Optimal spike-based signal representation on a neuromorphic chip

Julian Büchel jubueche@ethz.ch
Institute of Neuroinformatics
University of Zürich, ETH Zürich
Zürich, ZH, Switzerland

Editor:

Contents

Abstract

The question of how a population of neurons encodes a signal in the brain is one of the most fundamental, yet unsolved problems in neuroscience. In general, it is believed that neurons encode information via their firing rate, specifically via Poisson rate codes (Maass et al. (2002)). Due to the unreliability induced by the Poisson-process, the representation error using Poisson rate codes only scales with $\frac{1}{\sqrt{M}}$, where M is the number of spikes. This leads to the fact that many spikes fired are redundant and are thus a waste of energy, contrary to the fact that neural agents seek to minimize energy consumption. This leads to the natural question of whether there exists a more complex system that represents signals with a representation error that scales with $\frac{1}{M}$, making better use of individual spikes.

Keywords: Neuromorphic Processors, Optimal Signal Representation, Neural Coding

1. Introduction

Recently, Brendel et al. (2017) proposed a local online learning rule, guiding a recurrently connected network into a tightly balanced domain. The authors showed that keeping a tight balance, commonly referred to as E/I-balance, improves the representation error, leads to sparser spike trains, and assigns higher value to individual spikes. The goal of this thesis is to implement this local learning rule on a neuromorphic chip in-the-loop, meaning that weights are iteratively loaded onto the neuromorphic chip and updated offline using responses obtained from the chip. Furthermore, this thesis aims at investigating the possibility of implementing this learning rule directly on-chip using mixed analog/digital circuits, by identifying possible weaknesses of the proposed theory and resolving them. Having a circuit that learns optimal spike-based signal representation is desirable from many points of view. From a neuroscience perspective, the emulation of such learning rule in a biologically realisitic setting would show that brains are indeed capable of implementing a more complex way of representing signals and might lead to new theories on how this is actually achieved in a pure biological setting. Additionally, the possibility of representing signals with high precision while saving a lot of energy might be applicable in neural agents, operating autonomously in environments that are providing continous stimuli or IoT devices requiring ultra low power consumption. In a more classical Machine Learning setting, learning such a network would correspond to a standard Autoencoder (Vincent et al. (2008)) with a high dimensional, sparse latent space, possibly opening up new ways of performing learning tasks in neural agents.

2. Theory: Representing signals spike-by-spike

2.1 Network dynamics

[Brendel et al. (2017)] In this work, we assume that we have a population of N neurons into which N_x continous signals are fed using a linear transformation $\mathbf{F}$. We furthermore assume that the population of N neurons is fully interconnected by a matrix $\mathbf{\Omega}$. It is thus beneficial to assume the computationally efficient Leaky Integrate-and-Fire (LIF) neuron model (Lapicque (1907)). The network dynamics can thus be written as

$$\dot{V}(t) = -\lambda V(t) + \mathbf{F}^{\mathbf{T}} c(t) + \mathbf{\Omega} o(t) \tag{1}$$

Where $c(t)$ is the input signal at time t and $o(t)$ the spike train of the population, also at time t. By defining a filtered version of the input c and the spike train o, we can integrate Equation 1 to

$$V(t) = \mathbf{F^T} x(t) + \mathbf{\Omega} r(t) \tag{2}$$

where we define

$$\dot{r}(t) = -\lambda r(t) + o(t) \tag{3}$$

and

$$\dot{x}(t) = -\lambda x(t) + c(t) \tag{4}$$

For the simulations conducted in this thesis, uncorrelated inputs where used. These were generated by simply convolving white noise with a gaussian filter of width σ. An example of a typical input used throughout this thesis is presented in Figure 1.

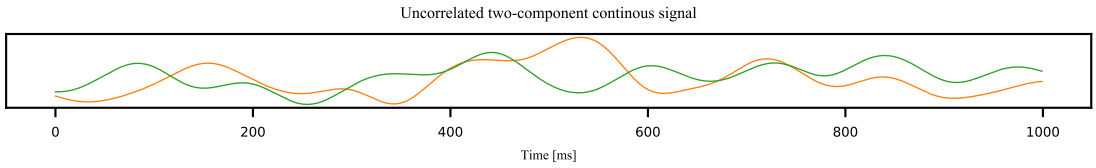

Figure 1: Input x of dimension $N_x = 2$ shown over a period of 1000 ms.

It should be noted that for the sake of mathematical simplicity, as well as ease of transformation to a neuromorphic chip, this setup does not respect Dales law (Dale (1935)), as it allows for neurons to be excitatory and inhibitory at the same time. However, (Brendel et al. (2017)) also derives learning rules that respect the more biologically plausible setup.

We furthermore assume a readout layer, or decoder, that reconstructs the input x using the rates exposed by the population. The reconstructed signal can be defined as

$$\hat{\mathbf{x}} = \mathbf{Dr}$$

where the decoder $\mathbf{D}$ is a matrix of size (N_x,N), $\mathbf{r}$ is a matrix of shape (N,N_{time}) and $\mathbf{x}$ is a matrix of shape (N_x,N_{time}).

2.2 The optimal decoder

Keeping in mind the goals that we are following, namely low reconstruction error, low firing rates and even distribution across neurons, we can define the following loss function, balancing low reconstruction error with sparse (*l1*-cost) and dense (*l2*-cost) spike trains.

$$L^* = \min_o \|\mathbf{x} - \mathbf{Dr}\|_2^2 + \mu\|\mathbf{r}\|_2^2 + \nu\|\mathbf{r}\|_1 \tag{5}$$

Because of Equation 3, one can see that enforcing *l2*-sparsity allows for dense population codes (since it keeps the rates genereally low, but does not enforce 0 values). In contrast, *l1*-sparsity encourages 0 values in the rates, which correponds to sparse spike trains (Tibshirani (1996)). This loss function gives us an explicit formula for the optimal decoder that is

achievable by the network. By taking the derivative of Equation 5 with respect to $\mathbf{D}$ and setting it to $\mathbf{0}$, we get

$$\mathbf{D}^* = \mathbf{x}\mathbf{r}^\mathbf{T}(\mathbf{r}\mathbf{r}^\mathbf{T})^{-1}$$

2.3 Optimal network connectivity

Following Brendel et al. (2017), we will now discuss how to optimize the network topology. In particular the synaptic weights and the thresholds under the assumption that the decoder is fixed. Note that letters such as r are not bold and thus stand for a vector in time, $r(t)$, which is abbreviated for clarity. The main idea (Bourdoukan et al. (2012)) behind the following derivation is that a neuron in the population should only spike if it contributes to the reduction of the loss function. This means that

$$L(\textit{neuron } n \textit{ spikes}) < L(\textit{neuron } n \textit{ does not spike}) \tag{6}$$

Using Equation 5, $L(\textit{neuron } n \textit{ spikes})$ can be written as

$$L(\textit{neuron } n \textit{ spikes}) = \|c - Dr - D_n\|_2^2 + \mu\|r + e_n\|_2^2 + \nu\|r + e_n\|_1$$

where e_n is the unit vector with $(e_n)_j = 1$ if $j = n$ and 0 else.

We observe that

$$\|r + e_n\|_2^2 = (\sum_{i,i\neq n} r_i) + (r_n + 1)^2 = \|r\|_2^2 + 2r_n + 1 \tag{7}$$

and

$$\nu\|r + e_n\|_1 = \nu(\sum_{i,i\neq n} |r_i|) + \nu|r_n + 1| =^* \nu(\sum_{i,i\neq n} |r_i|) + \nu|r_n| + \nu = \nu\|r\|_1 + \nu \tag{8}$$

where * holds because $\forall t\ r(t) \geq 0$. Using Equations 7 and 8 we can rewrite Equation 6 to

$$D_n^T c - D_n^T D r - \mu r_n > \frac{1}{2}(\|D_n\|_2^2 + \mu + \nu) \tag{9}$$

This condition enforces that neuron n only fires a spike, when it contributes to the reduction of the global error. We observe, under the assumption that the decoder is fixed, that the r.h.s. is constant and the l.h.s. is time-dependent, as it depends on c and r. It is therefore natural to denote the voltage of neuron n as

$$V_n(t) = D_n^T c(t) - D_n^T D r(t) - \mu r_n \tag{10}$$

and the threshold of the n-th neuron as

$$T_n = \frac{1}{2}(\|D_n\|_2^2 + \mu + \nu) \tag{11}$$

using the differential equations 4 and 3, as well as Equation 1 we rewrite Equation 10 to

$$V(t) = \mathbf{F^T}x(t) + \mathbf{\Omega}o(t)$$

where we substituted $\mathbf{F} = \mathbf{D}$ and $\mathbf{\Omega} = -\mathbf{D^T D} - \mu\mathbf{I} = -\mathbf{F^T F} - \mu\mathbf{I}$, which are the optimal feed-forward and recurrent weights in the network, with respect to the decoder.

We also note that since $\mathbf{\Omega} = -\mathbf{D^T D} - \mu\mathbf{I}$, we have $T = \frac{1}{2}(\text{diag}(\mathbf{\Omega}) + \nu)$. This way the error dictates the produced spike train, hence *error-driven coding*.

Since neurons have a "fixed" threshold, Equation 11 can be seen as a constraint on the decoder length for a neuron by simply rewriting it to

$$\|D_n\|_2^2 = 2T_n - \mu - \nu$$

2.4 Learning optimal recurrent connectivity

We have seen from the previous section that the voltage encodes the global loss and that a neuron fires a spike only to reduce the reconstruction error. This corresponds to the hypothesis that a tightly balanced network exhibits a low reconstruction error. This however corresponds to the state where $\mathbf{x} - \hat{\mathbf{x}} \approx 0$. We observe that the neuron has only access to the linearly transformed version: $F_n^T\mathbf{x} - F_n^T D\mathbf{r}$. However, this value should also be close to zero, as it reflects part of the reconstruction error.

$$\begin{aligned} \epsilon_n &= F_n^T\mathbf{x} - F_n^T\hat{\mathbf{x}} \\ &= F_n^T - F_n^T\mathbf{D}\mathbf{r} \\ &\approx 0 \end{aligned} \tag{12}$$

We furthermore observe, using the optimal feed-forward and recurrent connections, that Equation 12 corresponds to the network dynamics obtained in Equation 2. We can conclude that the signal is represented optimally if $\Omega_n = -F_n^T\mathbf{D}$, $\epsilon_n = V_n \approx 0$ and $\mathbf{D} = \mathbf{D}^*$. From this, we are now able to derive a learning rule for the recurrent weights, that will yield the optimal solution.

In order to achieve a balanced network, it is desirable that over the long run, the membrane voltages before and after a presynaptic spike cancel each other:

$$L(t_j) = \|\frac{1}{2}(V^{\text{before}}(t_j) + V^{\text{after}}(t_j))\|_2^2 \tag{13}$$

Note that we add the two voltages since we assume that the voltage jumps from the positive to the negative domain. After the arrival of a presynaptic spike at time t_j of neuron with index $k(j)$, the voltages are changed and the following relation is formed:

$$V^{\text{after}} = V^{\text{before}} + \mathbf{\Omega}e_{k(j)} \tag{14}$$

where $e_{k(j)}$ is the unit vector with 1 at the position of the neuron that spiked. Using this relation we can rewrite the Equation 14 to

$$L = \|V^{\text{before}} + \frac{1}{2}\mathbf{\Omega}e_{k(j)}\|_2^2$$

this term has the derivative, with respect to $\Omega_{n,k}$:

$$\Delta\Omega_{n,k} = -2V_n^{\text{before}} - \Omega_{n,k}$$

if neuron k spiked. The connection $\Omega_{n,k}$ is the connection from the spiking, presynaptic neuron with index k to the n-th neuron.

If we induce a small $l2$-cost on the rates, $\mathbf{\Omega}$ should ideally converge towards $-\mathbf{F^T D} - \mu\mathbf{I}$. This causes a minor change in the learning rule for the recurrent weights:

$$\Delta\Omega_{n,k} = -2(V_n^{\text{before}} + \mu r_n) - \Omega_{n,k} - \mu\delta_{n,k}$$

where $\delta_{n,k}$ is $[e_n]_j$ and therefore 1 if $n = j$.

2.5 Simulation and results

Learning the feed-forward connections happens on a much slower timescale and thus requires far more training iterations. Since the DYNAP-SE operates on a much slower, biologically more plausible timescale (Moradi et al. (2017)), training takes orders of magnitude longer. We therefore decided to omit learning the feed-forward weights and keep them constant after an initial random initialisation. The pseudo code presented below was adapted from (Brendel et al. (2017)) and shows the learning procedure.

Input: Number of training iterations N_{iter}
Output: Learned recurrent matrix $\mathbf{\Omega}$

$T \leftarrow \mathrm{N_{iter}} \cdot \mathrm{N_{time}}$
$thresh \leftarrow 0.5$
for $i \leftarrow 0$ **to** T **do**
  if $i \bmod \mathrm{N_{time}} = 0$ **then** `// Generate new input`
    $Input \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $Input \leftarrow A \cdot (Input * w)$
  end
  $t \leftarrow (i \bmod \mathrm{N_{time}}) + 1$
  $V_t \leftarrow (1 - \lambda dt) \cdot V_{t-1} + dt \cdot \mathbf{F^T}\mathrm{Input}_t + \mathbf{\Omega} o_{t-1} + \epsilon \cdot \mathcal{N}(0, 1)$
  $x_t \leftarrow (1 - \lambda dt) \cdot x_{t-1} + dt \cdot \mathrm{Input}_t$
  $(\max, k) \leftarrow \arg\max(V_t - thresh - 0.01 \cdot \mathcal{N}(0, 1))$
  if $\max \geq 0$ **then** `// Neuron k spiked`
    $\Omega_k \leftarrow \Omega_k - \epsilon_\Omega \cdot (\beta \cdot (V_t + \mu \cdot r_{t-1}) + \Omega_k + \mu \cdot I_k)$
    $r_t \leftarrow r_{t-1} + I_k$
  else
    $r_t \leftarrow r_{t-1}$
  end
  $r_t \leftarrow (1 - \lambda dt) \cdot r_t$
end

Algorithm 1: Algorithm describing the learning procedure without learning the feed-forward weights. Ω_k and I_k reference the k-th column of the respective matrix and $*$ is the convolution operator.

Given the randomly initialized feed-forward matrix $\mathbf{F}$, we can measure the distance between $\mathbf{\Omega}$ and $\mathbf{\Omega}^* = -\mathbf{F}^\mathbf{T}\mathbf{F}$, even if $\mathbf{F}$ is not learned. As expected, the recurrent connection matrix converges towards the optimal connectivity (Figure 2(i)). Initially, the network is in an unbalanced state and the voltage predominantly reflects the transformed input. After learning the recurrent weights, incoming excitatory signals are matched by recurrent inhibition to keep the network in a tightly balanced state. This effect can be observed in Figure 2(iv) where we show the variance of the membrane potential over time. By converging towards the optimal recurrent connection and keeping the network in a tight balance, we fulfill the conditions necessary for representing the signal optimally. The result of this can be seen in Figure 2(iii), where the reconstruction error converges towards the discretization limit imposed by the neurons. Since the learning rule keeps the network in a tight balance, no spike is fired unnecessarily. Each spike, following Equation 6, contributes to reducing the global reconstruction error. This way, the resulting spike train sparsifies, which can be observed in Figure 2(ii).

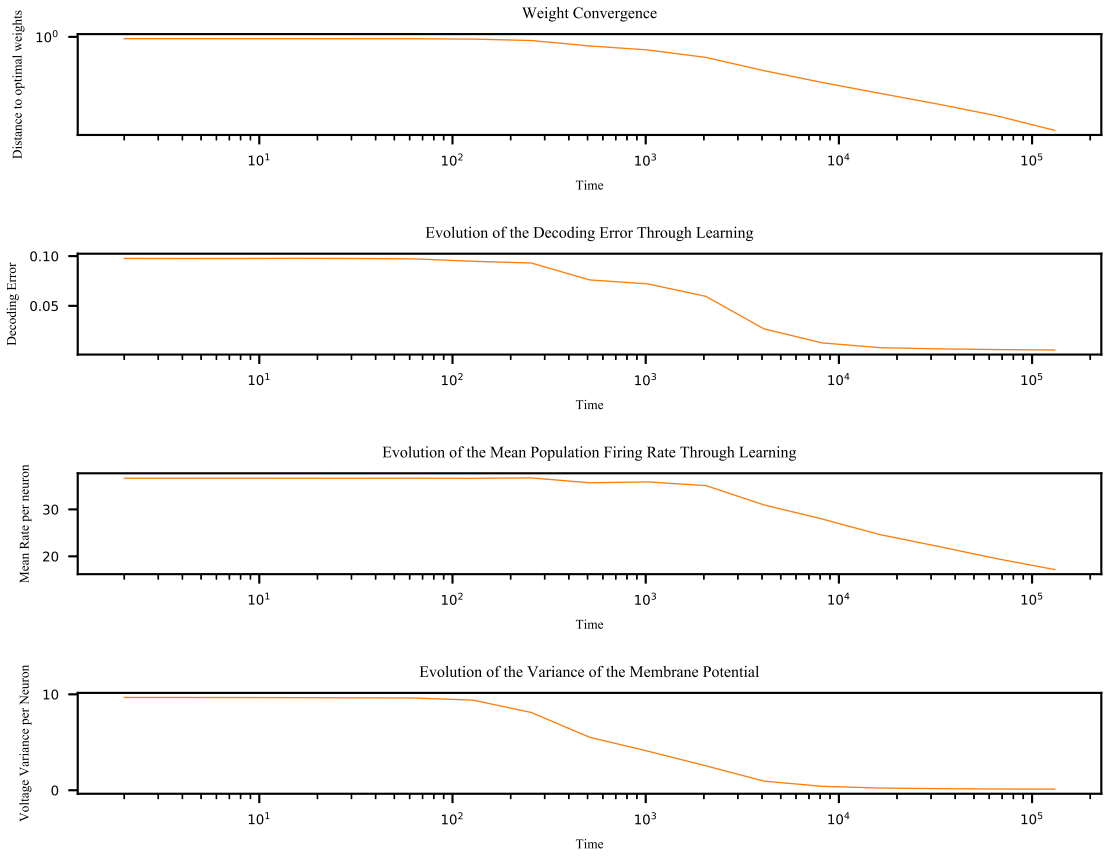

Figure 2: (i) Convergence of optimal weights (ii) Convergence of error (iii) Convergence of mean firing rate (iv) Convergence of voltage variance

To see the effect of learning, we trained the network for 140 iterations on different signals from the same domain. After the training process, we computed the optimal decoder for the network using $\mathbf{\Omega}_{\text{initial}}$ and for the network using $\mathbf{\Omega}_{\text{post learning}}$. We then generated a new input x, which was then fed into the networks. Using the decoder previously computed, we then reconstructed the signal and compared the reconstruction quality. Figure 3 shows the different reconstructions and spike trains of each network. As expected, results using $\mathbf{\Omega}_{\text{initial}}$ produced a poor reconstruction compared to the network using $\mathbf{\Omega}_{\text{post learning}}$. It is also visible from the spike trains that the voltage initially only represents the transformed input and later becomes more distributed across the population.

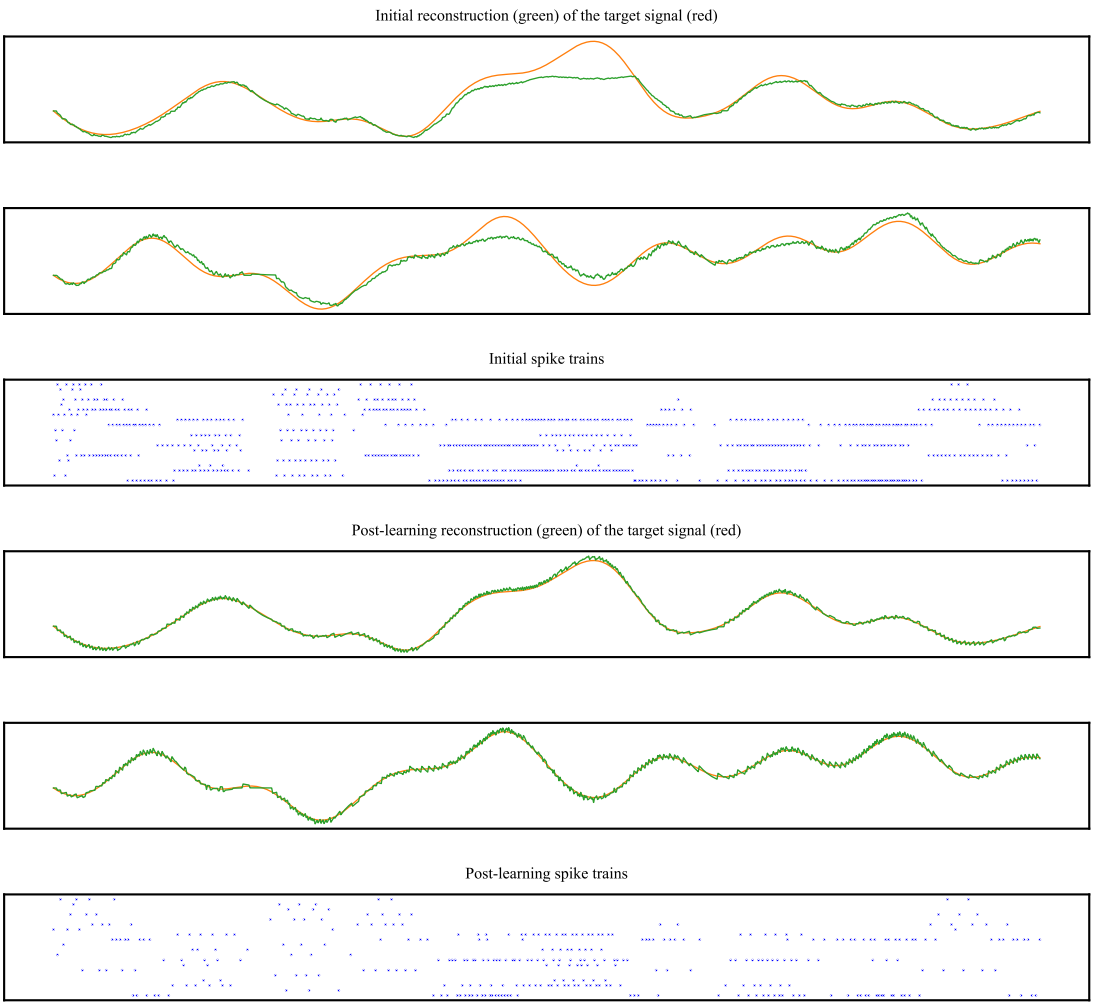

Figure 3: The two-dimensional input is reconstructed pre-learning (row 1 & 2) and post-learning (row 4 & 5). Due to the evolving tight balance, the population spike trains sparsify (row 3 & 6).

2.6 Limitations of the theory

Unfortunately, the theory proposed by (cite Brendel) has some limitations:
1) Discretization of the weights causes unsuccessful learning of the recurrent matrix and therefore does not lead to improved sparsity and reconstruction error.
2) Updating the rate, as well as all outgoing connections of *all* neurons that spiked, results in no improvement of the mean firing rate and the reconstruction error. Furthermore, the update utilizes the voltage *before* the spike propagated.

In this theory, tight balance of the input currents is of crucial importance. When discretizing the weights, this tight balance can no longer be achieved and no improvement of the reconstruction error is visible. Considering that synaptic weights in the human brain actually do have a reasonably high resolution (need citation), this constraint is solely imposed by the hardware we use and will be resolved in future chip designs by e.g. using multi-memristive synapses (Boybat et al. (2018)).

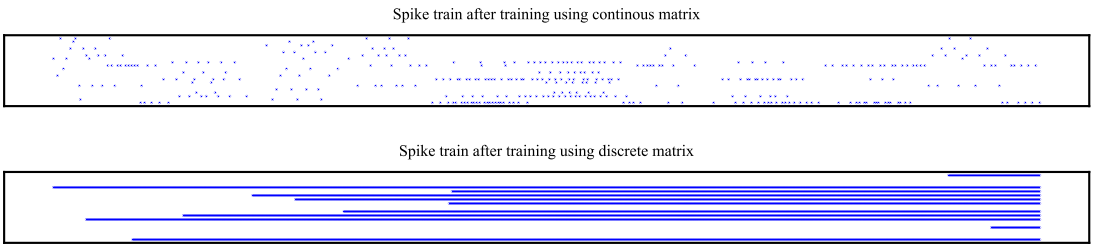

Figure 4: The spike train after training using continous weights (top), exhibits a usable, distributed pattern, while the learned, discretized weight matrix (bottom) shows a subset of neurons firing at a high frequency.

Assuming a continous timescale, there can not be an instance in time, where two neurons fire simultaneously. However, in the simulations, as well as on the DYNAP-SE, time is discretized and therefore the event that two neurons fire at the same time *can* happen. When adapting Equation 14 to account for this possibility and rewriting the update rule accordingly, we found that the recurrent weights were still learned. Furthermore, the variation in the membrane potential also decreased, which is explained by the fact that we are still minimizing the loss in Equation 13. However, the derivation of the optimal network connectivity in Equation 9 falsely assumes only one spike per time.

The only problem that we identified as biologically unrealistic is the fact that the postsynaptic neuron updates the synaptic weight based on the voltage *before* the spike propagated, meaning the voltage was not affected by the spike. This problem might be resolved by implementing a slow changing variable, approximating the voltage before the spike due to the slow update.

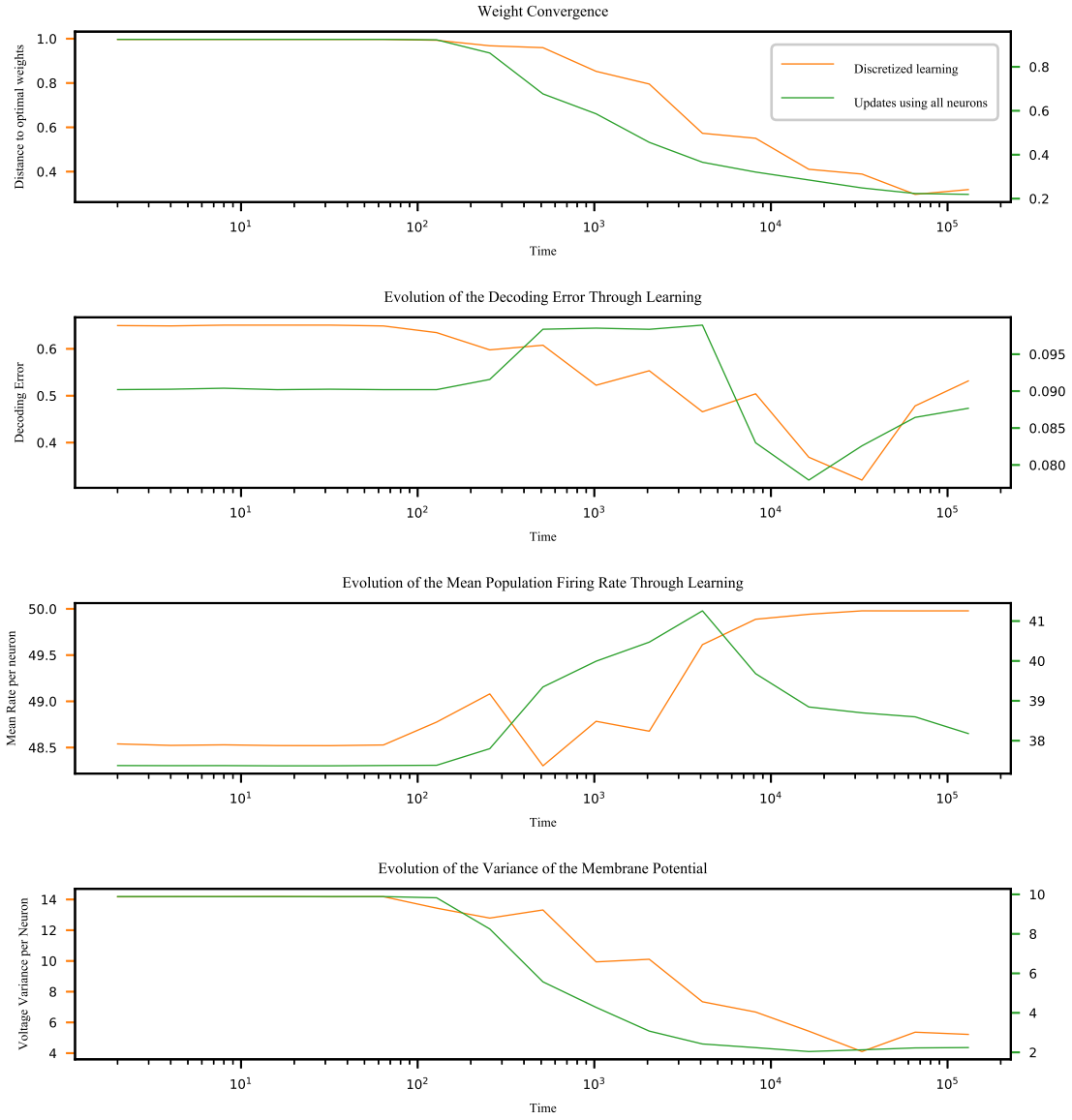

Figure 5: Convergence of different properties when learning with discrete weights (orange) and performing the updates with respect to all spiking neurons (green). It should be noted that the mean firing rate is really high compared to the successful learning (see Figure 2).

3. Hardware: The DYNAP-SE chip

The DYNAP-SE is a fully custom mixed signal neuromorphic chip developed by Ning Qiao, Saber Moradi, Fabio Stefanini and Giacomo Indiveri (Moradi et al. (2017)). It comprises 4 chips, each containing 4 cores with 4x256 neurons following the adaptive exponential integrate-and-fire neuron model (AEI&F) with a fan-in of 64 and fan-out of 4064. Each neuron has a Content Adressable Memory (CAM) block, containing 64 addresses representing the pre-synaptic neurons that the neuron is subscribed to. The DYNAP-SE comprises 4 different synapse types: Slow/Fast inhibitory and excitatory synapses. Using the Address-Event-Representation (AER) (Deiss et al. (1999)), spikes are asynchronously transmitted in the form of neuron addresses. The spike times are directly encoded in the event times and routed using a mixed 2D-mesh-hierachical routing scheme with microsecond precision.

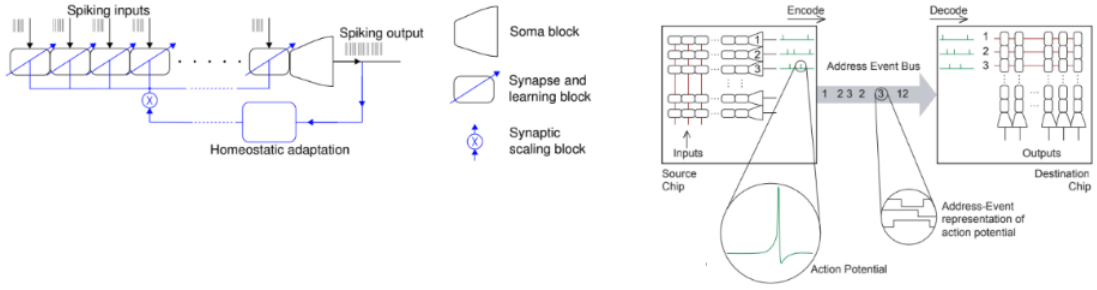

Figure 6: (l) Each neuron can utilize at most 64 synapses, meaning that it can have at most 64 unique connections. This imposes a severe constraint on network precision, since weights are usually a multiple of single synapses, as described in section (4.2.2). (r) Each neuron is assigned an address that is encoded as a digital word and transmitted as soon as the neuron spikes/triggers an event. Using the transmitted address and arrival time, spikes can be decoded again.

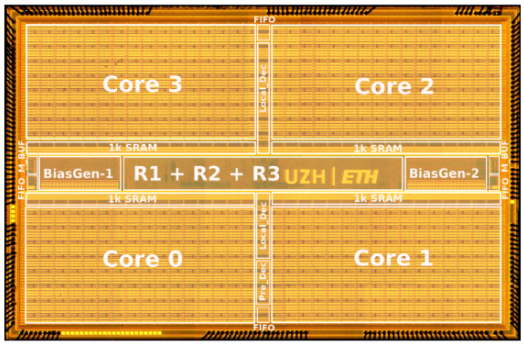

Figure 7: The DYNAP-SE has a core area of $0.36mm^2$ and utilizes roughly $1.68nJ$ per spike. In comparison: IBM's True North (Cassidy et al. (2016)) utilizes $3.9nJ$ on $0.094mm^2$ core area.

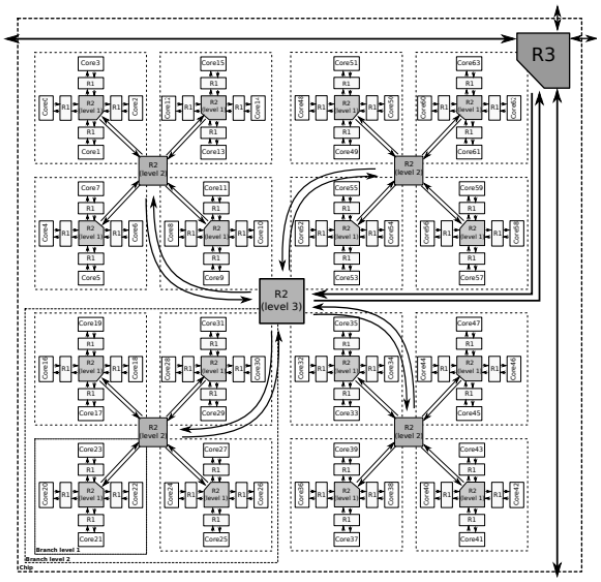

Figure 8: Following a tree-based routing strategy, groups of four cores can communicate with other cores in the same group using level 1 R2 routers or between different groups on the same tile using level 2 R2 routers. To reach another tile, the level 3 R2 router is used. Following a 2D-mesh approach, the R3 router can be used to communicate between multiple chips.

4. Learning optimal spike-based signal representations on the DYNAP-SE

4.1 Setup: Learning in-the-loop

Although spike-timing dependent plasticity is implemented on-chip (Moradi et al. (2017)), this learning algorithm requires an in-the-loop setup. The reason for this was already pointed out in section 2.6: Discretizing the weights in the simulation makes learning impossible, even with a relatively high number of bins. This is why we have to keep a continous off-chip version of the recurrent matrix during learning. In this setup, we repeatedly present a signal to the chip, record the emitted spikes and perform the matrix update off-chip. The updated matrix is then rescaled, discretized and loaded onto the chip and the process starts all over again (see Figure 9). For a more detailed description of the program see section 4.2.5.

Since this setup utilizes the DYNAP-SE, which operates on a biologically plausible timescale, this learning process takes orders of magnitudes longer than the off-chip simulation.

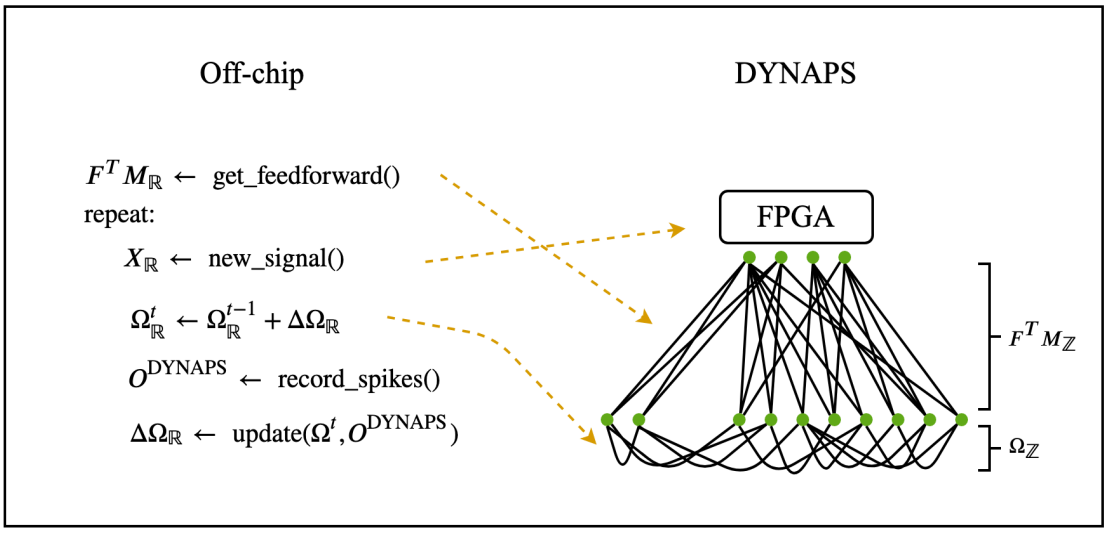

Figure 9: The feed-forward connections $\mathbf{F^T M}$ (derived in section 4.2.1) are initially discretized to $\mathbb{Z}$ and loaded onto the chip. The following four steps are executed repeatedly for a fixed number of iterations: A new signal is generated and stored in the FPGA memory, the recurrent weights $\mathbf{\Omega}$ are updated, discretized to $\mathbb{Z}$ and loaded onto the chip, the DYNAP-SE runs on the spiking input and finally $\mathbf{\Delta\Omega}$ is computed using the spike trains $\mathbf{O}^{\text{DYNAP-SE}}$ recorded from the DYNAP-SE.

4.2 Aligning on- and off chip network dynamics

4.2.1 Network with spiking input

Information in the brain is processed in the analog and digital domain. Spikes, emitted from pre-synaptic neurons, cause a change in the membrane potential of synapses leading to the emission of different neurotransmitters. Inputs to the brain are converted to spikes using different nerve cells, such as the photoreceptors in the retina. The DYNAP-SE also receives input in the form of spikes, which raises the question on how to convert a continous signal to spikes. One simple approach called Delta Modulation (Corradi and Indiveri (2015)), converts a 1D-signal to two channels, namely the UP and DOWN channel. The spikes emitted in the respective channels depend on the magnitude and direction of the current signal slope. Figure 10 illustrates the spikes of a 1D-signal using 0.05 as a spiking threshold in both directions.

Going back to the simulation, one can see that the input to the network is

$$\text{Input} = \mathbf{F^T} x$$

where $\mathbf{F}$ is a matrix of size (N_x,N). By integrating the spikes using

$$\hat{x}_t = (1 - \lambda)\hat{x}_{t-1} + \nu(o_t^{up} - o_t^{down}) \tag{15}$$

one can reconstruct a discretized version of the original signal $\hat{x}$ using appropriate values for λ and ν. Using

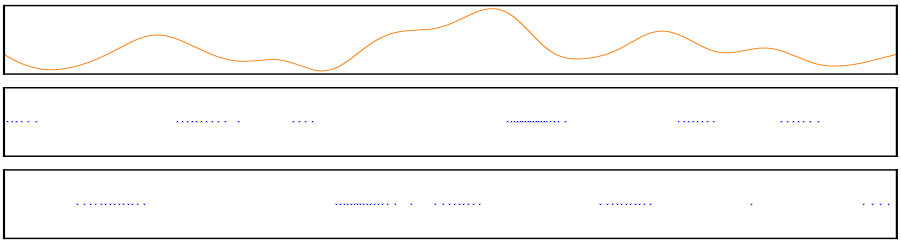

Figure 10: The signal (top) is split into two channels, representing the (upward) downward magnitude of the signal in the number of spikes.

$$M = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

the general form of Equation 15 becomes

$$\hat{x}_t = (1-\lambda)\hat{x}_{t-1} + \nu M o_t$$

In order to match the simulation with the DYNAP-SE, one could simply integrate the spikes and transform the reconstructed signal to get an approximation of the input using

$$\text{Input} = \mathbf{F}^T\hat{x} = \mathbf{F}^T((1-\lambda)\hat{x}_{t-1} + Mo_{t-1})$$

This however does not match the setup on the DYNAP-SE since the input is first transformed using M, then integrated and then again transformed using F. Contrary, the synapses on the DYNAP-SE first integrate the incoming spikes and then transform the integrated value using the weights. We therefore introduce a new variable I_t that symbolizes the current at time t of the synapses. I_t performs a simple integration of the input:

$$I_t = (1-\lambda)I_{t-1} + \nu o_t$$

By expanding the recursive definition for k time steps and assuming that $(1-\lambda)^k$ becomes negligibly small for some k we get that

$$\begin{aligned} F^T\hat{x}_t &= F^T((1-\lambda)\hat{x}_{t-1} + \nu M o_t) \\ &= F^T((1-\lambda) \dots (1-\lambda)((1-\lambda)\hat{x}_{t-k} + \nu M o_{t-k+1}) + \nu M o_{t-k+2} + \dots + \nu M o_t) \\ &\approx F^T((1-\lambda)^{k-1}\nu M o_{t-k+1} + (1-\lambda)^{k-2}\nu M o_{t-k+2} + \dots + (1-\lambda)^0 \nu M o_t) \\ &= (1-\lambda)^{k-1}\nu F^T M o_{t-k+1} + (1-\lambda)^{k-2}\nu F^T M o_{t-k+2} + \dots + (1-\lambda)^0 \nu F^T M o_t \end{aligned}$$

is equal to

$$
\begin{aligned}
F^T M I_t &= F^T M((1-\lambda)I_{t-1} + \nu o_t) \\
&= F^T M((1-\lambda)...(1-\lambda)((1-\lambda)I_{t-k} + \nu o_{t-k+1}) + \nu o_{t-k+2} + ... + \nu o_t) \\
&\approx F^T M((1-\lambda)^{k-1}\nu o_{t-k+1} + (1-\lambda)^{k-2}\nu o_{t-k+2} + ... + (1-\lambda)^0 \nu o_t) \\
&= (1-\lambda)^{k-1}\nu F^T M o_{t-k+1} + (1-\lambda)^{k-2}\nu F^T M o_{t-k+2} + ... + (1-\lambda)^0 \nu F^T M o_t
\end{aligned}
$$

We have shown that having real input transformed using a feed-forward matrix F is approximately the same as integrating a spike train o_t and transforming it using the new matrix $F^T M$. This scheme was successfully verified in the simulation. Figure 12 compares the reconstruction error over time between the continous-input network and the spiking-input network. Since the DYNAP-SE is doing the integration implicitly, we can simply set the feed-forward connections to a discretized version of $F^T M$ and feed the delta modulated input to the chip. Unfortunately, input precision is lost due to the discretization limit imposed by the spikes and more severly by the weight resolution on the DYNAP-SE, which will be covered in another section.

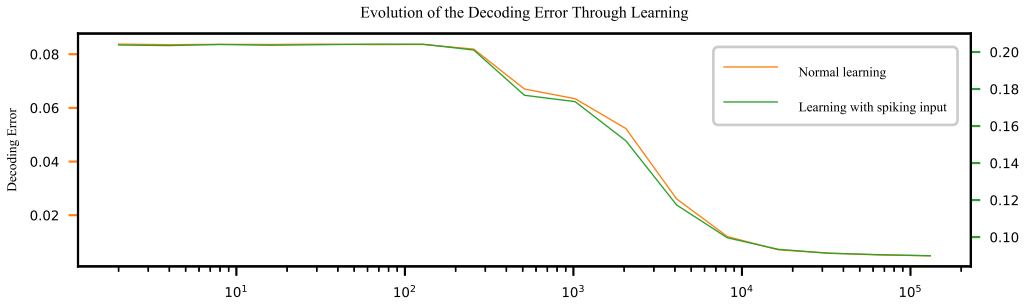

Figure 11: While the error convergence of the spiking-input and continous-input network matches, the error of the spiking-input network is increased by roughly a factor of 10, due to the loss of precision imposed by the spikes (note the green scale).

4.2.2 The weights on-chip

As pointed out in section 3, the fan-in of the DYNAP-SE is relatively small compared to the fan-out. One neuron can have a maximum of 64 incoming connections. This becomes a serious problem considering, that in a fully connected network the number of connections scales quadratically with the number of neurons. Luckily, the learned matrix exhibits a clear structure, with many values close to zero, so when discretized, only a few strong inhibitory or excitatory connections remain. Figure 12 compares the continous and discretized weight matrices post learning.

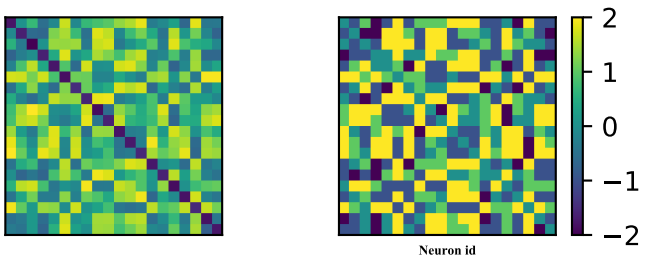

Figure 12: Due to discretization of the matrix, precision is lost. Note that in the right plot, only 5 distinct discrete values are visible, while the right matrix is continous. However, the global structure is preserved and the matrix is symmetric, as the graph should be undirected.

In order to understand the process of rescaling the weights, it is important to know that the DYNAP-SE only has two types of excitatory and inhibitory synapses, respectively. This means that the weights of these two synapses are actually set pre-learning and that connections with different weights have to be formed using multiples of these synapses. Assuming that we have discrete weights going from -10 to 10, with a resolution of 1, one can form a connection of strength 3 between two neurons, by forming three seperate connections using the excitatory synapse. Note that the weights have to be in $\mathbb{Z}$ since their absolute value indicates the number of synapses used and the sign the type of synapse ($-$ for inhibition and $+$ for excitation). Since this is the case, we can simply bin the continous weights and take the indices minus the number of incoming connections we allow. To illustrate this consider the following array:

$$X = [0.5, -0.14, 0.65, 1.52, -0.23, -0.23, 1.58, 0.77, -0.47, 0.54]$$

when binning with 20 bins (meaning we have 20 different weights, -10 being the smallest and 10 the biggest), this yields the following bin edges:

$$\text{bin edges} = [-0.47, -0.37, -0.26, -0.16, -0.06, \dots, 1.27, 1.37, 1.48, 1.58]$$

Taking the indices of the bins for each value minus the number of the maximum strength per connection (10) we get the weights:

$$\text{weights} = [0, -6, 1, 10, -7, -7, 10, 3, -10, 0]$$

Using this approach causes an undesired effect: Whenever a matrix is rescaled to a certain range, the information on the weight magnitute, relative to other matrices is lost. A very small matrix at the beginning of learning compared to a matrix with more extreme weights would have the same magnitude in the rescaled domain. This undesired effect can be easily resolved by providing a point of reference for the weight magnitudes in the form of minimum and maximum values in the binning process. These values were obtained from a previous test run in the simulation. Since it is just a heuristic and learning using the DYNAP-SE produces different weights, it can happen that weights exceed these limits. In this case they are simply assigned the minimum or maximum weight. We also verified that using eye-balled values like -0.5 and 0.5 worked.

Pushed back by this severe loss in precision and therefore disability to align the off-chip-dynamics with on-chip-dynamics, we had to compensate with the right choice of hyperparameters such as the threshold for the delta modulation and the weights of the inhibitory and excitatory synapses. To find out the right values, we tried out two different forms of hyperparameter tuning: Simple grid search and coordinate-descent-inspired grid search. The grid search approach simply iterated through every possible interleavings of hyperparameters, while the coordinate-descent version picked a parameter to optimize in each round and remembered the best value for each parameter. To measure quality of alignment, we passed a random signal through the on- and off-chip network using random feedforward and recurrent connections and compared the spike trains using different metrics.

4.2.3 Batched updates

Being a spike-dependent learning rule, weight-updates happen immediately after a spike has been fired. While being biologically plausible, this type of update is not achievable in our in-the-loop setup. To match this, we would have to chop up the signal to chunks of size ΔT and iteratively present, by ΔT increasing, parts of the signal and only do an update in the last ΔT, when there was a spike. Since a lot of overhead is created when setting the connections on the DYNAP-SE, this becomes impractical. Since the future behaviour of the neurons depend on the current weights, it is clear that accumulating the updates and performing the update at the end of the whole signal is not the same as changing the weights immediately. Assuming that the signal is relatively small and the recurrent weights do not change too much during one signal iteration, one can however approximate the online updates with a batch update. Contrary to the intuitive assumption that we had to scale the accumulated derivatives relative to the number of updates, we found that learning behaves quite similar to the online setting when the derivatives are not normalized as can be seen in Figure 13.

4.2.4 Further alignment using time-window update

One fundamental problem that will be resolved in a future chip design, is that membrane potentials can only be measured using an oscilloscope. This imposes a problem, since the weight updates rely on the neurons voltages. To circumvent this limitation, we tried to reconstruct the voltage V_t based on the transformed input $\mathbf{F^T}x(t)$ and the spikes $o^{DYNAP-SE}(t-1)$. Upon comparison of the reconstructed voltage and the simulation volt-

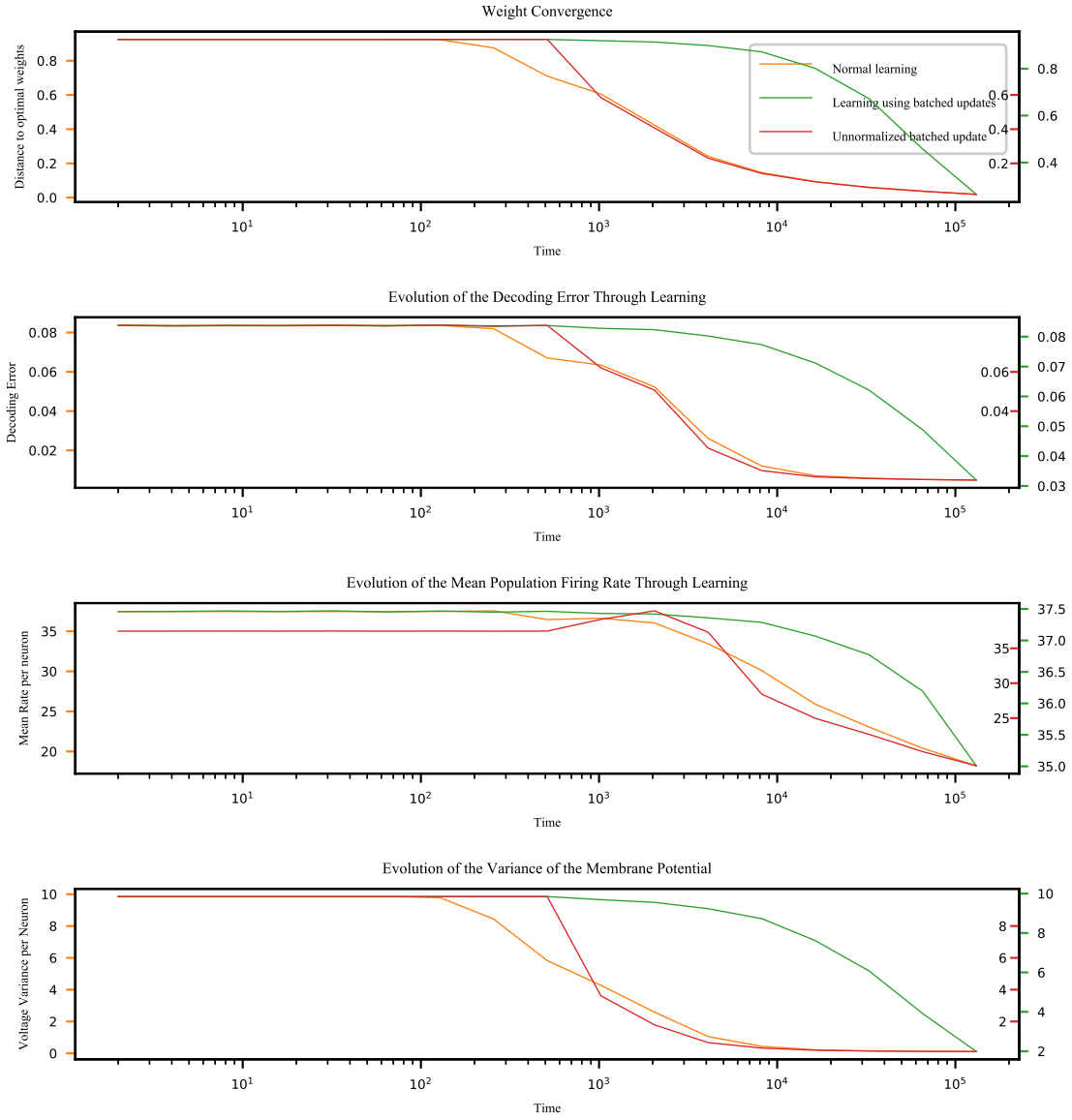

Figure 13: The convergence properties using the batched and online versions of the learning rule. The unscaled batched version performs equally well as the online version.

age, we realized that the reconstructed voltage of the population was generally higher than the simulation version. The reason for this is simple: In the simulation, there can be only one spike per time, but on the DYNAP-SE, occasionally multiple neurons spiked simultaneously, which we did not take into account. To resolve this mismatch, we computed the reconstructed voltage using

$$V(t) = \mathbf{F^T} x(t) + \mathbf{\Omega} o^{aligned}(t)$$

where $o_k^{aligned}(t)$ is one if $V_k(t-1)$ was the maximum value above threshold and the DYNAP-SE emitted a spike in a specified time-window $\pm\Delta T$. Note that because the network dynamics off- and on-chip are not perfectly aligned, we considered a small time-window, typically of size $\pm 5ms$, instead of checking if there was a spike at exactly time t. The final pseudo code is illustrated in the following section.

4.2.5 Final pseudo code

Input: Number of training iterations N_{iter}
Output: Learned recurrent matrices $\mathbf{\Omega}, \mathbf{\Omega}^{\mathbb{Z}}$

$thresh \leftarrow 0.5$
$T_{\text{delta modulator up}} \leftarrow 0.05$
$T_{\text{delta modulator down}} \leftarrow 0.05$
$\Delta T \leftarrow 5ms$
$\Omega_{\min} \leftarrow -0.5$
$\Omega_{\max} \leftarrow 0.5$
$\mathbf{F} \leftarrow \text{normalize}(\frac{1}{2}\mathcal{N}(\mathbf{0}, \mathbf{I}))$
$\mathbf{F^T M}^{\mathbb{Z}} \leftarrow \text{discretize}(\mathbf{F^T M})$
dynaps.set_feedforward($\mathbf{F^T M}^{\mathbb{Z}}$)
for $i \leftarrow 0$ **to** N_{iter} **do**
  Input $\leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Input $\leftarrow A \cdot (\text{Input} * w)$
  for $t \leftarrow 1$ **to** T_{signal} **do**
    $X_t \leftarrow (1 - \lambda dt) \cdot X_{t-1} + dt \cdot \text{Input}_t$
  end
  fpga.load_signal(X_t)
  $\mathbf{\Omega}_t \leftarrow \mathbf{\Omega}_{t-1} - \Delta\mathbf{\Omega}$
  $\mathbf{\Omega}_t^{\mathbb{Z}} \leftarrow \text{discretize}(\mathbf{\Omega}_t, \Omega_{\min}, \Omega_{\max})$
  dynaps.set_recurrent_connections($\mathbf{\Omega}_t^{\mathbb{Z}}$)
  $\mathbf{O}^{DYNAP-SE} \leftarrow$ dynaps.run()
  for $t \leftarrow 1$ **to** T_{signal} **do**
    $V_t \leftarrow (1 - \lambda dt) \cdot V_{t-1} + dt \cdot \mathbf{F^T}\text{Input}_t + \mathbf{\Omega} o_{t-1}^{\text{aligned}} + \epsilon\mathcal{N}(0, 1)$
    $(\max, k) \leftarrow \arg\max(V_t - thresh - 0.01 \cdot \mathcal{N}(0, 1))$
    if $\max \geq 0 \wedge \text{has\_spike}(\mathbf{O}_{k,\pm\Delta T}^{\text{DYNAP-SE}})$ **then**
      $\Delta\Omega_k \leftarrow \Delta\Omega_k + \epsilon_\Omega \cdot (\beta \cdot (V_t + \mu \cdot r_{t-1}) + \Omega_k + \mu \cdot \mathbf{I})$
      $o_t^{aligned} \leftarrow I_k$
      $r_t \leftarrow r_{t-1} + I_k$
    else
      $o_t^{aligned} \leftarrow \mathbf{0}$
      $r_t \leftarrow r_{t-1}$
    end
    $r_t \leftarrow (1 - \lambda dt) \cdot r_t$
  end
end

Algorithm 2: The final pseudo code illustrating the changes made, compared to algorithm 1. Note for example the batched updates or the discretized feed-forward matrix in the beginning. All hyperparameters used can be found in the Appendix.

4.3 Results

Algorithm 2 was run for $\mathrm{N_{iter}} = 270$ iterations with $N = 20$ neurons and $\mathrm{N_x} = 2$ input signals. These signals were converted to $\mathrm{N_{in}} = 4$ channels, representing the spiking input. In the following figures, $\Delta T = \pm 10$ms was used as a time-window. To recapitulate, this time-window is used to check, whether the DYNAP-SE emitted a spike in order to do the update of the neuron with the highest voltage above threshold. We assume that due to the mismatch, but the general alignment of the DYNAP-SE and the simulation, the neuron with the highest voltage above threshold in the simulation at time t must have spiked within a certain ΔT on the DYNAP-SE. It should be noted that this scheme can be fooled by simply setting the recurrent weights on-chip to zero and activating all the neurons, all the time. This way the condition that a spike occurred within a given time-window is always satisfied and algorithm 2 simply becomes a version of the simulation. However, this would render the results useless, as the spike train does not sparsify and thus does not produce an (almost) optimal code.

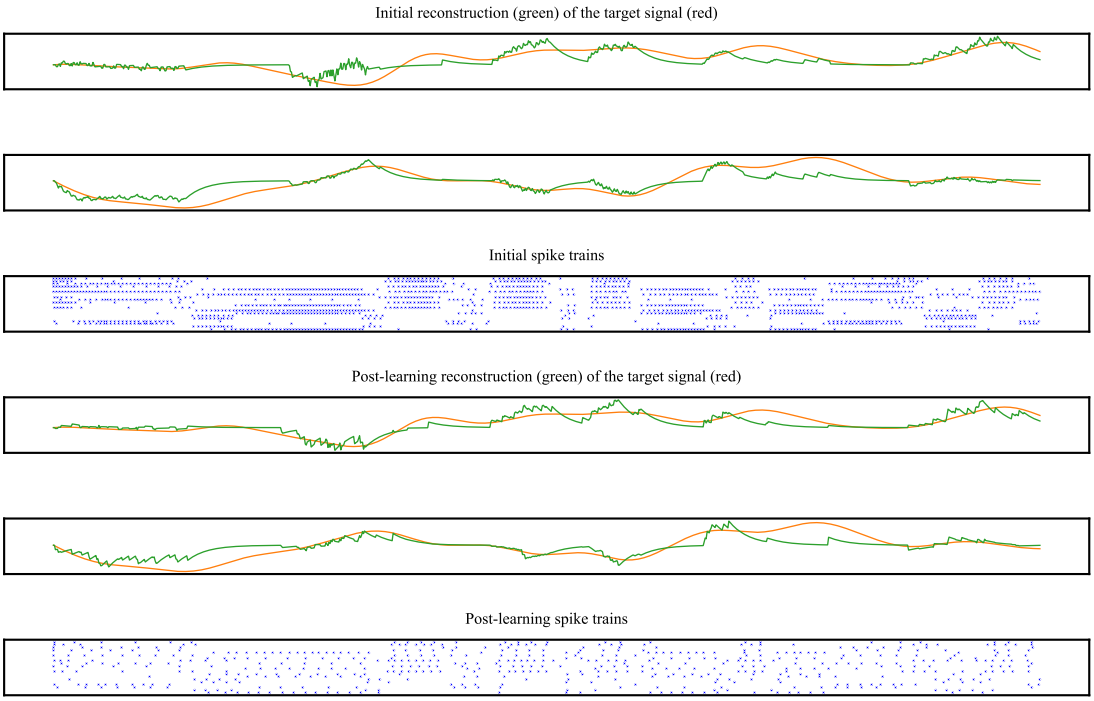

Figure 14: While reducing the reconstruction error, the spike train sparsifies. In fact, for this experiment 880 spikes were saved over a period of 1 second, which is more than 3 million in one hour. A neuron on the DYNAP-SE consumes roughly $1.68nJ$, so $5.04mJ$ were saved for only 20 neurons.

When comparing Figure 14 with Figure 3, especially the scale of the reconstruction error, it becomes clear that a lot of precision is lost, due to the simulation-DYNAP-SE mismatch. Nevertheless, the results show that this setup successfully learns the optimal recurrent matrix, reduces the mean firing rate, reduces the decoding error and balances the network (see Figure 15 for convergence behaviour).

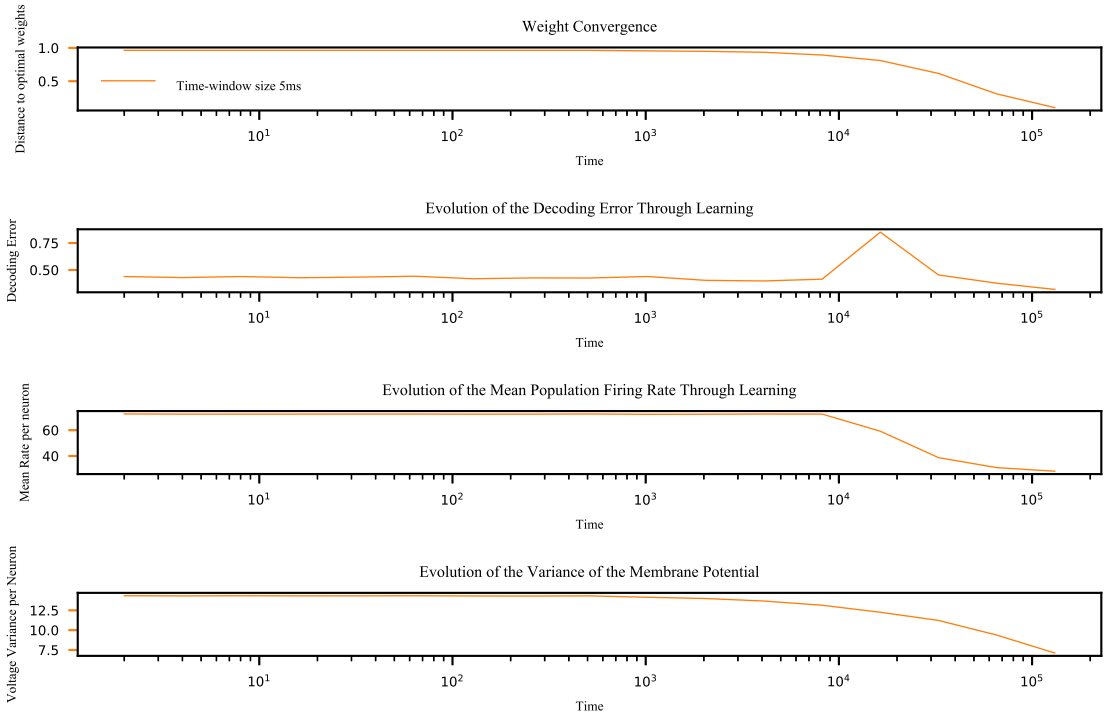

Figure 15: Besides the decoding error, all variables show stable convergence. Despite the fact that overall decoding error is reduced, learning is still unstable and has to be improved using an adaptive learning rate (see section 4.4).

4.4 Reinforcement Learning based step size adaptation

When neural agents interact with their environment, the quality of the internal signal representation is crucial: Imprecise representations might lead to unsuccessful execution of tasks, which can potentially cause harm. It is therefore desirable to adapt the learning mechanism to incorporate abstract rewards, besides the actual learning rules. This is particulary helpful in scenarios where learning is unstable and a reward can help to adapt the step size taken into the direction of the negative gradient. To concretise this idea in the context of Reinforcement Learning (**RL**), the reward would be the negative reconstruction error and the action would be the appropriate rescaling of the learning rate. Recently, Xu et al. (2019) proposed a **RL**-based approach for learning a learning-rate schedule. In this work, the reward is a feature comprising many performance indicators such as validation loss and loss-variance. Reward based adaptation is biologically plausible and allows to use multiple features, enabling the construction of more abstract rewards. To avoid hasty corrections based on short-sighted rewards, we employed a time horizon, which geometrically weighs n rewards from the past (Sutton (1988)). We define the reward to be

$$r(t) = \frac{\sum_{i=0}^{n} \theta_{t-i} \cdot \lambda^{-i}}{\sum_{i=0}^{n} \lambda^{-i}}$$

where λ_i is the individual reward at time i and λ was set to 1. After a "burn-in" phase, the initial reward is calculated and taken to be the baseline. Subsequent rewards are then rescaled w.r.t. the inital reward. Depending on how unstable the convergence is, this reward can be further rescaled by some exponent β, yielding extremely small step sizes. The complete update of the learning rate thus becomes:

$$\alpha_t = \begin{cases} \alpha_{\text{initial}} \cdot \frac{r(t)}{r(n)} & \text{if } \frac{r(t)}{r(n)} \geq 1 \\ \alpha_{\text{initial}} \cdot (\frac{r(t)}{r(n)})^{\beta} & \text{else} \end{cases}$$

The effect of this adaptation scheme is made visible in Figure 16, comparing rewards over time between no adaptation and the reinforcement learning based adaptation in a noisy learning environment.

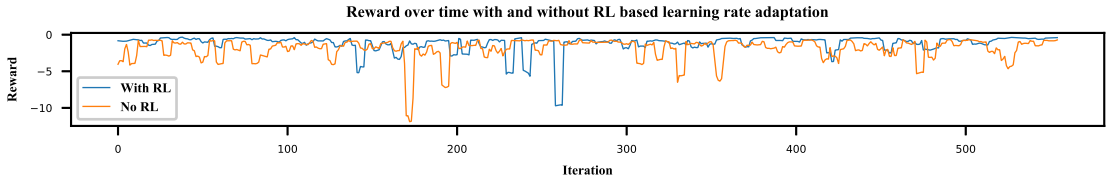

Figure 16: When discretizing the weights, convergence becomes unstable, which can be countered by an adaptive learning rate. While being biologically plausible, the **RL** scheme provides a flexible learning rate, ensuring more stable convergence. One can see that convergence is more stable after approximately 300 iterations.

When applying this scheme to the DYNAP-SE setting, learning becomes more stable as Figure 17 shows.

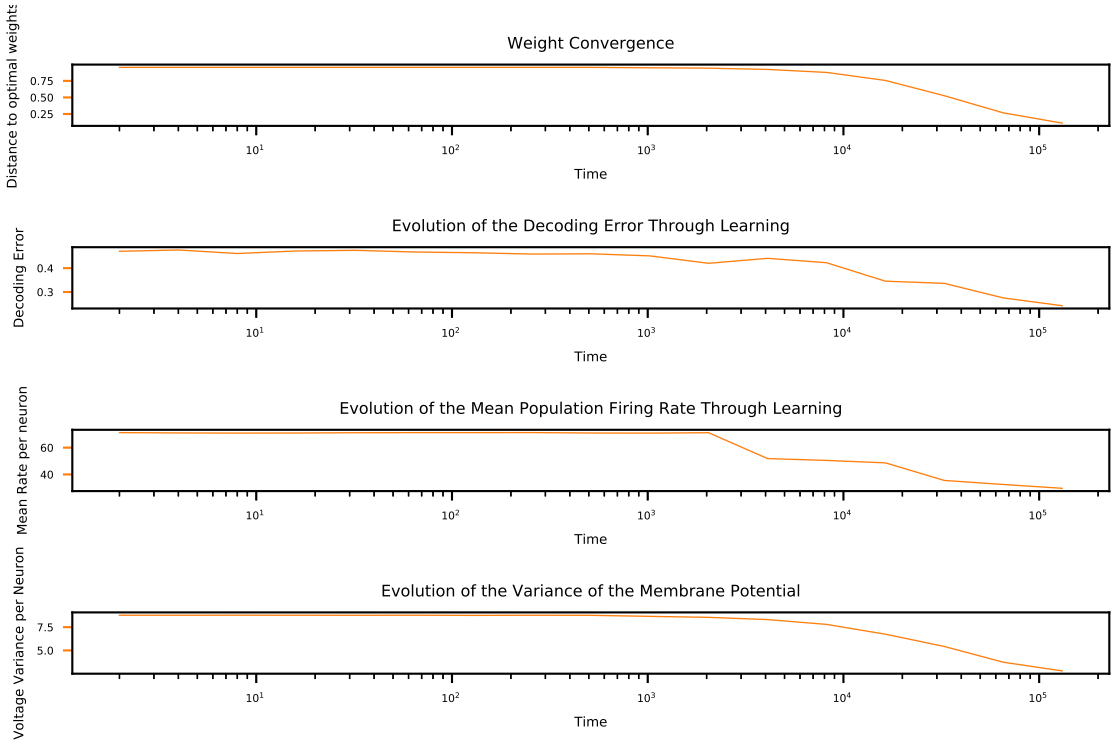

Figure 17: Convergence using the **RL**-based scheme becomes more stable (cf. Figure 15).

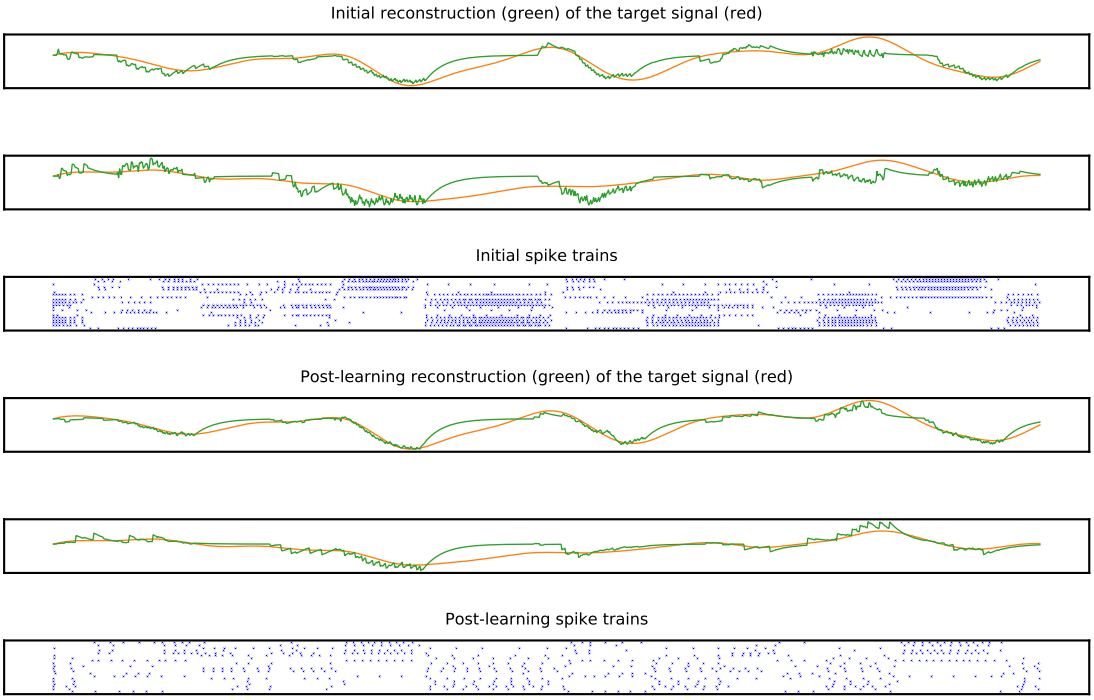

Figure 18: Reconstructed signal using a time-window of $\pm 10ms$ and **RL**-based lernig rate adaptation with $\beta = 10$ and $n = 5$. Compared to Figure 14, one can see that the reconstruction is indeed more precise.

5. Discussion

Acknowledgments

I would like to thank my supervisors Sergio Solinas and Giacomo Indiveri for insightful discussions, showing me valuable papers and teaching me various new concepts. I would also like to thank Dimitrii Zendrikov for helpful discussions and joint efforts in fixing bugs and Christian Machens, who was kind enough to supply me with code and great insights during my stay at his lab in Lisbon. Finally, I would like to thank my Co-Supervisor Onur Mutlu for making this thesis possible by supervising it.

References

Ralph Bourdoukan, David G. T. Barrett, Christian K. Machens, and Sophie Denève. Learning optimal spike-based representations. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2285–2293, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999325.2999390.

Irem Boybat, Manuel Le Gallo, S. R. Nandakumar, Timoleon Moraitis, Thomas Parnell, Tomas Tuma, Bipin Rajendran, Yusuf Leblebici, Abu Sebastian, and Evangelos Eleftheriou. Neuromorphic computing with multi-memristive synapses. *Nature Communications*, 9(1):2514, 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-04933-y. URL https://doi.org/10.1038/s41467-018-04933-y.

Wieland Brendel, Ralph Bourdoukan, Pietro Vertechi, Christian K. Machens, and Sophie Denéve. Learning to represent signals spike by spike, 2017.

Andrew S. Cassidy, Jun Sawada, Paul Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Filipp Akopyan, Bryan L. Jackson, and Dharmendra S. Modha. Truenorth: A high-performance, low-power neurosynaptic processor for multi-sensory perception, action, and cognition. 2016.

F. Corradi and G. Indiveri. A neuromorphic event-based neural recording system for smart brain-machine-interfaces. *IEEE Transactions on Biomedical Circuits and Systems*, 9(5): 699–709, Oct 2015. ISSN 1940-9990. doi: 10.1109/TBCAS.2015.2479256.

H. Dale. Pharmacology and nerve-endings (walter ernest dixon memorial lecture): (section of therapeutics and pharmacology). *Proceedings of the Royal Society of Medicine*, 28(3): 319–332, Jan 1935. ISSN 0035-9157. URL https://www.ncbi.nlm.nih.gov/pubmed/19990108. 19990108[pmid].

Stephen R. Deiss, Rodney J. Douglas, and Adrian M. Whatley. Pulsed neural networks. chapter A Pulse-coded Communications Infrastructure for Neuromorphic Systems, pages 157–178. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-626-13350-4. URL http://dl.acm.org/citation.cfm?id=296533.296540.

Louis Lapicque. *Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation*. 1907. URL https://fr.wikisource.org/wiki/Recherches_quantitatives_sur_l%27excitation_%C3%A9lectrique_des_nerfs_trait%C3%A9e_comme_une_polarisation.

Wolfgang Maass, Thomas Natschlager, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput.*, 14(11):2531–2560, November 2002. ISSN 0899-7667. doi: 10.1162/089976602760407955. URL http://dx.doi.org/10.1162/089976602760407955.

Saber Moradi, Qiao Ning, Fabio Stefanini, and Giacomo Indiveri. A scalable multi-core architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *CoRR*, abs/1708.04198, 2017. URL http://arxiv.org/abs/1708.04198.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988. ISSN 1573-0565. doi: 10.1007/BF00115009. URL https://doi.org/10.1007/BF00115009.

Robert Tibshirani. Regression shrinkage and selection via the lasso. 58(1):267–288, 1996. ISSN 00359246. URL www.jstor.org/stable/2346178.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390294. URL http://doi.acm.org/10.1145/1390156.1390294.

Zhen Xu, Andrew M. Dai, Jonas Kemp, and Luke Metz. Learning an adaptive learning rate schedule, 2019.