning rate.

If firing rates and weights are constrained to be non-negative, then you can easily see that this rule quickly leads to runaway excitation, since there is a positive feedback loop between weights and activity of the neuron. Even if we constrain the weights to some upper w_{max} then all of the weights will saturate at some point, meaning that the neuron will no longer exhibit any selectivity between its inputs.

Also, following this rule, there is only potentiation (analogous to LTP) of the synapses. To additionally model depression (LTD), we can use a modified version of the above rule, called the **covariance rule**:

$\tau_w \frac{d\boldsymbol{w}}{dt} = (\upsilon - \theta_\upsilon)\boldsymbol{u}$ where θ_υ is a threshold that controls what level of activity is needed to have potentiation of connections instead of depression. Note that this rule is still a positive feedback loop and so inherently unstable.

2.2 Imposing competition between connections

To prevent saturation of all weights and loss of selectivity, we have to introduce competition between different synapses, such that when one strengthens another has to weaken. Various ways have been proposed to do this.

- **BCM rule.** Bienenstock, Cooper and Munro[3] proposed a theory that explains the emergence of ocular dominance (neurons in V1 being selective for input from one of the two eyes) and orientation selectivity in the visual system, as well as what is seen in various induced-plasticity paradigms. The weight dynamics is given by:

  $\frac{d\boldsymbol{w}}{dt} = \eta \upsilon \boldsymbol{u}(\upsilon - \theta_\upsilon)$

  This, by itself, is not very different from the covariance rule. Indeed, it would also be unstable, were it not for the essential difference that the authors introduced, which is a *sliding* threshold. So the dynamics of the threshold is given by:

  $\tau_\theta \frac{d\theta_\upsilon}{dt} = \upsilon^2 - \theta_\upsilon$, where, as always, τ_θ is the time constant.

  In essence, when the firing rate becomes too high, it increases the threshold for potentiation, which has the effect of depressing a number of synapses. Thus, we have an implicit competition between synapses.

- **Subtractive normalization**. A more explicit way to introduce competition between synapses is to set the sum of all weights to a constant value. The sum of weights can be written as $\sum w_i = \boldsymbol{o}.\boldsymbol{w}$, where $\boldsymbol{o}$ is an n-dimensional vector of ones (and then the dot product gives a scalar value which is the sum of the values in vector $\boldsymbol{w}$). To accomplish the constraint, the learning rule is written as: $\tau_w \frac{d\boldsymbol{w}}{dt} = \upsilon \boldsymbol{u} - \frac{\upsilon(\boldsymbol{o}.\boldsymbol{u})\boldsymbol{o}}{n}$

  What this essentially means is that from the weight changes that would happen under the naive Hebb rule we subtract a constant amount that depends on the sum of all weights.

- **Oja's rule.**[4] This is an instance of **multiplicative normalization**, because for each weight the difference from the naive Hebb rule is *proportional* to the the weight itself. The dynamics is given by:

  $\tau_w \frac{d\boldsymbol{w}}{dt} = \upsilon \boldsymbol{u} - \alpha \upsilon^2 \boldsymbol{w}$, where α is a positive constant.

[3]See *EL Bienenstock, LN Cooper and PW Munro, 1982.* DOI: https://doi.org/10.1523/JNEUROSCI.02-01-00032.1982

[4]*Oja, E. J. Math. Biology (1982) 15: 267.* https://doi.org/10.1007/BF00275687

3 Model Setup

Here we will model just one linear neuron which receives two inputs for simplicity. Consider the case where there are two possible input patterns: $u = (1, 0)$ or $u = (1/2, \frac{\sqrt{3}}{2})$. You can initialize the weights at some value (a good starting point is 0.5 for both). Let $\tau_w = 100$ to start with (from this you can calculate η as well).

- Decide on a duration for your simulation and the timestep size. (I would suggest simulations with >1000 timesteps)
- Create an array that holds the input pattern the neuron receives at each timestep. The function *randsample* might come in handy.
- Create an array that will hold the weights of each connection at each timestep.
- Create an array that will hold the activity of the postsynaptic neuron at each timestep.

The above you will need for the simulation of every different rule. You might need additional structures specific to each rule. Remember that it's best to work with arrays in MATLAB instead of loops over values.

4 Simulating the naive Hebbian learning

Since this is the vanilla case, don't spend too much time in this section. It's mostly so that you get a feel for how the simulation should work. The essence of this lab is mostly in the later sections.

- First, constrain the weights between a w_{min} and a w_{max}. Then simulate the naive Hebb rule. Remember that you need to update the weights one timestep at a time, taking the previous timepoint's value as input to calculate the current one. Plot the evolution of the weights with time. Discuss your observations with your neighbours.
- Try different initial weights (and different for each neuron as well). Discuss your expectations *before* you run the simulation.
- Change the learning rate as well. Are your expectations matching the results?
- Now, simulate the covariance rule. Try this with different thresholds. (You can try starting with $\theta_v = 1$)

5 Simulating the BCM rule

Use the same neuron, with the same inputs, with an initial $\theta_v = 1$ and $\tau_\theta = 10$, to simulate the BCM rule.

- Before the simulation try to imagine what will happen to the weights. What is the highest value that a weight can take in the steady state? Make sure that your neuron is actually reaching the steady state.
- Plot the evolution of weights with time.
- Plot the evolution of the threshold θ_v with time.

After the previous steps, hopefully you'll end up with something like Figure 1, below.

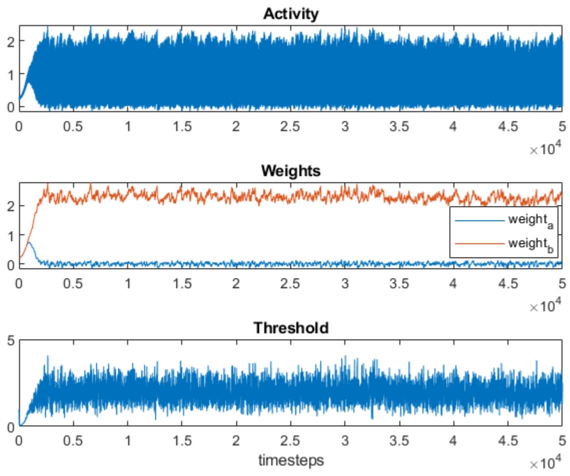

Figure 1: BCM rule results

- (Bonus but encouraged) If you want to try something more fancy, you can visualize phase portrait of the system. Look at the function *quiver* in MATLAB.
- Try different initial weights for the inputs, as well as different initial thresholds.
- What effect does the learning rate have on the evolution of the system?
- Explore what the effect of τ_θ is. Try a number of different values. You can get some really interesting patterns. You might even see something cool, like in Figure 2. (The parameters that produce Figure 2 are not the same that produce Figure 1!!) Discuss what you see and try to explain the behaviour in each case.

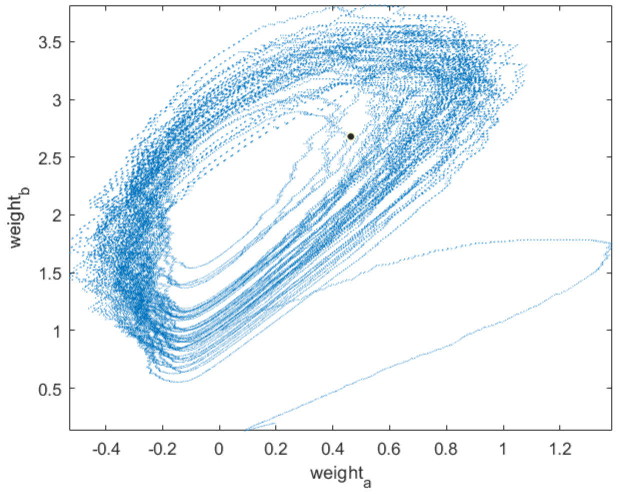

Figure 2: Phase portrait of the BCM rule

6 Subtractive Normalization

Now, try to simulate the subtractive normalization rule. In our case the second term of the right-hand side of the equation would just be: *v*sum(weights)*ones(1, n)/n.* If this is in some way unclear, ask for further explanations. Again try to explore the effect that different parameters have on the evolution of the system. Try to plot the phase portrait.

7 Oja's rule

Finally, simulate Oja's rule. Explore the effect of α. How does this rule differ from the other ones you have simulated?

8 (Extra)

- Try giving the neuron random inputs, instead of the fixed ones. Make these inputs positively or negatively correlated.
- Simulate a neuron that receives more than 2 connections, for example from 8 neurons representing different orientation angles. This would look at the emergence of orientation selectivity.