
Event-based update of synapses in voltage-based learning rules

Jonas Stapmanns^{1,2,*}, Jan Hahne³, Moritz Helias^{1,2}, Matthias Bolten³,
Markus Diesmann^{1,4,5} and David Dahmen¹

¹ *Institute of Neuroscience and Medicine (INM-6) and Institute for Advanced Simulation (IAS-6) and JARA Institute Brain Structure Function Relationship (INM-10), Jülich Research Centre, Jülich, Germany*

² *Institute for Theoretical Solid State Physics, RWTH Aachen University, Aachen, Germany*

³ *School of Mathematics and Natural Sciences, Bergische Universität Wuppertal, Wuppertal, Germany*

⁴ *Department of Physics, Faculty 1, RWTH Aachen University, Aachen, Germany*

⁵ *Department of Psychiatry, Psychotherapy and Psychosomatics, Medical Faculty, RWTH Aachen University, Aachen, Germany*

Correspondence*:

Jonas Stapmanns

jonas.stapmanns@rwth-aachen.de

ABSTRACT

Due to the point-like nature of neuronal spiking, efficient neural network simulators often employ event-based simulation schemes for synapses. Yet many types of synaptic plasticity rely on the membrane potential of the postsynaptic cell as a third factor in addition to pre- and postsynaptic spike times. Synapses therefore require continuous information to update their strength which a priori necessitates a continuous update in a time-driven manner. The latter hinders scaling of simulations to realistic cortical network sizes and relevant time scales for learning.

Here, we derive two efficient algorithms for archiving postsynaptic membrane potentials, both compatible with modern simulation engines based on event-based synapse updates. We theoretically contrast the two algorithms with a time-driven synapse update scheme to analyze advantages in terms of memory and computations. We further present a reference implementation in the spiking neural network simulator NEST for two prototypical voltage-based plasticity rules: the Clopath rule and the Urbanczik-Senn rule. For both rules, the two event-based algorithms significantly outperform the time-driven scheme. Depending on the amount of data to be stored for plasticity, which heavily differs between the rules, a strong performance increase can be achieved by compressing or sampling of information on membrane potentials. Our results on computational efficiency related to archiving of information provide guidelines for the design of learning rules in order to make them practically usable in large-scale networks.

Keywords: voltage-based plasticity rules, event-based simulation, spiking neural network simulator, NEST, Clopath rule, Urbanczik-Senn rule

1 INTRODUCTION

One mechanism for learning in the brain is implemented by changing the strengths of connections between neurons, known as synaptic plasticity. Already early on, such plasticity was found to depend on the activity of the connected neurons. Donald Hebb postulated the principle 'Cells that fire together, wire together' (Hebb, 1949). Later on, it was shown that plasticity is shaped by temporal coordination of activities even down to the level of individual spikes (Markram et al., 1997; Bi and Poo, 1998). Synaptic plasticity rules for spiking neural networks, such as spike timing-dependent plasticity (STDP, Gerstner et al. (1996)), consequently employ spike times of pre- and postsynaptic cells to predict the change in connections.

In recent years, a new class of biologically inspired plasticity rules has been developed that take into account the membrane potential of the postsynaptic neuron as an additional factor (for a review, see Mayr and Partzsch, 2010; Gerstner et al., 2014). The rule by Clopath et al. (2010) can be seen as a prototypical example for a voltage-based plasticity rule since long-term potentiation of synapses depends on the presynaptic spike arrival and a filtered version of the postsynaptic membrane potential. This additional voltage dependence enables the Clopath rule to describe phenomena that are not covered by ordinary STDP but can be observed in experimental data, such as the complex frequency dependence of the synaptic weight changes in spike pairing experiments (Sjöström et al., 2001). Furthermore it provides a mechanism for the creation of strong bidirectional connections in networks, which have been found to be overrepresented in some cortical areas (Song et al., 2005).

Further inspiration for recently proposed plasticity rules originates from the field of artificial neural networks. These networks showed great success in the past decade, for example in image or speech recognition tasks (Hinton et al., 2006; Krizhevsky et al., 2012; Hannun et al., 2014; LeCun et al., 2015). The involved learning paradigms, for example the backpropagation algorithm (Werbos, 1974; Parker, 1985; Lecun, 1985; Rumelhart et al., 1986), are, however, often not compatible with biological constraints such as locality of information for weight updates. To bridge the gap to biology, different biologically inspired approximations and alternatives to the backpropagation algorithm have been proposed (Neftci et al., 2017; Sacramento et al., 2018; Bellec et al., 2019; Cartiglia et al., 2020). A common feature of many of these rules is that weight updates not only depend on the output activity of pre- and postsynaptic cells, but also on a third factor, which is a time-continuous signal. A prominent example of such biologically and functionally inspired rules is the voltage-based plasticity rule proposed by Urbanczik and Senn (2014), where the difference between somatic and dendritic membrane potential serves as an error signal that drives learning. This rule, incorporated in complex microcircuits of multi-compartment neurons, implements local error-backpropagation (Sacramento et al., 2018).

Research on functionally inspired learning rules in biological neural networks is often led by the requirement to implement a particular function rather than efficiency. Present studies are therefore primarily designed to prove that networks with a proposed learning rule minimize a given objective function. Indeed many learning rules are rather simple to implement and to test in ad-hoc implementations where at any point the algorithm has access to all state variables. While the latter implementations are sufficient for a proof of principle, they are hard to reuse, reproduce and generalize. In particular, simulations are restricted to small network sizes, as the simulation code cannot be straight-forwardly distributed across compute nodes and thus parallelized. This also limits the simulation speed which is, in particular, problematic given that successful learning requires simulating networks for long biological times.

In parallel to the above efforts are long-term developments of simulation software for biological neural networks (for a review, see Brette et al., 2007). Such open-source software, combined with interfaces and

simulator-independent languages ([Davison et al., 2008](#); [Djurfeldt et al., 2010, 2014](#)), supports maintainability and reproducibility, as well as community driven development. The design of such simulators is primarily led by implementation efficiency. Code is optimized for neuron and synapse dynamics, with the aim to upscale simulations to biologically realistic network sizes. A modular structure of the code facilitates re-use and extensions in functionality. Therefore, one aim of the community should be the transfer of ad-hoc proof-of-principle implementations to these well-tested platforms. Given the differences in design principles behind the exploratory development of specific models and general-purpose simulation technology, this transfer is not trivial. In the current study, we show how to make voltage-based learning rules compatible with spiking neural network simulators that employ an event-driven update scheme for synapses.

Modern network simulators use individual objects for different neurons and synapses. One common strategy of parallelization is to distribute these objects across many compute processes ([Jordan et al., 2018](#); [Lytton et al., 2016](#)). Communication between neurons then implies exchange of information between compute processes. Neurons in the brain primarily communicate in an event-based fashion via spikes. The duration of these spike events is on the order milliseconds, which together with typical rates during physiological brain states of a few spikes per second yields a coupling that is sparse in time ([Figure 1A](#)). Spiking simulators emulate this communication by idealizing spikes as instantaneous events. Thus, in the absence of direct electrical coupling via gap junctions ([Kumar and Gilula, 1996](#); [Hahne et al., 2015](#); [Jordan et al., 2020a](#)), there is no neuronal interaction in between two spike events such that the dynamics of neuronal and synaptic state variables can be evolved independently in time. This led to the development of event-based simulation schemes, where synapses are only updated in their state at the times of incoming spikes ([Watts, 1994](#); [Morrison et al., 2005](#)). This significantly reduces the amount of function calls to synapse code and optimizes computational performance in network simulations. Modern spiking network simulators such as Aurnyn ([Zenke and Gerstner, 2014](#)), Brian ([Stimberg et al., 2014](#)), Neuron ([Carnevale and Hines, 2006](#)), NEST ([Gewaltig and Diesmann, 2007](#)) and Nevesim ([Pecevski et al., 2014](#)) therefore employ an event-based update scheme for synapses. Even though spike events at single synapses are rare, each single neuron typically receives a large amount of spikes in rapid succession due to its large number of incoming connections (in-degree). This suggests a time-driven update of neurons ([Figure 1B](#)). The resulting hybrid simulation scheme for neurons and synapses ([Morrison et al., 2005](#); [D’Haene et al., 2014](#); [Krishnan et al., 2017](#)) is nowadays commonly used across many spiking network simulators (for a review, see [Brette et al., 2007](#)).

An event-based scheme for synapses is perfectly suitable for classical STDP rules, which only rely on a comparison between the timings of spike events. Optimizations of simulations including STDP have been extensively discussed theoretically ([Rudolph and Destexhe, 2006](#); [Ros et al., 2006](#); [Morrison et al., 2007a](#)) and routinely used in the most common spiking network simulators ([Carnevale and Hines, 2006](#); [Gewaltig and Diesmann, 2007](#); [Goodman and Brette, 2013](#); [Pecevski et al., 2014](#); [Zenke and Gerstner, 2014](#)) as well as in neuromorphic hardware ([Serrano-Gotarredona et al., 2013](#); [Pfeil et al., 2013](#); [Nefci et al., 2014](#); [Galluppi et al., 2015](#); [Friedmann et al., 2016](#); [Thakur et al., 2018](#)). Some STDP variants also include the membrane potential of postsynaptic cells at the time points of the spike events as a gating variable ([Brader et al., 2007](#); [Diederich et al., 2018](#)). At the update, these rules only require the synapse to know the current value of the postsynaptic membrane potential in addition to the pre- and postsynaptic spike time. Obtaining this value from the neuron objects is efficient to implement and already employed in a range of neuromorphic systems ([Serrano-Gotarredona et al., 2013](#); [Galluppi et al., 2015](#); [Qiao et al., 2015](#); [Moradi et al., 2018](#); [Cartiglia et al., 2020](#)).

We here focus on more complex voltage-based learning rules where synapses continuously require information from the postsynaptic neurons in order to update their weights ([Clopath et al., 2010](#); [Mayr and Partzsch, 2010](#); [Brea et al., 2013](#); [Yger and Harris, 2013](#); [Urbanczik and Senn, 2014](#); [Albers et al., 2016](#)). This a priori breaks the idea behind an event-based update scheme. Therefore, previous attempts to incorporate such voltage-based plasticity in spiking network simulators resorted to time-driven synapse updates for NEST ([Jordan et al., 2020b](#)) and NEURON (see implementation of Clopath plasticity on ModelDB, [Hines et al. 2004](#)). These implementations therefore only profit from the simulation environment on the level of the implementation language, but have not been able to exploit the algorithmic optimizations and speedup of event-based synapse updates.

In this study we present an efficient archiving method for the history of postsynaptic state variables that allows for an event-based update of synapses and thus makes complex voltage-based plasticity rules compatible with state-of-the-art simulation technology for spiking neural networks. In particular, we derive two event-based algorithms that store a time-continuous or discontinuous history, respectively. These algorithms apply to plasticity rules with any dependence on post-synaptic state variables and therefore cover a large range of existing models ([Brader et al., 2007](#); [Mayr and Partzsch, 2010](#); [Legenstein and Maass, 2011](#); [Yger and Harris, 2013](#); [Brea et al., 2013](#); [Qiao et al., 2015](#); [Albers et al., 2016](#); [Sheik et al., 2016](#); [Brea et al., 2016](#); [Diederich et al., 2018](#); [Sacramento et al., 2018](#); [Cartiglia et al., 2020](#)). We theoretically analyze advantages of the two event-driven algorithms with respect to each other and compare to a straight-forward time-driven algorithm.

The presented simulation concepts are exemplified and evaluated in a reference implementation in the open source simulation code NEST ([Gewaltig and Diesmann, 2007](#); [Jordan et al., 2019](#)). The reference implementation thereby exploits existing functionality of a scalable software platform which can be used on laptops as well as supercomputers. NEST is employed by a considerable user community and equipped with an interface to the programming language Python ([Eppler et al., 2009](#)) that is currently widely used in the field of computational neuroscience ([Muller et al., 2015](#)). It supports relevant neuron models and connection routines for the construction of complex networks. Despite this flexibility the simulation engine shields the researcher from the difficulties of handling a model description in a distributed setting ([Morrison et al., 2005](#); [Plesser et al., 2015](#)).

To exemplify the general simulation algorithms, we here focus on the voltage-based plasticity rules by [Clopath et al. \(2010\)](#) and [Urbanczik and Senn \(2014\)](#). The two rules represent opposing ends of a family of learning rules in the amount of data required to compute weight updates. The Clopath rule by design only triggers plasticity in the vicinity of postsynaptic spike events; storing a history, which is non-continuous in time, thus becomes beneficial. In contrast, the Urbanczik-Senn rule considers noisy prediction errors based on postsynaptic membrane voltages and spikes. Such prediction errors never vanish and therefore always need to be stored to update the weights, leading to time-continuous histories. For a given span of biological time, simulations of the Urbanczik-Senn rule are therefore by design less efficient than those of the Clopath rule. However, we show that a compression of membrane potential information reduces this performance gap. Changing the learning rule to include a sparse sampling of the membrane voltage further increases efficiency and makes performance comparable to simulations with ordinary STDP.

Our study begins with a specification of the mathematical form of the learning rules that we consider ([Section 2.1](#)). We distinguish between classical STDP ([Section 2.2](#)) and voltage-based rules ([Section 2.3](#)) and present a special case where voltage-based rules can be efficiently implemented by compressing information on the postsynaptic membrane potential. We then introduce the Clopath and the Urbanczik-Senn rule as two examples of voltage-based plasticity ([Section 2.4](#) and [Section 2.5](#)). In [Section 3](#) we first

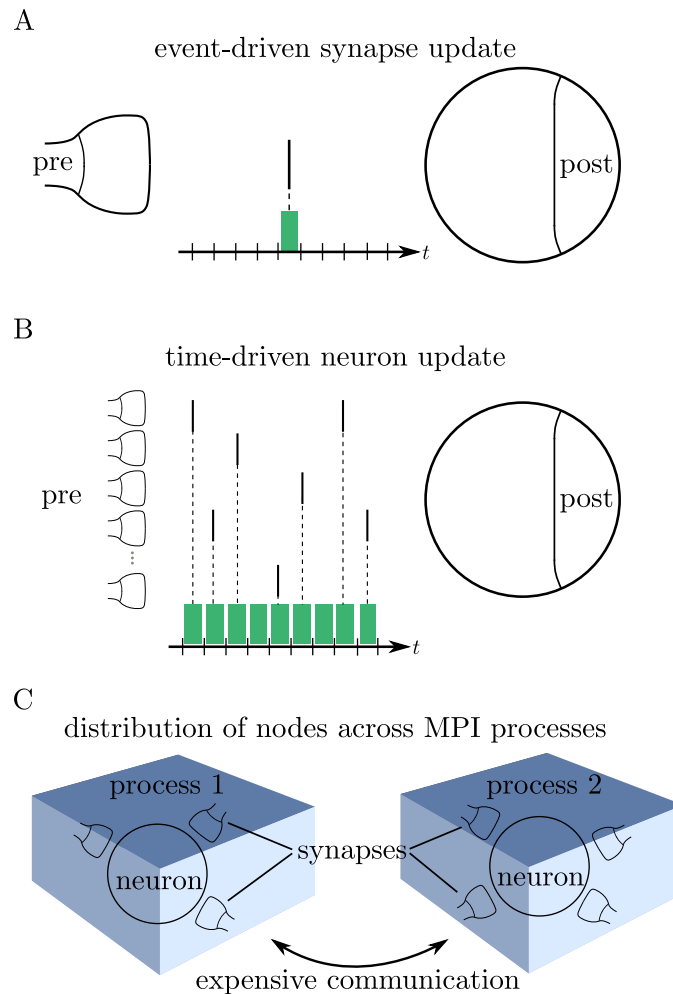


Figure 1. Update schemes for neurons and synapses. **A** A spike crosses a synapse from the presynaptic (pre) to the postsynaptic (post) neuron. Since this is a rare event, the synaptic weight is computed only when a spike is delivered, indicated by the green bar (event-driven update). **B** Neurons with a large in-degree receive spikes in rapid succession which suggests a time-driven update of the neuron’s state in each time step (green bars). **C** Since the computation of the synaptic weights requires information from the postsynaptic neuron, storing the synapses on the same compute node reduces the amount of expensive communication between compute processes.

contrast time- and event-driven schemes for updating synapses with voltage-based plasticity (Section 3.1). Subsequently, we detail a reference implementation of the algorithms in NEST (Section 3.2) and use this to reproduce results from the literature (Section 3.3). After that, we examine the performance of the reference implementation for the Clopath and the Urbanczik-Senn rule (Section 3.4). Conclusions from the implementation of the two rules are drawn in Section 3.5, followed by a general Discussion in Section 4. The technology described in the present article is available in the 2.20.1 release of the simulation software NEST as open source. The conceptual and algorithmic work is a module in our long-term collaborative project to provide the technology for neural systems simulations (Gewaltig and Diesmann, 2007).

2 MATERIALS AND METHODS

2.1 General structure of learning rules

The focus of this study are plasticity models of the general form

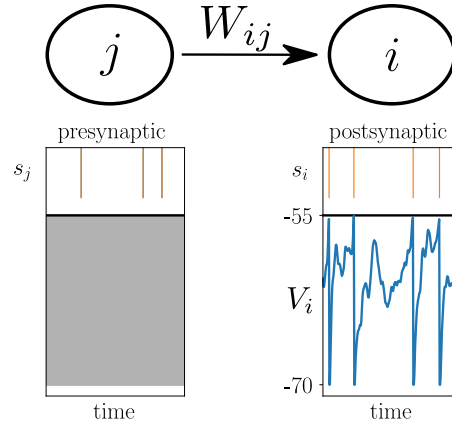


Figure 2. Voltage-based plasticity rules. The change ΔW_{ij} in synaptic strength between presynaptic neuron j and postsynaptic neuron i depends on the presynaptic spike train s_j , the postsynaptic spike train s_i and the postsynaptic membrane potential V_i .

$$\frac{dW_{ij}(t)}{dt} = F(W_{ij}(t), s_i^*(t), s_j^*(t), V_i^*(t)) \quad (1)$$

where the change $\frac{dW_{ij}(t)}{dt}$ of the synaptic weight W_{ij} between the presynaptic neuron j and postsynaptic neuron i is given by a function F that potentially depends on the current synaptic weight $W_{ij}(t)$, as well as on $s_i^*(t)$, $s_j^*(t)$, $V_i^*(t)$ which are causal functionals of the postsynaptic spike train s_i , the presynaptic spike train s_j , and the postsynaptic membrane potential V_i , respectively (Figure 2). Causal functional here refers to $s_i^*(t)$ potentially depending on all past values $s_i(t' \leq t)$; likewise $V_i^*(t)$ depends on $V_i(t' \leq t)$. Note that for simplicity of the notation, we only show one function F on the right hand side of (1), while generally there could be a sum of multiple functions or functionals F_α , where each one depends on spike trains and membrane potentials in a different manner. Note also that F mixes information of pre- and postsynaptic neurons, while the functionals denoted by $*$ only need to take into account information of either the pre- or postsynaptic neuron. In cases where F is a functional, i.e. where F depends on the whole time course of its arguments, it can take into account an additional joint history dependence on s_i^* , s_j^* and V_i^* . A special case, the Urbanczik-Senn learning rule, is discussed further below.

One can formally integrate (1) to obtain the weight change between two arbitrary time points t and T

$$\Delta W_{ij}(t, T) = \int_t^T dt' F(W_{ij}(t'), s_i^*(t'), s_j^*(t'), V_i^*(t')). \quad (2)$$

2.2 Spike-timing dependent plasticity

In general, the integral on the right hand side of the equation cannot be calculated analytically. There is, however, a notable exception, which is the model of spike-timing dependent plasticity (STDP). This model is a form of Hebbian plasticity that relies on the exact spike times of pre- and postsynaptic neurons and ignores any effect of the postsynaptic membrane potential. The dependence on the exact spike times becomes apparent by the fact that either the pre- or postsynaptic spike functional is the spike train itself,

for example

$$s_i^*(t) = s_i(t) = \sum_k \delta(t - t_i^k), \quad (3)$$

where t_i^k is the k -th spike of the i -th neuron. This yields a plasticity rule that reads (Morrison et al., 2008)

$$\frac{dW_{ij}(t)}{dt} = -f_-(W_{ij}(t))s_{-,i}^*(t)s_j(t) + f_+(W_{ij}(t))s_{+,j}^*(t)s_i(t) \quad (4)$$

with functions f_{\pm} that model the weight dependence, and functionals $s_{\pm}^*(t) = (\kappa_{\pm} * s)(t)$ given as convolutions of spike trains with kernels κ_{\pm} , which in the classical STDP rule correspond to one-sided exponential decays. The appearance of the raw spike trains (delta distributions) in the differential equation of the STDP model renders the integration of the ODE trivial

$$\Delta W_{ij}(t, T) = - \sum_{\text{spikes } k} f_-(W_{ij}(t_j^k))\kappa_{-,i}(t_j^k) + \sum_{\text{spikes } l} f_+(W_{ij}(t_i^l))\kappa_{+,j}(t_i^l), \quad (5)$$

where $t_j^k, t_i^l \in [t, T]$. An update of the synaptic weight between any two time points only requires knowledge of the weight and spike functionals at the timing of the pre- and postsynaptic spikes.

For models that do not solely rely on exact spike times, but for example on filtered versions of the spike trains, much more information is needed in order to calculate a weight update $\Delta W_{ij}(t, T)$ between any two time points. This makes the computation more involved: the synapse needs all values of $W_{ij}(t')$, $s_i^*(t')$, $s_j^*(t')$, $V_i^*(t')$ for $t' \in [t, T]$ to update its weight. The remainder of this study describes different approaches to this problem and their advantages and disadvantages.

2.3 Voltage-based plasticity

In a time-driven neuron update, the membrane potential in many simulators is computed at each simulation step $t^\alpha = \alpha \cdot h$, where h is the simulation step size and $\alpha \in \mathbb{N}$. For plasticity models that rely on the membrane potential, the time discretization of (2) therefore yields

$$\Delta W_{ij}(t, T) = \sum_{\text{steps } \alpha} \Delta W_{ij}(t^\alpha, t^{\alpha+1}), \quad (6)$$

$$\Delta W_{ij}(t^\alpha, t^{\alpha+1}) = \int_{t^\alpha}^{t^{\alpha+1}} dt' F(W_{ij}(t'), s_i^*(t'), s_j^*(t'), V_i^*(t^\alpha)). \quad (7)$$

which, in comparison to the small sum over spikes in the STDP rule (5), contains a large sum over all time steps t^α in between time points t and T . As the membrane potential is only known at time points $t' = t^\alpha$, it enters (7) in a piecewise constant manner – hence the argument $V(t^\alpha)$. The synapse therefore predominantly needs information of the postsynaptic neuron in order to update its weight. Thus, in a distributed simulation framework, where neurons are split across multiple compute processes, it is beneficial to store the synapses at the site of the postsynaptic neurons in order to reduce communication (Figure 1C). This confirms the earlier design decision of Morrison et al. (2005) who place synapses at the site of the postsynaptic neuron to reduce the amount of data communicated by the presynaptic site.

If weight changes ΔW_{ij} depend on the synaptic weight themselves, then (7) cannot be used in practice as intermediate weights $W_{ij}(t')$ for $t^\alpha < t' < t^{\alpha+1}$ are not known. In this scenario, weight changes have

to be calculated on the simulation grid with $W_{ij}(t') \rightarrow W_{ij}(t^\alpha)$ in case of a forward Euler scheme, or $W_{ij}(t') \rightarrow W_{ij}(t^{\alpha+1})$ in case of a backward Euler scheme. In the following we, for simplicity, stick to the forward Euler setting and arrive at the core computation for voltage-based plasticity rules

$$\Delta W_{ij}(t^\alpha, t^{\alpha+1}) = \int_{t^\alpha}^{t^{\alpha+1}} dt' F(W_{ij}(t^\alpha), s_i^*(t'), s_j^*(t'), V_i^*(t^\alpha)). \quad (8)$$

Given that s_i and s_j are spike trains, the functionals s_i^* and s_j^* are obtained trivially from the kernels of their corresponding Volterra expansions. If F in addition does not depend on s_i^* and s_j^* in a too complicated manner, which is usually the case (see examples below), the integral in (8) can be calculated analytically.

2.3.1 Compression of postsynaptic information

The major operation of the plasticity scheme in terms of frequency and complexity is the computation of infinitesimal weight changes $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$. Since the presynaptic spike train s_j^* enters F in (8), the same postsynaptic information on s_i^* and V_i^* is used many times for very similar computations: the membrane potential trace of each neuron is effectively integrated many times. Is there a way to employ the result of the computation $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$ for neuron j for the computations $\Delta W_{ik}(t^\alpha, t^{\alpha+1})$ for other neurons $k \neq j$? In a simple setting, where F factorizes into $F(W_{ij}(t), s_i^*(t), s_j^*(t), V_i^*(t)) = s_j^*(t) G(s_i^*(t), V_i^*(t))$ with $s_j^*(t) = (\kappa * s_j)(t)$ and

$$\kappa(t) = H(t) \frac{1}{\tau} e^{-\frac{t}{\tau}}, \quad (9)$$

defined via the Heaviside step function $H(x)$, we can make use of the property

$$s_j^*(t) = (s_j^*(t_{LS}) + \tau^{-1}) e^{-(t-t_{LS})/\tau}, \quad (10)$$

where $t > t_{LS}$ and t_{LS} denotes the last spike time of the presynaptic neuron. In this case the weight update in between two spike events factorizes

$$\Delta W_{ij}(t_{LS}, t_S) = \underbrace{(s_j^*(t_{LS}) + \tau^{-1})}_{=: \bar{x}_j(t_{LS})} \underbrace{\int_{t_{LS}}^{t_S} dt' e^{-(t'-t_{LS})/\tau} G(s_i^*(t'), V_i^*(t'))}_{=: \Delta W_i(t_{LS}, t_S)} \quad (11)$$

where the latter integral $\Delta W_i(t_{LS}, t_S)$ is independent of the presynaptic spike train s_j^* . Moreover, ΔW_i depends on t_{LS} only via an exponential prefactor. Thus, an integral $\Delta W_i(t_1, t_2)$ over an arbitrary time interval $t_{LS} \leq t_1 < t_2 \leq t_S$ which is completely independent of any presynaptic information, can be used as a part of the whole integral $\Delta W_i(t_{LS}, t_S)$ since it can be decomposed as

$$\Delta W_i(t_{LS}, t_S) = \Delta W_i(t_{LS}, t_1) + e^{-\frac{t_1-t_{LS}}{\tau}} \Delta W_i(t_1, t_2) + e^{-\frac{t_2-t_{LS}}{\tau}} \Delta W_i(t_2, t_S).$$

Therefore, whenever an integral of the postsynaptic quantities s_i^* and V_i^* is computed, it can be used to advance the weight update of all incoming connections and the integration only needs to be performed once. To account for the generally different last spike times t_{LS} of the incoming connections, the postsynaptic neuron stores the different $\Delta W_i(t_{LS}, t)$ in a so-called *compressed history*. At the time of an incoming spike event, $\Delta W_i(t_{LS}, t_S)$ can be read out by the synapse for the correct t_{LS} of that synapse and be combined with the stored presynaptic spike trace s_j^* .

2.4 Example 1: Clopath plasticity

The Clopath rule ([Clopath et al., 2010](#)) was designed as a voltage-based STDP rule that accounts for nonlinear effects of spike frequency on weight changes which had been previously observed in experiments ([Sjöström et al., 2001](#)). It does so by using the evolution of the postsynaptic membrane voltage around postsynaptic spike events instead of the postsynaptic spikes themselves. This requires a neuron model that takes into account features of membrane potential excursions near spike events, such as modified adaptive exponential integrate-and-fire (aeif) model neurons that are used in the original publication ([Clopath et al. \(2010\)](#), see [Section 5.2](#)) or Hodgkin-Huxley (hh) neurons that are used in a NEURON reference implementation on ModelDB ([Hines et al., 2004](#)).

The plasticity rule is of the general form (1) with a sum of two different functions F_α on the right hand side. It treats long-term depression (LTD) and potentiation (LTP) of the synaptic weight in the two terms F_{LTD} and F_{LTP} , with

$$F_{\text{LTD}}(s_j(t), V_{i,\text{LTD}}^*(t)) = -A_{\text{LTD}} s_j(t) V_{i,\text{LTD}}^*(t) \quad (12)$$

$$\text{with } V_{i,\text{LTD}}^* = (\bar{u}_- - \theta_-)_+,$$

$$\bar{u}_-(t) = (\kappa_- * V_i)(t - d_s)$$

and

$$F_{\text{LTP}}(s_j^*(t), V_{i,\text{LTP}}^*(t)) = A_{\text{LTP}} s_j^*(t) V_{i,\text{LTP}}^*(t) \quad (13)$$

$$\text{with } s_j^* = \kappa_s * s_j,$$

$$V_{i,\text{LTP}}^* = (\bar{u}_+ - \theta_+)_+ (V_i - \theta_+)_+,$$

$$\bar{u}_+(t) = (\kappa_+ * V_i)(t - d_s).$$

Here $(x - x_0)_+ = H(x - x_0) (x - x_0)$ is the threshold-linear function and $H(x)$ is the Heaviside step function. A_{LTD} and A_{LTP} are prefactors controlling the relative strength of the two contributions. κ_\pm are exponential kernels of the form (9), which are applied to the postsynaptic membrane potential, and κ_s is an exponential kernel applied to the presynaptic spike train. The time-independent parameters θ_\pm serve as thresholds below which the (low-pass filtered) membrane potential does not cause any weight change ([Figure 3](#)). Note that A_{LTP} can also depend on the membrane potential. This case is described in [Appendix Section 5.5](#).

In a reference implementation of the Clopath rule by C. Clopath and B. Torben-Nielsen available on ModelDB ([Hines et al., 2004](#)), there is a subtle detail not explicitly addressed in the original journal article. In their implementation the authors introduce an additional delay d_s between the convolved version of the membrane potentials \bar{u}_\pm and the bare one (cf. parameter d_s in (12) and (13)). The convolved potentials are shifted backwards in time by the duration of a spike d_s (see [Tables S1 and S3](#) in supplement). As a result, the detailed shape of the excursion of the membrane potential during a spike of the postsynaptic neuron does not affect the LTP directly but only indirectly via the low-pass filtered version \bar{u}_+ , see green background in [Figure 3B](#). Incorporating this time shift in \bar{u}_\pm is essential to reproduce the results from [Clopath et al. \(2010\)](#) on spike-pairing experiments.

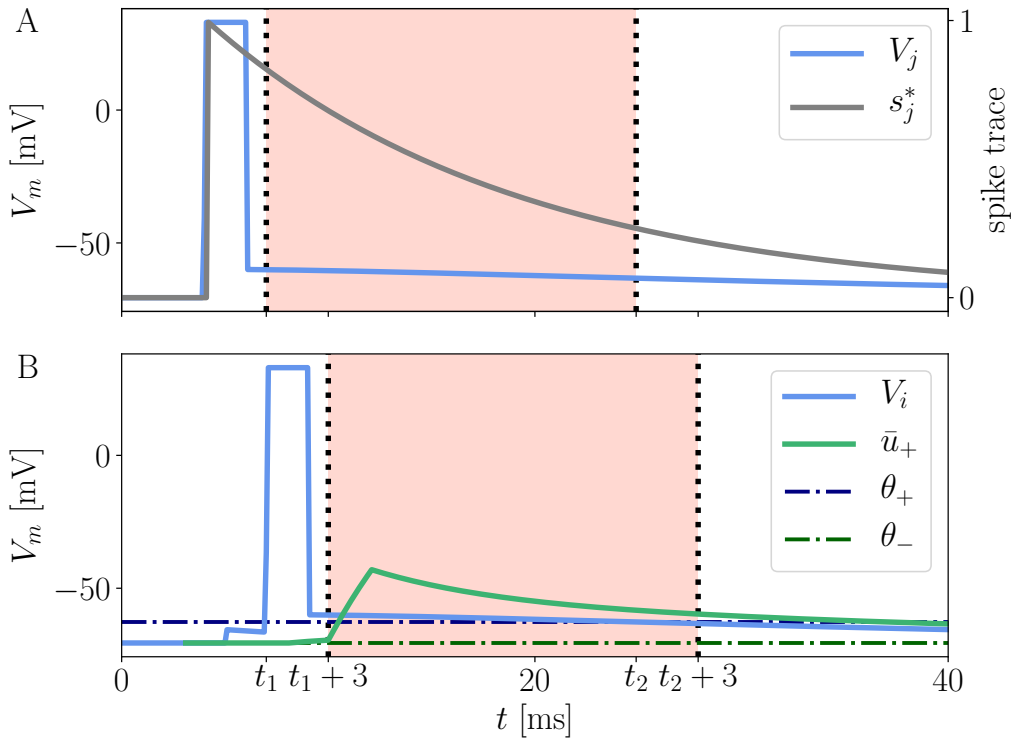


Figure 3. Illustration of LTP contribution to the Clopath rule. A presynaptic neuron (panel A) and a postsynaptic neuron (panel B) emit a spike at $t_{sp,pre} = 4$ ms and $t_{sp,post} = 6$ ms, respectively. The presynaptic spike elicits a trace s_j^* (gray) at the synapse. The excursion of the postsynaptic membrane potential V_i (panel B, blue) elevates the low-pass filtered potential \bar{u}_+ (green) so that both V_i and \bar{u}_+ exceed the respective thresholds θ_+ (dash-dotted, dark blue) and θ_- (dash-dotted, dark green), cf. (13), between t_1 and t_2 . Only within this period, shifted by $d_s = 3$ ms, which is for times $t_1 + 3$ ms $< t < t_2 + 3$ ms (panel B, red background), see Section 3.2.3 for details, the LTP of the synaptic weight is non-vanishing because of the threshold-linear functions in 13. The shift by $d_s = 3$ ms does not apply to the spike trace (panel A, red background). The rectangular shape of the spikes is achieved by a clamping of the membrane potential to $V_{clamp} = 33$ mV for a period of $t_{clamp} = 2$ ms.

The depression term F_{LTD} depends on the unfiltered spike train s_j . It can thus be treated analogous to ordinary STDP rules (cf. (4)ff). In particular, $V_{i,LTD}^*$ only needs to be available for time points of presynaptic spikes (potentially taking into account additional delays of the connection). The potentiation term F_{LTP} , however, depends on the filtered spike train s_j^* ; $V_{i,LTP}^*$ consequently needs to be known also for times in between spike events.

2.5 Example 2: Urbanczik-Senn plasticity

The Urbanczik-Senn rule (Urbanczik and Senn, 2014) applies to synapses that connect to dendrites of multicompartment model neurons. The main idea of this learning rule is to adjust the weights of dendritic synapses such that the dendrite can predict the firing rate of the soma. The dendrite expects the firing rate to be high when the dendrite's membrane potential is elevated due to many incoming spikes at the dendrite, and to be low if there are only a few incoming spikes. Thus, for this prediction to be true, synapses that transmit a spike towards the dendrite while the firing rate of the soma is low are depressed and those that provide input while the soma's firing rate is high are facilitated. Learning can be triggered by applying

a teacher signal to the neuron via somatic synapses such that the actual somatic firing deviates from the dendritic prediction.

The plasticity rule is again of the general form (1), with a functional F on the right hand side that reads

$$F[s_j^*, V_i^*] = \eta \kappa * (V_i^* s_j^*) \quad (14)$$

$$\text{with } V_i^* = (s_i - \phi(V_i)) h(V_i), \quad (15)$$

$$s_j^* = \kappa_s * s_j.$$

with exponential filter kernels κ and κ_s and nonlinearities ϕ and h . Note that F depends on the postsynaptic spike train s_i via V_i^* . The latter can be interpreted as a prediction error, which never vanishes as spikes s_i (point process) are compared against a rate prediction $\phi(V_i)$ (continuous signal).

In order to solve (1), we need to integrate over $F[s_j^*, V_i^*]$ (cf. (2)). Writing down the convolution with κ explicitly, we obtain

$$\begin{aligned} \Delta W_{ij}(t, T) &= \int_t^T dt' F[s_j^*, V_i^*](t') \\ &= \int_t^T dt' \eta \int_0^{t'} dt'' \kappa(t' - t'') V_i^*(t'') s_j^*(t''). \end{aligned} \quad (16)$$

A straight forward implementation of this expression is inefficient in terms of memory usage and computations because of the two nested integrals. However, since the kernels κ and κ_s are exponentials, one can perform one of the integrations analytically (see appendix Section 5.1 for a derivation) to rewrite the weight change as

$$\begin{aligned} \Delta W_{ij}(t, T) &= \eta \left[I_1(t, T) - I_2(t, T) + I_2(0, t) \left(1 - e^{-\frac{T-t}{\tau\kappa}} \right) \right], \quad (17) \\ \text{with } I_1(a, b) &= \int_a^b dt V_i^*(t) s_j^*(t), \\ I_2(a, b) &= \int_a^b dt e^{-\frac{b-t}{\tau\kappa}} V_i^*(t) s_j^*(t), \end{aligned}$$

which is in line with the general formulation discussed in Section 2.3.

3 RESULTS

In the following, we first discuss time- and event-driven update schemes for synapses with voltage-based plasticity. Then we present a reference implementation for the Clopath rule (Clopath et al., 2010) and the Urbanczik-Senn rule (Urbanczik and Senn, 2014) in the spiking network simulator NEST (Jordan et al., 2019). Finally, we show that these implementations reproduce results of the original works and we assess their simulation performance on a distributed computing architecture.

3.1 Time-driven vs event-driven update scheme for synapses with voltage-based plasticity

Let's assume in the following that t_{LS} and t_S denote two consecutive spike times of a presynaptic neuron j . The synaptic weight $W_{ij}(t_S)$ corresponding to the spike at time t_S can be obtained from the weight

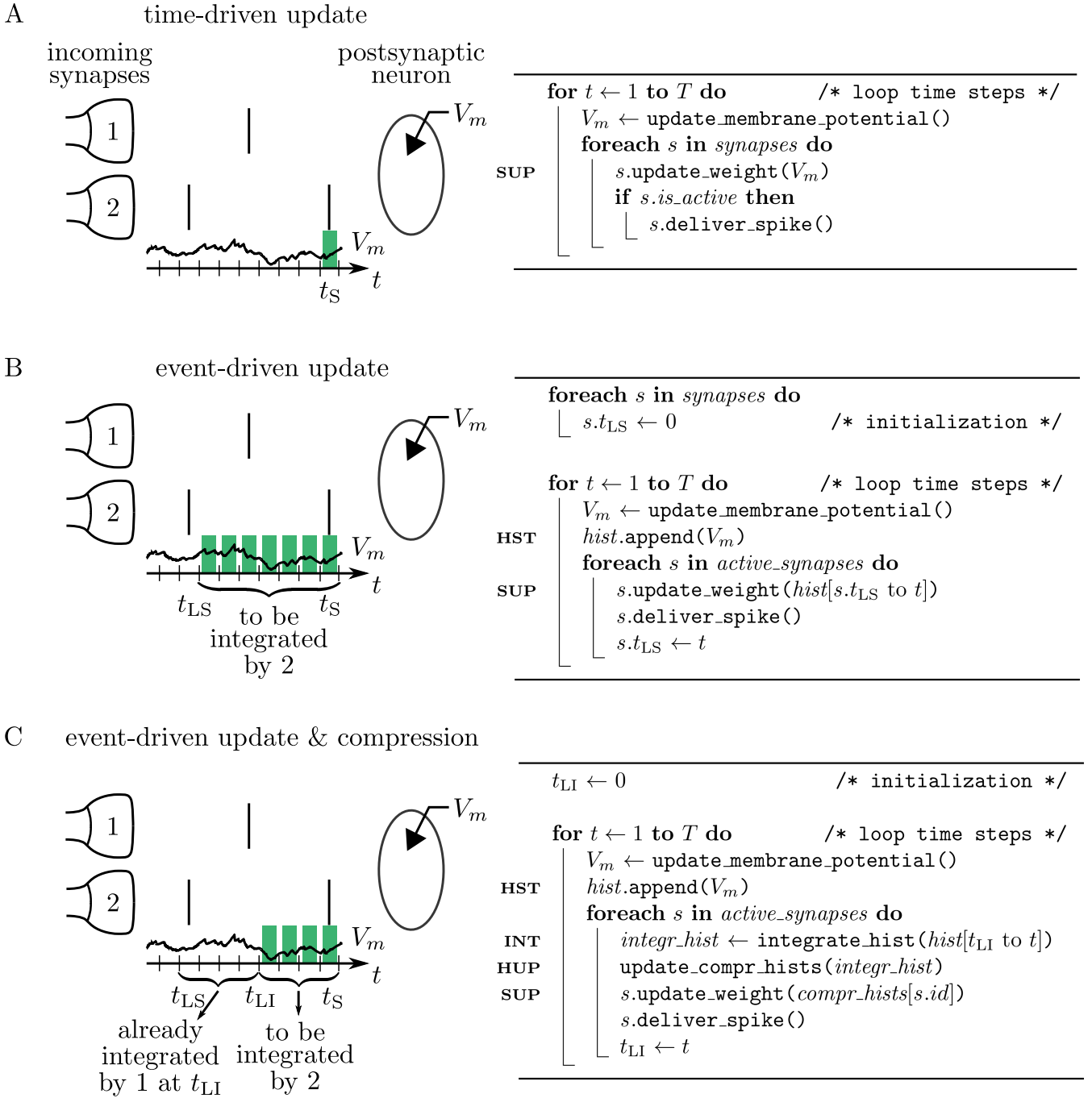


Figure 4. Simulation concepts. Left: illustration of processing the postsynaptic voltage trace $V_m(t)$ for three simulation concepts. Two incoming synapses (1 and 2) transmit spikes (black, vertical bars) to the postsynaptic neuron. Depending on the algorithm, a different number of past membrane potentials has to be stored (green blocks) so that synapse 2 can update its weight when it delivers the spike at time t_s . Right: corresponding pseudocodes. **A** In the time-driven update scheme the synaptic weight change is evaluated in every time step of the simulation for all the synapses. This requires only the latest value of the membrane potential to be accessible by the synapse to update its weight at t_s (see line marked SUP in pseudocode). **B** In the event-driven update scheme the computation of the synaptic weight change is performed only if a spike crosses the synapse. Therefore, storage of the time trace of V_m (see HST in code) from the last spike delivered by synapse 2 at t_{LS} up to the current time step t_s is needed. **C** In the compressed event-driven update scheme synapse 2 uses the time trace of V_m integrated from the last incoming spike at t_{LI} up to the current time step t_s (see INT in code) to complete its weight update (see SUP in code) and also to advance that of synapse 1. The preceding part of V_m from t_{LS} to t_{LI} was already integrated and applied to all incoming synapses (see HUP in code) by synapse 1 when it delivered the spike at t_{LI} .

	time-driven	event-driven	event-driven & compression
history length L	1	I	i
synapse function calls M	$K \cdot T$	$K \cdot T/I$	$K \cdot T/I$
weight change computations C	$K \cdot T$	$K \cdot T$	T
history entry manipulations H	T	T	$K \cdot T/I \cdot i$

Table 1. Comparison of synapse update schemes. From the view point of a postsynaptic neuron, the table shows the maximal length of the history L , the number of function calls M of synapse code, the number of computations C of infinitesimal weight changes $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$, and the number of history entry manipulations H for a simulation of T time steps, a uniform inter-spike interval I between spikes of a single presynaptic neuron, and an in-degree K for each neuron and no delays. For the event-driven compression scheme the entries show the length of the compressed history where i is the number of different spike times within the last inter-spike interval I .

$W_{ij}(t_{LS})$ at the time of the previous spike at t_{LS} and (6) by employing (8) to calculate the latter. As F mixes information of the pre- and postsynaptic neurons, this computation should be done in the synapse. Since there are no spikes in between t_{LS} and t_S , it does not matter when the synapse is performing the updates of its weight. Two possibilities are: 1) Neurons calculate their own s^* and V^* for the current time step and make it accessible to the synapse to enable direct readout and update according to (8) in every time step. This method corresponds to a time-driven update of synapses (Figure 4A). 2) Neurons store a history of s^* and V^* and the synapse reads out this information at t_S , i.e. at the time where the weight update becomes relevant for the network. This method corresponds to an event-driven update of synapses (Figure 4B). Both methods have their advantages and disadvantages analyzed in the following section.

3.1.1 Time-driven scheme

In a time-driven update scheme the information on the membrane potential is directly processed by the synapses such that only the current value of the membrane potential needs to be stored, corresponding to a membrane potential history of length $L = 1$ (Figure 5 and Table 1). For a simulation of T time steps, the history needs to be manipulated $H = T$ times: the single stored value gets updated once per time step. The price that comes with the short history is that synapses need to be updated as often as neurons. This amounts to $M = K \cdot T$ function calls to synapse code for each neuron. Here K denotes the in-degree of each neuron. Each function call of synapse code causes a single computation of $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$, giving rise to in total $C = K \cdot T$ computations per neuron. The membrane potential trace is thus effectively integrated K times; once for each synapse. As both K and T are large numbers in typical simulations of plastic cortical networks, the amount of function calls and computations is therefore large in this setting. The time-driven scheme furthermore forces the execution of synapse code also at time steps where no update would be required, i.e. at time steps, where s_i^*, s_j^*, V_i^* have values for which $\Delta W_{ij}(t^\alpha, t^{\alpha+1}) = 0$. In addition, for delayed connections a history of V_i^* of length $L = d_{\max}$ of the maximal delay d_{\max} measured in simulation steps needs to be stored. We here assume the delay to be on the postsynaptic side; it represents the time the fluctuations of the somatic membrane potential propagate back through the dendrites to the synapses. Therefore, F does not depend on $V_i^*(t)$, but on $V_i^*(t - d_j)$ with a delay d_j encoding the location of the synapse with presynaptic neuron j .

3.1.2 Event-driven scheme

In an event-driven update scheme for synapses, the time trace of the membrane potential V_i^* needs to be stored until all incoming synapses have read out the information to update their weight for a given period. The storage and management of such a history can be expensive in terms of memory and runtime. In each time step, the value of the current membrane potential is appended to the history, leading to $H = T$ history manipulations for a simulation of T time steps. Assuming for simplicity a homogeneous

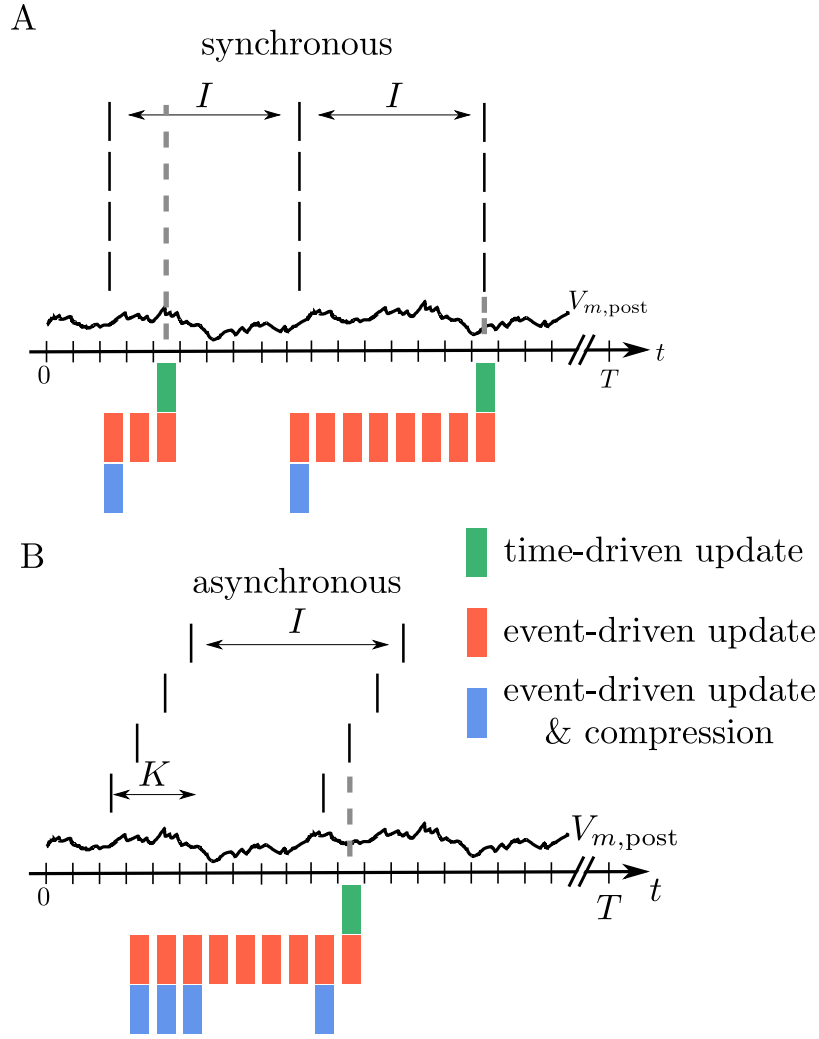


Figure 5. Illustration of buffer sizes for different simulation schemes in case of fully synchronous or asynchronous spikes. **A** All incoming spikes arrive synchronously: In the time-driven scheme the synaptic weight is updated in every time step of the simulation, so that only the current value of $V_{m,\text{post}}$ needs to be available (green). In the event-driven scheme every synapse processes $V_{m,\text{post}}$ from the last spike to the current one. Therefore, the relevant time trace needs to be stored (yellow). In the compressed event-driven scheme this part of $V_{m,\text{post}}$ is processed only once and used to update the weight of all the synapses. Since the weight change is a function of the last spike time which is the same for all the synapses, only one value needs to be updated (red). In this situation the length L of the compressed history is $i = 1$, see [Table 1](#). **B** All incoming spikes arrive in different time bins: For the time-driven and the event-driven scheme the scenario is similar to panel A. For the compressed event-driven scheme the number of values that need to be updated equals the number of incoming synapses K so that $i = K$.

inter-spike interval of I time steps between consecutive spikes of single neurons, we in the following showcase some qualitative history sizes. As synapses need all values of V_i^* in between two consecutive spikes, the maximum history length is $L = I$ ([Figure 5](#)). In case of different firing rates, I corresponds to the maximum inter-spike interval of any of the presynaptic neurons. Synapse code in this scheme is, however, only called in the event of a spike, leading to only $M = K \cdot T/I$ function calls per neuron, where T/I is the number of spikes passing a single synapse during the simulation of T time steps. The total amount of computations C of weight changes $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$ is of course unchanged with respect to the time-driven scheme; they are just split across less function calls ($C = M \cdot L = K \cdot T$). [Table 1](#)

immediately shows the trade-off between memory consumption (length of history) and run time (number of function calls): the event-based scheme consumes more memory, but is faster than the time-driven scheme. Note that since a history of the membrane potential is stored anyway, this scheme is naturally applicable to connections with different delays. A further performance increase can be achieved in plasticity rules, where weight changes only happen under certain conditions on V_i^* : if values $\Delta W_{ij}(t^\alpha, t^{\alpha+1}) \neq 0$ are rare, a non-continuous history can be stored. In such a scenario, time stamps need to be stored alongside the membrane potential to enable synapses to read out the correct time intervals (see Section 3.2.3).

3.1.3 Event-driven compression

The event-driven compression scheme is a modified event-driven scheme that makes use of the fact that for a specific class of plasticity rules the integrated time trace of the membrane potential V_i^* can be used to advance the weight update of all incoming synapses, see Section 2.3.1. Therefore, the time trace of V_i^* stored in the postsynaptic neuron only needs to extend back to the last incoming spike (denoted by t_{LI} in Figure 1 C). This way the history of V_i^* is always short, as the total rate of incoming spikes is high in physiological network states. Due to the dependence of the weight update on the time of the last spike that crossed the synapse, the postsynaptic neuron stores the compressed history of length $L = i$, where i is the number of different spike times within the last inter-spike interval I . (Figure 5). The compressed history is consequently never larger than the history length $L = I$ of the ordinary event-driven scheme (Figure 5 B). For synchronous spikes where the last presynaptic spike time is the same for all synapses, the compressed history, however, contains only one entry (Figure 5 A). Still, synapse code is executed at every spike event, giving rise to $M = K \cdot T/I$ function calls. The full membrane potential trace of length T is effectively only integrated once, amounting to in total $C = T$ infinitesimal weight change computations that are performed in batches in between any two incoming spike events (Table 1). The price for this is that history updates are more expensive: instead of appending a single entry in each time step, at each spike event the full compressed history is updated, giving rise to in total $H = M \cdot i = K \cdot T \cdot i/I$ history entry manipulations, as opposed to $H = T$ in the time- and ordinary event-driven schemes Table 1. In practice, infinitesimal weight change computations are, however, often more costly than history updates, such that the compression algorithm achieves a performance increase (see Section 3.4).

Finally, a drawback of the event-driven compression is that it relies on the fact that all synapses use the same processed membrane potential V_i^* . For distributed delays, $\Delta W_{ij}(t_{LS}, T)$ has a dependence on the presynaptic neuron j via $V_i^*(t - d_j)$. In this case, a separate compressed history needs to be stored for every different delay of connections to the neuron.

3.2 Reference implementation in network simulator with event-based synapse updates

This section describes the implementation of two example voltage-based plasticity rules by Clopath et al. (2010) and Urbanczik and Senn (2014) in a spiking neural network simulator that employs a time-driven update of neurons and an event-based update of synapses. While the naming conventions refer to our reference implementation in the simulation software NEST, the algorithms and concepts presented below are portable to other parallel spiking network simulators.

The Clopath and Urbanczik-Senn rule are chosen as widely used prototypical models of voltage-based plasticity. The differences in the two rules help to exemplify the advantages and disadvantages of the algorithms discussed in Section 3.1. As originally proposed, they are implemented here for two different types of neuron models, Ad-ex/Hodgkin-Huxley point-neurons for the Clopath rule (aeif_psc_delta_clopath/hh_psc_alpha_clopath) and two-compartment Poisson neurons (pp_cond_exp_mc_urbanczik) for the Urbanczik-Senn rule. Extensions to multiple dendritic compartments in the latter case are straight forward. Our implementation of aeif_psc_delta_clopath

follows the reference implementation on ModelDB which introduced a clamping of the membrane potential after crossing the spiking threshold to mimic an action potential. Details can be found in [Section 5.2](#).

The plasticity rules differ in the state variable that is being stored and its interpretation. For the Clopath rule, the stored variable is a thresholded and filtered version of the membrane potential that takes into account characteristics of membrane potential evolution within cells in the vicinity of spike events. The restriction to temporal periods around spikes suggests to implement a history that is non-continuous in time. In contrast, the Urbanczik-Senn rule uses the dendritic membrane potential to predict the somatic spiking; the resulting difference is taken as an error signal that drives learning. This error signal never vanishes and thus needs to be stored in a time-continuous history.

Finally, the proposed infrastructure for storing both continuous and non-continuous histories is generic so that it can also be used and extended to store other types of signals such as external teacher signals.

3.2.1 Exchange of information between neurons and synapses

The implementation of voltage-based plasticity rules in NEST follows the modular structure of NEST, key part of which is the separation between neuron and synapse models. This separation makes it easy for a newly added neuron model to be compatible with existing synapse models and vice versa. A downside is that information, such as values of parameters and state variables, is encapsulated within the respective objects. Simulations in NEST employ a hybrid parallelization scheme: OpenMP threads are used for intra node parallelization and the Message Passing Interface (MPI) for inter node communication. In parallel simulations, synapses are located at the same MPI process as the postsynaptic neurons ([Morrison et al., 2005](#)). Thereby, no communication between MPI processes is needed for the required transfer of information between postsynaptic neurons and synapses to compute weight changes of connections and only one spike needs to be communicated by a given source neuron for all target neurons living on the same MPI process.

The model of STDP requires synapses to access spike times of postsynaptic neurons. In order to provide a standardized transfer of this information across all neuron models that support STDP, in recent years the so-called `Archiving_Node` has been introduced as a parent class of the respective neuron models ([Morrison et al., 2007a](#)). It provides member functions to store and access spike histories. If a neuron model supports STDP, it only needs to be a child of `Archiving_Node` and contain one additional line of code, namely a call of the function `set_spiketime()`, which stores the time of outgoing spike events. We here extended this framework for voltage-based plasticity rules and enhanced the functionality of the archiving node by the member functions `write_history()`, `get_history()` and `compress_history()` to additionally store, read out and manipulate voltage traces or other continuous signals (for Details, see [Section 5.3](#)). To avoid overhead for simulations with only STDP synapses, we introduced two child classes of `Archiving_Node`, `Clopath_Archiving_Node` and `Urbanczik_Archiving_Node`, that each provide containers and functions for the specific histories required for the two plasticity rules. Neuron models that support the respective synapse model then derive from the child classes instead of the root level archiving node.

3.2.2 Delays and `min_delay` communication

All synapses implemented in NEST are so far purely event-driven. To assess the performance of the time-driven update scheme of synapses with voltage-based plasticity, we also implemented a time-driven version of the Clopath and Urbanczik-Senn synapse. Spiking network simulators exploit the delay of connections to reduce communication between compute processes ([Morrison et al., 2005](#)): Instead of sending each spike individually, spikes are buffered and sent in a batch after a certain period. The length of this period, the `min_delay`, corresponds to the minimal delay of all connections in the network. The

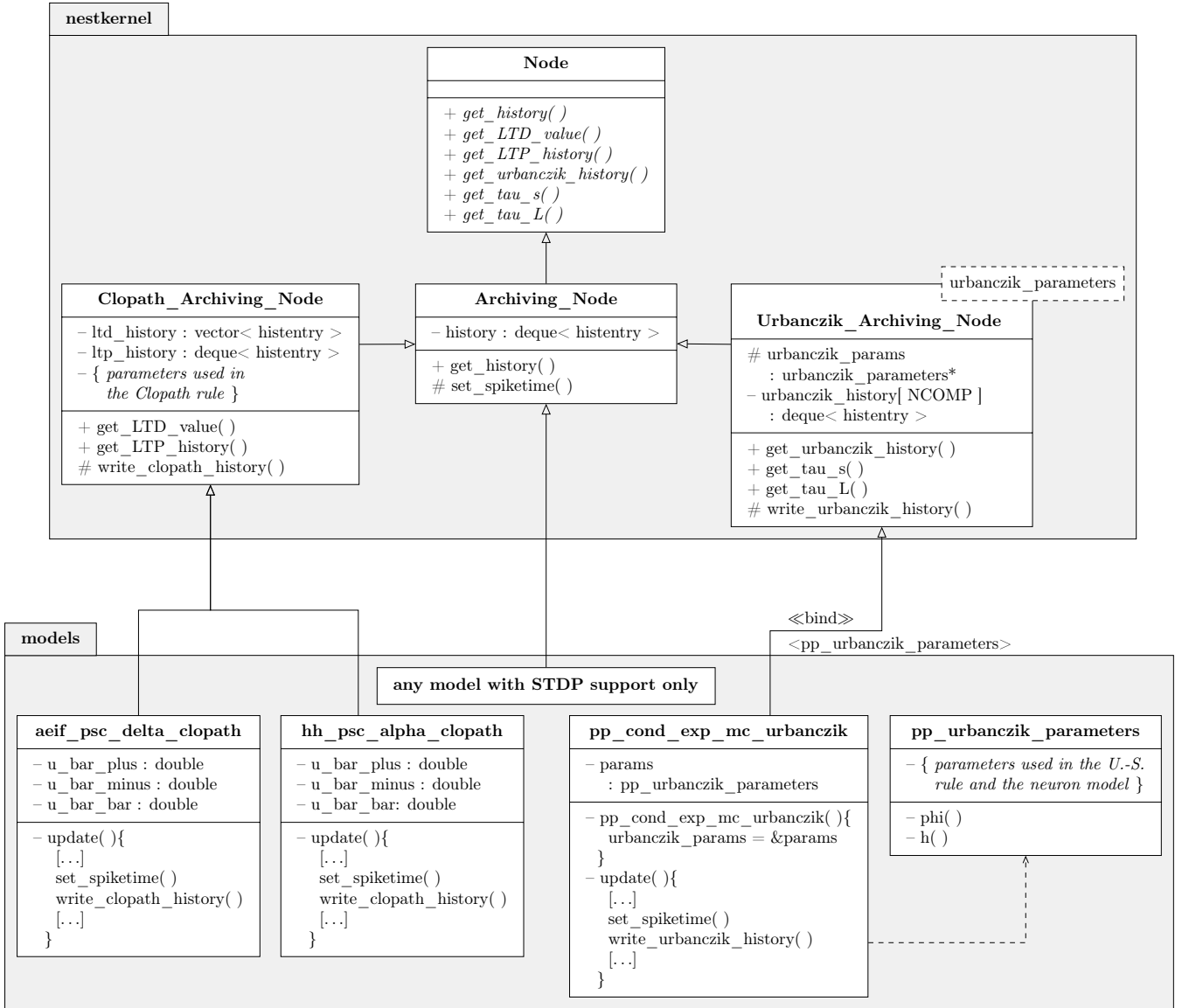


Figure 6. Class diagram of NEST classes and functions. Simplified class diagram for embedding the Clopath (left) and Urbanczik-Senn rule (right) in the NEST infrastructure. The code is distributed across the nestkernel and neuron models. nestkernel contains the base class Node of all neurons models. Models that support ordinary STDP are derived from the Archiving_Node, models that can use the Clopath synapse (aeif_psc_delta_clopath and hh_psc_alpha_clopath) or Urbanczik-Senn synapse (pp_cond_exp_mc_urbanczik) are derived from the Clopath_Archiving_Node or the Urbanczik_Archiving_Node, respectively. The latter add the required functions for storing and managing the history of continuous quantities. The model pp_cond_exp_mc_urbanczik requires a helping class pp_urbanczik_parameters because the Urbanczik_Archiving_Node needs to access functions and parameters that are specific to the neuron model and therefore not located in the Urbanczik_Archiving_Node to keep its implementation more general.

buffering of spikes within this period is possible because the earliest time point that a spike at time t_S can affect the postsynaptic membrane potential is at $t = t_S + \text{min_delay}$. In between t_S and t neurons are decoupled such that their state variables can be propagated forward in time independent of each other and in a batch (Morrison and Diesmann, 2008). We implemented the same min_delay update scheme for synapses, by imposing a function call to time-driven synapses in every min_delay period to update

their synaptic weight. If `min_delay` equals the simulation step size h , this scheme corresponds to the scheme explained in Section 3.1.1. Making use of the `min_delay` infrastructure of NEST speeds up simulations with time-driven synapses in the case $d > h$ as fewer function calls to synapses are needed (see Section 3.4). In case of simulations with synaptic delays, the time-driven update scheme requires the storage of a history of the membrane potential of length `max_delay`.

Storing state variables in event-driven schemes is more complex as the history does not have a fixed length `max_delay`. Instead it needs to be dynamically extended and shortened. A long history can occupy a large amount of memory and its processing by the synapses becomes computationally expensive. Therefore, it is advantageous to optimize the way how information is stored and accessed and how entries that are no longer needed can be identified for deletion. For details of these optimizations in our NEST implementation, see Section 5.3.

As discussed in Section 3.1.3, the event-based compression scheme relies on the fact that all synapses to one postsynaptic neuron employ the same V_i^* . This is not the case if delays of the corresponding connections are distributed. The compression scheme can therefore only be efficient if all delays have a fixed value. If spikes are processed and synapses are updated in a chronological order, then a well-defined segment of the history of V_i^* can be integrated and the compressed history can be updated. In NEST, spikes are, however, buffered within a period of `min_delay` before being sent and processed. Consequently, synapses are not necessarily updated in chronological order. Therefore, the event-based compression scheme can only be implemented in NEST in the case where delays equal the simulation time step. Future work may explore whether the latter restriction could be overcome by sorting all incoming spike events of a given postsynaptic neuron prior to delivery.

3.2.3 Specifics of Clopath plasticity

We implement both an adaptive exponential integrate-and-fire neuron model (`aeif_psc_delta_clopath`) and a Hodgkin-Huxley neuron model (`hh_psc_alpha_clopath`) supporting Clopath plasticity. These implementations consider the filtered versions \bar{u}_\pm of the membrane potential as additional state variables of the neuron. Thereby, they can be included in the differential equation solver of the neurons to compute their temporal evolution. Parameters of κ_\pm consequently need to be parameters of the neuron object rather than the synapse. The same is true for the values of θ_\pm ; they are used in the neuron to determine whether $V_{i,LTP}^*$ and $V_{i,LTD}^*$ evaluate to zero, which systematically happens due to the Heaviside functions in their definitions.

The LTD mechanism is convenient to implement within the event-driven framework: when the synapse is updated at time t , it reads the values $\bar{u}_-(t-d)$ and θ_- from its target and computes the new weight. Here, d denotes the dendritic delay of the connection that accounts for the time it takes to propagate somatic membrane potential fluctuations to the synapse. The archiving node contains a cyclic buffer, also called ring buffer, that stores the history of \bar{u}_- for the past `max_delay` time steps so that the synapse can access a past value of this quantity. Consequently, the LTD history is always short and can be forgotten in a deterministic fashion.

The computation of the weight change due to LTP requires the evaluation of the integral over $V_{i,LTP}^*(t)$. The latter is stored in the archiving node as a vector whose elements are objects that contain three values: the corresponding time t , the value of $V_{i,LTP}^*$ and an access counter that initially is set to zero.

3.2.3.1 Time-driven update:

For simulations with homogeneous delays equal to the simulation time step, the history of $V_{i,LTP}^*$ always contains only a single value as it is read out in every time step by all synapses. For larger delays, the history

is of length `max_delay`, and each synapse reads out a segment of length `min_delay`, increasing the access counter of the corresponding entries by one. For the last synapse that requests a certain segment, the access counter then equals the in-degree K , which is the criterion to delete the corresponding entries from the history. Although for simplicity done in our reference implementation, the time-driven scheme does not require us to store the time stamp t of each history entry. The overhead of this additional number is, however, negligible.

3.2.3.2 Event-driven update:

In event-driven schemes, the history of $V_{i,LTP}^*$ dynamically grows and shrinks depending on the spikes of presynaptic neurons. Since many values of $V_{i,LTP}^*$ are zero, it is beneficial to only store the non-zero values. In this case, a time stamp of each entry is required to assign values of the non-continuous history of $V_{i,LTP}^*$ to their correct times. In case of the uncompressed scheme, when a synapse j is updated at time t_S of a spike, it requests the part of the history between the last spike t_{LS} and the current spike t_S (minus the dendritic delay, see [Section 5.3](#)) from the archiving node. This history segment is then integrated in synapse j and used for its weight update. Each synapse thus integrates the history $V_{i,LTP}^*$ anew ([Section 3.1.2](#)). For the compressed scheme, the history of $V_{i,LTP}^*$ is integrated between the last incoming spike at t_{LI} and the current spike at t_S inside the archiving node. Using this newly integrated time trace, the weight of synapse j is updated and the compressed history for all other last spike times is advanced. Afterwards the history of $V_{i,LTP}^*$ is deleted. Thereby, $V_{i,LTP}^*$ is only integrated once for all synapses.

In any case, the integrated history of $V_{i,LTP}^*$ needs to be combined with the presynaptic spike trace s_j^* . The latter is easily computed analytically inside the synapse because it is an exponential decay of the corresponding value at the time of the last spike. At the end of the update process the trace is increased by τ_s^{-1} to account for the trace of the current spike, where τ_s is the time constant of the kernel κ_s .

3.2.4 Specifics of Urbanczik-Senn plasticity

Following the original publication ([Urbanczik and Senn, 2014](#)), we implement a Poisson spiking neuron model (`pp_cond_exp_mc_urbanczik`) supporting Urbanczik-Senn plasticity. One peculiarity of this model is that the gain function ϕ that translates the membrane potential into a firing rate also enters the plasticity rule through V^* . Therefore ϕ as well as its parameters need to be known by the neuron and the synapse. Creating an additional helper class (`pp_urbanczik_parameters`) as a template argument for the corresponding archiving node (`Urbanczik_Archiving_Node`) and neuron model (`pp_cond_exp_mc_urbanczik`) solves this problem ([Figure 6](#)): it contains all parameters and functions required by both classes. As explained in [Section 2.5](#), the representation (17) is more beneficial for implementing the Urbanczik-Senn rule than that of (16). The first two integrals in (17) only extend from t to T ; history entries for times smaller than t are not needed and can be deleted after the corresponding update. The dependence on the full history back until 0 arising from the convolution with κ is accumulated in the last term in (17), which the synapse computes with the help of storing one additional value $I_2(0, t)$. At the end of a weight update this value is overwritten by the new value $I_2(0, T) = e^{-\frac{T-t}{\tau\kappa}} I_2(0, t) + I_2(t, T)$ which is then used in the next update. Either the synapse (time- and event-driven update) or the archiving node (event-driven compression) compute the two integrals I_1 and I_2 but in all cases the archiving node stores the history of $V_i^*(t)$.

3.3 Reproduction of results in literature

The reference implementation of the Clopath plasticity reproduces the results from [Clopath et al. \(2010\)](#) on the frequency dependence of weight changes in spike-pairing experiments and the emergence of bidirectional connections in small all-to-all connected networks ([Figure 7](#)). The setup of the spike-pairing experiment in [Figure 7A](#) consists of two neurons connected via a plastic synapse. The pre- and postsynaptic

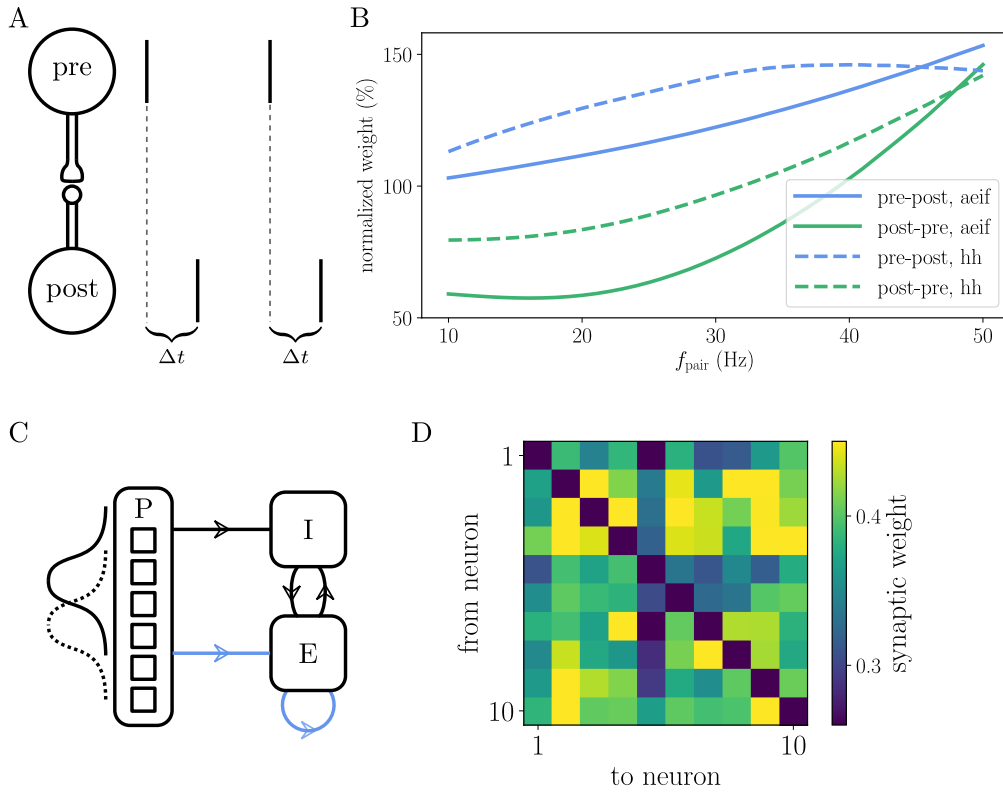


Figure 7. Reproduction of results with Clopath rule. **A** Setup of the spike pairing experiment. Two neurons (“pre” and “post”) that are connected by a plastic synapse receive input so that they spike one after another with a delay Δt . The change of the synaptic weight is computed according to the Clopath rule as a function of the frequency f_{pair} with which the spike pairs are induced. **B** Result of the spike pairing experiment. The relative change of the synaptic weight after five spike pairs as a function of f_{pair} is shown for two different neuron models (aeif: solid lines, Hodgkin-Huxley: dashed lines). The blue lines represent a setup where the postsynaptic neuron fires after the presynaptic one (pre-post, $\Delta t = 10$ ms) and the green lines represent the opposite case (post-pre, $\Delta t = -10$ ms). This panel corresponds to figure 2b of [Clopath et al. \(2010\)](#). **C** Setup of the network that produces strong bidirectional couplings. The network consists of an inhibitory (I) and an excitatory (E) population which receive Poisson spike trains (P) as an external input. The firing rate of the latter is modulated with a Gaussian shape whose center is shifted every 100 ms. The external input connections to the excitatory population are plastic as well as the connections within the excitatory population (indicated by blue arrows). **D** Synaptic weights of the all-to-all connected excitatory neurons after the simulation of the network. Strong bidirectional couplings can be found, e.g. between neurons 2 and 3, 2 and 9, and 4 and 7. The setup of this experiment is similar to that shown in figure 5 of [Clopath et al. \(2010\)](#). A more detailed description of the two experiments can be found in [Section 5.4](#).

neuron are forced to spike with a time delay of Δt multiple times which leads to a change in synaptic weight that depends on the frequency of the spike pairs ([Figure 7B](#)). The setup of the small network is shown in [Figure 7C](#). The weights of the plastic synapses within the recurrently connected excitatory population are initialized all to the same value. At the end of the simulation during which the network receives a time varying input, some pairs of neurons show strong bidirectional connections ([Figure 7D](#)). See [Appendix Section 5.4](#) for details on the setup of both experiments as implemented in NEST.

The basic use of the Urbanczik-Senn rule in NEST is exemplified in [Figure 8](#) which shows the reproduction of a simple learning experiment from the original publication ([Urbanczik and Senn, 2014](#)). Here the neuron is supposed to transform spike patterns in the input to the dendritic compartment into a sinusoidal modulation of the somatic membrane potential. This target potential is determined by an external teaching

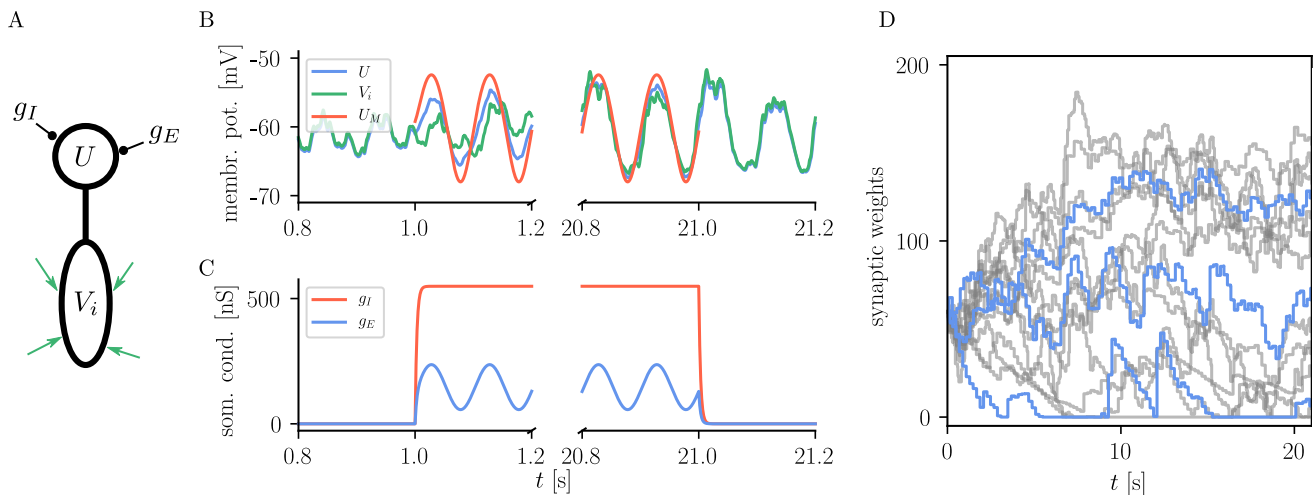


Figure 8. Reproduction of results with Urbanczik-Senn rule. **A** Setup of a simple learning task using the Urbanczik-Senn plasticity rule. The somatic conductances g_I and g_E of a two-compartment neuron are modulated such that they induce a teaching signal with sinusoidal shape. The dendrite receives a repeating spike pattern as an input via plastic synapses (green arrows). **B** The synapses adapt their weights so that the somatic membrane potential U (blue) and the dendritic prediction V_i (green) follow the matching potential U_M (red) after learning. **C** Excitatory (g_E) and inhibitory (g_I) somatic conductances that produce the teaching signal. **A** and **B** corresponds to figure 1 of [Urbanczik and Senn \(2014\)](#). **D** Temporal evolution of the synaptic weights during learning. For the sake of better overview, only a subset of weights is shown (gray) with three randomly chosen time traces highlighted in blue. Synapses in NEST fulfill Dale’s principle which means that a weight update cannot convert an excitatory into an inhibitory synapse and vice versa giving rise to the rectification at zero.

signal during learning. Via minimizing the error between the dendritic prediction of the somatic membrane potential and the actual somatic membrane potential, weights of dendritic synapses are organizing such that the neuron can produce the desired membrane potential. There is, however, no stop-learning region in the Urbanczik-Senn rule (for a modified version, see [Cartiglia et al. 2020](#)): The error never vanishes completely which causes weights to keep changing despite an overall good approximation of the target signal. Details of the experiment and NEST setup can be found in Appendix [Section 5.6](#).

3.4 Performance of the reference implementations

3.4.1 Clopath plasticity

In order to evaluate the performance of the implementation of the Clopath rule in NEST, in a weak-scaling setup, we simulate excitatory-inhibitory networks of increasing size, but fixed in-degree K . As we expect the performance to critically depend on the number of synapses, we examine two scenarios: a small in-degree $K = 100$, and a rather large in-degree $K = 5000$. While the first case might be suitable for small functional networks, the latter in-degree represents a typical number for cortical networks. Further details on network and simulation parameters are given in supplementary Table S5. As a reference, we also simulate the same network with STDP synapses, which require much less computations as they rely solely on spike times. To achieve the same network state, that is the same spikes, for the different connectivity rules, we impose the weights to stay constant across time by setting learning rates to zero. This way all computations for weight changes are being performed, but just not applied. This has the additional advantage that reasonable asynchronous irregular network states are simple to find based on predictions for static synapses ([Brunel, 2000](#)).

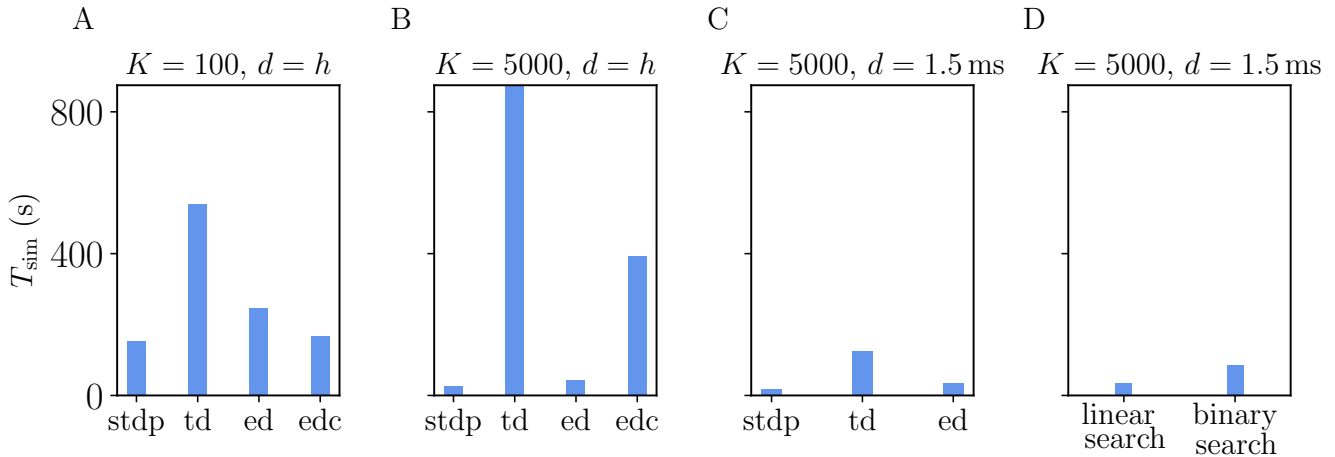


Figure 9. Comparison of simulation times T_{sim} for excitatory-inhibitory networks with different implementations of the Clopath plasticity in NEST. Simulation times exclude network building and only account for updates of the dynamical state of the system. The following implementations are shown: “stdp”: standard implementation of STDP synapse, “td”: time-driven implementation of Clopath synapse, “ed”: event-driven scheme as included in NEST 2.18, “edc”: event driven compression. **A** Network of size $N = 1.92 \cdot 10^6$ with small in-degree $K = 100$ and all synapses having a delay d equal to the resolution of the simulation $h = 0.1$ ms. **B** Network of size $N = 1.54 \cdot 10^5$ with large in-degree $K = 5000$ and $d = h$. **C** Same network as in panel B but $d = 1.5$ ms (for $d > h$ “edc” not compatible with NEST, see Section 3.1.3). In A, B, and C both “ed” and “edc” use linear search of the history and access counters, see 5.3. **D** Comparison between “ed”-implementations using linear search and binary search of the history. All simulations use 768 threads distributed over 32 compute nodes each running one MPI process. Further parameters as in supplementary Table S5.

The Clopath rule has originally been proposed for connections without delays (Clopath et al., 2010). Therefore, we first evaluate its performance in this setting (delay equals simulation time step), which is, however, not the natural setting for a simulator like NEST that makes use of delays to speed up communication between compute processes. The first observation is that, as expected, simulations with Clopath synapses are slower than those with ordinary STDP (Figure 9). Given the update of synapses in every simulation step, the time-driven scheme for Clopath synapses is much slower than the event-driven scheme (Figure 9A). The difference becomes larger the more synapses there are (Figure 9B). Introducing a delay leads to less function calls to synapses (once every `min_delay`) and therefore increases the speed of the time-driven scheme (Figure 9C). Its simulation times, however, remain much above the event-driven scheme. This comparison illustrates the benefit of event-driven updates for Clopath synapses.

How does compression of the history change the picture? As discussed in Section 3.1.3, compression has the advantage of not integrating the membrane potential history for each synapse separately. A downside of the event-based compression is that it requires storing one history entry for each last spike time of presynaptic neurons. For large in-degrees, this history is therefore longer than the history of $V_{i,\text{LTP}}^*$, which we implemented as non-continuous for the Clopath rule. Consequently, the event-based compression scheme only outperforms the ordinary event-driven scheme for small in-degrees (Figure 9A), but not for large in-degrees (Figure 9B). Given that the compression can only be implemented in NEST for connections with delay equal to the resolution of the simulation (see Section 3.2.2), the method of choice is therefore the ordinary event-driven scheme (Section 3.1.2). Although a bit slower, its run-time is on the same order of

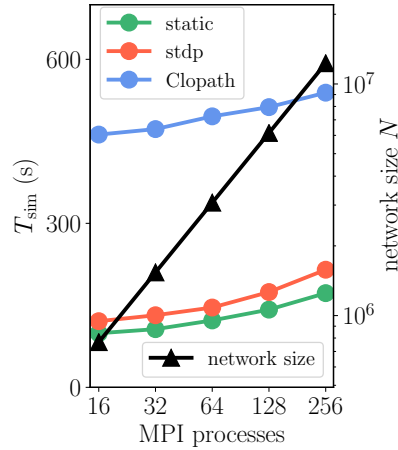


Figure 10. Scaling of simulation time T_{sim} with network size for 2 s of biological time: Clopath plasticity. Weak scaling: computational resources (horizontal axis) increase proportionally to network size N (black curve and triangles, right vertical axis). Event-driven scheme for Clopath rule (blue) compared to static (green) and STDP synapse (red). Network and simulation parameters as in supplementary Table S5 with in-degree $K = 5000$. For all simulations each compute node runs one MPI process with 24 threads.

magnitude as the ordinary STDP synapse, with similar weak-scaling behavior (Figure 10). The additional computations with respect to STDP result in a constant overhead.

Another advantage of having short non-continuous histories is that searching the history at readout is fast. A simple linear iteration scheme is therefore even faster than a binary search (Figure 9D) because the latter search requires an additional list of presynaptic spike times (see Section 5.3) which is unnecessary overhead in this scenario. As a result the ordinary event-driven scheme with linear history iteration is the most general and efficient scheme and therefore integrated into NEST 2.18 (Jordan et al., 2019).

3.4.2 Urbanczik-Senn plasticity

The Urbanczik-Senn rule, in its original version, does not account for delays in connections (Urbanczik and Senn, 2014). As for the Clopath rule, we therefore first evaluate its performance for connections with delays that equal the simulation time step. We compare the results to networks with ordinary STDP synapses, again setting all learning rates to zero to maintain the same network state across different types of plasticity. Naturally, the processing of the membrane potential information makes the Urbanczik-Senn plasticity less efficient to simulate than networks with ordinary STDP synapses (Figure 11). Note that the absolute numbers of simulation times are not directly comparable to simulations with Clopath plasticity (Figure 9) as network sizes are smaller here (Table S5 in supplement). Networks with small and large in-degrees behave qualitatively similar: given the long continuous history that needs to be stored and read out, the event-driven scheme does not significantly outperform the time-driven scheme (Figure 11A,B). In the network with small in-degree, the time-driven scheme is even slightly faster (Figure 11A). This behavior reverses for large in-degrees as the number of synapse calls grows stronger than the length of the history (Figure 11B). However, given that the length of the history is so critical in this rule, the compression algorithm can in both cases achieve a significant increase in performance (Figure 11A,B). This performance increase is larger the smaller the in-degree, as the compressed history becomes shorter (Figure 11A). Due to current NEST specifics (see Section 3.2.2), the compression algorithm cannot be used in settings with delays that are larger than the simulation time step (Figure 11C): Here, as expected, the time-driven scheme becomes faster than in the $d = h$ case, but it is in general still comparable in performance to the

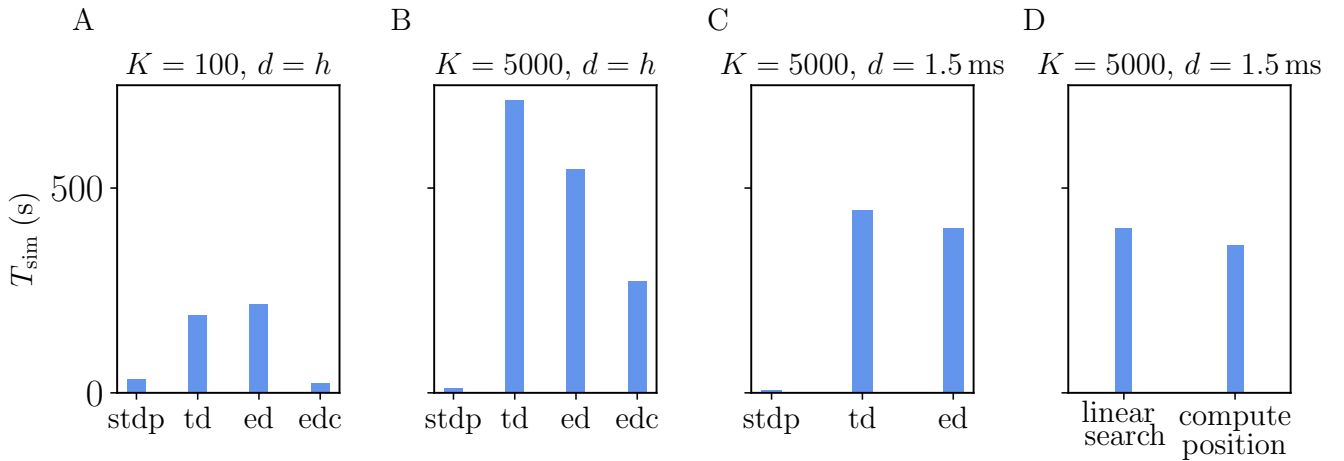


Figure 11. Comparison of simulation times T_{sim} for excitatory-inhibitory networks with different implementations of the Urbanczik-Senn plasticity in NEST. The following implementations are shown: “stdp”: standard implementation of STDP synapse in NEST, “td”: time-driven implementation of Urbanczik-Senn synapse, “ed”: event-driven scheme, “edc”: event driven compression. **A** Network of size $N = 3.84 \cdot 10^5$ with small in-degree $K = 100$ and all synapses having a delay d equal to the resolution of the simulation $h = 0.1 \text{ ms}$. **B** Network of size $N = 3.84 \cdot 10^4$ with large in-degree $K = 5000$ and $d = h$. **C** Same network as in panel B but $d = 1.5 \text{ ms}$ (for $d > h$ “edc” not compatible with NEST, see Section 3.1.3). In A, B, and C both “ed” and “edc” use linear search of the history and the access counters, see 5.3. **D** Comparison between “ed”-implementations using linear search and binary search of the history. All simulations use 768 threads distributed over 32 compute nodes each running one MPI process. Details on network parameters in Table S5 in supplement.

event-driven scheme. The latter is therefore the method of choice for simulations with delayed connections; for zero-delay connections, the compression algorithm performs best. Whether the history readout is done via linear iteration or via computing positions of history entries has no significant impact on the performance (Figure 11D). Therefore, the simple linear iteration is integrated in NEST 3.

We furthermore employ a weak-scaling setup with excitatory-inhibitory networks of increasing size and fixed in-degree $K = 5000$ (Figure 12A,B, and Table S5 in supplement). Apart from a constant offset, the scaling of simulation time T_{sim} for updating neurons and synapses is similar for Urbanczik, static and STDP synapses. With increasing network size N and proportionally increasing number of MPI processes, T_{sim} rises only slightly (Figure 12B), indicating almost ideal weak-scaling behavior. The constant offset in T_{sim} is larger than for Clopath synapses as the Urbanczik-Senn rule requires longer histories of membrane potentials and a more extensive history management.

3.5 Conclusions

The analyses of the Clopath and the Urbanczik-Senn plasticity as prototypical examples for rules that rely on storage of discontinuous versus continuous histories show that the former are much faster to simulate, in particular for large networks that require distributed computing. For discontinuous histories, the event-driven scheme is most generally applicable and efficient, which makes corresponding rules easy to integrate into modern simulators with event-based synapses. The performance gap between the different rules should be kept in mind in the design of new learning rules. Furthermore, it is worthwhile to test modifications of existing learning rules to decrease the amount of stored information.

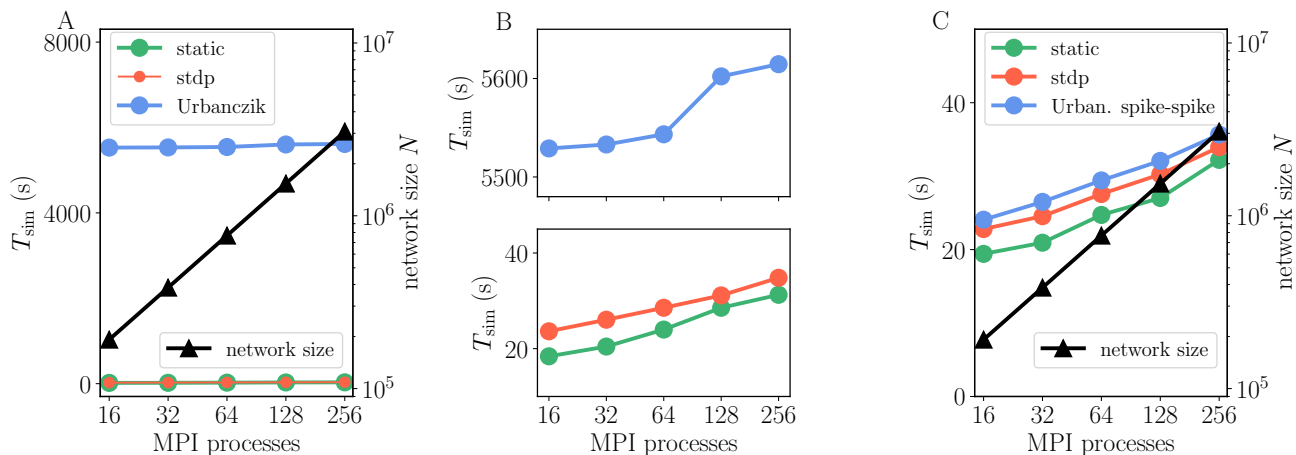


Figure 12. Scaling of simulation times T_{sim} with network size for 2 s of biological time: Urbanczik-Senn plasticity. Same weak scaling as in Figure 10. **A** Event-driven Urbanczik-Senn rule (blue) compared to static (green) and STDP synapse (red). On the scale of the vertical axis the red curve (STDP synapses) falls on top of the green curve (static synapses), indicated by finer line width and marker size of the former. **B** Same simulation time data as in A but with a smaller range on the vertical axis. Upper panel: enlargement of Urbanczik-Senn data. Lower panel: enlargement of data for static and STDP synapses. **C** Spike-spike version of the Urbanczik-Senn rule compared to static and STDP synapse. Network and simulation parameters as in supplementary Table S5 with in-degree $K = 5000$. For all simulations each compute node runs one MPI process with 24 threads.

For illustration, we here test a spike-based alternative to the original Urbanczik-Senn rule, where we replace the rate prediction $\phi(V_i(t))$ in V^* of (15) by a noisy estimate, which we generate by a non-homogeneous Poisson generator with rate $\phi(V_i(t))$, see Section 5.7. The prediction error then results in a comparison of somatic and dendritic spikes, s_i and s_i^{dend} , respectively; it is therefore purely based on point processes. In terms of storage and computations, the rule thereby becomes similar to ordinary STDP (cf. (5)). This becomes apparent in the weak-scaling experiment in Figure 12C, which shows that the modification of the learning rule results in a speedup of a factor 10 to 30 arriving essentially at the same run time as the ordinary STDP rule.

When changing learning rules to improve the efficiency of an implementation, the question is in how far the modified rule, in our example including the noisy estimate of the dendritic prediction, still fulfills the functionality that the original rule was designed for. Generally, without control of the error any simulation can be made arbitrarily fast. Therefore Morrison et al. (2007b) define efficiency as the wall-clock time required to achieve a given accuracy. We test in the appendix (Figure 13) whether the dynamics is still robust enough to achieve proper learning and function in the reproduced task of Figure 8. The learning works as well as in the original Urbanczik-Senn rule. However, given the simplicity of the chosen task, this result may not generalize to other more natural tasks. We leave a more detailed investigation of this issue to future studies. The basic exploration here, however, illustrates how taking into account the efficiency of implementations can guide future development of learning rules to make them practically usable for large-scale simulations of brain networks.

4 DISCUSSION

This work presents efficient algorithms to implement voltage-based plasticity in modern neural network simulators that rely on event-based updates of synapses (for a review, see Brette et al., 2007). This

update scheme restricts function calls of synapse code to time points of spike events and thereby improves performance in simulations of biologically plausible networks, where spike events at individual synapses are rare and the total number of synapses is large compared to the number of neurons. In complex voltage-based plasticity rules, synapses, however, rely on continuous information of state variables of postsynaptic cells to update their strength, which naturally suggests their time-driven update. Instead, we here propose an efficient archiving of voltage traces to enable event-based synapse updates and detail two schemes for storage, read out and post-processing of time-continuous or discontinuous information. We show their superior performance with respect to time-driven update both theoretically and with a reference implementation in the neural network simulation code NEST for the rules proposed in [Clopath et al. \(2010\)](#) and [Urbanczik and Senn \(2014\)](#).

Event-driven update schemes for voltage-based plasticity come at the expense of storing possibly long histories of a priori continuous state variables. Such histories not only require space in memory but they also affect the runtime of simulations, which we focus on here. The time spent for searching and post-processing the history to calculate weight updates increases with increasing length, and these operations have to be done for each synapse. Therefore, in addition to an ordinary event-driven scheme, we devised a compression scheme that becomes superior for long histories as occurring in the Urbanczik-Senn rule. In particular for networks with small in-degrees or synchronous spiking, the compression scheme results in a shorter history. It further reduces the total amount of computations for weight changes by partially re-using results from other synapses thereby avoiding multiple processing of the history. For short histories as occurring in the Clopath rule, the compression results in unnecessary overhead and an increase in history size as one entry per last presynaptic spike time needs to be stored instead of a discontinuous membrane potential around sparse postsynaptic spike events. We here, for simplicity, contrasted time- and event-driven update schemes. However, further work could also investigate hybrid schemes, where synapses are not only updated at spike events, but also on a predefined and coarse time grid to avoid long histories and corresponding extensive management. A similar mechanism is used in [Kunkel et al. \(2011\)](#) to implement a normalization of synaptic weights. The corresponding technical details can be found in [Kunkel \(2015, ch. 5.2\)](#).

The storage and management of the history as well as complex weight change computations naturally reduce the performance of simulations with voltage-based plasticity in comparison to static or STDP synapses. The latter only require information on spike times which is much less data compared to continuous signals. Nevertheless, given that the Clopath rule is based on thresholded membrane potentials and consequently short, discontinuous histories, the performance and scaling of the event-driven algorithms is only slightly worse than for ordinary STDP. Time-driven implementations cannot employ this model feature and update weights also in time steps where no adjustment would be required, leading to significantly slower simulations. The performance gain of using event-driven schemes is less pronounced for the Urbanczik-Senn rule as, by design, histories are typically long. In this case, the compression scheme naturally yields better results in terms of runtime. Our own modification of the Urbanczik-Senn rule only requires storage of sparsely sampled membrane potentials, giving rise to the same performance as STDP. Generally, an algorithm is faster if it requires less computations. However, opportunities for vectorization and cache efficient processing, outside of the scope of the present manuscript, may change the picture.

We here chose the Clopath and the Urbanczik-Senn rule as two prototypical models of voltage-based plasticity. While both rules describe a voltage dependence of weight updates, their original motivation as well as their specific form are different: The Clopath rule refines standard STDP models to capture biologically observed phenomena such as frequency dependence of weight changes ([Sjöström et al., 2001](#)).

For this it is sufficient to take into account membrane potential traces in the vicinity of spike events, leading to storage of time-discontinuous histories in our implementation. In contrast, the Urbanczik-Senn rule is functionally inspired by segregating dendritic and somatic compartments of cells and using the difference between somatic output and dendritic prediction as a teacher signal for dendritic synapses. The teacher signal is by construction never vanishing, imposing the need to store a time-continuous history. The original publications of both rules had a great and long-lasting impact on the field. The Clopath rule has been used in a variety of studies ([Clopath and Gerstner, 2010](#); [Ko et al., 2013](#); [Litwin-Kumar and Doiron, 2014](#); [Sadeh et al., 2015](#); [Bono and Clopath, 2017](#); [Maes et al., 2020](#)), partly in modified versions which are, however, still compatible with the here presented simulation algorithms. The same holds for the Urbanczik-Senn rule ([Brea et al., 2016](#); [Sacramento et al., 2018](#)).

The current implementation, which is published and freely available in NEST 2.20.1, supports an adaptive exponential integrate-and-fire and a Hodgkin-Huxley neuron model for the Clopath rule. The former is used in the original publication ([Clopath et al., 2010](#)) and the latter appears on ModelDB ([Hines et al., 2004](#)) in code for the Clopath rule for the NEURON simulator ([Hines and Carnevale, 2001](#)). For the Urbanczik-Senn rule, NEST currently supports the two-compartment Poisson model neuron of the original publication ([Urbanczik and Senn, 2014](#)). A three-compartment version as used in [Sacramento et al. \(2018\)](#) or other models are straight forward to integrate into the current simulation framework. However, with voltage-based plasticity rules, borders between neurons and synapses become blurred as these rules often depend on specifics of the employed neuron models rather than only spike times as for standard STDP. Consequently, archiving nodes might need to have specific functionalities, which, in light of the zoo of existing neuron models, could easily lead to a combinatorial explosion of code. These problems can in future be overcome with automatic code generation using NESTML that only creates and compiles code that is needed for the specified model simulations ([Plotnikov et al., 2016](#)).

While the here presented implementation refers to the neural network simulator NEST ([Gewaltig and Diesmann, 2007](#)), the proposed algorithms and simulation infrastructure are compatible with any network simulator with event-driven update of synapses, such as, for example, NEURON ([Lytton et al., 2016](#), cf. ch. 2.4) and Brian2 ([Stimberg et al., 2014](#)). Furthermore, applicability is not restricted to the Clopath and Urbanczik-Senn rule, but the framework can be adapted to any other learning rule that relies on state variables of postsynaptic neurons. State variables hereby not only encompass membrane potentials such as, for example, in the LCP rule by [Mayr and Partzsch \(2010\)](#), the Convallis rule by [Yger and Harris \(2013\)](#), the voltage-triple rule by [Brea et al. \(2013\)](#), the MPDP rule by [Albers et al. \(2016\)](#), the voltage-gated learning rules by [Brader et al. \(2007\)](#), [Sheik et al. \(2016\)](#), [Qiao et al. \(2015\)](#), [Diederich et al. \(2018\)](#) and [Cartiglia et al. \(2020\)](#), or the branch-specific rule by [Legenstein and Maass \(2011\)](#), but also, for example, firing rates of stochastic neuron models or rate models ([Brea et al., 2016](#); [Sacramento et al., 2018](#)), or other learning signals ([Neftci et al., 2017](#); [Bellec et al., 2019](#)). The infrastructure in NEST allows for the storage of time-continuous and discontinuous histories and therefore poses no restrictions on the dependence of the learning rule on the postsynaptic state variables. The here developed machinery could be also used to store external teacher signals that are provided to model neurons by stimulation devices mimicking brain or environmental components not explicitly part of the model. Since synapses are located at the compute process of the postsynaptic neuron, readout of state variables from presynaptic neurons comes with large costs in simulations on distributed computing architectures and is therefore not considered here. Due to specifics of the present NEST code in spike delivery, the event-driven compression proposed here is only applicable in NEST for delays that equal the simulation time step. Such a restriction can be readily overcome in a simulation algorithm that performs a chronological update of synapses.

In general, one has to distinguish two types of efficiency in the context of simulating plastic networks: Firstly, the biological time it takes the network to learn a task by adapting the weights of connections. Secondly, the wall-clock time it takes to simulate this learning process. Both times crucially depend on the employed plasticity rule. In this study, we focus on the wall-clock time and argue that this can be optimized by designing learning rules that require storing only minimal information on postsynaptic state variables. Ideally, the plasticity rule contains unfiltered presynaptic or postsynaptic spike trains to reach the same performance as in ordinary STDP simulations. If rules, however, need to capture the pre- and post-spike dynamics of postsynaptic neurons, it is beneficial to make use of thresholded state variables as in the example of the Clopath rule as this yields short, time-discontinuous histories. Reducing the amount of information available for synapses to adjust their weights can in general slow down the learning. We present a modification of the Urbanczik-Senn rule where the dendritic prediction of the somatic firing contains an additional sampling step with Poisson spike generation. This modification significantly reduces the simulation time. For the here presented simple task, learning speed is largely unaffected, but generally a performance decrease is to be expected when error signals become more noisy. Therefore, there is a trade-off between learning speed and simulation speed, which should be considered in the design process of new learning rules. [Cartiglia et al. \(2020\)](#) propose another modification of the Urbanczik-Senn rule underlying the model in [Sacramento et al. \(2018\)](#): this simplification only requires postsynaptic membrane potentials at the time of spike events, which makes the rule much more efficient to simulate and applicable to neuromorphic hardware. [Bono and Clopath \(2017\)](#) simplify the Clopath rule in an analogous fashion to allow for its event-based simulation in the spiking network simulator Brian ([Stimberg et al., 2014](#)). Our general framework supports systematic testing of such simplifications in terms of simulation performance and functionality.

For the plasticity rules by [Clopath et al. \(2010\)](#) and [Urbanczik and Senn \(2014\)](#), we present a highly scalable reference implementation that is published and freely available in NEST 2.20.1. The parallelism of the NEST implementation enables simulations of plastic networks of realistic size on biologically plausible time scales for learning. The field of computational neuroscience recently entered a new era with the development of large-scale network models ([Markram et al., 2015](#); [Schmidt et al., 2018](#); [Billeh et al., 2020](#)). Emulating the dynamics of cortical networks, such models are so far restricted to static connections. We here provide simulation algorithms for plasticity mechanisms that are required for augmenting such complex models with functionality. It is our hope that incorporating both biologically and functionally inspired plasticity models in a single simulation engine fosters the exchange of ideas between communities towards the common goal of understanding system-level learning in the brain.

CODE AVAILABILITY

The reference implementation for the event-driven update scheme of synapses with Clopath and Urbanczik-Senn plasticity was reviewed by the NEST initiative and is publicly available in NEST 2.20.1. The PyNEST code for model simulations and Python scripts for the analysis and visualization of results are fully available at <https://doi.org/10.5281/zenodo.4565188>.

CONFLICT OF INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ACKNOWLEDGMENTS

We thank Claudia Clopath and Wulfram Gerstner for explaining details of their reference implementation and the underlying biological motivation. Moreover, we thank Hedyeh Rezaei and Ad Aertsen for suggesting to implement the Clopath rule in NEST and Charl Linssen, Alexander Seeholzer, Renato Duarte for carefully reviewing our implementation. Finally we thank Walter Senn, Mihai A. Petrovici, Laura Kriener and Jakob Jordan for fruitful discussions on the Urbanczik-Senn rule and our proposed spike based version. We further gratefully acknowledge the computing time on the supercomputer JURECA (**Jülich Supercomputing Centre, 2015**) at Forschungszentrum Jülich granted by firstly the JARA-HPC Vergabegremium (provided on the JARA-HPC partition, jinb33) and secondly by the Gauss Centre for Supercomputing (GCS) (provided by the John von Neumann Institute for Computing (NIC) on the GCS share, hwu12). This project has received funding from the Helmholtz Association Initiative and Networking Fund under project number SO-092 (Advanced Computing Architectures, ACA) and the European Union’s Horizon 2020 research and innovation programme under grant agreements No 785907 (HBP SGA2) and No 945539 (HBP SGA3). Partly supported by the Helmholtz young investigator group VH-NG-1028. All network simulations carried out with NEST (<http://www.nest-simulator.org>). This work was supported by the Jülich-Aachen Research Alliance Center for Simulation and Data Science (JARA-CSD) School for Simulation and Data Science (SSD).

5 APPENDIX

5.1 Analytical integration in Urbanczik-Senn rule

To derive (17) it is convenient to first investigate $\Delta W_{ij}(0, t)$ and $\Delta W_{ij}(0, T)$ and then compute $\Delta W_{ij}(t, T) = \Delta W_{ij}(0, T) - \Delta W_{ij}(0, t)$. Assuming that the simulation starts at $t = 0$, the weight change from the start to time t is given by

$$\begin{aligned} \Delta W_{ij}(0, t) &= \eta \int_0^t dt' \int_0^{t'} dt'' \kappa(t' - t'') V_i^*(t'') s_j^*(t'') \\ &= \eta \int_0^t dt'' \int_{t''}^t dt' \kappa(t' - t'') V_i^*(t'') s_j^*(t'') \\ &= \eta \int_0^t dt'' [\tilde{\kappa}(t - t'') - \tilde{\kappa}(0)] V_i^*(t'') s_j^*(t'') \\ &= \eta [-I_2(0, t) + I_1(0, t)] \end{aligned}$$

where we exchanged the order of integration from the first to the second line. In the third line we introduced $\tilde{\kappa}(t)$ defined by $\kappa(t) = \frac{\partial}{\partial t} \tilde{\kappa}(t)$ and in the fourth line we defined the two integrals

$$I_1(t_1, t_2) = - \int_{t_1}^{t_2} dt' \tilde{\kappa}(0) V_i^*(t') s_j^*(t'), \quad I_2(t_1, t_2) = - \int_{t_1}^{t_2} dt' \tilde{\kappa}(t - t') V_i^*(t') s_j^*(t').$$

In case of the Urbanczik-Senn rule

$$\tilde{\kappa}(t) = -e^{-\frac{t}{\tau\kappa}}$$

which implies the identities

$$I_1(t_1, t_2 + \Delta t) = I_1(t_1, t_2) + I_1(t_2, t_2 + \Delta t), \quad I_2(t_1, t_2 + \Delta t) = e^{-\frac{t_2 - t_1}{\tau\kappa}} I_2(t_1, t_2) + I_2(t_2, t_2 + \Delta t),$$

which we use to write the weight change from t to T as

$$\begin{aligned}\Delta W_{ij}(t, T) &= \Delta W_{ij}(0, T) - \Delta W_{ij}(0, t) \\ &= \eta [-I_2(0, T) + I_1(0, T) + I_2(0, t) - I_1(0, t)] \\ &= \eta \left[I_1(t, T) - I_2(t, T) + I_2(0, t) \left(1 - e^{-\frac{T-t}{\tau_R}} \right) \right].\end{aligned}$$

This is the result 17.

5.2 Voltage clamping of the adaptive exponential integrate-and-fire model

For the Clopath rule the change of the synaptic weight strongly depends on the excursion of the membrane potential V_m around a spike of the postsynaptic neuron which causes \bar{u}_\pm to cross the respective thresholds θ_\pm so that (12) and (13) yield nonvanishing results. Within the original neuron model (Brette and Gerstner, 2005) u is reset immediately after it reached the spiking threshold so that the shape of the action potential is not described accurately. In our NEST implementation of `aEIF_psc_delta_clopath` we adapted the approach of the reference implementation on ModelDB (Hines et al., 2004) and introduced a clamping of u to a fixed value V_{clamp} for a period of t_{clamp} before it is reset. The reference implementation is restricted to a simulation resolution of exactly 1 ms and sets u to two different values for the two subsequent simulation steps after a spike. To be independent of the resolution of the simulation, the implementation in NEST uses a constant V_{clamp} . In the simulations we set t_{clamp} to 2 ms and V_{clamp} to 33 mV. These values are chosen to match the behavior of the reference implementation.

5.3 History management

There are three points that need to be considered in the context of history management: First, which information needs to be stored. Second, how to search and read out the history. Third, how to identify and remove information that is no longer needed. The first and third point mainly affect memory usage, while the second point impacts the simulation time as searching within shorter histories is faster.

There are four different histories to which our considerations apply. The one to store the membrane potential V_i^* , the compressed history $\Delta W_i(t_{LS}, T)$ used only for the compressed event-driven scheme, the history to store the spike times s_i of the postsynaptic neuron (also used for ordinary STDP), and finally one might need a history that stores the last spike time for every incoming synapse (see below for details).

5.3.0.1 Adding information to the history:

This paragraph concerns only the history that stores the time trace of V_i^* . In every time step of the simulation, neurons call the protected function `write_history()` of the archiving node and pass the current value of the (low-pass filtered) membrane potential. The archiving node then computes the derived quantities V_i^* or combinations of V_i^* and s_i^* , and saves them in the history, which is of type `vector`. It is more efficient to do the computations inside the archiving node and not in the synapse for two reasons: Firstly, the computation is done only once and then used for all incoming synapses. This way no direct exchange of information between different synapses is required. Secondly, the archiving node does not need to store the raw membrane potentials before readout, but can directly store the derived quantities V_i^* , which reduces the memory consumption, especially in cases where only a non-continuous history is needed.

5.3.0.2 Readout of information from the history:

Let's assume t_{LS} and t_S be the times of the last and the current spike of a synapse. At time t_S that synapse then needs to request a part from $t_1 = t_{LS} - d$ to $t_2 = t_S - d > t_1$ of the history that ranges from $t_{\text{start}} < t_1$ to $t_{\text{end}} > t_2$. This part is shifted with respect to the spike times by a delay d which models the time of

signal propagation from the postsynaptic soma back to the synapse. The software framework NEST of our reference implementation uses only one variable to represent the delay from synapse to soma and the delay in the opposite direction. Consequently, when a spike arrives at the synapse of the postsynaptic neuron, the synapse sees a membrane potential from the past. In case every time step of the simulation adds a new entry to the history, one can easily compute the positions of the entries corresponding to $t_{1/2}$ by just knowing t_{start} and t_{end} . As pointed out in 3.2 this is the case for the Urbanczik-Senn plasticity rule. If the history is not continuous in time, like in case of the Clopath rule, this scheme is not applicable. Instead, we add a time stamp s as an additional variable to each entry and search for those with the smallest/greatest s within the interval (t_1, t_2) using e.g. a linear or a binary search. Searching for the positions that define the start and the end of the requested interval is slower than computing them directly. Nevertheless, a non-continuous history can lead to a large acceleration of simulations as we discussed in case of the Clopath rule (Section 3.4.1). Here, only values of the membrane potential in the vicinity of a spike of the postsynaptic neuron are needed so that neglecting the majority of values in between leads to a non-continuous history but saves memory.

Technically, the archiving node contains a function called `get_history()` which expects two iterators `start` and `finish` and two times t_1 and t_2 . When executed, the function sets the iterators to point to the correct entries of the history of the postsynaptic neuron corresponding to t_1 and t_2 , respectively. Having received the correct position of the pointers, the synapse evaluates the integral (6). In the event-driven compression scheme, the integration (11) is not done inside the synapse but inside the `archiving_node`. The reason for this is that the compressed history $\Delta W_i(t_{LS}, t_S)$, which is updated in case of an incoming spike, is stored inside the `archiving_node`. This way no exchange of information is needed. The synapse only triggers the updating process by calling the function `compress_history()` of the `archiving_node`. Internally, the `archiving_node` can use `get_history()` to obtain the part of the history that has to be integrated. Even though the linear search a priori might seem less efficient than a binary search or direct computation of the position, it turns out that it has an advantage in that it iterates consecutively over the history entries which can be employed to identify data no longer needed. Therefore, especially for short histories a simple iteration that comes without any overhead is fastest (see Section 3.4.1).

5.3.0.3 Removing information from the history:

To prevent the history from occupying an unnecessary amount of memory, it is crucial to have a mechanism to delete those entries that have been used by all incoming synapses. The simplest implementation to identify these entries is to add one additional variable to each entry called *access counter* initialized to zero when the entry is created. When a synapse requests a part from t_1 to t_2 of the history, the algorithm iterates over all entries $t_1 < t < t_2$ and increases the access counters by one. After the update of the synaptic weight all entries whose access counters are equal to the number of incoming synapses are deleted. This scheme can be combined easily with a linear search starting the iteration from the oldest entry of the history.

For long histories a linear search is inefficient and should be replaced by a binary search or direct computation of positions if applicable. To avoid iteration within long histories, we replace access counters by a vector that stores the last spike time t_{LS} for every incoming synapse. If a synapse delivers a spike, it updates its entry in that vector by replacing t_{LS} by the time stamp of the current spike. After each weight update, searching the vector for the smallest t_{LS} allows us to safely remove all membrane potentials with time stamps $t < \min(\{t_{LS,i}\})$. In practice, we can further improve this mechanism with two technical details. Firstly, n incoming spikes with the same time stamp can share the same entry t_{LS} which we then have to provide with a counter that goes down from n to zero in steps of one whenever one of the n

synapses sends a new spike for a time $t > t_{LS}$. Secondly, we can avoid the search for the smallest t_{LS} by making sure that the entries t_{LS} are in chronological order. This can be easily achieved if a synapse does not update its entry in the vector but removes it and appends a new one at the end of the vector.

5.4 Implementation of experiments using Clopath rule

5.4.1 Spike pairing experiment

The setup of the spike pairing experiment from [Clopath et al. \(2010\)](#) presented in [Figure 7A,B](#) consists of two neurons connected via a plastic synapse. The pre- and postsynaptic neuron are forced to spike with a time delay of Δt by stimulation with `spike_generators` at times $t_{\text{pre}}^{(i)} = t^{(i)}$ and $t_{\text{post}}^{(i)} = t^{(i)} + \Delta t$, respectively. A positive time shift $\Delta t > 0$ refers to the presynaptic neuron spike before the postsynaptic one (pre-post pairing, solid lines in [Figure 7](#)) and vice versa. Spike pairs $(t_{\text{pre}}^{(i)}, t_{\text{post}}^{(i)})$ are induced with frequency $f_{\text{pair}} = \frac{1}{t^{(i+1)} - t^{(i)}}$ and the weight change of the synapse is measured after a set of five pairs. In our simulation using NEST the presynaptic neuron is modeled as a `parrot_neuron` and the postsynaptic neuron is either of type `aeif_psc_delta_clopath` or `hh_psc_alpha_clopath`. In NEST `parrot_neurons` are model neurons that emit a spike whenever they receive one. In this setup they are required because devices like `spike_generators` support only static synapses in NEST so that we cannot connect the postsynaptic neuron directly to the `spike_generator` via a plastic synapse. The initial weight of the `clopath_synapse` connecting the two neurons is given by w_{init} . In this experiment we use the Clopath rule with fixed amplitude A_{LTD} . A list with all the parameters can be found in Table S1 in supplement.

5.4.2 Emergence of strong bidirectional couplings

In this experiment after [Clopath et al. \(2010\)](#), a small network of N_I inhibitory and N_E excitatory neurons subject to an external input develops strong bidirectional couplings between neurons of the excitatory population. The input is given by N_p Poisson spike trains with rates

$$f_p^{(j)} = A_p e^{-\frac{(j-\mu_p)^2}{2\sigma_p^2}} + c_p, \quad (18)$$

where $j = 1, \dots, N_p$. The center μ_p of the Gaussian is drawn randomly from a set s_p of possible values and a new value is drawn after each time interval t_μ . The total number of intervals is N_μ . In our simulation with NEST we used `aeif_psc_delta` model neurons with the same parameters (cf. Table S3 in supplement) for both the inhibitory and the excitatory population. The simulation is divided into N_μ intervals between which the rates of the N_p `poisson_generators` are set according to (18). The `poisson_generators` are connected in a one-to-one manner to N_p `parrot_neurons` which in turn are connected to the network. The details of the latter connectivity can be found in Table S2 in supplement. In NEST a `poisson_generator` that is connected to several target model neurons generates an independent Poisson spike train for each of these neurons. Thus, the intermediate step via `parrot_neurons` is required to provide neurons in the network with common Poisson inputs. Moreover, a direct connection from a device like a `poisson_generator` to a model neuron via a plastic synapse is not possible in NEST. In this experiment, the Clopath rule with homeostasis (time dependent prefactor for LTD, cf. [Section 5.5](#)) is used. [Figure 7 C](#) shows the weights of the synapses among the excitatory population after the simulation.

5.5 Implementation of homeostasis $A_{\text{LTD}}(\bar{u})$

For the network simulations presented in Clopath et al. (2010), the authors use a slightly modified version of the Clopath rule defined in (12): The constant factor A_{LTD} is replaced by a voltage dependent term

$$A_{\text{LTD}}(\bar{u}) = A_{\text{LTD}} \left(\frac{\bar{u}}{u_{\text{ref}}} \right)^2$$

to take into account homeostatic processes. The quantity \bar{u} is a temporal average of the quantity $\bar{u}_-(t)$ over a time window of $T = 1$ s and u_{ref} is a reference value. An exact temporal average requires storing the time trace of $\bar{u}_-(t)$ for the entire interval T . This would cancel the advantage of keeping only a sparse history as discussed in 3.2.3.2 where storage of time traces is needed only in the vicinity of spikes. Therefore, deviating from the original work by Clopath et al. (2010), we implement an additional low-pass filtering $\bar{u}(t) = (\kappa_{\text{low}} * \bar{u}_-)(t)$ with an exponential kernel $\kappa_{\text{low}}(t) = H(t) \exp(-t/\bar{\tau})$ instead. Like \bar{u}_{\pm} , \bar{u} is passed as an additional state variable to the solver.

5.6 Implementation of experiment using Urbanczik-Senn rule

In the simulation experiment shown in Figure 8 the dendrite of a conductance-based two-compartment model neuron receives a spike pattern of duration T as an input via plastic synapses. The pattern consists of N_p independent Poisson spike trains with a firing rate f_p . For learning, the pattern is repeated N_{rep} times. Dendritic synapses adapt their weights so that after learning the somatic membrane potential U and the dendritic prediction V_w^* follow a matching potential U_M . The latter is created by somatic input via two `spike_generators` that are connected via a static excitatory or inhibitory connection, respectively. Both spike generators send spikes in every simulation step. Inhibitory input spikes have a constant weight to generate a constant somatic inhibitory conductance g_I . Excitatory spikes have a modulated weight to generate a periodic excitatory conductance g_E . The input to the dendritic compartment is provided by N_p `spike_generators` each of which is connected to one `parrot_neuron` which in turn is connected to the dendrite via a plastic `urbanczik_synapse`. The intermediate `parrot_neurons` are required since in NEST the `spike_generators` can have only static synapses as outgoing connections. The spike times of the `spike_generators` are set to repeatedly generate the spike pattern created before the start of the actual simulation. The neuron's state variables are read out by a `multimeter` and the synaptic weights by a `weight_recorder`.

5.7 Experiment with modified version of the Urbanczik-Senn rule

The weight change of the Urbanczik-Senn rule as presented in Section 2.5 in line with the original publication is driven by the prediction error

$$V_i^* = (s_i - \phi(V_i)) h(V_i),$$

where s_i is the somatic spike train and V_i the dendritic prediction of the somatic membrane potential U_i . Instead of integrating over the difference between the spike train and the rate $\phi(V_i)$ (spike-rate), one can derive two variants

$$\begin{aligned} V_i^* &= \left(s_i - s_i^{\text{dend}} \right) h(V_i) \quad (\text{spike} - \text{spike}) \quad \text{and} \\ V_i^* &= \left(\phi(U_i) - \phi(V_i) \right) h(V_i) \quad (\text{rate} - \text{rate}). \end{aligned}$$

In the first one (spike-spike) we replaced the dendritic rate prediction by a noisy realization s_i^{dend} using an inhomogeneous Poisson process with rate $\phi(V_i)$. In the second one (rate-rate) the somatic spike train is

replaced by the rate of the underlying Poisson process which is computed by applying the rate function ϕ to the somatic potential U_i . The learning of a matching potential U_M as described in Section 3.3 also works in these two cases. Figure 13 shows the learning curve for all three variants of the Urbanczik-Senn rule. The loss is defined as the average mismatch between U_i and U_M averaged over one period T_p of the input pattern

$$\frac{1}{T_p} \int dt (U(t) - U_M(t))^2.$$

The decrease of the loss as a function of the pattern repetitions has a similar shape for all three variants with a significantly higher variance in case of the spike-spike version.

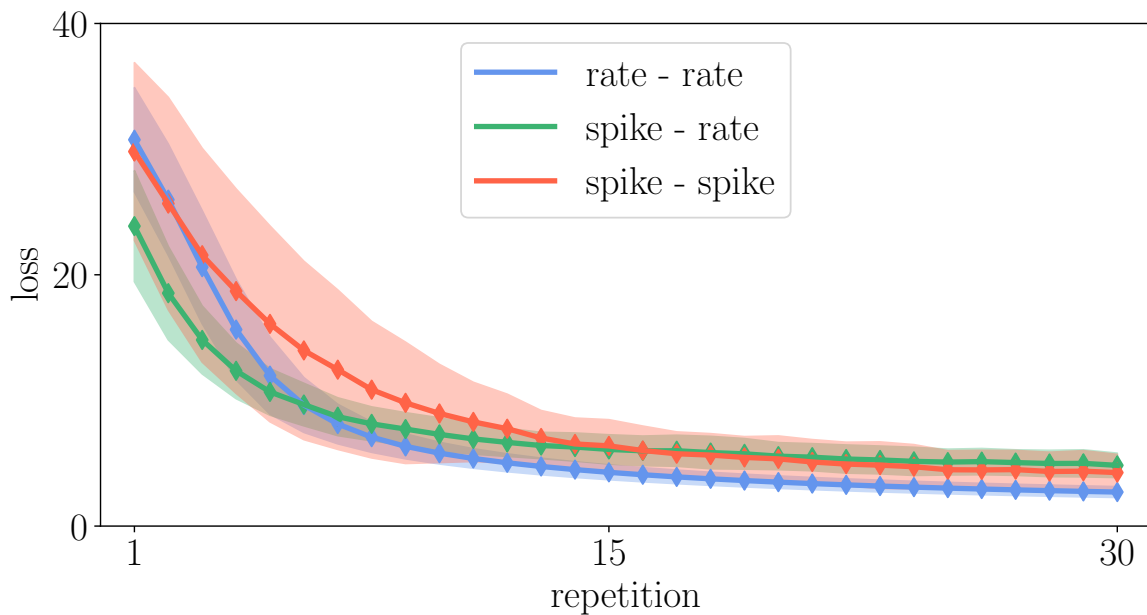


Figure 13. Comparison of learning curves in the experiment described in Section 5.6 for different variants of the Urbanczik-Senn plasticity rule. The loss is averaged over 128 trials of different input patterns. Solid curves denote the mean value and the shaded area the corresponding standard deviation of the loss.

REFERENCES

- Albers, C., Westkott, M., and Pawelzik, K. (2016), Learning of precise spike times with homeostatic membrane potential dependent synaptic plasticity, *PLOS ONE*, 11, 2, 1–28, doi:10.1371/journal.pone.0148948
- Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., et al. (2019), A solution to the learning dilemma for recurrent networks of spiking neurons, *bioRxiv*, 738385
- Bi, G. and Poo, M. (1998), Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type, *J. Neurosci.*, 18, 10464–10472
- Billeh, Y. N., Cai, B., Gratiy, S. L., Dai, K., Iyer, R., Gouwens, N. W., et al. (2020), Systematic integration of structural and functional data into multi-scale models of mouse primary visual cortex, *Neuron*
- Bono, J. and Clopath, C. (2017), Modeling somatic and dendritic spike mediated plasticity at the single neuron and network level, *Nat. Commun.*, 8, 1, 1–17

- Brader, J. M., Senn, W., and Fusi, S. (2007), Learning real world stimuli in a neural network with spike-driven synaptic dynamics, *Neural Comput.*, 19, 11, 2881–2912
- Brea, J., Gaal, A. T., Urbanczik, R., and Senn, W. (2016), Prospective coding by spiking neurons, *PLOS Comput. Biol.*, 12, 6, 1–25, doi:10.1371/journal.pcbi.1005003
- Brea, J., Senn, W., and Pfister, J.-P. (2013), Matching recall and storage in sequence learning with spiking neural networks, *J. Neurosci.*, 33, 23, 9565–9575
- Brette, R. and Gerstner, W. (2005), Adaptive exponential integrate-and-fire model as an effective description of neuronal activity, *J. Neurophysiol.*, 94, 5, 3637–3642
- Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., et al. (2007), Simulation of networks of spiking neurons: A review of tools and strategies, *J. Comput. Neurosci.*, 23, 3, 349–398, doi:10.1007/s10827-007-0038-6
- Brunel, N. (2000), Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons, *Journal of Computational Neuroscience*, 8, 3, 183–208, doi:10.1023/a:1008925309027
- Carnevale, N. T. and Hines, M. L. (2006), *The NEURON Book* (Cambridge University Press, Cambridge)
- Cartiglia, M., Haessig, G., and Indiveri, G. (2020), An error-propagation spiking neural network compatible with neuromorphic processors, in 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS) (IEEE), 84–88
- Clopath, C., Büsing, L., Vasilaki, E., and Gerstner, W. (2010), Connectivity reflects coding: a model of voltage-based STDP with homeostasis, *Nat. Neurosci.*, 13, 344–352
- Clopath, C. and Gerstner, W. (2010), Voltage and spike timing interact in stdp—a unified model, *Frontiers in synaptic neuroscience*, 2, 25
- Davison, A., Brüderle, D., Eppler, J., Kremkow, J., Müller, E., Pecevski, D., et al. (2008), PyNN: a common interface for neuronal network simulators, *Front. Neuroinformatics*, 2, 11, doi:10.3389/neuro.11.011.2008
- D’Haene, M., Hermans, M., and Schrauwen, B. (2014), Toward unified hybrid simulation techniques for spiking neural networks, *Neural computation*, 26, 6, 1055–1079
- Diederich, N., Bartsch, T., Kohlstedt, H., and Ziegler, M. (2018), A memristive plasticity model of voltage-based stdp suitable for recurrent bidirectional neural networks in the hippocampus, *Scientific reports*, 8, 1, 1–12
- Djurfeldt, M., Davison, A. P., and Eppler, J. M. (2014), Efficient generation of connectivity in neuronal networks from simulator-independent descriptions, *Front. Neuroinformatics*, 8, 43, doi:10.3389/fninf.2014.00043
- Djurfeldt, M., Hjorth, J., Eppler, J. M., Dudani, N., Helias, M., Potjans, T. C., et al. (2010), Run-time interoperability between neuronal network simulators based on the MUSIC framework, *Neuroinformatics*, 8, 43–60
- Eppler, J. M., Helias, M., Müller, E., Diesmann, M., and Gewaltig, M. (2009), PyNEST: a convenient interface to the NEST simulator, *Front. Neuroinformatics*, 2, 12, doi:10.3389/neuro.11.012.2008
- Friedmann, S., Schemmel, J., Grübl, A., Hartel, A., Hock, M., and Meier, K. (2016), Demonstrating hybrid learning in a flexible neuromorphic hardware system, *IEEE transactions on biomedical circuits and systems*, 11, 1, 128–142
- Galluppi, F., Lagorce, X., Stamatias, E., Pfeiffer, M., Plana, L. A., Furber, S. B., et al. (2015), A framework for plasticity implementation on the spinnaker neural architecture, *Frontiers in Neuroscience*, 8, 429
- Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1996), A neuronal learning rule for sub-millisecond temporal coding, *Nature*, 383, 76–78

- Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014), *Neuronal Dynamics. From single Neurons to Networks and Models of Cognition* (Cambridge University Press, Cambridge)
- Gewaltig, M.-O. and Diesmann, M. (2007), NEST (NEural Simulation Tool), *Scholarpedia*, 2, 4, 1430, doi:10.4249/scholarpedia.1430
- Goodman, D. and Brette, R. (2013), Brian simulator, *Scholarpedia*, 8, 1, 10883
- Hahne, J., Helias, M., Kunkel, S., Igarashi, J., Bolten, M., Frommer, A., et al. (2015), A unified framework for spiking and gap-junction interactions in distributed neuronal network simulations., *Front. Neuroinformatics*, 9, 22, doi:10.3389/fninf.2015.00022
- Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., et al. (2014), Deep speech: Scaling up end-to-end speech recognition, *arXiv preprint arXiv:1412.5567*
- Hebb, D. O. (1949), *The organization of behavior: A neuropsychological theory* (John Wiley & Sons, New York)
- Hines, M. L. and Carnevale, N. T. (2001), NEURON: a tool for neuroscientists, *Neuroscientist*, 7, 2, 123–135, doi:10.1177/107385840100700207
- Hines, M. L., Morse, T., Migliore, M., Carnevale, N. T., and Shepherd, G. M. (2004), ModelDB: A database to support computational neuroscience, *J. Comput. Neurosci.*, 17, 1, 7–11, doi:10.1023/B:JCNS.0000023869.22017.2e
- Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006), A fast learning algorithm for deep belief nets, *Neural computation*, 18, 7, 1527–1554
- Jordan, J., Helias, M., Diesmann, M., and Kunkel, S. (2020a), Efficient communication in distributed simulations of spiking neuronal networks with gap junctions, *Front. Neuroinformatics*, 14, 12, doi:10.3389/fninf.2020.00012
- Jordan, J., Ippen, T., Helias, M., Kitayama, I., Sato, M., Igarashi, J., et al. (2018), Extremely scalable spiking neuronal network simulation code: From laptops to exascale computers, *Front. Neuroinformatics*, 12, 2, doi:10.3389/fninf.2018.00002
- Jordan, J., Mørk, H., Vennemo, S. B., Terhorst, D., Peyser, A., Ippen, T., et al. (2019), Nest 2.18.0, doi:10.5281/zenodo.2605422
- Jordan, J., Schmidt, M., Senn, W., and Petrovici, M. A. (2020b), Evolving to learn: discovering interpretable plasticity rules for spiking networks, *arXiv preprint arXiv:2005.14149*
- Jülich Supercomputing Centre (2015), JUQUEEN: IBM Blue Gene/Q[®] supercomputer system at the Jülich Supercomputing Centre
- Ko, H., Cossell, L., Baragli, C., Antolik, J., Clopath, C., Hofer, S. B., et al. (2013), The emergence of functional microcircuits in visual cortex, *Nature*, 496, 7443, 96–100
- Krishnan, J., Porta Mana, P., Helias, M., Diesmann, M., and Di Napoli, E. A. (2017), Perfect detection of spikes in the linear sub-threshold dynamics of point neurons, *Front. Neuroinformatics*, 11, 75, doi:10.3389/fninf.2017.00075
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, in *Advances in neural information processing systems*, 1097–1105
- Kumar, N. M. and Gilula, N. B. (1996), The gap junction communication channel, *Cell*, 84, 3, 381–388, doi:10.1016/S0092-8674(00)81282-9
- Kunkel, S. (2015), Simulation technology for plastic neuronal networks on high-performance computers, 10.6094/UNIFR/10419, 1–188, doi:10.6094/UNIFR/10419
- Kunkel, S., Diesmann, M., and Morrison, A. (2011), Limits to the development of feed-forward structures in large recurrent neuronal networks, *Front. Comput. Neurosci.*, 4

- Lecun, Y. (1985), Une procedure d'apprentissage pour reseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks), in Proceedings of Cognitiva 85, Paris, France, 599–604
- LeCun, Y., Bengio, Y., and Hinton, G. (2015), Deep learning, *Nature*, 521, 7553, 436–444
- Legenstein, R. and Maass, W. (2011), Branch-specific plasticity enables self-organization of nonlinear computation in single neurons, *J. Neurosci.*, 31, 30, 10787–10802, doi:10.1523/JNEUROSCI.5684-10.2011
- Litwin-Kumar, A. and Doiron, B. (2014), Formation and maintenance of neuronal assemblies through synaptic plasticity, *Nat. Commun.*, 5, 1, 1–12
- Lytton, W. W., Seidenstein, A. H., Dura-Bernal, S., McDougal, R. A., Schürmann, F., and Hines, M. L. (2016), Simulation neurotechnologies for advancing brain research: parallelizing large networks in NEURON, *Neural computation*, 28, 10, 2063–2090, doi:10.1162/neco_a_00876
- Maes, A., Barahona, M., and Clopath, C. (2020), Learning spatiotemporal signals using a recurrent spiking network that discretizes time, *PLoS computational biology*, 16, 1, e1007606
- Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997), Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs, *Science*, 275, 213–215
- Markram, H., Müller, E., Ramaswamy, S., Reimann, M. W., Abdellah, M., Sanchez, C. A., et al. (2015), Reconstruction and simulation of neocortical microcircuitry, *Cell*, 163, 2, 456–492, doi:10.1016/j.cell.2015.09.029
- Mayr, C. G. and Partzsch, J. (2010), Rate and pulse based plasticity governed by local synaptic state variables, *Frontiers in Synaptic Neuroscience*, 2, 33
- Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2018), A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs), *IEEE Transactions on Biomedical Circuits and Systems*, 12, 1, 106–122, doi:10.1109/tbcas.2017.2759700
- Morrison, A., Aertsen, A., and Diesmann, M. (2007a), Spike-timing dependent plasticity in balanced random networks, *Neural Comput.*, 19, 1437–1467, doi:10.1162/neco.2007.19.6.1437
- Morrison, A. and Diesmann, M. (2008), Maintaining causality in discrete time neuronal network simulations, in P. b. Graben, C. Zhou, M. Thiel, and J. Kurths, eds., *Lectures in Supercomputational Neurosciences: Dynamics in Complex Brain Networks* (Springer, Berlin, Heidelberg), 267–278, doi:10.1007/978-3-540-73159-7_10
- Morrison, A., Diesmann, M., and Gerstner, W. (2008), Phenomenological models of synaptic plasticity based on spike-timing, *Biol. Cybern.*, 98, 6, 459–478, doi:10.1007/s00422-008-0233-1
- Morrison, A., Mehring, C., Geisel, T., Aertsen, A., and Diesmann, M. (2005), Advancing the boundaries of high connectivity network simulation with distributed computing, *Neural Comput.*, 17, 8, 1776–1801, doi:10.1162/0899766054026648
- Morrison, A., Straube, S., Plesser, H. E., and Diesmann, M. (2007b), Exact subthreshold integration with continuous spike times in discrete-time neural network simulations, *Neural Comput.*, 19, 1, 47–79, doi:10.1162/neco.2007.19.1.47
- Müller, E., Bednar, J. A., Diesmann, M., Gewaltig, M.-O., Hines, M., and Davison, A. P. (2015), Python in neuroscience, *Front. Neuroinformatics*, 9, 11, doi:10.3389/fninf.2015.00011
- Neftci, E., Das, S., Pedroni, B., Kreuz-Delgado, K., and Cauwenberghs, G. (2014), Event-driven contrastive divergence for spiking neuromorphic systems, *Frontiers in Neuroscience*, 7, 272
- Neftci, E. O., Augustine, C., Paul, S., and Detorakis, G. (2017), Event-driven random back-propagation: Enabling neuromorphic deep learning machines, *Frontiers in Neuroscience*, 11, 324
- Nordlie, E., Gewaltig, M.-O., and Plesser, H. E. (2009), Towards reproducible descriptions of neuronal network models, *PLoS Comput. Biol.*, 5, 8, e1000456, doi:10.1371/journal.pcbi.1000456

- Parker, D. (1985), Learning logic, *Technical Report TR-47*
- Pecevski, D., Kappel, D., and Jonke, Z. (2014), Nevesim: event-driven neural simulation framework with a python interface, *Frontiers in Neuroinformatics*, 8, 70
- Pfeil, T., Grübl, A., Jeltsch, S., Müller, E., Müller, P., Petrovici, M. A., et al. (2013), Six networks on a universal neuromorphic computing substrate, *Front. Neurosci.*, 7, 11
- Plesser, H., Diesmann, M., Gewaltig, M.-O., and Morrison, A. (2015), Nest: the neural simulation tool, in D. Jaeger and R. Jung, eds., *Encyclopedia of Computational Neuroscience* (Springer New York), 1849–1852, doi:10.1007/978-1-4614-6675-8_258
- Plotnikov, D., Blundell, I., Ippen, T., Eppler, J. M., Morrison, A., and Rumpe, B. (2016), NESTML: a modeling language for spiking neurons, in *Modellierung 2016 Conference*, volume 254 of *LNI* (Bonner Köllen Verlag, Bonn), volume 254 of *LNI*, 93–108
- Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al. (2015), A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses, *Front. Neurosci.*, 9, 141, doi:10.3389/fnins.2015.00141
- Ros, E., Carrillo, R., Ortigosa, E. M., Barbour, B., and Agís, R. (2006), Event-driven simulation scheme for spiking neural networks using lookup tables to characterize neuronal dynamics, *Neural computation*, 18, 12, 2959–2993
- Rudolph, M. and Destexhe, A. (2006), Event-based simulation strategy for conductance-based synaptic interactions and plasticity, *Neurocomputing*, 69, 10-12, 1130–1133
- Rumelhart, E., David, Hinton, E., Geoffrey, and Williams, J., Ronald (1986), Learning representations by back-propagating errors, *Nature*, 323, 533–536, doi:10.1038/323533a0
- Sacramento, J., Costa, R. P., Bengio, Y., and Senn, W. (2018), Dendritic cortical microcircuits approximate the backpropagation algorithm, in *Advances in Neural Information Processing Systems*, 8721–8732
- Sadeh, S., Clopath, C., and Rotter, S. (2015), Emergence of functional specificity in balanced networks with synaptic plasticity, *PLoS Comput. Biol.*, 11, 6, 1–27, doi:10.1371/journal.pcbi.1004307
- Schmidt, M., Bakker, R., Shen, K., Bezgin, G., Diesmann, M., and van Albada, S. J. (2018), A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas, *PLoS Comput. Biol.*, 14, 10, e1006359, doi:10.1371/journal.pcbi.1006359
- Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., and Linares-Barranco, B. (2013), Stpd and stdp variations with memristors for spiking neuromorphic learning systems, *Frontiers in Neuroscience*, 7, 2
- Sheik, S., Paul, S., Augustine, C., and Cauwenberghs, G. (2016), Membrane-dependent neuromorphic learning rule for unsupervised spike pattern detection, in *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (IEEE), 164–167
- Sjöström, P., Turrigiano, G., and Nelson, S. (2001), Rate, timing, and cooperativity jointly determine cortical synaptic plasticity, *Neuron*, 32, 1149–1164
- Song, S., Sjöström, P., Reigl, M., Nelson, S., and Chklovskii, D. (2005), Highly nonrandom features of synaptic connectivity in local cortical circuits, *PLoS Biol.*, 3, 3, e8
- Stimberg, M., Goodman, D., Benichoux, V., and Brette, R. (2014), Equation-oriented specification of neural models for simulations, *Frontiers in Neuroinformatics*, 8, 6, doi:10.3389/fninf.2014.00006
- Thakur, C. S., Molin, J. L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., et al. (2018), Large-scale neuromorphic spiking array processors: A quest to mimic the brain, *Frontiers in Neuroscience*, 12, 891
- Urbanczik, R. and Senn, W. (2014), Learning by the dendritic prediction of somatic spiking, *Neuron*, 81, 3, 521–528

- Watts, L. (1994), Event-driven simulation of networks of spiking neurons, *Advances in neural information processing systems*, 927–927
- Werbos, P. (1974), *Beyond regression : new tools for prediction and analysis in the behavioral sciences* (Harvard University)
- Yger, P. and Harris, K. D. (2013), The convallis rule for unsupervised learning in cortical networks, *PLOS Comput. Biol.*, 9, 10, 1–16, doi:10.1371/journal.pcbi.1003272
- Zenke, F. and Gerstner, W. (2014), Limits to high-speed simulations of spiking neural networks using general-purpose computers, *Frontiers in Neuroinformatics*, 8, 76

6 SUPPLEMENTAL MATERIAL

A: Simulation parameters		
Symbol	Value	Description
f_{pair}	[10, 11, ..., 50] Hz	frequency of occurrence of spike pairs
Δt	± 10 ms	time shift of spike pair
w_{init} [mV]	0.5 mV or pA/ms	initial weight (unit is mV for aeif and pA/ms for hh neuron)
B: Parameters of aeif_psc_delta_clopath		
Symbol	Value	Description
E_L	-54.402 mV	leak reversal potential
E_{Na}	50.0 mV	sodium reversal potential
E_K	-77.0 mV	potassium reversal potential
g_L	30.0 nS	leak conductance
g_{Na}	$12 \cdot 10^3$ nS	sodium peak conductance
g_K	$3.6 \cdot 10^3$ nS	potassium peak conductance
C_m	100 pF	membrane capacitance
τ_{ex}	0.2 ms	rise time of the exc. synaptic alpha funct.
τ_{in}	2.0 ms	rise time of the inh. synaptic alpha funct.
θ_-	-64.9 mV	threshold
θ_+	-35 mV	threshold
A_{LTD}	$1 \cdot 10^{-4}$ 1/mV	amplitude of LTD
A_{LTP}	$12 \cdot 10^{-4}$ 1/mV ²	amplitude of LTP
τ_-	10 ms	time constant of \bar{u}_-
τ_+	114 ms	time constant of \bar{u}_+
τ_s	15 ms	time constant of s_j^*
d_s	5 ms	delay of \bar{u}_\pm
C: Parameters of hh_psc_alpha_clopath		
Symbol	Value	Description
E_L	-70.6 mV	leak reversal potential
g_L	30 nS	leak conductance
C_m	281 pF	membrane capacitance
V_{reset}	-60 mV	reset value of membr. pot. after spike
V_{peak}	33 mV	spike detection threshold
ΔT	2 mV	slope factor
τ_w	144 ms	spike adaptation time constant
τ_z	40 ms	spike adaptation time constant
$V_{\text{th,max}}$	30.4 mV	threshold potential after spike
$\tau_{V,\text{th}}$	50 ms	threshold potential time constant
a	4 nS	subthreshold adaptation
b	0.0805 pA	spike triggered adaptation
θ_-	-70.6 mV	threshold of \bar{u}_-
θ_+	-45.3 mV	threshold of \bar{u}_+
A_{LTD}	$14 \cdot 10^{-5}$ 1/mV	amplitude of LTD
A_{LTP}	$8 \cdot 10^{-5}$ 1/mV ²	amplitude of LTP
τ_-	10 ms	time constant of \bar{u}_-
τ_+	7 ms	time constant of \bar{u}_+
τ_s	15 ms	time constant of s_j^*
d_s	4 ms	delay of \bar{u}_\pm

Table S1. Parameters of the spike pairing experiment using the Clopath rule. The values for the aeif model are taken from (Clopath et al., 2010, Tab. 1 and appendix) and those for the hh model are extracted from the reference implementation by B. Torben-Nielsen on ModelDB (Hines et al., 2004).

A: Model summary			
Populations	Three: excitatory, inhibitory, external input		
Connectivity	all-to-all, fixed out-degree, fixed in-degree		
Neuron model	adaptive exponential integrate-and-fire (aeif, Clopath)		
Plasticity	Clopath synapse		
Input	independent homogeneous Poisson spike trains		
Measurements	synapse weight		
B: Populations			
Name	Elements	Population size	
E	aeif/two-comp.	$N_E = 10$	
I	aeif/two-comp.	$N_I = 3$	
E _{ext}	Poisson generator	$N_p = 500$	
C: Connectivity			
Name	Source	Target	Pattern
ExcExc	E	E	all-to-all (no autapses)
ExcInh	E	I	fixed in-degree $C_E = 8$
InhExc	I	E	fixed out-degree $C_I = 6$
ExtExc	E _{Ext}	E	all-to-all
ExtInh	E _{Ext}	I	all-to-all
D: Neurons			
Name	aeif_psc_delta_clopath		
Type	adaptive exponential integrate-and-fire		
Details	see Clopath et al. (2010)		
Parameters	see Table S3		
E: Synapses			
Name	Model	Initial weight [mV]	Max. weight [mV]
ExcExc	clopath_synapse	0.25	0.75
ExcInh	static_synapse	1.0	—
InhExc	static_synapse	1.0	—
ExtExc	clopath_synapse	random uniform from $[0.5, 1.5]$	3.0
ExtInh	static_synapse	random uniform from $[0.0, 0.5]$	—
F: Input			
Type	Poisson generator		
Description	homogeneous Poisson spike trains, independent for each neuron, modulated rate		
Parameters	see Table S3		

Table S2. Model description of a small excitatory-inhibitory network after Nordlie et al. (2009). This network reproduces the emergence of strong bidirectional couplings using the Clopath rule shown in Figure 7. The values of the parameters are shown in [Table S3](#).

A: Parameters of aeif_psc_delta_clopath		
Symbol	Value	Description
E_L	-70.6 mV	leak reversal potential
g_L	30 nS	leak conductance
C_m	281 pF	membrane capacitance
V_{reset}	-60 mV	reset value of membr. pot. after spike
V_{peak}	20 mV	spike detection threshold
Δ_T	2 mV	slope factor
τ_w	144 ms	spike adaptation time constant
τ_z	40 ms	spike adaptation time constant
$V_{\text{th,max}}$	30.4 mV	threshold potential after spike
$\tau_{V,\text{th}}$	50 ms	threshold potential time constant
a	4 nS	subthreshold adaptation
b	0.0805 pA	spike triggered adaptation
θ_-	-70.6 mV	threshold of \bar{u}_-
θ_+	-45.3 mV	threshold of \bar{u}_+
A_{LTD}	$14 \cdot 10^{-5} \text{ 1/mV}$	amplitude of LTD
u_{ref}	60	reference value for \bar{u}
$\bar{\tau}$	$1.5 \cdot 10^3 \text{ ms}$	time constant of \bar{u}
A_{LTP}	$8 \cdot 10^{-5} \text{ 1/mV}^2$	amplitude of LTP
τ_-	10 ms	time constant of \bar{u}_-
τ_+	7 ms	time constant of \bar{u}_+
τ_s	15 ms	time constant of s_j^*
d_s	4 ms	delay of \bar{u}_\pm
B: Input parameters		
Symbol	Value	Description
A_p	66 [Hz]	amplitude of Gaussian rate profile
σ_p	10	width of Gaussian rate profile
c_p	0.48 [Hz]	offset of Gaussian rate profile
s_p	[25, 75, ..., 475]	set of possible values for the center of the Gaussian μ_p
t_μ	100 [ms]	interval after which a new value for μ_p is drawn
N_μ	100	number of intervals t_μ

Table S3. Neuron and input parameters for simulation of network producing bidirectional connections using the Clopath rule. The values are taken from (Clopath et al., 2010, Tab. 1 and appendix). The same values are used for the performance measurements shown in Figures 9 and 10.

A: Simulation parameters		
Symbol	Value	Description
T_p	1000 ms	pattern duration
N_{rep}	100	number of pattern repetitions
N_p	200	number of input spike trains
f_p	10 Hz	input firing rate
w_{gE}	$(18 \sin(2\pi t) + 4.8)$ nS	weights to generate periodic excitatory conductance
w_{gI}	3 nS	weights to generate constant inhibitory conductance
B: Parameters of pp_cond_exp_mc_urbanczik (soma)		
Symbol	Value	Description
C_m	300 pF	membrane capacitance
E_L	-70 mV	leak reversal potential
g_L	30.0 nS	leak conductance
E_{ex}	0.0 mV	exc. reversal potential
E_{in}	-75.0 mV	inh. reversal potential
τ_{ex}	3.0 ms	rise time of the exc. synaptic alpha funct.
τ_{in}	3.0 ms	rise time of the inh. synaptic alpha funct.
t_{ref}	3.0 ms	refractory time
C: Parameters of pp_cond_exp_mc_urbanczik (dendrite)		
Symbol	Value	Description
C_m	300 pF	membrane capacitance
E_L	-70 mV	leak reversal potential
g_L	30.0 nS	leak conductance
τ_{ex}	3.0 ms	rise time of the exc. synaptic alpha funct.
τ_{in}	3.0 ms	rise time of the inh. synaptic alpha funct.
$\phi(U)$	$\frac{0.15 \text{ kHz}}{1 + \frac{1}{2} \exp\left(\frac{-55 \text{ mV} - U}{3 \text{ mV}}\right)}$	rate function
g_{sp}	600.0 nS	coupling dendrite to soma

Table S4. Parameters of the simulation of the learning experiment using the Urbanczik-Senn rule. The same values of the neuron parameters are used for the performance measurements shown in Figures 11 and 12.

A: Model summary				
Populations	Three: excitatory, inhibitory, external input			
Connectivity	random with fixed indegree			
Neuron model	adaptive exponential integrate-and-fire (aeif, Clopath)/ two-compartment Poisson (two-comp., Urbanczik-Senn)			
Plasticity	Clopath synapse/ Urbanczik-Senn synapse			
Input	independent homogeneous Poisson spike trains			
Measurements	—			
B: Populations				
Name	Elements	Population size		
E	aeif/two-comp.	$N_E = 4N_I$		
I	aeif/two-comp.	N_I		
E _{ext}	Poisson generator	1		
C: Connectivity				
Name	Source	Target	Pattern	Weight
Exc	E	E+I	fixed in-degree C_E	J_{ex}
Inh	I	E+I	fixed in-degree C_I	J_{in}
Ext	E _{Ext}	E+I	all-to-all	J
D: Neurons				
Name	aeif_psc_delta_clopath			
Type	adaptive exponential integrate-and-fire			
Details	see Clopath et al. (2010)			
Parameters	see Table S3			
Name	pp_cond_exp_mc_urbanczik			
Type	two-compartment neuron with spike generation via inhomogeneous Poisson process			
Details	see Urbanczik and Senn (2014)			
Parameters	see Table S4			
E: Synapses				
Name	Model			
Exc	clopath/urbanczik_synapse			
Inh	clopath/urbanczik_synapse			
Ext	static_synapse			
F: Input				
Type	Description			
Poisson generator	homogeneous Poisson spike trains, independent for each neuron, rate $f_{\text{ext}} = \nu_{\text{ext}} C_E$			

Table S5. Model description of Brunel network after Nordlie et al. (2009). This network is used to produce the performance measurement shown in Figures 9, 10, 11, and 12. The values of the parameters are shown in [Table S6](#).

A: Global simulation parameters		
Symbol	Value	Description
T	$2 \cdot 10^3$ ms	biological time
h	0.1 ms	resolution
B: Network sizes		
Symbol	Value	Description
$N = N_E + N_I$	$1.92 \cdot 10^6$	number of neurons in Clopath simulation with small indegree $K = 100$
N	$1.54 \cdot 10^5$	number of neurons in Clopath simulation with large indegree $K = 5000$
N	$3.84 \cdot 10^5$	number of neurons in Urbanczik simulation with small indegree $K = 100$
N	$3.84 \cdot 10^4$	number of neurons in Urbanczik simulation with large indegree $K = 5000$
C: Connectivity		
Symbol	Value	Description
g	7.0	ratio inh./exc. weight
J	0.1	postsynaptic amplitude The unit depends on the neuron model. In case of the aeif model and the Clopath rule it is [mV] and for the Urbanczik-Senn rule it is [pA]
J_{ex}	J	amplitude of exc. postsyn. potential
J_{in}	$-gJ_{\text{ex}}$	amplitude of inh. postsyn. potential
$K = C_E + C_I$	100 or 5000	total number of excitatory synapses per neuron
C_E	$0.8K$	number of excitatory synapses per neuron
C_I	$0.2K$	number of inhibitory synapses per neuron
η	0.0	learning rate
D: External input		
Symbol	Value	Description
ν_{ext}	$6.75 \cdot 10^{-3}$ Hz	factor in rate of external Poisson input $f_{\text{ext}} = \nu_{\text{ext}}C_E$

Table S6. Parameters of the Brunel network.