Optimal spike-based signal representation on a neuromorphic chip

institute of neuroinformatics

Outline

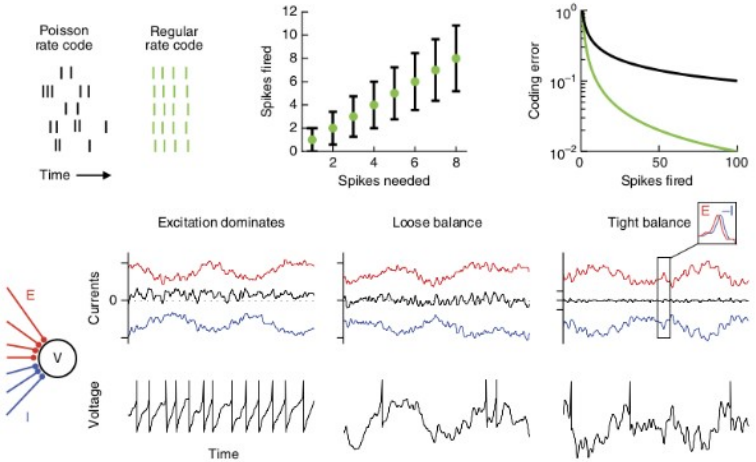
- Introduction & Problem statement
- Goals of the thesis
- The DYNAP-SE
- Learning in-the-loop
- Experiments & Discussion

Signal representation is a fundamental problem

- Unclear how population of neurons represent a signal *effectively* with usable precision
- Coding error using Poisson rate codes scales with $\frac{1}{\sqrt{M}}$
- Representing multiple signals requires many spikes $\rightarrow$ waste of energy (caused by irregularity of Poisson codes)
- Goal: Learning rule driving network in state where
  - Coding error scales with $\frac{1}{M}$
  - Code is *robust* to noise
  - Code is *energy efficient*

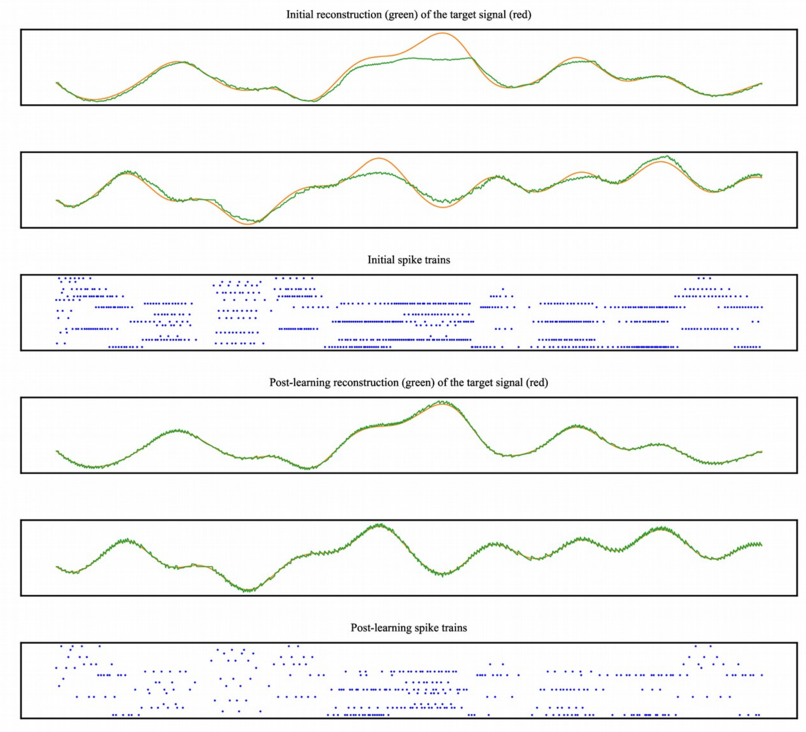
Possible solution: Tightly balanced networks

- Balance achieved by matching excitatory with inhibitory input (E/I-balance)
- Balanced networks produce Poisson-like codes
- Tight balance: Small fluctuations are matched (e.g. excitation followed by inhibition)



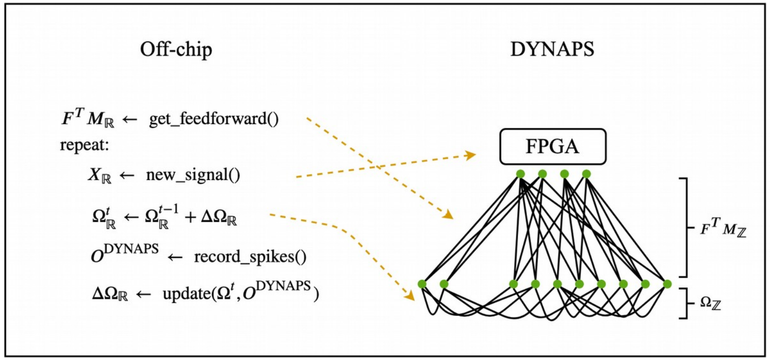
Learning to represent signals spike by spike

- Key insights/contributions:
  - Neural circuits behave accurately and every spike "counts"
  - Derivation of local online learning rule:
    - Condition for spike: Neuron reduces reconstruction error (*error-driven coding*) → Derivation of optimal weights
    - Minimizing voltage deviations → reaching optimal weights → low reconstruction error



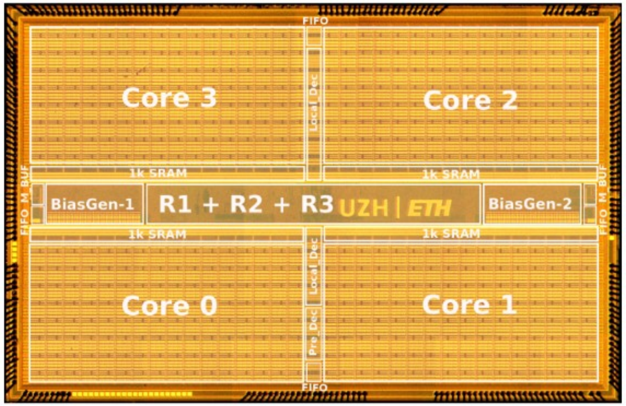
Goals of the thesis

- Implement this learning rule on neuromorphic chip (DYNAP-SE) in-the-loop
- Lay foundation to implement directly on-chip
- Identify possible weaknesses and suggest improvements

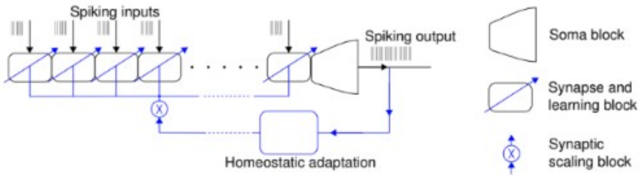


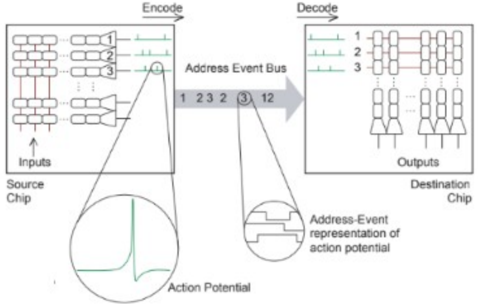
DYNAP-SE

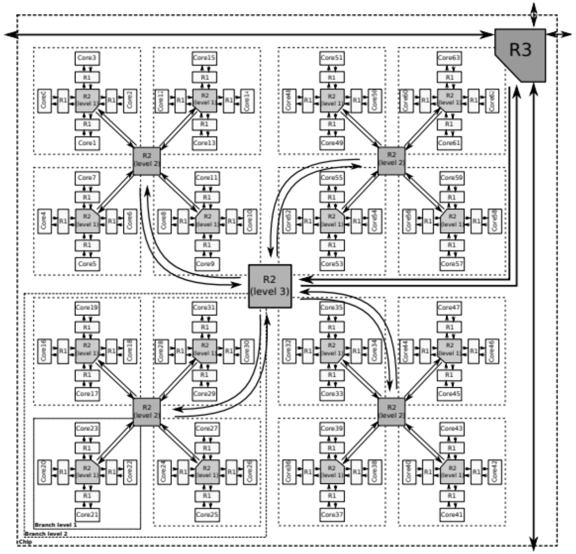
- Mixed analog/digital circuits
- Completely asynchronous (no central clock)
- Ultra low power consumption e.g. 276uA @ 1.3V with all neurons firing at 30Hz (CPU uses 30W @14V = 2.1A)
- Fan-in: 64 | Fan-out: 4064
- 256 silicon neurons/core, 64 synapses/neuron
- Neuron model: Adaptive-Exponential I&F



DYNAP-SE - Routing

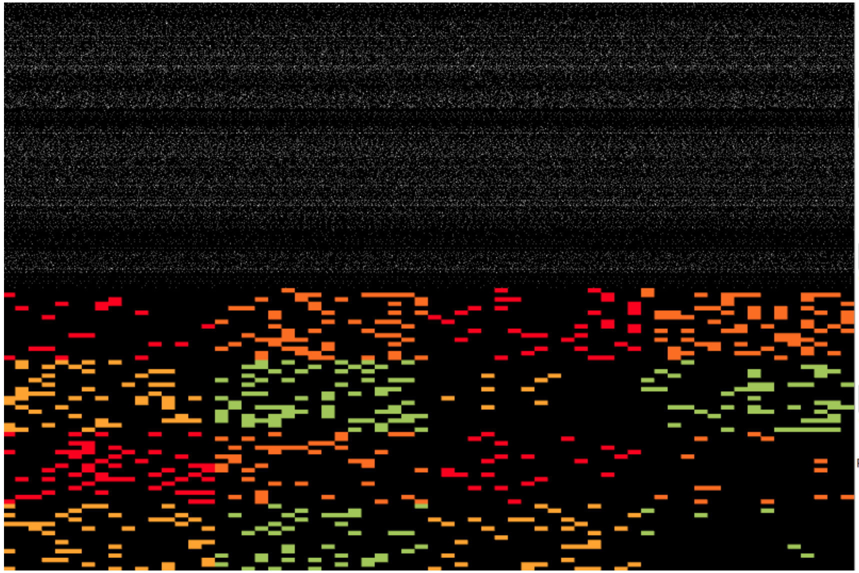


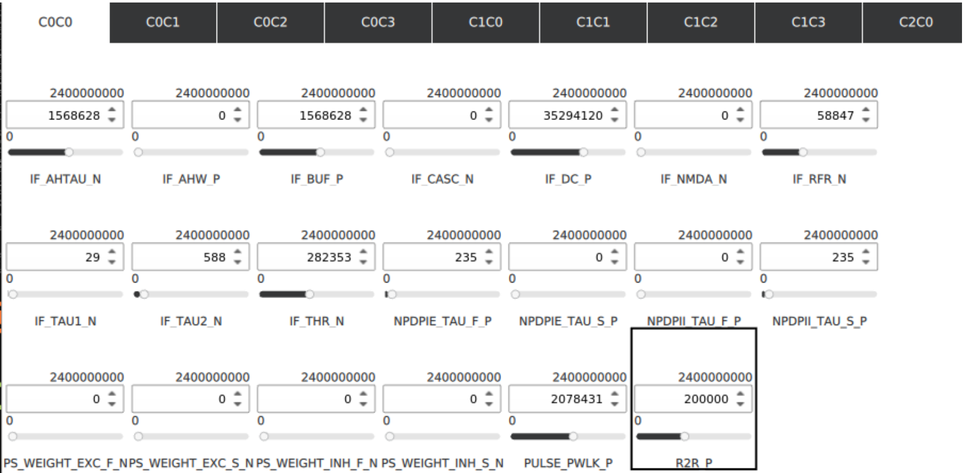




- **A**ddress-**E**vent-**R**epresentation
- Combination of hierarchical and 2D-mesh routing
- Why hierachical? Smaller address space when assuming locality → save memory
- Spike times encoded in Address Event times

Demo

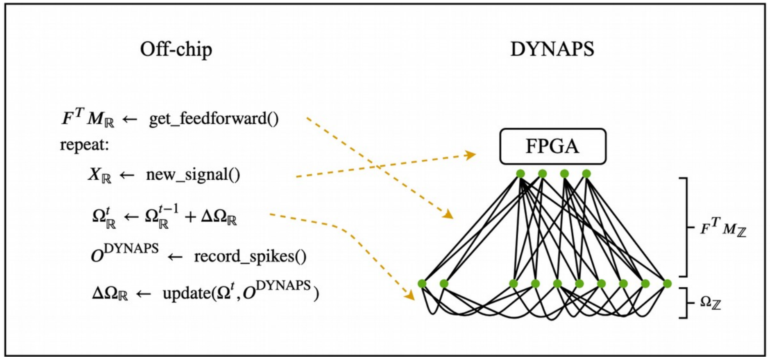




C0C0
C0C1
C0C2
C0C3
C1C0
C1C1
C1C2
C1C3
C2C0
2400000000
1568628
0
IF_AHTAU_N
2400000000
0
0
IF_AHW_P
2400000000
1568628
0
IF_BUF_P
2400000000
0
0
IF_CASC_N
2400000000
35294120
0
IF_DC_P
2400000000
0
0
IF_NMDA_N
2400000000
58847
0
IF_RFR_N
2400000000
29
0
IF_TAU1_N
2400000000
588
0
IF_TAU2_N
2400000000
282353
0
IF_THR_N
2400000000
235
0
NPDPIE_TAU_F_P
2400000000
0
0
NPDPIE_TAU_S_P
2400000000
0
0
NPDPII_TAU_F_P
2400000000
235
0
NPDPII_TAU_S_P
2400000000
0
0
PS_WEIGHT_EXC_F_N
2400000000
0
0
PS_WEIGHT_EXC_S_N
2400000000
0
0
PS_WEIGHT_INH_F_N
2400000000
0
0
PS_WEIGHT_INH_S_N
2400000000
2078431
0
PULSE_PWLK_P
2400000000
200000
0
R2R_P

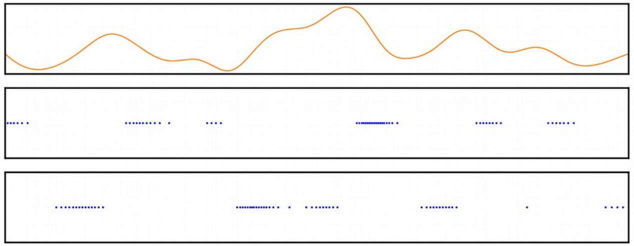
Learning in-the-loop

- Only train recurrent connections
- Mismatches between simulation and in-the-loop implementation:
  - Input must be in spike-form
  - Weight updates must be batched
  - Weight matrices must be discrete
  - Cannot measure voltage in software → Need to estimate from spikes and input
  - Update only neuron with highest voltage above threshold



Universität Zürich[UZH] | ETH zürich

Learning in-the-loop - Input



- Use delta modulation to transform continuous signal to spikes
- Could simply integrate signal using $\hat{x}_t = (1-\lambda)\hat{x}_{t-1} + \mu(o_t^{up} - o_t^{down})$
- General form for 2 inputs : $\hat{x}_t = (1-\lambda)\hat{x}_{t-1} + \mu M o_{t-1}$ with $M = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$
- $\rightarrow$ Feed-forward matrix becomes $F^T M$ and integration is performed by synapses

Learning in-the-loop - Updates

- In the simulation: Update rate and weights *only* for neuron highest above the threshold

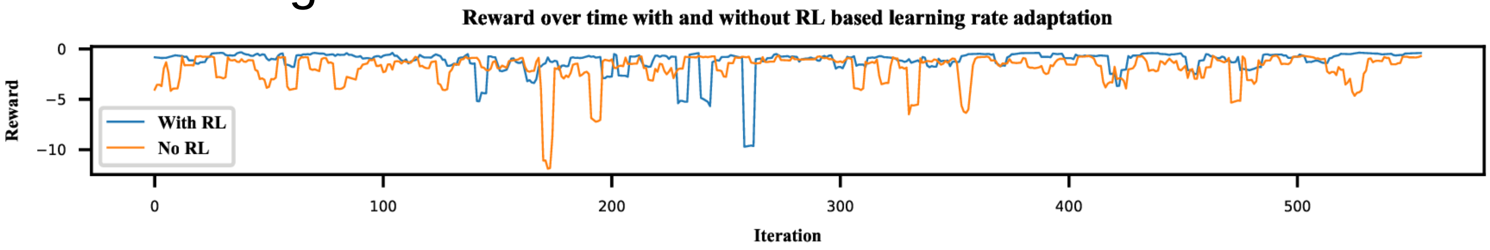
$$
\begin{aligned}
&(\max, k) \leftarrow \arg\max(V_t - \mathit{thresh} - 0.01 \cdot \mathcal{N}(0,1)) \\
&\textbf{if } \max \geq 0 \textbf{ then } \texttt{// Neuron k spiked} \\
&\quad \Omega_k \leftarrow \Omega_k - \epsilon_\Omega \cdot (\beta \cdot (V_t + \mu \cdot r_{t-1}) + \Omega_k + \mu \cdot I_k) \\
&\quad r_t \leftarrow r_{t-1} + I_k \\
&\textbf{else} \\
&\quad r_t \leftarrow r_{t-1} \\
&\textbf{end} \\
&r_t \leftarrow (1 - \lambda dt) \cdot r_t
\end{aligned}
$$

- To align with the DYNAP-SE, check if *assumption* of highest neuron above threshold matches DYNAP-SE recordings within some time-frame

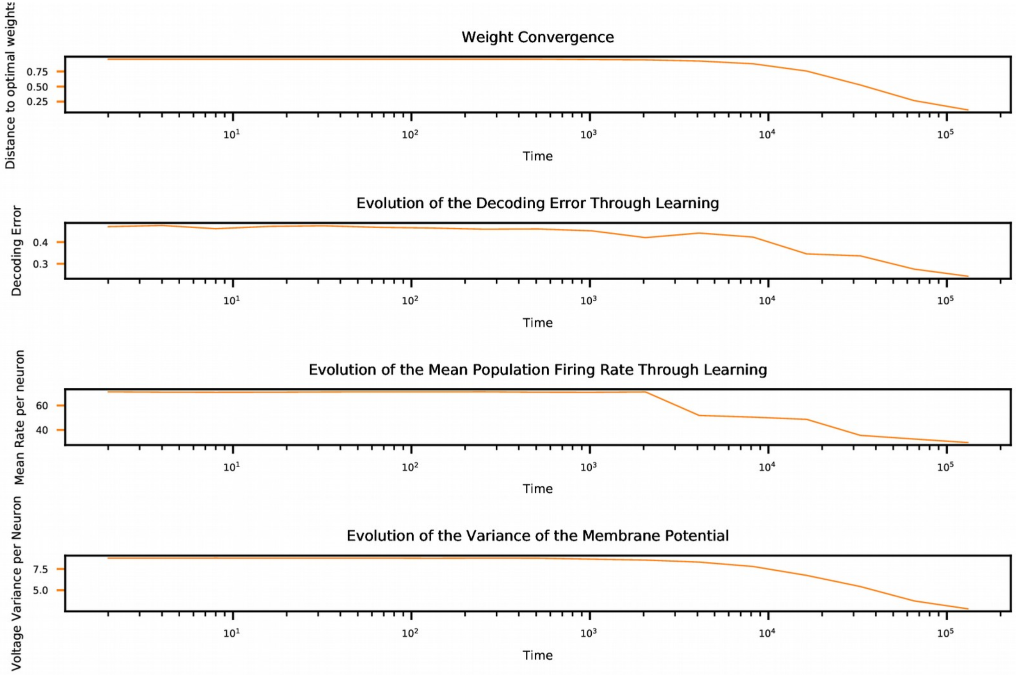
$$
\begin{aligned}
&(\max, k) \leftarrow \arg\max(V_t - \mathit{thresh} - 0.01 \cdot \mathcal{N}(0,1)) \\
&\textbf{if } \max \geq 0 \land \text{has\_spike}(\mathbf{O}^{\text{DYNAPS}}_{k,\pm\Delta T}) \textbf{ then} \\
&\quad \Delta\Omega_k \leftarrow \Delta\Omega_k + \epsilon_\Omega \cdot (\beta \cdot (V_t + \mu \cdot r_{t-1}) + \Omega_k + \mu \cdot \mathbf{I}) \\
&\quad o_t^{aligned} \leftarrow I_k \\
&\quad r_t \leftarrow r_{t-1} + I_k \\
&\textbf{else} \\
&\quad o_t^{aligned} \leftarrow \mathbf{0} \\
&\quad r_t \leftarrow r_{t-1} \\
&\textbf{end} \\
&r_t \leftarrow (1 - \lambda dt) \cdot r_t
\end{aligned}
$$

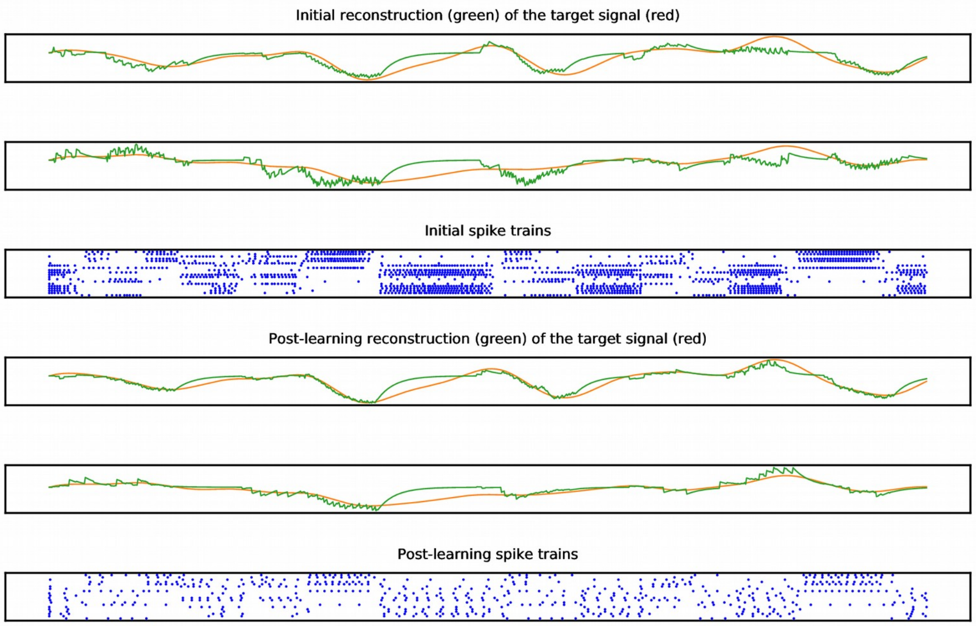
Experiments & Discussion

- Used 20 Neurons, trained for 140 iterations, 2 input signals
- Used Reinforcement Learning based approach to adapt learning rate
  Reward = Reconstruction error
  Action = Scale learning rate
- Intuition: Rewards for good signal representation should keep weights in current regime



Experiments & Discussion





Experiments & Discussion

- Very close to simulation → constant excitement would render condition always true and produce same results as simulation
- Update rule from the paper utilizes voltage *before* spike propagated, but can rewrite to use voltage after

$$
\begin{aligned}
&n \leftarrow \arg\max\left(\mathbf{V} - \mathbf{T}\right) - \xi_T(\tau)) \\
&\textbf{if } V_n > T_n \textbf{ then} \\
&\quad o_n(\tau) = 1 \\
&\quad \mathbf{F}_n(\tau) = \mathbf{F}_n(\tau - 1) + \epsilon_F(\alpha\mathbf{x}(\tau - 1) - \mathbf{F}_n(\tau - 1)) \\
&\quad \mathbf{\Omega}_n(\tau) = \mathbf{\Omega}_n(\tau - 1) - \epsilon_\Omega(\beta(\boxed{\mathbf{V}(\tau - 1)} + \mu\mathbf{r}(\tau - 1)) + \mathbf{\Omega}_n(\tau - 1))
\end{aligned}
$$

- Last problem: Learning fails when discretizing weights
  Derive for discrete weights?
- Goal: Resolve issues and implement in circuit on next chip-iteration