Kent Academic Repository

Constraints on Hebbian and STDP learned weights of a spiking neuron

Dominique Chu and Huy Le Nguyen

CEMS, School of Computing, University of Kent, CT2 7NF, Canterbury, UK

dfc@kent.ac.uk

December 15, 2020

Abstract

We analyse mathematically the constraints on weights resulting from Hebbian and STDP learning rules applied to a spiking neuron with weight normalisation. In the case of pure Hebbian learning, we find that the normalised weights equal the promotion probabilities of weights up to correction terms that depend on the learning rate and are usually small. A similar relation can be derived for STDP algorithms, where the normalised weight values reflect a difference between the promotion and demotion probabilities of the weight. These relations are practically useful in that they allow checking for convergence of Hebbian and STDP algorithms. Another application is novelty detection. We demonstrate this using the MNIST dataset.

1 Introduction

Hebbian learning [1] is a well established approach to train neural networks that is not based on gradient descend. Instead, weights are updated based on correlations between input and output. There are a number of features that make Hebbian learning an attractive method to learn in the context of artificial intelligence (AI). For one, it is a local rule and does not require the computation of gradients. This makes it resource efficient and suitable in situations where the computation of gradients is not practical [2]. Secondly, it is biologically plausible and as such can be used to understand principles of learning in the brain. Thirdly, and for applications in AI perhaps most importantly, Hebbian learning can be used as a part of supervised learning [3, 4] but more importantly it is a powerful method for unsupervised learning [5, 6]. For example, it is well known that under certain conditions the weight vectors of networks trained with Hebbian learning rules indicate the principal component vector of the training dataset [7, 8]. Given that the vast majority of all available data is unlabelled, this makes Hebbian learning an important part of the toolbox of AI.

While Hebbian learning approaches are not commonly used to train rate-coding neural networks, they have become an important approach for training spiking neurons [1]. These are a type of artificial neuron that retains a "memory" of its state over time. This internal state is often called the "membrane potential" and typically is a real valued variable that decays over time. Their memory makes spiking neurons naturally suitable for processing temporal input data, such as video or audio. It is now well established that a spiking neuron is a more powerful information processors than the perceptron [9, 10]. Another advantage of spiking architectures is that they are biologically plausible [11] and can be implemented in energy efficient special purpose hardware [12, 13, 14]. Implementations of spiking neurons including the tempotron [15] or the chronotron [16] and others [17, 18] demonstrate in practice the abilities of spiking neurons to learn to classify spatio-temporal patterns [3, 19] including multi-label classification [20, 21].

The focus of this article is to understand generic properties of weights that result from Hebbian training algorithms applied to a single continuous time spiking neuron. We shall probe various versions of Hebbian learning rules as well as the asymmetric *spike-timing dependent plasticity* (STDP) [22] rule, where a Hebbian update is combined with an anti-Hebbian weight decrease for those weights that are active immediately after an output spike was generated. We also investigate how different ways to normalise weights during Hebbian learning [23] impact on the allowed weights.

The most common way in the literature to model the weights resulting from various training regimes is to identify the eigenvalues of the weight update matrix [24]. The problem with this approach is that it assumes that the weight evolution is a deterministic process. This is clearly not true because the update dynamics of weights is normally stochastic due to the random input. An alternative is to use master equation approaches [24, 25, 26] to capture stochastic aspects, but this is rarely feasible. Approximations to the master equations, especially the Fokker-Planck equation are common, but still difficult to solve and in fact problematic [27].

Here, we take a different approach. Rather than attempting to derive the distribution of weights, we concentrate exclusively on the *metastable* and *absorbing states*, i.e. weight values that the neuron will have *after* a long time of training and *for* a long time. Our main result is a mathematical relation between the allowed metastable weight combinations that are consistent with Hebbian training and the probability of the weights to be increased/decreased during the Hebbian learning process. For a simple Hebbian learning rule we find that the probability of weight promotion equals the value of the normalised weight up to some correction terms that depend on the learning rate.

The significance of this result is that it makes it possible to determine practically whether or not a set of weights is adapted to the statistical bias of the input. This is relevant for the following reasons: Firstly, our results provide a novel characterisation of the weights that will evolve in the long run during Hebbian learning. This is interesting in itself, but will hopefully also open up novel pathways for the mathematical analysis of the behaviour of spiking neural networks trained using Hebbian learning. In particular, it may lead to novel insights about the statistical distribution of weights, which has been of substantial research interest [28, 29, 30, 31]. Secondly and more practically relevant, as we discuss below, our results can be used to adapt the learning rate of the neuron. In many contexts it is beneficial to have a large learning rate at the beginning and a small or vanishing learning rate when the neuron is close to what it should learn. The relationships we derive make it possible to estimate how far the neuron is away from ideal weights. The learning rate can be decreased accordingly. Finally, as we will show below, the relationships we derive can also be used for novelty detection.

2 Results

In this article, we will consider a number of variants of Hebbian learning. The basic idea of Hebbian learning is best summarised by the well known tenet "What fires together, wires together." In the context of an artificial neuron whose input channels are weighted this would mean that the activity of an input channel can lead to an increase of the weight of the input channel if it happens within a short temporal window before the neuron itself spikes. There is then a sense of the input "causing" the output spike of the neuron.

This basic Hebbian learning rule is asymmetric in the sense that only activities before the output spike matter, and the only option for weight changes is to increase the weights (although this is typically paired with some mechanism to prevent unlimited weight growth). This basic Hebbian scheme is frequently supplemented by a rule that decreases the weights of input channels firing *after* an output spike happened. This *asymmetric* rule is usually referred to as *spike timing dependent plasticity* (STDP) rule.

For both the symmetric and asymmetric rule there are many variants in the literature. For example, the amount of weight increase can depend on the precise distance from the output spike, i.e. the closer the input spike to the output spike the greater the decrease/increase of the corresponding weights. Furthermore, this dependence could be linear or non-linear.

Below, we will first concentrate on asymmetric (Hebbian) rules. We will model a number of variants of these rules, including different normalisation schemes. As a main result, we will find that in steady state the weights of a channel are directly related to the probability of the channel being promoted. We then extend this to the case of symmetric rules, where we find a similar albeit slightly more complex relationship between weights and promotion probabilities.

2.1 Model of the spiking neuron

Throughout this contribution we model the spiking neuron as having a time-dependent internal state $V \in \mathbb{R}^+$ — the "membrane potential." The neuron receives discrete input spikes in continuous time each arriving via one of N different channels; these are modelled as instantaneous spikes $\delta(t - T_k^i)$ arriving at channel i

and time T_k where $t, T_k \in \mathbb{R}^+$; $\delta(\cdot)$ denotes the Dirac delta function. Each input channel i is associated with a weight w_i, such that at spike time T_k^i the membrane potential V increases instantaneously by w_i. The membrane potential also decays (or "leaks") in time with a rate constant $d \geq 0$, such that the instantaneous rate of decay at time t is $V(t)d$. The neuron also has a spiking threshold. If V crosses the threshold from below, then (i) the membrane potential is reset to 0 (ii) the weights are adjusted according to one of the learning rules described below. Crossing of the threshold is normally also associated with the generation of an output spike. Since we only model a single neuron, output spikes will be of no further relevance here.

2.2 Relationship between promotion probability and weights

2.2.1 The long-term behaviour of the weight vector

The simplest way to derive the steady-state values of weights $w_1, w_2, \ldots, w_N$ of a neuron is to analyse the fixed points of the weight update rule $f(x_1, \ldots, x_N, w_1, \ldots, w_N, y, t) \overset{!}{=} 0$, which is a function of the inputs $x_1, \ldots, x_N$ to the neuron as well as its output y.

$$\Delta w_i = f(x_1, \ldots, x_N, w_1, \ldots, w_N, y, t) \tag{1}$$

This function f could, for example, be the famous *Oja rule*, i.e.

$$f(x_i, w_i, y) = x_i \cdot y - y^2 \cdot w_i$$

In steady state, updates would leave the weight vector unchanged, i.e. on average

$$x_i \cdot y = y^2 \cdot w_i \tag{2}$$

This update rule can be shown to lead to a fixed point where the weights of the neuron align with the principal component vector of the input data if the input data has a vanishing mean [24]. In general, Hebbian updates of the weight vector rules will not compute the principal component of the input function, but compute some other functions. In any case, for the steady state weight vector it will always be the case that $f(x_i, w_i, y) = 0$ for all i.

A slightly different perspective on this is possible. A Hebbian update rule that avoids unlimited growth of weight values requires an update rule that contains both weight increases according to some rule $f^+ = f^+(x_1, \ldots, x_N, w_1, \ldots, w_N, y, \epsilon)$ but also weight decreases $f^- = f^-(x_1, \ldots, x_N, w_1, \ldots, w_N, y, \epsilon)$, where $f^+, -f^- \geq 0$. Any particular update event is either positive (f^+) and leads to a weight promotion or negative (f^-) effecting a weight demotion. A quantity of interest is the average of the step size $s_i := \Delta w_i$ during an update event given by

$$\langle s_i \rangle := p_i f^+ + (1 - p_i) f^-, \tag{3}$$

where p_i is the (unknown) probability that the i-th weight is promoted. A necessary condition for weights to stabilise is $\langle s_i \rangle = 0$. Formally, we can now solve the right hand side of Eq. 3 for p_i to obtain a relationship between the promotion probability and the value of the weights.

We now consider the case of a simple Hebbian learning rule for a spiking neuron in continuous time with weight normalisation. The scenario we consider is as follows:

1. Whenever the neuron receives an input from channel i then the membrane potential is increased by w_i.

2. If the membrane potential exceeds a critical threshold value θ, then an output spike is generated and the membrane potential is reset to 0.

   (a) If the k-th input channel was active at least τ time units before an output spike was generated, then the k-th weight is increased by some fixed amount ϵ — the *learning rate*.

   (b) Following an update step weights are normalised by setting $w_i \to w_i / \left(\sum_j w_j^l \right)^{\frac{1}{l}}$, for some real number $l \geq 1$ and all $i = 1, \ldots, N$.

It is now straightforward to derive a relationship between w_i and the probability p_i that the i-th weight will be increased during a particular promotion event. The simplest case is $l = 1$, i.e. when the weights are always normalised to 1 following an update event. This case also coincides with Oja's rule. The average step size can then be written as follows:

$$\langle s_i \rangle = p_i \left(\frac{w_i + \epsilon}{1 + \epsilon} - w_i \right) - (1 - p_i) \left(w_i - \frac{w_i}{1 + \epsilon} \right). \tag{4}$$

The first term on the rhs of this equation simply accounts for the increase of weight i by ϵ followed by a weight normalisation which leads to a slight decrease again. The net effect is always an increase, which happens with probability p_i. The second term describes that during any given weight update event, it may also be the case that any of the other $j \neq i$ weights are increased instead. In this case the weight is normalised without being increased. This leads to a net decrease of the i-th weight. Note that the probabilities to be increased and decreased are typically a function of the weights themselves. Solving Eq. 4 for p_i, we obtain the condition:

$$p_i = w_i \tag{5}$$

This means that, after a sufficiently long training period the weights will be equal to the promotion probability. In the limiting case of $\tau \to 0$ the weight would be the probability that the corresponding channel fired last before an output spike was triggered by the neuron.

The same procedure, although with a bit more notational effort, can be applied to the case of $l > 1$. In this case the normalisation factor depends on which weight was promoted and Eq. 3 becomes:

$$\langle s_i \rangle = p_i \left(\frac{w_i + \epsilon}{\varsigma_i} - w_i \right) + \sum_{j \neq i} p_j \left(\frac{w_i}{\varsigma_j} - w_i \right), \tag{6}$$

where $\varsigma_i := (w_1^l + \cdots + w_{i-1}^l + (w_i + \epsilon)^l + w_{i+1}^l + \ldots + w_N^l)^{\frac{1}{l}}$ and p_j is the probability that the weight of the j-th channel is promoted.

From Eq. 6 we can now establish a relation between the promotion probability and the weights

$$p_i w_i \left(\frac{1}{\varsigma_i} - 1 \right) + p_i \frac{\epsilon}{\varsigma_i} + w_i \sum_{j \neq i} p_j \left(\frac{1}{\varsigma_j} - 1 \right) = 0$$

$$p_i \frac{\epsilon}{\varsigma_i} = w_i \left(1 - \sum_j p_j \frac{1}{\varsigma_j} \right)$$

$$\frac{p_i}{\varsigma_i} = \frac{w_i}{\epsilon} \left(1 - \left\langle \frac{1}{\varsigma} \right\rangle \right). \tag{7}$$

The rhs of this equation is still somewhat opaque in that the average over the inverse normalisation factors ς_i is dependent on w_i and p_i such that further analysis is necessary. In order to evaluate this average it is useful to write $\varsigma_i = (\mathcal{W} + (w_i + \epsilon)^l - w_i^l)^{\frac{1}{l}}$, where $\mathcal{W} := \sum_i w_i^l$. For a l-normalised weight vector we have $\mathcal{W} = 1$. Since ϵ is a small term, we can expand the expression to first order in ϵ, which yields:

$$\varsigma = \exp \left(\frac{\ln \left(\mathcal{W} + \exp \left(l \ln \left(w_i \right) \right) - w_i^l \right)}{l} \right) - \frac{\exp \left(\frac{l \ln(w_i) + \ln \left(\mathcal{W} + \exp(l \ln(w_i)) - w_i^l \right)}{l} \right)}{w_i \left(\mathcal{W} + \exp \left(l \ln \left(w_i \right) \right) - w_i^l \right)} \epsilon + O \left(\epsilon^2 \right)$$

$$\approx \frac{\mathcal{W}^{\frac{1+l}{l}} w + \exp \left(\frac{l^2 \ln(w_i) + \ln(\mathcal{W})}{l} \right) \epsilon}{\mathcal{W} w_i}$$

$$= \mathcal{W}^{\frac{1}{l}} + w_i^{l-1} \mathcal{W}^{\frac{1-l}{l}} \epsilon = 1 + w_i^{l-1} \epsilon \tag{8}$$

Next, we evaluate the average over the inverse normalisation term $\left\langle \frac{1}{\varsigma} \right\rangle$. First, we take note of the general relation

$$\frac{1}{A + x} = \frac{1}{A} - \frac{1}{A^2} x + O(x^2),$$

and expand the average to first order in the second term:

$$\left\langle \frac{1}{\varsigma} \right\rangle = \sum_i p_i \frac{1}{1 + w_i^{l-1}\epsilon}$$

$$\approx \sum_i p_i \left(1 - w_i^{l-1}\epsilon\right)$$

$$= 1 - \sum_i p_i w_i^{l-1}\epsilon$$

$$= 1 - \langle w^{l-1}\rangle\epsilon \tag{9}$$

Substituting this back into Eq. 7 and re-arranging the terms we find the first order expression of the promotion probability in terms of weight:

$$p_i = \frac{\langle w^{l-1}\rangle}{1 - w_i^{l-1}\epsilon} w_i \tag{10}$$

This expression is compact, but contains implicitly p_i on the right hand side. Taking advantage of the fact that the promotion probability is normalised, we can establish that the rhs of the equation is the normalised weight, thus obtaining

$$p_i = \frac{w_i}{\sum_j w_j} \qquad \sum_j w_j = \frac{\left(1 - \langle w_i^{l-1}\rangle\epsilon\right)}{\langle w^{l-1}\rangle}. \tag{11}$$

Note that in deriving this latter equation, we did not make any explicit assumptions about the length τ of the time window. This information is implicit in the p_i in that the meaning of p_i changes as τ is adjusted. While the length of the time window will clearly affect the weights that the system will find, it does not change the constraint that the normalised weights equal the promotion probability.

2.3 Symmetric STDP

In practical applications of spiking networks, the basic Hebbian learning rule is often augmented by an additional anti-Hebbian term. In STDP rules weights are increased when the corresponding input fires before an output spike is generated and they are decreased when the input fires shortly after an output spike. We assume, for the moment, that the change of the weights is $\pm\epsilon$ if the input channel fires within a period of $\pm\tau$ time units of an output spike.

This case can be modelled largely along the same lines as above, but some additional notational complexities become necessary. The expression for the average step size $\langle s_i\rangle$ of the weight w_i needs to be extended to take into account the anti-Hebbian component. For simplicity, we assume that the Hebbian increases/decreases are by $\pm\epsilon$, although the results to follow can be adjusted easily for non-equal changes.

In analogy to equation 6, we can write the average step size:

$$\langle s_i\rangle = k^{\mathrm{p}} \left[w_i p_i \left(\frac{1}{\varsigma_i^{\mathrm{p}}} - 1\right) + p_i \frac{\epsilon}{\varsigma_i^{\mathrm{p}}} + w_i \sum_{j \neq i} p_j \left(\frac{1}{\varsigma_j^{\mathrm{p}}} - 1\right) \right]$$

$$+ k^{\mathrm{d}} \left[w_i q_i \left(\frac{1}{\varsigma_i^{\mathrm{d}}} - 1\right) - q_i \frac{\epsilon}{\varsigma_i^{\mathrm{d}}} + w_i \sum_{j \neq i} q_j \left(\frac{1}{\varsigma_j^{\mathrm{d}}} - 1\right) \right] \tag{12}$$

Here the first term in the square brackets is identical to Eq. 6 and formulates the contribution to step sizes as they arise from the weight promotion events. The unconditional probability of a weight promotion is k^{p} and the conditional probability p_i that there is a promotion event and that the i-th channel is promoted. The second term takes into account weight demotions which happen with the unconditional probability k^{d}; q_j denotes the conditional probability that given a weight is demoted it is that of the j-th input channel.

A few clarifying remarks are appropriate before continuing. Firstly, the definition of $k^{\mathrm{p}}, k^{\mathrm{d}}$ implies the normalisation condition $k^{\mathrm{p}} + k^{\mathrm{d}} = 1$. Secondly, q_i and p_i are conditional probabilities, but for notational

simplicity, we will henceforth refer to them as probabilities throughout the rest of this article. Finally, note that in Eq. 12 it is necessary to distinguish between 2 different normalisation terms, that is ς_i^{p}, which is the same as above and $\varsigma_i^{\mathrm{d}} := (w_1^l + w_2^l + \ldots + w_{i-1}^l + (w_i - \epsilon)^l + w_{i+1}^l + \ldots)^{\frac{1}{l}}$, which arises from the demotion of weights. Note also that in Eq. 12 the minus sign in front of the second term in the second line comes from the fact that in the case of a demotion the weight change equals $(w_i - \epsilon)/\varsigma_j^{\mathrm{d}} - w_i$.

To proceed we manipulate Eq. 12 in analogy to the Hebbian case to find an expression for the average step size.

$$\langle s_i \rangle = k^{\mathrm{p}} w_i \left(\left\langle \frac{1}{\varsigma} \right\rangle_{\mathrm{p}} - 1 \right) + k^{\mathrm{d}} w_i \left(\left\langle \frac{1}{\varsigma} \right\rangle_{\mathrm{d}} - 1 \right) + k^{\mathrm{p}} p_i \frac{\epsilon}{\varsigma_i^{\mathrm{p}}} - k^{\mathrm{d}} q_i \frac{\epsilon}{\varsigma_j^{\mathrm{d}}} \tag{13}$$

Here, $\langle \cdot \rangle_{\mathrm{d}}$ denotes the average over all demotion events and q_i is the probability that channel i is demoted. In steady state, we then obtain the following relation for the promotion probability:

$$\frac{p_i}{\varsigma_i^{\mathrm{p}}} - \frac{k^{\mathrm{d}}}{k^{\mathrm{p}}} \frac{q_i}{\varsigma_i^{\mathrm{d}}} = -w_i \left(\frac{k^{\mathrm{p}} \left(\left\langle \frac{1}{\varsigma} \right\rangle_{\mathrm{p}} - 1 \right) - k^{\mathrm{d}} \left(\left\langle \frac{1}{\varsigma} \right\rangle_{\mathrm{d}} - 1 \right)}{k^{\mathrm{p}} \epsilon} \right) \tag{14}$$

We can now substitute in the first order approximations for the normalisation terms Eqs. 8 and 9. Note that these are different for the down and up probabilities in that the sign of the first order correction is swapped. This gives us:

$$\frac{k^{\mathrm{p}}}{\varsigma_i^{\mathrm{p}}} p_i - \frac{k^{\mathrm{d}}}{\varsigma_i^{\mathrm{d}}} q_i = w_i \left(k^{\mathrm{p}} \left\langle w^{l-1} \right\rangle_{\mathrm{p}} - k^{\mathrm{d}} \left\langle w^{l-1} \right\rangle_{\mathrm{d}} \right) \tag{15}$$

Here, the learning rate has apparently fallen out of the equation. It will be re-introduced by substituting in the approximation of the normalisation factors, which now become first order correction terms. Using once again Eq. 8, we obtain

$$k^{\mathrm{p}} p_i (1 - w_i^{l-1} \epsilon) - k^{\mathrm{d}} q_i (1 + w_i^{l-1} \epsilon) = w_i \left(k^{\mathrm{p}} \left\langle w^{l-1} \right\rangle_{\mathrm{p}} - k^{\mathrm{d}} \left\langle w^{l-1} \right\rangle_{\mathrm{d}} \right). \tag{16}$$

As before, we calculate the averages in the second term on the rhs by exploiting that both probabilities must sum to 1.

$$\sum_j w_j = \frac{k^{\mathrm{p}} (1 - \langle w^{l-1} \rangle_{\mathrm{p}} \epsilon) - k^{\mathrm{d}} (1 + \langle w^{l-1} \rangle_{\mathrm{d}} \epsilon)}{k^{\mathrm{p}} \left\langle w^{l-1} \right\rangle_{\mathrm{p}} - k^{\mathrm{d}} \left\langle w^{l-1} \right\rangle_{\mathrm{d}}}. \tag{17}$$

With this, we obtain a relationship between the promotion/demotion probabilities and the weights.

$$\frac{k^{\mathrm{p}} p_i (1 - w_i^{l-1} \epsilon) - k^{\mathrm{d}} q_i (1 + w_i^{l-1} \epsilon)}{k^{\mathrm{p}} (1 - \langle w^{l-1} \rangle_{\mathrm{p}}) - k^{\mathrm{d}} (1 + \langle w^{l-1} \rangle_{\mathrm{d}})} = \frac{w_i}{\sum_j w_j} \tag{18}$$

This tells us that up to corrections in the learning rate the normalised weights equal a difference between the *unconditional* promotion probability k^{p} and demotion probability k^{d}. In the case of $k^{\mathrm{p}} = 1$ and $k^{\mathrm{d}} = 0$ this reduces to the relation we found for the pure Hebbian learning rule Eq. 10, as it should.

In practical applications of STDP the update step is often chosen to be non-constant. A common model choice is to increase/decrease the weights that fire immediately before/after the output spike most, and much less those input channels that were active after the output spike. The update rule is normally determined by some function $\epsilon(\tau)$ for example a truncated exponential function to determine the weight increases/decreases.

In order to model this most general case a straightforward, if notationally tedious extension of Eq. 12 is necessary. The main change is that the step size now becomes a function of time since the last spike, $\epsilon = \epsilon(\tau)$ and $-\tau \le \tau \le \tau$. Eq. 12 needs to be adjusted accordingly by integrating over all possible spike times τ, such that, for example, the first term becomes,

$$k^{\mathrm{p}} \left[w_i \sum_{j \in \pi_i} \int_{-\tau}^0 p_j f_j(\tau) \left(\frac{1}{\varsigma_j^{\mathrm{p}}(\tau)} - 1 \right) d\tau \right]. \tag{19}$$
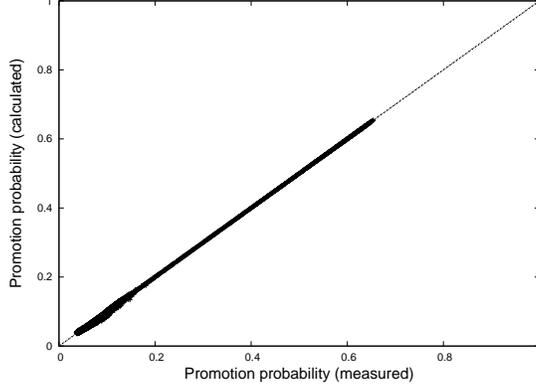
Figure 1: Comparing p_i, calculated using Eq. 14 with the estimated value from the simulation. This is an example with 10 input channels, $\epsilon = 0.0001$ and a $\tau = 0.07$. Each input channel fires randomly with a rate of 0.9. The dashed line indicates the diagonal.

Here f_j^{p} is a probability density defining the spike time distributions of input channel j before an output spike is generated. The normalisation condition on the probabilities holds, i.e.

$$\sum_j \int_{-\tau}^0 p_j f_j^{\mathrm{p}}(\tau) d\tau = 1.$$

If we now define an average learning rate specific to the index i as $\langle \epsilon \rangle_{\mathrm{d},i} := \int_{-\tau}^0 f_i^{\mathrm{p}}(\tau)\epsilon(\tau)d\tau$ then we can perform the analogous analysis to arrive at the same expression as Eq. 14 to relate the weights and the promotion probability.

$$p_i = q_i \frac{k_{\mathrm{d}}}{k_{\mathrm{d}}} \frac{\langle \epsilon \rangle_{\mathrm{d},i} - \langle \epsilon^2 \rangle_{\mathrm{d},i} w^{l-1}}{\langle \epsilon \rangle_{\mathrm{p},i} - \langle \epsilon^2 \rangle_{\mathrm{p},i} w^{l-1}} + w_i \frac{k_{\mathrm{p}} \langle w^{l-1}\epsilon \rangle_{\mathrm{p}} - k_{\mathrm{d}} \langle w^{l-1}\epsilon \rangle_{\mathrm{d}}}{k_{\mathrm{p}}(\langle \epsilon \rangle_{\mathrm{p},i} - \langle \epsilon^2 \rangle_{\mathrm{p},i} w^{l-1})} \tag{20}$$

Here the order of terms is somewhat obscured by the fact that average learning rates appear in all terms. Note, that the terms that are linear in $\langle \epsilon \rangle$ are actually of order zero, whereas the quadratic terms are of first order. Keeping this in mind it is straightforward to see that the case of $\epsilon(\tau) = $ constant reduces to equation 18. Finally, in the case of pure Hebbian learning $k_{\mathrm{d}} = 0$ and $k_{\mathrm{p}} = 0$, we find again that the normalised weights equal the promotion probability, yielding in zero-th order:

$$p_i = \frac{\langle w^{l-1}\epsilon \rangle_{\mathrm{p}}}{\langle \epsilon \rangle_{\mathrm{p},i}} w_i. \tag{21}$$

2.3.1 Decay model

Weight normalisation is not the only way to maintain finite weights in Hebbian learning rules. Another method is weight decay. The idea is that in addition to weight increases due to the Hebbian learning rule, weights decrease (or "decay") continuously according to some rule. There are numerous possible ways in which weight decay could be implemented. Most of them are much simpler to analyse than the weight normalisation. Here, we only hint at possible results by way of a specific model. Concretely we assume a setting with asymmetric STDP with some window length τ and a constant ϵ. During each promotion event all weights are decreased by a constant fraction δ of the weight value and then increased by a fixed amount of ϵ. The average step size is then much easier to write down than in the normalisation case.

$$\langle s_i \rangle = p_i(\epsilon - \delta w_i) - (1 - p_i)\epsilon \tag{22}$$

It is trivial to find an exact steady state solution to this.

$$w_i = \frac{\epsilon}{\delta} p_i \tag{23}$$

Hence, again, in the case of Hebbian learning combined with weight decay, the normalised weights equal the promotion probability and the weight normalisation factor is given by $\sum_j w_j = \frac{\epsilon}{\delta}$.

7

2.4 Some interesting examples

We now illustrate the above results using numerical examples of Hebbian trained neurons. For most examples below, we shall use unbiased random input data, but we also confirm the constraints on biased input data. We also demonstrate novelty detection using the MNIST data. All experiments below were conducted on an Intel i7-6700 CPU with 32 GB RAM.

2.4.1 2 weights only

As the first example, we probe the simplest possible case of a neuron with two input channels, $l = 1$ and trained with an asymmetric Hebbian learning rule. While of no clear practical use, this case is interesting because it is possible to calculate the promotion probability without simulating the neuron. To do this, we enumerated all possible input sequences of length 13 (which was sufficiently long to capture all cases). These cases are:

1. All 13 firing events are from input channel 1.

2. The first 12 firing events are from input channel 1, but the last one from input channel 2.

3. The first 11 firing events are from input channel 1, the 12th from input channel 2, the 13th from input channel 1

4. etc...

Having generated all possible input sequences, we then generated waiting times between all input spikes by sampling from an exponential distribution with appropriately chosen rate parameter. For each set of sequences thus generated, we then determined the point of the sequences where the membrane potential first exceeds our (arbitrarily) chosen threshold of 0.94 and truncated the sequence at this point. This left us with the set of all possible paths to crossing the threshold.

Given all truncated sequences, we then counted the number of instances where the sequence was truncated at weight 1, i.e. where the output spike was triggered by the first channel. Then we divided this number by the total number of sequences to obtain p_1. This probability depends on the weights. For example, trivially $p_1 = 0$ if $w_1 = 0$ and similarly, it must be $p_1 = 1$ for $w_1 = 1$. In-between these two extreme values p_1 can only be calculated by exhaustive enumeration of all possible sequences; see Fig. 2a for a numerical example of p_1 vs w_1. Note that due to the method we use, the results we generated depend on the random seed, in that repetitions will give slightly different inter-spike intervals each time. However, we found the variation between seeds to be negligible.

A consequence of Eq. 5 is that the weights of a trained neuron are limited to the values where p_i intersects the diagonal in Fig. 2a. To check this, we initialised a neuron with many different values of w_1 and $w_2 = 1 - w_1$ and trained it using Oja's rule using an input spiking rate of 0.9 for both channels and a threshold of $\theta = 0.94$. We then recorded the weight of input channel 1 after 300000 time units as a function of the initial weight; see Fig. 2b. We restricted the analysis to $w_1 \geq 0.5$. The rest of the weights space can be inferred by symmetry.

In the case of no decay of the membrane potential, i.e. $d = 0$, Fig. 2a suggests that there are only 4 weight values that meet the constraint $p_1 = w_1$. Out of those 4 values, however, we observed only 3 in the Hebbian trained neuron; see Fig. 2b: (i) For low initial values the weights remain symmetrically distributed with an average of $1/2$ for each of the channels. (ii) The second stable value is ≈ 0.63. (iii) Very high initial weights lead to a convergence to $w_1 = 1$. The fourth point with a vanishing step size, at around $w_1 \approx 0.68$, is not taken by the system.

The standard interpretation of this is that $w_1 \approx 0.68$ is an "unstable fixed point." Indeed, a dynamic instability in Fig. 2a exists if (when read from from left to right) p_1 crosses the diagonal from below. In this case, the weights will be driven away from the fixed point. On the other hand, if p_1 crosses the diagonal from above, then the weights will be driven towards the stable state. Following this reasoning, it is understandable why the system omits one of the possible steady state weight values. However, there is more to this.

If we increase the decay rate of the membrane potential, i.e. $d > 0$, then we find that the first weight either takes the value of 1, i.e. captures all the weight, or it takes a smaller decay dependent value; see Fig.
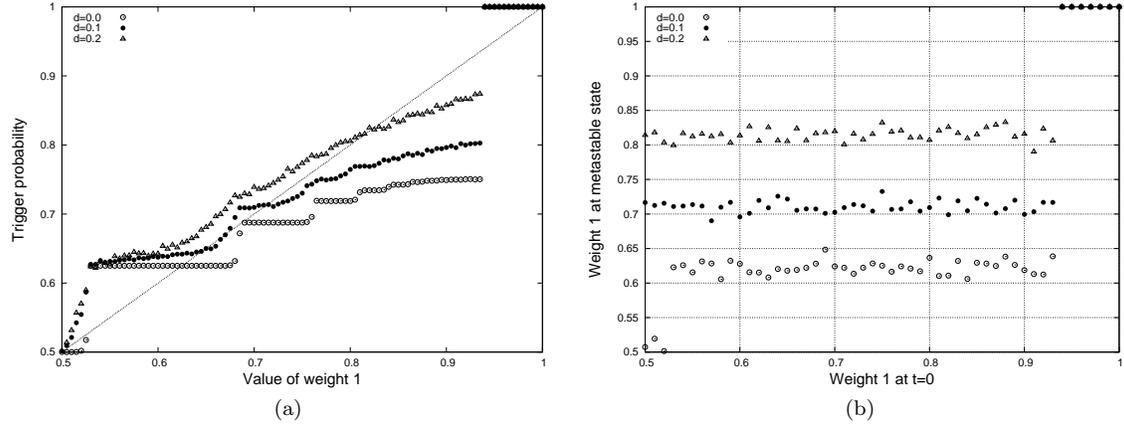
Figure 2: (a) Assuming a neuron with 2 input weights and a threshold set to 0.94 and $l = 1$. The figure shows p_1 as a function of w_1 calculated explicitly as described in main text for three different decay rates of membrane potential ($d = \{0.0, 0.1, 0.2\}$). (b) Weights of the neuron after Hebbian training for 300000 time units assuming a step size of 0.0005. Each point corresponds to a single simulation/calculation in both figures. The p_i values are limited to p_i values where the corresponding graph in (a) crosses the diagonal.
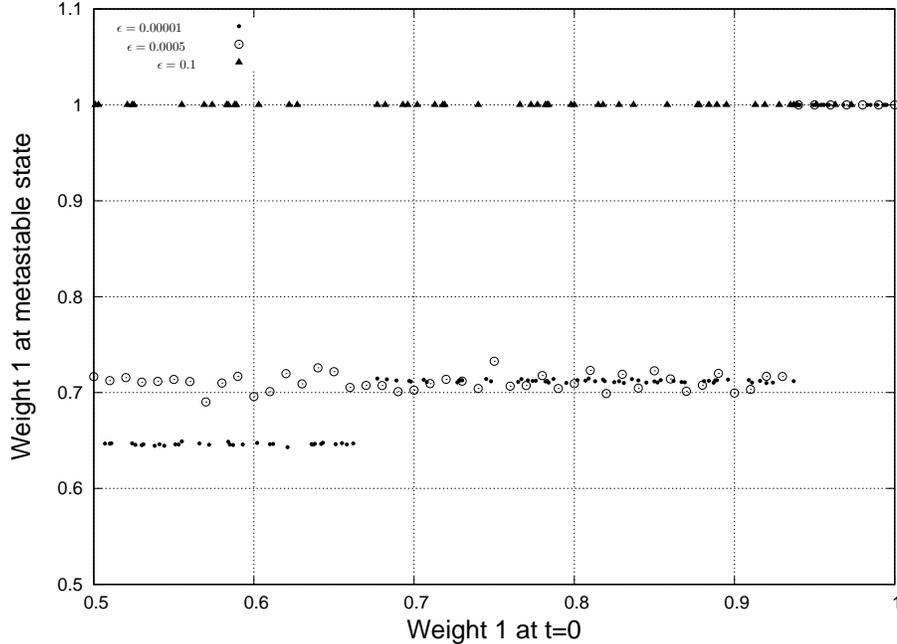

Figure 3: As Fig. 2b, but now showing the case of $d = 0.1$ only for 3 different step sizes. The intermediate step size is the same data as in Fig. 2b. The lowest step size $\epsilon = 0.00001$ occupies and additional metastable state. The largest step size (implemented as a uniformly distributed, random number smaller than 0.1) reaches the absorbing state from all initial weights.

2b. Fig. 2a suggests that there are two more points with vanishing step size: (i) an unstable point, (ii) a stable point (≈ 0.625) (which is not taken).

In order to understand this, some subtleties need to be taken into account in the interpretation of the results. The evolution of weights over time is fundamentally a random walk in weight space, rather than a deterministic dynamical system. In a stochastic system, the stable points are not actually stable, in the sense of dynamical systems theory, but rather they are points that can capture the dynamics *for some time*. These points are thus better described as *metastable states*. In principle escape from these metastable states is always possible. The escape rate depends on the step size (i.e. the learning rate) and the size of the basin of attraction. Note that in addition to the metastable states, the current setup also has absorbing-states from which no escape is possible. These are $w_1 = 1$ and $w_1 = 0$, respectively.

We can therefore expect the system to occupy different meta-stable states for different amounts of time depending on the size of the basin of attraction of the metastable point and the step size chosen. Larger step sizes lead to smaller dwell times of the system in the neighbourhood of metastable states.

This is illustrated in Fig. 3 which shows the metastable states as a function of the starting values for 3 different step sizes and a decay rate of $d = 0.1$. Of the three considered step sizes, only the smallest one occupies the metastable state at ≈ 0.625; in contrast, when we chose a very large step-size, then the system always converged to the absorbing state. In-between these two extreme cases, the Hebbian learning drove the weights to two different metastable states, depending on the initial condition. The learning rate is thus not merely a parameter that determines how fast metastable states are approached, but also which ones.

2.4.2 Biased inputs

In the previous examples, the input spikes of the two channels are assumed to be independent, identically distributed. Our derivations did not use this assumption, hence the results should not depend on it. To check this, we biased inputs to channel 1 relative to channel 2, while keeping the combined frequency of firing constant. For example, a bias of 0.6 means that 60% of input signals come via channel 1.

Fig. 4a shows p_1 vs w_1 for various biases. It compares simulations where Hebbian learning is switched on with a simulation where weights are initialised randomly and then kept constant. The piecewise constant shape of the evolved weights vs the starting weights confirms that the trained values take only 1 of two possible values (up to some noise). Furthermore, if the prediction of Eq. 5 is correct, then the trained weights should take values where the promotion probability equals the normalised weights. This means that the trained and untrained curves should intersect at the diagonal, which they do in Fig. 4a. Thus, as expected the probabilities p_i depend on the input data, but the constraint of Eq. 6 has to be fulfilled regardless.

2.4.3 More than 2 weights

Next, we consider an example of a neuron with 10 inputs and non-iid inputs. In this case, it is no longer practical to calculate the promotion probabilities exactly. Instead, we need to determine it from a full simulation of the neuron. The example simulation in Fig. 4b shows that the value of the evolved weights remains bound to those values where the promotion probability is equal to the weights.

2.4.4 Novelty detection

Eq. 18 makes it possible to estimate how far the weights of the neuron are from their steady state. We reproduce this equation here:

$$\underbrace{\frac{k^{\mathrm{p}}p_i(1 - w_i^{l-1}\epsilon) - k^{\mathrm{d}}q_i(1 + w_i^{l-1}\epsilon)}{k^{\mathrm{p}}(1 - \langle w^{l-1}\rangle_{\mathrm{p}}) - k^{\mathrm{d}}(1 + \langle w^{l-1}\rangle_{\mathrm{d}})}}_{=:L_i} = \frac{w_i}{\sum_j w_j} \tag{18}$$

From this we can now define the indicator

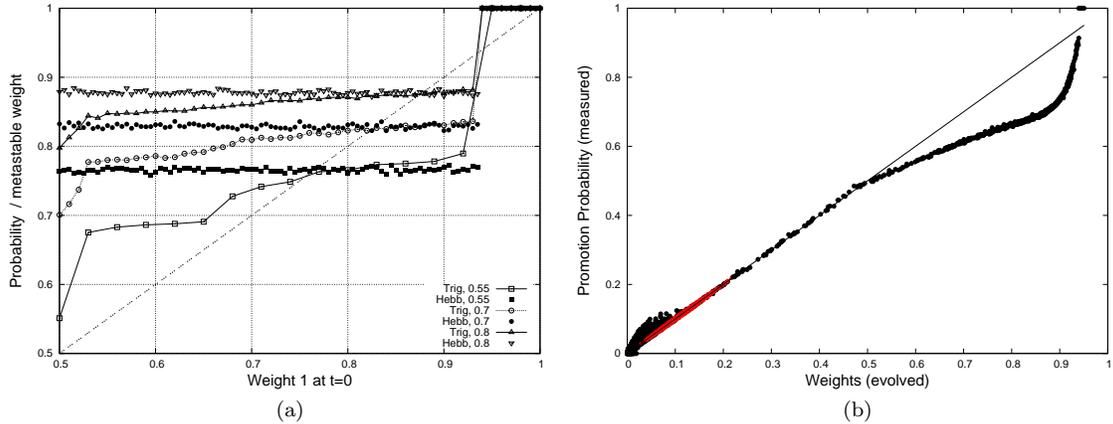$$\Delta := \sum_i \left| L_i - \frac{w_i}{\sum_j w_j} \right|.$$

10

Figure 4: (a) Example simulations for the case of $d = 0.1$ for three different biases again from a neuron with 2 input channels. Here, the probability that input 1 fires is $0.55, 0.7$ and 0.8 respectively. The points labelled "Trig" show the probability that the corresponding weight triggers an output spike calculated exactly. The points labelled "Hebb" show the final weight reached by Hebbian training as a function of the initial weight. (b) The black dots show the relationship between p_i and the weight for a model with 10 input channels no modification of weights. The red points represent weights evolved using Hebbian learning with $l = 1, \tau = 0.15$. The frequency of input channels is fixed but randomly chosen from a Gaussian distribution with a standard deviation of 0.15 and a mean of 0.9, i.e. all ten input channels had slightly different frequencies.

Note that this indicator cannot be negative. In the ideal case of the neuron being perfectly adapted, we would expect from Eq. 18 that the indicator $\Delta = 0$. For finite learning rates, however, the weights will tend to be perturbed away from the steady state, such that in reality $\Delta \geq 0$. It is possible to monitor Δ continuously during the operation of the neuron, which in turn makes it possible to estimate how far the weights are from a steady state. In steady state the value of Δ will fluctuate around some small, but positive value. Away from steady state, Eq. 18 no longer holds; Δ will then deviate substantially from 0. It is thus possible to use the value of Δ to check for changes in the input statistics of a neuron. This turns the spiking neuron into a novelty detector.

To demonstrate this, we initialised a neuron with 10 input channels to random weights. Then we provided input with firing rates that are randomly distributed around 0.9, as in Fig. 1. We let the neuron adjust its weights for 333333 time units, at which point we draw new random frequencies for all the channels. At time 666666 we do the same again. Throughout the simulation, we monitor the value of Δ. Fig. 5a shows two trajectories of Δ, which differ only by the random seed. As expected, following initialisation Δ descends quickly to a value close to 0, where it remains untill the input statistics changes (indicated by grey bars in Fig. 5a); subsequent to these changes of inputs there is a substantial, but transient, deviation of Δ from its steady state value. We conclude, that the indicator Δ can be used for novelty detection.

Next, we tested novelty detection on the MNIST dataset of hand-written digits [32] to ensure that it works on real-world data as well. The images of the MNIST datasets have 28 pixels each with a greyscale value. This format is not immediately suitable for processing by a spiking neuron. We therefore pre-processed the images from the MNIST as follows: Firstly, rather than using the entire image (consisting of 784 pixels), we limited ourselves to the horizontal line 14 of the image. This is merely to speed up the computation. With this, we are then left for each image with a vector of intensities $I := [I_i, I_2, \ldots, I_{28}]$ representing the intensities of each pixel in the line. We convert this vector into a single spike as follows:

1. Compute the normalised intensity vector, $\bar{I} = \frac{I}{\sum_{i=1}^{28} I_i}$.

2. Draw a single random index j of $\bar{I}$ with probability $p(j) = \bar{I}_j$.
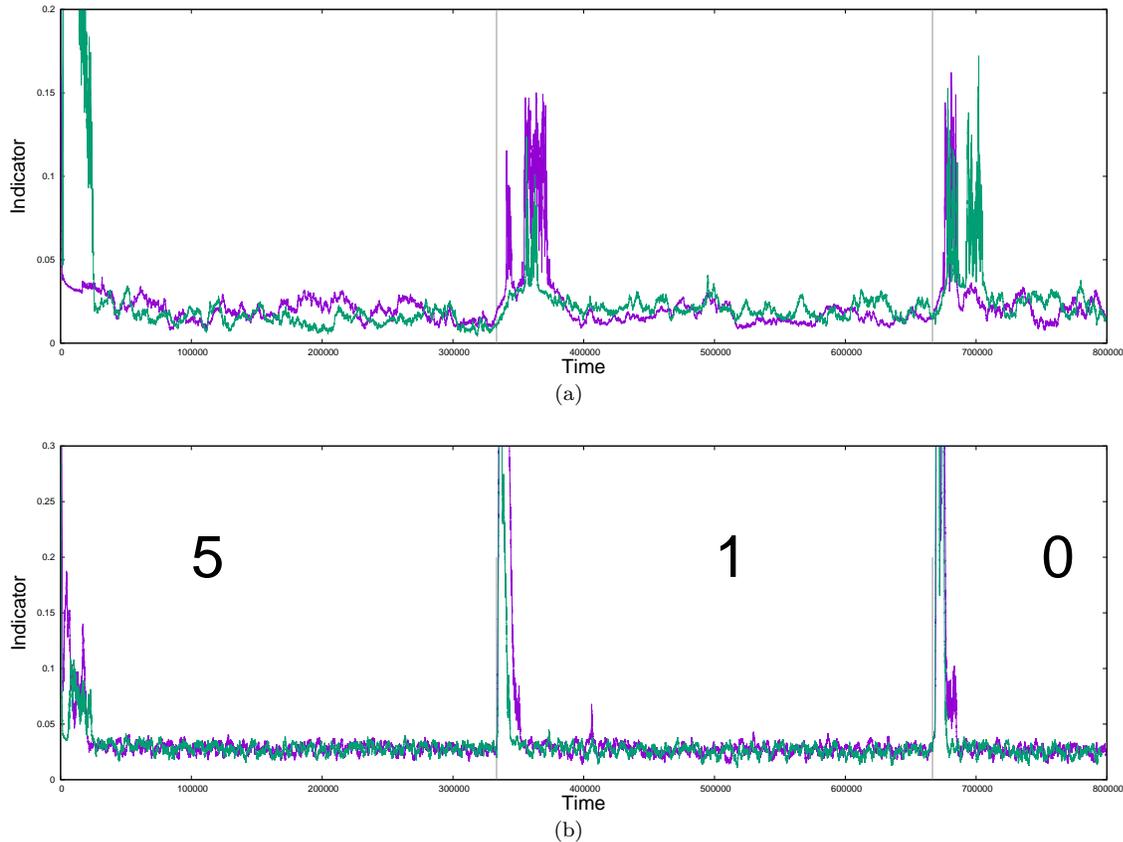
3. Provide an input spike to channel j of the neuron.

11

(a)

(b)

Figure 5: Distance from steady state as measured by Δ.(a) The neuron has 10 input channels and is initialised with random weights. Each input channel fires randomly with a frequency drawn from a Gaussian distribution with mean $\mu = 0.9, \sigma = 1$. At times 333333 and 666666 (indicated by the grey bars) the input frequencies are re-drawn. The graph shows two different simulations. At the time of the initialisation Eq. 18 does not hold, but after some time the weights reach steady state and the difference approaches 0. Clearly, the change of input statistics results in a transient spike, indicating a large deviation from Eq. 18. To prevent random deviations of the neuron from its steady state value, we made the learning rate dependent on the value of Δ via $\epsilon = \min(0.001, \exp(-1/4\Delta))$. (b) Same, but now data from the MNIST data-set. We show the neuron images of the digits, 1, 5 and 0. See main text for detailed explanation.

The entire procedure we followed is as follows: We initialised a neuron with 28 input channels. We then generated firing times such that the neuron as a whole receives input with a frequency of 0.9×28. For each scheduled input spike we then chose a randomly drawn image of the MNIST dataset; based on this image, we then chose an input channel as outline in the previous paragraph; a spike is then provided into this channel at the given time.

In the experiments we performed, we started by presenting only images drawn from the set of images labelled as digit 5. After 333333 time units, we then showed images of the digit 1, followed by the digit 0 after time 666666. Each change of image was marked by a spike of the value of the indicator Δ; see Fig. 5b. This demonstrates that the neuron can act as a novelty detector for classes of images.

3   Discussion

In this article we derived constraints on the weights of a continuous time spiking neuron, trained using Hebbian learning. Perhaps the most surprising conclusion we found is apparent from Eq. 11. Irrespective of the normalisation rule one uses, the weights resulting from Hebbian learning rules are constrained with respect to the normalised weights, that is weight normalisation to 1. The actual exponent used for the normalisation, in contrast, appears only as a higher order correction.

In the case of pure Hebbian learning we find that (up to corrections of first order) the normalised weights equal the promotion probability. The same will not be generically true for any set of weights, such that this relation can be used to characterise weights evolved by Hebbian learning.

In the case of asymmetric STDP the constraints on the weights become more complicated. In this case, the evolved weights have to be in a relationship with both the demotion and the promotion probabilities. This is best seen by Eq. 18, which is reproduced here in the zero-order version for clarity:

$$\frac{w_i}{\sum_j w_j} = \frac{k_{\mathrm{p}} p_i - k_{\mathrm{d}} q_i}{k_{\mathrm{p}} - k_{\mathrm{d}}} \tag{24}$$

This tells us that the normalised weights equal the difference between the *unconditional* promotion probability and the *unconditional* demotion probability divided by the difference between the total promotion and demotion probability for all weights. Note that the normalisation exponent does not appear in the zero order approximation that we reproduced here. This means that in lowest order the weights have to fulfil the same conditions irrespective of the exponent used for normalisation. Given that the values of weights will be smaller or equal to 1, the correction terms will be bounded by $1 \pm \epsilon$. To illustrate this, when using a learning rate of 0.0001 then the correction to the second term is at most 1.0001 and at least 0.999, but could be much closer to 1. We hasten to stress that our results do not imply that there is no difference in the dynamics of weight evolution between various normalisation schemes. Yet, when it comes to the constraints that are fulfilled by a metastable set of weight the exponent does not matter.

The relationships we derived are surprisingly simple and valid across a wide range of assumptions. This makes them intrinsically interesting, while also providing some novel insights into the kinds of weights one obtains from Hebbian learning. Apart from this, they are also useful practically. Since relations such as Eq. 18 are only valid in steady state, they can be used to monitor the distance of the weights from steady state. An immediate application of this is novelty detection, as we demonstrated above. Another possible use of this is to make the learning rate dependent on the distance from the steady state. In this way, it is possible, for example, to freeze the behaviour of the neuron as it approaches its steady state.

Our results also provide an interesting perspective on the step size ϵ, which acts as the learning rate but doubles up as a correction parameter in approximations. The normal intuition in those situations is that, as the correction parameter is decreased, the system becomes more and more similar to a specific zero-order solution.

This is the case here as well. As far as the corrections are concerned the step size only leads to a small correction to the zero-order equation, as discussed above. There is, however, an additional subtlety. At the same time as the learning rate is only a correction term to the constraints, it may have a major impact on the solutions that can be found within reasonable times. For the case of a minimal neuron with only two input channels this effect is illustrated above in Fig. 3. The insight remains valid for neurons of any number of input channels. Small learning rates will tend to get stuck in the first possible combination of solutions

that meet the steady state conditions, whereas for larger learning rates the neuron will settle on metastable states with larger basins of attraction. In this context, it is also interesting to recall that in search-based optimisation, the learning rate is thought to impact the speed of convergence of the algorithm as well as the accuracy with which any optimal value can be approached. In contrast, here we found that the learning rate can impact on the type of meta-stable behaviour that the system occupies.

In this paper we considered mostly the case of randomly firing input channels. This allowed us to illustrate the behaviour of the learning rules in absence of any effects that are specific to particular datasets or choices on the input statistics. The equations we used do not contain any explicit assumptions about the nature of the input frequencies, in particular they do not rely on the fact that we chose random/unbiased input. The constraints we derived, therefore, remain valid, irrespective of input data; see also Fig. 4b for an example. However, note that the conditional promotion and demotion probabilities are a function of the weights and a function of the input data. Therefore, if the input data changes, then the conditional probabilities change.

The results presented here are exclusively about a single neuron. It will be interesting to extend this to networks of neurons. Clearly, then the constraints have to be true for each individual component of the network. We leave it to future work to explore the consequences of this for the possible weight configurations of a spiking neural network.

References

[1] R. Kempter, W. Gerstner, and J. van Hemmen. Hebbian learning and spiking neurons. *Physical Review E*, 59:4498–4514, Apr 1999.

[2] C. Lammie, T. Hamilton, A. van Schaik, and M. Azghadi. Efficient FPGA implementations of pair and triplet-based STDP for neuromorphic architectures. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(4):1558–1570, April 2019.

[3] Y. Hao, X. Huang, M. Dong, and B. Xu. A biologically plausible supervised learning method for spiking neural networks using the symmetric STDP rule. *Neural Networks*, 121:387–395, January 2020.

[4] P. Falez, P. Tirilly, I. Bilasco, P. Devienne, and P. Boulet. Multi-layered spiking neural network with target timestamp threshold adaptation and stdp, 2019.

[5] J. Scott, T. Hayes, A. Ozcan, and W. Wilcke. Synaptic plasticity in an artificial hebbian network exhibiting continuous, unsupervised, rapid learning. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop on - NICE 19*. ACM Press, 2019.

[6] J. Lo. Unsupervised hebbian learning by recurrent multilayer neural networks for temporal hierarchical pattern recognition. In *2010 44th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, March 2010.

[7] X. Kong, C. Hu, and Z. Duan. *Principal Component Analysis Networks and Algorithms*. Springer Singapore, 2017.

[8] E. Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, November 1982.

[9] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[10] R. Rubin, R. Monasson, and H. Sompolinsky. Theory of spike timing-based neural classifiers. *Physical Review Letters*, 105(21):1–4, 2010.

[11] D. Depannemaecker, L. Santos, A. Rodrigues, C. Scorza, F. Scorza, and A. Guimarães de Almeida. Realistic spiking neural network: Non-synaptic mechanisms improve convergence in cell assembly. *Neural Networks*, 122:420–433, February 2020.

[12] C. Lin, A. Wild, G. Chinya, T. Lin, M. Davies, and H. Wang. Mapping spiking neural networks onto a manycore neuromorphic architecture. *SIGPLAN Not.*, 53(4):78–89, June 2018.

[13] L. Plana, D. Clark, S. Davidson, S. Furber, J. Garside, E. Painkras, J. Pepper, S. Temple, and J. Bainbridge. Spinnaker: Design and implementation of a gals multicore system-on-chip. *J. Emerg. Technol. Comput. Syst.*, 7(4):17:1–17:18, December 2011.

[14] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, 38(1):82–99, 2018.

[15] R. Gütig and H. Sompolinsky. The tempotron: a neuron that learns spike timing-based decisions. *Nature Neuroscience*, 9(3):420–428, 2006.

[16] Răzvan V. Florian. The chronotron: A neuron that learns to fire temporally precise spike patterns. *PLoS ONE*, 7(8), 2012.

[17] Filip Ponulak. ReSuMe-new supervised learning method for Spiking Neural Networks. *Inst. Control Information Engineering, Poznan Univ.*, 22(2):467–510, 2005.

[18] Qiang Yu, Huajin Tang, Kay Chen Tan, and Haizhou Li. Precise-Spike-Driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns. *PLoS ONE*, 8(11):1–16, 2013.

[19] A. Maes, M. Barahona, and C. Clopath. Learning spatiotemporal signals using a recurrent spiking network that discretizes time. *PLOS Computational Biology*, 16(1):e1007606, January 2020.

[20] R. Gutig. Spiking neurons can discover predictive features by aggregate-label learning. *Science*, 351(6277):aab4113–aab4113, March 2016.

[21] J. Fil and D. Chu. Minimal spiking neuron for solving multi-label classification tasks. *Neurocomputing*, 2020. in press.

[22] N. Caporale and Y. Dan. Spike timing–dependent plasticity: A hebbian learning rule. *Annual Review of Neuroscience*, 31(1):25–46, July 2008.

[23] K. Miller and D. MacKay. The role of constraints in hebbian learning. *Neural Computation*, 6(1):100–126, 1994.

[24] W. Gerstner and W. Kistler. *Spiking Neuron Models*. Cambridge University Press, August 2002.

[25] W. Wiegerinck. *Stochastic dynamics of on-line learning in neural networks*. [Sl: sn], 1996.

[26] G. Orr and Todd K. Leen. Weight space probability densities in stochastic learning: Ii. transients and basin hopping times. In *NIPS*, pages 507–514, 1992.

[27] T. Leen and R. Friel. Stochastic perturbation methods for spike-timing-dependent plasticity. *Neural Computation*, 24(5):1109–1146, May 2012.

[28] M. Ahmed. Trained neural networks ensembles weight connections analysis. In *The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2018)*, pages 242–251. Springer International Publishing, 2018.

[29] J. Frankle, D. Schwab, and A. Morcos. The early phase of neural network training. In *International Conference on Learning Representations*, 2020.

[30] B. Perthame, D. Salort, and G. Wainrib. Distributed synaptic weights in a LIF neural network and learning rules. *Physica D: Nonlinear Phenomena*, 353-354:20–30, September 2017.

[31] B. Barbour, N. Brunel, V. Hakim, and J. Nadal. What can we learn from synaptic weight distributions? *Trends in Neurosciences*, 30(12):622–629, December 2007.

[32] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.