

# Computational Model of Equivalence Class Formation based on Hebbian Learning

Ángel Eugenio Tovar

UNAM

aetovar@unam.mx ; eugenio.tovar@gmail.com

This is a guide on how to use this simulator of equivalence class formation.

## BACKGROUND

The simulator is built as an artificial neural network and focuses on exploring the effects of training protocols and learning restrictions on the “relatedness” between trained and transitive relations in equivalence classes.

Notice that since the connections between stimuli (e.g., A and B) are bidirectional in the model, the symmetry relations are developed with equal values as the trained relations.

The simulator creates an artificial neural network with a single layer of fully connected neurons that uses localist representations; it creates as many neurons as stimuli declared for a given experiment.

The learning algorithm of this simulator is a Hebbian rule that implements strengthening and weakening of connections to stabilize and limit weight values. The algorithm combines benefits from supervised and reinforcement learning, and weight changes are biologically viable following notions from long-term potentiation, long-term depression, and metaplasticity.

See the full rationale and theoretical background of the learning algorithm in:

- Tovar, Á. E., & Westermann, G. (2017). A Neurocomputational Approach to Trained and Transitive Relations in Equivalence Classes. *Frontiers in Psychology*, 8. <https://doi.org/10.3389/fpsyg.2017.01848>
- Tovar, Á. E., Westermann, G., & Torres, A. (2018). From altered synaptic plasticity to atypical learning: A computational model of Down syndrome. *Cognition*, 171, 15-24. <https://doi.org/10.1016/j.cognition.2017.10.021>

There are slight differences in the activation functions and the weakening (long-term depression, LTD) of weights between the aforementioned papers. This simulator implements weakening as in Tovar et al., (2018) and the activation function is a simplified equation ( $act = net\_input / 1 + net\_input$ ) that produces values that are easier to manage within the algorithm, consequently typical threshold values must be set around 0.35.

## SOFTWARE

The model is implemented in **Matlab**, and it runs in this software and in **Octave** (free software). We additionally suggest using Excel during the designment of the simulation experiments.

## SCRIPT

The simulation uses one function only to run equivalence experiments. The function is called **hebbian\_equivalence** (in the file `hebbian_equivalence.m`) and runs both training and test trials.

The function accepts 10 input arguments and returns 3 outputs. Additional input and output arguments can be added by users.

## SYNTAX

```
[W,Test_relatedness,W_epochs] = hebbian_equivalence(S,C,T,...  
epochs,random,LTP_threshold,beta,W,Stest,Ttest)
```

### *Input Arguments*

The first three input arguments; **S**, **C**, and **T**, are matrices of **binary values** (can accept continuous values as well) with the codification of **training trials**. **S** presents the codification of sample stimuli, **C** the codification of comparison stimuli and **T** the codification of target responses (i.e., the right comparisons).

**S**, **C** and **T**, must be the same size, with rows representing each trial, and columns representing each stimuli in the experiment. Codification of trials in rows must match between matrices (e.g., the comparisons in the *i*th row of **C** are the

ones that the modeler wants to present along with the sample in the  $i$ th row in  $S$ , and the correct comparison of this trial must be codified in the  $i$ th row in  $T$ ).

The next input arguments declare the learning parameters and experimental settings:

**epochs** – are declared with an integer value (min = 1). Each epoch presents the total number of training trials codified in the training matrices.

**random** – set to 1 to determine that the training trials are presented in random order, otherwise the training trials are presented in the order determined in the training matrices.

**LTP\_threshold** – this value models the LTD/LTP threshold. Unit coactivations above the threshold lead to weight strengthening, otherwise lead to weight weakening. Increasing this value allows simulations of learning disabilities. In this simulator we suggest modeling “typical” thresholds with values around 0.35.

**beta** – is the learning rate for training trials. We suggest typical values around 0.25

**W** – Is the weight matrix. Input a previously trained Weight matrix (e.g., to run an additional training stage), or set this argument to 0 to initialize a new simulation (e.g., to simulate a “new” participant).

The last two input arguments; **Stest** and **Ttest**, are used for presentation of **test trials**. These tests just need the specification of what stimulus is the sample, codified in  $S_{test}$ , and what stimulus, codified in  $T_{test}$ , is the comparison under evaluation (either the correct or an incorrect comparison, depending on the research question).  $S_{test}$  and  $T_{test}$  must be the same size, with rows representing each test trial, and columns representing each stimuli in the experiment. Columns must be the same size as those declared during training. Codification of test trials must match in rows between these matrices.

### ***Output Arguments***

There are three output arguments;

**W** is the weight matrix, its structure is that of a correlation matrix, and it shows relatedness values between all possible stimuli pairs in the experiment. Relatedness is computed in the range 0 – 1, with values close to 1 representing higher relatedness between stimuli.

The **Test\_relatedness** output is a matrix with relatedness (W) values for each test trial (in rows) as evaluated after each training epoch (in columns).

**W\_epochs** is a 3 dimensional matrix with all W matrices obtained after each training epoch.

### **WORKING EXAMPLES**

#### **1. Simulation of a training procedure for the establishment of three classes composed of three members each.**

This experiment explores the learning of AB and BC baseline relations, and the emergence of transitive AC in three stimulus classes. The training consists of the presentation of blocks with the next stimulus relations: A1B1, B1C1, A2B2, B2C2, A3B3, B3C3 trials. During each training trial three comparison stimuli are presented; for example, training of A1B1 is presented with A1 as the sample, and B1, B2, and B3 are presented as comparisons.

This procedure considers that all baseline relations are randomly ordered in blocks. And a total of 30 training blocks are presented.

The above description already contains all the information needed for this simulation.

#### ***Preparation of matrices***

We suggest using Excel for this step because in Excel some cells can be used to label the matrix rows and columns and this is a bit more tricky in Matlab,

remember that we will not use labels in the Matlab matrices. Also, modifying training and test matrices is easier this way. You can use the **working\_examples.xlsx** file that includes the matrices used for these examples.

Let's start by designing the training matrices. We already know that each training matrix is size 6x9 (6 rows and 9 columns), because the experimenter will present blocks of 6 trials and there are a total o 9 stimuli. The structure of the sample matrix looks as follows:

Training Trial	Sample								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
1									
2									
3									
4									
5									
6									

The stimuli are ordered along the columns, and trials are listed in the rows. The order of stimuli in columns is flexible as long as it matches the order in the comparison and target matrices. Actually, the structure of the sample, comparison and target matrices is exactly the same.

The blue area in the above figure will contain the binary codification of samples, and the labels presented in bold letters are only for guidance and will not be used in Matlab.

Codification of the first trial, A1B1 requires presentation of A1 as a sample, and none of the other stimuli is presented. This is codified as follows

Training Trial	Sample								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
1 (A1B1)	1	0	0	0	0	0	0	0	0

We can proceed to codify the comparison stimuli in the first vector of the comparison matrix. Since this trial presents B1, b2 and B3, it should look as follows:

Training Trial	Comparison								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
1 (A1B1)	0	1	0	0	1	0	0	1	0

We use a green area for the comparison vectors to highlight that this is a different matrix.

Finally, we need to specify which one of the comparisons is the target stimulus. We codify B1 in the target matrix as follows:

Training Trial	Target								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
1 (A1B1)	0	1	0	0	0	0	0	0	0

We use an orange area for the target vectors to highlight that this is a different matrix.

Following this procedure, we can design all the vectors for the reaming trained relations. The final training matrices should look as follows:

Training Trial	Sample								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
A1B1	1	0	0	0	0	0	0	0	0
B1C1	0	1	0	0	0	0	0	0	0
A2B2	0	0	0	1	0	0	0	0	0
B2C2	0	0	0	0	1	0	0	0	0
A3B3	0	0	0	0	0	0	1	0	0
B3C3	0	0	0	0	0	0	0	1	0

Training Trial	Comparison								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
A1B1	0	1	0	0	1	0	0	1	0
B1C1	0	0	1	0	0	1	0	0	1
A2B2	0	1	0	0	1	0	0	1	0
B2C2	0	0	1	0	0	1	0	0	1
A3B3	0	1	0	0	1	0	0	1	0
B3C3	0	0	1	0	0	1	0	0	1

Training Trial	Target								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
A1B1	0	1	0	0	0	0	0	0	0
B1C1	0	0	1	0	0	0	0	0	0
A2B2	0	0	0	0	1	0	0	0	0
B2C2	0	0	0	0	0	1	0	0	0
A3B3	0	0	0	0	0	0	0	1	0
B3C3	0	0	0	0	0	0	0	0	1

Let's now create the matrices for the test trials.

Consider a simple scenario where we are only interested in knowing whether AC relations emerged in the model. To evaluate A1C1, we need to codify a test trial where the sample is A1 in the Stest matrix, and the target is C1 in the Ttest matrix, as in the following figures:

Test Trial	Sample test								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
A1C1	1	0	0	0	0	0	0	0	0

Test Trial	Target test								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
A1C1	0	0	1	0	0	0	0	0	0

The test trials A2C2 and A3C3 complete the test matrices which should look like these:

Test Trial	Sample test								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
A1C1	1	0	0	0	0	0	0	0	0
A2C2	0	0	0	1	0	0	0	0	0
A3C3	0	0	0	0	0	0	1	0	0
Test Trial	Target test								
	A1	B1	C1	A2	B2	C2	A3	B3	C3
A1C1	0	0	1	0	0	0	0	0	0
A2C2	0	0	0	0	0	1	0	0	0
A3C3	0	0	0	0	0	0	0	0	1

We can now go to Matlab to run the simulation.

Open Matlab and go to *command window*. From this window you can create the variable names of the training and test matrices. Use zero values just to create them all, as in the next example:

Type the next lines in *Command Window*

```
>> sample_train = 0;
```

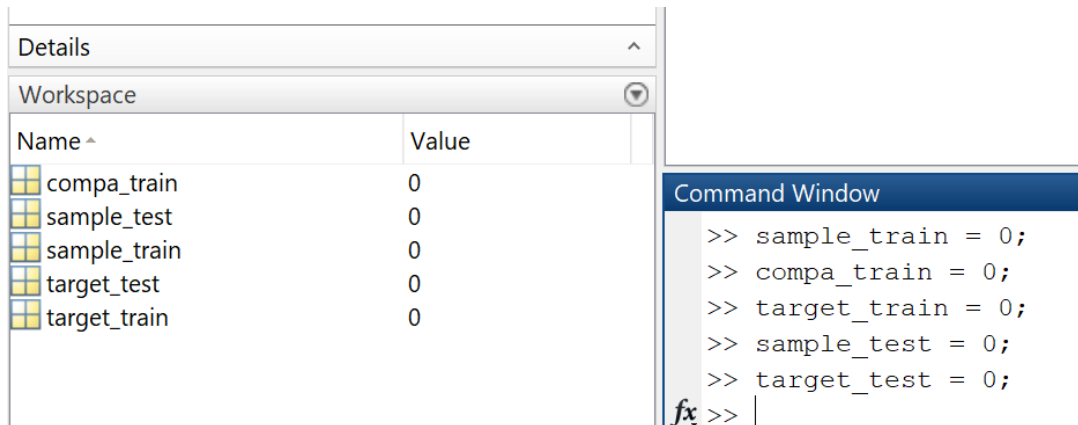
```
>> compa_train = 0;
```

```
>> target_train = 0;
```

```
>> sample_test = 0;
```

```
>> target_test = 0;
```

It should look like this:



You will see each variable listed in the *Workspace* after you create it in *Command Window*. Once you have created them, open one of them by double clicking on its name in the *Workspace*. This will open this variable in the *Variable Editor* of Matlab, which looks like a spreadsheet. For example, double click on “sample\_train” will open this:

Variables - sample\_train

sample\_train

1x1 double

	1	2	3	4
1	0			
2				
3				
4				
5				
6				

Now go to the Excel file, copy the sample vectors (only the colored area without the labels!) that you designed for the training phase, and copy them in the *Variable Editor* of Matlab. The result will look like this:

Variables - sample\_train

sample\_train

6x9 double

	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	0	1	0

Notice that colors and other properties will not be copied from Excel to Matlab.



Do the same for the remaining matrices.

Now everything is ready to run the simulation! go to *Command Window* in Matlab and run the simulator by declaring all input arguments of the function and choose the names for the desired outputs.

You can run this line:

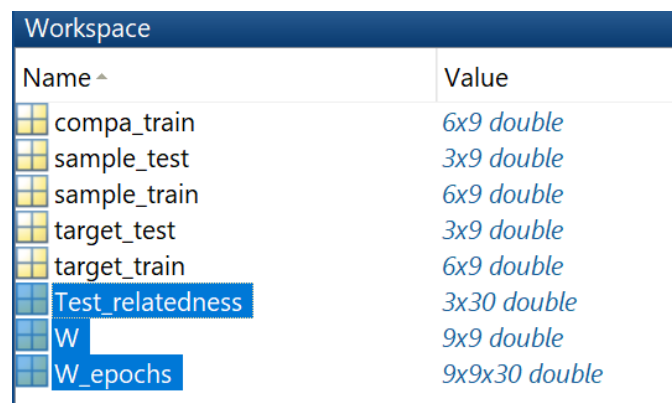
```
[W,Test_relatedness,W_epochs] = hebbian_equivalence(sample_train,...  
compa_train,target_train,30,1,.35,.2,0,sample_test,target_test)
```

Notice that apart from the training and test matrices the above line declares 30 epochs (training blocks), in which the order of trials is randomly determined (declared with 1), we use an LTP\_threshold of 0.35, a learning rate of 0.2, and we initialize weights at 0 values.

Pressing Enter will run this simulation.

## Analyzing Simulation Results

After running the simulation, three output matrices will be included in the *Workspace*. You can either call them by typing their names in *Command Window* or open them in the *Variable Editor* by double clicking on them.



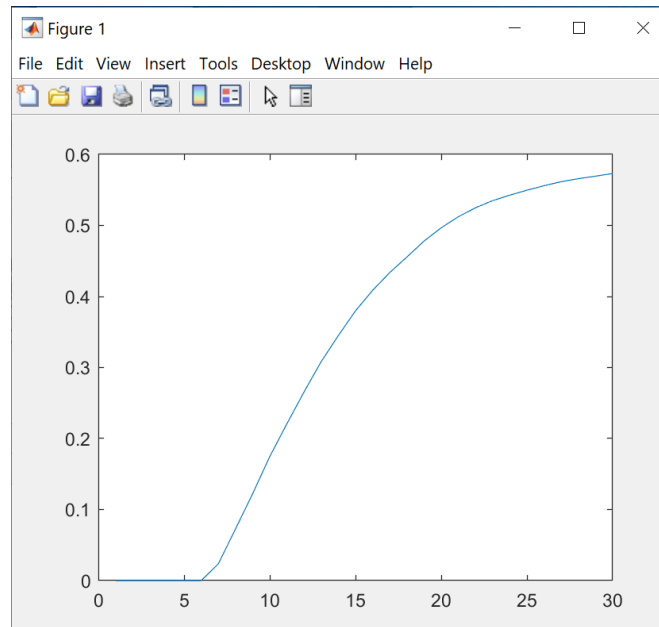
Name ^	Value
compa_train	6x9 double
sample_test	3x9 double
sample_train	6x9 double
target_test	3x9 double
target_train	6x9 double
Test_relatedness	3x30 double
W	9x9 double
W_epochs	9x9x30 double

Let's focus for this example on the Test\_relatedness matrix. Take a look at the first row of this matrix. This row contains the modeled value of relatedness between A1 and C1, which compose our first transitive trial A1C1. Its 30 columns show how relatedness between A1 and C1 developed after each of the 30 training epochs. To visualize this growing relatedness you can plot this value with the *plot* function built

in Matlab. Just indicate that you want to plot all the columns from the first row of this matrix with the next command line:

```
plot(Test_relatedness(1,:))
```

And you will see the next plot:



As you can notice, the relatedness between A1 and C1, increases as the training continues and its trajectory shows a typical learning curve. The last relatedness value between A1 and C1 in this example is 0.5731 (each simulation varies because this is a stochastic network).

Run the next command line in Command Window to create a plot with all the test trials:

```
>> plot(Test_relatedness')
```

Let's now take a look at the second output matrix W. We suggest to copy this matrix and paste it in Excel, where you can include labels for the rows and columns. Since W has the same structure as a correlation matrix, prepare the labels in a way that the label columns are the same as those designed for the training and test matrix. And the row labels are the transpose of them. Look at the next figure taken from Excel, which already has the labels and the W matrix from Matlab.

	A1	B1	C1	A2	B2	C2	A3	B3	C3
A1	0	0.8327	0.5731	0	0	0	0	0	0
B1	0.8327	0	0.8603	0	0	0	0	0	0
C1	0.5731	0.8603	0	0	0	0	0	0	0
A2	0	0	0	0	0.8494	0.5796	0	0	0
B2	0	0	0	0.8494	0	0.8604	0	0	0
C2	0	0	0	0.5796	0.8604	0	0	0	0
A3	0	0	0	0	0	0	0	0.8681	0.5779
B3	0	0	0	0	0	0	0.8681	0	0.8468
C3	0	0	0	0	0	0	0.5779	0.8468	0

This W matrix shows the final connection weights or relatedness values in the network after 30 blocks of training. In the figure, the trained and transitive relations from Class 1 are highlighted. Notice that the relatedness between A1 and C1 matches the last value in the Test\_relatedness matrix described above. You can also see that, for all classes, relatedness between transitive relations (AC) is always lower than relatedness between trained relations (AB and BC).

*Examples on running experiments of nodal distance and evaluation of trained and derived relations are under development for future updates of this document.*