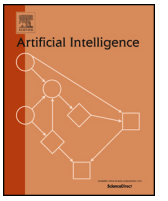
Artificial Intelligence 346 (2025) 104360

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



A theory of synaptic neural balance: From local to global order ☆

Pierre Baldi *, Antonios Alexos, Ian Domingo, Alireza Rahmansetayesh

Department of Computer Science, University of California, Irvine, United States of America

ARTICLE INFO

Keywords:
Neural networks
Deep learning
Activation functions
Regularization
Scaling
Neural balance

ABSTRACT

We develop a general theory of synaptic neural balance and how it can emerge or be enforced in neural networks. For a given additive cost function R (regularizer), a neuron is said to be in balance if the total cost of its input weights is equal to the total cost of its output weights. The basic example is provided by feedforward networks of ReLU units trained with L_2 regularizers, which exhibit balance after proper training. The theory explains this phenomenon and extends it in several directions. The first direction is the extension to bilinear and other activation functions. The second direction is the extension to more general regularizers, including all L_p $(p > 0)$ regularizers. The third direction is the extension to non-layered architectures, recurrent architectures, convolutional architectures, as well as architectures with mixed activation functions and to different balancing algorithms. Gradient descent on the error function alone does not converge in general to a balanced state, where every neuron is in balance, even when starting from a balanced state. However, gradient descent on the regularized error function ought to converge to a balanced state, and thus network balance can be used to assess learning progress. The theory is based on two local neuronal operations: scaling which is commutative, and balancing which is not commutative. Finally, and most importantly, given any set of weights, when local balancing operations are applied to each neuron in a stochastic manner, global order always emerges through the convergence of the stochastic balancing algorithm to the same unique set of balanced weights. The reason for this convergence is the existence of an underlying strictly convex optimization problem where the relevant variables are constrained to a linear, only architecture-dependent, manifold. Simulations show that balancing neurons prior to learning, or during learning in alternation with gradient descent steps, can improve learning speed and performance thereby expanding the arsenal of available training tools. Scaling and balancing operations are entirely local and thus physically plausible in biological and neuromorphic neural networks.

Contents

☆ Work in part supported by ARO grant 76649-CS to P.B.

* Corresponding author.
E-mail address: pfbaldi@uci.edu (P. Baldi).

https://doi.org/10.1016/j.artint.2025.104360
Received 31 October 2024; Received in revised form 6 May 2025; Accepted 12 May 2025

Available online 16 May 2025

1. Introduction

One of the most common complaints against neural networks is that they provide at best "black-box" solutions to problems. In particular, when large neural networks are trained on complex tasks, they produce large arrays of synaptic weights that have no clear structure and are difficult to interpret. Thus finding any kind of structure in the weights of large neural networks is of great interest. Here we study a particular kind of structure we call synaptic neural balance and the conditions under which it emerges. Neural balance in general refers to some kind of homeostatic process that facilitates information processing in neural systems. Synaptic neural balance is different from other forms of neural balance which are based on the connectivity patterns or the activities of neurons, such as the biological notion of balance between excitation and inhibition [15,13,16,20,30]. In contrast, we use the term synaptic neural balance to refer to any systematic relationship between the input and output synaptic weights of individual neurons, or layers of neurons. Here we consider the case where the cost of the input weights is equal to the cost of the output weights, where the cost is defined by some cost function or regularizer. One of the most basic examples of such a relationship is when the sum of the squares of the input weights of a neuron is equal to the sum of the squares of its output weights.

Basic Example: The basic example where this happens is with a neuron with a ReLU activation function inside a network trained to minimize an error function with L_2 regularization. If we multiply the incoming weights of the neuron by some $\lambda > 0$ and divide the outgoing weights of the neuron by the same λ, it is easy to see that this double scaling operation does not affect in any way the contribution of the neuron to the rest of the network. Thus, any component of the error function that depends only on the input-output function of the network remains unchanged. However, the value of the L_2 regularizer changes with λ and we can ask what is the value of λ that minimizes the corresponding contribution given by:

$$\sum_{i \in IN} (\lambda w_i)^2 + \sum_{i \in OUT} (w_i/\lambda)^2 = \lambda^2 A + \frac{1}{\lambda^2} B \tag{1.1}$$

where IN and OUT denote the set of incoming and outgoing weights respectively, $A = \sum_{i \in IN} w_i^2$, and $B = \sum_{i \in OUT} w_i^2$. The product of the two terms on the right-hand side of Equation (1.1) is equal to AB and does not depend on λ. Since of all the rectangles with the same area the square has the shortest perimeter, the minimum is achieved when these two terms are equal, which yields: $(\lambda^*)^4 = B/A$ for the optimal λ^*. The corresponding new set of weights, $v_i = \lambda^* w_i$ for the input weights and $v_i = w_i/\lambda^*$ for the outgoing weights, must be balanced: $\sum_{i \in IN} v_i^2 = \sum_{i \in OUT} v_i^2$. This is because the optimal scaling factor for these rescaled weights must be $\lambda^* = 1$. Furthermore, if an entire network of ReLU neurons is properly trained using a standard error function with an L_2 regularizer, at the end of training one observes a remarkable phenomenon: for each ReLU neuron, the norm of the incoming synaptic weights is approximately equal to the norm of the outgoing synaptic weights, i.e. every neuron is balanced. Our goal here is to gain a deeper understanding of this basic example and of how general the phenomenon of synaptic balance is.

There have been isolated previous studies of this kind of balance [12,32] from different perspectives or under special conditions, restricted to particular neuronal models, architectures, or cost functions. Many of these studies propose to favor balance by adding an extra term to the loss function forcing the difference between input and output synaptic costs to be close to zero, or the ratio between input and output synaptic costs to be close to one [38]. The work in [32] proposes a new biologically plausible learning rule to increase robustness of neural dynamics in non-linear recurrent networks resulting in a form of synaptic balance. In [12], the authors show that if a deep network is initialized in a balanced state with respect to the sum of squares metric, and if training progresses with an infinitesimal learning rate, then the balance is preserved throughout training. Related results are also described in [2]. Other studies have also explored symmetry and balance effects on training neural networks. For example, [22] shows that training with stochastic gradient descent does not work well in highly unbalanced neural networks. As a result, these authors propose a rescaling-invariant solution analyzed in [24]. In the same vein, other authors have proposed that learning in neural networks can be accelerated with rescaling transformations [39,1] without focusing on synaptic balancing. In [29] multiplicative rescaling factors, one at each hidden unit, are used to balance the weights with a proof of convergence, but not of uniqueness.

Here we take a different, more unified, approach aimed at uncovering the generality of synaptic neuronal balance phenomena, the learning conditions under which they occur, as well as new local balancing algorithms and their convergence properties. We explain

and study synaptic neural balance in its generality in terms of activation functions, regularizers, network architectures, and training stages. In particular, we systematically answer questions such as: Why does balance occur? Does it occur only with ReLU neurons? Does it occur only with L_2 regularizers? Does it occur only in certain architectures, e.g., fully connected feedforward architectures, as opposed to locally connected, convolutional, or recurrent architectures? Does it occur only at the end of training? In the process of answering these questions, we introduce local scaling and balancing operations for individual neurons or entire neural layers. Furthermore, we show that when these local operations are applied stochastically, a global balanced state always emerges, and this state is *unique* and depends only on the initial weights, but not on the order in which the neurons are balanced. In one rare case, the state can be solved analytically (Appendix B). While this is primarily a theoretical paper, we also provide experimental results to corroborate the theory and to show that balancing neurons, before or during training, can improve learning convergence and performance in both feedforward and recurrent networks. We also discuss the locality of balancing and its plausibility in biological or neuromorphic neural networks.

The rest of the article is organized as follows. In Section 2, we study classes of activation functions that satisfy certain properties, in particular the class of homogeneous activation functions, which is equivalent to the BiLU (bi-linear units) class of activation functions. In Sections 3 and 4, we define the scaling and balancing operations respectively, and some of their variations and properties. In Section 5, we introduce general classes of regularizers, or cost functions, and derive corresponding general balancing equations. In Section 6, we generalize the balance theory beyond homogeneous neurons to encompass, for instance, the BiPU (bi-power units) class of activation functions. In Section 7, we show that learning by gradient descent on a regularized error function ought to converge to balanced weights. In Section 8, we examine different network balancing algorithms, in particular using deterministic or stochastic schedules. In Section 9, we prove the main convergence theorem and derive the underlying geometric structure. In Section 10, we report the results of simulations to corroborate the theory, in particular the convergence of balancing to a unique optimum balanced state. In Section 11, we report the results of simulations showing how balancing can be applied to improve or accelerate learning. Biological and neuromorphic and other considerations are discussed in Section 12 before the Conclusion. In Appendix A, we provide a proof of universal approximation properties for BiLU neurons. In Appendix B, we provide an analytic solution for the unique global balanced optimum in the special case of a chain of linear neurons or, more generally, for a layered linear feed-forward network with tied balancing. It may be easier for the reader to review all the proofs first and return to them only after the general picture has been acquired.

2. Homogeneous and BiLU activation functions

In this section, we generalize the basic example of the introduction from the standpoint of the activation functions. In particular, we consider homogeneous activation functions (defined below). The importance of homogeneity has been previously identified in somewhat different contexts [23]. Intuitively, homogeneity is a form of linearity with respect to weight scaling and thus it is useful to motivate the concept of homogeneous activation functions by looking at other notions of linearity for activation functions. This will also be useful for Section 6 where even more general classes of activation functions are considered.

2.1. Additive activation functions

Definition 2.1. A neuronal activation function $f : \mathbb{R} \to \mathbb{R}$ is additively linear if and only if $f(x + y) = f(x) + f(y)$ for any real numbers x and y.

Proposition 2.2. *The class of additively linear and continuous activation functions is exactly equal to the class of linear activation functions, i.e., activation functions of the form $f(x) = ax$ ($a \in \mathbb{R}$).*

Proof. Linear activation functions are additively linear. Conversely, if f is additively linear, the following three properties are true:
(1) One must have: $f(nx) = nf(x)$ and $f(x/n) = f(x)/n$ for any $x \in \mathbb{R}$ and any $n \in \mathbb{N}$. As a result, $f(n/m) = nf(1)/m$ for any integers n and m ($m \neq 0$).
(2) Furthermore, $f(0 + 0) = f(0) + f(0)$ which implies: $f(0) = 0$.
(3) And thus $f(x - x) = f(x) + f(-x) = 0$, which in turn implies that $f(-x) = -f(x)$.
From these properties, it is easy to see that $f(x) = xf(1)$ for every x and thus f is linear. □

2.2. Multiplicative activation functions

Definition 2.3. A neuronal activation function $f : \mathbb{R} \to \mathbb{R}$ is multiplicative if and only if $f(xy) = f(x)f(y)$ for any real numbers x and y.

Proposition 2.4. *The class of continuous multiplicative activation functions is exactly equal to the class of functions comprising the functions: $f(x) = 0$ for every x, $f(x) = 1$ for every x, and all the even and odd functions satisfying $f(x) = x^c$ for $x \geq 0$, where c is any constant in $\mathbb{R}$.*

Proof. It is easy to check the functions described in the proposition are multiplicative. Conversely, assume f is multiplicative. For both $x = 0$ and $x = 1$, we must have $f(x) = f(xx) = f(x)f(x)$ and thus $f(0)$ is either 0 or 1, and similarly for $f(1)$. If $f(1) = 0$,

then for any x we must have $f(x) = 0$ because: $f(x) = f(1x) = f(1)f(x) = 0$. Likewise, if $f(0) = 1$, then for any x we must have $f(x) = 1$ because: $1 = f(0) = f(0x) = f(0)f(x) = f(x)$. Thus, in the rest of the proof, we can assume that $f(0) = 0$ and $f(1) = 1$. By induction, it is easy to see that for any $x \geq 0$ we must have: $f(x^n) = f(x)^n$ and $f(x^{1/n}) = (f(x))^{1/n}$ for any integer (positive or negative). As a result, for any $x \in \mathbb{R}$ and any integers n and m we must have: $f(x^{n/m}) = f(x)^{n/m}$. By continuity this implies that for any $x \geq 0$ and any $r \in R$, we must have: $f(x^r) = f(x)^r$. Now there is some constant c such that: $f(e) = e^c$. And thus, for any $x > 0$, $f(x) = f(e^{\log x}) = [f(e)]^{\log x} = e^{c \log x} = x^c$. To address negative values of x, note that we must have $f[(-1)(-1 = f(1) = 1f(-1)^2$. Thus, $f(-1)$ is either equal to 1 or to -1. Since for any $x > 0$ we have $f(-x) = f(-1)f(x)$, we see that if $f(-1) = 1$ the function must be even ($f(-x) = f(x) = x^c$), and if $f(-1) = -1$ the function must be odd ($f(-x) = -f(x)$). □

We will return to multiplicative activation function in a later section.

2.3. Linearly scalable activation functions

Definition 2.5. A neuronal activation function $f : \mathbb{R} \to \mathbb{R}$ is linearly scalable if and only if $f(\lambda x) = \lambda f(x)$ for every $\lambda \in \mathbb{R}$.

Proposition 2.6. *The class of linearly scalable activation functions is exactly equal to the class of linear activation functions, i.e., activation functions of the form* $f(x) = ax$.

Proof. Linear activation functions are linearly scalable. For the converse, if f is linearly scalable we must have $f(\lambda x) = \lambda f(x) = x f(\lambda)$ for any x and any λ. By taking $\lambda = 1$, we get $f(x) = f(1)x$ and thus f is linear. □

Thus the concepts of linearly additive or linearly scalable activation function are of limited interest since both of them are equivalent to the concept of linear activation function. A more interesting class is obtained if we consider linearly scalable activation functions, where the scaling factor λ is constrained to be positive ($\lambda > 0$), also called homogeneous functions.

2.4. Homogeneous activation functions

Definition 2.7. (Homogeneous) A neuronal activation function $f : \mathbb{R} \to \mathbb{R}$ is homogeneous if and only if: $f(\lambda x) = \lambda f(x)$ for every $\lambda \in \mathbb{R}$ with $\lambda > 0$.

Remark 2.8. Note that if f is homogeneous, $f(\lambda 0) = \lambda f(0) = f(0)$ for any $\lambda > 0$ and thus $f(0) = 0$. Thus it makes no difference in the definition of homogeneous if we set $\lambda \geq 0$ instead of $\lambda > 0$).

Remark 2.9. Clearly, linear activation functions are homogeneous. However, there exists also homogeneous functions that are non-linear, such as ReLU or leaky ReLU activation functions.

We now provide a full characterization of the class of homogeneous activation functions.

2.5. BiLU activation functions

We first define a new class of activation functions, corresponding to bilinear units (BiLU), consisting of two half-lines meeting at the origin. This class contains all the linear functions, as well as the ReLU and leaky ReLU functions, and many other functions.

Definition 2.10. (BiLU) A neuronal activation function $f : \mathbb{R} \to \mathbb{R}$ is bilinear (BiLU) if and only if $f(x) = ax$ when $x < 0$, and $f(x) = bx$ when $x \geq 0$, for some fixed parameters a and b in $\mathbb{R}$.

These include linear units ($a = b$), ReLU units ($a = 0, b = 1$), leaky ReLU ($a = \epsilon; b = 1$) units, and symmetric linear units ($a = -b$), all of which can also be viewed as special cases of piece-wise linear units [34], with a single hinge. One advantage of ReLU and more generally BiLU neurons, which is very important during backpropagation learning, is that their derivative is very simple and can only take one of two values (a or b).

Proposition 2.11. *A neuronal activation function* $f : \mathbb{R} \to \mathbb{R}$ *is homogeneous if and only if it is a BiLU activation function.*

Proof. Every function in BiLU is clearly homogeneous. Conversely, any homogeneous function f must satisfy: (1) $f(0x) = 0f(x) = f(0) = 0$; (2) $f(x) = f(1x) = f(1)x$ for any positive x; and (3) $f(x) = f(-u) = f(-1)u = -f(-1)x$ for any negative x. Thus f is in BiLU with $a = -f(-1)$ and $b = f(1)$. □

In Appendix A, we provide a simple proof that networks of BiLU neurons, even with a single hidden layer, have universal approximation properties. In the next two sections, we introduce two fundamental neuronal operations, scaling and balancing, that can be applied to the incoming and outgoing synaptic weights of neurons with BiLU activation functions.

3. Scaling

Definition 3.1. (Scaling) For any BiLU neuron i in network and any $\lambda > 0$, we let $S_\lambda(i)$ denote the synaptic scaling operation by which the incoming connection weights of neuron i are multiplied by λ and the outgoing connection weights of neuron i are divided by λ.

Note that because of the homogeneous property the scaling operation does not change how neuron i affects the rest of the network. In particular, the input-output function of the overall network remains unchanged after scaling neuron i by any $\lambda > 0$. Note also that scaling always preserves the sign of the synaptic weights to which it is applied, and the scaling operation can never convert a non-zero synaptic weight into a zero synaptic weight, or vice versa.

As usual, the bias is treated here as an additional synaptic weight emanating from a unit clamped to the value one. Thus scaling is applied to the bias.

Proposition 3.2. *(Commutativity of Scaling) Scaling operations applied to any pair of BiLU neurons i and j in a neural network commute: $S_\lambda(i)S_\mu(j) = S_\mu(j)S_\lambda(i)$, in the sense that the resulting network weights are the same, regardless of the order in which the scaling operations are applied. Furthermore, for any BiLU neuron i: $S_\lambda(i)S_\mu(i) = S_\mu(i)S_\lambda(i) = S_{\lambda\mu}(i)$.*

As a result, any set I of BiLU neurons in a network can be scaled simultaneously or in any sequential order while leading to the same final configuration of synaptic weights. If we denote by $1, 2, \ldots, n$ the neurons in I, we can for instance write: $\prod_{i\in I} S_{\lambda_i}(i) = \prod_{\sigma(i)\in I} S_{\lambda_{\sigma(i)}}(\sigma(i))$ for any permutation σ of the neurons. Likewise, we can collapse operations applied to the same neuron. For instance, we can write: $S_5(1)S_2(2)S_3(1)S_4(2) = S_{15}(1)S_8(2) = S_8(2)S_{15}(1)$

Definition 3.3. (Coordinated Scaling) For any set I of BiLU neurons in a network and any $\lambda > 0$, we let $S_\lambda(I)$ denote the synaptic scaling operation by which all the neurons in I are scaled by the same λ.

4. Balancing

Definition 4.1. (Balancing) Given a BiLU neuron in a network, the balancing operation $B(i)$ is a particular scaling operation $B(i) = S_{\lambda^*}(i)$, where the scaling factor λ^* is chosen to optimize a particular cost function, or regularizer, asociated with the incoming and outgoing weights of neuron i.

For now, we can imagine that this cost function is the usual L_2 (least squares) regularizer, but in the next section, we will consider more general classes of regularizers and study the corresponding optimization process. For the L_2 regularizer, as shown in the next section, this optimization process results in a unique value of λ^* such that sum of the squares of the incoming weights is equal to the sum of the squares of the outgoing weights, hence the term "balance". Note that $B(B(i)) = B(i)$ and that, as a special case of scaling operation, the balancing operation does not change how neuron i contributes to the rest of the network, and thus it leaves the overall input-output function of the network unchanged.

Unlike scaling operations, balancing operations in general do not commute as balancing operations (they still commute as scaling operations). Thus, in general, $B(i)B(j) \neq B(j)B(i)$. This is because if neuron i is connected to neuron j, balancing i will change the connection between i and j, and, in turn, this will change the value of the optimal scaling constant for neuron j and vice versa. However, if there are no non-zero connections between neuron i and neuron j then the balancing operations commute since each balancing operation will modify a different, non-overlapping, set of weights.

Definition 4.2. (Disjoint neurons) Two neurons i and j in a neural network are said to be disjoint if there are no non-zero connections between i and j.

Thus in this case $B(i)B(j) = S_{\lambda^*}(i)S_{\mu^*}(j) = S_{\mu^*}(j)S_{\lambda^*}(i) = B(j)B(i)$. This can be extended to disjoint sets of neurons.

Definition 4.3. (Disjoint Set of Neurons) A set I of neurons is said to be disjoint if for any pair i and j of neurons in I there are no non-zero connections between i and j.

For example, in a layered feedforward network, all the neurons in a layer form a disjoint set, as long as there are no intra-layer connections or, more precisely, no non-zero intra-layer connections. All the neurons in a disjoint set can be balanced in any order resulting in the same final set of synaptic weights. Thus we have:

Proposition 4.4. *If we index by $1, 2, \ldots, n$ the neurons in a disjoint set I of BiLU neurons in a network, we have: $\prod_{i\in I} B(i) = \prod_{i\in I} S_{\lambda_i^*}(i) = \prod_{\sigma(i)\in I} S_{\lambda^*_{\sigma(i)}}(\sigma(i)) = \prod_{\sigma(i)\in I} B(\sigma(i))$ for any permutation σ of the neurons.*

Finally, we can define the coordinated balancing of any set I of BiLU neurons (disjoint or not disjoint).

Definition 4.5. (Coordinated Balancing) Given any set I of BiLU neurons (disjoint or not disjoint) in a network, the coordinated balacing of these neurons, written as $B_{\lambda^*}(I)$, corresponds to coordinated scaling all the neurons in I by the same factor λ^*, Where λ^* minimizes the cost functions of all the weights, incoming and outgoing, associated with all the neurons in I.

Remark 4.6. While balancing corresponds to a full optimization of the scaling operation, it is also possible to carry a partial optimization of the scaling operation by choosing a scaling factor that reduces the corresponding contribution to the regularizer without minimizing it.

5. General framework and single neuron balance

In this section, we generalize the kinds of regularizer to which the notion of neuronal synaptic balance can be applied, beyond the usual L_2 regularizer and derive the corresponding balance equations. Thus we consider a network (feedforward or recurrent) where the hidden units are BiLU units. The visible units can be partitioned into input units and output units. For any hidden unit i, if we multiply all its incoming weights $IN(i)$ by some $\lambda > 0$ and all its outgoing weights $OUT(i)$ by $1/\lambda$ the overall function computed by the network remains unchanged due to the BiLU homogeneity property. In particular, if there is an error function that depends uniquely on the input-output function being computed, this error remains unchanged by the introduction of the multiplier λ. However, if there is also a regularizer R for the weights, its value is affected by λ and one can ask what is the optimal value of λ with respect to the regularizer, and what are the properties of the resulting weights. This approach can be applied to any regularizer. For most practical purposes, we can assume that the regularizer is continuous in the weights (hence in λ) and lower-bounded. Without any loss of generality, we can assume that it is lower-bounded by zero. If we want the minimum value to be achieved by some $\lambda > 0$, we need to add some mild condition that prevents the minimal value to be approached as $\lambda \to 0^)$, or as $\lambda \to +\infty$. For instance, it is enough if there is an interval $[a, b]$ with $0 < a < b$ where R achieves a minimal value R_{min} and $R \geq R_{min}$ in the intervals $(0, a]$ and $[b, +\infty)$. Additional (mild) conditions must be imposed if one wants the optimal value of λ to be unique, or computable in closed form (see Theorems below). Finally, we want to be able to apply the balancing approach

Thus, we consider overall regularized error functions, where the regularizer is very general, as long as it has an additive form with respect to the individual weights:

$$\mathcal{E}(W) = E(W) + R(W) \quad \text{with} \quad R(W) = \sum_w g_w(w) \tag{5.1}$$

where W denotes all the weights in the network and $E(W)$ is typically the negative log-likelihood (LMS error in regression tasks, or cross-entropy error in classification tasks). We assume that the g_w are continuous, and lower-bounded by 0. To ensure the existence and uniqueness of minimum during the balancing of any neuron, We will assume that each function g_w depends only on the magnitude $|w|$ of the corresponding weight, and that g_w is monotonically increasing from 0 to $+\infty$ ($g_w(0) = 0$ and $\lim_{x\to+\infty} g_w(x) = +\infty$). Clearly, L_2, L_1 and more generally all L_p regularizers are special cases where, for $p > 0$, L^p regularization is defined by: $R(W) = \sum_w |w|^p$.

When indicated, we may require also that the functions g_w be continuously differentiable, except perhaps at the origin in order to be able to differentiate the regularizer with respect to the λ's and derive closed form conditions for the corresponding optima. This is satisfied by all forms of L_p regularization, for $p > 0$.

Remark 5.1. Often one introduces scalar multiplicative hyperparameters to balance the effect of E and R, for instance in the form: $\mathcal{E} = E + \beta R$. These cases are included in the framework above: multipliers like β can easily be absorbed into the functions g_w above.

Theorem 5.2. *(General Balance Equation). Consider a neural network with BiLU activation functions in all the hidden units and overall error function of the form:*

$$\mathcal{E} = E(W) + R(W) \quad \text{with} \quad R(W) = \sum_w g_w(w) \tag{5.2}$$

where each function $g_w(w)$ is continuous, depends on the magnitude $|w|$ alone, and grows strictly monotonically from $g_w(0) = 0$ to $g_w(+\infty) = +\infty$. For any setting of the weights W and any hidden unit i in the network and any $\lambda > 0$ we can multiply the incoming weights of i by λ and the outgoing weights of i by $1/\lambda$ without changing the overall error E. Furthermore, there exists a unique value λ^* where the corresponding weights v ($v = \lambda^* w$ for incoming weights, $v = w/\lambda^*$ for the outgoing weights) achieve the balance equation:

$$\sum_{v \in IN(i)} g_w(v) = \sum_{v \in OUT(i)} g_w(v) \tag{5.3}$$

Proof. Under the assumptions of the theorem, E is unchanged under the rescaling of the incoming and outgoing weights of unit i due to the homogeneity property of BiLUs. Without any loss of generality, we assume that at the beginning: $\sum_{w\in IN(i)} g_w(w) < \sum_{w\in OUT(i)} g_w(w)$. As we increase λ from 1 to $+\infty$, by assumptions on functions g_w, the term $\sum_{w\in IN(i)} g_w(\lambda w)$ increases continuously from its initial value to $+\infty$, whereas the term $\sum_{w\in OUT(i)} g_w(w/\lambda)$ decreases continuously from its initial value to 0. Thus, there is a unique value λ^* where the balance is realized. If at the beginning $\sum_{w\in IN(i)} g_w(w) > \sum_{w\in OUT(i)} g_w(w)$, then the same argument is applied by decreasing λ from 1 to 0. □

Remark 5.3. For simplicity, here and in other sections, we state the results in terms of a network of BiLU units. However, the same principles can be applied to networks where only a subset of neurons are in the BiLU class, simply by applying scaling and balancing operations to only those neurons. Furthermore, not all BiLU neurons need to have the same BiLU activation functions. For instance, the results still hold for a mixed network containing both ReLU and linear units.

Remark 5.4. In the setting of Theorem 5.2, the balance equations do not necessarily minimize the corresponding regularization term. This is addressed in the next theorem.

Remark 5.5. Finally, zero weights ($w = 0$) can be ignored entirely as they play no role in scaling or balancing. Furthermore, if all the incoming or outgoing weights of a hidden unit were to be zero, it could be removed entirely from the network

Theorem 5.6. *(Balance and Regularizer Minimization) We now consider the same setting as in Theorem 5.2, but in addition we assume that the functions g_w are continuously differentiable, except perhaps at the origin. Then, for any neuron, there exists at least one optimal value λ^* that minimizes $R(W)$. Any optimal value must be a solution of the consistency equation:*

$$\lambda^2 \sum_{w \in IN(i)} w g'_w(\lambda w) = \sum_{w \in OUT(i)} w g'_w(w/\lambda) \tag{5.4}$$

Once the weights are rebalanced accordingly, the new weights must satisfy the generalized balance equation:

$$\sum_{w \in IN(i)} w g'(w) = \sum_{w \in OUT(i)} w g'(w) \tag{5.5}$$

In particular, if $g_w(w) = |w|^p$ for all the incoming and outgoing weights of neuron i, then the optimal value λ^* is unique and equal to:

$$\lambda^* = \Big(\frac{\sum_{w \in OUT(i)} |w|^p}{\sum_{w \in IN(i)} |w|^p} \Big)^{1/2p} = \Big(\frac{||OUT(i)||_p}{||IN(i)||_p} \Big)^{1/2} \tag{5.6}$$

The decrease $\Delta R \geq 0$ in the value of the L_p regularizer $R = \sum_w |w|^p$ is given by:

$$\Delta R = \left(\Big(\sum_{w \in IN(i)} |w|^p \Big)^{1/2} - \Big(\sum_{w \in OUT(i)} |w|^p \Big)^{1/2} \right)^2 \tag{5.7}$$

After balancing neuron i, its new weights satisfy the generalized L_p balance equation:

$$\sum_{w \in IN(i)} |w|^p = \sum_{w \in OUT(i)} |w|^p \tag{5.8}$$

Proof. Due to the additivity of the regularizer, the only component of the regularizer that depends on λ has the form:

$$R(\lambda) = \sum_{w \in IN(i)} g_w(\lambda w) + \sum_{w \in OUT(i)} g_w(w/\lambda) \tag{5.9}$$

Because of the properties of the functions g_w, R_λ is continuously differentiable and strictly bounded below by 0. So it must have a minimum, as a function of λ where its derivative is zero. Its derivative with respect to λ has the form:

$$R'(\lambda) = \sum_{w \in IN(i)} w g'_w(\lambda w) + \sum_{w \in OUT(i)} (-w/\lambda^2) g'_w(w/\lambda) \tag{5.10}$$

Setting the derivative to zero, gives:

$$\lambda^2 \sum_{w \in IN(i)} w g'_w(\lambda w) = \sum_{w \in OUT(i)} w g'_w(w/\lambda) \tag{5.11}$$

Assuming that the left-hand side is non-zero, which is generally the case, the optimal value for λ must satisfy:

$$\lambda = \Big(\frac{\sum_{w \in OUT(i)} w g'_w(w/\lambda)}{\sum_{w \in IN(i)} w g'_w(\lambda w)} \Big)^{1/2} \tag{5.12}$$

If the regularizing function is the same for all the incoming and outgoing weights ($g_w = g$), then the optimal value λ must satisfy:

$$\lambda = \Big(\frac{\sum_{w \in OUT(i)} w g'(w/\lambda)}{\sum_{w \in IN(i)} w g'(\lambda w)} \Big)^{1/2} \tag{5.13}$$

In particular, if $g(w) = |w|^p$ then $g(w)$ is differentiable except possibly at 0 and $g'(w) = s(w) p |w|^{p-1}$, where $s(w)$ denotes the sign of the weight w. Substituting in Equation (5.13), the optimal rescaling λ must satisfy:

$$\lambda^* = \Big(\frac{\sum_{w\in OUT(i)} ws(w)|w|^{p-1}}{\sum_{w\in IN(i)} w|ws(w)|^{p-1}}\Big)^{1/2p} = \Big(\frac{\sum_{w\in OUT(i)}|w|^p}{\sum_{w\in IN(i)}|w|^p}\Big)^{1/2p} = \Big(\frac{||OUT(i)||_p}{||IN(i)||_p}\Big)^{1/2} \tag{5.14}$$

At the optimum, no further balancing is possible, and thus $\lambda^* = 1$. Equation (5.11) yields immediately the generalized balance equation to be satisfied at the optimum:

$$\sum_{w\in IN(i)} wg'(w) = \sum_{w\in OUT(i)} wg'(w) \tag{5.15}$$

In the case of L_P regularization, it is easy to check by applying Equation (5.15), or by direct calculation that:

$$\sum_{w\in IN(i)} |\lambda^* w|^p = \sum_{w\in OUT(i)} |w/\lambda^*|^p \tag{5.16}$$

which is the generalized balance equation. Thus after balancing neuron, the weights of neuron i satisfy the L_p balance (Equation (5.8)). The change in the value of the regularizer is given by:

$$\Delta R = \sum_{w\in IN(i)} |w|^p + \sum_{w\in OUT(i)} |w|^p - \sum_{w\in IN(i)} |\lambda^* w|^p - \sum_{w\in OUT(i)} |w/\lambda^*|^p \tag{5.17}$$

By substituting λ^* by its explicit value given by Equation (5.14) and collecting terms gives Equation (5.7). □

Remark 5.7. The monotonicity of the functions g_w is not needed to prove the first part of Theorem 5.6. It is only needed to prove uniqueness of λ^* in the L_p cases.

Remark 5.8. Note that the same approach applies to the case where there are multiple additive regularizers. For instance with both L^2 and L^1 regularization, in this case the function f has the form: $g_w(w) = \alpha w^2 + \beta|w|$. Generalized balance still applies. It also applies to the case where different regularizers are applied in different disconnected portions of the network.

Remark 5.9. The balancing of a single BiLU neuron has little to do with the number of connections. It applies equally to fully connected neurons, or to sparsely connected neurons.

6. Scaling and balancing beyond BiLU activation functions

So far we have generalized ReLU activation functions to BiLU activation functions in the context of scaling and balancing operations with positive scaling factors. While in the following sections we will continue to work with BiLU activation functions, in this section we show that the scaling and balancing operations can be extended even further to other activation functions. The section can be skipped if one prefers to progress towards the main results on stochastic balancing.

Given a neuron with activation function $f(x)$, during scaling instead of multiplying and dividing by $\lambda > 0$, we could multiply the incoming weights by a function $g(\lambda)$ and divide the outgoing weights by a function $h(\lambda)$, as long as the activation function f satisfies:

$$f(g(\lambda)x) = h(\lambda)f(x) \tag{6.1}$$

for every $x \in \mathbb{R}$ to ensure that the contribution of the neuron to the rest of the network remains unchanged. Note that if the activation function f satisfies Equation (6.1), so does the activation function $-f$. In Equation (6.1), λ does not have to be positive–we will simply assume that λ belongs to some open (potentially infinite) interval (a, b). Furthermore, the functions g and h cannot be zero for $\lambda \in (a, b)$ since they are used for scaling. It is reasonable to assume that the functions g and h are continuous, and thus they must have a constant sign as λ varies over (a, b).

Now, taking $x = 0$ gives $f(0) = h(\lambda)f(0)$ for every $\lambda \in (a, b)$, and thus either $f(0) = 0$ or $h(\lambda) = 1$ for every $\lambda \in (a, b)$. The latter is not interesting and thus we can assume that the activation function f satisfies $f(0) = 0$. Taking $x = 1$ gives $f(g(\lambda)) = h(\lambda)f(1)$ for every λ in (a, b). For simplicity, let us assume that $f(x) = 1$. Then, we have: $f(g(\lambda)) = h(\lambda)$ for every λ. Substituting in Equation (6.1) yields:

$$f(g(\lambda)x) = f(g(\lambda))f(x) \tag{6.2}$$

for every $x \in \mathbb{R}$ and every $\lambda \in (a, b)$. This relation is essentially the same as the relation that defines multiplicative activation functions over the corresponding domain (see Proposition 2.4), and thus we can identify a key family of solutions using power functions. Note that we can define a new parameter $\mu = g(\lambda)$, where μ ranges also over some positive or negative interval I over which: $f(\mu x) = f(\mu)f(x)$.

6.1. Bi-power units (BiPU)

Let us assume that $\lambda > 0$, $g(\lambda) = \lambda$ and $h(\lambda) = \lambda^c$ for some $c \in \mathbb{R}$. Then the activation function must satisfy the equation:

$$f(\lambda x) = \lambda^c f(x) \tag{6.3}$$

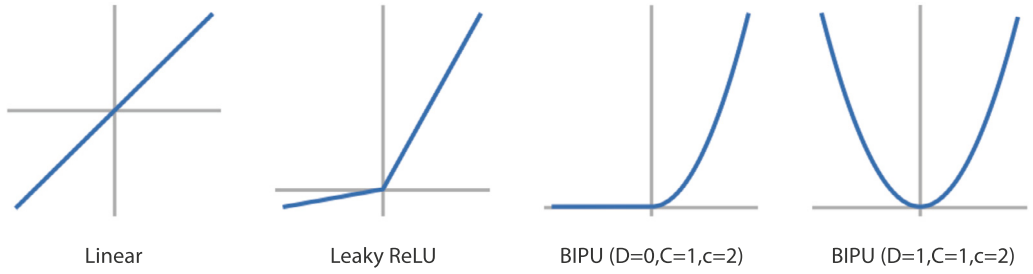


Fig. 1. BiPU activation functions (Bi-Power-Units) as described in Equation (6.3).

for any $x \in \mathbb{R}$ and any $\lambda > 0$. Note that if $f(x) = x^c$ we get a multiplicative activation function. More generally, these functions are characterized by the following proposition.

Proposition 6.1. *The set of activation functions f satisfying $f(\lambda x) = \lambda^c f(x)$ for any $x \in \mathbb{R}$ and any $\lambda > 0$ consist of the functions of the form:*

$$f(x) = \begin{cases} Cx^c & \text{if} \quad x \geq 0 \\ Dx^c & \text{if} \quad x < 0 \end{cases} \tag{6.4}$$

where $c \in \mathbb{R}$, $C = f(1) \in R$, and $D = f(-1) \in \mathbb{R}$. We call these bi-power units (BiPU). If, in addition, we want f to be continuous at 0, we must have either $c > 0$, or $c = 0$ with $C = D$.

Given the general shape, these activation functions can be called BiPU (Bi-Power-Units). Note that in the general case where $c > 0$, C and D do not need to be equal. In particular, one of them can be equal to zero, and the other one can be different from zero giving rise to "rectified power units" (Fig. 1).

Proof. By taking $x = 1$, we get $f(\lambda) = f(1)\lambda^c$ for any $\lambda > 0$. Let $f(1) = C$. Then we see that for any $x > 0$ we must have: $f(x) = Cx^c$. In addition, for every $\lambda > 0$ we must have: $f(\lambda 0) = f(0) = \lambda^c f(0)$. So if $c = 0$, then $f(x) = C = f(1)$ for $x \geq 0$. If $c \neq 0$, then $f(0) = 0$. In this case, if we want the activation function to be continuous, then we see that we must have $c \geq 0$. So in summary for $x > 0$ we must have $f(x) = f(1)x^c = Cx^c$. For the function to be right continuous at 0, we must have either $f(0) = f(1) = C$ with $c = 0$ or $f(0) = 0$ with $c > 0$. We can now look at negative values of x. By the same reasoning, we have $f(\lambda(-1)) = f(-\lambda) = \lambda^c f(-1)$ for any $\lambda > 0$. Thus for any $x < 0$ we must have: $f(x) = f(-1)|x|^c = D|x|^c$ where $D = f(-1)$. Thus, if f is continuous, there are two possibilities. If $c = 0$, then we must have $C = f(1) = D(f-1)-$ and thus $f(x) = C$ everywhere. If $c \neq 0$, then continuity requires that $c > 0$. In this case $f(x) = Cx^c$ for $x \geq 0$ with $C = f(1)$, and $f(x) = Dx^c$ for $x < 0$ with $f(-1) = D$. In all cases, it is easy to check directly that the resulting functions satisfy the functional equation given by Equation (6.3). □

6.2. Scaling BiPU neurons

A BiPU neuron can be scaled by multiplying its incoming weight by $\lambda > 0$ and dividing its outgoing weights by $1/\lambda^c$. This will not change the role of the corresponding unit in the network, and thus it will not change the input-output function of the network.

6.3. Balancing BiPU neurons

As in the case of BiLU neurons, we balance a multiplicative neuron by asking what is the optimal scaling factor λ that optimizes a particular regularizer. For simplicity, here we assume that the regularizer is in the L_p class. Then we are interested in the value of $\lambda > 0$ that minimizes the function:

$$\lambda^p \sum_{w \in IN} |w|^p + \frac{1}{\lambda^{pc}} \sum_{w \in OUT} |w|^p \tag{6.5}$$

A simple calculation shows that the optimal value of λ is given by:

$$\lambda^* = \left(\frac{c \sum_{OUT} |w|^p}{\sum_{IN} |w|^p} \right)^{1/p(c+1)} \tag{6.6}$$

Thus after balancing the weights, the neuron must satisfy the balance equation:

$$c \sum_{OUT} |w|^p = \sum_{IN} |w|^p \tag{6.7}$$

in the new weights w.

So far, we have focused on balancing individual neurons. In the next two sections, we look at balancing across all the units of a network. We first look at what happens to network balance when a network is trained by gradient descent and then at what happens to network balance when individual neurons are balanced iteratively in a regular or stochastic manner.

7. Network balance: gradient descent

A natural question is whether gradient descent (or stochastic gradient descent) applied to a network of BiLU neurons, with or without a regularizer, converges to a balanced state of the network, where all the BiLU neurons are balanced. So we first consider the case where there is no regularizer ($\mathcal{E} = E$). The results in [12] may suggest that gradient descent may converge to a balanced state. In particular, they write that for any neuron i:

$$\frac{d}{dt}\Big(\sum_{w\in IN(i)} w^2 - \sum_{w\in OUT(i)} w^2\Big) = 0 \tag{7.1}$$

Thus the gradient flow exactly preserves the difference between the L_2 cost of the incoming and outgoing weights or, in other words, the derivative of the L_2 balance *deficit* is zero. Thus if one were to start from a balanced state and use an infinitesimally small learning rate one ought to stay in a balanced state at all times.

However, it must be noted that this result was derived for the L_2 metric only, and thus would not cover other L_p forms of balance. Furthermore, it requires an infinitesimally small learning rate. In practice, when any standard learning rate is applied, we find that gradient descent does *not* converge to a balanced state (Fig. 1). However, things are different when a regularizer term is included in the error functions as described in the following theorem.

Theorem 7.1. *Gradient descent in a network of BiLU units with error function $\mathcal{E} = E + R$ where R has the properties described in Theorem 5.6 (including all L_p) must converge to a balanced state, where every BiLU neuron is balanced.*

Proof. By contradiction, suppose that gradient descent converges to a state that is unbalanced and where the gradient with respect to all the weights is zero. Then there is at least one unbalanced neuron in the network. We can then multiply the incoming weights of such a neuron by λ and the outgoing weights by $1/\lambda$ as in the previous section without changing the value of E. Since the neuron is not in balance, we can move λ infinitesimally so as to reduce R, and hence $\mathcal{E}$. But this contradicts the fact that the gradient is zero. □

Remark 7.2. In practice, in the case of stochastic gradient descent applied to $E + R$, at the end of learning the algorithm may hover around a balanced state. If the state reached by the stochastic gradient descent procedure is not approximately balanced, then learning ought to continue. In other words, the degree of balance could be used to monitor whether learning has converged or not. Balance is a necessary, but not sufficient, condition for being at the optimum.

Remark 7.3. If early stopping is being used to control overfitting, there is no reason for the stopping state to be balanced. However, the balancing algorithms described in the next section could be used to balance this state.

8. Network balance: stochastic or deterministic balancing algorithms

In this section, we look at balancing algorithms where, starting from an initial weight configuration W, the BiLU neurons of a network are balanced iteratively according to some deterministic or stochastic schedule that periodically visits all the neurons. We can also include algorithms where neurons are partitioned into groups (e.g. neuronal layers) and neurons in each group are balanced together.

8.1. Basic stochastic balancing

The most interesting algorithm is when the BiLU neurons of a network are iteratively balanced in a purely stochastic manner. This algorithm is particularly attractive from the standpoint of physically implemented neural networks because the balancing algorithm is local and the updates occur randomly without the need for any kind of central coordination. As we shall see in the following section, the random local operations remarkably lead to a unique form of global order. The proof for the stochastic case extends immediately to the deterministic case, where the BiLU neurons are updated in a deterministic fashion, for instance by repeatedly cycling through them according to some fixed order.

8.2. Subset balancing (independent or tied)

It is also possible to partition the BiLU neurons into non-overlapping subsets of neurons, and then balance each subset, especially when the neurons in each subset are disjoint of each other. In this case, one can balance all the neurons in a given subset, and repeat this subset-balancing operation subset-by-subset, again in a deterministic or stochastic manner. Because the BiLU neurons in each subset are disjoint, it does not matter whether the neurons in a given subset are updated synchronously or sequentially (and in which

order). Since the neurons are balanced independently of each other, this can be called independent subset balancing. For example, in a layered feedforward network with no lateral connections, each layer corresponds to a subset of disjoint neurons. The incoming and outgoing connections of each neuron are distinct from the incoming and outgoing connections of any other neuron in the layer, and thus the balancing operation of any neuron in the layer does not interfere with the balancing operation of any other neuron in the same layer. So this corresponds to independent layer balancing,

As a side note, balancing a layer h, may disrupt the balance of layer $h+1$. However, balancing layer h and $h+2$ (or any other layer further apart) can be done without interference of the balancing processes. This suggests also an alternating balancing scheme, where one alternatively balances all the odd-numbered layers, and all the evenly-numbered layers.

Yet another variation is when the neurons in a disjoint subset are tied to each other in the sense that they must all share the same scaling factor λ. In this case, balancing the subset requires finding the optimal λ for the entire subset, as opposed to finding the optimal λ for each neuron in the subset. Since the neurons are balanced in a coordinated or tied fashion, this can be called coordinated or tied subset balancing. For example, tied layer balancing must use the same λ for all the neurons in a given layer. It is easy to see that this approach leads to layer synaptic balance which has the form (for an L_p regularizer):

$$\sum_i \sum_{w \in IN(i)} |w|^p = \sum_i \sum_{w \in OUT(i)} |w|^p \tag{8.1}$$

where i runs over all the neurons in the layer. This does *not* necessarily imply that each neuron in the layer is individually balanced. Thus neuronal balance for every neuron in a layer implies layer balance, but the converse is not true. Independent layer balancing will lead to layer balance. Coordinated layer balancing will lead to layer balance, but not necessarily to neuronal balance of each neuron in the layer. Layer-wise balancing, independent or tied, can be applied to all the layers and in deterministic (e.g. sequential) or stochastic manner. Again the proof given in the next section for the basic stochastic algorithm can easily be applied to these cases (see also Appendix B).

8.3. Synaptic balance with weight sharing and convolutional neural networks

Suppose that two connections share the same weight so that we must have: $w_{ij} = w_{kl}$ at all times. In general, when the balancing algorithm is applied to neuron i or j, the weight w_{ij} changes, and the same change must be applied to w_{kl}. The latter may disrupt the balance of neuron k or l. Furthermore, if balancing of neuron i or j leads to an increase of $R(w_{ij})$, then when the weight is shared the total value of the regularizer R can conceivably increase. However, by using the convergence results in the following sections, one can see that if all the neurons in the network are periodically visited and balanced, the weights will converge to a final configuration, even when weight sharing is applied.

The case of convolutional networks (CNNs) is somewhat special, since *all* the incoming weights of the neurons that share the same convolutional kernel are shared. However, in general, the outgoing weights are not shared. Furthermore, certain operations, such as max-pooling, are not homogeneous. So, if one trains a CNN with E alone, or even with $E+R$, one should not expect any kind of balance to emerge in the convolution units. However, all the other BiLU units in the network should be balanced by the end of training by the same argument used above for gradient descent. The balancing algorithm applied to individual neurons, or the independent layer balancing algorithm, will not balance individual neurons sharing the same convolution kernel. The only balancing algorithm that could lead to some convolution layer balance, but not to individual neuronal balance, is the coordinated layer balancing, where the same λ is used for all neurons in the same convolution layer and the same filter (or channel), provided that their activation functions are BiLU functions.

Thus, when training a convolutional neural network with BiLU convolution neurons, one can choose between two main approaches for each convolutional layer and each channel. The first approach is to use a single λ shared by all neurons in the same channel (coordinated layer balancing). The second is to use a different λ for each neuron in the channel. In this case, the optimal balanced weights are calculated independently for each neuron in the channel, and then the average of each weight is retained to preserve the weight sharing properties of the channel. This is similar to how the weight changes corresponding to gradient descent with respect to E (negative log-likelihood) are computed to preserve the weight sharing properties. In addition, each of these two approaches can be carried either fully by using the corresponding optimal λs, or partially by taking an incremental step towards the optimal λs. While balancing in general is a local operation, and thus is plausible in biological or neuromorphic neural systems, balancing applied to shared weights and CNNs raises additional challenges in biological or neuromorphic neural systems [25].

9. Convergence of balancing algorithms

We now consider the basic stochastic balancing algorithm, where BiLU neurons are iteratively and stochastically balanced. It is essential to note that balancing a neuron j may break the balance of another neuron i to which j is connected. Thus convergence of iterated balancing is not obvious. There are three key questions to be addressed for the basic stochastic algorithm, as well as all the other balancing variations. First, does the value of the regularizer converges to a finite value? Second, do the weights themselves converge to fixed finite values representing a balanced state for the entire network? And third, if the weights converge, do they always converge to the same values, irrespective of the order in which the units are being balanced? In other words, given an initial state W for the network, is there a unique corresponding balanced state, with the same input-output functionalities?

9.1. Notation and key questions

For simplicity, we use a continuous time notation. After a certain time t each neuron has been balanced a certain number of times. While the balancing operations are not commutative as balancing operations, they are commutative as scaling operations. Thus we can reorder the scaling operations and group them neuron by neuron so that, for instance, neuron i has been scaled by the sequence of scaling operations:

$$S_{\lambda_1^*}(i)S_{\lambda_2^*}(i)\dots S_{\lambda_{n_{it}}^*}(i) = S_{\Lambda_i(t)}(i) \tag{9.1}$$

where n_{it} corresponds to the count of the last update of neuron i prior to time t, and:

$$\Lambda_i(t) = \prod_{1\le n\le n_{it}} \lambda_n^*(i) \tag{9.2}$$

For the input and output units, we can consider that their balancing coefficients λ^* are always equal to 1 (at all times) and therefore $\Lambda_i(t) = 1$ for any visible unit i.

Thus, we first want to know if R converges. Second, we want to know if the weights converge. This question can be split into two sub-questions: (1) Do the balancing factors $\lambda_n^*(i)$ converge to a limit as time goes to infinity. Even if the $\lambda_n^*(i)$'s converge to a limit, this does not imply that the weights of the network converge to a limit. After a time t, the weight $w_{ij}(t)$ between neuron j and neuron i has the value $w_{ij}\Lambda_i(t)/\Lambda_j(t)$, where $w_{ij} = w_{ij}(0)$ is the value of the weight at the start of the stochastic balancing algorithm. Thus: (2) Do the quantities $\Lambda_i(t)$ converge to finite values, different from 0? And third, if the weights converge to finite values different from 0, are these values unique or not, i.e. do they depend on the details of the stochastic updates or not? These questions are answered by the following main theorem.

9.2. Convergence of the basic stochastic balancing algorithm to a unique optimum

Theorem 9.1. *(Convergence of Stochastic Balancing) Consider a network of BiLU neurons with an error function $\mathcal{E}(W) = E(W) + R(W)$ where R satisfies the conditions of Theorem 5.2 including all L_p ($p > 0$). Let W denote the initial weights. When the neuronal stochastic balancing algorithm is applied throughout the network so that every neuron is visited from time to time, then $E(W)$ remains unchanged but $R(W)$ must converge to some finite value that is less or equal to the initial value, strictly less if the initial weights are not balanced. In addition, for every neuron i, $\lambda_i^*(t) \to 1$ and $\Lambda_i(t) \to \Lambda_i$ as $t \to \infty$, where Λ_i is finite and $\Lambda_i > 0$ for every i. As a result, the weights themselves must converge to a limit W' which is globally balanced, with $E(W) = E(W')$ and $R(W) \ge R(W')$, and with equality if only if W is already balanced. Finally, W' is unique as it corresponds to the solution of a strictly convex optimization problem in the variables $L_{ij} = \log(\Lambda_i/\Lambda_j)$ with linear constraints of the form $\sum_\pi L_{ij} = 0$ along any path π joining an input unit to an output unit and along any directed cycle (for recurrent networks). Stochastic balancing projects to stochastic trajectories in the linear manifold that run from the origin to the unique optimal configuration.*

Proof. Each individual balancing operation leaves $E(W)$ unchanged because the BiLU neurons are homogeneous. Furthermore, each balancing operation reduces the regularization error $R(W)$, or leaves it unchanged. Since the regularizer is lower-bounded by zero, the value of the regularizer must approach a limit as the stochastic updates are being applied.

For the second question, when neuron i is balanced at some step, we know that the regularizer R decreases by:

$$\Delta R = \left(\Big(\sum_{w\in IN(i)} |w|^p \Big)^{1/2} - \Big(\sum_{w\in OUT(i)} |w|^p \Big)^{1/2} \right)^2 \tag{9.3}$$

If the convergence were to occur in a finite number of steps, then the coefficients $\lambda_i^*(t)$ must become equal and constant to 1 and the result is obvious. So we can focus on the case where the convergence does not occur in a finite number of steps (indeed this is the main scenario, as we shall see at the end of the proof). Since $\Delta R \to 0$, we must have:

$$\sum_{w\in IN(i)} |w|^p \to \sum_{w\in OUT(i)} |w|^p \tag{9.4}$$

But from the expression for λ^* (Equation (5.14)), this implies that for every i, $\lambda_n^*(i) \to 1$ as time increases ($n \to \infty$). This alone is not sufficient to prove that $\Lambda_i(t)$ converges for every i as $t \to \infty$. However, it is easy to see that $\Lambda_i(t)$ cannot contain a sub-sequence that approaches 0 or ∞ (Fig. 2). Furthermore, not only ΔR converges to 0, but the series $\sum \Delta R$ is convergent. This shows that, for every i, $\Delta_i(t)$ must converge to a finite, non-zero value Δ_i. Therefore, all the weights must converge to fixed values given by $w_{ij}(0)\Lambda_i/\Lambda_j$.

Finally, we prove that given an initial set of weights W, the final balanced state is unique and independent of the order of the balancing operations. The coefficients Λ_i corresponding to a globally balanced state must be solutions of the following optimization problem:

$$\min_\Lambda R(\Lambda) = \sum_{ij} \Big| \frac{\Lambda_i}{\Lambda_j} w_{ij} \Big|^p \tag{9.5}$$

under the simple constraints: $\Lambda_i > 0$ for all the BiLU hidden units, and $\Lambda_i = 1$ for all the visible (input and output) units. In this form, the problem is not convex. Introducing new variables $M_j = 1/\Lambda_j$ is not sufficient to render the problem convex. Using variables

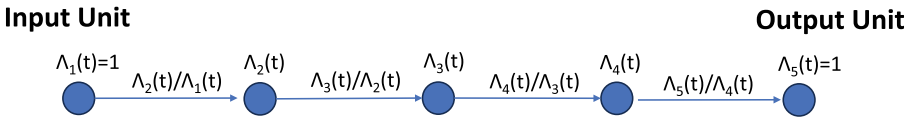


Fig. 2. A path with three hidden BiLU units connecting one input unit to one output unit. During the application of the stochastic balancing algorithm, at time t each unit i has a cumulative scaling factor $\Lambda_i(t)$, and each directed edge from unit j to unit i has a scaling factor $M_{ij}(t) = \Lambda_i(t)/\Lambda_j(t)$. The $\lambda_i(t)$ must remain within a finite closed interval away from 0 and infinity. To see this, imagine for instance that there is a subsequence of $\Lambda_3(t)$ that approaches 0. Then there must be a corresponding subsequence of $\Lambda_4(t)$ that approaches 0, or else the contribution of the weight $w_{43}\Lambda_4(t)/\Lambda_3(t)$ to the regularizer would go to infinity. But then, as we reach the output layer, the contribution of the last weight $w_{54}\Lambda_5(t)/\Lambda_4(t)$ to the regularizer goes to infinity because $\Lambda_5(t)$ is fixed to 1 and cannot compensate for the small values of $\Lambda_4(t)$. And similarly, if there is a subsequence of $\Lambda_3(t)$ going to infinity, we obtain a contradiction by propagating its effect towards the input layer.

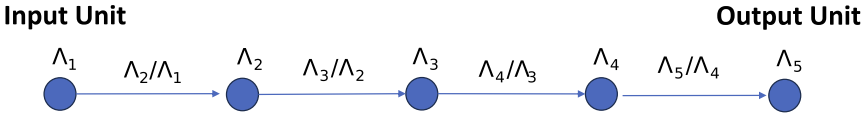


Fig. 3. A path with five units. After the stochastic balancing algorithm has converged, each unit i has a scaling factor Λ_i, and each directed edge from unit j to unit i has a scaling factor $M_{ij} = \Lambda_i/\Lambda_j$. The products of the M_{ij}'s along the path are given by: $\frac{\Lambda_2}{\Lambda_1}\frac{\Lambda_3}{\Lambda_2}\frac{\Lambda_4}{\Lambda_3}\frac{\Lambda_5}{\Lambda_4} = \frac{\Lambda_5}{\Lambda_1}$. Accordingly, if we sum the variables $L_{ij} = \log M_{ij}$ along the directed path, we get $L_{21} + L_{32} + L_{43} + L_{54} = \log\Lambda_5 - \log\Lambda_1$. In particular, if unit 1 is an input unit and unit 5 is an output unit, we must have $\Lambda_1 = \Lambda_5 = 1$ and thus: $L_{21} + L_{32} + L_{43} + L_{54} = 0$. Likewise, in the case of a directed cycle where unit 1 and unit 5 are the same, we must have: $L_{21} + L_{32} + L_{43} + L_{54} + L_{15} = 0$.

$M_{ij} = \Lambda_i/\Lambda_j$ is better, but still problematic for $0 < p \leq 1$. However, let us instead introduce the new variables $L_{ij} = \log(\Lambda_i/\Lambda_j)$. These are well defined since we know that $\Lambda_i/\Lambda_j > 0$. The objective now becomes:

$$\min R(L) = \sum_{ij} |e^{L_{ij}} w_{ij}|^p = \sum_{ij} e^{pL_{ij}} |w_{ij}|^p \tag{9.6}$$

This objective is strictly convex in the variables L_{ij}, as a sum of strictly convex functions (exponentials). However, to show that it is a convex optimization problem we need to study the constraints on the variables L_{ij}. In particular, from the set of Λ_i's it is easy to construct a unique set of L_{ij}. However what about the converse?

Definition 9.2. A set of real numbers L_{ij}, one per connection of a given neural architecture, is self-consistent if and only if there is a unique corresponding set of numbers $\Lambda_i > 0$ (one per unit) such that: $\Lambda_i = 1$ for all visible units and $L_{ij} = \log\Lambda_i/\Lambda_j$ for every directed connection from a unit j to a unit i.

Remark 9.3. This definition depends on the graph of connections, but not on the original values of the synaptic weights. Every balanced state is associated with a self-consistent set of L_{ij}, but not every self-consistent set of L_{ij} is associated with a balanced state.

Proposition 9.4. *A set L_{ij} associated with a neural architecture is self-consistent if and only if $\sum_\pi L_{ij} = 0$ where π is any directed path connecting an input unit to an output unit or any directed cycle (for recurrent networks).*

Remark 9.5. Thus the constraints associated with being a self-consistent configuration of L_{ij}' s are all linear. This resulting linear manifold $\mathcal{L}$ depends only on the architecture, i.e., the graph of connections, but not on the actual weight values. The strictly convex function $R(L_{ij})$ depends on the actual weights W. Different sets of weights W produce different convex functions over the same linear manifold. If E denotes the total number of connections, then $\dim\mathcal{L} \leq E$. In order to infer all the Λ_i, there must exist at least one constrained path going through each node i. Thus, in a layered feedforward network, the dimension of $\mathcal{L}$ is given by: $\dim\mathcal{L} = E - M$, where here M denotes the size of the largest layer.

Remark 9.6. One could coalesce all the input units and all output units into a single unit, in which case a path from an input unit to and output unit becomes also a directed cycle. In this representation, the constraints are that the sum of the L_{ij} must be zero along any directed cycle. In general, it is not necessary to write a constraint for every path from input units to output units. It is sufficient to select a representative set of paths such that every unit appears in at least one path.

Remark 9.7. All the results in this section remain true in a mixed network containing both BiLU neurons and non-BiLU neurons, as long as one uses $\Lambda_i = 1$ for any non-BiLU neuron.

Proof. If we look at any directed path π from unit i to unit j, it is easy to see that we must have:

$$\sum_\pi L_{kl} = \log\Lambda_i - \log\Lambda_j \tag{9.7}$$

This is illustrated in Figs. 3 and 4. Thus along any directed path that connects any input unit to any output unit, we must have $\sum_\pi L_{ij} = 0$. In addition, for recurrent neural networks, if π is a directed cycle we must also have: $\sum_\pi L_{ij} = 0$. Thus in short we only

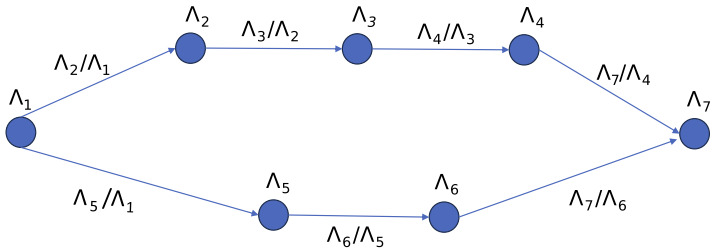


Fig. 4. Two hidden units (1 and 7) connected by two different directed paths 1-2-3-4-7 and 1-5-6-7 in a BiLU network. Each unit i has a scaling factor Λ_i, and each directed edge from unit j to unit i has a scaling factor $M_{ij} = \Lambda_i/\Lambda_j$. The products of the M_{ij}'s along each path are equal to: $\frac{\Lambda_2}{\Lambda_1}\frac{\Lambda_3}{\Lambda_2}\frac{\Lambda_4}{\Lambda_3}\frac{\Lambda_7}{\Lambda_4} = \frac{\Lambda_5}{\Lambda_1}\frac{\Lambda_6}{\Lambda_5}\frac{\Lambda_7}{\Lambda_6} = \frac{\Lambda_7}{\Lambda_1}$. Therefore the variables $L_{ij} = \log M_{ij}$ must satisfy the linear equation: $L_{21} + L_{32} + L_{43} + L_{74} = L_{51} + L_{65} + L_{76} = \log\Lambda_7 - \log\Lambda_1$.

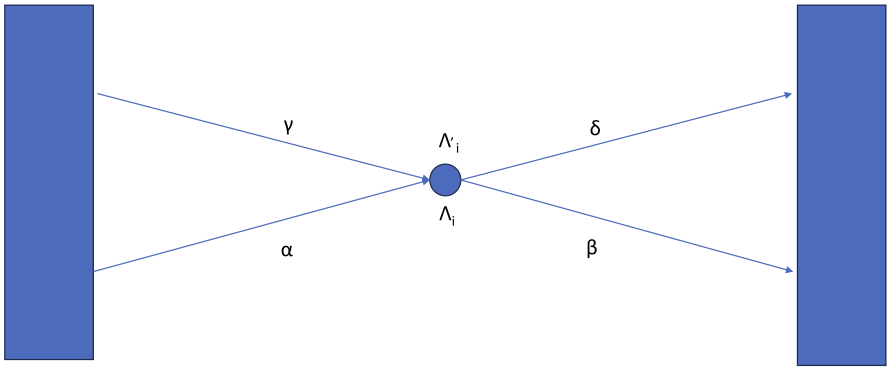


Fig. 5. Consider two paths $\alpha + \beta$ and $\gamma + \delta$ from the input layer to the output layer going through the same unit i. Let us assume that the first path assigns a multiplier Λ_i to unit i and the second path assigns a multiplier Λ'_i to the same unit. By assumption we must have: $\sum_\alpha L_{ij} + \sum_\beta L_{ij} = 0$ for the first path, and $\sum_\gamma L_{ij} + \sum_\delta L_{ij} = 0$. But $\alpha + \delta$ and $\gamma + \beta$ are also paths from the input layer to the output layer and therefore: $\sum_\alpha L_{ij} + \sum_\delta L_{ij} = 0$ and $\sum_\gamma L_{ij} + \sum_\beta L_{ij} = 0$. As a result, $\sum_\alpha L_{ij} = \log\Lambda_i = \sum_\gamma L_{ij} = \Lambda'_i$. Therefore the assignment of the multiplier Λ_i must be consistent across different paths going through unit i.

need to add linear constraints of the form: $\sum_\pi L_{ij} = 0$. Any unit is situated on a path from an input unit to an output unit. Along that path, it is easy to assign a value Λ_i to each unit by simple propagation starting from the input unit which has a multiplier equal to 1. When the propagation terminates in the output unit, it terminates consistently because the output unit has a multiplier equal to 1 and, by assumption, the sum of the multipliers along the path must be zero. So we can derive scaling values Λ_i from the variables L_{ij}. Finally, we need to show that there are no clashes, i.e. that it is not possible for two different propagation paths to assign different multiplier values to the same unit i. The reason for this is illustrated in Fig. 5. □

We can now complete the proof Theorem 9.1. Given a neural network of BiLUs with a set of weights W, we can consider the problem of minimizing the regularizer $R(L_{ij}$ over the self-admissible configuration L_{ij}. For any $p > 0$, the L_p regularizer is strictly convex and the space of self-admissible configurations is linear and hence convex. Thus this is a strictly convex optimization problem that has a unique solution (Fig. 6). Note that the minimization is carried over self-consistent configurations, which in general are not associated with balanced states. However, the configuration of the weights associated with the optimum set of L_{ij} (point A in Fig. 6) must be balanced. To see this, imagine that one of the BiLU units–unit i in the network is not balanced. Then we can balance it using a multiplier λ_i^* and replace Λ_i by $\Lambda'_i = \Lambda_i\lambda^*$. It is easy to check that the new configuration including Λ'_i is self-consistent. Thus, by balancing unit i, we are able to reach a new self-consistent configuration with a lower value of R which contradicts the fact that we are at the global minimum of the strictly convex optimization problem.

We know that the stochastic balancing algorithm always converges to a balanced state. We need to show that it cannot converge to any other balanced state, and in fact that the global optimum is the only balanced state. By contradiction, suppose it converges to a different balanced state associated with the coordinates (L_{ij}^B) (point B in Fig. 6). Because of the self-consistency, this point is also associated with a unique set of (Λ_i^B) coordinates. The cost function is continuous and differentiable in both the L_{ij}'s and the Λ_i's coordinates. If we look at the negative gradient of the regularizer, it is non-zero and therefore it must have at least one non-zero component $\partial R/\partial\Lambda_i$ along one of the Λ_i coordinates. This implies that by scaling the corresponding unit i in the network, the regularizer can be further reduced, and by balancing unit i the balancing algorithm will reach a new point (C in Fig. 6) with lower regularizer cost. This contradicts the assumption that B was associated with a balanced stated. Thus, given an initial set of weights W, the stochastic balancing algorithm must always converge to the same and unique optimal balanced state W^* associated with the self-consistent point A. A particular stochastic schedule corresponds to a random path within the linear manifold from the origin (at time zero all the multipliers are equal to 1, and therefore for any i and any j: $M_{ij} = 1$ and $L_{ij} = 0$) to the unique optimum point A. □

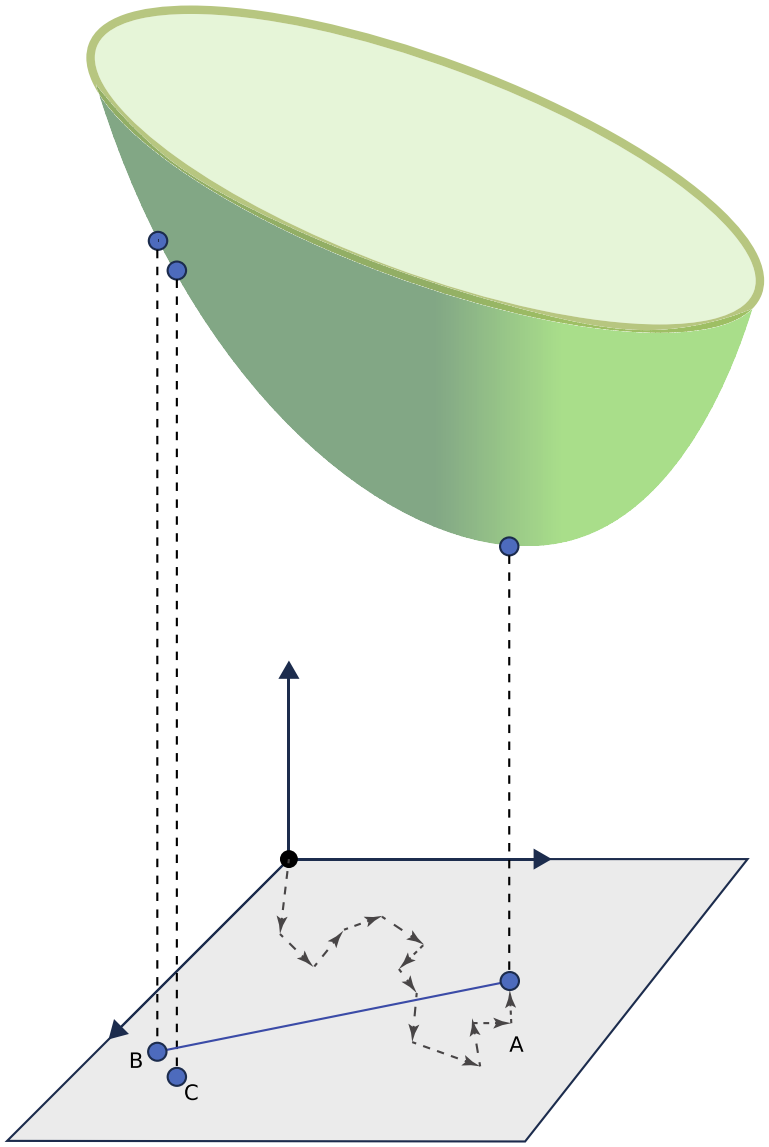


Fig. 6. The problem of minimizing the strictly convex regularizer $R(L_{ij}) = \sum_{ij} e^{pL_{ij}} |w_{ij}|^p$ $(p > 0)$, over the linear (hence convex) manifold of self-consistent configurations defined by the linear constraints of the form $\sum_\pi L_{ij} = 0$, where π runs over input-output paths. The regularizer function depends on the weights. The linear manifold depends only on the architecture, i.e., the graph of connections. This is a strictly convex optimization problem with a unique solution associated with the point A. At A the corresponding weights must be balanced, or else a self-consistent configuration of lower cost could be found by balancing any non-balanced neuron. Finally, any other self-consistent configuration B cannot correspond to a balanced state of the network, since there must exist balancing moves that further reduce the regularizer cost (see main text). Stochastic balancing produces random paths from the origin, where $L_{ij=} \log M_{ij} = 0$, to the unique optimum point A.

Remark 9.8. From the proof, it is clear that the same result holds also for any deterministic balancing schedule, as well as for tied and non-tied subset balancing, e.g., for layer-wise balancing and tied layer-wise balancing. In the Appendix, we provide an analytical solution for the case of tied layer-wise balancing in a layered feed-forward network.

Remark 9.9. Stochastic balancing of synapses can be applied to any network, even networks comprised entirely of non-homogeneous neurons. In this case, stochastic balancing will still converge to a unique stable configuration of the synaptic weights. However, the overall function implemented by the balanced network may differ from the function implemented by the network at the start of the stochastic balancing. Experiments on balancing sigmoidal neurons are reported in Section 11.

Remark 9.10. In principle, balancing (or scaling) can be applied independently of the regularization approach used. For instance, during training, one could alternate scaling and stochastic gradient descent steps, where the scaling step is performed with respect to L_1, but the gradient descent step is performed with respect to an L_2-regularized error function.

Remark 9.11. Dropout is another form of regularization [6]. Dropout and synaptic balance are not exclusive and could be used together during training in different ways. During training with dropout, full or partial balance could be applied at the beginning of training, at the end of training, or during training, by interleaving dropout and balancing steps. Another possibility in partial balancing is to randomly select the neurons to be balanced using dropout-like probabilities (dropout balancing).

9.3. Partial balancing

Partial forms of network balance can be carried out in many ways. At the level of individual neurons, as mentioned in Remark 4.6, partial balance can be obtained using a favorable, but not optimal, scaling factor. At the level of networks, partial balance can be obtained by using partial balancing (i.e. favorable, but not optimal, scaling) at the level of all, or a subset, of neurons. Partial balancing can also be obtained by applying full balancing to only a subset of neurons, or by applying full balancing to all the neurons but without iterating until convergence to the unique balanced state. In other words, partial balancing can refer to any approach that reduces the value of the regularizer $R(L_{ij})$ without reaching the unique global optimum for the network (Fig. 6). It should be clear from the proof of the main theorem that iterated partial balancing will converge to the global optimum as long as all the neurons in the network are periodically visited and the scaling factors are not taken arbitrarily close to 1 prior to convergence to the global minimum. Note that balancing can also be viewed as a form of EM algorithm where the E and M steps can be taken fully or partially. Partial balancing can be faster and is used in some of the simulations of Section 11. In these simulations, partial balancing of the network refers to a single pass of full balancing of all the neurons in the network, applied layer by layer, going from the input layer to the output layer. Partial balancing can be interleaved with gradient descent learning steps.

9.4. Convergence to a unique optimum for BiPU stochastic balancing

We have seen that a generalized form of scaling and balancing can be defined for more general units than BiLUs, in particular for BiPUs. Thus now we consider a network of units with activations functions f satisfying the relationship: $f(\lambda x) = \lambda^c f(x)$ (note that this includes BiLU units for $c = 1$). We even allow c to vary from unit to unit.

It is easy to see that most of the analyses above done for BiLU units apply to this generalization. In particular, if we apply stochastic generalized balancing, in the limit the positive multipliers of each connection w_{ij} must satisfy:

$$M_{ij} = \Lambda_i / \Lambda_j^{c_j} \tag{9.8}$$

As above, we can define a new set of variables $L_{ij} = \log M_{ij}$ and, for any $p > 0$, the regularizer $R(L) = \sum_{ij} e^{pL_{ij}} |w_{ij}|^p$ is strictly convex. What is different, however, is the set of constraints on the variables L_{ij}. These are the constraints that allow one to compute the variables Λ_i uniquely from the variables L_{ij} (or, equivalently, the variables M_{ij}). This is addressed by the following theorem.

Theorem 9.12. *Under the same conditions of Theorem 9.1, but using activation functions that satisfy for each unit i the relationship $f(\lambda x) = \lambda^{c_i} f(x)$, the corresponding stochastic generalized balancing algorithm converges to the unique minimum of a strictly convex optimization problem in the variables L_{ij}. The strictly convex objective function is given by $R(L) = \sum_{ij} e^{pL_{ij}} |w_{ij}|^p$. The constraints are linear and of the form:*

$$\sum_{i \in \pi} \left(\prod_{k=i}^{n} c_k \right) L_{ii-1} = 0 \tag{9.9}$$

for each path π from an input unit to an output unit, going sequentially through the units $0, 1, \ldots, n$, where 0 corresponds to the input unit, and n corresponds to the output unit of the path. The set of paths in the constraints must cover all the units in the network.

Proof. Let us assume that there is a consistent set of multipliers $\Lambda_0, \ldots, \Lambda_n$ associated with the coefficients $L_{ii-1} = \log M_{ii-1}$ along the path π, with $\Lambda_0 = \Lambda_n = 1$. Since $M_{ii-1} = \Lambda_i / \Lambda_{i-1}^{c_{i-1}}$, we can derive the multipliers Λ_i iteratively by propagating information from the input unit to the output unit, in the form:

$$\Lambda_i = M_{ii-1} \Lambda_{i-1}^{c_{i-1}} \quad \text{or} \quad \log \Lambda_i = L_{ii-1} + c_{i-1} \log \Lambda_{i-1} \tag{9.10}$$

Using the boundary conditions $\Lambda_0 = \Lambda_n = 1$ gives the formula in Theorem 9.12. The same arguments given for BiLU units can be used to complete the proof. □

Remark 9.13. Note that if all the units have the same exponent c associated with the scaling of their activation functions, then the linear constraints have the simplified form:

$$\sum_{i \in \pi} c^{n+1-i} L_{ii-1} = 0 \tag{9.11}$$

Next we present two sets of simulations. The first set of simulations aims to corroborate the theory. The second set of simulations is more practical and shows how balancing can be used in practice during training to improve overall performance.

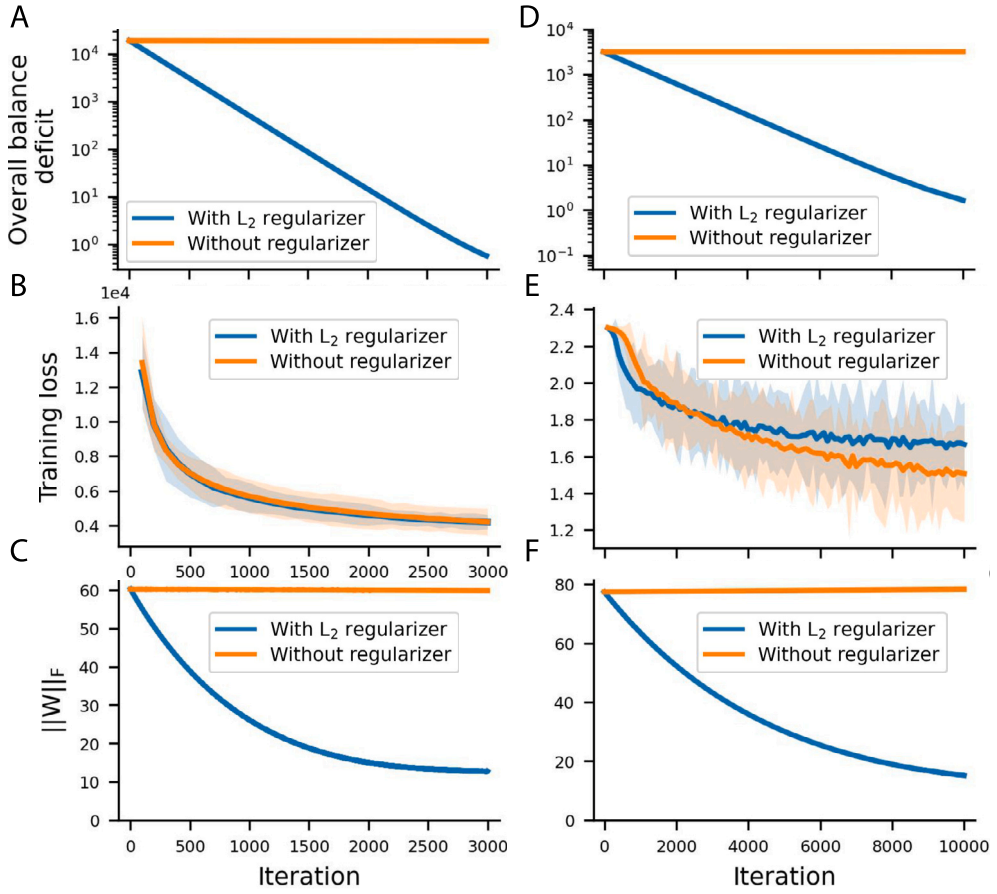


Fig. 7. SGD applied to E alone, in general, does not converge to a balanced state, but SGD applied to $E + R$ converges to a balanced state. (A-C) Simulations use a deep fully connected autoencoder trained on the MNIST dataset. **(D-F)** Simulations use a deep locally connected network trained on the CIFAR10 dataset. **(A,D)** Regularization leads to neural balance. **(B,E)** The training loss decreases and converges during training (these panels are not meant for assessing the quality of learning when using a regularizer). **(C,F)** Using weight regularization decreases the norm of weights. **(A-F)** Shaded areas correspond to one s.t.d. around the mean (in some cases the s.t.d. is small and the shaded area is not visible).

10. Simulations: corroboration of the theory

The code and experiments for this section are publicly available at https://github.com/ARahmansetayesh/a-theory-of-neural-synaptic-balance. To further corroborate the results, we ran multiple experiments. Here we report the results from two series of experiments. The first one is conducted using a six-layer, fully connected, autoencoder trained on MNIST [11] for a reconstruction task with ReLU activation functions in all layers and the sum of squares errors loss function. The number of neurons in consecutive layers, from input to output, is 784, 200, 100, 50, 100, 200, 784. Stochastic gradient descent (SGD) learning by backpropagation is used for learning with a batch size of 200.

The second one is conducted using three locally connected layers followed by three fully connected layers trained on CIFAR10 [21] for a classification task with leaky ReLU activation functions in the hidden layers, a softmax output layer, and the cross entropy loss function. The number of neurons in consecutive layers, from input to output, is 3072, 5000, 2592, 1296, 300, 100, 10. Stochastic gradient descent (SGD) learning by backpropagation is used for learning with a batch size of 5.

In all the simulation figures (Figs. 7, 8, and 9) the left column presents results obtained from the first experiment, while the right column presents results obtained from the second experiment. While we used both L_1 and L_2 regularizers in the experiments, in the figures we report the results obtained with the L_2 regularizer, which is the most widely used regularizer. In Figs. 7 and 8, training is done using batch gradient descent on the MNIST and CIFAR data. The balance deficit for a single neuron i is defined as: $(\sum_{w \in IN(i)} w^2 - \sum_{w \in OUT(i)} w^2)^2$, and the overall balance deficit is defined as the sum of these single-neuron balance deficits across all the hidden neurons in the network. The overall deficit is zero if and only if each neuron is in balance. In all the figures, $||W||_F$ denotes the Frobenius norm of the weights.

Fig. 7 shows that learning by gradient descent with a L_2 regularizer results in a balanced state. Fig. 8 shows that even when the network is initialized in a balanced state, without the regularizer the network can become unbalanced if the fixed learning rate is not very small. Fig. 9 shows that the local stochastic balancing algorithm, by which neurons are randomly balanced in asynchronous fashion, always converges to the same (unique) global balanced state.

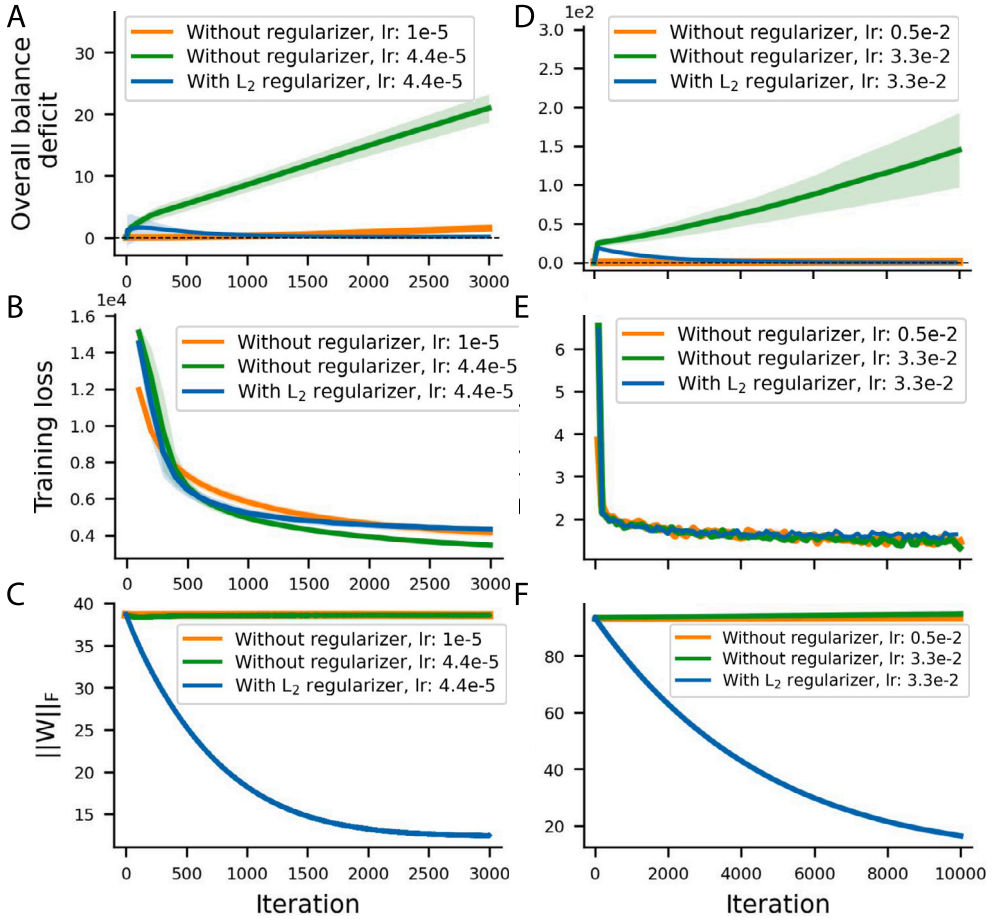


Fig. 8. Even if the starting state is balanced, SGD does not preserve the balance unless the learning rate is infinitely small. (A-C) Simulations use a deep fully connected autoencoder trained on the MNIST dataset. **(D-F)** Simulations use deep locally connected network trained on the CIFAR10 dataset. **(A-F)** The initial weights are balanced using the stochastic balancing algorithm. Then the network is trained by SGD. **(A,D)** When the learning rate (lr) is relatively large, without regularization, the initial balance of the network is rapidly disrupted. **(B,E)** The training loss decreases and converges during training (these panels are not meant for assessing the quality of learning when using a regularizer). **(C,F)** Using weight regularization decreases the norm of the weights. **(A-F)** Shaded areas correspond to one s.t.d. around the mean (in some cases the s.t.d. is small and the shaded area is not visible).

11. Simulations: improving training

The code and experiments for this section are publicly available at https://github.com/antonyalexos/Neural-Balance. In these simulations, we show that full balancing before training, as well as alternating partial balancing with stochastic gradient descent, are effective and practical regularization approaches that often outperform stochastic gradient descent paired with a traditional regularization function. We train all of our models on a server equipped with 8 Nvidia RTX A6000 Ada Generation graphics cards, with 384 GB of total memory, run on CUDA version 12.4. To ensure that our results are reproducible and fair, we repeat the experiments 8 times with 8 different random seeds shared across the different methodologies and take the average. In the experiments we use both L_1 and L_2 balancing. To assess both full balance before training and partial balance during training, we combine a variety of neural balancing operations. For the full balancing operation, we apply the balancing operation to the neurons of the model after weight initialization and before training until the ratio of the input to output norms converges to within 0.01 of 1. To assess partial balance during training, we perform partial balancing operations on the neurons of the model at every epoch during training. The partial balancing procedure applies the balancing operation to the neurons in the network only once, usually going from input neurons to output neurons without reaching equilibrium. In all the Tables in this section, optimal results are shown in bold.

11.1. Toy experiment on a circle toy dataset

The toy experiment uses a simple 2-dimensional concentric circle classification task and a simple network containing 2 input neurons, for each coordinate in the 2-dimensional plane, a hidden layer with 5 neurons, and a single output neuron for the binary classification task. The input and hidden layers apply a ReLU activation, and the output neuron uses a logistic activation for classification. The layers are fully connected. The loss is calculated using the binary cross-entropy for classification tasks. We use Adaptive

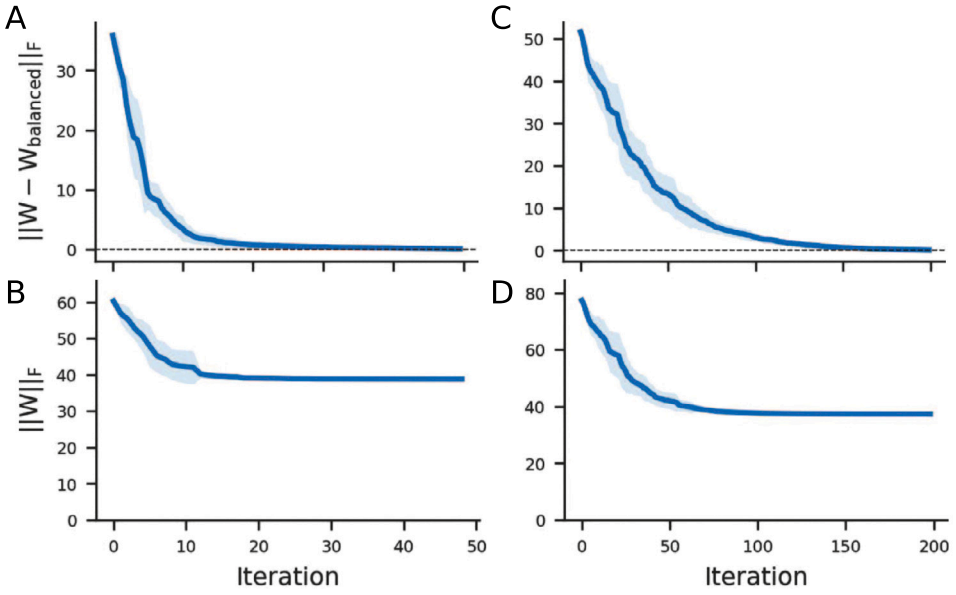


Fig. 9. Stochastic balancing converges to a unique global balanced state (A-B) Simulations use a deep fully connected autoencoder trained on the MNIST dataset. **(C-D)** Simulations use deep locally connected network trained on the CIFAR10 dataset. **(A,C)** The weights of the network are initialized randomly and saved. The stochastic balancing algorithm is applied and the resulting balanced weights are denoted by $W_{balanced}$. The stochastic balancing algorithm is applied 1,000 different times. In all repetitions, the weights converge to the same value $W_{balanced}$. **(B,D)** Stochastic balancing decreases the norm of the weights. **(A-D)** Shaded areas correspond to one standard deviation around the mean.

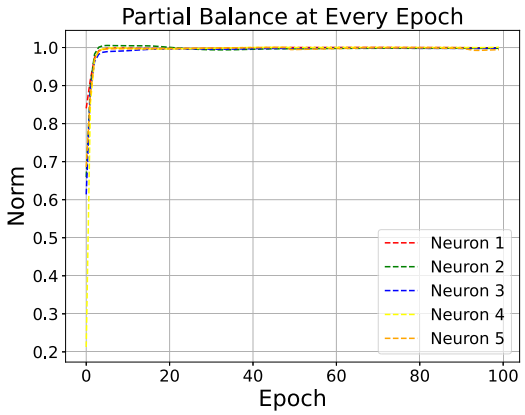


Fig. 10. Partial neural balancing performed at every epoch on a toy model with 5 neurons, trained on a concentric-circle binary classification dataset. We observe the effect of partial balancing applied at each learning epoch and show that input and output weight norms become equalized for every neuron over time and their ratio converges to 1.

Moment Estimation (Adam) as the optimizer during training, with a learning rate of 0.01, and we train the model for 1000 epochs. To compare the outcome of full balancing with partial balancing operations, we perform both during training at every epoch. The full balancing procedure applies the balancing operation to the neurons in the network until the ratio of the input and output norms of every neuron converge within .01 of 1. The partial balancing procedure applies the balancing operation to the neurons of the neural network only once every epoch, proceeding from the input to the output neurons.

Fig. 10 illustrates the effect of partial balancing applied at each learning epoch and shows that input and output weight norms become equalized for every neuron over time. Fig. 11 illustrates the effect of full balancing with instantaneous equalization of the input and output norms of all the neurons. While both approaches converge to a balanced state, partial balancing is computationally less intensive and thus provides an effective alternative.

11.2. Full balancing before training applied to fully connected networks

In these experiments, we study the effect of full balancing prior to training on fully connected networks of various sizes. The networks are trained on the MNIST dataset. The networks are fully connected with an input size of 784, representing the flattened input shape of 28x28, and an output size of 10. The 2 layer model has a single hidden layer with 256 neurons; the 3 layer model has

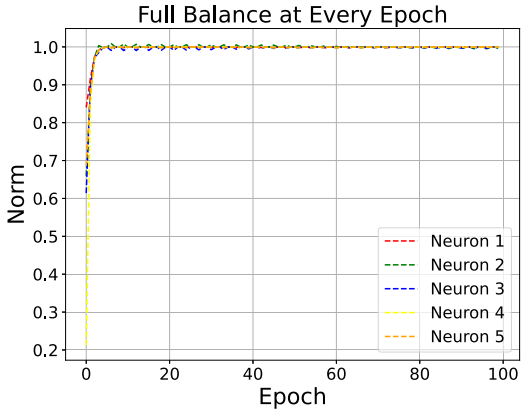


Fig. 11. Full neural balancing performed at every epoch on a toy model with 5 neurons, trained on a concentric-circle binary classification dataset. We observe the effect of full balancing applied at each learning epoch and show that input and output weight norms become equalized for every neuron over time and their ratio converges to 1.

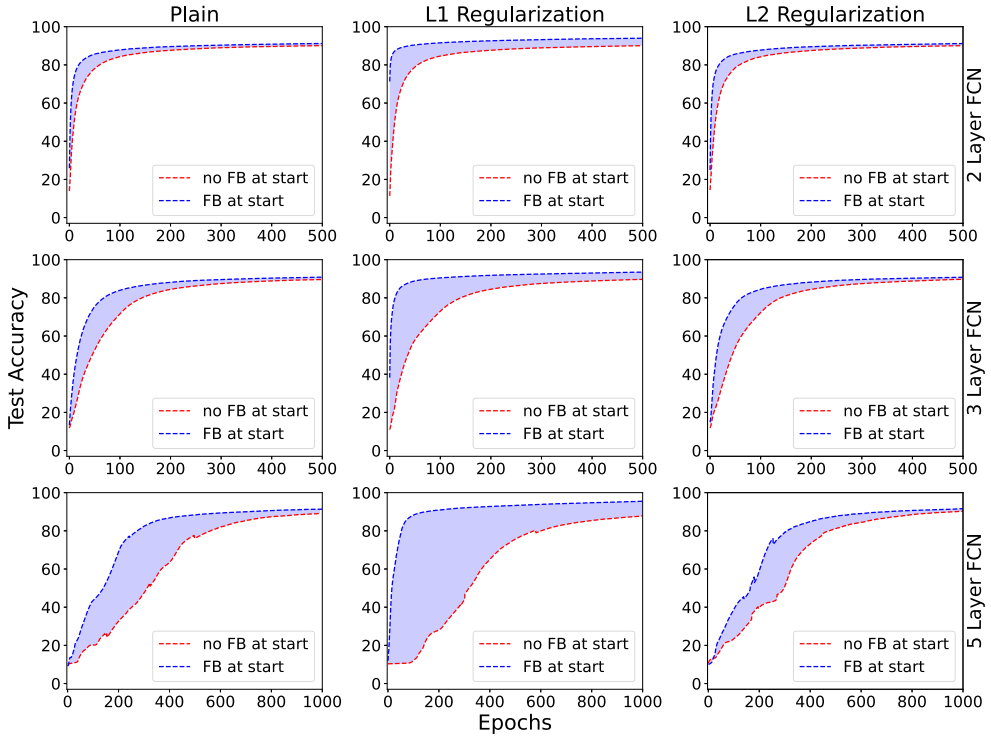


Fig. 12. The effect of full balancing (FB) before the start of training on various sizes of fully connected networks (FCN) with 2,3, and 5 hidden layers, trained on the MNIST dataset, without regularization, with L_1 regularization, and with L_2 regularization. FB at the beginning of learning results in faster convergence and higher test accuracy. The shading in each figure highlights the improvement. Table 1 reports the final text accuracy values.

2 hidden layers with 25 and 128 neurons; and the 5 layer model has 4 hidden layers with 512, 256, 128, and 64 neurons. All neurons have ReLU activation functions. We optimize the cross entropy loss for classification with stochastic gradient descent with a learning rate of 0.001. For experiments using L_1 and L_2 regularization, we use a regularization coefficient $\lambda = 0.015$ as we found it to be the most effective in this specific experiment via hyperparameter tuning.

In Fig. 12 and Table 1, we evaluate the impact of applying the full balancing operation prior to the commencement of training. Compared to standard initialization, full balancing leads to faster convergence and higher overall accuracy when using the same model architecture, hyperparameters, and training methodologies. Notably, the size of the benefit seems to increase with the size of the models. Fig. 12 provides a comparison between full balancing and no balancing applied to a Fully Connected Network (FCN) before training. Each square within the grid represents a combination of model size and training methodology, consistently demonstrating that neural balancing enhances the rate of convergence and model accuracy.

Table 1
Test accuracy after training of plain, L_1-regularized, and L_2 regularized fully connected networks trained on MNIST, comparing full balancing (FB) before training with no balancing before training. As already observed in Fig. 12, full balancing before training results in higher test accuracy.

Type	No FB at Start			FB at Start		
	Plain	L1 Reg.	L2 Reg.	Plain	L1 Reg.	L2 Reg.
2 Layer FCN	$90.09 \pm 0.06\%$	$90.05 \pm 0.08\%$	$90.062 \pm 0.11\%$	$\mathbf{91.22 \pm 0.13\%}$	$\mathbf{93.96 \pm 0.11\%}$	$\mathbf{91.18 \pm 0.05\%}$
3 Layer FCN	$89.594 \pm 0.05\%$	$89.67 \pm 0.09\%$	$89.70 \pm 0.23\%$	$\mathbf{90.83 \pm 0.06\%}$	$\mathbf{93.47 \pm 0.11\%}$	$\mathbf{90.79 \pm 0.1\%}$
5 Layer FCN	$89.09 \pm 0.12\%$	$87.85 \pm 0.2\%$	$90.3 \pm 0.22\%$	$\mathbf{91.37 \pm 0.1\%}$	$\mathbf{95.50 \pm 0.26\%}$	$\mathbf{91.59 \pm 0.2\%}$

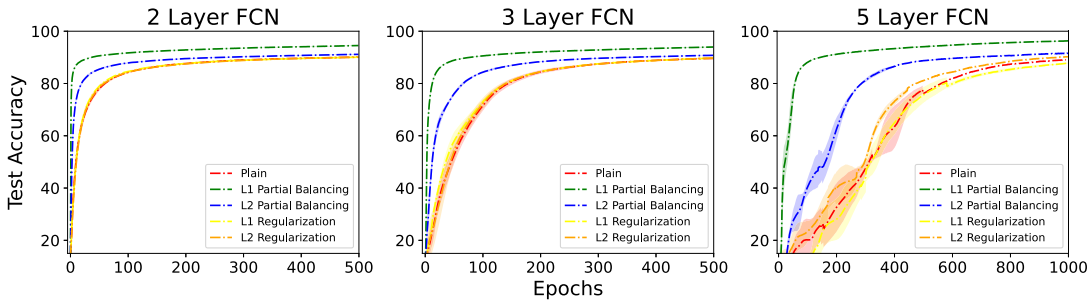


Fig. 13. The effect of partial balancing during training on various sizes of fully connected networks (FCN) with 2,3, and 5 hidden layers, trained on the MNIST dataset, without regularization, with L_1 regularization, and with L_2 regularization. Partial balancing results in faster convergence and higher test accuracy. Table 2 contains the tabularized data from the plots for easier comparison.

Table 2
Comparison of test accuracy obtained with different training methods and 3 architectures of increasing depth size. As already observed in Fig. 13, L1 partial balancing outperforms the other training methodologies on all model sizes.

Type	Plain	L1 PB	L2 PB	L1 Reg.	L2 Reg.
2-FCN	$91.22 \pm 0.05\%$	$\mathbf{94.54 \pm 0.11\%}$	$91.19 \pm 0.14\%$	$93.96 \pm 0.08\%$	$91.18 \pm 0.11\%$
3-FCN	$90.84 \pm 0.06\%$	$\mathbf{93.94 \pm 0.16\%}$	$90.86 \pm 0.11\%$	$93.47 \pm 0.09\%$	$90.79 \pm 0.23\%$
5-FCN	$91.37 \pm 0.12\%$	$\mathbf{96.26 \pm 0.11\%}$	$91.63 \pm 0.12\%$	$95.48 \pm 0.2\%$	$91.59 \pm 0.22\%$

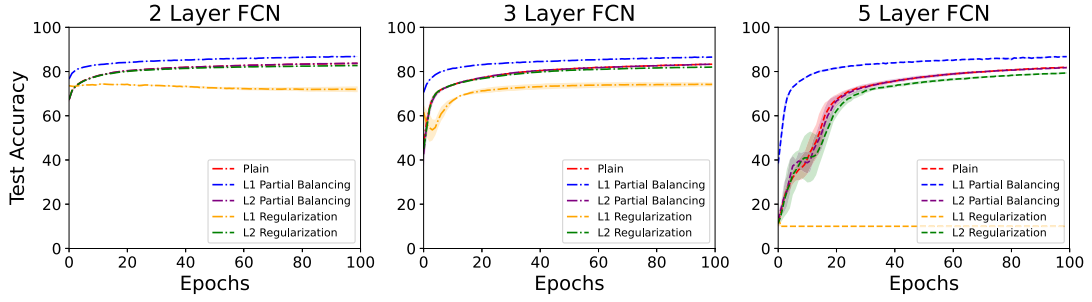


Fig. 14. The effect of partial balancing during training on various sizes of fully connected networks (FCN) with 2,3, and 5 hidden layers, trained on the FashionMNIST dataset. Partial balancing results in faster convergence and higher test accuracy. As the size of the models increases, partial balancing during training helps models converge faster and perform better.

11.3. Partial balancing during training applied to fully connected networks

Here we conduct the exact same experiments as in the previous section 11.2, but this time we use partial balancing during training in alternation with stochastic gradient descent steps.

From Fig. 13 and Table 2, we observe that partial balancing during training results in faster convergence and better test accuracy.

For increased robustness, we conduct the same tests using the FashionMNIST dataset. We utilize similar fully connected networks of various sizes and implement partial balancing at every epoch. Similar improvements in convergence and performance are observed. Irrespective of model size or training methodology, partial neural balancing markedly enhances the rate of convergence and overall test accuracy. As the models grow bigger, the inclusion of partial neural balancing in training helps models converge faster and perform better when compared to the other techniques (see Fig. 14).

Table 3
Test accuracy for a recurrent neural network trained on the IMDB sentiment analysis dataset, comparing plain, L_1-regularized, and L_2-regularized models with and without full balancing (FB) at the start of training. In all cases, full balancing before training results in higher test accuracy at the end of training.

Type	Plain	L1 Regularization	L2 Regularization
No FB at Start	88.26%	$88.23 \pm 0.06\%$	$88.22 \pm 0.15\%$
FB at Start	**88.64%**	$\mathbf{88.24 \pm 0.047\%}$	$\mathbf{88.57 \pm 0.09\%}$

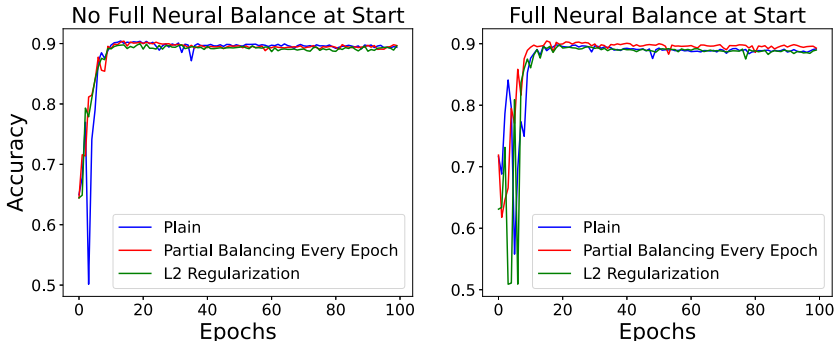


Fig. 15. Comparison between plain training (no regularization, no partial balancing) training with partial balancing, and training with L_2 regularization of a recurrent neural network on the IMDB dataset. We also contrast the standard random initialization with full balancing performed before the start of training. Both partial balancing during training, and full balancing before training, result in faster convergence, and higher overall accuracy.

11.4. Full balancing before training applied to recurrent neural networks

We continue the assessment of synaptic neural balancing by applying it to recurrent neural networks (RNNS). More specifically we assess the effect of full balancing before training. The experiments are carried on the IMDB sentiment analysis dataset. We use a recurrent neural network with an input layer, and output layer, and a fully-connected recurrent hidden layer. Plain text is first passed through an embedding layer, converting it to an embedding vector of length 100. The network uses 256 hidden neurons, which are fully connected among themselves. These recurrent neurons are updated three times synchronously before computing the final output. The final classification output is produced by a single logistic neuron. We use the binary cross entropy loss for training together with stochastic gradient descent with a learning rate of 0.001. For experiments using L_1 and L_2 regularization, we use a regularization rate of $\lambda = 0.0001$ as we found it to be the most effective via hyperparameter tuning.

The results are shown in Table 3, demonstrating that, in all cases, when full balancing is performed before training, the model has better final accuracy.

11.5. Partial balancing during training applied to recurrent neural networks

Similarly, here we assess the application of partial balancing during training, and demonstrate its efficacy at increasing the speed of convergence and the overall test accuracy, for recurrent neural network architectures. The experiments are also carried out on the IMDB sentiment analysis dataset, using the same recurrent network as in the previous section.

The main results are summarized in Fig. 15 where we compare the test accuracy achieved with plain training (no regularization and no balancing), training with partial balancing, and training with L_2 regularization. We also compare implementing full balancing before the start of training to standard random initialization. Both partial balancing during training and full balancing before training lead to faster convergence and higher accuracy.

11.6. Balancing with limited data

To further examine the regularizing benefits of synaptic balancing, here we vary the amount of available training data. For these experiments, we use both the MNIST handwritten digit classification dataset and the IMDB sentiment analysis dataset. Unlike the previous experiments, we use a stratified sample of the dataset to simulate a limited-data environment for the models. We use Scikit-Learn to take a stratified split of the overall dataset. For the MNIST dataset, we take 600 samples, representing 1% of the full dataset, with 60 samples from each label, in order to maintain the label balance present in the main dataset. For the IMDB dataset, we take 1250 samples, representing 5% of the full dataset, with 625 samples from the positive and negative labels, to preserve an equal distribution of labels.

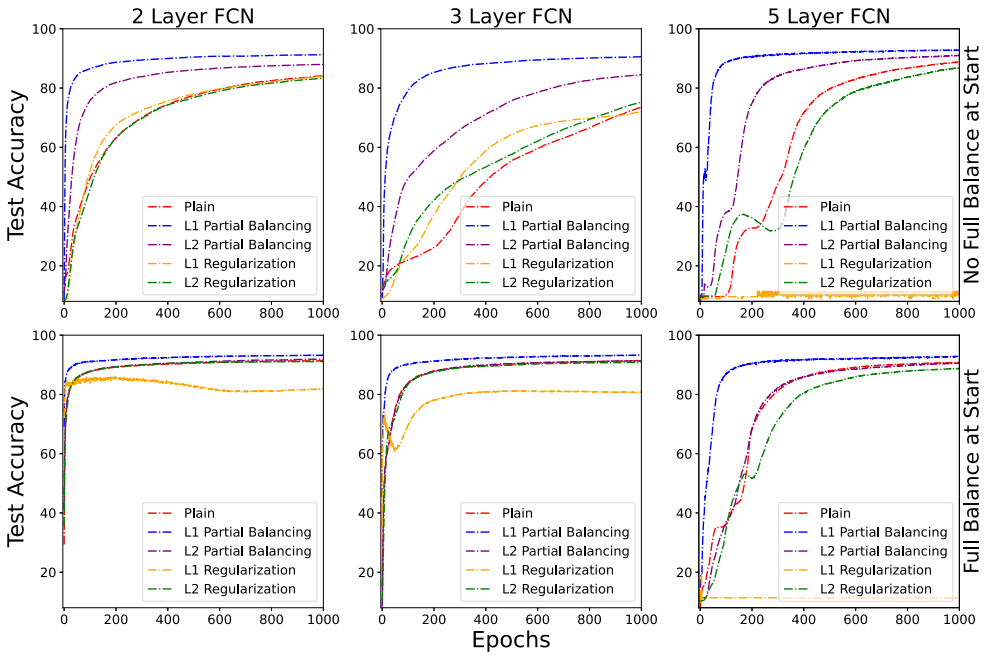


Fig. 16. Comparison between plain training (no balancing, no regularization), training with partial balancing, and training with regularization, with and without a full balancing at the start of training, using different fully connected networks trained on only 1% of the MNIST dataset. Table 4 shows the final test accuracy results for each of the training methodologies. Neural balancing consistently improves both the rate of convergence and the overall accuracy of the model.

Table 4
Test accuracy of various methodologies trained using 1% of the MNIST dataset to simulate a limited data environment. Consistently with Fig. 16, in all cases the use of full balancing at the start of training not only increases the rate of convergence, but also leads to higher accuracy.

Type	No FB at Start			FB at Start		
	2 Layer FCN	3 Layer FCN	5 Layer FCN	2 Layer FCN	3 Layer FCN	5 Layer FCN
Plain	$84.15 \pm 0.15\%$	$73.49 \pm 0.18\%$	$88.9 \pm 0.03\%$	$\mathbf{91.39 \pm 0.04\%}$	$\mathbf{91.42 \pm 0.08\%}$	$\mathbf{90.86 \pm 0.21\%}$
L1 PB	$91.25 \pm 0.5\%$	$90.57 \pm 0.1\%$	$92.87 \pm 0.1\%$	$\mathbf{93.26 \pm 0.01\%}$	$\mathbf{93.3 \pm 0.03\%}$	$\mathbf{92.92 \pm 0.11\%}$
L2 PB	$87.99 \pm 0.11\%$	$84.53 \pm 0.16\%$	$91.06 \pm 0.22\%$	$\mathbf{91.94 \pm 0.03\%}$	$\mathbf{91.37 \pm 0.08\%}$	$\mathbf{90.59 \pm 0.22\%}$
L1 Reg.	$83.92 \pm 0.16\%$	$72.03 \pm 0.18\%$	$11.35 \pm 0.09\%$	$\mathbf{85.98 \pm 0.017\%}$	$\mathbf{81.33 \pm 0.05\%}$	$\mathbf{19.79 \pm 0.03\%}$
L2 Reg.	$83.35 \pm 0.16\%$	$75.21 \pm 0.16\%$	$86.81 \pm 0.25\%$	$\mathbf{91.16 \pm 0.04\%}$	$\mathbf{90.86 \pm 0.08\%}$	$\mathbf{88.78 \pm 0.22\%}$

For the experiments with the MNIST handwritten digit recognition dataset, we use the same fully connected network architecture used for the experiments that are performed on the whole dataset. We use the cross entropy loss with stochastic gradient descent with a learning rate of 0.001. For experiments using L_1 and L_2 regularization, we use a regularization rate $\lambda = 0.015$ as we found it to be the most effective via hyperparameter tuning. We test a combination of full balancing and partial balancing before or during training. With full balancing, the balancing operation is performed after initialization and before training until the ratio of the input to output norms of each neuron equalizes within a threshold of .001 of 1. For partial balancing, we balance each neuron once from inputs to output. This partial balancing step is carried out at the end of each training epoch.

For the experiments with the IMDB sentiment analysis dataset, we use the same recurrent neural network architecture that is used for the experiments performed on the whole dataset. We use binary cross entropy loss with stochastic gradient descent with a learning rate of 0.001. For experiments using L_2 regularization, we use a regularization rate $\lambda = 0.0001$ as we found it to be the most effective via hyperparameter tuning. We assess the performance of partial balancing by comparing it to L_2 regularization and plain train (no regularizaation, no balancing). For partial balancing, we balance each neuron once from inputs to output. This partial balancing step is carried out at the end of each training epoch.

In data-scarce environments, neural networks incorporating balancing techniques exhibit accelerated convergence and enhanced accuracy compared to both unmodified models and those employing traditional regularization methods. Both full balancing at the start of learning, and partial balancing during learning lead to notable improvements.

We continue this assessment by training the same RNN as above using only 5% of the IMBD dataset. The results are provided in Fig. 17 and Table 5. Partial balancing leads to faster convergence and higher accuracy, and thus in general balancing can improve generalization in data scarce environments.

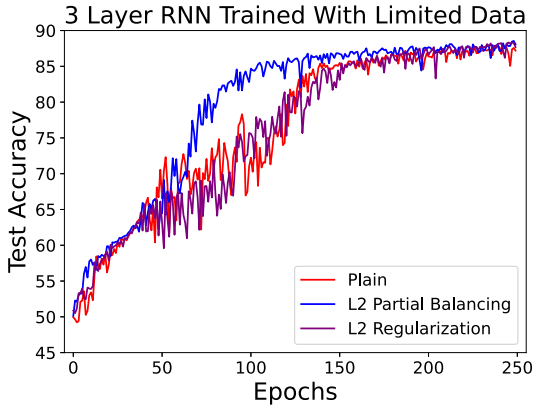


Fig. 17. Comparison between plain training (no regularization, no balancing), training with partial balancing, and training with L_2 regularization using a RNN and only 5% of the IMDB dataset. L2 partial balancing demonstrates both faster convergence and a modest improvement in final accuracy. While the performance gains are not dramatic, they are consistent across runs. Table 5 shows the final test accuracy.

Table 5
Test accuracy for a recurrent neural network trained on 5% of the IMDB sentiment analysis dataset, comparing plain training, training with partial balancing, and training with L_2 regularization. Consistently with the results in Fig. 17, L_2 partial balancing performed at every epoch leads to the highest degree of accuracy.

Type	Plain	L2 Partial Balancing	L2 Regularization
	$87.87 \pm 0.12\%$	$\mathbf{88.55 \pm 0.1\%}$	$88.34 \pm 0.09\%$

11.7. Balancing with non-BiLU activation functions

In all the previous experiments, we used neurons with ReLU activation functions. Here, we conduct similar experiments with non-BiLU neurons, in particular using sigmoidal neuron (tanh and logistic). For these experiments, we parallel the full balance experiments in section 11.2, and the partial balancing experiments in section 11.3. For both experiments, we use the same fully connected feedforward architectures with the same sizes, and the same hyperparameter settings as in the aforementioned sections.

In Fig. 18, we observe an improvement in both the rate of convergence and the final accuracy when full balancing is applied at the start of training. This is true across the architectures that were tested and across the training methodologies, with and without regularization. Thus balancing at the start of training appears to be useful even when non-BiLU neurons are used. Note that tanh units behave like BiLU units, and more precisely like linear units, when the weights are very small, i.e., at the beginning of training.

Next, we continue with a comparison of partial balancing with other training methodologies on the MNIST dataset, using tanh activation functions. We observe in Fig. 19 and Table 7 that partial balancing improves convergence and accuracy. In this case, the best results are obtained with L_1 partial balancing. The use of L_1 partial balancing significantly improves the rate of convergence and the final test accuracy of the models, irrespective of their size. Again this demonstrates the effectiveness of balancing even in models that use non-BiLU activation functions.

11.8. Balancing in CNNs

To investigate the effects of weight balancing in CNNs during training, we evaluate the two main algorithms described in Section 8.3: coordinated channel balancing (one λ per channel) and neuron-wise balancing (one λ per neuron in each channel). In this experiment, both methods are applied during training using the CIFAR-10 data and the VGG-11 and VGG-16 architectures and the L_2 cost function for the synaptic weights. The VGG-11 architecture has about 133 millions parameters, and the VGG-16 architecture has about 138 millions parameters.

The results are shown in Fig. 20 and Fig. 21 respectively. The term “plain” refers to training without balancing or regularization. As in the other experiments in this section, error bars are obtained using eight runs carried out with eight different random seeds. In these figures, we see that channel balancing tends to learn in fewer epochs and achieves slightly higher accuracy when compared to neuron-wise balancing or plain training.

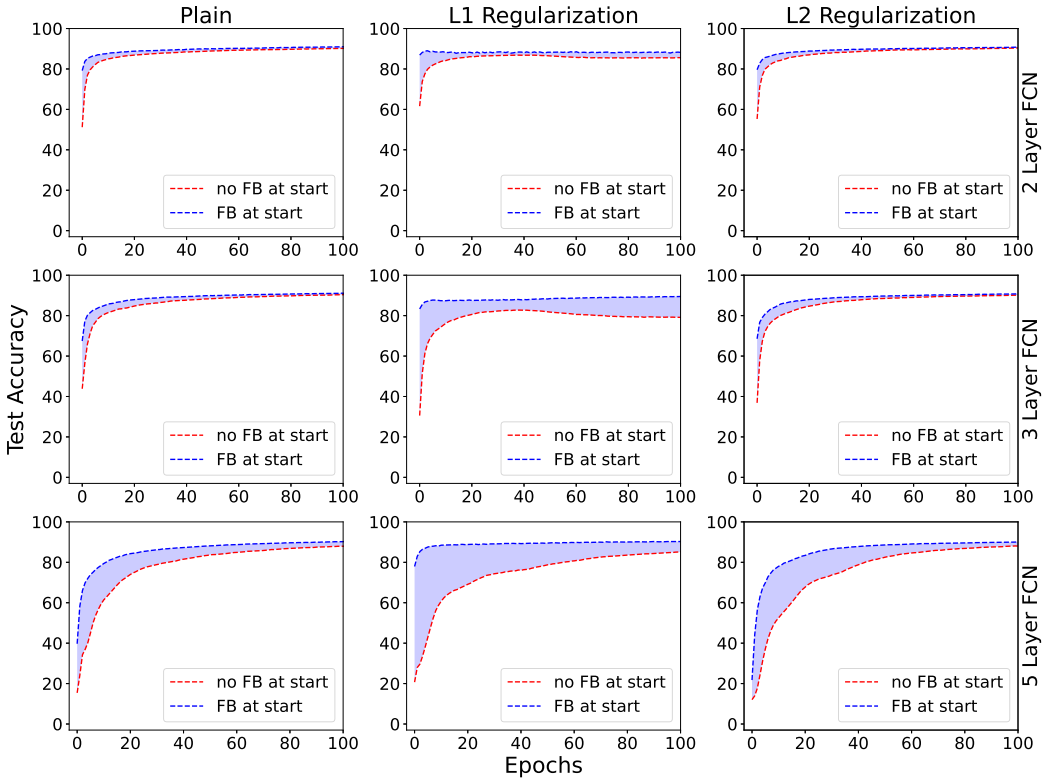


Fig. 18. Effect of full balancing (FB) before the start of training of fully connected feedforward networks of various depths, with tanh activation functions, using different training approaches: plain (no regularization, no balancing), training with L_1 or L_2 regularization. Regardless of the training method, full balancing at the start results in faster convergence and higher accuracy. The shading illustrates the performance improvement. Table 6 provides the final accuracy values.

Table 6
Test accuracy for plain, L_1-regularized, and L_2-regularized fully connected feedforward networks with tanh units trained on the MNIST dataset, comparing the effect of full balancing before training to no full balancing before training. Consistently with the results displayed in Fig. 18, full balancing before training results in faster convergence, as well as higher test accuracy, across all methods.

Type	No FB at Start			FB at Start		
	Plain	L1 Reg.	L2 Reg.	Plain	L1 Reg.	L2 Reg.
2 Layer FCN	$91.69 \pm 0.12\%$	$87.11 \pm 0.1\%$	$91.72 \pm 0.03\%$	$\mathbf{91.97 \pm 0.24\%}$	$\mathbf{89 \pm 0.19\%}$	$\mathbf{91.9 \pm 0.05\%}$
3 Layer FCN	$92.28 \pm 0.2\%$	$82.75 \pm 0.15\%$	$92.03 \pm 0.09\%$	$\mathbf{92.58 \pm 0.14\%}$	$\mathbf{90.23 \pm 0.13\%}$	$\mathbf{92.08 \pm 0.06\%}$
5 Layer FCN	$91.64 \pm 0.23\%$	$87.61 \pm 0.2\%$	$90.99 \pm 0.13\%$	$\mathbf{92.48 \pm 0.25\%}$	$\mathbf{91.28 \pm 0.16\%}$	$\mathbf{91.91 \pm 0.15\%}$

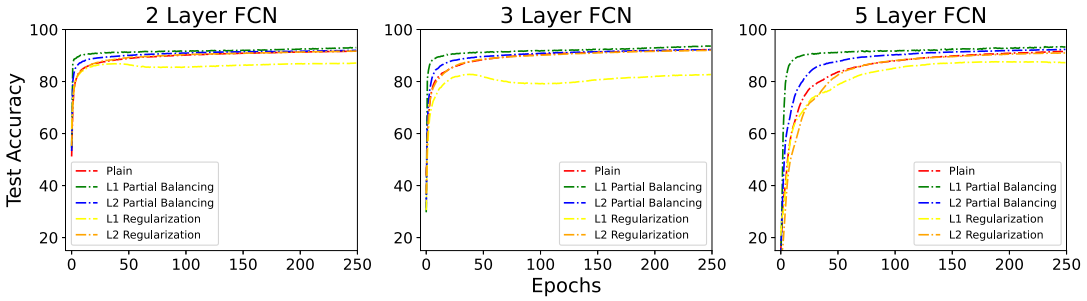


Fig. 19. Comparison of partial L_1 and L_2 balancing to L_1-regularized and L_2-regularized training for different architectures of tanh units trained on the MNIST dataset. As the models grow bigger, partial balancing helps the models converge faster and perform better. Table 7 reports the final test accuracy values.

Table 7
Comparison of test accuracy across training methods including plain training (no regularization, no balancing), partial balancing (PB) using L_1 or L_2 costs, and L_1 or L_2 regularized training using the MNIST dataset for fully connected networks (FCNs) of depth 2, 3, and 5. The networks use tanh activation functions. As observed in the plots in Fig. 19, L_1 partial balancing outperforms the other training methodologies across all model sizes.

Type	Plain	L1 PB	L2 PB	L1 Reg.	L2 Reg.
2-FCN	$91.69 \pm 0.12\%$	$\mathbf{93.07 \pm 0.26\%}$	$92.0 \pm 0.19\%$	$87.11 \pm 0.1\%$	$91.72 \pm 0.03\%$
3-FCN	$92.28 \pm 0.2\%$	$\mathbf{93.66 \pm 0.26\%}$	$92.27 \pm 0.17\%$	$82.75 \pm 0.15\%$	$92.03 \pm 0.09\%$
5-FCN	$91.64 \pm 0.23\%$	$\mathbf{93.44 \pm 0.26\%}$	$92.36 \pm 0.1\%$	$87.61 \pm 0.2\%$	$90.99 \pm 0.13\%$

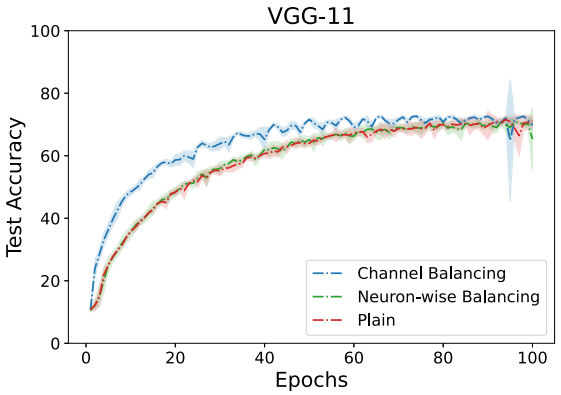


Fig. 20. Comparison between balancing strategies applied during training of the VGG-11 (∼ 133M parameters) architecture from scratch, using the CIFAR-10 data. Channel balancing achieves the fastest convergence and highest final accuracy. Neuron-wise balancing shows marginal improvements in convergence speed and accuracy over plain training by stochastic gradient descent with respect to E.

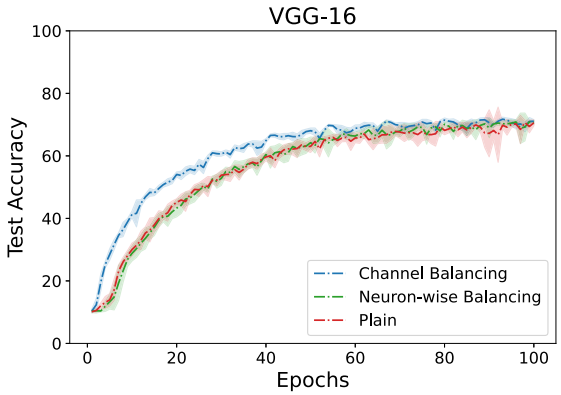


Fig. 21. Comparison between balancing strategies applied during training of the VGG-16 (∼ 138M parameters) architecture from scratch, using the CIFAR-10 data. Channel balancing achieves the fastest convergence and highest final accuracy. Neuron-wise balancing shows marginal improvements in convergence speed and accuracy over plain training by stochastic gradient descent with respect to E.

12. Discussion

The work in [32] contains results that are closest to ours, but from a somewhat different perspective.[1] Their perspective is primarily biological with an exclusive focus on recurrent neural networks. Their approach derives a local learning rule to enhance neural network robustness by minimizing sensitivity to noise. This sensitivity-minimizing rule is a balancing rule that aims to equilibrate the aggregate incoming and outgoing synapses of each neuron (see Fig. 2 in their work) and is consistent with experimental evidence of heterosynaptic plasticity. Furthermore, these authors show that their approach converges, that it can be applied with general cost functions, and that the dynamics induced by the rule can be approximated through a heat equation. Our approach is motivated by the observation of emergent balance during neural network training. The formalism we use is different and we derive new results in terms

[1] Our work was carried out independently and roughly at the same time–preliminary versions were submitted to, and rejected by, several conferences, including NeurIPS 2023 and 2024.

of the generality of the cost functions, the generality of the activation functions (e.g., BiPU neurons), the geometric properties of the underlying optimization problems (Fig. 6), and the kinds of neural architectures (e.g., feedforward, convolutional, recurrent). We also derive analytic solution in special cases (Appendix B), as well as a variety of different balancing algorithms (e.g., partial balancing, full balancing). Finally, we conducted completely different simulations focused on training various neural network architectures from actual data with backpropagation and balancing. We believe that the heat equation approximation could be applied in our context too but have left it for future work.

Although the theory of synaptic neural balance is a mathematical theory that stands on its own, it is worth considering some of its possible consequences and applications, at the theoretical, algorithmic, biological and neuromorphic hardware levels.

12.1. Theory

Theories of deep learning in networks of McCulloch and Pitt neurons often proceed by fixing the kinds of activation functions and neurons that are being used. At one extreme end of the spectrum, linear networks are found for which a rich theory is available [5,4]. At the other most non-linear end of the spectrum, one finds networks of unrestricted Boolean functions which are also amenable to fairly general analyses [3]. In between, one typically considers networks of linear threshold neurons [9], or sigmoidal neurons (logistic or tanh activation functions), or ReLU neurons [26]. The results shown here suggest that results obtained for networks of linear or ReLU neurons, may be extendable to networks of BiLU neurons, especially when the homogeneity property plays an essential role, and perhaps also other neurons such as RePU neurons. In short, a line of investigation suggested by this work to study all the basic questions about deep learning (e.g. capacity, generalization, universal approximation properties) in networks of BiLU neurons of increasing architectural complexity. It is easy to show, for instance, that BiLU networks have universal approximation properties (see Appendix).

Our results show that for a given architecture with weights W, there is an entire equivalence class of weights with the same overall performance, associated with scaling operations and the underlying linear manifold. Global balancing can be viewed as a systematic way of selecting a unique canonical representative within the class, associated with the corresponding balanced network. Among other things, the existence of such equivalence classes implies that the information in the training data does not need to be able to specify the individual weights, but may instead specify the equivalence class of the weights. This is tied to the formal notion of capacity [9] and explain in part why large networks can be trained with less than $|W|$ examples while not overfitting [7]. It is worth noting that the equivalence classes corresponding to weights that provide the same input-output function may be even larger than what is given by the linear manifold of scalings, as they could contain other operations besides scaling, such as permuting the neurons of any given layer in the fully connected case. However, in the case of a feedforward network of BiLUs where the units are numbered, and connections run only from lower-numbered units to higher-numbered units, then each unit has its unique connectivity pattern and the equivalence classes may be restricted to scaling operations.

Another theoretical application is the study of learning in linear networks with L_p regularization. For instance, it is well known that a feedforward, fully connected, linear autoencoder with bottleneck layers trained to minimize the sum of square reconstruction error E has a unique global optimum, up to trivial transformations, corresponding to Principal Component Analysis (PCA) in the bottleneck layers. Furthermore, that problem has no spurious local minima and all other critical points of the error function are saddle points associated with projections onto linear spaces spanned by the non-principal components. What happens when an L_p regularizer R is added to the reconstruction error, so that the overall error is given by $E = E + \beta R$. For β very large, R will dominate and the optimal solution is to have all the weights equal to 0. However, when β is small, the optimal solution is provided by the theory presented here. The error is dominated by E so the optimum is associated with the PCA solution, as described above, which can then be refined by balancing to further reduce R and reach the global optimum.

12.2. Algorithms

The theory and the simulations in Section 11 show that synaptic balance has a number of different applications. It can be used: (1) To initialize the weights of a network at the start of training to better condition the learning; (2) To check that a network trained to minimize $E + R$ has been properly trained by checking that each neuron is balanced with respect to R; (3) To balance a non-properly trained network at the end of training; (4) To better regularize a network by alternating SGD steps applied to E, or even $E + R$, with partial or full balancing steps. This may improve the convergence or final performance of learning; (5) this of course can be applied to all situations where regularization is beneficial, including the case of smaller training sets; and (6) To tweak a network trained with ah L_p regularization, towards a state where it is L_q regularized (with $q \neq p$) without retraining it. Thus, in short, balancing is a little bit like dropout: it ought to be added to the arsenal of tools available to improve neural network training.

12.3. Biology

The balancing operations are *local* [8], in the sense that they involve only the pre- and post-synaptic weights of a neuron. Thus, unlike backpropagation, balancing is plausible in a physical neural system, as opposed to a digitally simulated neural system. Thus synaptic balancing could be of interest in neuroscience or in neuromorphic engineering, for instance from the standpoint of learning or memory maintenance. While there is extensive literature in neuroscience on the balance between excitation and inhibition in biological networks [37,33], there is little evidence in favor or against neuronal synaptic balance in the sense described here. However, there is some evidence for the existence of homeostatic processes that scale the input synaptic weights to regulate the activity of

neurons [35,10,36]. In addition, there is also evidence that biological neurons can scale their intrinsic excitability (spike threshold) to regulate their activity [17,14]. It is at least conceivable that these two scaling mechanisms could be at play and work together in some situations (see also [32]). In any case, current technology is quite far from allowing one to measure the strength of all the incoming and outgoing synapses of a biological neuron in an animal brain. Thus, it is difficult to draw any definitive conclusions on the existence of some kind of homeostasis between the incoming and outgoing synapses of biological neurons. Exploratory experiments could potentially be carried either in simpler organisms with a small number of well-defined neurons, such as *C. elegans*, or in cultured neurons. In addition, simulations could be carried out in more detailed compartmental neural network models, possibly ones where inhibitory and excitatory neurons are segregated in uneven proportions–the overall ratio of excitatory to inhibitory neurons in the mammalian cortex is roughly 80% to 20% [27], with biologically-observed connectivity patterns. Finally, as a most speculative and for now untestable hypothesis, one may conjecture that biological neurons may undergo balancing phases, and that perhaps those phases occur during the night, with a close association between synaptic balancing and dreaming.

12.4. Neuromorphic hardware

In physical neural networks (e.g. biological neural networks, neuromorphic chips), as opposed to digitally simulated neural networks, the algorithms for adjusting synaptic weights must be local both in space and time. Yet these networks must exhibit good global properties. Thus, algorithms that are local and lead to global order, such as synaptic balancing, are of particular interest when considering physical, non-simulated, neural systems. For example, while a deep neuron embedded in a physical network may have no way of monitoring the global training error, conceivably it could sense and monitor its degree of balance, since that is an entirely local property. The degree of balance could be used as a local proxy to monitor global learning progress. Furthermore, the physical properties of the underlying hardware could constrain the synaptic weights or the learning algorithms in ways that could require or benefit from some form of synaptic balance.

Optimization strategies for training neuromorphic spiking neural networks with low energy consumption are analyzed in [31] (see also [28]), including how the homogeneous properties of ReLU neurons can influence the number of spikes generated at each layer and the average energy consumption at each layer. Thus there is a direct connection between synaptic balance and some of the neuromorphic literature using scaling methods to control spiking rates and energy consumption. In particular, balancing may help minimize energy consumption. Similar considerations apply to memristor-based neuromorphic hardware [18,19].

13. Conclusion

The theory of synaptic neural balance explains some basic findings regarding L_2 balance in feedforward networks of ReLU neurons and extends them in several directions. The first direction is the extension to BiLU and other activation functions (BiPU). The second direction is the extension to more general regularizers, including all L_p ($p > 0$) regularizers. The third direction is the extension to non-layered architectures, recurrent architectures, convolutional architectures, as well as architectures with mixed activation functions. The theory is based on two local neuronal operations: scaling which is commutative, and balancing which is not commutative. Finally, and most importantly, given any initial set of weights, when local balancing operations are applied in a stochastic or deterministic manner, global order always emerges through the convergence of the balancing algorithm to the same unique set of balanced weights. The reason for this convergence is the existence of an underlying convex optimization problem where the relevant variables are constrained to a linear, only architecture-dependent, manifold. Balancing can be applied, fully or partially, before, during, or at after training and can help improve the convergence or final performance. Scaling and balancing operations are local and thus may have applications in physical, non-digitally simulated, neural networks where the emergence of global order from local operations may lead to better operating characteristics and lower energy consumption.

Contributions: The theory was developed by PB. The simulations to corroborate the theory were carried out by A. R. The simulations to improve training were carried out by I.D. and A. A. The manuscript was written by P.B. with input from all the other authors. All authors reviewed the final version.

CRediT authorship contribution statement

Pierre Baldi: Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Antonios Alexos:** Writing – review & editing, Visualization, Software. **Ian Domingo:** Writing – review & editing, Visualization, Validation, Software. **Alireza Rahmansetayesh:** Writing – review & editing, Visualization, Validation, Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Universal approximation properties of BiLU neurons

Here we show that any continuous real-valued function defined over a compact set of the Euclidean space can be approximated to any degree of precision by a network of BiLU neurons with a single hidden layer. As in the case of the similar proof given in [4] using

linear threshold gates in the hidden layer, it is enough to prove the theorem for a continuous function f: $0,1 \to \mathbb{R}$. In the hidden layer, we are going to use ReLU activations with slope λ, i.e., activation functions h_λ satisfying: $h_\lambda(x) = 0$ for $x < 0$ and $h_\lambda(x) = \lambda x$ for $0 \leq x$.

Theorem A.1. *(Universal Approximation Properties of BiLU Neurons) Let f be any continuous function from $[0,1]$ to $\mathbb{R}$ and $\epsilon > 0$. Let h_λ be the ReLU activation function with slope $\lambda \in \mathbb{R}$s. Then there exists a feedforward network with a single hidden layer of neurons with ReLU activations of the form h_λ, and a single linear output neuron, capable of approximating f everywhere within ϵ (sup norm).*

Proof. Since f is continuous on a compact set, it is uniformly continuous. Thus, there exists $\alpha > 0$ such that for any x_1 and x_2 in the $[0,1]$ interval:

$$|x_2 - x_1| < \alpha \implies |f(x_2) - f(x_1)| < \epsilon \tag{A.1}$$

Let N be an integer such that $1 < N\alpha$, and let us slice the interval $[0,1]$ into N consecutive slices of width $h = 1/N$, so that within each slice the function f cannot jump by more than ϵ. Let us connect the input unit to all the hidden units with a weight equal to 1. Let us have N hidden units numbered $1, \ldots, N$ with biases equal to $0, 1/N, 2/N, ..., N_1/N$ respectively and activation functions of the form h_{λ_k}. It is essential that different units be allowed to have different slopes λ_k. The input unit is connected to all the hidden units and all the weights on these connections are equal to 1. Thus when x is in the k-th slice, $(k-1)/N \leq x < k/N$, all the units from $k+1$ to N have an output equal to 0, and all the units from 1 to k have an output determined by the corresponding slopes. All the hidden units are connected to the output unit with weights $\beta_1, \ldots, \beta_N$, and β_0 is the bias of the output unit. We want the output unit to be linear. In order for the ϵ approximation to be satisfied, it is sufficient if in the $(k-1)/N \leq x < k/N$ interval, the output is equal to the line joining the point $f((k-1)/N)$ to the point $f(k/N)$. In other words, if $x \in [(k-1)/N, k/N)$, then we want the output of the network to be:

$$\beta_0 + \sum_{i=1}^{k} \beta_i \lambda_i (x - (i-1)h) = f(\frac{k-1}{N}) + \frac{f(\frac{k}{N}) - f(\frac{k-1}{N})}{h}(x - (k-1)h) \tag{A.2}$$

By equating the y-intercept and slope of the lines on the left-hand side and the right-hand side of Equation (A.2), we can solve for the weights β's and the slopes λ's. □

As in the case of the similar proof using linear threshold functions in the hidden layer (see [4]) this proof can easily be adapted to continuous functions defined over a compact set of $\mathbb{R}^n$, even with a finite number of finite discontinuities, and into $\mathbb{R}^m$.

Appendix B. Analytical solution for the unique global balanced state

Here we directly prove the convergence of stochastic balancing to a unique final balanced state, and derive the equations for the balanced state, in the special case of tied layer balancing (as opposed to single neuron balancing). The Proof and the resulting equations are also valid for stochastic balancing (one neuron at a time) in a layered architecture comprising a single neuron per layer. Let us call tied layer scaling the operation by which all the incoming weights to a given layer of BiLU neurons are multiplied by $\lambda > 0$ and all the outgoing weights of the layer are multiplied by $1/\lambda$, again leaving the training error unchanged. Let us call layer balancing the particular scaling operation corresponding to the value of λ that minimizes the contribution of the layer to the L_2 (or any other L_p) regularizer value. This optimal value of λ^* results in layer-wise balance equations: the sum of the squares of all the incoming weights of the layer must be equal to the sum of the squares of all the outgoing weights of the layer in the L_2 case, and similarly in all L^P cases.

Theorem B.1. *Assume that tied layer balancing is applied iteratively and stochastically to the layers of a layered feedforward network of BiLU neurons. As long as all the layers are visited periodically, this procedure will always converge to the same unique set of weights, which will satisfy the layer-balance equations at all layers, irrespective of the details of the schedule. Furthermore, the balance state can be solved analytically.*

Proof. Every time a layer balancing operation is applied, the training error remains the same, and the L_2 (or any other L_p) regularization error decreases or stays the same. Since the regularization error is always positive, it must converge to a certain value. Using the same arguments as in the proof of Theorem 9.1, the weights must also converge to a stable configuration, and since the configuration is stable all its layers must satisfy the layer-wise balance equation. The key remaining question is why is this configuration unique and can we solve it analytically? Let $A_1, A_2, \ldots A_N$ denote the matrices of connections between the layers of the network. Let $\Lambda_1, \Lambda_2, \ldots, \Lambda_{N-1}$ be $N-1$ strictly positive multipliers, representing the limits of the products of the corresponding λ_i^* associated with each balancing step at layer i, as in the proof of Theorem 9.1. In this notation, layer 0 is the input layer and layer N is the output layer (with $\Lambda_0 = 1$ and $\Lambda_N = 1$).

After converging, each matrix A_i becomes the matrix $\Lambda_i/\Lambda_{i-1} A_i = M_i A_i$ for $i = 1 \ldots N$, with $M_i = \Lambda_i/\Lambda_{i-1}$. The multipliers M_i must minimize the regularizer while satisfying $M_1 \ldots M_N = 1$ to ensure that the training error remains unchanged. In other words, to find the values of the M_i's we must minimize the Lagrangian:

$$\mathcal{L}(M_1, \dots, M_N) = \sum_{i=1}^{N} ||M_i A_i||^2 + \mu(1 - \prod_{i=1}^{N} M_i) \tag{B.1}$$

written for the L^2 case in terms of the Frobenius norm, but the analysis is similar in the general L_p case. From this, we get the critical equations:

$$\frac{\partial \mathcal{L}}{\partial M_i} = 2M_i||A_i||^2 - \mu M_1 \dots M_{i-1} M_{i+1} \dots M_N = 0 \quad \text{for } i = 1, \dots, N \quad \text{and} \quad \prod_{i=1}^{N} M_i = 1 \tag{B.2}$$

As a result, for every i:

$$2M_i||A_i||^2 - \frac{\mu}{M_i} = 0 \quad \text{or} \quad \mu = 2M_i^2||A_i||^2 \tag{B.3}$$

Thus each $M_i > 0$ can be expressed in a unique way as a function of the Lagrangian multiplier μ as: $M_i = (\mu/2||A_i||^2)^{1/2}$. By writing again that the product of the M_i is equal to 1, we finally get:

$$\mu^N = 2^N \prod_{i=1}^{N} ||A_i||^2 \quad \text{or} \quad \mu = 2 \prod_{i=1}^{N} ||A_i||^{2/N} \tag{B.4}$$

Thus we can solve for M_i:

$$M_i = \frac{\mu}{2||A_i||^2} = \frac{\prod_{i=1}^{N} ||A_i||^{2/N}}{||A_i||^2} \qquad \text{for } i = 1, \dots, N \tag{B.5}$$

Thus, in short, we obtain a unique closed-form expression for each M_i. From there, we infer the unique and final state of the weights, where $A_i^* = M_i A_i$. Note that each M_i depends on all the other M_j's, again showcasing how the local balancing algorithm leads to a unique global solution. □

Data availability

We use well known publicly available data sets. All the code and experiments are available on GitHub at the urls given in the paper.

References

[1] Marco Armenta, Thierry Judge, Nathan Painchaud, Youssef Skandarani, Carl Lemaire, Gabriel Gibeau Sanchez, Philippe Spino, Pierre-Marc Jodoin, Neural teleportation, Mathematics 11 (2) (2023) 480.

[2] Sanjeev Arora, Nadav Cohen, Noah Golowich, Wei Hu, A convergence analysis of gradient descent for deep linear neural networks, arXiv preprint arXiv: 1810.02281, 2018.

[3] P. Baldi, Autoencoders, unsupervised learning, and deep architectures, in: Proceedings of 2011 ICML Workshop on Unsupervised and Transfer Learning, J. Mach. Learn. Res. 27 (2012) 37–50.

[4] P. Baldi, Deep Learning in Science, Cambridge University Press, Cambridge, UK, 2021.

[5] P. Baldi, K. Hornik, Neural networks and principal component analysis: learning from examples without local minima, Neural Netw. 2 (1) (1989) 53–58.

[6] P. Baldi, P. Sadowski, The dropout learning algorithm, Artif. Intell. 210C (2014) 78–122.

[7] Pierre Baldi, Deep learning over-parameterization: the shallow fallacy, in: Northern Lights Deep Learning Conference, Proceedings Machine Learning Research, 2024, pp. 7–12.

[8] Pierre Baldi, Peter Sadowski, A theory of local learning, the learning channel, and the optimality of backpropagation, Neural Netw. 83 (2016) 61–74.

[9] Pierre Baldi, Roman Vershynin, The capacity of feedforward neural networks, Neural Netw. 116 (2019) 288–311, arXiv preprint arXiv:1901.00434.

[10] Marina Chistiakova, Nicholas M. Bannon, Jen-Yung Chen, Maxim Bazhenov, Maxim Volgushev, Homeostatic role of heterosynaptic plasticity: models and experiments, Front. Comput. Neurosci. 9 (2015) 89.

[11] Li Deng, The mnist database of handwritten digit images for machine learning research, IEEE Signal Process. Mag. 29 (6) (2012) 141–142.

[12] Simon S. Du, Wei Hu, Jason D. Lee, Algorithmic regularization in learning deep homogeneous models: layers are automatically balanced, Adv. Neural Inf. Process. Syst. 31 (2018).

[13] Rachel E. Field, James A. D'amour, Robin Tremblay, Christoph Miehl, Bernardo Rudy, Julijana Gjorgjieva, Robert C. Froemke, Heterosynaptic plasticity determines the set point for cortical excitatory-inhibitory balance, Neuron 106 (5) (2020) 842–854.

[14] Bertrand Fontaine, José Luis Peña, Romain Brette, Spike-threshold adaptation predicted by membrane potential dynamics in vivo, PLoS Comput. Biol. 10 (4) (2014) e1003560.

[15] Robert C. Froemke, Plasticity of cortical excitatory-inhibitory balance, Annu. Rev. Neurosci. 38 (2015) 195–219.

[16] Oliver D. Howes, Ekaterina Shatalina, Integrating the neurodevelopmental and dopamine hypotheses of schizophrenia and the role of cortical excitation-inhibition balance, in: Biological Psychiatry, 2022.

[17] Chao Huang, Andrey Resnik, Tansu Celikel, Bernhard Englitz, Adaptive spike threshold enables robust and temporally precise neuronal encoding, PLoS Comput. Biol. 12 (6) (2016) e1004984.

[18] Dmitry Ivanov, Aleksandr Chezhegov, Mikhail Kiselev, Andrey Grunin, Denis Larionov, Neuromorphic artificial intelligence systems, Front. Neurosci. 16 (2022) 1513.

[19] Yu Ji, YouHui Zhang, ShuangChen Li, Ping Chi, CiHang Jiang, Peng Qu, Yuan Xie, WenGuang Chen, Neutrams: neural network transformation and co-design under neuromorphic hardware constraints, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), IEEE, 2016, pp. 1–13.

[20] Dongshin Kim, Jang-Sik Lee, Neurotransmitter-induced excitatory and inhibitory functions in artificial synapses, Adv. Funct. Mater. 32 (21) (2022) 2200497.

[21] Alex Krizhevsky, Geoffrey Hinton, Learning Multiple Layers of Features from Tiny Images, CIFAR, 2009.

[22] Behnam Neyshabur, Russ R. Salakhutdinov, Nati Srebro, Path-sgd: path-normalized optimization in deep neural networks, Adv. Neural Inf. Process. Syst. 28 (2015).
[23] Behnam Neyshabur, Ryota Tomioka, Ruslan Salakhutdinov, Nathan Srebro, Data-dependent path normalization in neural networks, arXiv preprint arXiv:1511.06747, 2015.
[24] Behnam Neyshabur, Ryota Tomioka, Nathan Srebro, Norm-based capacity control in neural networks, in: Conference on Learning Theory, PMLR, 2015, pp. 1376–1401.
[25] J. Ott, E. Linstead, N. LaHaye, P. Baldi, Learning in the machine: to share or not to share, Neural Netw. 126 (2020) 235–249.
[26] Philipp Petersen, Felix Voigtlaender, Optimal approximation of piecewise smooth functions using deep relu neural networks, Neural Netw. 108 (2018) 296–330.
[27] J.L.R. Rubenstein, Michael M. Merzenich, Model of autism: increased ratio of excitation/inhibition in key neural systems, Genes Brain Behav. 2 (5) (2003) 255–267.
[28] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, Shih-Chii Liu, Conversion of continuous-valued deep networks to efficient event-driven networks for image classification, Front. Neurosci. 11 (2017) 294078.
[29] Lawrence K. Saul, Weight-balancing fixes and flows for deep learning, Trans. Mach. Learn. Res. (2023).
[30] Farshad Shirani, Hannah Choi, On the physiological and structural contributors to the dynamic balance of excitation and inhibition in local cortical networks, bioRxiv, 2023–01, 2023.
[31] Martino Sorbaro, Qian Liu, Massimo Bortone, Sadique Sheik, Optimizing the energy consumption of spiking neural networks for neuromorphic applications, Front. Neurosci. 14 (2020) 662.
[32] Christopher H. Stock, Sarah E. Harvey, Samuel A. Ocko, Surya Ganguli, Synaptic balancing: a biologically plausible local learning rule that provably increases neural network noise robustness without sacrificing task performance, PLoS Comput. Biol. 18 (9) (2022) e1010418.
[33] Roberta Tatti, Melissa S. Haley, Olivia K. Swanson, Tenzin Tselha, Arianna Maffei, Neurophysiology and regulation of the balance between excitation and inhibition in neocortical circuits, Biol. Psychiatry 81 (10) (2017) 821–831.
[34] A. Tavakoli, F. Agostinelli, P. Baldi, SPLASH: learnable activation functions for improving accuracy and adversarial robustness, Neural Netw. 140 (2021) 1–12, arXiv:2006.08947.
[35] Gina Turrigiano, Homeostatic synaptic plasticity: local and global mechanisms for stabilizing neuronal function, Cold Spring Harb. Perspect. Biol. 4 (1) (2012) a005736.
[36] Gina G. Turrigiano, The dialectic of Hebb and homeostasis, Philos. Trans. R. Soc. Ser. B, Biol. Sci. 372 (1715) (2017) 20160258.
[37] Carl Van Vreeswijk, Haim Sompolinsky, Chaos in neuronal networks with balanced excitatory and inhibitory activity, Science 274 (5293) (1996) 1724–1726.
[38] Liu Yang, Jifan Zhang, Joseph Shenouda, Dimitris Papailiopoulos, Kangwook Lee, Robert D. Nowak, A better way to decay: proximal gradient training algorithms for neural nets, in: OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop), 2022.
[39] Bo Zhao, Nima Dehmamy, Robin Walters, Rose Yu, Symmetry teleportation for accelerated optimization, Adv. Neural Inf. Process. Syst. 35 (2022) 16679–16690.