BCM Learning Rule

Report for the final project of the course "Neural Information Processing" in SoSe 2025 by Julia Hattendorf.

Introduction

The BCM (Bienenstock, Cooper, and Munro) rule was introduced in 1982 to model the development of stimulus sensitivity in neurons. In the visual cortex, for example, neurons become selectively responsive to specific orientations or spatial features based on their sensory experience. The rule proposes that a neuron's synaptic strengths are potentiated (LTP) or depressed (LTD) based on its average activity over time.

The Model

Note: Variable names from the original paper have been adjusted to modern machine learning conventions for readability.

The forward pass is the usual weighted sum:

$$y = \sum_i w_i x_i$$

The BCM rule differs to hebbian learning in that the update of the weight is not a product of the presynaptic and postsynaptic activity but of the presynaptic activity, x, and a modification function, ϕ, of the postsynaptic activity, y, and a threshold θ. Additionally, the authors allow the option for a weight decay term $-\epsilon w_i$ with a variable of decay ϵ. For the ith weight, the update looks like this:

$$\frac{dw_i}{dt} = \phi(y)x_i - \epsilon w_i$$

The modification function and its threshold, θ, make sure that the weight adaption relates to the postsynaptic activity: If the activity of a neuron low $(y < \theta)$ then its weight update will be negative. Vice versa, if it is higher $(y > \theta)$ the weight update will be positive. The weights should thus adapt to favor some inputs over others, so the rule induces competition between inputs.

The threshold itself also adapts based on the postsynaptic activity. So if the activity rises or falls so will the threshold. Ideally, the threshold would then prevent runaway activation.

In the original paper the threshold is calculated with the temporal average value, $E(\cdot)$, of the activity y and a positive constant y_o. The average value is additionally taken to the power of another positive constant p.

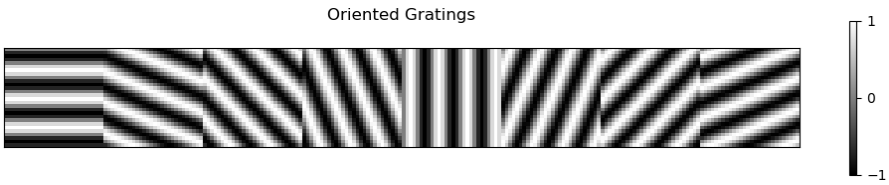
$$\theta = E^p[(y/y_o)]$$

Interestingly, after this threshold model was introduced, biological evidence for it could be found in experiments (Cooper & Bear, 2012).

Implementation & Problems

Dataset

For the dataset I chose synthetic oriented gratings as inputs. I then chose to construct an MNIST-like dataset, with 28x28 pixel per image and to repeat the orientations to have 400 samples in total. The dataset then had the shape (400, 784).



8 orientations evenly spaced over 180°.

Implementation Choices

I implemented the BCM learning rule in a class with PyTorch tensors. This allows for easy access of hyperparameters and traces. As is convention, I also implemented the update to be calculated on data batches (SGD-like).

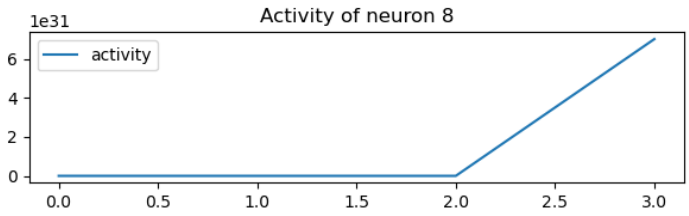
For the threshold, instead of a temporal average, I calculated the average of output activity per neuron over the input batch (using spatial instead of temporal average is discussed by Intrator & Cooper, (1992)).

For the modification function some sources state that the original function used in the paper is $\phi = y(y - \theta)$ (Udeigwe et al., 2017) (Blais & Cooper, 2008). Although I could not confirm this with the paper, I chose it as a starting point.

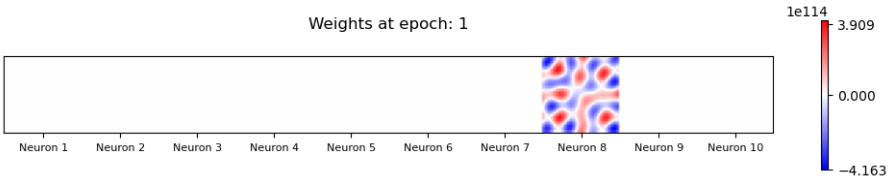
Problem: Runaway Activation

The BCM rule is supposed to solve runaway activity in hebbian learning but when I started implementing this rule, I had a big problem: runaway activation. When using the function $\phi = y(y - \theta)$ it is quite apparent that there is no upper bound to modifications and one can only rely on the threshold.

Running the model for 1 epoch, with 10 neurons and a batch size of 100:



The activity of a single neuron while training.



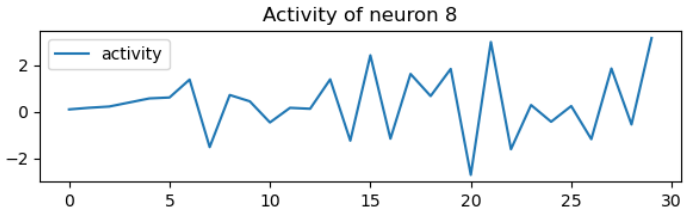
The weights of all neurons after training.

Notice that one neuron is dominating. In some runs, weights of other neurons are growing too but generally the weights of one neuron dominate. My interpretation of this is that the neuron forms weights that are good enough to be a "catch-all" for many different inputs. For example, the weights of neuron 8 above in the figure could activate for vertical, horizontal, and diagonal input patterns. Having a catch-all weight matrix means the neuron's activity will always be above the threshold, further adjusting the weights positively. Learning for this neuron will result in a "snowball effect".

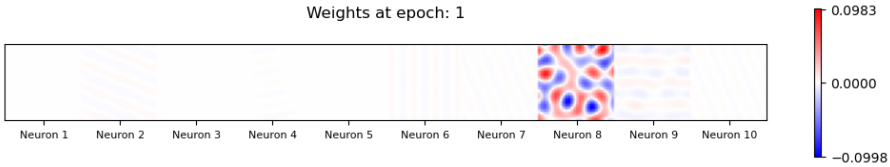
Solution 1: Weight normalization

The simplest fix for too large weights seems to be to simply normalize the weight in every update.

While that does help with the runaway activation problem, we still have the "snowball" problem of one neuron dominating:



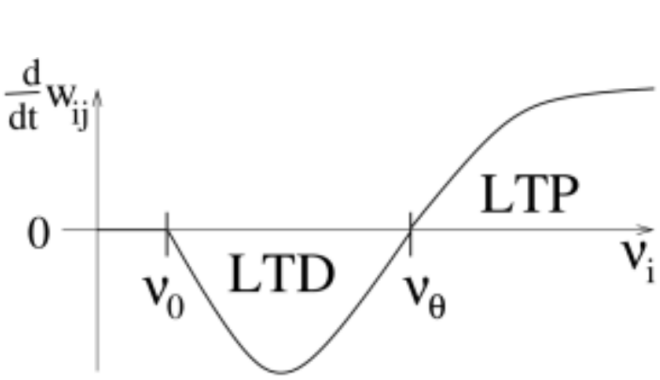
The activity of a single neuron while training.

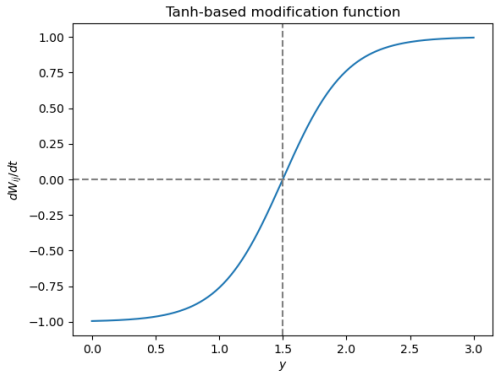


The weights of all neurons after training.

Solution 2: $\phi \rightarrow tanh$

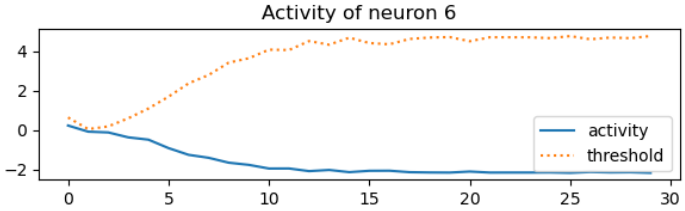
Since some sources show the modification function with a sigmoid-ish trajectory for $y > \theta$, I tried using a $tanh$-based function to model both the LTD and LTP effect.



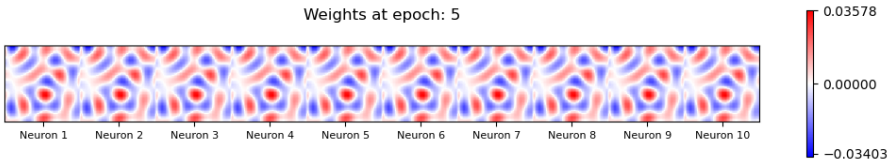


On the left the modification function according to the book Neuronal Dynamics by Gerstner et al. (2014). On the right my own variant $phi = tanh(2 * (y - \theta))$.

This also helped with the runaway problem but did not give promising results in terms of feature extraction.



The activity and threshold of a single neuron while training.



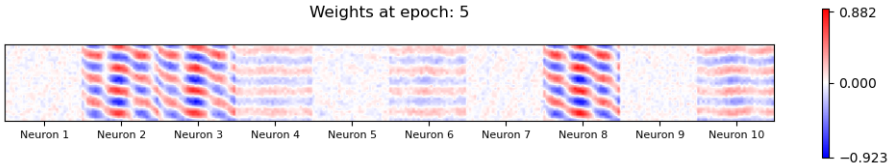
The weights of all neurons after training.

Solution 3: Change the implementation of BCM

In order to find another solution, I looked at the implementation for the paper (Squadrani et al., 2022) that can be found on the github account of Nico Curti. They use the implementation proposed by Law and Cooper in 1994. The differences to the original BCM implementation are:

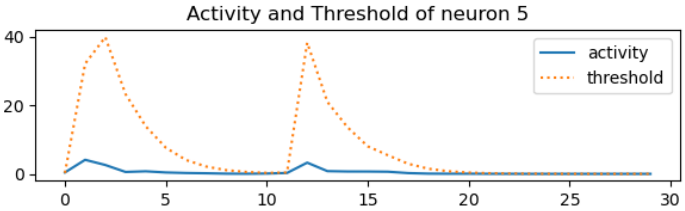
- adding ReLU as activation function: $y = ReLU(\sum_i w_i x_i)$
- calculating the threshold with the moving average: $\theta_t = \gamma\theta_{t-1} + (1-\gamma)\langle y^2 \rangle_{b_t}$ where $\langle \cdot \rangle_{b_t}$ is the average over the batch of training patterns
- dividing the weight update by the threshold: $\frac{dw_i}{dt} = \frac{y(y-\theta)x_i}{\theta}$

I found that no single of those differences is enough to ensure stability, but that all three are needed. After implementing all these changes, the form of the weight is improved and the orientations are picked up:



The weights of all neurons after training.

Additionally, the relationship of threshold and activity is clearer. Below one can clearly see how spikes in output activity are picked up by the threshold and how the activity is regulated down:

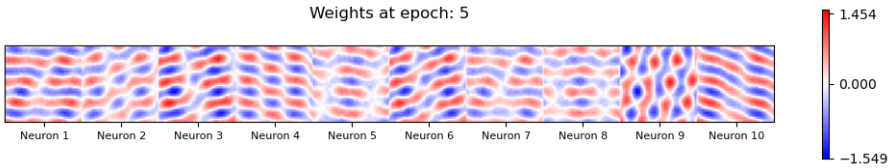


The activity and threshold of a single neuron while training.

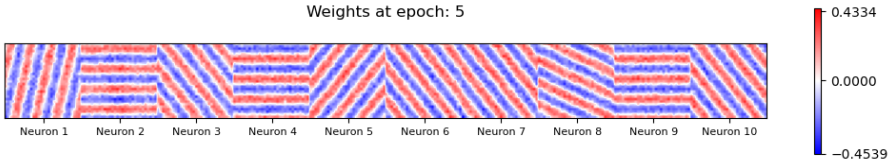
Timescales

To add another layer of complexity to this implementation, I added separate timescales for the change of threshold, τ_θ, and the weight adaption, τ_w. To find an intuition for good values, I consulted the paper by Udeigwe et al. (2017), where the authors examined the dynamical properties for different values of τ_θ and τ_w. From the paper I gathered that $\tau_\theta > \tau_w$.

I unfortunately could not set up a proper grid-search for these parameters due to time constraints and manually tried out a few combinations:



Weights for tau_th = 0.1 and tau_w = 0.05



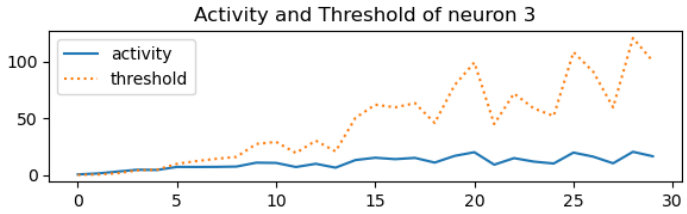
Weights for tau_th = 0.1 and tau_w = 0.001

The last combination of $\tau_\theta = 0.1$ and $\tau_w = 0.001$ is the one I will be using from now on for the analysis.

Results and Analysis

Threshold preventing runaway activity

The threshold is successfully regulating the activity. Looking at t=20 and t=25, the spike in activation is picked up by the threshold and the activation in the next step is reduced.

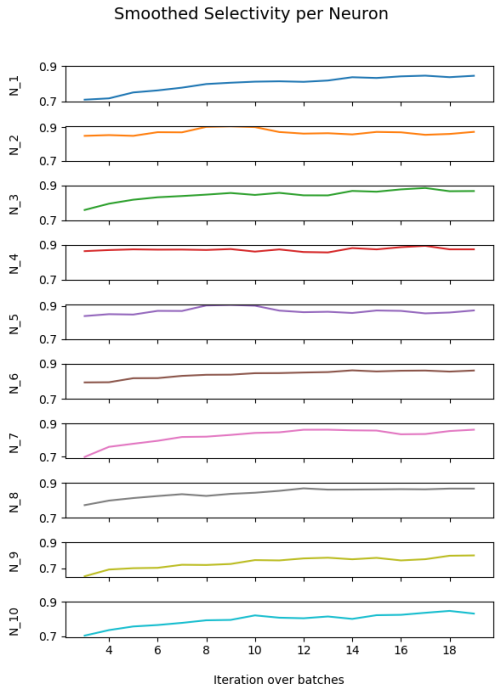


The activity and threshold of a single neuron while training.

Selectivity increases with learning

To have some measure for the "performance" of a neuron, I chose selectivity, i.e. how a neuron responds to certain stimuli compared to others. The general index for selectivity as proposed by Bienenstock, Cooper and Munro (Eq. 3, p. 33):

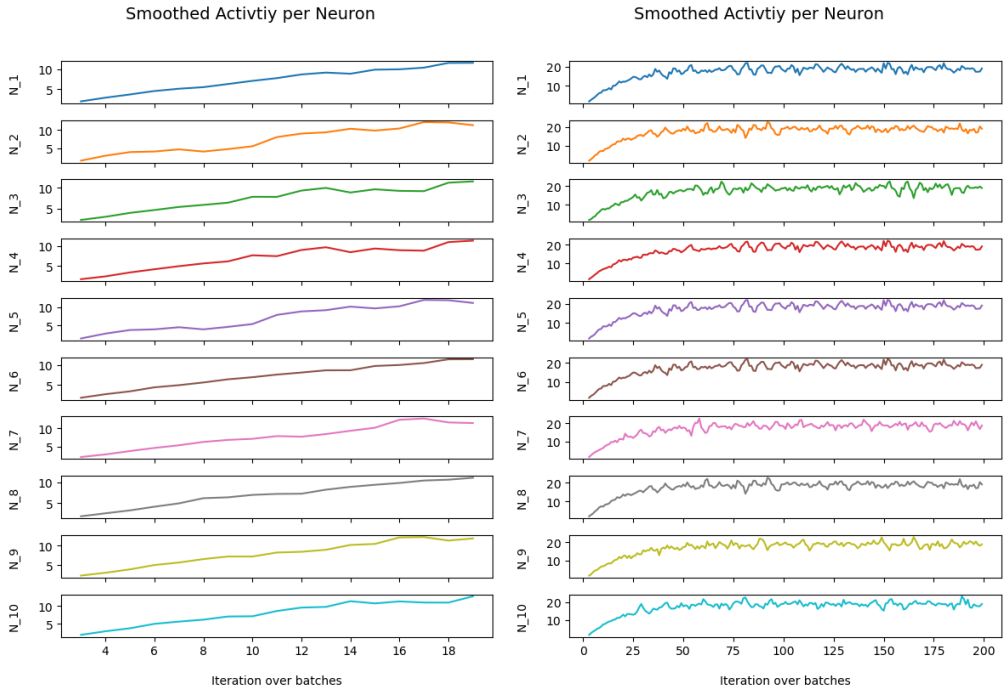
$$Sel_d(N) = 1 - \frac{\text{mean response of N with respect to d}}{\text{max response of N with respect to d}}$$



The selectivity of all neurons is increasing over training.

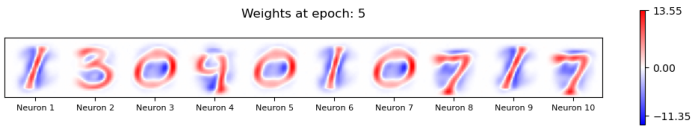
Stability

When looking at the activity of the neurons for 5 epochs (left) it looks like the runaway problem might still be there. However, training the neurons for 50 epochs (right) showed that the activity eventually converges. Both were trained with a batch size of 100.



The activity of the neurons converge. Left trained for 5 epochs, right for 50 epochs.

Additionally, I ran this model on MNIST to see if it can still perform well for a different dataset:



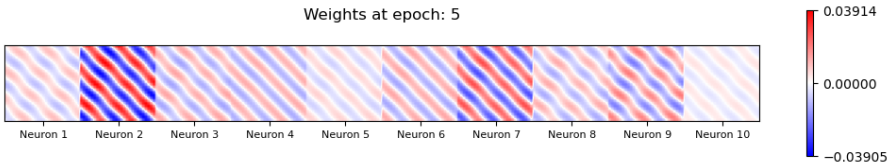
Weights for MNIST.

Comparison

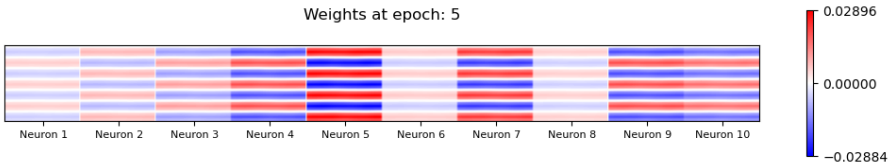
For comparison purposes, the learning rate is $\eta = 0.05$ for all models, which have been trained for 5 epochs with a batch size of 100.

Hebb with weight normalization

Training the neurons with the (normed) hebb rule seems to lead to the same input image being picked up across all neurons. In different runs, different images are dominant:



The weights of all neurons after training with hebb.

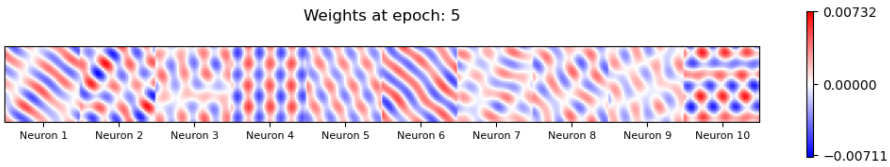


The weights of all neurons after training with hebb.

Oja

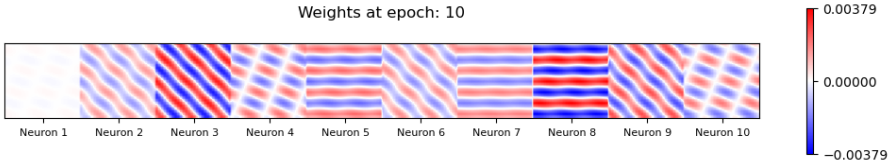
With Oja's rule I needed to train the network a little longer to achieve similar results to hebb and BCM.

For comparison, here is Oja's rule with 5 epochs:



The weights of all neurons after training with Oja's rule.

And here for 10 epochs:



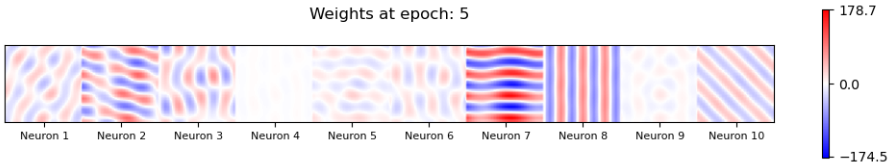
The weights of all neurons after training with Oja's rule.

Future ideas

WTA & BCM

Since BCM induces competition on an input pattern level and WTA induces competition on a neuron level, it would be interesting to try and merge them together and see what happens. However, it is noted that both models come with a potential "snowball effect" problem of one neuron being chosen over and over, either due to winning the competition or due to having a catch-all matrix and having high activations.

I attempted to combine the two, but am not quite satisfied with the results. Below are the weights trained with this combination and τ_θ = 1 and τ_w = 0.5.



The weights of all neuros after training with a WTA & BCM combination.

Investigating Timescales

My implementation has been based on the assumption that the "homeostatic" timescale, i.e. τ_θ is faster than the "learning" time scale, i.e. τ_w. I gathered this assumption from the work of Udeigwe et al. (2017). However, the authors also mention that this finding seems to be somewhat contradictory to the findings gathered from biological experiments, where the process of homeostasis seems to be slower than the process of learning. The authors also mention that there may be slow and fast homeostatic processes. It would be interesting to investigate this dilemma further.

Effect of Inhibitory Synapses

In the original paper, the authors mentioned that one could show the effect of "inhibition" by setting the negative values of the weight update to zero. It would be interesting to see how the learning is affected by this and to try "weighting" the LTD- and the LTP-part differently.

Activation functions

The BCM model does not provide an activation function for the postsynaptic activity. However, using an activation function was useful for me to achieve stable results. Trying out different activation functions and how they affect results would be interesting.

Literature

Bienenstock, E. L., Cooper, L. N., & Munro, P. W. (1982). Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. Journal of Neuroscience, 2(1), 32-48.
accessed via https://www.jneurosci.org/content/jneuro/2/1/32.full.pdf

Blais, B. S., & Cooper, L. (2008). BCM theory. Scholarpedia, 3(3), 1570.
accessed via http://www.scholarpedia.org/article/BCM_theory

Cooper, L. N., & Bear, M. F. (2012). The BCM theory of synapse modification at 30: interaction of theory with experiment. Nature Reviews Neuroscience, 13(11), 798-810.
accessed via https://brabeeba.github.io/neuralReadingGroup/cooper.pdf

Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). Neuronal dynamics: From single neurons to networks and models of cognition. Cambridge University Press.
accessed via https://neuronaldynamics.epfl.ch/online/Ch19.S2.html

Intrator, N., & Cooper, L. N. (1992). Objective function formulation of the BCM theory of visual cortical plasticity: Statistical connections, stability conditions. Neural Networks, 5(1), 3-17.
accessed via https://www.sciencedirect.com/science/article/pii/S0893608005800036

Squadrani, L., Curti, N., Giampieri, E., Remondini, D., Blais, B., & Castellani, G. (2022). Effectiveness of Biologically Inspired Neural Network Models in Learning and Patterns Memorization. Entropy, 24(5), 682.
accessed via https://pmc.ncbi.nlm.nih.gov/articles/PMC9141587/
Implementation on Nico Curti's github: https://github.com/Nico-Curti/plasticity/blob/main/plasticity/model/bcm.py

Udeigwe, L. C., Munro, P. W., & Ermentrout, G. B. (2017). Emergent dynamical properties of the BCM learning rule. The Journal of Mathematical Neuroscience, 7, 1-32.
accessed via https://mathematical-neuroscience.springeropen.com/articles/10.1186/s13408-017-0044-6