Backpropagation of Hebbian plasticity for continual learning

Thomas Miconi
The Neurosciences Institute
La Jolla, CA, USA
miconi@nsi.edu
All software used for the present article available at http://github.com/thomasmiconi

1 Introduction

Backpropagation can train neural networks to perform remarkably complex tasks. However, it is generally used to produce fixed-weights networks, with no further changes in connectivity after training. By contrast, biological brains are able to learn from experience, and alter network connections, during their lifetime. This ability to learn from experience allows the organism to deal with changing or unpredictable features of the environment.

Lifetime long-term plasticity in living brains generally follows the Hebbian principle: a cell that consistently contributes in making another cell fire will build a stronger connection to that cell. Note that this generic principle can be implemented in many different ways, including covariance learning, instar and outstar rules, BCM learning, etc. (see [4] and references therein).

Several methods have been proposed to endow artificial neural networks with long-term memories amenable to backpropagation training, including most recently neural Turing machines [1, 2] and memory networks [3]. However, it would be useful to incorporate the powerful, well-studied principle of Hebbian plasticity in backpropagation training.

Here we derive analytical expressions for activity gradients in neural networks with Hebbian plastic connections. Using these expressions, we can use backpropagation to train not just the baseline weights of the connections, but also their plasticity. As a result, the networks "learn how to learn" in order to solve the problem at hand: the trained networks automatically perform fast learning of unpredictable environmental features during their lifetime, expanding the range of solvable problems. We test the algorithm on simple tasks including pattern completion, one-shot learning, and reversal learning. The algorithm successfully learns how to learn the relevant associations from one-shot instruction, and fine-tunes the temporal dynamics of plasticity to allow for continual learning in response to changing environmental parameters.

2 Gradients in networks with Hebbian synapses

We consider networks where the strength of each connection can vary according to Hebbian plasticity over the course of the network's lifetime. Each network is fully specified by fixed parameters which determine both the baseline weight *and* the degree of plasticity of each connection. After training, these parameters are fixed and unchanging over the network's lifetime, but govern the way in which each connection changes over time, as a result of experience, according to Hebbian plasticity.

To model Hebbian plasticity, we maintain a time-dependent quantity for each connection in the network, which we call the *Hebbian trace* for this connection. As noted above, there are many possible expressions for Hebbian plasticity [4]. In this paper, we use the simplest stable form of Hebbian trace, namely, the running average of the product of pre- and post-synaptic activities. Thus,

for a given target cell, the Hebbian trace associated with its k-th incoming connection is defined as follows:

$$Hebb_k(t) = (1-\gamma) * Hebb_k(t-1) + \gamma * x_k(t) * y(t) \tag{1}$$

where $y(t)$ is the activity of the post-synaptic cell, $x_k(t)$ is the activity of the pre-synaptic cell, and γ is a time constant. While other expressions of Hebbian plasticity are possible, this simple form turns out to be adequate for our present purposes and simplifies the mathematics.

The Hebbian trace is maintained automatically, independently of network parameters, for each connection. Given this Hebbian trace, the actual strength of the connection at time t is determined by two fixed parameters: a fixed weight w_k, which determines the "baseline", unchanging component of the connection; and a *plasticity parameter* α_k, which specifies how much the Hebbian trace influences the actual connection. More formally, the response y of a given cell can be written as a function of its inputs as follows (b is a bias parameter):

$$y(t) = \tanh\left\{\sum_{k \in inputs} [w_k x_k(t) + \alpha_k Hebb_k(t-1) x_k(t)] + b\right\} \tag{2}$$

In order to use backpropagation, we must find the gradient of y over the w_k and α_k parameters. Crucially, these gradients will necessarily involve activities at previous times, because under plasticity, neural responses at time t influence future responses at times $t+n$ due to their effects on the Hebbian trace. Fortunately, these gradients turn out to have a simple, recursive form.

Omitting the $\tanh$ nonlinearity for clarity, we get the following expressions:

$$\frac{\partial y(t_z)}{\partial w_k} = x_k(t_z) + \sum_{l \in inputs} [\alpha_l x_l(t_z) \sum_{t_u < t_z} (1-\gamma)\gamma^{t_z - t_u} x_l(t_u) \frac{\partial y(t_u)}{\partial w_k}] \tag{3}$$

$$\frac{\partial y(t_z)}{\partial \alpha_k} = x_k(t_z) Hebb_k(t_z) + \sum_{l \in inputs} [\alpha_l x_l(t_z) \sum_{t_u < t_z} (1-\gamma)\gamma^{t_z - t_u} x_l(t_u) \frac{\partial y(t_u)}{\partial \alpha_k}] \tag{4}$$

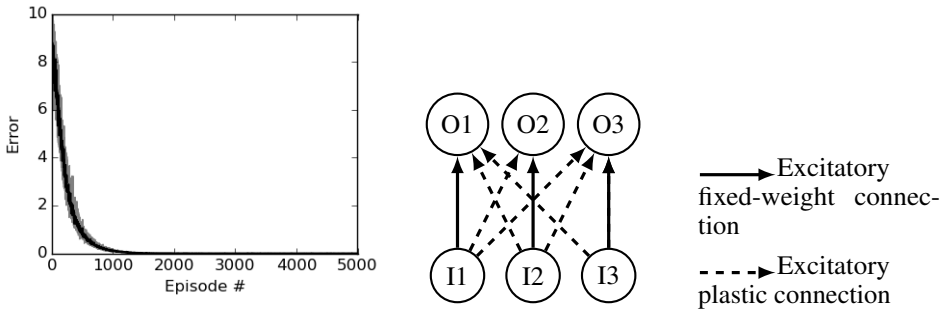
These equations express the gradient of $y(t_z)$ as a function of the gradients of $y(t_z < t_u)$, that is, recursively. To include the $\tanh$ nonlinearity, we simply apply the chain rule with the well known identity $\frac{\partial \tanh(y)}{\partial y} = (1 - \tanh(y)^2)$.

3 Experiments

In all tasks described below, lifetime experience is divided into *episodes*, each of which lasts for a certain number of timesteps. At the beginning of each episode, all Hebbian traces are initialized to 0. Then, at each timestep, the network processes an input pattern and produces an output according to its current parameters, and the Hebbian traces are updated according to Equation 1. Furthermore, errors and gradients are also computed. At the end of each episode, the errors and gradients at each timestep are used to update network parameters (weights and plasticity coefficients) according to error backpropagation and gradient descent. The whole process iterates for a fixed number of episodes. For the last 500 episodes, training stops and networks parameters are frozen.

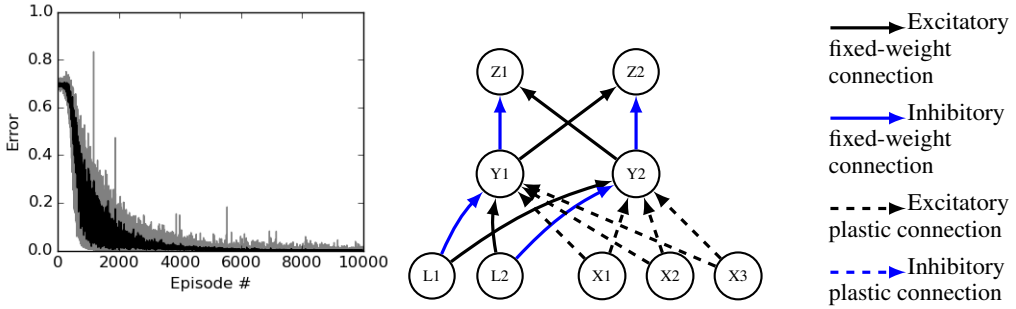
3.1 Pattern Completion

To test the BOHP method, we first apply it to a task for which Hebbian learning is known to be efficient, namely, pattern completion. The network is composed of an input and an output layer, each having N neurons. In every episode, the network is first exposed to a random binary vector of length N with at least one nonzero element. This binary vector represents the pattern to be learned. Then, at the next timestep, a partial pattern containing only one of the non-zero bits of the pattern (all other bits set to 0) is presented. The task of the network is to produce the full pattern in the output layer.



(a) Training error (b) Typical trained network

Figure 1: Results for the pattern completion experiment. (a) Mean absolute error per timestep over each episode, for mutually exclusive stimuli. The dark line indicates median error over 20 runs, while shaded areas indicate interquartile range. For the last 500 episodes, training is halted and parameters are frozen. (b) Schema of a typical network after training. Only 3 elements shown for clarity (actual pattern size: 8 elements).



(a) Training error (b) Typical trained network

Figure 2: Results for the one-shot learning experiment. (a) Median absolute error per timestep over each episode. Conventions are as in Figure 1. (b) Schema of a typical network after training. In addition to the label nodes L1 and L2, only 3 pattern elements shown for clarity (actual pattern size: 8 elements). See text for details.

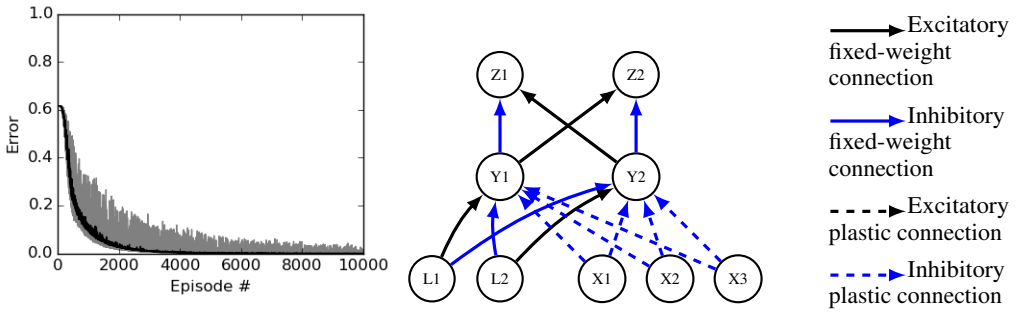
The error for each episode is the Manhattan distance between the network's output at the second time step and the full pattern (network response during the first step is ignored).

The algorithm reliably learns to solve the task, and the final networks after training show an easily interpretable structure (Figure 1).

3.2 One-shot learning of arbitrary patterns

In this task, at each episode, the network must learn to associate each of two random binary vectors with its label. The labels are simply two-element vectors, set to 01 for one of the patterns, and 10 for the other. Importantly, learning is one-shot: at the first timestep, the input consists of the first pattern, suffixed with label 01; and at the second timestep, the input vector is the second pattern, suffixed with label 10. These are the only times the labels are presented as inputs: at all other times, the input is one of the patterns, suffixed with the neutral sufix 00, and the network's output must be the label associated with the current pattern.

Patterns are random vectors of N elements, each having value 1 or -1, with at least one position differing between the two patterns to be learned ($N = 8$ for all experiments). The networks have an input layer ($N+2$ nodes), a hidden layer (2 nodes), and an output layer (2 nodes). For simplicity, only the first layer of weights (input-to-hidden) can have plasticity. The final layer implements softmax competition betwen the nodes. Each episode lasts 20 timestep, of which only the first two contain the expected label for each pattern. We use cross-entropy loss between the output values and the expected label at each time step, except for the 2-step learning period during which network output is ignored.



(a) Training error (b) Typical trained network

Figure 3: Results for the reversal learning experiment. Conventions are as in Figure 1. (a) Mean absolute error per timestep over each episode. (b) Schema of a typical network after training. Notice the negative plasticity connections from the pattern nodes to the hidden layer. See text for details.

Again, the algorithm reliably learns to solve the task (Figure 2). The trained networks are organized in such a way that one hidden node learns the pattern with label 01, and the other learns the pattern associated with label 10: they receive strong, fixed (positive and negative) connections from the label bits, but receive only strong plastic connections (with zero fixed-weight connections) from the pattern bit. This allows each node to be imprinted to the corresponding pattern. The weights between hidden and top layer ensures that the top two nodes produce the adequate label.

Note that some networks displayed a somewhat different pattern where all connections between pattern nodes and hidden nodes has negative plasticity coefficients. We discuss this in the next section.

3.3 Reversal learning

Previous experiments show that the algorithm can train networks to learn fast associations of environmental inputs. But can it also teach networks to adapt to a changing environment - that is, to perform continual learning over their lifetime?

To test this, we adapt the previous one-shot learning task into a continual learning task: halfway through each episode, we invert the two patterns, so that the pattern previously associated with label 01 is now associated with label 10, and vice-versa. We show each of the pattern with its updated label once. Then we resume showing input patterns with neutral, 00 suffixes, and expect the network's output to be the adequate new label for each input pattern.

The algorithm also successfully learns to solve this problem (Figure 3). The networks are somewhat similar to the ones obtained in one-shot learning, but with an important difference: connections from pattern input to hidden nodes now consistently have *negative* plasticity coefficients. This feature appears crucial for reversal learning, because clipping plasticity coefficients to positive value prevents successful learning in this task (though it has no effect on the one-shot learning task). Why does reversal learning seem to require negative plasticity?

It is well-known thath Hebbian plasticity creates a positive feedback: correlation between input and output increases the connection weight, which in turn increases the correlation in firing, etc. This would pose a problem for one-shot reversal learning, because by the time the new patterns are shown, the existing associations would be too strong to be erased in a single timestep. However, with negative plasticity coefficients, the opposite is true: the large Hebbian trace created on initial imprinting becomes self-decreasing due to a *negative* feedback loop. Thus, when the second association is shown, the existing Hebbian traces are now small enough to be completely erased (and indeed reversed) in a single presentation, which would not be the case under positive plasticity and increasing Hebbian traces.

In short, the BOHP method has not only determined which connections must be plastic to learn an association; it can also develop a precise fine-tuning of the temporal dynamics of this plasticity, by modulating the sign of plasticity coefficients. This remarkable result confirms the potential of BOHP to deal with temporal dynamics in environmental learning.

References

[1] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: (Oct. 2014). arXiv: 1410.5401 [cs.NE].

[2] Adam Santoro et al. "One-shot Learning with Memory-Augmented Neural Networks". In: (2016). arXiv: 1605.06065 [cs.LG].

[3] Sainbayar Sukhbaatar et al. "End-To-End Memory Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C Cortes et al. Curran Associates, Inc., 2015, pp. 2440–2448.

[4] Zlatko Vasilkoski et al. "Review of stability properties of neural plasticity rules for implementation on memristive neuromorphic hardware". In: *The 2011 International Joint Conference on Neural Networks (IJCNN)*. 2011, pp. 2563–2569.