

# 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

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Theory: Representing signals spike-by-spike</b>	<b>2</b>
2.1	Introduction . . . . .	2
2.2	Network dynamics . . . . .	2
2.3	The optimal decoder . . . . .	3
2.4	Optimal network connectivity . . . . .	4
2.5	Learning optimal recurrent connectivity . . . . .	5
2.6	Simulation and results . . . . .	6
2.7	Limitations of the theory . . . . .	9
<b>3</b>	<b>Hardware: The DYNAPS chip</b>	<b>11</b>
<b>4</b>	<b>Learning optimal spike-based signal representations on the DYNAPS</b>	<b>11</b>
4.1	Setup: Learning in-the-loop . . . . .	11
4.2	Aligning on- and off chip network dynamics . . . . .	12
4.2.1	Network with spiking input . . . . .	12
4.2.2	The weights on-chip . . . . .	14
4.2.3	Batched updates . . . . .	15
4.2.4	Further alignment using time-window update . . . . .	17
4.2.5	Final pseudo code . . . . .	18
4.3	Results . . . . .	19
4.4	Reinforcement Learning based step size adaptation . . . . .	21
<b>5</b>	<b>Discussion</b>	<b>22</b>

## Abstract

This paper describes the mixtures-of-trees model, a probabilistic model for discrete multidimensional domains. Mixtures-of-trees generalize the probabilistic trees of Chow and Liu (1968) in a different and complementary direction to that of Bayesian networks. We present efficient algorithms for learning mixtures-of-trees models in maximum likelihood and Bayesian frameworks. We also discuss additional efficiencies that can be obtained when data are “sparse,” and we present data structures and algorithms that exploit such sparseness. Experimental results demonstrate the performance of the model for both density estimation and classification. We also discuss the sense in which tree-based classifiers perform an implicit form of feature selection, and demonstrate a resulting insensitivity to irrelevant attributes.

**Keywords:** Bayesian Networks, Mixture Models, Chow-Liu Trees

## 1. Introduction

- Why is it important to represent signals? - What are desirable properties when representing a signal? (fault tolerance (not only a few neurons contribute), cost efficiency, reconstruction error, generalization, adaptation) - Where could this be useful? -Machine learning (transform input to sparse high dimensional vectors and use representation to eg learn, denoise, cluster etc) - Neuro Agents who encounter signals of the same domain very often and want to represent these signals with as few spikes as possible - Links to general thesis that agents want to minimize free energy? Friston?

## 2. Theory: Representing signals spike-by-spike

### 2.1 Introduction

Is tight E/I-balance better than loose? What do the studies suggest? What do rodents or other animals do? What about rate encoding? In the following we derive a learning rule that achieves a tight E/I-balance and minimizes reconstruction error and generates sparse spike trains.

We will show only the learning rules for the recurrent connections.

### 2.2 Network dynamics

In this work, we assume that we have a population of  $N$  neurons into which  $N_x$  continuous 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 (cite LIF). The network dynamics can thus be written as

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

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

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

where we define

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

and

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

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

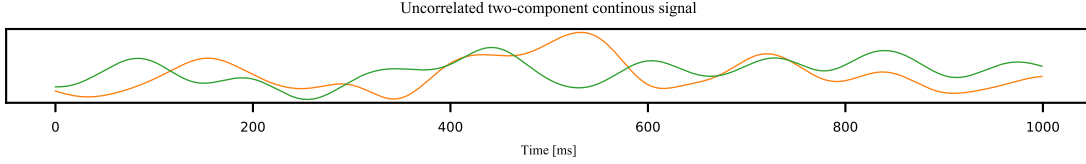


Figure 1: Input  $x$  of dimension  $N_x = 2$  shown over a period of 1000 ms.

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 Dale's law, as it allows for neurons to be excitatory and inhibitory at the same time. However, (cite Brendel) 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

$$\hat{\mathbf{x}} = \mathbf{D}\mathbf{r} \quad (5)$$

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

### 2.3 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.

$$L^* = \min_{\mathbf{r}} \|\mathbf{x} - \mathbf{D}\mathbf{r}\|_2^2 + \mu \|\mathbf{r}\|_2^2 + \nu \|\mathbf{r}\|_1 \quad (6)$$

Because of Equation 3, one can see that enforcing  $l2$ -sparsity allows for dense population codes (since it keeps the rates generally low, but does not enforce 0 values). In contrast,  $l1$ -sparsity encourages 0 values in the rates, which corresponds to sparse spike trains (cite spars.). This loss function gives us an explicit formula for the optimal decoder that is achievable by the network. By taking the derivative of Equation 6 with respect to  $\mathbf{D}$  and setting it to  $\mathbf{0}$ , we get

$$\mathbf{D}^* = \mathbf{x}\mathbf{r}^T(\mathbf{r}\mathbf{r}^T)^{-1} \quad (7)$$

## 2.4 Optimal network connectivity

Following (cite Brendel), 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 (cite Learning optimal spike-based representations) 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

$$L(\text{neuron } n \text{ spikes}) < L(\text{neuron } n \text{ does not spike}) \quad (8)$$

Using Equation 6,  $L(\text{neuron } n \text{ spikes})$  can be written as

$$L(\text{neuron } n \text{ spikes}) = \|c - Dr - D_n\|_2^2 + \mu\|r + e_n\|_2^2 + \nu\|r + e_n\|_1 \quad (9)$$

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

We observe that

$$\|r + e_n\|_2^2 = \left(\sum_{i, i \neq n} r_i\right)^2 + (r_n + 1)^2 = \|r\|_2^2 + 2r_n + 1 \quad (10)$$

and

$$\nu\|r + e_n\|_1 = \nu\left(\sum_{i, i \neq n} |r_i| + |r_n + 1|\right) =^* \nu\left(\sum_{i, i \neq n} |r_i| + |r_n| + 1\right) = \nu\|r\|_1 + \nu \quad (11)$$

where  $*$  holds because  $\forall t \ r(t) \geq 0$ . Using Equations 10 and 11 we can rewrite Equation 8 to

$$D_n^T c - D_n^T Dr - \mu r_n > \frac{1}{2}(\|D_n\|_2^2 + \mu + \nu) \quad (12)$$

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

$$V_n(t) = D_n^T c(t) - D_n^T Dr(t) - \mu r_n \quad (13)$$

and the threshold of the  $n$ -th neuron as

$$T_n = \frac{1}{2}(\|D_n\|_2^2 + \mu + \nu) \quad (14)$$

using the differential equations 4 and 3, as well as Equation 1 we rewrite Equation 13 to

$$V(t) = \mathbf{F}^T x(t) + \mathbf{\Omega} o(t) \quad (15)$$

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

We also note that since  $\mathbf{\Omega} = -\mathbf{D}^T \mathbf{D} - \mu \mathbf{I}$ , 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 14 can be seen as a constraint on the decoder length for a neuron by simply rewriting it to

$$\|D_n\|_2^2 = 2T_n - \mu - \nu \quad (16)$$

## 2.5 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, as discussed in section (ref section). 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.

$$\begin{aligned} \epsilon_n &= F_n^T \mathbf{x} - F_n^T \hat{\mathbf{x}} \\ &= F_n^T - F_n^T D \mathbf{r} \\ &\approx 0 \end{aligned} \quad (17)$$

We furthermore observe, using the optimal feed-forward and recurrent connections, that Equation 17 corresponds to the network dynamics obtained in Equation 2. We can conclude that the signal is represented optimally if  $\Omega_n = -F_n^T D$ ,  $\epsilon_n = V_n \approx 0$  and  $D = D^*$ . 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:

$$L(t_j) = \left\| \frac{1}{2}(V^{\text{before}}(t_j) + V^{\text{after}}(t_j)) \right\|_2^2 \quad (18)$$

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:

$$V^{\text{after}} = V^{\text{before}} + \mathbf{\Omega} e_{k(j)} \quad (19)$$

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 19 to

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

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

$$\Delta \Omega_{n,k} = -2V_n^{\text{before}} - \Omega_{n,k} \quad (20)$$

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  $l_2$ -cost on the rates,  $\Omega$  should ideally converge towards  $-\mathbf{F}^T \mathbf{D} - \mu \mathbf{I}$ . This causes a minor change in the learning rule for the recurrent weights:

$$\Delta \Omega_{n,k} = -2(V_n^{\text{before}} + \mu r_n) - \Omega_{n,k} - \mu \delta_{n,k} \quad (21)$$

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

## 2.6 Simulation and results

Learning the feed-forward connections happens on a much slower timescale and thus requires far more training iterations. Since the DYNAPS operates on a much slower, biologically more plausible timescale (cite DYNAPS), 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 (cite Brendel) and shows the learning procedure.

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

```

 $T \leftarrow N_{\text{iter}} \cdot N_{\text{time}}$ 
 $\text{thresh} \leftarrow 0.5$ 
for  $i \leftarrow 0$  to  $T$  do
  if  $i \bmod N_{\text{time}} = 0$  then // Generate new input
     $\text{Input} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
     $\text{Input} \leftarrow A \cdot (\text{Input} * w)$ 
  end
   $t \leftarrow (i \bmod N_{\text{time}}) + 1$ 
   $V_t \leftarrow (1 - \lambda dt) \cdot V_{t-1} + dt \cdot \mathbf{F}^T \text{Input}_t + \Omega o_{t-1} + \epsilon \cdot \mathcal{N}(0, 1)$ 
   $x_t \leftarrow (1 - \lambda dt) \cdot x_{t-1} + dt \cdot \text{Input}_t$ 
   $(\max, k) \leftarrow \arg \max(V_t - \text{thresh} - 0.01 \cdot \mathcal{N}(0, 1))$ 
  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. The used parameters can be found in the Appendix.  $\Omega_k$  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  $\Omega$  and  $\Omega^* = -\mathbf{F}^T \mathbf{F}$ , 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(ii) 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 8, contributes to reducing the global reconstruction error. This way, the resulting spike train sparsifies, which can be observed in Figure 2(iv).

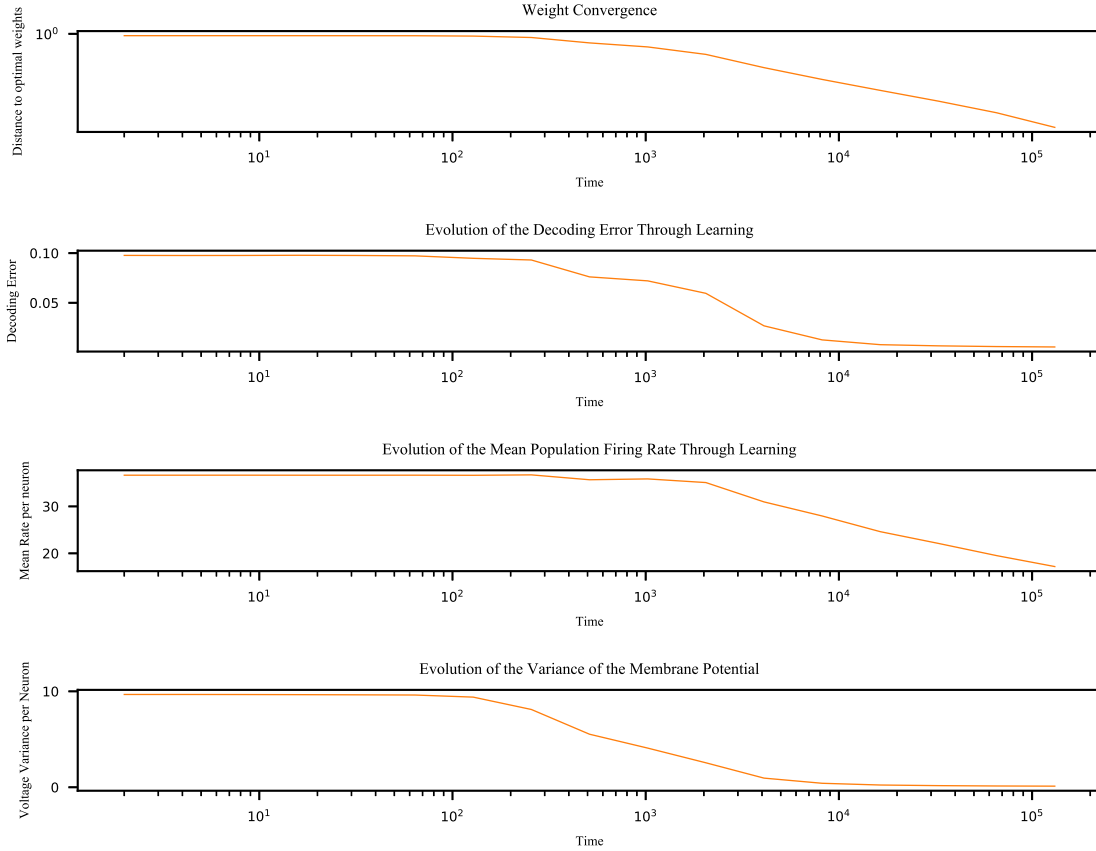


Figure 2: Multiple figures in here: (i) Convergence of optimal weights (ii) Convergence of membrane variance (iii) Convergence of error (iv) Convergence of firing rate

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  $\Omega_{\text{initial}}$  and for the network using  $\Omega_{\text{post learning}}$ . 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  $\Omega_{\text{initial}}$  produced a poor reconstruction compared to the network using  $\Omega_{\text{post learning}}$ . 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.

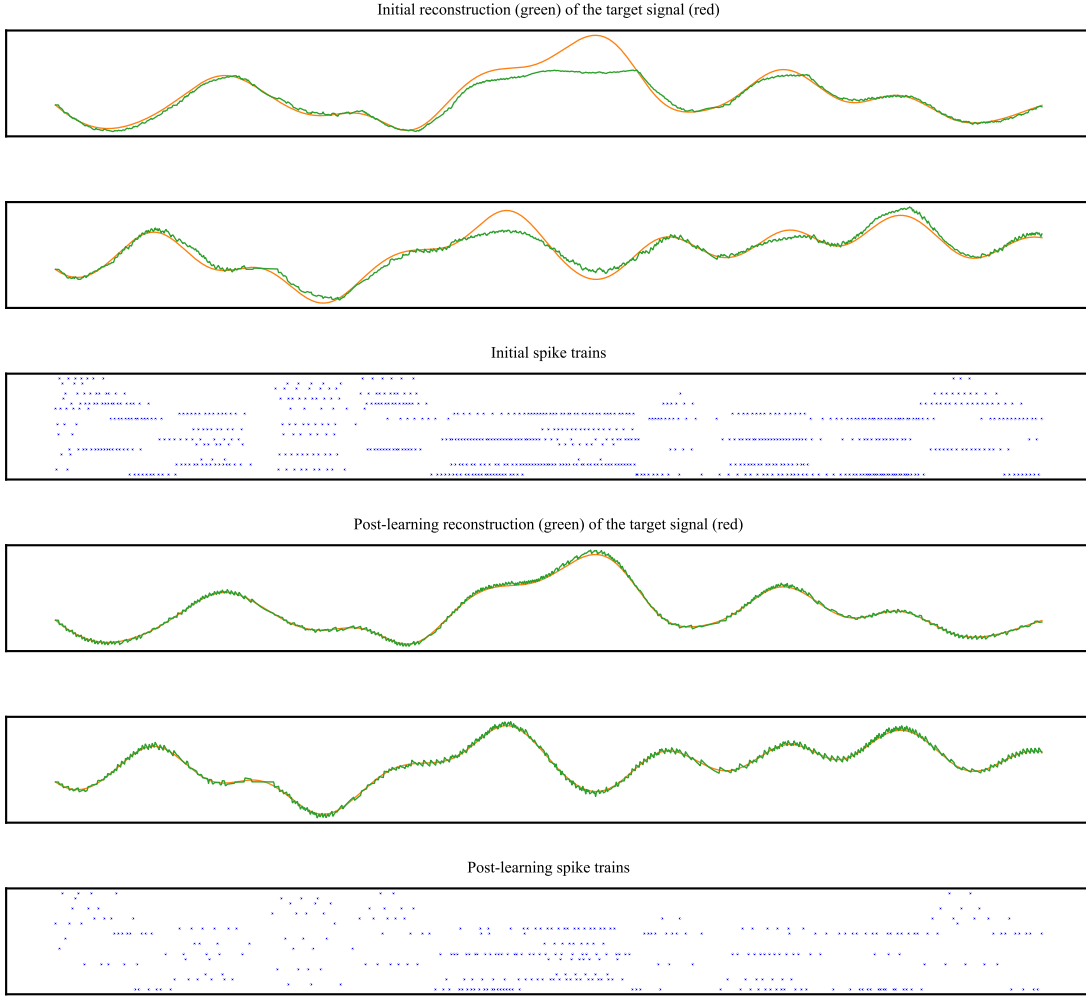


Figure 3: Multiple figures in here: (i) Convergence of optimal weights (ii) Convergence of membrane variance (iii) Convergence of error (iv) Convergence of firing rate



## 2.7 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.

In terms of biological plausibility, the first issue is far more severe than the second one. It is well known that, although microcircuits can show dense connections (cite nature2016), the weight resolution in the brain is limited to only a few bits (need citation). In this theory, tight balance of the input currents is of crucial importance. When discretizing the weights, this balance can no longer be achieved and therefore the network seems to fire uncontrollably, as can be seen in Figure 4. Naturally, this uncontrolled firing causes high reconstruction error and the learned matrix becomes irrelevant (Figure 5).

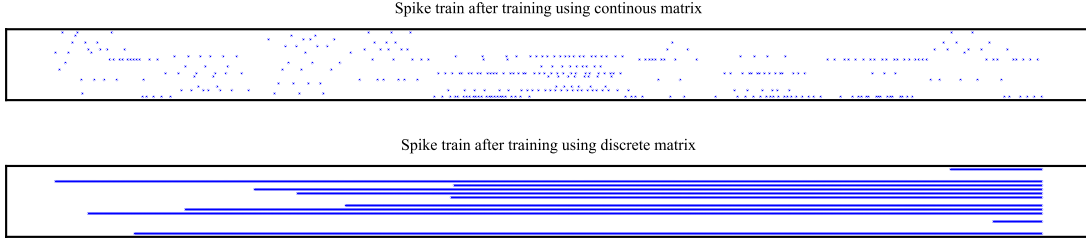


Figure 4: The spike train after training using continous weights (top), exhibits a usable, distributed pattern, while the learned, discretized weight matrix (bottom) shows a subset of neurons firing at a high frequency.

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 DYNAPS (cite DYNAPS), time is discretized and therefore the event that two neurons fire at the same time *can* happen. When adapting Equation 19 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 18. However, the derivation of the optimal network connectivity in Equation 12 falsely assumes only one spike per time.

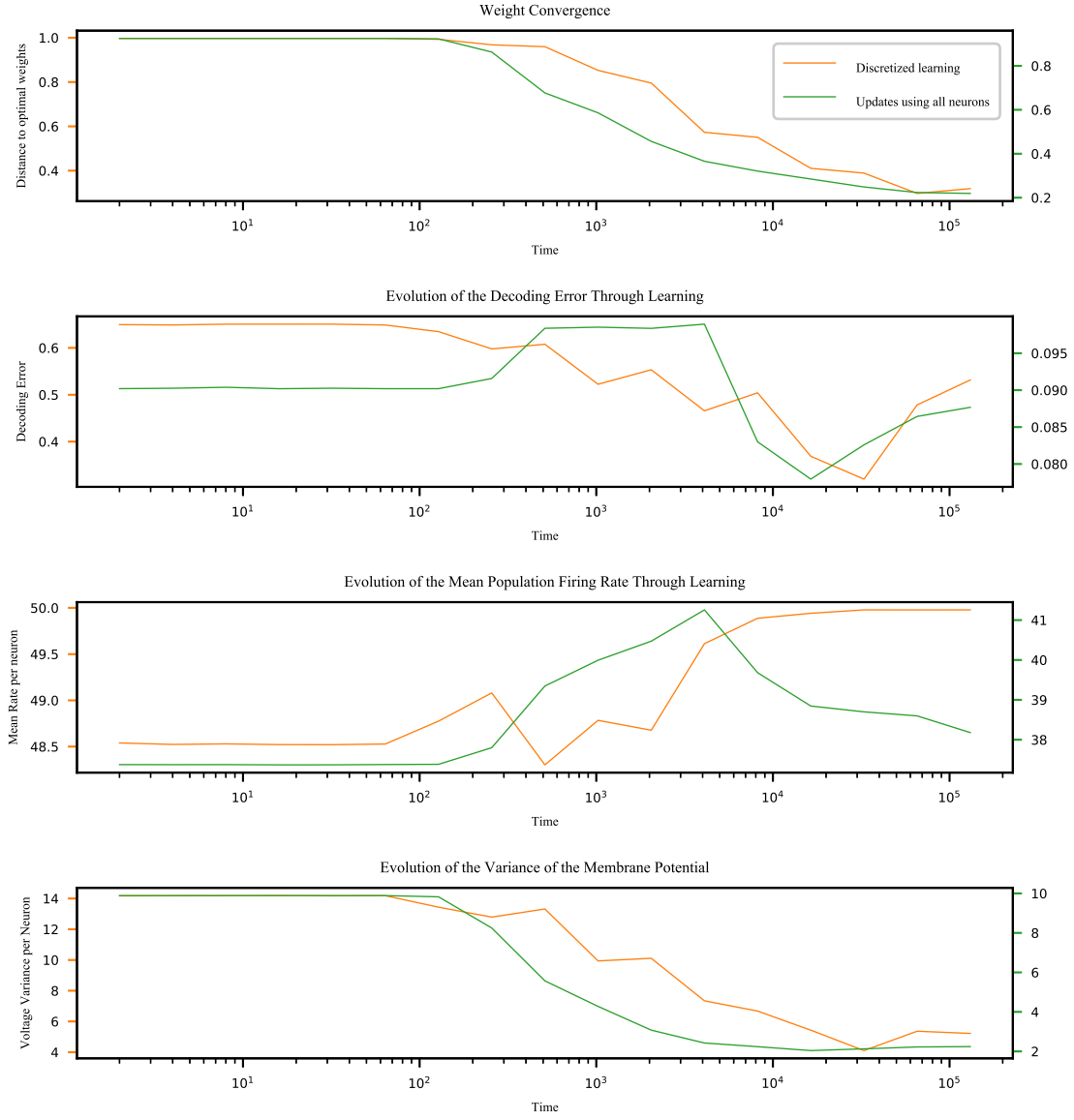


Figure 5: Convergence of different properties when learning with discrete weights (orange) and performing the updates with respect to all spiking neurons (green). It should be noted that the mean firing rate is really high compared to the successful learning (see Figure 2).

### 3. Hardware: The DYNAPS chip

## 4. Learning optimal spike-based signal representations on the DYNAPS

### 4.1 Setup: Learning in-the-loop

Although spike-timing dependent plasticity is implemented on-chip (citation), this learning algorithm requires an in-the-loop setup. The reason for this was already pointed out in section 2.7: 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 continuous 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 6). For a more detailed description of the program see section 4.2.5.

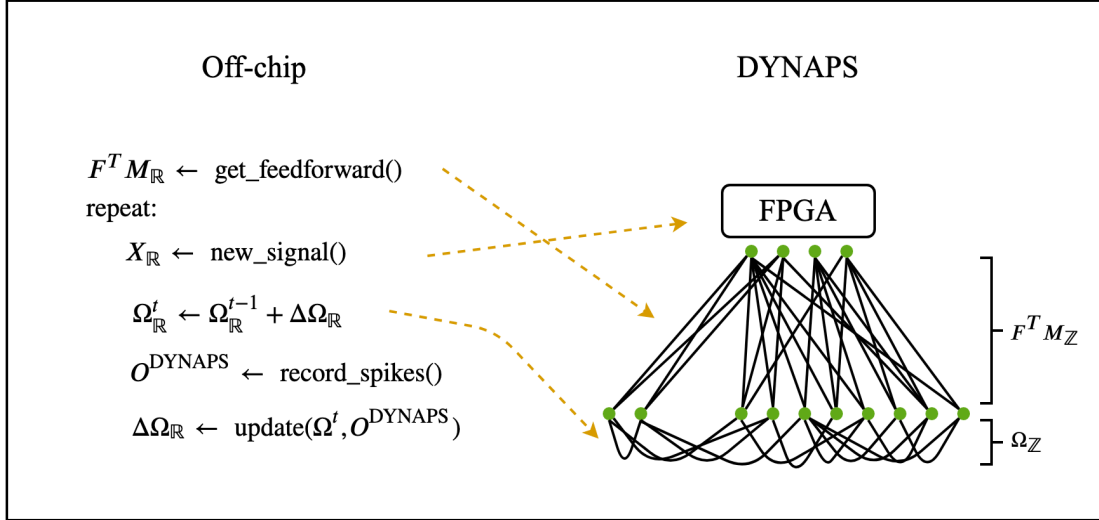


Figure 6: The feed-forward connections  $\mathbf{F}^T \mathbf{M}$  (derived in section 4.2.1) are initially discretized to  $\mathbb{Z}$  and loaded onto the chip. The following four steps are executed repeatedly for a fixed number of iterations: A new signal is generated and stored in the FPGA memory, the recurrent weights  $\Omega$  are updated, discretized to  $\mathbb{Z}$  and loaded onto the chip, the DYNAPS runs on the spiking input and finally  $\Delta\Omega$  is computed using the spike trains  $\mathbf{O}^{\text{DYNAPS}}$  recorded from the DYNAPS.

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

## 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 DYNAPS also receives input in the form of spikes, which raises the question on how to convert a continuous signal to spikes. One simple approach called Delta Modulation (citationHere), 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 7 illustrates the spikes of a 1D-signal using 0.05 as a spiking threshold in both directions.

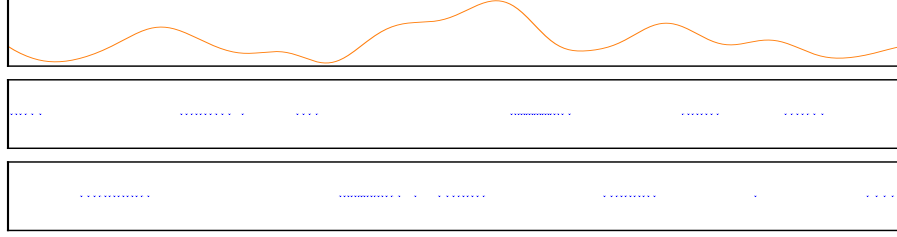


Figure 7: The signal (top) is split into two channels, representing the (upward) downward magnitude of the signal in the number of spikes.

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

$$\text{Input} = \mathbf{F}^T x \quad (22)$$

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

$$\hat{x}_t = (1 - \lambda)\hat{x}_{t-1} + \nu(o_t^{up} - o_t^{down}) \quad (23)$$

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

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

the general form of Equation 23 becomes

$$\hat{x}_t = (1 - \lambda)\hat{x}_{t-1} + \nu M o_t \quad (24)$$

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

$$\text{Input} = \mathbf{F}^T \hat{x} = \mathbf{F}^T ((1 - \lambda)\hat{x}_{t-1} + M o_{t-1}) \quad (25)$$

This however does not match the setup on the DYNAPS since the input is first transformed using  $M$ , then integrated and then again transformed using  $F$ . Contrary, the synapses on the DYNAPS 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

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

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 9 compares the reconstruction error over time between the continuous-input network and the spiking-input network. Since the DYNAPS 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 severely by the weight resolution on the DYNAPS, which will be covered in another section.

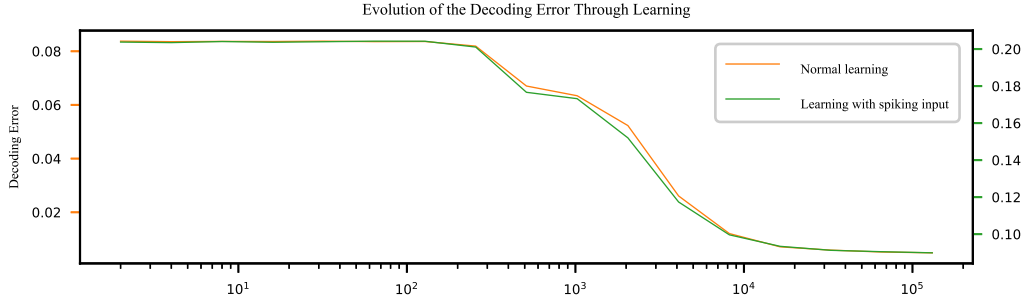


Figure 8: While the error convergence of the spiking-input and continuous-input network matches, the error of the spiking-input network is increased by roughly a factor of 10, due to the loss of precision imposed by the spikes (note the green scale).

#### 4.2.2 THE WEIGHTS ON-CHIP

As pointed out in section 3, the fan-in of the DYNAPS 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 XX compares the continuous and discretized weight matrices post learning.



Figure 9: Caption here!

In order to understand the process of rescaling the weights, it is important to know that the DYNAPS 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 separate 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 continuous 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 magnitude, 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 DYNAPS 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  $\Delta T$  and iteratively present, by  $\Delta T$  increasing, parts of the signal and only do an update in the last  $\Delta T$ , when there was a spike. Since a lot of overhead is created when setting the connections on the DYNAPS, 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 10.

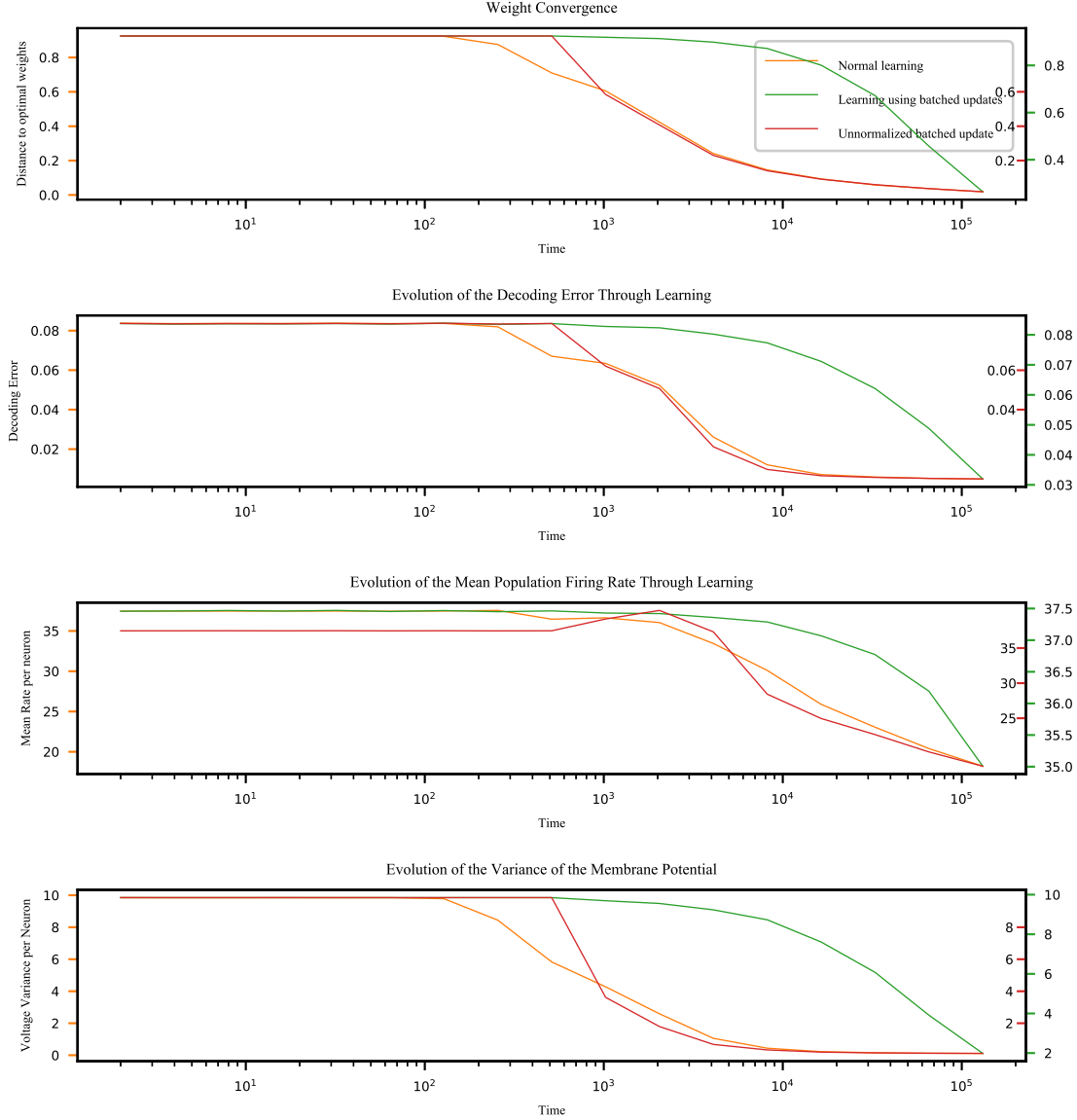


Figure 10: Caption here!



## 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  $o^{DYNAPS}(t-1)$ . Upon comparison of the reconstructed voltage and the simulation voltage, 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 DYNAPS, occasionally multiple neurons spiked simultaneously, which we did not take into account. To resolve this mismatch, we computed the reconstructed voltage using

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

where  $o_k^{aligned}(t)$  is one if  $V_k(t-1)$  was the maximum value above threshold and the DYNAPS 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  $N_{\text{iter}}$   
**Output:** Learned recurrent matrices  $\Omega, \Omega^{\mathbb{Z}}$

```

 $thresh \leftarrow 0.5$ 
 $T_{\text{delta modulator up}} \leftarrow 0.05$ 
 $T_{\text{delta modulator down}} \leftarrow 0.05$ 
 $\Delta T \leftarrow 5ms$ 
 $\Omega_{\min} \leftarrow -0.5$ 
 $\Omega_{\max} \leftarrow 0.5$ 
 $\mathbf{F} \leftarrow \text{normalize}(\frac{1}{2}\mathcal{N}(\mathbf{0}, \mathbf{I}))$ 
 $\mathbf{F}^T \mathbf{M}^{\mathbb{Z}} \leftarrow \text{discretize}(\mathbf{F}^T \mathbf{M})$ 
dynaps.set_feedforward( $\mathbf{F}^T \mathbf{M}^{\mathbb{Z}}$ )
for  $i \leftarrow 0$  to  $N_{\text{iter}}$  do
  Input  $\leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
  Input  $\leftarrow A \cdot (\text{Input} * w)$ 
  for  $t \leftarrow 1$  to  $T_{\text{signal}}$  do
     $X_t \leftarrow (1 - \lambda dt) \cdot X_{t-1} + dt \cdot \text{Input}_t$ 
  end
  fpga.load_signal( $X_t$ )
   $\Omega_t \leftarrow \Omega_{t-1} - \Delta \Omega$ 
   $\Omega_t^{\mathbb{Z}} \leftarrow \text{discretize}(\Omega_t, \Omega_{\min}, \Omega_{\max})$ 
  dynaps.set_recurrent_connections( $\Omega_t^{\mathbb{Z}}$ )
   $\mathbf{O}^{DYNAPS} \leftarrow \text{dynaps.run}()$ 
  for  $t \leftarrow 1$  to  $T_{\text{signal}}$  do
     $V_t \leftarrow (1 - \lambda dt) \cdot V_{t-1} + dt \cdot \mathbf{F}^T \text{Input}_t + \Omega o_{t-1}^{\text{aligned}} + \epsilon \mathcal{N}(0, 1)$ 
     $(\max, k) \leftarrow \arg \max(V_t - thresh - 0.01 \cdot \mathcal{N}(0, 1))$ 
    if  $\max \geq 0 \wedge \text{has\_spike}(\mathbf{O}_{k, \pm \Delta T}^{DYNAPS})$  then
       $\Delta \Omega_k \leftarrow \Delta \Omega_k + \epsilon_{\Omega} \cdot (\beta \cdot (V_t + \mu \cdot r_{t-1}) + \Omega_k + \mu \cdot \mathbf{I})$ 
       $o_t^{\text{aligned}} \leftarrow I_k$ 
       $r_t \leftarrow r_{t-1} + I_k$ 
    else
       $o_t^{\text{aligned}} \leftarrow \mathbf{0}$ 
       $r_t \leftarrow r_{t-1}$ 
    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  $N_{\text{iter}} = 270$  iterations with  $N = 20$  neurons and  $N_x = 2$  input signals. These signals were converted to  $N_{\text{in}} = 4$  channels, representing the spiking input. In the following figures,  $\Delta T = \pm 10\text{ms}$  was used as a time-window. To recapitulate, this time-window is used to check, whether the DYNAPS 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 DYNAPS and the simulation, the neuron with the highest voltage above threshold in the simulation at time  $t$  must have spiked within a certain  $\Delta T$  on the DYNAPS. 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 the 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.

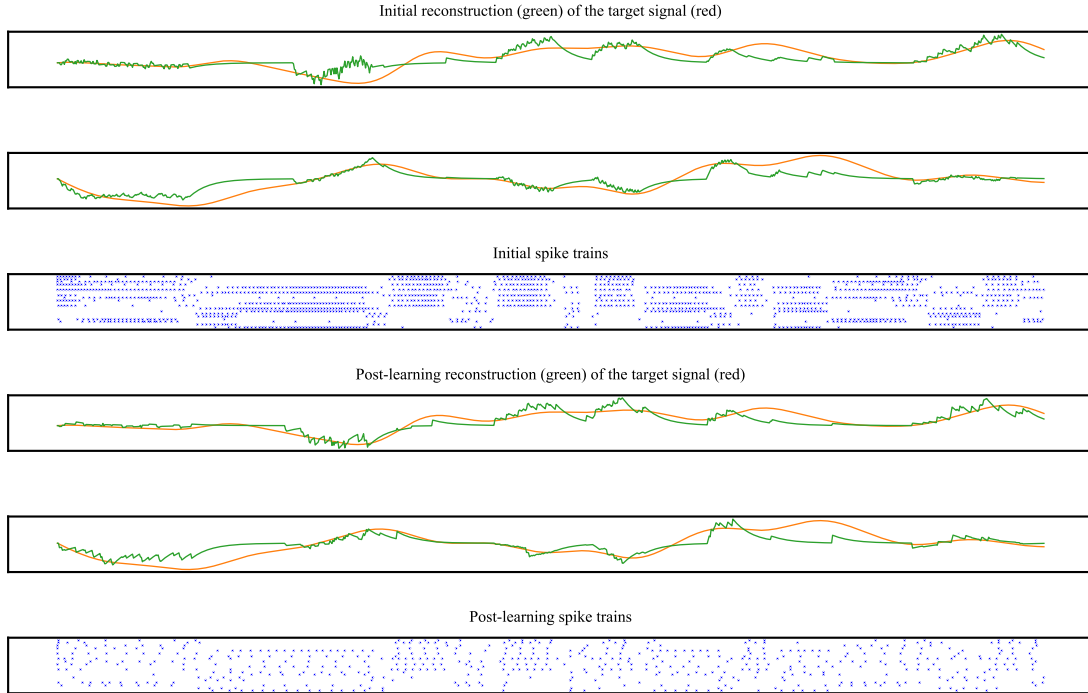


Figure 11: While reducing the reconstruction error, the spike train sparsifies. In fact, for this experiment 880 spikes were saved over a period of 1 second, which is more than 3 million in one hour. Assuming that a neuron consumes  $2.468 \cdot 10^J$ , we saved  $0.781J$ , only for 20 neurons.

When comparing Figure 11 with Figure 3, especially the scale of the reconstruction error, it becomes clear that a lot of precision is lost, due to the simulation-DYNAPS 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 12 for convergence behaviour).

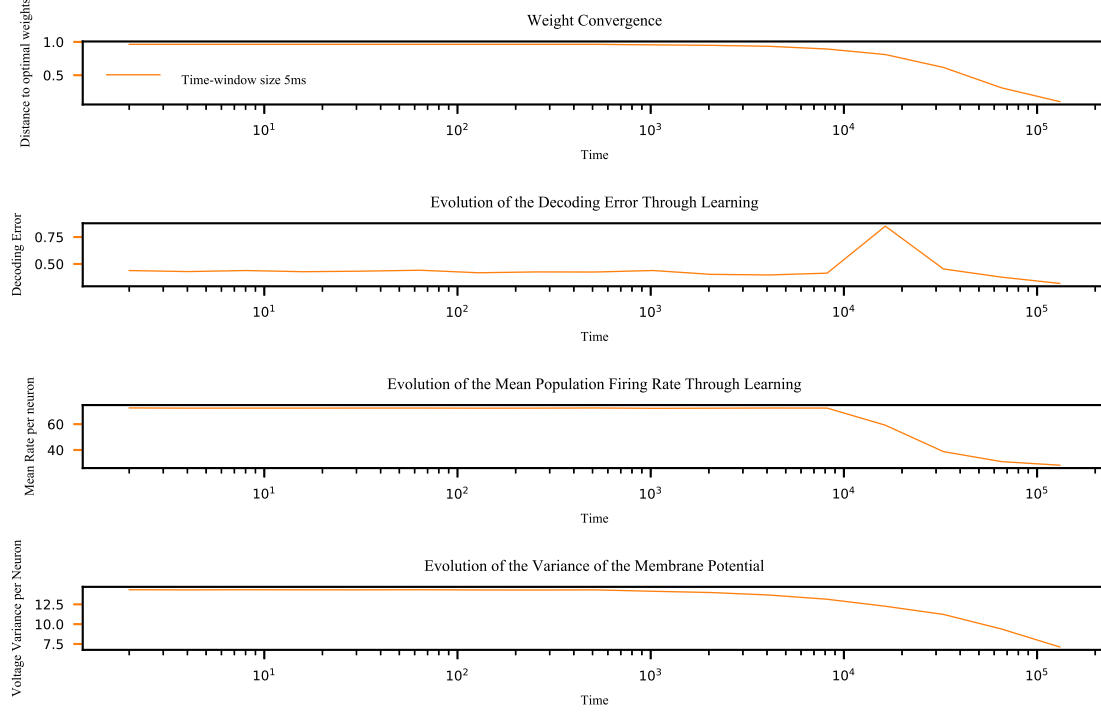


Figure 12: Please reproduce figure with correct size!

#### 4.4 Reinforcement Learning based step size adaptation

When neural agents interact in their environment, the quality of the internal signal representation is crucial. Imprecise representations might lead to unsuccessful execution of tasks such as reacting to various types of stimuli. It therefore makes sense that the learning of spike-based signal representations should be reward guided. In traditional Machine Learning, many approaches for learning a learning rate schedule have been proposed. Recently, Xu et al. (2019) proposed learning the schedule based on reward, composed of different features. Xu et al. (2019) were able to show that this learning adaptation scheme outperformed standard adaptation techniques. Despite the fact that the proposed scheme even improves performance, reward based learning rate adaptation is more biologically plausible. It seems hard to imagine that the brain performs learning rate reduction after a given number of steps. Furthermore, reward based adaptation allows to use multiple features for the reward and weigh them differently. Additionally, more abstract rewards can be used such as the successful identification of a spoken word. In the typical Reinforcement Learning framework, our reward is the reconstruction error on some unseen signal and the action is a rescaling of the initial learning rate. To avoid hasty corrections based on short-sighted rewards, a time horizon is employed, geometrically weighting  $n$  rewards from the past. 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  $\lambda_i$  is the individual reward at time  $i$  and  $\lambda$  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 initial reward. Depending on how unstable the convergence is, this rescaled reward can be further rescaled by some exponent  $\beta$ , yielding extremely small step sizes in case of reward. The complete update of the learning rate thus becomes:

$$\alpha_t = \begin{cases} \alpha_{\text{initial}} \cdot \frac{r(t)}{r(n)} & \text{if } \frac{r(t)}{r(0)} \geq 1 \\ \alpha_{\text{initial}} \cdot \left(\frac{r(t)}{r(n)}\right)^\beta & \text{else} \end{cases}$$

The effect of this adaptation scheme is made visible in Figure 13, comparing rewards over time between no adaptation and the reinforcement learning based adaptation.



Figure 13: Please reproduce figure with correct size!

When applying this scheme to the DYNAPS setting, learning becomes more stable and as Figure 14 shows.

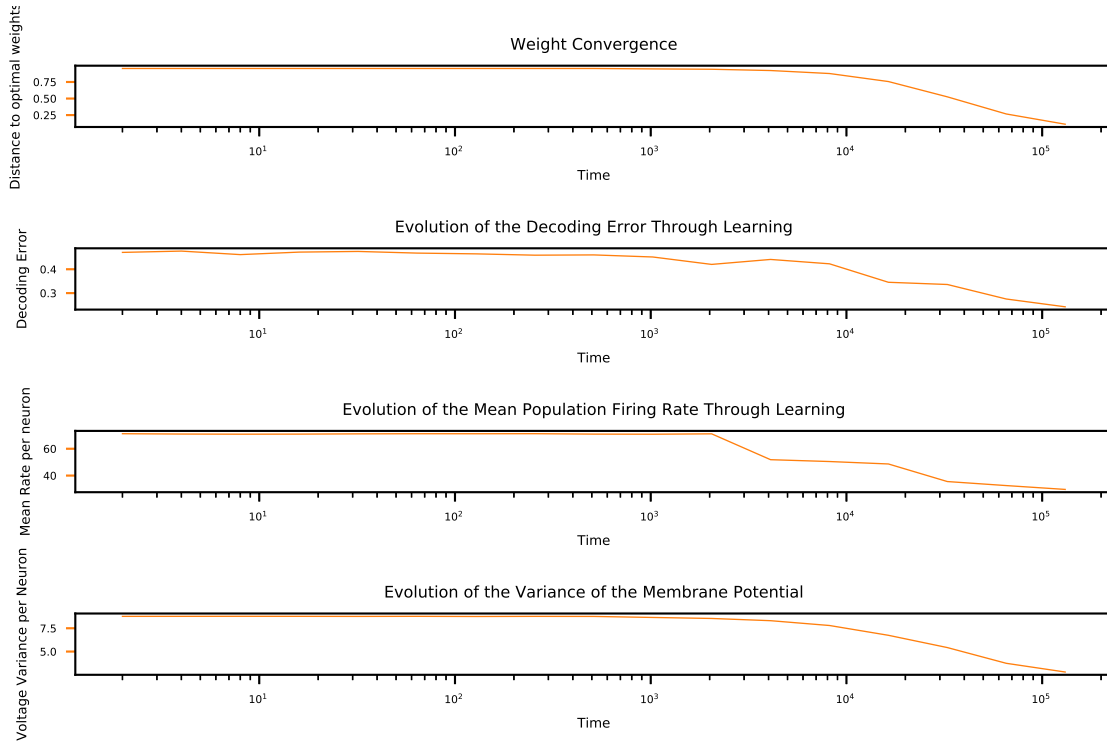


Figure 14: Please reproduce figure with correct size!

## 5. Discussion

### Acknowledgments

We would like to acknowledge support for this project from the National Science Foundation (NSF grant IIS-9988642) and the Multidisciplinary Research Program of the Department of Defense (MURI N00014-00-1-0637).

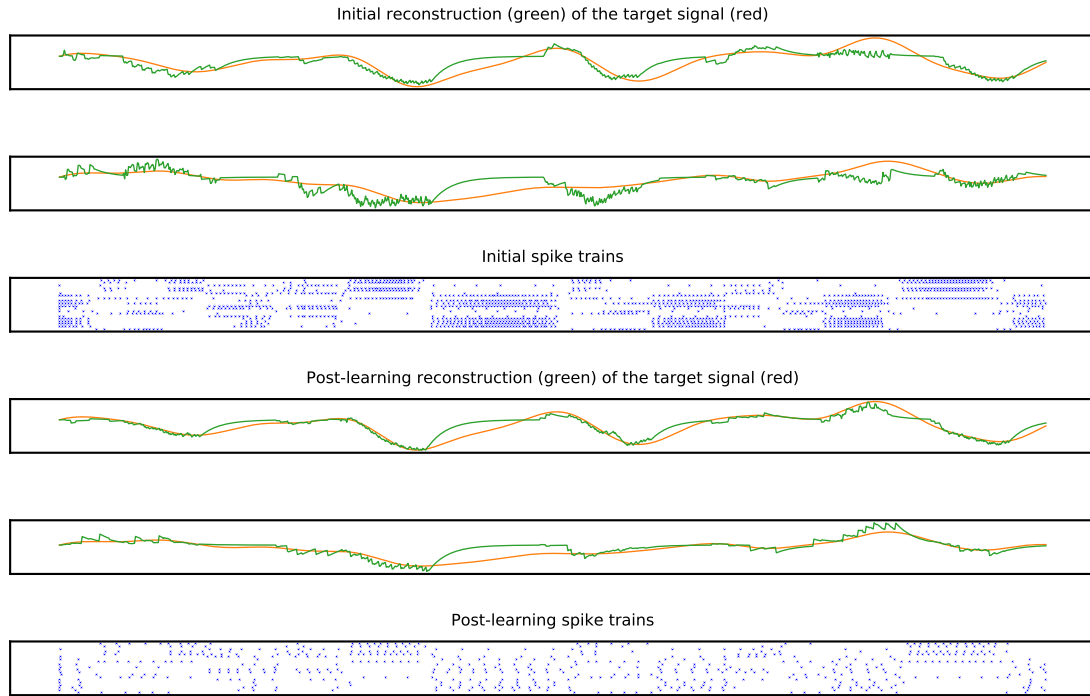


Figure 15: Please reproduce figure with correct size!

## Appendix A.

### References

- C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.
- Zhen Xu, Andrew M. Dai, Jonas Kemp, and Luke Metz. Learning an adaptive learning rate schedule, 2019.