Michael G. Müller, BSc

Variable Binding through Assemblies in Spiking Neural Networks

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor:
Assoc.Prof. Dipl.-Ing. Dr.techn. Robert Legenstein
Institute of Theoretical Computer Science

Graz, August 2017

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

A thorough understanding of the processes underlying the cognitive capabilities of humans has remained elusive. Among the open issues is the binding problem, i.e. the question of how bits of information are tied together in the brain. This work tackles the problem using computer simulations which show that networks of spiking neurons can perform simple binding operations through their dynamics. The circuit is built on mechanism used in the brain which have been firmly established by experimental studies: winner-take-all dynamics within groups of neurons as well as spike-timing dependent plasticity. The model in this work is based on an existing model, which is improved with respect to the biological plausibility of the implementation. A new mode of information storage is introduced which is consistent with findings of experimental neuroscience concerning the retention of information in working memory in the human cortex. Furthermore, an optimization algorithm with very few hyperparameters is introduced which allows the rapid optimization of high-dimensional parameter spaces with constraints. This algorithm was used to tune the parameters of the variable binding model.

Zusammenfassung

Die menschliche Wahrnehmung und die ihr zugrunde liegenden Prozesse werfen weiterhin Fragen auf. Ein Beispiel hierfür ist das sogenannte Bindungsproblem, welches sich mit der Frage beschäftigt, wie im Gehirn verschiedene Informationsbausteine miteinander verknüpft werden. In dieser Arbeit nähern wir uns dem Problem mit Hilfe von Computersimulationen. Es wird gezeigt, dass Netzwerke bestehend aus spikenden Neuronen einfache Bindungsoperationen durchführen können. Dabei werden Mechanismen genutzt, welche auch im Gehirn verwendet werden und durch eine Vielzahl von Studien gut begründet sind. Unter anderem kommen Gruppen von Neuronen mit Winner-Take-All-Funktionalität sowie synaptische Plastizität zum Einsatz. Diese Arbeit baut auf einem existierenden Modell auf. Die Implementierung des Modells konnte in dieser Arbeit biologisch realistischer gestaltet werden. Darüber hinaus wurde ein neues Verfahren der Informationsspeicherung eingeführt, welches mit experimentellen Daten zum Arbeitsgedächtnis im menschlichen Gehirn übereinstimmt. Außerdem wird ein neuer Ansatz zur effizienten Optimierung von beschränkten hochdimensionalen Kostenfunktionen vorgestellt, welcher genutzt wurde, um die Parameter des Modells zu optimieren.

Acknowledgment

First, I would like to thank my supervisor Robert Legenstein for the opportunity to work on such an interesting research topic, which has been very rewarding. The challenges of beginning to work in a research environment have been greatly eased by the helpful assistance of all the friendly people at IGI. I would like to thank Michael Hoff in particular for motivation and advice.

Facing the end of my time as a student, this is a good opportunity to thank all of the people who have accompanied me on the way. First and foremost, I acknowledge that I am greatly indebted to Moritz Kampelmühler for team work and moral boost during countless strenuous courses. May there be more challenges to come.

I also need to express my deep gratitude towards my parents for their support not only during my studies.

This work would not have been possible without the love of my life, Debbie. Thank you for everything.

What would any statement of acknowledgment be without some random gem? – "Knock it off, Napoleon! Make yourself a dang quesadilla!"

Contents

1 Introduction

Underlying the cognitive capabilities of humans – which are generally taken for granted – are processes which continue to intrigue and puzzle researchers due to their abysmal complexity. One question which remains open despite having been investigated for a long time is the problem of neural coding, i.e. the question of how information is represented in the brain. From its array of sensory inputs, the brain acquires information about the outside world and stores it in some useful representation. The tasks which humans can perform demand that these representations of information allow further processing, e.g. for making decisions based upon them. Yet, the nature of the symbols which are used in the brain to store even the simplest bits of information remains elusive.

The question of the exact processes which underlie the representations used by the brain is closely related to the binding problem. Originally formulated in [1], the binding problem considers how information can be represented in a structured way. A typical example is a visual scene which is viewed by an observer. Somehow, objects in the scene are assigned by the brain with attributes describing their properties and relations to each other. The binding problem now considers the following question: how is the information encoding some object's identity (e.g. "ball") bound to information encoding its attributes (e.g. "blue")? This binding allows the observer to retrieve information in various different ways, e.g. when facing the questions "Which color does the ball have?" and "Which object is blue?".

The binding problem appears in several different forms [2]. In this work, we consider a general binding setup, which asks how some content can be bound to a placeholder (or variable). This binding is regarded as one of the most important atomic computations performed by the brain [3]. Specifically, we consider this problem in a linguistic setup: when listening to a sentence, how is the identity of some entity occuring in the sentence bound to its respective role? Consider, for instance, the sentences "The baby hit Grandpa." and "Grandpa hit the baby." which greatly differ in meaning (and possibly in importance for the listener). How can these two sentences be distinguished, given that they both use the same set of words? A simple ordering, e.g. attempting to decode the meaning by assuming that the agent always appears first in the sentence, does not suffice, since we may rephrase the two sentences to read "Grandpa was hit by the baby." and "The baby was hit by Grandpa.". Generally, language allows for vast numbers of combinations of words in ways which are syntactically correct and which make sense for a human listener (or reader). Therefore, some other mechanism is needed: one which specifically binds together identity and role.

In this work, we aim to tackle the binding problem using a number of

established and emerging concepts from experimental neuroscience. Central to the model described here is the discovery of concept cells [4] in the medial temporal lobe (MTL) of the human cortex. These sparsely firing neurons become active only when a specific concept is presented to probands through some input. For example, some neurons fire exclusively when the picture of a certain celebrity is seen, or when the name of the celebrity is heard.

Generally, neurons are assumed to form groups which display the same behavior [5]. These assemblies (or ensembles) of neurons make neural responses more robust to noise or the loss of a single neuron. Building on the notion of concept cells, we use assemblies of neurons which encode different concepts. In our model, a number of assemblies encoding different concepts together form the so-called content space. The presence of a given concept (like "baby" or "Grandpa" from the example sentences above) are encoded through the sparse activity of one of these assemblies.

It has furthermore been observed that neurons in the MTL can rapidly encode new memories, e.g. associations between pictures which are presented together [6]. We thus introduce separate groups of neurons, termed variable spaces, which also consist of assemblies of neurons. These variable spaces can serve as placeholders which store pointers to some concept represented by some assembly in the content space. For instance, one variable space could store a pointer to the agent in the sentence. Then, for the sentence "The baby hit Grandpa.", this variable space would hold a point to the assembly encoding "baby" within the content space.

It is important to point out that variable spaces do not store a copy of the data that was stored in them. Instead, they can only be read out by restoring the activity in the content space. Therefore, the representation of some piece of information can be fundamentally different in the content space and the variable spaces and need not reflect a shared encoding (e.g. firing patterns). The assembly which stores a pointer in the variable space emerges in a transient manner, which has been postulated as a principle for neuronal assemblies [7].

Evidence for this architecture has been presented by an experiment by Frankland and Greene [8]. They showed that the identity of both an agent and a patient in a sentence can be reliably detected from functional magnetic resonance imaging (fMRI) recordings from the MTL. Furthermore, the different roles were encoded in different subregions of the MTL, confirming the view that different bits of information are stored in different variable spaces.

In [9], the results of the experiments of Frankland and Greene were reproduced using the setup of one content space and one or more variable spaces. Using a rather abstract neuron model, it was shown here that assemblies forming in variable spaces allow for reliable recall of the activity in the content space after delay periods. Other operations considered were copying the contents of one variable space into another and comparing the

contents of two variable spaces.

The present work builds on this model and aims to expand it by removing some biologically unrealistic constraints. To do so, the circuitry required for the variable binding was implemented in the neural circuit simulator NEST (Neural Simulation Tool) [10, 11], which aims to model the electrophysiological properties of neurons rather closely.

A number of contributions are made in this work:

- First, the model presented in this work removes some of the biologically problematic constraints from the model introduced in [9]. There, inhibition within each neural space (i.e. both content and variable spaces) is modeled by a mechanism which reads out the membrane potential of each neuron and subsequently injects an inhibitory current into each of them based on the overall amount of activity. Here, we model the inhibition required for neural spaces through an external pool of inhibitory neurons which regulate the activity of excitatory neurons. Furthermore, in [9], the connectivity between the content space and each variable space is assumed to be symmetric, whereas symmetric connectivity rarely occurs in biological neural networks. In this work, we use random connectivity between all neural spaces.

- We introduce another possibility for retaining the information stored in variable space over time. In [9], the neurons within each neural space posses an adaptive excitability mechanism, which ensures that previously active neurons are more likely to become active again after some delay. This allows for variable spaces to be completely inhibited while retaining some pointer. In addition to reproducing the results with a similar mechanism, it is shown in this work that the information can also be stored in persistent activity of variable spaces over a delay period. This is an intriguing alternative since persistent activity is known to underlie working memory in different brain areas.

- Due to the more realistic modeling of the NEST simulator, many new parameters were introduced in the models used this work and had to be tuned. As using a standard method for high-dimensional optimization (Differential Evolution as implemented in a standard Python library) proved to produce poor results, we introduce a novel optimization technique which allows for the rapid optimization of high-dimensional error landscapes with constraints. In addition to using this optimization algorithm for finding good parameters for the variable binding model, we show that it produces competitive results on standard optimization test functions when compared to a number of classical optimization schemes.

The remainder of this work is structured as follows. In Section 2, the models of spiking neurons and synaptic plasticity which are deployed in this

work are introduced. Furthermore, related work concerning other models for variable binding is discussed. In Section 3, we construct soft winner-take-all circuits in NEST. This is a prerequisite for the implementation of variable binding with neural spaces since each neural space acts as a winner-take-all circuit. In Section 4, we construct and evaluate variable binding models in NEST and perform some of the experiments from [9]. In Section 5, the new optimization algorithm is described and its performance is evaluated. The main text concludes with a discussion in Section 6. Further details such as an in-depth investigation of the neuron model, the parameters used for all simulations, and the bibliography are given in the Appendix.

2 Background

This section discusses the models for neuronal activity and synaptic plasticity on which the model investigated in this work is built upon. Furthermore, related work is reviewed.

2.1 Spiking Neuron Models

Neurons are the cells which make up the nervous system and through their properties allow for the processing of signals within animals and humans. They have been widely studied both in vitro and in vivo, leading to an understanding of their electrophysiological properties.

Although there are many different types of neurons, the essential characteristics are shared among all of them (Figure 1). Each neuron has a number of fibers – the dendrites – which collect input signals which lead to local electrical potentials. The dendrites relay these potentials until they reach the cell body, the soma. Here, the potentials are integrated, and if a certain threshold is reached, the soma generates an action potential. This action potential, or spike, is then propagated down the axon towards synapses which form connections with other neurons.

Due to the surge of interest in machine learning in recent years, the most commonly used neuron model is the model used in artificial neural networks. It implements a nonlinear transformation of the form

$$y = f(\mathbf{w}^T \mathbf{x}) \,, \tag{1}$$

with vectors $\mathbf{x}, \mathbf{w} \in \mathbb{R}^N$ and some function $f : \mathbb{R} \to \mathbb{R}$. This transformation maps an N-dimensional ($N \in \mathbb{N}$) input vector $\mathbf{x}$ onto a scalar y by means of a weight vector $\mathbf{w}$ and some nonlinear function f [12]. Networks consisting of huge numbers of these neurons have been successfully applied in machine learning to various problems ranging from image recognition [13] to controlling complex environments like video games [14].

Despite drawing inspiration from biological neurons [15], this type of neuron model — in which scalars are passed from one unit to another and an external clocking device controls the flow of execution — does not resemble biological neurons very well. In biological neurons, many biochemical processes take place simultaneously. Different ion concentrations inside the cell compared to its surroundings lead to the polarization of the membrane [16]. This membrane potential is one of the main variables which determine the state of the neuron, and it is continuous in time (unlike the state of neurons in artificial neural networks).

Furthermore, biological neurons communicate mostly via spikes, which are propagated through the axon, where synapses connect the presynaptic

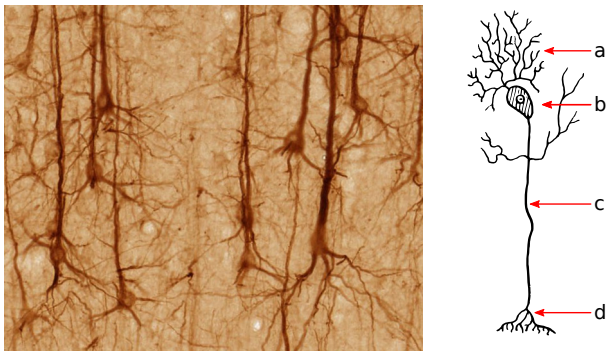

Figure 1: Biological neurons. Left: pyramidal neurons in macaque prefrontal cortex. (Source: brainmaps.org, CC BY 3.0). Right: sketch of a neuron with dendrites (a) delivering input to the soma (b), where currents are integrated. If a spike is generated, it is propagated through the axon (c) towards connections to other neurons' dendrites (d).

neuron to other (postsynaptic) ones. The action potential is transmitted via chemical messengers in the synapse and can cause a postsynaptic potential in the receiving neuron. This kind of interaction is discrete: action potentials have a stereotypical shape and duration, and information is transmitted only via the presence or absence of a spike. This leads to a fundamentally different form of communication than in artificial neural networks, where scalars are passed.

It follows that different neuron models are necessary to model networks of biological neurons and their interactions. While some models target biophysical properties such as voltage-gated ion channels [17], it is easier to focus on the electrical properties. The first property is the summation (integration) of currents which arise due to presynaptic spikes and influence the membrane potential in the soma. The second property is the generation of an action potential once the somatic membrane potential has cross a certain threshold value. Furthermore, the soma has a characteristical resting potential, to which it returns after the membrane potential has increased due to some input.

These properties naturally lead to the Leaky Integrate-and-Fire (LIF) model: the soma is a leaky integrator which sums up incoming currents but always returns to its default value. If the integrated value reaches some threshold, a spike is generated. The first part can be modeled [18] by the equation

$$\tau_m \frac{du}{dt} = -(u - u_0) + R_m \cdot I \ , \tag{2}$$

which describes the development of the membrane potential u over time.

The neuron possesses a membrane resistance R_m as well as a membrane capacitance C_m, which define the integration time constant $\tau_m = R_m C_m$. Incoming currents I arising from presynaptic activity are integrated, and u decays to u_0 when the input vanishes.

Spike generation in this model works by means of an external mechanism which monitors u over time. If its value exceeds some threshold ϑ, an output spike is generated and the value of u is set to a reset potential u_r, from which it may again evolve freely. Typically, the reset potential is chosen as $u_r < u_0$ so the neuron undergoes a period of reduced excitability, which implements the relative refractory period which is observable in neurons [16]. It is also possible to clamp the value to u_r for some period of time to implement an absolute refractory period. Output spikes add some fixed quantity to the input currents I of postsynaptic neurons, thus, the communication between neurons is discrete (as desired).

Using both a relative and absolute refractory effects as well as a reset potential $u_r \neq u_0$, the linear differential equation (2) has the solution

$$\begin{aligned} u(t) = u_0 + (u_r - u_0) \exp\left(-\frac{t - \hat{t} - \Delta_{\text{abs}}}{\tau_m}\right) \\ + \frac{1}{C_m} \int_0^{t-\hat{t}-\Delta\text{abs}} \exp\left(-\frac{s}{\tau_m}\right) I(t-s) ds \end{aligned} \tag{3}$$

for $t > \hat{t} + \Delta_{\text{abs}}$, where $\hat{t}$ is the time of the most recent somatic spike and Δ_{abs} is the duration of the absolute refractory period. We see here that the first two summands describe the stationary and refractory behavior, while the integral models the response to input currents (which include presynaptic spikes). Replacing the terms with generic response kernels, we may write

$$u(t) = \eta(t - \hat{t} - \Delta_{\text{abs}}) + \int_0^{t-\hat{t}-\Delta\text{abs}} \kappa(s) I(t-s) ds \tag{4}$$

with the two newly introduced kernels $\eta(s) = u_0 + (u_r - u_0) \exp(-s/\tau_m)$ and $\kappa(s) = \exp(-s/\tau_m)$.

This idea of using generic kernels which may be fitted to experimental data leads to the Spike Response Model (SRM) [19]. It uses the membrane potential

$$u(t) = \eta(t - \hat{t} - \Delta_{\text{abs}}) + \int_0^{\infty} \kappa(s) I(t-s) ds \tag{5}$$

where spikes are generated if the membrane potential crosses a time-dependent threshold $\vartheta(t)$ from below.[1] This model is more general than the LIF model

[1] The model given here is a simplified version of the more complex general SRM and is also referred to as SRM_0.

since different kernel shapes can be used. Often, an additional kernel $\epsilon(s)$ is used to describe the responses to incoming action potentials (in contrast to currents injected into the neuron).

Due to the time-dependent firing threshold $\vartheta(t)$, the SRM is well-suited to model the stochasticity of neuronal activity. Many noise sources occur in vivo, from background noise due to the diverse inputs to neurons [18] to the unreliability of vesicle release in synapses [20, 21]. These can be modeled by choosing the behavior of $\vartheta(t)$ appropriately.

When fitting the SRM to data from in vitro neuron recordings, it was shown that the spiking probability of neurons is well approximated by an exponential function of the membrane potential [22]. The exponential dependency had already been at the core of another commonly used neuron model [23]. Using this observation, the threshold crossing condition for spike emission can be replaced by an instantaneous spiking probability $\rho(u)$ according to

$$\rho(t) = a \cdot \exp\left(b \cdot u(t)\right) \tag{6}$$

with some coefficients a and b. The neuron then represents an inhomogeneous Poisson process with rate $\rho(t)$, with the probability of a spike occuring within some time interval Δt being close to $\rho(t) \cdot \Delta t$ if Δt is sufficiently small. This model is thus well suited for implementation in digital simulators which work using discrete time steps and will be used in our simulations.

Before we move on to models of synaptic plasticity, it is important to point out the limitations of this model. All models introduced here describe so called point neurons, where all input currents are summed up at a single location (the soma). While this is a reasonable approximation and these models achieve good results when fitted to experimental data, biological neurons are much more complex. Previously, the dendritic tree, through which postsynaptic potentials travel to the soma, was thought of as a merely passive conductance. Recently, it has become clear that this is not the case and the dendrites have a much more active function in information processing [24, 25]. Neuron models consisting of more than a single point of integration — so called multi-compartment neurons — have subsequently been applied successfully in machine learning [26] as well as in computational neuroscience [27, 28]. Furthermore, neurons possess a number of homeostatic mechanisms like intrinsic excitability [29] which are not modeled by (6).

2.2 Models of Synaptic Plasticity

Artificial neural networks generally produce continuous outputs which are differentiable with respect to the network weights, thus, these networks can be conveniently trained by gradient descent and error backpropagation [30].

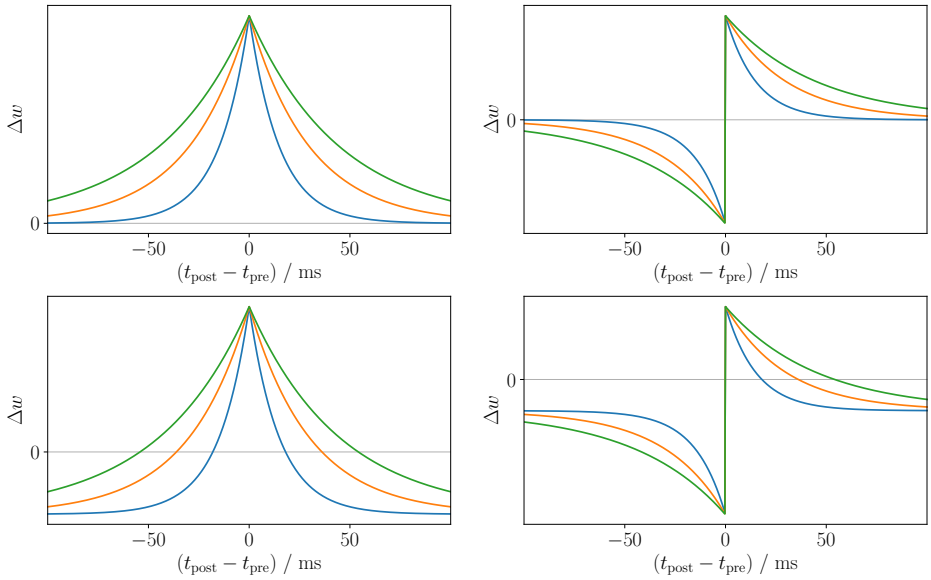

Figure 2: Different types of STDP learning windows. Each plot shows the weight change as a function of the time difference of pre- and postsynaptic spikes for three different time constants. Note that all curves here are perfectly symmetric or antisymmetric, this is not necessarily the case for data obtained from biological neurons. Top left: Hebbian plasticity, correlated firing leads to potentiation. Top right: standard antisymmetric STDP between excitatory neurons, potentiation or depression depends on the order of pre- and postsynaptic spikes. Bottom left and right: Hebbian and antisymmetric STDP with negative offsets, respectively.

Biological neural networks, however, differ in a few important ways: First, they communicate using spikes, which produces non-differentiable signals, and second, the connectivity is generally sparse. How can these networks be modified to produce a desired behavior? One commonly used possibility, which has been thoroughly established experimentally, is by means of synaptic plasticity.

In 1949, Donald Hebb famously published a postulate that is commonly phrased as "neurons which fire together, wire together" [5]. In this view, synapses between neurons which show correlated activity should experience long-term potentiation (LTP), while the synaptic weights between neurons which fire in an uncorrelated manner should potentially decline and undergo long-term depression (LTD). Indeed, it has been shown that there exists a link between synchronous activity and connectivity (i.e. an existing synapse and a large synaptic weight), albeit in a slightly different form: it was found that changes are introduced in synaptic weights depending on the timing difference between a pairing of a pre- and a postsynaptic spike. This mechanism is called spike timing-dependent plasticity (STDP) and has been firmly established experimentally [31, 32].

With STDP, the difference in timing of spikes in the pre- and the postsynaptic neurons defines the magnitude of the weight change. In its most common form, which is typically found between excitatory neurons, the weight is increased if the presynaptic spike has aided the postsynaptic neuron in eliciting a somatic spike, i.e. the weight is increased if the presynaptic spike occured before the postsynaptic one. If neurons fire in the opposite order, the weight is decreased because apparently the neuronal activity is uncorrelated. This leads to an antisymmetric learning window (Figure 2, top right) instead of the symmetric learning window resulting from Hebbian plasticity (Figure 2, top left).

Using these learning rules, the amounts of potentiation and depression may not be well balanced. In biological neurons, regulatory processes such as synaptic scaling ensure the overall weight balance of neurons [33]. These processes often occur on long time scales and thus are not part of the experimentally obtained STDP learning curves. A simple way to model the regulatory processes and to ensure that the amount of potentiation is bounded is by introducing a negative offset to the learning windows (Figure 2, bottom left and right).

To implement STDP in simulations of spiking neural networks, a mathematical model is required. One commonly used model [34] uses the update equations

$$\Delta w(\Delta t) = \begin{cases} \lambda f_+(w) \cdot \mathrm{e}^{-|\Delta t|/\tau_+} & \text{if } \Delta t \geq 0 \\ -\alpha\lambda f_-(w) \cdot \mathrm{e}^{-|\Delta t|/\tau_-} & \text{if } \Delta t < 0 \end{cases} \tag{7}$$

which depend on the timing difference of pre- and postsynaptic spikes $\Delta t = t_{\text{post}} - t_{\text{pre}}$. Here, $\tau_+, \tau_- > 0$ are time constants defining the width of the learning window, α determines the size of the negative Δt update in relation to the positive one, and λ is a learning rate. The two weight-dependent kernels $f_+(w)$ and $f_-(w)$ are given as

$$f_+(w) = (1 - w)^{\mu_+} \qquad \text{and} \tag{8}$$
$$f_-(w) = w^{\mu_-} \ . \tag{9}$$

To obtain the curves depicted in Figure 2 (top left and right), we set $\mu_+ = \mu_- = 0$ to get weight-independent updates and set $\tau_+ = \tau_-$ for perfect (anti-)symmetry. Then, using $\alpha = -1$ gives the Hebbian style plasticity, while $\alpha = 1$ leads to antisymmetric updates. A negative offset can be added by simply subtracting a constant in both cases in (7).

This formulation of STDP and other variants where the weight update depends only on the timing difference of pre- and postsynaptic spikes as well as on the current weight value have been shown to be computationally powerful. Networks using STDP in winner-take-all (WTA) circuits have been shown to perform expectation maximization (EM) on given inputs [35, 36].

However, STDP alone is unlikely to enable the learning of sufficiently stable representations in the brain, since this plain form of STDP provides no mechanism for consolidating weights after learning has been successful. Extensions of STDP have been proposed which use refined mechanisms to address this problem, e.g. with reward-modulated learning [37]. Furthermore, from an investigation of the exact cellular mechanisms of synaptic plasticity a more complex picture has emerged. It has been mentioned above that the dendrites play a more significant role in processing information within each neuron. Dendritic signals also play a crucial role for synaptic plasticity, especially local dendritic spikes [38, 39]. Subsequently, new neuron models have been proposed which incorporate mechanisms like branch-strength potentiation [27] and other fundamental cellular mechanisms [40]. These discoveries have also lead to new hypotheses about the functional role of excitatory neurons [41, 42].

2.3 Related Work

This work investigates variable binding, a mechanism which was proposed as a central computation operation in cognitive systems [3] in the form of pointers stored in assemblies of spiking neurons [9]. Here, we briefly review other models for variable binding which have been proposed. Different types of mechanisms have been brought forward as hypotheses. We focus on three groups: anatomical, convolutional, and pointer-based variable binding.

Anatomical Binding. The anatomical binding scheme seeks to address the binding problem by providing several variables (or registers) which can each store some content. Since items may occur more than once in sentences (e.g. "The red truck is larger than the blue truck.") spaces must be present in multiple instances. This leads to a coding problem: it is not clear how to ensure that the activity in multiple spaces is identical when binding to the same concept. A solution to this problem is proposed in [43]: an external, associative network which is trained to reliably translate the contents stored in some space in a specific encoding to the encoding used in another space. This network has a number of stable attractor states, each of which encodes a unique symbol. Thus, the representations in each distinct variable space need not be the same.

The approach investigated in this work has a somewhat different motivation: instead of solving the problem of anatomical binding by proposing an external network, we start with the experimentally verified concept cells [4], which are unique in our model. Since variables only store pointers to these concept cells (or assemblies), the activity in different variable spaces is not required to be identical. This circumvents the problem of anatomical variable binding. On the other hand, the content space from this approach could be regarded as exactly the associative network which translates different representations into each other.

Convolutional Binding. One of the earliest models deploying convolutional variable binding introduces Holographic Reduced Representations [44]. These are high-dimensional representations of (generic) items in form of constant-length vectors. Associations are formed between items by performing a circular convolution between the two corresponding vectors. (Regular convolution or other operations expand the dimensionality of the vector space, which is difficult to handle.) The output of such a convolution operation is a so-called trace, which is a vector of the same length as each item, and which stores a single association. From such a trace, one of the associated items can be restored given the other item. More complex association types, e.g. sequences, can also be formed. Using this framework, variable binding is implemented simply as a regular association of an item representing the variable with another item representing the content.

Another type of convolution variable binding is used in [45], in which a large-scale model of the brain is introduced. This model exhibits a variety of different behaviors and is composed of several different modules resembling different parts of the cortex and the basal ganglia. The model uses the Semantic Pointer Architecture [46]: data is generally represented by high-dimensional vectors, but the communication between modules utilizes compressed representations of these vectors. These pointers typically have lower dimensionality than the content they reference, nevertheless, the

compression is fully reversible. The compression mechanism can either be learned or explicitly defined. In the model, each vector is encoded in the activity of a population of spiking neurons, while the vector representation can be recovered using a decoding mechanism.

While convolutional variable binding approaches provide an intriguing mathematical framework for the representation of memory associations, it is unclear to what extent the underlying mechanisms are implemented in the brain. Generally, these approaches rely on specific connectivity for manipulation of the high-dimensional representation of data, e.g. specific circuitry for computing the circular convolution and its inverse in [44] as well as the (de-)compression mechanisms in [45]. Such specialized connectivity has to date not been found in the brain. Furthermore, in the presented models the encoding of high-dimensional data in spike trains generally relies rather heavily on the contribution of each individual neuron encoding the vector. It is thus unclear how these mechanisms could be implemented in the noisy environment of the brain, where background noise, continuous rewiring of all connections and failures of individual neurons are common.

In contrast to the mechanisms used by convolution variable binding, the model investigated in this work generally assumes no specific circuitry for manipulation of high-dimensional vectors. Concepts are not represented by some vector, but by the activity of concept cells, which have have been found in numerous studies [4]. Thus, items can be represented by the activity of an assembly of neurons. Different assemblies do not necessarily need to have the same size, and they are robust against the loss of one or a small number of neurons. Furthermore, only standard STDP is used in this work for neuronal interaction.

Pointer-Based Binding. The model investigated in this work falls into this category, since pointers are stored in variable spaces, which link towards a unique area in which items are represented.

In [47], a variable binding scheme is proposed which is motivated in terms of neuroanatomy. In this model, a number of variable slots is available in the pre-frontal cortex, to which values can be assigned. The routing of information from and to these slots is maintained by the basal ganglia. One set of slots is used to store items, another is used to assign the role of these items in a sentence. In simulations with biologically realistic neurons, this model is shown to perform well on a simple sentence encoding and decoding task. This model is similar to the one discussed in this work in a number of ways, e.g. it also uses pointers to address data. However, an elaborate controller mechanism is necessary to guide the interaction of content and pointer slots even for a simple store and recall task. The model investigated in this work uses globally available concept cells, which have been established in the neuroscience literature [4] to avoid the necessity for complex circuitry.

Another work which uses pointers is presented in [48]. Here, a variable binding mechanism binds the activity of a content space to a pointer space using synaptic plasticity. Associations are formed through the joint activation of the two spaces. This leads to the potentiation of synaptic weights between the two. The pointer can later be activated to restore the activity in the content space (which must be set to function in a winner-take-all fashion during this time). To allow pointers to be re-assigned, all synaptic weights decay over time at a constant rate, thus, assemblies only persist for a short period of time. Furthermore, two atomic neural computations for variables are proposed in [48] which go further than simple store and recall tests: the copying of one pointer to another variable, and a mechanism for comparing the contents of two variables. In contrast to this model, the mechanisms investigated in this work focus on the emergence of assemblies in neural spaces which are persistent over time, thus, synaptic weights and neuron activities are not correlated at all times. This is biologically more realistic in terms of the timescales on which synaptic plasticity is thought to take place on.

Finally, a functionally similar approach to the one presented in this work is the Neural Blackboard Architecture [49]. In blackboard architectures, the working data is stored in some spaces (the blackboard) which is accessible to a set of processors. Each of these can read and modify the contents of the blackboard according to some specific functionality which they implement. In [49], items are not copied to the blackboard from elsewhere, but the current state of the blackboard binds to some items, which are represented as assemblies of neurons. A number of such blackboards is used as variables which may store pointers to specific contents. Furthermore, these variables can be combined in various ways to allow the encoding of structural relations between different variables. For these operations, specific circuitry is necessary, which is one of the main criticism of this model since evidence for such circuits remains to be found in the brain. While this model is similar to the model discussed in this work in terms of functionality of variable spaces, it does not provide extensive simulations to support the proposed architecture. Some experiments are conducted with populations modeled as a whole. This work instead focuses on the neuron- and synapse level implementation of variable binding.

3 Soft Winner-Take-All Circuits

Winner-take-all (WTA) circuits form an essential component of the variable binding model in this work. They are important because such a neuronal motif ensures that one or more neurons of a population respond exclusively to some input pattern. Populations with WTA functionality also lie at the heart of other models which have been proposed for variable binding [47, 48].

In contrast to hard WTA circuits, where a single neuron becomes active if the corresponding input is present (investigated in [36], for instance), we use soft winner-take-all (SWTA) motifs. Here, multiple neurons represent each input pattern. This leads to the formation of assemblies of neurons encoding different input classes, which have the advantage of being more robust than single neurons (e.g. the loss of a single neuron has no drastic influence on the overall network behavior).

In this section, we implement SWTA motifs in NEST which show similar behavior to those used in [9].

3.1 EI-Motifs

EI-motifs consist of an excitatory and an inhibitory population of neurons, which interact with each other. Usually, the E-pool is thought to perform some computational function, while the I-pool mainly acts as a regulator for the dynamics of the excitatory neurons, i.e. to stabilize the average population rate in some regime. In the brain, excitatory neurons interact with a host of inhibitory neurons, thus, modeling inhibition through a separate pool of spiking neurons is preferred over other mechanisms (such as using a filtered version of the network activity to calculate an inhibitory current for all neurons as done in [9]).

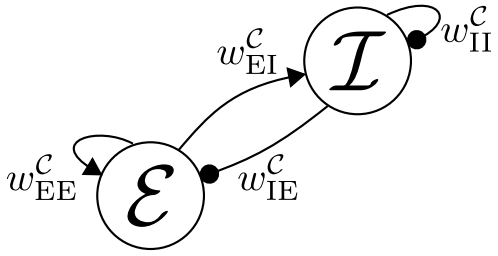

Figure 3: Network layout of an EI-motif. $\mathcal{E}$ denotes the pool of excitatory, $\mathcal{I}$ the pool of inhibitory neurons. Arrows denote excitatory, circles inhibitory connections.

A schematic illustration of an EI-motif is depicted in Figure 3. Neurons from the E-pool project to the I-pool with excitatory connections, while the I-pool provides inhibition to the E-pool. Each pool also has recurrent

connections, which are excitatory and inhibitory for the E- and I-pool, respectively. It follows that the weight range is positive for w_{EE} and w_{EI} and negative for w_{IE} and w_{II}. While sometimes all possible connections of some type are present ("all-to-all" connectivity), usually, connections are drawn at random with connection probabilities $p_{\mathrm{EE}}, p_{\mathrm{EI}}, p_{\mathrm{IE}}$, and p_{II}. Each connection furthermore uses some synaptic delay, which in our case is either constant or static but randomly drawn at connection creation time.

When using EI-motifs as part of a larger neuronal circuit, typically, the E-pool receives input from other parts of the network and also projects elsewhere. This allows the E-pool to implement useful information processing functionality.

Next, we describe the neuron model and the network parameters used for the experiments in this section.

3.2 Implementing EI-Motifs in NEST

First, we describe the neuron model used in our simulations. An in-depth description of the neuron model from [9] and the arguments leading to the model used in NEST are given in Appendix A.

In the model presented in this work, each neural space consists of a population of excitatory and inhibitory neurons. The excitatory pool is used to perform information processing, while the inhibitory pool stabilizes the activity of the excitatory neurons. For both types, we use neurons which fire according to an inhomogeneous Poisson process when they are not refractory as described in the previous section (6). Specifically, each neuron i fires with rate $\rho_i(t)$, which is given by

$$\rho_i(t) = c_1 \cdot V_i'(t) + c_2 \cdot (\mathrm{e}^{c_3 \cdot V_i'(t)} - 1) \;, \tag{10}$$

where c_1, c_2, and c_3 can be set to obtain either exponential or linear behavior. The effective membrane potential $V_i'(t)$ is the sum of the actual membrane potential $V_i(t)$ and an adaptive bias term. The former is given by

$$\begin{aligned} V_i(t) = \mathrm{e}^{-\Delta t/\tau_m} V_i(t-\Delta t) + (1-\mathrm{e}^{-\Delta t/\tau_m}) \frac{R_m}{1000} \left(I_{i,c}(t-\Delta t) + I_e\right) \\ + z_{\mathrm{scale}} I_{i,s}(t) \;, \end{aligned} \tag{11}$$

where Δt is the duration of each time step, τ_m is the membrane time constant, and the currents $I_{i,c}(t) = \sum_j w_{ij} I_j(t)$ and $I_{i,s}(t) = \sum_j w_{ij} z_j(t)$ result from injected currents from current generators and from synaptic input, respectively. The value of z_{scale} can be adjusted to scale the response to spikes. I_e is a fixed bias current.

We use neurons with membrane resistance $R_m = 10$ MΩ and membrane capacitance $C_m = 1000$ pF in our simulations, thus, the membrane time

constant is $\tau_m = 10$ ms. All neurons undergo an absolute refractory period which is static per neuron but sampled from a Γ-distribution at neuron creation time. No excitability (adaptive bias) is used in the experiments in this section.

The excitatory neurons should show exponential behavior to achieve WTA dynamics [36]. We thus set $c_1 = 0$. Furthermore, we use $c_2 = 1000$, $c_3 = 100$, and $z_{\text{scale}} = 5 \cdot 10^{-4}$ to obtain similar behavior as in [9] (see Appendix A for details). The inhibitory neurons are linear ($c_2 = c_3 = 0$) with $c_1 = 1000$.

For synaptic plasticity, a learning rule inspired by a model implementing hard WTA circuits [36] is used. Each pair of pre- and postsynaptic spikes leads to a weight update according to

$$\Delta w(\Delta t) = \begin{cases} \eta \cdot \mathrm{e}^{-|\Delta t|/\tau_+} - A_- & \text{if } \Delta t \geq 0 \\ \eta \cdot \alpha \cdot \mathrm{e}^{-|\Delta t|/\tau_-} - A_- & \text{if } \Delta t < 0 \end{cases}, \tag{12}$$

where τ_+ and τ_- are time constants which determine the width of the learning window, A_- sets an offset, α determines the shape of depressing updates in contrast to facilitating ones, and η is a learning rate. In this work, we use similar parameters to those given in [9] for connections from an input population to the excitatory neurons with $\alpha = 0$, $\tau_+ = 20$ ms, $A_- = 0.4$. For reasons discussed below (Section 4.2), we use different values for the recurrent connections within the excitatory pool, for which we set $\alpha = 1$, $\tau_+ = \tau_- = 25$ ms, and $A_- = 0.5$.

Next, we derive the values for the bias current as well as for the connections within and between neuron populations.

In our model, the input and output weights as well as the recurrent weights of the E-pool should be subject to plasticity. The weight values for w_{IE}, w_{IE}, and w_{II} are static for the I-pool to reliably provide inhibition for the E-pool.

In [50] and [51], an EI-motif without recurrent excitatory connections is investigated. To achieve SWTA dynamics, a prior activity distribution is assumed: a discrete Gaussian with some mean μ and variance σ^2 which is truncated to the positive domain (as only a non-negative number of neurons can be active at some time). Using this activity prior, the bias current for excitatory neurons is given by

$$I_e = \frac{2\mu - 1}{2\gamma\sigma^2}, \tag{13}$$

where γ corresponds to c_3 from (65) in our case as it is the factor scaling the membrane potential in the exponential when calculating the firing rate. The weights are set according to

$$w_{\mathrm{EI}} = 10 \cdot \frac{c_{\mathrm{PSP}}}{p_{\mathrm{EI}}} \,, \tag{14}$$

$$w_{\mathrm{IE}} = -\frac{c_{\mathrm{PSP}}}{\gamma\sigma^2} \,, \text{ and} \tag{15}$$

$$w_{\mathrm{II}} = -10 \cdot \frac{c_{\mathrm{PSP}}}{p_{\mathrm{II}}} \,, \tag{16}$$

where if $p^{\mathrm{II}} \approx p^{\mathrm{EI}}$, we can also use $w^{\mathrm{II}} = -w^{\mathrm{EI}}$. Here, c_{PSP} is a correction term for the PSP shape and denotes ratio of the integrals over the PSP used in [51] and the PSP which is used instead:

$$c_{\mathrm{PSP}} = \frac{\int \alpha_0(s)ds}{\int \alpha(s)ds} \,. \tag{17}$$

$\alpha_0(s)$ has rectangular shape with height 1 for $t \in [0, 10]$ ms, thus, $\int \alpha_0(s)ds = 0.01$ s.

connection	E→I	I→E	I→I
probability	0.575	0.6	0.55
synaptic delay / ms	0.5	0.5	1
weight	17.391	−0.333	−16.667

Table 1: Parameters for EI-motif.

Since both the neuron model and the update equations in [51] are significantly different than ours, we set $\gamma = 1$ and $c_{\mathrm{PSP}} = 1$ for calculating the weights for our network.[2] Furthermore, biologically plausible connection probabilities for the connections between the E-pool and the I-pool are also given by [51].[3]

To calculate the weights, we now only need to choose the parameters μ and σ^2 for the activity prior. In [51], μ is chosen < 0, however, we need to set $\mu > 0$ to get a nonzero firing rate.[4] We therefore choose $\mu = 1$ and $\sigma^2 = 5$ for our parameter calculations. Furthermore, the synaptic delays for connections between E- and I-pool were set to 0.5 ms to get the same overall

[2]The motivation for former is that in our model, c_3 and z_{scale} are set to achieve similar dynamics to the original, which itself is similar (but not identical) to the model in used to derive these relations for setting the weights. For c_{PSP}, we note that their model does not behave as a leaky integrator, thus, we need to scale our delta-shaped PSP by the membrane time constant $\tau_m = 10$ ms which corresponds to the integral over $\alpha_0(s)$, thus leading to ratio of 1.

[3]No probability for connections E→E is given there.

[4]The intuition of the EI-motif is that the bias sets the network activity rather high, which is then scaled to the desired target activity by the inhibition, which is stronger or weaker depending on the amount of external input to the E-pool.

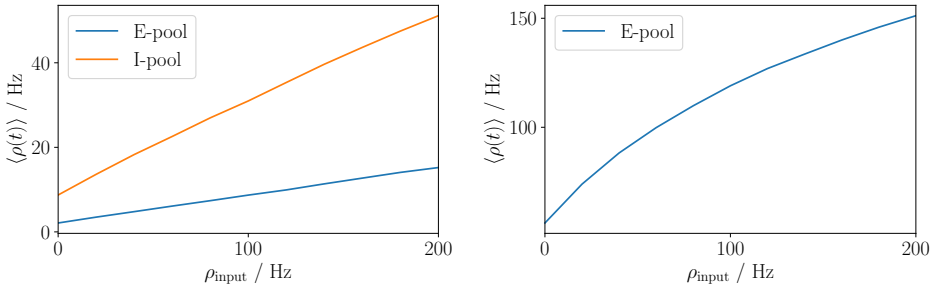

Figure 4: Left: balance of activity in an EI-motif. E-pool and I-pool consist of 1000 and 250 neurons, respectively. Input to the E-pool is provided by 25 neurons firing stochastically with rate ρ_{input}. Each excitatory neuron receives input from each input neuron, the weights for these connections are static and drawn from $\mathcal{U}(0, 0.8)$. No recurrent connections within the E-pool are present. Right: same setup, but no I-pool is present.

delay in inhibition as in the original model which uses a filtered version of excitatory neuron activity to calculate the inhibition. The final parameters are given in Table 1.

To test these parameters, we construct an EI-motif with some external units connecting to the E-pool. We vary the firing rate of these units and plot the average firing rate within each pool. The activity of the E-pool should stay more or less constant, which is the case (Figure 4).

3.3 SWTA Circuits in NEST

After constructing EI-motifs with balanced dynamics in NEST, we move on to implementing SWTA circuits. These build on top of the EI-motif described above, and use the same the neuron and synapse model.

Model Description. The network layout is shown in Figure 5. The SWTA circuit, which consists of a pool of 1000 excitatory and a pool of 250 inhibitory neurons, receives input from a different pool $\mathcal{X}$. These input neurons fire according to Poisson processes and display different patterns to the network through their firing rates. Each pattern resembles a binary vector, where ones are coded as a high firing rate (100 Hz) and zeros as a low firing rate (0.1 Hz).

The neuron parameters have been described above and are summarized in Table 6 in Appendix C.1.

The network parameters are given in Table 7 in Appendix C.1. The connection parameters were chosen according to [51] with $\mu = 1$, $\sigma^2 = 5$ and $c_{\text{PSP}} = 1$ as parameters for the activity prior as described above. In contrast to their model, the model used here also has recurrent connections

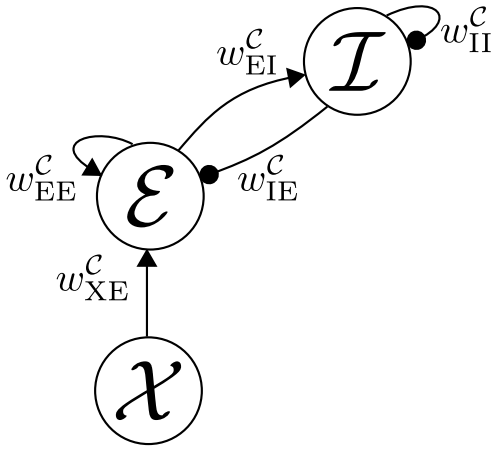

Figure 5: Network layout. An input population $\mathcal{X}$ has been added to the EI-motif shown in Figure 3. Arrows denote excitatory, circles inhibitory connections.

within the E-pool with connection probability 0.1. Synaptic plasticity is applied to the connections X→E and E→E. The parameters can also be found in the Appendix in Table 7.

Training Results The training procedure takes place as follows. To obtain one trained SWTA circuit, the network is presented with 200 input patterns for 200 ms each. For each pattern, the firing rates of the neurons in the input (X) population are set appropriately. Consecutive patterns are separated by 200 ms of random input, where all input neurons fire with $\rho = 12.5$ Hz. Five different input patterns are used, each of which is represented by 25 input neurons. When some pattern is presented, the corresponding neurons firing at a high rate (100 Hz), while the others fire at a low rate (0.1 Hz). After this training period, each pattern is presented to the network once, and the firing rates of the excitatory neurons are measured. If they fire at a rate > 50 Hz, they are counted to the assembly which has formed for this input pattern. Furthermore, it is tested whether neurons respond uniquely to one input pattern or to various ones.

This leads to the following results (averages and standard deviations are given over 10 distinct training runs): of all 1000 excitatory neurons, 334.5 ± 10.6 belong to an assembly after training. The average assembly size is 66.9 ± 10.3. Neurons did not belong to more than one assembly in any of the 10 runs. The progress of learning within the E-pool is shown in Figure 6. Since the learning rate was chosen rather high, the network quickly shows signs of learning. The activity of the input neurons as well as of the E- and I-pool during the test phase are shown in Figure 7.

Overall, the network performs well at separating the input patterns.

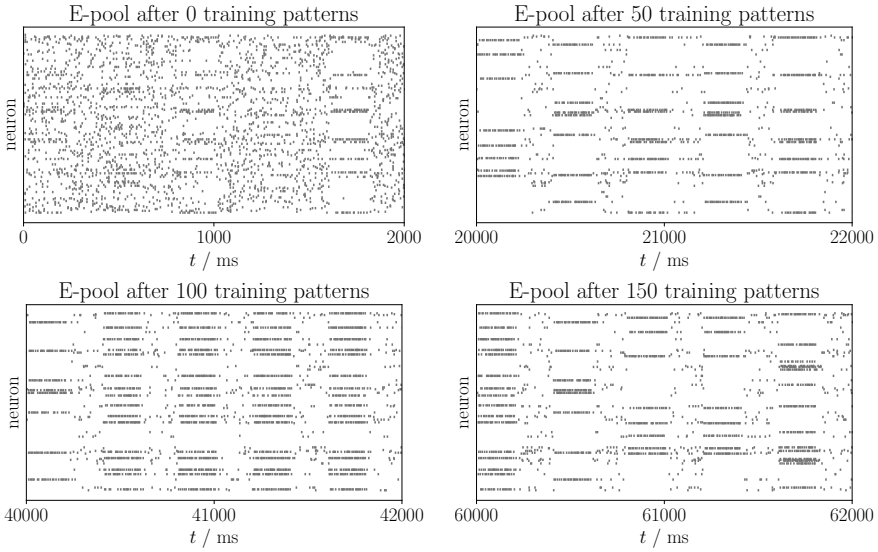

Figure 6: Spikes in E-pool as training progresses. Only spikes of 200 neurons are shown. Note that assemblies responding exclusively to a single pattern emerge rapidly, thus, the overall responses are already fairly stable after 50 presentations of input patterns.

The resulting assemblies are very stable and show no overlap. However, the resulting assembly sizes are somewhat smaller than those obtained in the original model (where each assembly consists of around 80 neurons). While the overall number of neurons in all assemblies is fairly constant, the individual assemblies show a great deal of size variation. This is largely due to the random noise which is given to the network in between pattern presentations: if it is turned off, the resulting assemblies are larger and the assembly sizes show less variance.

Analysis We conclude this section by performing some additional analysis of the network behavior of the NEST SWTA circuits.

Figure 8 shows a histogram of the number of spikes per neuron during the training period. This value is significant since learning only takes place when neurons fire. Thus, the network dynamics need to be adjusted in a way that ensures that enough spikes are elicited in each neuron. In the figure, we can see two groups of neurons: one group shows rather few spikes, while neurons the other one spike very often during training. The first group consists of neurons which are not part of any of the five assemblies. Since their connections do not allow a strong response to an input pattern, their weights decrease. Due to the offset in the learning rule, only a rather small number of spikes is required for the weights to reach zero. On the other

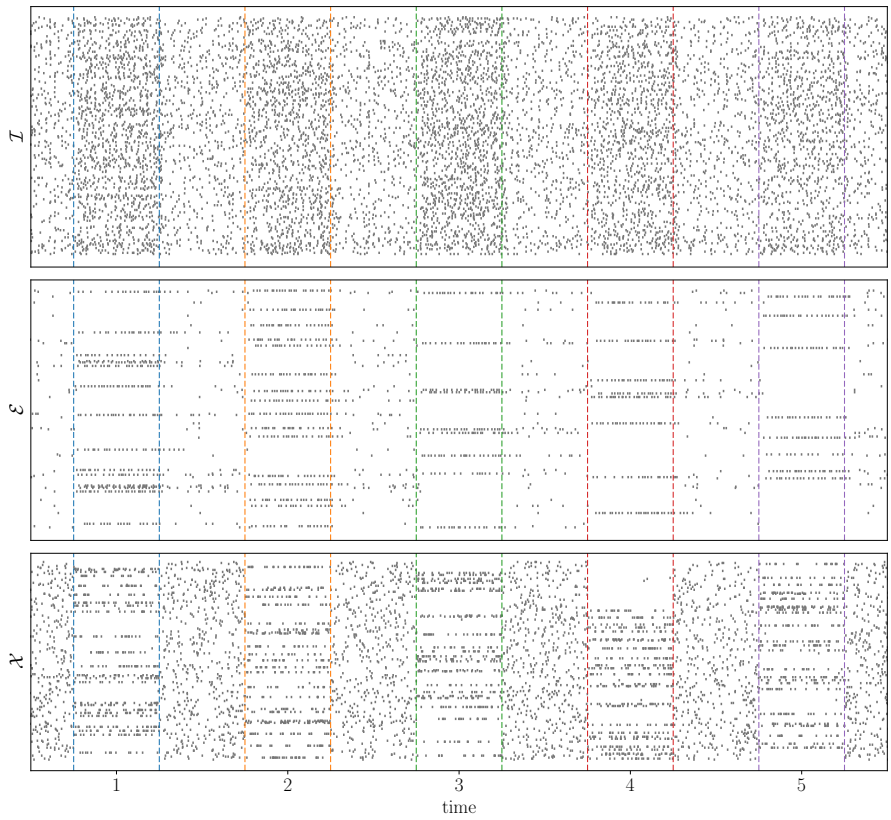

Figure 7: Activity of populations during test phase: input $\mathcal{X}$, excitatory population $\mathcal{E}$, and inhibitory population $\mathcal{I}$. Each of the five patterns from the training phase is present once for 200 ms (as during training) with beginning an end marked by the colored lines. When no pattern is presented, all input neurons fire randomly. Only 200 neurons per population are shown.

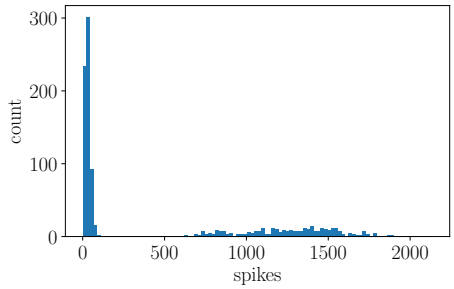

Figure 8: Histogram of number of spikes per neuron during the entire training phase. This plot shows that since most neurons experience some number of spikes during training, learning takes place at their synapses. Two groups are visible: one with a small number of spikes, and one with a far larger number. These correspond to neurons which form assemblies (many spikes) and those which do not (few spikes, synapse weights decline rapidly at the beginning of the training).

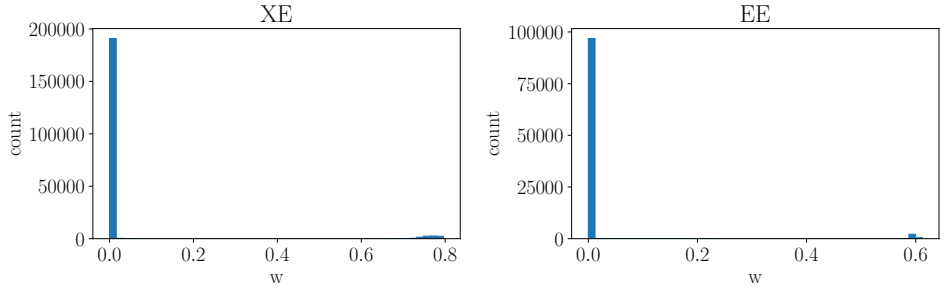

Figure 9: Weight distribution of XE and EE connections (cf. Figure 5) after training in the NEST model.

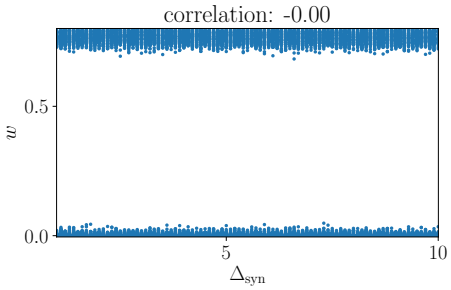

Figure 10: Correlation of synaptic delay and weight after training on XE connections. The quantization of Δ_{syn} arises from NEST requiring synaptic delays to be a multiple of the time constant ($\Delta t = 0.1$ ms by default).

hand, the neurons which are part of an assembly fire continuously during the training period and show a large number of spike in total.

Next, Figure 9 shows the distribution of weights after training. Here, it is clear that the learning procedure has resulted in weights which lead to a strong tuning of assemblies to their respective input (large XE weights, i.e. the weights from input to the E-pool, cf. Figure 5) while the response to other inputs is low (small XE weights). The same holds true for EE weights, which are large within the assemblies which have formed.

Finally, we investigate the correlation of the (randomly drawn) synaptic delay for connections from the input to the excitatory pool and the resulting weight after training. The synaptic delays for these connections can be quite larger, as they are sampled from a uniform distribution $\mathcal{U}(1, 10)$. For connections with a large delay, the inhibition of the I-pool might begin to suppress network activity before the postsynaptic neuron can fire, thus leading to depression. One might therefore conclude that smaller delays generally lead to larger weights.

To test this hypothesis, we investigate the correlation of synaptic delay (Figure 10). As can be seen, no correlation is present after training. The correlation coefficient between delay and weight was also calculated to be (approximately) zero. Apparently, the input is strong enough to make excitatory neurons fire even when the delay is large. Thus, drawing the synaptic delay for X→E connections randomly may serve to decrease correlated firing activity within the E-pool, but it generally does not interfere with learning.

4 Variable Binding

In the previous section, we constructed SWTA circuits, which perform unsupervised separation of input patterns. These circuits are the building blocks of the variable binding model investigated in this work, as each neural space has winner-take-all functionality. In this section, we build on top of the model described above to perform variable binding in a spiking neural network.

4.1 Description of the Original Model

The model presented in this work is based on the model introduced in [9]. We therefore begin by describing this model and discussing some of its properties.

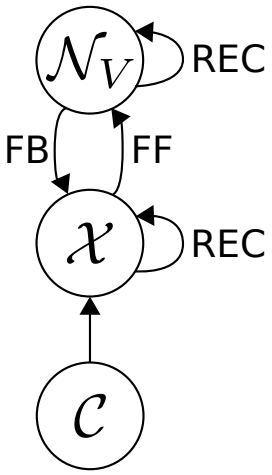

Figure 11: Variable binding network layout. X is the input population, $\mathcal{C}$ is the content space, and $\mathcal{N}_V$ is a variable space used for computations.

Network Architecture The basic network layout of the variable binding model described in [9] is depicted in Figure 11. It is an extension of the circuit used above (cf. Figure 5): an input population $\mathcal{X}$ connects to a neural space (which is an SWTA circuit), which we call the content space.

To perform variable binding, the concept space is now additionally linked to other neural spaces, the so-called variable spaces $\mathcal{N}_V$, $\mathcal{N}_U$, etc. which correspond to pointer variables V, U, an so on. Figure 11 shows the layout with a single variable space present. Each neural space has recurrent connections (REC). The content space projects to each variable space via feed-forward (FF) connections. Furthermore, there are symmetric feed-back connections from each variable space to $\mathcal{C}$.

All five connections visible in Figure 11 undergo synaptic plasticity of the same kind (but with different parameters). Both the content space and the variable spaces have inhibition mechanisms with which all activity and synaptic plasticity can be suppressed.

Functionality The presence of the newly introduced variable spaces allow simple variable binding operations to be performed in this model. Each variable space corresponds to some pointer variable which stores a pointer to some assembly within $\mathcal{C}$. From this pointer, the activity of the target assembly can be restored, e.g. after some delay period. Thus, the variable spaces allow concepts (encoded by active assemblies in the content space) to be tied to roles: if some variable space e.g. encodes the agent when decoding the meaning of a sentence, it can be linked to the entity filling the role in the given sentence.

We now briefly describe a simple binding experiment. Before any experiment takes place, the content space is trained on a number of input patterns until stable assemblies have formed in it (as described above for the SWTA circuits). During this phase, the variable spaces are inhibited.

The experiment then takes place as follows. To store a pointer in a variable, the target assembly in $\mathcal{C}$ must be active. Thus, the input population $\mathcal{X}$ is set to drive the specific assembly. Now, the variable space which is to store the pointer is disinhibited. Through the feed-forward connections, the variable space is activated, and a stable assembly emerges. This assembly is closely tied to the assembly in the concept space by the feed-forward and feed-back connections, which undergo rapid synaptic plasticity.

This pointer in the variable space can be stored over time by means of an adaptive excitability mechanism. This mechanism is implemented as an adaptive bias of each neuron which is increased every time a neuron fires (and otherwise decays over time). After a delay period where all activity is suppressed through global inhibition, this excitability mechanism leads to previously active neurons firing preferentially after all neural spaces are disinhibited. Then, the previously active assemblies in the content space and the variable space drive each other into a stable regime, and thus, the neurons firing in the content space during this period correspond to the neurons from the target assembly.

This experiment is a basic operation in which a pointer is stored and recalled in a variable space. The experiments performed in [9] furthermore include copying the contents from one variable space to another one and comparing the contents of two variable spaces.

Issues In the original model, the feed-forward and feed-back connections are symmetric, i.e. they connect the same two neurons even though the weights and plasticity parameters are different for both direction. In biolog-

ical neural networks, however, perfectly symmetric connections usually do not occur. In this work, we will investigate the possibility of using random connections between the two spaces in the NEST model.

Introducing this change will obviously pose a challenge to the model, as it is then not guaranteed that any given assembly in one space can actually reach all the neurons which project to it. This problem can be resolved by increasing the size of the assemblies inside variable spaces, which is the behavior which naturally emerges in the network when symmetric connectivity is not available.

The problems concerning the implementation of inhibition in the original model were already discussed above and have been solved by using an I-pool to provide inhibition within each SWTA circuit.

4.2 Variable Binding in NEST

Using the SWTA circuits described in the previous section as building blocks, we now construct a network in NEST which performs variable binding. At the same time, we aim to resolve the issues described above.

Architecture The overall network architecture (Figure 12) is the same as in the original model and consists of an input population $\mathcal{X}$, the content space $\mathcal{C}$ and a variable space $\mathcal{N}_V$ (cf. Figure 11). Each neural space is an SWTA circuit and thus – in the NEST model – consists of interconnected pools of excitatory and inhibitory neurons. Since there are many more connections in this models, it has many more parameters which need to be determined. While some are defined by the underlying SWTA functionality that each neural space must provide, many additional degrees of freedom remain.

The content space $\mathcal{C}$ has ingoing connections from an input population $\mathcal{X}$, these weights belong to the content space in the NEST model and are called XE-weights within $\mathcal{C}$ as they connect the input $\mathcal{X}$ to the E-pool of $\mathcal{C}$. Furthermore, each neural space has incoming weights from the other neural spaces; these weights are called SE-weights and are also assigned to the postsynaptic neural space in the NEST model.

The inhibition of neural spaces is implemented by a bias current generator which injects a current with amplitude -2 pA into each neuron if the neural space is inhibited. If the space is disinhibited, the current generator is switched off, and neurons resume their activity.

Avoiding Symmetric Connections The original model uses symmetric connections (with separate weights for each direction) between the content space $\mathcal{C}$ and the variable spaces. To obtain a more biologically realistic model, we draw all connections between the two neural spaces randomly,

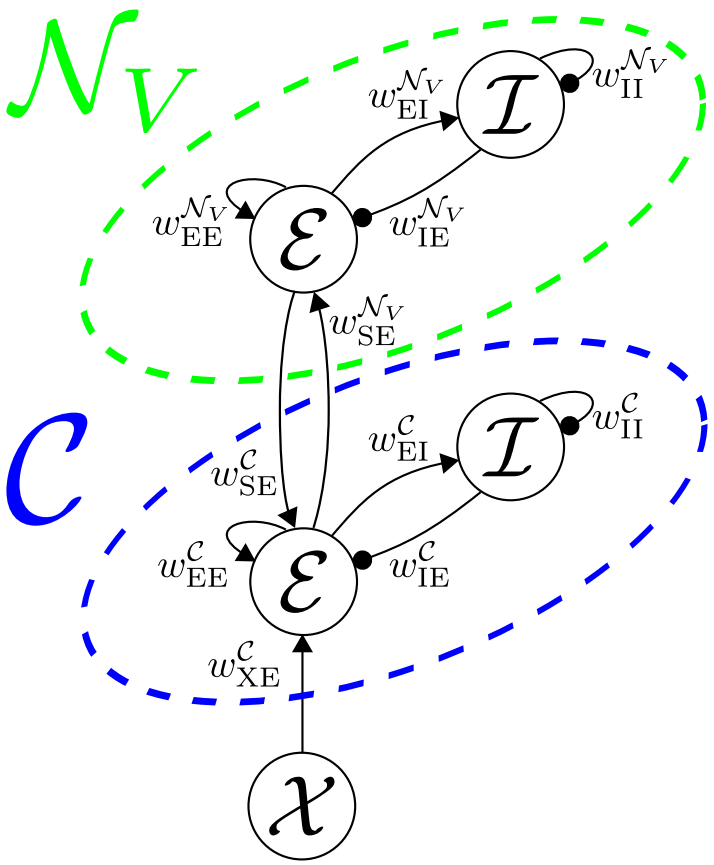

Figure 12: Layout of network consisting of content space $\mathcal{C}$ and a single variable space $\mathcal{N}_V$. Arrows denote excitatory, circles inhibitory connections. The weights of connections between two neural spaces (w_{SE}) are placed inside the same neural space as they belong to in the NEST implementation.

thus, symmetric connections are rather unlikely. It may also occur that a neuron in one space has no connection (or very few connections) to an assembly in the other space. Thus, we need to make provisions for allowing the existing connections to become stronger.

One method to get strong correlations within recurrent neural networks is to use a symmetric STDP window, which allows for a more "Hebbian"-style plasticity in the original sense (fire together, wire together). A learning window of this kind is depicted in Figure 26. This type of plasticity has been shown to occur in the human brain [33], however, it may result in neuron populations persistently exciting themselves very forcefully, and thus to instable network dynamics. It is therefore imperative to carefully balance the weight growth which arises due to this plasticity with the network inhibition.

The synapse model `stdp_synapse_sem` (see Appendix B.2) allows for

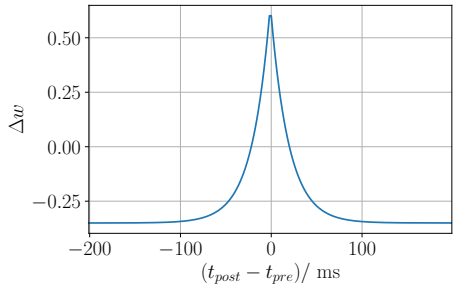

Figure 13: Symmetric learning window which can enforce formation of strong and persistent links.

using this kind of plasticity by setting the α parameter to -1 to get perfect symmetry (see Appendix B.2). Setting $\alpha = 0$ gives the kind of plasticity used in [9].

Since the learning task becomes more challenging when using biologically realistic non-symmetric connections, we alter the parameters for the content space used in [9] to allow assemblies to form stronger recurrent connections. The STDP type for the recurrent (EE) connections within $\mathcal{C}$ was changed to use this symmetric learning window, furthermore the allowed recurrent weight range was increased (see full parameter list in Appendix C.2). The recurrent connections within variable spaces also use this kind of plasticity.

Additionally, we double the size of the variable spaces compared to the original model. Each content space consists of 1000 excitatory neurons. Since the connections between content and variable spaces are not symmetric, a larger group of neurons is required in the variable space to ensure that each neuron in $\mathcal{C}$ receives sufficient input. While the assemblies in the content spaces consist of usually around 60 neurons, the corresponding assemblies in the variable space can be as large as 200 or even 250 neurons. With these sizes, obviously, a variable space size of 1000 neurons is not enough to store five distinct patterns. Therefore, the variable spaces will consist of 2000 excitatory neurons (and 500 inhibitory neurons, to keep the ratio of excitatory to inhibitory neurons constant).

4.3 Experiments without neuronal excitability

We now move on to performing experiments with neural space as described in [9]. We will focus on the first two experiments implemented there. Although an adaptive excitability mechanism is central to the model in [9] on which this work is based on[5], we do not make use of such a mechanism here. The variable spaces are thus not inhibited during delay periods where they

[5]Cf. Appendix A.1, in particular (29) and (46).

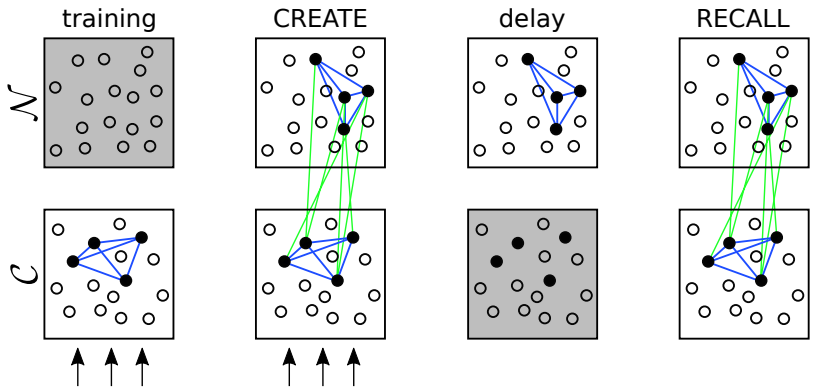

Figure 14: Illustration of the LOAD / RECALL experiment. First panel: the content space $\mathcal{C}$ is trained on some input (indicated by arrows), and stable assemblies form (recurrent connections, blue) in response to some input pattern. The variable space $\mathcal{N}_V$ is inhibited (indicated by the gray background). Second panel: while $\mathcal{C}$ is driven from the input, $\mathcal{N}_V$ is disinhibited. Connections form between spaces (green) and within $\mathcal{N}_V$ (blue). Third panel: during a delay period, $\mathcal{C}$ is inhibited. In the current experiment, $\mathcal{N}_V$ may remain active during this time. Final panel: in the RECALL phase, the original activity in $\mathcal{C}$ is restored from the activity in $\mathcal{N}_V$, and the same neurons are active as before.

should retain a pointer to allow the information to be kept in the recurrent activity.

4.3.1 LOAD / RECALL

This first experiment tests the ability of variable spaces to reliably store a pointer to an assembly in the content space, which should be activated after some delay (Figure 14).

Procedure. Here, the network consist of just two neural spaces (as shown above in Figure 12). The variable space $\mathcal{N}_V$ stores a pointer to some content in $\mathcal{C}$, which is recalled after some delay. This experiment will also be the benchmark we use for finding good parameters for our model.

The procedure of this experiment starts with a content space being trained as an SWTA circuit on five different input patterns. This is achieved as follows.

To train the content space $\mathcal{C}$, a randomly selected pattern is presented for 200 ms, after which 200 ms of random input is given (all input neurons fire at a rate of 12.5 Hz). This is repeated 200 times. Afterwards, each pattern is tested to measure the assembly which has formed in $\mathcal{C}$: the patterns are presented for 200 ms while the firing rates of the excitatory neurons are measured.

The setup of the content space has a few stochastic components. First, the recurrent excitatory connections within the content space are drawn at random. Furthermore, all weights are initialized randomly in some range. To obtain robust parameters which are proven to work independently of the specific random initialization, multiple different instances of the content space have been constructed and will be used during training and testing.

Next, the variable space is set up. To form assemblies in $\mathcal{N}_V$, we perform a CREATE operation for each pattern. While the input is given to $\mathcal{C}$, the variable space is disinhibited. Each CREATE operation lasts for 1000 ms. During the second half of this period, the firing rates in $\mathcal{N}_V$ are measured to determine the assembly which has formed here.

After completing this setup procedure, the experiment takes place as follows.

- **LOAD.** First, a pattern is loaded into the variable space $\mathcal{N}_V$. The procedure is the same as for the CREATE operation, except that it lasts only for 200 ms. The assemblies in the variable space have already been formed, and are now activated. Afterwards, the input neurons in $\mathcal{X}$ fire randomly for the remainder of the experiment.
- **DELAY.** Next, the content space is inhibited for 5000 ms, while $\mathcal{N}_V$ may remain active, thus, the information given to it can be kept in the recurrent activity of the variable space.
- **RECALL.** Finally, we perform a RECALL operation from $\mathcal{N}_V$ to $\mathcal{C}$. After waiting for another 50 ms[6], the content space is disinhibited for 200 ms. The variable space should now drive the content space in such a way that the same assembly forms there which was previously active.

To assess the success of the experiment, the active neurons in $\mathcal{C}$ are recorded during the second half (i.e. the latter 200 ms) of the RECALL phase. They are then compared to the assembly which was active during the presentation of the target pattern after the training phase of the content space.

Parameter Selection We use this setup to determine the parameters for the NEST model. A gradient-free optimization algorithm (see Section 5) is used to find good values for 23 free parameters of the content space and the variable space. As cost function we use the number of neurons which are classified during the RECALL operation as either missing (i.e. they were active when testing the assemblies after training, but not during RECALL) or excess (i.e. they are active during RECALL but previously were not).

[6]This is done for consistency reasons with respect to the following experiments using neuronal excitability. If anything, this makes the procedure more difficult in the current setup.

Two different instances of trained content spaces (with different random connections) were used during training to obtain robust parameters. Since the trial-to-trial variance can be quite high, each content space was tested three times on each pattern (i.e. LOAD / RECALL for each of the five patterns was tested three times with different random connections between $\mathcal{C}$ and $\mathcal{N}_V$ as well as recurrent connections in $\mathcal{N}_V$) when evaluating the cost.

During the optimization, boundaries were enforced on the parameters to ensure that the results are reasonable values. The final results are given in Appendix C.2 in Tables 8 and 9. All parameters for EE connections within variable spaces as well as parameters for connections between variable spaces and the content space were made accessible to the optimization algorithm, in particular, the algorithm was free to use synaptic plasticity with a symmetric learning window ($\alpha = -1$) or with an asymmetric one ($\alpha = 0$) on both feed-forward and feed-back connections between $\mathcal{C}$ and $\mathcal{N}_V$ as well as on the recurrent connections within $\mathcal{N}_V$. The final parameters only use a symmetric learning window for the recurrent connections within each neural space.

To assess the quality of different resulting parameter sets, their performance was tested on three different content space instances. This corresponds to the common machine learning technique of using a validation set which is distinct from the training set.

Results The activity within all neuron populations during a typical run is shown in Figure 15. Before the LOAD operation begins, the content space is disinhibited, and the input neurons fire randomly. Then, a specific pattern is presented, and the corresponding assembly becomes active in $\mathcal{C}$. Simultaneously, the variable space is disinhibited, and the previously (i.e. during the CREATE operation) formed assembly also becomes active there. During the delay period, the content space is inhibited and shows no activity. The variable space remains disinhibited in the current setup, and the previously active neurons continue to fire during the delay period. However, the activity is somewhat sparse, and the neuron firing rates are quite low and well below the thresholds defined for active assemblies. During the RECALL operation, the content space is disinhibited, and the original assembly becomes active again. No neuronal excitability is used in this experiment.

To quantify the performance over different trials, which always show slight variations in neuronal responses, we define a success criterion: if at least 80% of the neurons of the original assembly in $\mathcal{C}$ are active during the RECALL phase, and if furthermore the number of additionally active neurons does not exceed 20% (of the original assembly size), the performance is regarded as successful. Using this criterion, we perform many trials on the different content space instances and report the results, which are given in Tables 2 and 3.

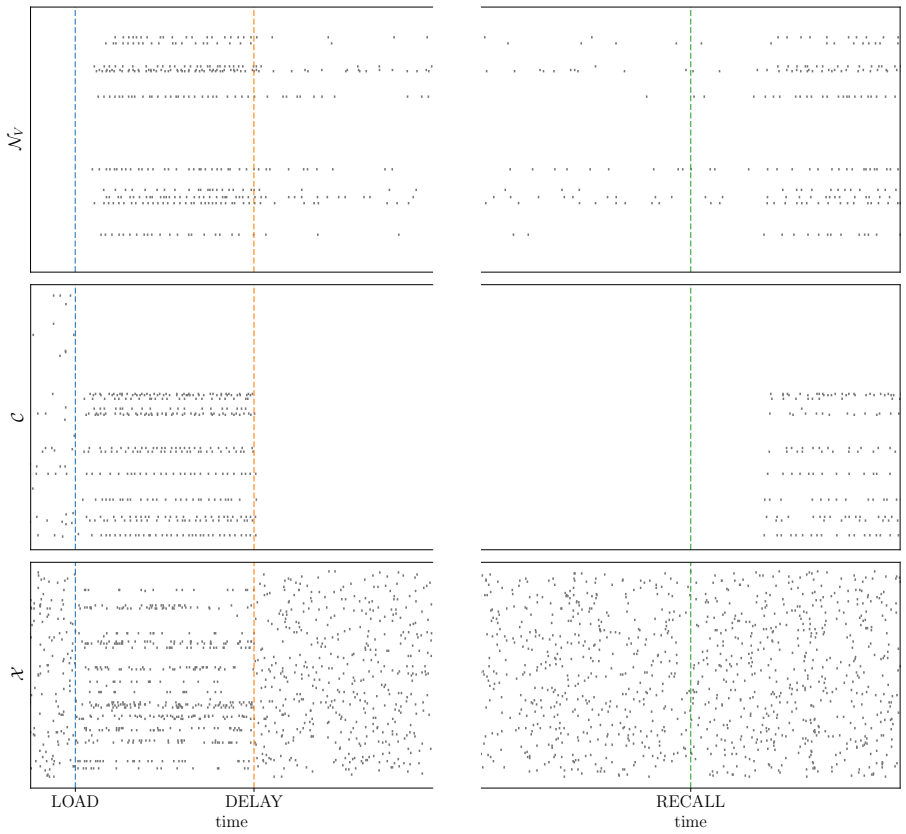

Figure 15: Spikes in input population $\mathcal{X}$, content space $\mathcal{C}$, and variable space $\mathcal{N}_V$ during a successfully LOAD / RECALL sequence. The beginning of each command is marked on the x-axis. The spikes of 200 neurons of each population are shown. For the neural spaces, the neurons were randomly selected; for the input population, this is the entire population. The time axis is drawn to scale, but is broken during the long delay period.

	used for	pat.	size in $\mathcal{C}$	shared	missing	excess	success
$\mathcal{C}_1$	training	1	86	86.2 ± 1.1	2.2 ± 1.2	0.2 ± 0.6	10/10
		2	54	53.0 ± 0.6	1.0 ± 0.6	0.2 ± 0.4	10/10
		3	80	77.5 ± 1.7	2.2 ± 1.5	0.2 ± 0.4	10/10
		4	56	54.7 ± 1.2	1.3 ± 1.2	0.0 ± 0.0	10/10
		5	52	50.2 ± 1.1	1.8 ± 1.1	0.0 ± 0.0	10/10
$\mathcal{C}_2$	training	1	66	63.2 ± 0.9	1.8 ± 0.9	0.0 ± 0.0	10/10
		2	54	52.3 ± 1.1	1.6 ± 1.1	0.1 ± 0.3	10/10
		3	87	82.7 ± 2.4	3.8 ± 2.4	0.2 ± 0.4	10/10
		4	51	49.8 ± 0.8	1.1 ± 0.8	0.2 ± 0.4	10/10
		5	79	74.2 ± 2.3	4.7 ± 2.3	0.0 ± 0.0	10/10
$\mathcal{C}_3$	validation	1	61	59.8 ± 1.0	1.1 ± 1.0	0.2 ± 0.4	10/10
		2	63	61.2 ± 1.7	1.7 ± 1.7	0.0 ± 0.0	10/10
		3	66	63.7 ± 1.2	2.2 ± 1.3	0.0 ± 0.0	10/10
		4	80	74.9 ± 2.2	4.4 ± 2.2	0.4 ± 0.4	10/10
		5	68	64.4 ± 1.9	3.6 ± 1.9	0.0 ± 0.0	10/10
$\mathcal{C}_4$	validation	1	40	38.7 ± 1.0	1.3 ± 1.1	2.7 ± 1.5	10/10
		2	93	91.7 ± 1.1	2.6 ± 1.2	1.6 ± 0.9	10/10
		3	64	62.2 ± 1.6	1.7 ± 1.6	0.0 ± 0.0	10/10
		4	41	49.2 ± 1.1	1.8 ± 1.1	0.0 ± 0.0	10/10
		5	60	56.8 ± 1.8	3.1 ± 1.8	0.0 ± 0.0	10/10
$\mathcal{C}_5$	validation	1	88	85.5 ± 1.6	2.5 ± 1.6	0.0 ± 0.0	10/10
		2	58	55.2 ± 1.2	2.7 ± 1.2	0.0 ± 0.0	10/10
		3	58	56.7 ± 0.7	1.2 ± 0.7	0.0 ± 0.0	10/10
		4	62	59.2 ± 1.6	2.7 ± 1.6	0.0 ± 0.0	10/10
		5	66	62.8 ± 1.6	3.1 ± 1.6	0.0 ± 0.0	10/10

Table 2: Results on the LOAD / RECALL task without using neuronal excitability on pre-trained content spaces instances used in training ($\mathcal{C}_1$ and $\mathcal{C}_2$) and for validation ($\mathcal{C}_3$ to $\mathcal{C}_5$). Each content space contains five patterns. For each of these, the following quantities are given: assembly size in $\mathcal{C}$ after training, shared, missing, and excess neurons $\mathcal{C}$ during RECALL (averaged over 10 independent trials), number of successful trials according to the 80% criterion (see text).

	used for	pat.	size in $\mathcal{C}$	shared	missing	excess	success
$\mathcal{C}_6$	test	1	65	63.8 ± 1.1	1.1 ± 1.1	0.0 ± 0.0	10/10
		2	66	64.2 ± 2.1	1.7 ± 2.1	0.0 ± 0.0	10/10
		3	61	59.2 ± 1.0	1.7 ± 1.1	0.0 ± 0.0	10/10
		4	73	68.9 ± 1.5	4.0 ± 1.5	0.0 ± 0.0	10/10
		5	81	77.0 ± 1.8	3.7 ± 1.9	0.1 ± 0.3	10/10
$\mathcal{C}_7$	test	1	47	45.0 ± 2.3	2.0 ± 2.3	0.5 ± 0.5	10/10
		2	64	62.2 ± 1.3	1.8 ± 1.3	0.0 ± 0.0	10/10
		3	76	73.4 ± 1.7	3.6 ± 1.7	0.0 ± 0.0	10/10
		4	82	77.2 ± 2.0	4.0 ± 1.8	0.5 ± 0.6	10/10
		5	80	75.2 ± 1.6	4.7 ± 1.6	0.0 ± 0.0	10/10
$\mathcal{C}_8$	test	1	55	54.2 ± 1.0	0.6 ± 1.0	0.4 ± 0.6	10/10
		2	53	51.3 ± 1.4	1.6 ± 1.4	0.1 ± 0.3	10/10
		3	74	71.2 ± 1.7	2.6 ± 1.6	0.2 ± 0.6	10/10
		4	86	79.5 ± 2.3	5.9 ± 2.2	1.3 ± 1.3	10/10
		5	76	71.4 ± 1.9	4.5 ± 1.9	0.0 ± 0.0	10/10
$\mathcal{C}_9$	test	1	90	86.7 ± 1.4	3.0 ± 1.7	0.2 ± 0.4	10/10
		2	72	69.5 ± 1.9	2.2 ± 1.9	0.1 ± 0.3	10/10
		3	54	52.8 ± 1.0	1.1 ± 1.0	0.1 ± 0.3	10/10
		4	75	71.9 ± 1.8	2.8 ± 1.9	0.2 ± 0.4	10/10
		5	55	52.2 ± 1.2	2.7 ± 1.2	0.0 ± 0.0	10/10
$\mathcal{C}_{10}$	test	1	59	58.2 ± 0.7	0.8 ± 0.7	0.0 ± 0.0	10/10
		2	46	44.2 ± 1.3	1.7 ± 1.3	0.8 ± 0.9	10/10
		3	78	75.2 ± 0.8	3.3 ± 1.9	0.2 ± 0.4	10/10
		4	65	63.1 ± 1.7	1.8 ± 1.7	0.0 ± 0.0	10/10
		5	81	77.9 ± 1.6	4.0 ± 1.6	0.0 ± 0.0	10/10

Table 3: Results on the LOAD / RECALL task without using neuronal excitability on pre-trained content spaces instances used as generalization test. (See caption of Table 2 for details.)

First, we measure the success rate on the two content space instances which were used during training and the three other instances used for validation (Table 2). On each of these, each pattern is tested 10 times. All trials were successful. Generally, a small number of neurons is missing in $\mathcal{C}$ during the RECALL, but very few neurons are active which were not part of the assembly before.

To test the generalization of these results, we conduct the same test on five new randomly generated content space instances (Table 3). On all of these, the results are similar to above, and no trial failed. This shows that the parameters are highly robust.

4.3.2 LOAD / COPY / RECALL

We move on to the second task, in which an additional variable space $\mathcal{N}_U$ is introduced. The task is to copy the content of $\mathcal{N}_V$ after a LOAD operation to another neural space, and then perform a successful RECALL from there. We use the parameters determined above for the extended circuit, thus, this experiment is a further test of how suitable they are for more elaborate binding operations.

Procedure The preparations for the experiment are similar to those in the previous one. Again, we use pre-trained content space instances. Assemblies for each content pattern are formed in both variable spaces $\mathcal{N}_V$ and $\mathcal{N}_U$ by performing CREATE operations.

The experiment then takes place as follows.

- **LOAD.** We begin by performing a LOAD from the content space $\mathcal{C}$ into the first variable space $\mathcal{N}_V$. To do so, the content space is driven by the target pattern from the input population $\mathcal{X}$, and the assembly which has formed there during the training phase becomes active. Simultaneously, the corresponding assembly in $\mathcal{N}_V$ also is activated. This phase lasts for 200 ms, after which the input neurons in $\mathcal{X}$ fire randomly for the remaining duration of the experiment. The other variable space $\mathcal{N}_U$ remains inhibited here.

- **DELAY.** The content space is inhibited for 400 ms, while $\mathcal{N}_V$ remains active.

- **COPY.** This operation takes place in multiple stages. For the first 50 ms, nothing is changed.[7] Then, the content space is disinhibited and the activity in it is restored as $\mathcal{N}_V$ and $\mathcal{C}$ interact. Then, for 100 ms, the other variable space $\mathcal{N}_U$ is disinhibited, and the assembly corresponding to the input pattern becomes active there.

[7] See the description of the previous experiment.

- **DELAY.** The first variable space $\mathcal{N}_V$ and the content space are inhibited. This time, $\mathcal{N}_U$ may remain active. This phase again lasts for 400 ms.
- **RECALL.** Finally, we perform a regular RECALL operation from $\mathcal{N}_U$ to $\mathcal{C}$. During the latter 100 ms of this phase, the activity in $\mathcal{C}$ is measured and used as to assess the performance.

The success is measured by comparing the neurons active in the content space during the second half of the final RECALL operation with those which were active when testing the patterns after training. The same success criterion as for the previous experiment is used.

Results A typical run can be seen in Figure 16. Here, we can see that the procedure is successful, and the activity in $\mathcal{C}$ after the RECALL operation closely resembles the earlier activity.

We again perform tests on the same 10 content space instances for the previous experiment. For each instance of $\mathcal{C}$, each pattern is tested twice. To measure the success rate, we use the same criterion as above (if at least 80% of the neurons of the original assembly in $\mathcal{C}$ are active during the RECALL phase, and if the number of additionally active neurons does not exceed 20%, the performance is regarded as success). This yields the following results: On the content space instances 1-5 (which were used for training and validation of the parameters, see above), each COPY operation is successful. The generalization test on instances 6-10 shows similar results: of the total number of 50 tests on these content space instances, a single run fails. This shows that the parameters found above allow robust execution of the COPY operation.

4.4 Experiments using neuronal excitability

We now turn to using an adaptive neuronal excitability with may change over time as used in [9]. Previously, the variable spaces were not inhibited during the delay periods within each experiment, and thus, the information could be stored in their persistent activity, maintained by the recurrent connections within the pool of excitatory neurons. For the following experiments, we will prevent this by also inhibiting the variable spaces during delay periods. Since the information must be stored somewhere, we now need to use adaptive neuronal excitabilities (see Appendices A-C, in particular (73) and (74) for implementation details). These increase when neurons fire and slowly decay over time, leading to an increased excitability of the previously active neurons after short delay periods.

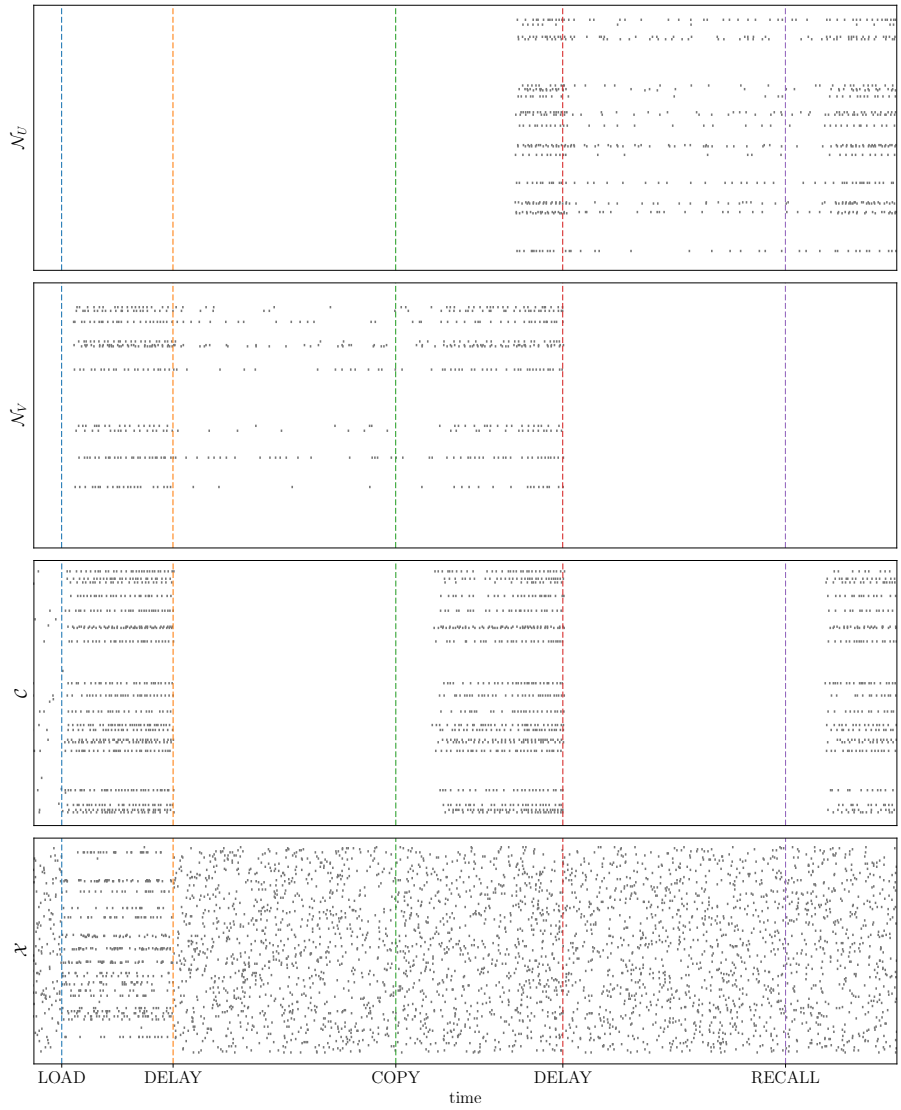

Figure 16: Spikes in input population $\mathcal{X}$, content space $\mathcal{C}$, and variable spaces $\mathcal{N}_V$ and $\mathcal{N}_U$ during a successfully LOAD / COPY / RECALL sequence. The beginning of each command is marked on the x-axis. The spikes of 200 neurons of each population are shown. For the three neural spaces, the neurons were randomly selected; for the input population, this is the entire population. The time axis is drawn to scale.

4.4.1 LOAD / RECALL

First, we investigate the simple case of a LOAD/RECALL sequence.

Procedure The procedure of this experiment is identical to the one described above in Section 4.3.1, except that during the delay period, not only the content space is inhibited, but also the variable space $\mathcal{N}_V$. Thus, the information about the stored pattern needs to be stored in the adaptive biases, which lead to an increased excitability of the previously active neurons after the variable space is disinhibited. As above, no adaptive bias is used for the content space.

During the recall phase, the content space remains inhibited for the first 50 ms. This allows the variable space to establish some activity through its recurrent connections before driving the content space back into the original regime.[8]

Since the inhibition of neural spaces is modeled in a soft way (injection of current into each neuron), a large adaptive bias could lead to firing of the neurons even if their neural space is in the inhibited state. Thus, these biases are clipped at some value (see Appendix C.2). Also, the values of the adaptive biases are reset between different experiments, which is equivalent to waiting for some period of time between different runs.

Parameter Selection We start with the parameters used above in the experiments without neuronal excitability. First, a good value for the additive quantity for the adaptive bias q_{sfa} was found by performing some runs with the final setup and the parameters used above. After that, the same optimization scheme as above was used to achieve a good performance on all validation instances of the content space. The cost function for the optimization as the same as above. Only some fine-tuning was necessary. The final parameters are given in Appendix C.2 in Tables 10 and 11. Finally, the clipping value for the adaptive bias was set in a rather strict way to avoid the spilling of activity of neurons within the variable spaces into the delay periods.

Results A typical run is depicted in Figure 17. As can be seen, the activity is similar to the one shown in Section 4.3.1, but now, the variable space $\mathcal{N}_V$ is silent during the delay period. Some minor spilling of activity into this phase can occur due to the soft way in which the inhibition takes place, however, this lasts only for a short period.

In the example run which is shown here, the activity in $\mathcal{C}$ is restored reliably. We again evaluate the performance on the same content space

[8] This timing scheme was also followed for simplicity reasons in the previous experiments where no excitability was used. There, this is not necessary, but poses a slightly more difficult problem, since the content space then has less time to restore its activity.

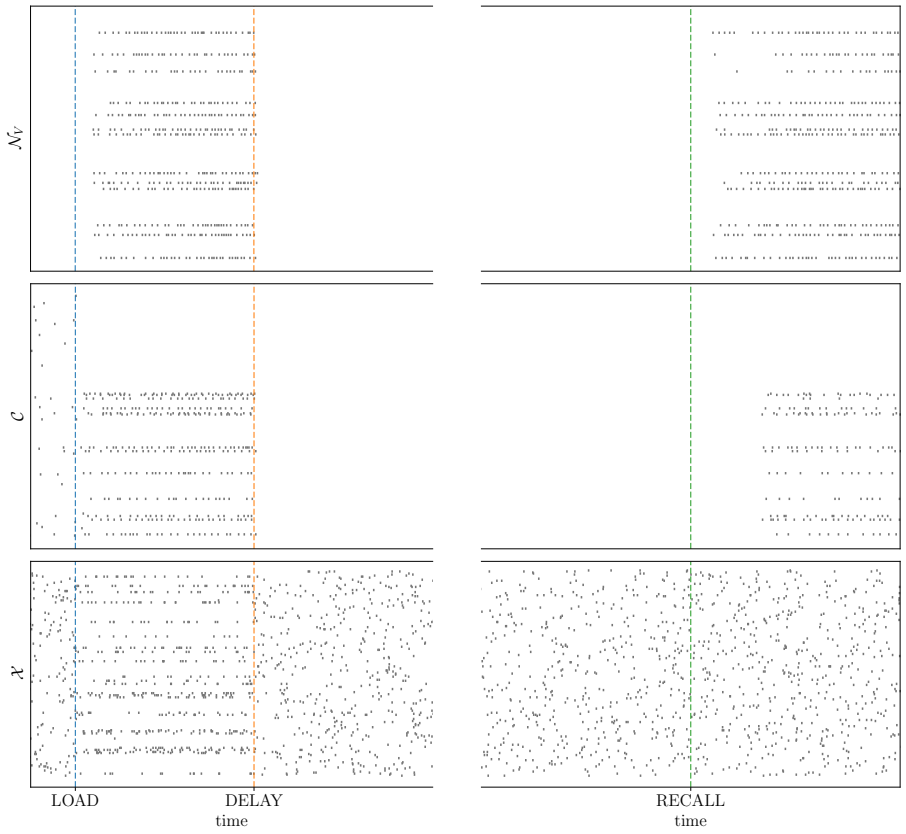

Figure 17: Spikes in input population $\mathcal{X}$, content space $\mathcal{C}$, and variable space $\mathcal{N}_V$ during a successfully LOAD / RECALL sequence making use of neuronal excitability in the variable space. The beginning of each command is marked on the x-axis. The spikes of 200 neurons of each population are shown; for the neural spaces, the neurons were randomly selected. For the input population, this is the entire population. The time axis is drawn to scale, but is broken during the long delay period.

	used for	pat.	size in $\mathcal{C}$	shared	missing	excess	success
$\mathcal{C}_1$	training	1	86	85.4 ± 1.2	3.2 ± 0.9	0.2 ± 0.6	10/10
		2	54	52.3 ± 1.2	1.6 ± 1.2	0.0 ± 0.0	10/10
		3	80	77.9 ± 1.2	2.1 ± 1.2	0.0 ± 0.0	10/10
		4	56	54.7 ± 1.0	1.2 ± 1.0	0.0 ± 0.0	10/10
		5	52	51.1 ± 0.9	0.9 ± 0.9	0.0 ± 0.0	10/10
$\mathcal{C}_2$	training	1	66	64.9 ± 0.8	1.1 ± 0.8	0.0 ± 0.0	10/10
		2	54	53.2 ± 0.9	0.6 ± 0.9	0.1 ± 0.3	10/10
		3	87	82.7 ± 2.3	4.2 ± 2.3	0.0 ± 0.0	10/10
		4	51	50.6 ± 0.6	0.4 ± 0.6	0.0 ± 0.0	10/10
		5	79	75.4 ± 2.3	3.3 ± 2.2	0.2 ± 0.4	10/10
$\mathcal{C}_2$	validation	1	61	60.2 ± 0.5	0.8 ± 0.6	0.0 ± 0.0	10/10
		2	63	61.5 ± 0.8	1.5 ± 0.8	0.0 ± 0.0	10/10
		3	66	64.2 ± 1.0	1.6 ± 0.9	0.1 ± 0.3	10/10
		4	80	76.4 ± 1.2	3.1 ± 1.3	0.4 ± 0.6	10/10
		5	68	66.2 ± 1.3	1.7 ± 1.3	0.0 ± 0.0	10/10
$\mathcal{C}_4$	validation	1	40	39.1 ± 0.5	0.9 ± 0.5	0.5 ± 0.5	10/10
		2	93	90.2 ± 2.1	4.0 ± 1.8	1.7 ± 0.9	10/10
		3	64	62.2 ± 1.2	1.8 ± 1.2	0.0 ± 0.0	10/10
		4	41	49.8 ± 1.0	1.1 ± 1.0	0.0 ± 0.0	10/10
		5	60	58.5 ± 1.2	1.5 ± 1.2	0.0 ± 0.0	10/10
$\mathcal{C}_5$	validation	1	88	84.7 ± 1.4	3.0 ± 1.4	0.2 ± 0.4	10/10
		2	58	56.7 ± 1.2	1.2 ± 1.2	0.0 ± 0.0	10/10
		3	58	56.5 ± 1.0	1.5 ± 1.0	0.0 ± 0.0	10/10
		4	62	60.2 ± 0.9	1.7 ± 0.9	0.0 ± 0.0	10/10
		5	66	64.0 ± 0.8	1.8 ± 1.1	0.2 ± 0.4	10/10

Table 4: Results on the LOAD / RECALL task using neuronal excitability on pre-trained content spaces instances used in training ($\mathcal{C}_1$ and $\mathcal{C}_2$) and for validation ($\mathcal{C}_3$ to $\mathcal{C}_5$). Each content space contains five patterns. For each of these, the following quantities are given: assembly size in $\mathcal{C}$ after training, shared, missing, and excess neurons $\mathcal{C}$ during RECALL (averaged over 10 independent trials), number of successful trials according to the 80% criterion (see text).

	used for	pat.	size in $\mathcal{C}$	shared	missing	excess	success
$\mathcal{C}_6$	test	1	65	64.2 ± 0.7	0.6 ± 0.7	0.1 ± 0.3	10/10
		2	66	64.2 ± 1.1	1.7 ± 1.1	0.0 ± 0.0	10/10
		3	61	59.7 ± 0.8	1.2 ± 0.8	0.0 ± 0.0	10/10
		4	73	70.0 ± 1.7	2.8 ± 1.7	0.0 ± 0.0	10/10
		5	81	77.2 ± 2.4	3.7 ± 2.4	0.0 ± 0.0	10/10
$\mathcal{C}_7$	test	1	47	46.2 ± 0.8	0.8 ± 0.8	0.1 ± 0.3	10/10
		2	64	61.7 ± 0.6	2.2 ± 0.6	0.0 ± 0.0	10/10
		3	76	74.7 ± 1.2	2.1 ± 1.3	0.1 ± 0.3	10/10
		4	82	78.9 ± 2.0	2.8 ± 1.9	0.0 ± 0.0	10/10
		5	80	76.9 ± 1.3	3.0 ± 1.4	0.1 ± 0.3	10/10
$\mathcal{C}_8$	test	1	55	53.7 ± 0.7	1.3 ± 0.7	0.0 ± 0.0	10/10
		2	53	52.1 ± 0.7	0.9 ± 0.7	0.0 ± 0.0	10/10
		3	74	70.9 ± 0.9	3.0 ± 1.0	0.1 ± 0.3	10/10
		4	86	81.2 ± 1.7	4.7 ± 1.9	0.9 ± 1.2	10/10
		5	76	72.0 ± 1.7	3.7 ± 1.3	0.2 ± 0.6	10/10
$\mathcal{C}_9$	test	1	90	80.2 ± 17.8	9.5 ± 17.9	6.0 ± 17.3	9/10
		2	72	70.2 ± 1.4	1.8 ± 1.4	0.0 ± 0.0	10/10
		3	54	52.7 ± 1.3	1.2 ± 1.3	0.0 ± 0.0	10/10
		4	75	71.5 ± 2.3	3.3 ± 2.3	1.8 ± 5.7	9/10
		5	55	52.8 ± 1.5	2.1 ± 1.5	0.0 ± 0.0	10/10
$\mathcal{C}_{10}$	test	1	59	58.5 ± 0.6	0.5 ± 0.6	0.1 ± 0.3	10/10
		2	46	45.6 ± 0.4	0.4 ± 0.4	0.0 ± 0.0	10/10
		3	78	75.9 ± 1.1	2.8 ± 1.0	0.2 ± 0.4	10/10
		4	65	63.5 ± 1.2	1.5 ± 1.2	0.0 ± 0.0	10/10
		5	81	78.4 ± 2.1	3.3 ± 2.1	0.1 ± 0.3	10/10

Table 5: Results on the LOAD / RECALL task using neuronal excitability on pre-trained content spaces instances used as generalization test. (See caption of Table 4 for details.)

instances as in the first experiment where $\mathcal{N}_V$ was not inhibited during the delay period. The results are shown in Tables 4 and 5. As can be seen, the procedure works very reliably across many different content space instances, with only two unsuccessful runs.

4.4.2 LOAD / COPY / RECALL

Naturally, we repeat the second experiment with the parameters obtained in the previous experiment to see if they are also robust in more elaborate operation sequences.

Procedure The experimental procedure closely follows the one described above (cf. Section 4.3.2) where the variable spaces are not inhibited during delay periods. We recap and expand it here. We assume that the content space has been trained and all assemblies in both variables spaces have been formed with the CREATE operation (as described above). The procedure then is as follows.

- **LOAD.** First, a pattern is loaded from $\mathcal{C}$ into the first variable space $\mathcal{N}_V$. This is done by presenting the pattern to the content space by the input population $\mathcal{X}$ while $\mathcal{N}_V$ is disinhibited. This phase lasts for 200 ms. Afterwards, the input population fires randomly for the rest of the experiment.
- **DELAY.** All neural spaces are inhibited for 400 ms.
- **COPY.** This operation consists of multiple stages. First, the variable space $\mathcal{N}_V$ is disinhibited and the activity in it is restored by the increased excitability of the previously active neurons. After 50 ms, the content space is also disinhibited, and for 150 ms, the activity there may settle. Up to this point, this corresponds to a regular RECALL operation. Then, for 100 ms, we additionally disinhibit the other variable space $\mathcal{N}_U$.
- **DELAY.** All neural spaces are inhibited again for 400 ms.
- **RECALL.** We perform a regular RECALL operation from $\mathcal{N}_U$ to $\mathcal{C}$. For the first 50 ms, only the variable space is disinhibited, after which also the content space is additionally activated for 150 ms. During the latter 100 ms of this phase, the activity in $\mathcal{C}$ is measured and used as a comparison measure.

As can be seen, this procedure closely resembles that of the case without neuronal excitability described above, except for the fact that the variable spaces are also inhibited during delay periods.

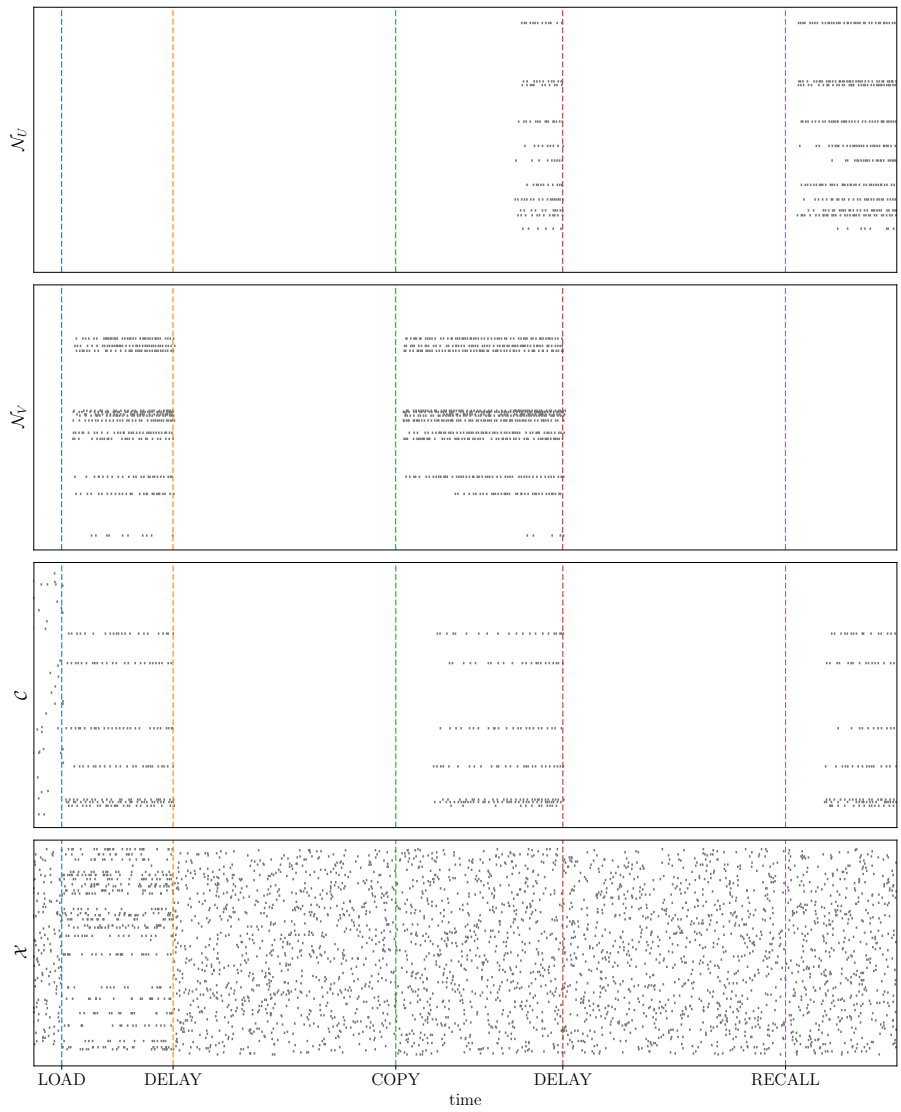

Figure 18: Spikes in input population $\mathcal{X}$, content space $\mathcal{C}$, and variable spaces $\mathcal{N}_V$ and $\mathcal{N}_U$ during a successfully LOAD / COPY / RECALL sequence making use of neuronal excitability in the variable spaces. The beginning of each command is marked on the x-axis. The spikes of 200 neurons of each population are shown; for the three neural spaces, the neurons were randomly selected. For the input population, this is the entire population. The time axis is drawn to scale.

Results Figure 18 shows the spiking activity of a typical run of this experiment. Similar to the previous experiment, some activity of the variable spaces can spill into the delay periods, but it is not prolonged. Here, we can see that after the RECALL phase, the activity in the content space closely resembles that of the first LOAD phase in terms of high firing rates.

We test this procedure (as above) by performing two copy operations for each of the five patterns on all ten content space instances. All of these tests succeed in activating the same assembly during the RECALL phase as during the testing of the content space. This indicates high robustness of the parameters to all sources of randomness, in particular the stochastic connectivity between neural spaces and within them.

5 Efficient Optimization of High-Dimensional Spaces with Constraints

This section describes the optimization procedure which was used to find the parameters of the variable binding model (described in the previous chapter).

5.1 Motivation

The variable binding model described above has a large number of parameters. While some were determined by constraints or could be set to reasonable values a priori, 23 parameters remained free. These parameters needed to be optimized to ensure good performance of the model. At the same time, there are constraints on many neuron parameters to maintain biological plausibility, e.g. time constants of STDP learning curves should not be to small (say, less than 5 ms) and also not too large (e.g. 50 ms). These factors posed a 23-dimensional optimization problem with constraints where the cost function is non-differentiable.

To tackle this problem, a Differential Evolution (DE) [52] optimizer was initially utilized. This stochastic optimization algorithm uses a population of candidates which are mutated and recombined in each iteration. The implementation of the algorithm used the Latin Hypercube sampling (LHS) scheme [53] to randomly place the population in the allow space in the first iteration.

It was observed that during this first iteration (involving calculating the cost for a population of candidates), somewhat good results were achieved. The reason for this is that LHS generates samples which are random but nevertheless are distributed over the allowed parameter space to cover a large portion of it. Thus, by chance, generally good regions of the value space were found. All subsequent iterations of DE, however, failed to produce any advancement in terms of the resulting cost values. This is likely due to the method of proposing new candidates deployed by DE, which uses recombination, and thus is prone to setting parameters to unbeneficial values which lie between two good (possibly equally acceptable) local minima.

Under the assumption that a good local minimum produces a sufficiently low cost value, performing local updates will lead to the solution when a good region is found. We thus suggest a different optimization scheme in which the magnitude of proposed updates is decreased from global to very local over a fixed number of iterations. The goal is to achieve a good optimization performance on high-dimensional optimization problems which are locally convex while using as few hyperparameters as possible. This allows for the

application of the algorithm to a broad range of optimization problems while requiring little setup effort.

5.2 Algorithm

We assume a high-dimensional error surface which is not convex but has distinct local minima with acceptable cost values and furthermore has upper and lower bounds for each dimension. Thus, the optimization should consist of two steps: first, a promising region of the high-dimensional space must be found; then, local optimization should be applied to reach the nearest local minimum. Since we assume an error landscape which is generally non-convex, using gradient descent or approximations of the gradient will not produce good results for the first phase. We therefore use stochastic updates.

Since the goal is to have as few hyperparameters as possible, we do not make a clear distinction between the two phases. Instead, we propose stochastic updates relative to the current position which are drawn from a uniform distribution with a maximum step size σ which decays over time. The number of iterations N (which is equivalent to the number of function calls) must be set by the user. In the beginning, updates should span (more or less) the entire range for each dimension, leading to a search for a good region of the parameter space. Here, we have $\sigma = \sigma_0 = 1$ for updates spanning the full range. As the number of iterations approaches the maximum value N, the step size is decreased further and further towards a given minimum value $\bar{\sigma}$ for the last iteration, and the current region is locally optimized to reach the nearest local minimum. In each step, the allow range for a parameter update must stay within the bounds, so the upper and lower bounds of the uniform distribution (determined by σ) are clipped if necessary.

Furthermore, we assume that it is beneficial not to update all parameters at once. Thus, we draw a random update mask from a binomial distribution with parameter p_m in each iteration. The value of p_m must also be set by the user.

The algorithm then works as follows. We assume that an initial value $\mathbf{x}_0$ is given, as well as upper and lower bounds for each dimension ($\mathbf{x}_\mathrm{max}$ and $\mathbf{x}_\mathrm{min}$, respectively). The optimization target is the function f. We perform one function call to determine the value of f for $\mathbf{x}_0$.

The step size σ linearly decays each iteration. Given N, the step size at some iteration n is

$$\sigma_n = \sigma_0 - (\sigma_0 - \bar{\sigma}) \cdot \frac{n-1}{N-2} \,, \tag{18}$$

where σ_0 is the initial step size (usually set to $\sigma_0 = 1$ for updates to on average cover the entire value range), and $\bar{\sigma}$ is the final value for the step size during the final iteration (the default of $\bar{\sigma} = 0.001$ is a reasonable

choice in many cases). A random mask for the update targets is generated according to

$$\text{draw } \mathbf{m} \sim \textsc{Binomial}^D(p_\text{m}) \tag{19}$$

which represents D independent draws from a binomial distribution with probability p_m. It should be verified that at least one element in $\mathbf{m}$ is nonzero. Furthermore, update proposals are generated for each dimension which are drawn from a D-dimensional uniform distribution and clipped appropriately:

$$\mathbf{u}_\text{min} \leftarrow \max\left(\mathbf{x}_\text{min}, \mathbf{x}_n - \tfrac{1}{2}\sigma_n\left(\mathbf{x}_\text{max} - \mathbf{x}_\text{min}\right)\right) \tag{20}$$

$$\mathbf{u}_\text{max} \leftarrow \min\left(\mathbf{x}_\text{max}, \mathbf{x}_n + \tfrac{1}{2}\sigma_n\left(\mathbf{x}_\text{max} - \mathbf{x}_\text{min}\right)\right) \tag{21}$$

$$\text{draw } \mathbf{r} \sim \mathcal{U}^D(0,1) \tag{22}$$

$$\mathbf{u} \leftarrow \mathbf{u}_\text{min} + \left(\mathbf{u}_\text{max} - \mathbf{u}_\text{min}\right) \circ \mathbf{r} \tag{23}$$

Here, $\circ$ denotes the Hadamard product (element-wise multiplication), and the min and max functions are applied element-wise to the two given vectors[9]. Since the allowed range is clipped to the bounds for each dimension, the updates are biased towards the center of the allowed interval at the beginning of the procedure. As the step size decreases further and further, this effect diminishes. Next, we generate the candidate $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} \leftarrow (1 - \mathbf{m}) \circ \mathbf{x}_{n-1} + \mathbf{m} \circ \mathbf{u} \tag{24}$$

To complete an iteration, we perform a greedy update of $\mathbf{x}$. If the cost of $\hat{\mathbf{x}}$ is smaller than that of $\mathbf{x}$, the former replaces the latter in the next iteration. Otherwise, $\mathbf{x}$ is kept.

The entire procedure is given in Algorithm 1. As can be seen, there are only four hyperparameters: N, σ_0, $\bar{\sigma}$, and p_m. Good default values for σ_0 and $\bar{\sigma}$ have been given above. Also, we argue that using $p_\text{m} = 0.5$ is a reasonable choice for high-dimensional spaces. Thus, it is sufficient to choose the number of iterations if global optimization should be performed. In case a good proposal value $\mathbf{x}_0$ exists, the algorithm can be run with a smaller value of σ_0 to only perform local optimization.

Due to the fact that updates are drawn from a uniform distribution we name this algorithm $\mathcal{U}$-Decay.

[9] i.e. $\min(\mathbf{a}, \mathbf{b}) = \mathbf{c}$ with elements $c_i = \min(a_i, b_i)\ \forall i$.

Algorithm 1: $\mathcal{U}$-Decay

input : $\mathbf{x}_0, \mathbf{x}_{\min}, \mathbf{x}_{\max} \in \mathbb{R}^D$,
$0 < \sigma_0 \leq 1,\ 0 < \bar{\sigma} \leq 1$,
$0 < p_{\mathrm{m}} < 1$,
N

output: $\mathbf{x}_{n-1}, f_{n-1}$

$f_0 \leftarrow f(\mathbf{x}_0)$; `// evaluate cost of initial value`

$n \leftarrow 1$;

while $n < N$ **do**

  $\sigma_n \leftarrow \sigma_0 - (\sigma_0 - \bar{\sigma}) \cdot \frac{n-1}{N-2}$; `// current step size`

  $\mathbf{u}_{\min} \leftarrow \max\left(\mathbf{x}_{\min}, \mathbf{x}_n - \frac{1}{2}\sigma_n\left(\mathbf{x}_{\max} - \mathbf{x}_{\min}\right)\right)$; `// lower bound`

  $\mathbf{u}_{\max} \leftarrow \min\left(\mathbf{x}_{\max}, \mathbf{x}_n + \frac{1}{2}\sigma_n\left(\mathbf{x}_{\max} - \mathbf{x}_{\min}\right)\right)$; `// upper bound`

  draw $\mathbf{r} \sim \mathcal{U}^D(0,1)$;

  $\mathbf{u} \leftarrow \mathbf{u}_{\min} + (\mathbf{u}_{\max} - \mathbf{u}_{\min}) \circ \mathbf{r}$;

  repeat

    draw $\mathbf{m} \sim \textsc{Binomial}^D(p_{\mathrm{m}})$; `// draw random update mask`

  until $\|\mathbf{m}\| > 0$;

  $\hat{\mathbf{x}} \leftarrow (1 - \mathbf{m}) \circ \mathbf{x}_{n-1} + \mathbf{m} \circ \mathbf{u}$; `// create candidate`

  $\hat{f} \leftarrow f(\hat{\mathbf{x}})$; `// evaluate candidate`

  if $\hat{f} < f_n$ **then**

    $f_n \leftarrow \hat{f}$;

    $\mathbf{x}_n \leftarrow \hat{\mathbf{x}}$;

  else

    $f_n \leftarrow f_{n-1}$;

    $\mathbf{x}_n \leftarrow \mathbf{x}_{n-1}$;

  end

  $n \leftarrow n + 1$;

end

5.3 Comparison to Other Optimization Algorithms

The proposed algorithm uses stochastic updates and does not rely on any first- and second-order information (no approximations of the gradient or the Hessian are computed). This provides a clear distinction from algorithms which propose deterministic updates (e.g. the Simplex method [54]) as well as from first- and second-order methods (e.g. Gradient Descent with Finite Differences, Quasi-Newton methods like L-BFGS-B [55, 56]). These have been shown to work well on convex problems, but often have difficulty on complex error landscapes.

Furthermore, only a single candidate is kept at all times, in contrast to algorithms which maintain a population (e.g. Particle Swarm Optimization [57], Natural Evolution Strategies [58]). This may lead to faster convergence on simple problems with many acceptable local minima since the number of evaluations of the cost function is smaller.

Simulated Annealing (SA) [59] is a popular stochastic global optimization algorithm which shows the closest resemblance to $\mathcal{U}$-Decay. There are two major differences: first, SA does not perform greedy updates, but uses a decaying temperature as acceptance criterion. While it can be shown theoretically that SA can thus find the global optimum (given enough time), this may hamper the results when few iterations are used (and local minima are sufficient). Furthermore, no exact scheme for drawing updates is defined by SA. $\mathcal{U}$-decay does not require accepting updates which increase the cost since it samples from the entire space at the beginning of the optimization run, and its greedy update scheme ensures that the algorithm halts at the best value encountered. The main benefit over SA is that an update scheme is defined by the algorithm with reasonable default values which should produce good results on a broad range of problems.

5.4 Results on Optimization Test Functions

We consider four optimization test functions [60] in D dimensions (i.e. each test function f is $f : \mathbb{R}^D \to \mathbb{R}$) with input vectors $\mathbf{x} = [x_1, x_2, ..., x_D]^T$. Each function poses a different challenge to the optimization algorithms in terms of the shape of the error landscape. The first one is the Rosenbrock function

$$f(\mathbf{x}) = \sum_{i=1}^{D-1} \left[(1 - x_i)^2 + 100 \cdot \left(x_{i+1} - x_i^2 \right)^2 \right] \tag{25}$$

where we constrain all components to $x_i \in [-2, 2]\ \forall i$. The Rosenbrock function has its global minimum at $x_i = 1\ \forall i$. The error landscape has a single, global valley, which is relatively easy to find, while finding the absolute minimum within this valley is not trivial.

The second function is the Rastrigin function

$$f(\mathbf{x}) = 10 \cdot D + \sum_{i=1}^{D} \left[x_i^2 - 10 \cdot \cos(2\pi x_i) \right] \tag{26}$$

where we restrict each x_i to $[-5, 5]$. It has a global minimum at $x_i = 0 \; \forall i$. The error surface is characterized by an overall convex form with many rather deep local minima.

The next function is Ackley's function

$$f(\mathbf{x}) = -a \cdot \exp\left(-b \cdot \sqrt{\frac{1}{n} \sum_{i=1}^{D} x_i^2} \right) - \exp\left(\frac{1}{n} \sum_{i=1}^{D} \cos(c \cdot x_i) \right) + a - e \tag{27}$$

with $a = 20, \; b = \frac{1}{5}, \; c = 2\pi$. Here, where we confine each x_i to the range $[-30, 30]$. Ackley's function has its global minimum at $x_i = 0 \; \forall i$. Generally, the surface is flat, but shows a large number of very small local minima and thus appears to be noisy.

Finally, we investigate the Chasm function

$$f(\mathbf{x}) = \frac{1000 \cdot |x_1|}{1000 \cdot |x_1| + 1} + 0.01 \cdot \sum_{i=2}^{D} |x_i| \tag{28}$$

in $[-5, 5]$, which poses a very difficult problem. The global minimum lies at $x_i = 0 \; \forall i$. This function is extremely flat – thus providing hardly any gradient information – except for a single, steep chasm which gives the function its name.

All four test functions are shown in two dimensions within their specific ranges in Figure 19.

Figure 20 shows an illustrating example of the $\mathcal{U}$-Decay algorithm on the Rosenbrock function. The development of the cost over time is shown as well as the position on the error surface (top left and right). Furthermore, the shrinking range of allowed parameter updates is illustrated for both dimensions (bottom left and right).

We now compare the performance of the proposed algorithm to four other optimization schemes: Simulated Annealing (SA) in two variants (one performing global update steps over the entire value space, the other performing local update steps in a small radius around the current value) as the closest related algorithm, Differential Evolution (DE) as an algorithm using a larger population, and L-BFGS-B as an algorithm which relies on an approximated gradient. The implementations of L-BFGS-B and DE were provided by SciPy [61]. L-BFGS-B is set to approximate the gradient itself, no higher-order information is given by the test functions. We test each

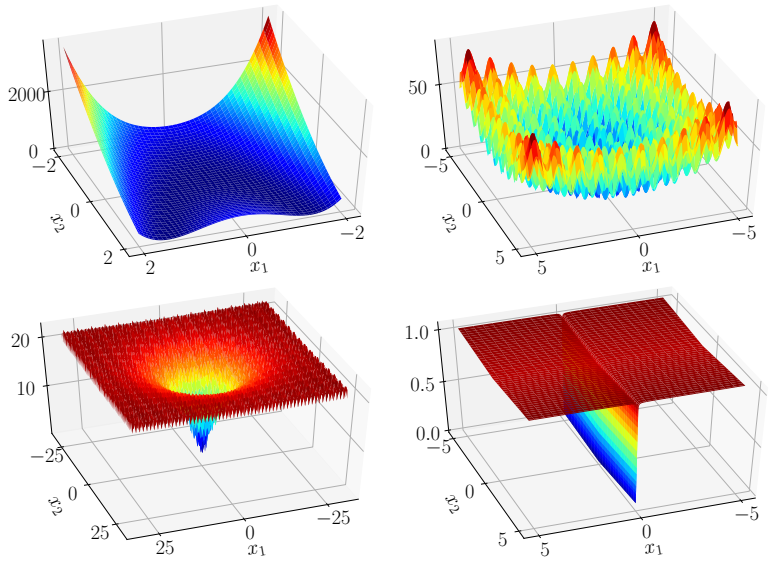

Figure 19: Error surfaces of the Rosenbrock function (top left), the Rastrigin function (top right), Ackley's function (bottom left), and the Chasm function (bottom right) in 2D within the ranges given in the text.

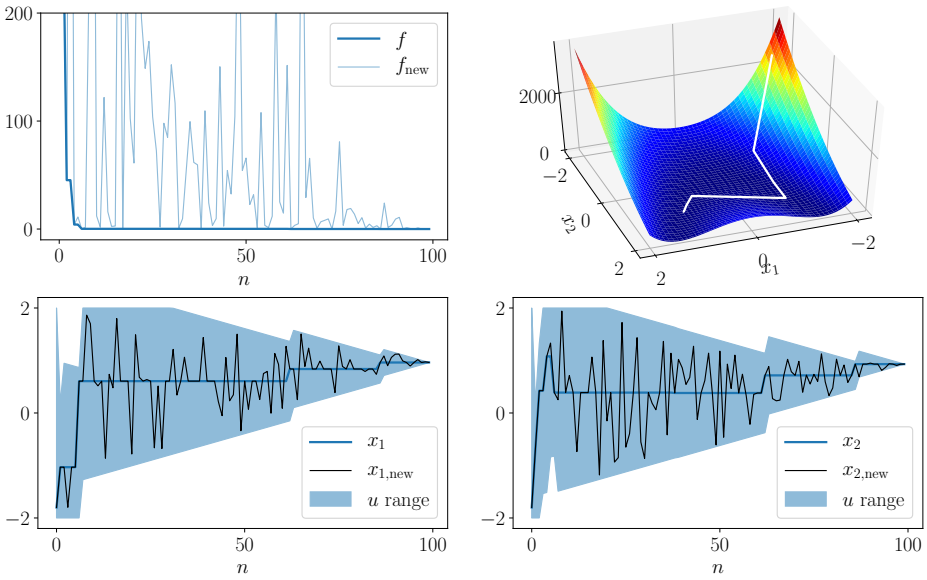

Figure 20: Example optimization run on the Rosenbrock function. Top left: cost value and value of proposed update over iteration. Top right: path on error surface taken during optimization. Bottom left and right: value of coordinate and proposed value for coordinate within the allow range for x and y, respectively.

algorithm on each test function using 100 random initializations. DE uses a Latin Hypercube Sampler to create its initial population. We then run the algorithms until 100 function calls have been performed. (Note that the implementation of L-BFGS-B does not necessarily comply to that bound, and often the number of function calls here exceeded 100.) All parameters for these tests are given in Appendix D in Table 12.

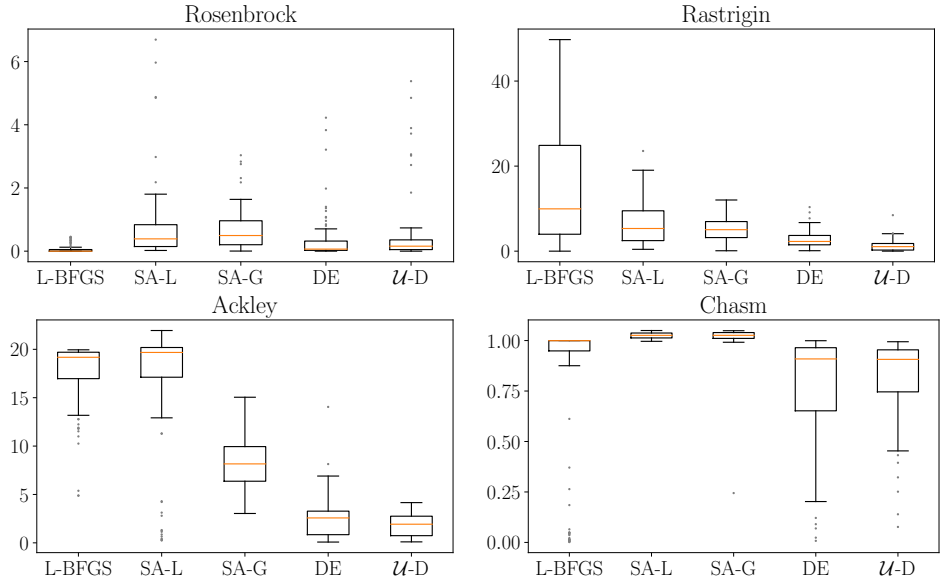

Figure 21: Results for optimization in 2 dimensions. Depicted is a boxplot of best cost value of 100 optimization iterations (or 100 function calls in case more than one call may occur per iteration) on four test functions. On each, 100 different runs with random initialization were performed. The box extends from the lower to upper quartile values of the data, the line within it shows the median. The whiskers extend to the most extreme data point within 1.5 times the interquartile range. The algorithms are (from left to right) L-BFGS-B, Simulated Annealing with local updates, Simulated Annealing with global updates, Differential Evolution, and $\mathcal{U}$-Decay. Note that the implementation of L-BFGS does not allow imposing a hard upper bound on the number of function calls, so the number of function calls here ranges from 100 to 120 (typically) and sometimes to more than 200.

First, we investigate the performance on the given test functions in 2 dimensions. Figure 21 shows the results. As can be seen, the Rosenbrock function allows for a good gradient estimation by L-BFGS-B, and this algorithm consistently produces the lowest cost. Also, DE benefits from the LHS period and gives good results, with $\mathcal{U}$-Decay close by. SA here already shows somewhat worse performance in both variants.

For both the Rastrigin function and Ackley's function, $\mathcal{U}$-Decay gives the best results. Here, no reliable gradient information can be extracted from L-BFGS-B, leading to very poor performance.

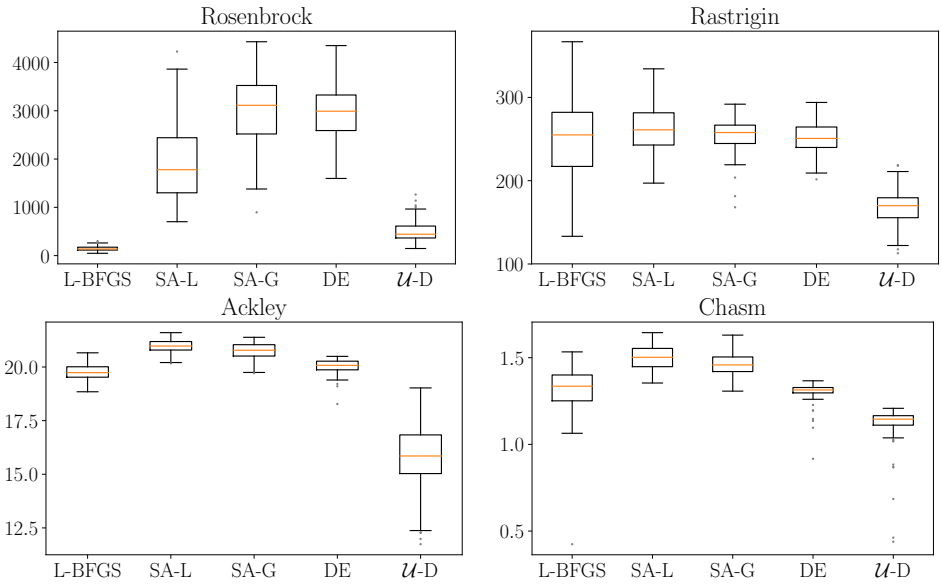

Figure 22: Results for optimization in 20 dimensions. (See caption of Figure 21 for details.)

On the challenging Chasm function, DE again gives slightly better performance than $\mathcal{U}$-Decay, with all other algorithms typically failing.

Since the proposed algorithm is designed for high-dimensional spaces, we next perform the same test using the test functions in 20 dimensions (Figure 22). Again, the shape of the Rosenbrock function allows L-BFGS-B to outperform all other algorithms, but $\mathcal{U}$-Decay is the only algorithm showing consistently high performance over all different types of test functions. In particular, it now gives far better results than DE. One reason for this is that DE chooses its population size according to the input dimensionality, so for $D = 20$, only a single epoch is performed by DE.

5.5 Discussion

In this section, we have presented a new algorithm for optimization in high-dimensional spaces with constraints which aims to be very user friendly. Only four hyperparameters are required, with reasonable defaults existing for three of them. Using this approach, better results could be achieved on four different optimization test functions compared to three other standard optimization methods. While some of these excelled at one specific type of function, the approach presented here showed consistently good results across all different categories.

The new algorithm is also flexible enough to be deployed in different scenarios. If an initial value is given which is known to be close to a local

minimum, it can be used with a smaller value of σ_0 to perform rapid local optimization. This suggests using it in combination with elaborate sampling techniques like the Latin Hypercube Sampler, which can be used to generate samples covering most of the high-dimensional value space with a smaller number of samples to find good starting regions.

Further potential for improving $\mathcal{U}$-Decay lies in changing the decay of the step size σ from linear to exponential or logarithmic, which may produce better results depending on the type of optimization problem. Furthermore, the parameter update mask may be removed if the updates are sampled from a zero-mean uniform distribution, where σ is then taken as its standard deviation. This way, the algorithm could be modified to require even less hyperparameters.

6 Discussion and Conclusion

This work aims to present an approach for solving the binding problem with a network of spiking neurons. Building on a model introduced in [9], the approach presented here focuses on providing a biologically plausible implementation, which lead to the choice of the NEST simulator as an implementation framework. Furthermore, the parameters for neuron model, network connectivity, and synaptic plasticity were chosen to lie in value ranges which may well be used in the brain.

At the heart of the model lies the content space, in which the activity of neuronal assemblies encode some entity, similar to concept cells which have been experimentally verified in the medial temporal lobe (MTL) [4]. Other groups of neurons, called variable spaces, store pointers to assemblies in the concept space, which may be used to re-activate the specific assembly after some delay. This allows the binding of entities to roles e.g. when processing the meaning of individual words within a sentence.

In the variable binding model presented here, distinct assemblies form in each of the variable spaces as they store pointers to specific assemblies in the content space. This behavior is in agreement with the experimental results from [8]. There, it was shown that the agent and the patient of a sentence can be decoded from the network activity of different subregions of the MTL in human listeners. This provides evidence that in the brain, different groups of neurons bind to different concepts when e.g. the meaning of a sentence is being decoded. In our model, these groups of neurons correspond to the different variable spaces.

The variable binding model was shown to support the execution of a number of atomic computations. Pointers stored in variable spaces allow the restoration of the activity in the concept space after some delay, thus performing an action akin to recalling the value of a variable. Pointers can also be copied from one variable space to another one, from which the original activity in the content space can then be restored reliably.

The model presented in this work introduces a number of improvements over the model described in [9]. First, each neural space is implemented here as an EI-motif with distinct populations of excitatory and inhibitory neurons. The inhibitory population provides the regulatory inhibition for the excitatory pool, which is necessary to obtain winner-take-all dynamics. This implementation is reasonable from a biological standpoint since EI-motifs are ubiquitous in the human cortex [62].

Furthermore, this work introduces non-symmetric connectivity between the different neural spaces. In [9], the excitatory neurons of the concept space form symmetric connections to the excitatory neurons of variable spaces, although the weights are different for both directions. However, symmetric connectivitiy is rarely observed in the brain. In this work, we

use random connectivity for all connections. This leads to the emergence of larger assemblies in the variable spaces, which are necessary for each neuron in the content space to receive sufficient input from a pointer in a variable space. The experiments presented in this work show that with the given parameters, the circuit shows robust performance and computations with pointers in variable spaces are very reliable. Thus, the presented model can generally be regarded as biologically plausible with respect to the neuronal implementation.

This work also introduces a new mode of storage of information: activity can be maintained over time in the variable spaces through persistently active assemblies. In [9], variable spaces are inhibited during delay periods, and the information about a stored pointer is kept in the neuronal excitability, which is implemented as an adaptive bias which increases when neurons fire. While reproducing the results with this network mode, we have shown that pointers can also be stored without making use of any excitability mechanism by simply allowing the variable spaces to remain active during delay periods. It should be pointed out that the firing rates during these waiting periods are quite low and the network dynamics remain in a stable regime.

Retaining information in a population of spiking neurons over time through persistent activity is known to be one of the mechanisms underlying working memory in cortex. It has been shown experimentally that this kind of activity sustains working memory in the prefrontal cortex during delayed response tasks [63] and it is also observable in other brain areas [64]. These findings suggest that persistent activity may indeed be the mechanism that neural spaces in the brain use to store information over time. It has further been shown that the contents which are being held in working memory are encoded in the specific activity of working memory areas [65]. The possibility of decoding contents of working memory from the neuronal activity in the corresponding brain areas is strikingly similar to the experiments by Frankland and Greene [8] on which the model discussed in this work builds upon (see above).

If information is stored in variable spaces through persistent activity, one problem may occur when more complex sequences of operations are performed. For instance, if a pointer is stored in some variable space, and the content space is engaged in an unrelated operation during the delay period and therefore not inhibited (which was the case in the simulations performed in this work), then the activity of the variable space must be kept from influencing the content space during that time. This inhibition of the connections between neural spaces (while still allowing recurrent activity with the variable space) could well be implemented in the brain since it is known that inhibition from SOM neurons can target individual dendritic branches or even specifically act on individual synapses [66].

Furthermore, a new optimization algorithm for constrained parameter spaces was introduced in Section 5. The algorithm (named $\mathcal{U}$-Decay) aims

to perform fast optimization in high dimensions with as few hyperparameters as possible. It was shown that the algorithm achieves competitive results on a number of standard optimization test functions when compared to established optimization schemes. Furthermore, the parameters for the variable binding model discussed in this work were found using this procedure. The algorithm presented here is fairly straightforward. A number of extensions of this algorithm are possible and remain to be explored (see Section 5.5).

Some of the variable binding models present in the literature propose solutions which use rather abstract models of the underlying neuronal circuitry [43] or are purely theoretical [49]. In contrast, the variable binding scheme explored in this work considers the low-level implementation of a solution to the binding problem where the implementation specifics conform to the current state of the art in neurobiology. In the following, we discuss some limitations of the model and explore possibilities for further work.

First, it should be verified that the given model enables the robust execution of the remaining operations investigated in [9]. There, a third atomic computation proposed by [48] was investigated: comparing the contents of two variable spaces. In the implementation in [9], two variable spaces with some pointers are activated after each other and their contents recalled to the content space. A read-out neuron (separate from the other circuitry) connects to the content space and encodes through its activity whether the contents of the two variable spaces are the same. This poses a natural extension to the NEST model described in this work.

Next, the given variable binding model depends on some external control circuitry which provides the inhibition and disinhibition signals for all neural spaces as operations are executed. This is necessary for meaningful computations to emerge in the network. In an extension of the present model, this circuitry could also be implemented as a network of spiking neurons controlling the state of all neural spaces. The inhibition and disinhibition of neural space was implemented as an external current injected into each excitatory neuron of the population. Using an external controller network, this inhibition could instead be provided by spikes from inhibitory neurons. Again, it would be possible to separate the inhibition of neurons from the inhibition of the pathways connecting different neural spaces to allow the information to be kept as persistent activity.

Another question remaining to be investigated is whether it is possible to use the variable spaces in a more dynamic way. Assemblies in the brain are often short-lived and may not necessarily require stable connectivity for the generation of meaningful output signals [7]. Yet, in the model investigated in this work, each concept has an associated assembly in each variable space which has formed previously and which remains rather stable over time. The problem of resetting the contents of variable spaces was addressed in [48] in an interesting way, where all synaptic weights between the content space and variable spaces are assumed to constantly decay over time. Other

options could be explored.

Apart from these low-level implementation details, the model in its current form possesses some limitations with respect to the complexity of the binding problems it can solve. In the original formulation of the binding problem [1], it concerned combinations of attributes of objects in a visual scene, with the difficulty lying in providing the correct combination of attributes. Other variations of the binding problem exist [2] which are also more complex than the simple problem considered in this work. The model presented here only considers the binding of pointer variables to a single concept. It should be investigated how combinations of objects with one or even multiple attributes can be effectively implemented to allow the decoding of the rich hierarchies of meaning encoded in even rather simple sentence structures.

One fairly simple approach would be to store multiple concepts, or concepts and their respective attributes, simultaneously inside a single neural space. This could be implemented in a number of simple ways: first, the concept space could simultaneously encode combinations of entities and their attributes. Then, it would have to be sufficiently large, which may easily lead to problems as a huge number of combinations must be encoded. Another possibility would be to have the concept space encoding separate entities and properties which are bound together in a single variable space. Then, the pointer in the variable space should restore the activity of multiple assemblies in the content space, which may again easily lead to size problems. Alternatively, there could be separate variable spaces for entities and their properties which are grouped together. This avoids the problem of individual neural spaces becoming very large, however, many different variable spaces are then required. Also, it is unclear how the group of variable spaces could interact in meaningful ways.

This last idea shows similarities to the Neural Blackboard Architecture [49] which also considers the decoding of sentences. Here, the identities of concepts occuring within a sentence are stored in variables, which can be linked and combined in different ways to obtain meaningful structures. For example, three variable could store the concepts “baby”, “Grandpa”, and “hitting”. Each of these variables now posses mechanisms allowing them to be linked together to form some structure, e.g. by forming a link between “baby” and “hitting”. These links can encode the type of bound being formed, e.g. by indicating that “baby” is the agent performing the “hitting” action. Similarly, entities could be linked to properties. This enables this architecture to encode complex structures which account for the high expressivitiy of language.

Using this approach, different structural relations could be implemented between variables in our model. This requires an elaborate external controller circuit, but could greatly increase the computational power of the model: e.g. a variable binding to the object in a sentence could also bind

to another variable which stores some adjective (e.g. the ball hit the black truck). However, this does not solve the problem of how this meaning is decoded from the auditory presentation of the sentence which at some point consist simply of a collection of words without any structure. The decoding of which concepts belong together and which word is an attribute for which entity would then necessarily have to be implemented in the controller network. It is unclear how this problem could be solved. Nevertheless, this approach seems likely to enhance the computational power of the model presented in this work.

In summary: the approach to variable binding used in this work shows promising results and serves as a good starting point for further extensions to more complex binding problems.

Appendices

A Neuron Model

In this Appendix, we analyze the neuron model of the original model [9] and the most similar neuron model provided by NEST. Our goal is to use NEST to obtain a neuron model with similar behavior.

A.1 Original Neuron Model

First, we conduct a detailed examination of the properties of the neuron model used the original model [9]. Neuron models have various properties which are significant for their behavior. These are here investigated one by one.

Leaky Integrator Behavior. The neurons used in the original work firing stochastically. Their firing rate $\rho_i(t)$ is determined their membrane potential, which is given by

$$u_i(t) = \left(1 - \frac{\Delta t}{\tau_m}\right) u_i(t - \Delta t) + \frac{\Delta t}{\tau_m} G(t) \left(\mathrm{e}^{I_i(t) - I_{\mathrm{Inh}}(t-\Delta t) + I_e + b_i(t)} - 1\right) , \tag{29}$$

where Δt is the discrete time step, τ_m is the membrane time constant, $G(t) \in \{0, 1\}$ describes the current inhibition state of the neural space, and $I_i(t)$, $I_{\mathrm{Inh}}(t)$, and I_e are the input, inhibitory, and bias currents, respectively. The term $b_i(t)$ implements an adaptive bias.

To examine the behavior of this model, we need to assess the behavior of difference equations of the form

$$x(t) = \left(1 - \frac{\Delta t}{\tau}\right) x(t - \Delta t) + \frac{\Delta t}{\tau} D(t) . \tag{30}$$

This models (29), assuming that $G(t) = 1$, i.e. the neural space is active, and further denoting the driving input term as $D(t)$. Then, as $\Delta t \to dt$, we get

$$\frac{dx(t)}{dt} + \frac{1}{\tau} x(t) = \frac{1}{\tau} D(t) \tag{31}$$

which is an inhomogeneous first-order differential equation. Since the membrane potential starts at $x(0) = 0$ and remains resting without external input, the homogeneous is the trivial solution $x(t) = 0\ \forall t$. The inhomogeneous solution is given by

$$x(t) = \frac{1}{\tau} \mathrm{e}^{-t/\tau} \int_0^t \mathrm{e}^{\zeta/\tau} D(\zeta) d\zeta \tag{32}$$

$$= \frac{1}{\tau} \int_0^t \mathrm{e}^{-\frac{t-\zeta}{\tau}} D(\zeta) d\zeta \tag{33}$$

which represents a convolution between the driving input term $D(t)$ and the exponentially decaying response term $\mathrm{e}^{-t/\tau}$. Thus, this neuron model implements the behavior of a leaky integrator.

However, since we have $\Delta t > 0$, this is not the actual behavior over time, which we can derive using the $\mathcal{Z}$-transform. Setting $G(t) = 1$ and incorporating all terms in the exponential into a generic $I(t)$, we may write

$$u_i(t) = \left(1 - \frac{\Delta t}{\tau_m}\right) u_i(t - \Delta t) + \frac{\Delta t}{\tau_m} \underbrace{\left(\mathrm{e}^{I(t)} - 1\right)}_{=:D(t)} . \tag{34}$$

Then, with $u_i(t) \overset{\mathcal{Z}}{\longleftrightarrow} \hat{U}(z)$ and $D(t) \overset{\mathcal{Z}}{\longleftrightarrow} \hat{D}(z)$, we obtain

$$\hat{U}(z) = \frac{\Delta t}{\tau_m} \cdot \frac{z}{z - (1 - \frac{\Delta t}{\tau_m})} \cdot \hat{D}(z) , \tag{35}$$

which corresponds in the time domain to the convolution

$$u_i(t) = (k * D)(t) \tag{36}$$

between the driving input term $D(t)$ and

$$k(t) = \frac{\Delta t}{\tau_m} \left(1 - \frac{\Delta t}{\tau_m}\right)^t . \tag{37}$$

Thus, depending on Δt and τ_m, the decay of the membrane potential may be faster or slower than that of $\exp(-t/\tau)$ (as derived above).

Firing Rate. The neuron membrane potential at each timestep is compared to a random number $\mathcal{U}(0, 1)$ to determine if it spikes. Thus, the firing rate is given by

$$\rho_i(t) = \frac{1}{\Delta t} \max\left(\min(u_i(t), 1), 0\right) \tag{38}$$

where the max(·) follows from the fact that the spike rate cannot be negative and the min() is needed since the neuron cannot spike more often than at each timestep. An alternative notation is

$$\rho_i(t) = \frac{1}{\Delta t} \left[u_i(t)\right]_0^1 \tag{39}$$

where $[\cdot]_x^y$ denotes clipping to the range $[x, y]$. In the following, we will assume that $0 \leq u_i(t) \leq 1$ and thus neglect the boundary values, i.e. we use the identity

$$\rho_i(t) = \frac{1}{\Delta t} u_i(t) \ . \tag{40}$$

Next, we investigate the influence of the terms in the exponent of the driving input term in (29).

Response to Bias Currents. The neuron model does not provide for constant input currents other than an offset current I_e setting the average neural activity. To investigate the response to some value of this current after the membrane potential starts at $u_i(t = 0) = 0$ at the beginning of the simulation, we investigate the fixed points of (29) while setting $G(t) = 1$ and neglecting all other influences. This can be done by simply setting $u_i(t) \stackrel{!}{=} u_i(t + \Delta t)$. We find that

$$\lim_{t \to \infty} u_i(t) = \mathrm{e}^{I_e} - 1 \ , \tag{41}$$

thus, the time step Δt and the membrane time constant τ_m do not play a role in the steady-state solution and the membrane potential approaches the actual value of the input current.

It follows that the steady-state firing rate $\bar{\rho}_i$ elicited by constant current is

$$\bar{\rho}_i = \frac{1}{\Delta t} \cdot (\mathrm{e}^{I_e} - 1) \tag{42}$$

if refractory effects are neglected.

Response to Spike Input. The input current which arises due to incoming spikes is given by

$$I_i(t) = \sum_{j \in \mathcal{S}_i^{FF}} w_{ij} z_j(t) + \sum_{j \in \mathcal{S}_i^{FB}} w_{ij} z_j(t - \Delta t) + \sum_{j \in \mathcal{S}_i^{REC}} w_{ij} z_j(t - \Delta t) \ , \tag{43}$$

where three sums run over the feed-forward, feed-back, and recurrent connections, respectively. Thus, each presynaptic spike evokes a delta-shape postsynaptic potential (PSP) $\alpha(t) = \delta(t)$.

A single spike at a synapse with weight w at $t = 0$ thus leads to a membrane potential of $K\mathrm{e}^{-t/\tau_m}$. The height of the induced exponential K can be inferred from (29) and is $K = \frac{\Delta t}{\tau_m}(\mathrm{e}^w - 1)$. Thus, neglecting all other influences, the instantaneous firing rate evoked by a single incoming spike at time t_0 is

$$\rho_i(t_0) = \frac{1}{\Delta t} \cdot \frac{\Delta t}{\tau_m}(\mathrm{e}^w - 1) = \frac{1}{\tau_m}(\mathrm{e}^w - 1) \,, \tag{44}$$

where the correction factor $1/\Delta t$ was added to obtain the firing rate in Hz. For $t > t_0$, the firing rate then decays with τ_m.

Inhibition. The inhibitory input current is given by

$$I_{\mathrm{Inh}}(t) = \left(1 - \frac{\Delta t}{\tau_{\mathrm{Inh}}}\right) I_{\mathrm{Inh}}(t - \Delta t) + \frac{\Delta t}{\tau_{\mathrm{Inh}}} G(t) \left[\sum_k u_k(t) - \Theta_{\mathrm{Inh}}\right]_{-2}^{4} \,, \tag{45}$$

where Θ_{Inh} sets the target average activity and $[\cdot]_x^y$ denotes clipping to the range $[x, y]$ as described above. The sum runs over all neurons in the neural space, thus, the inhibition current is a filtered version of the neuronal activity. The inhibition current is the same for every neuron within a neural space.

Similar to the membrane potential, this equation resembles a leaky integrator of the form of (31).

This abstract model of inhibition of neural spaces is problematic since it assumes that a neuron has knowledge of the membrane potential of other neurons in the neural space. However, this is not the case: the only information being transmitted between neurons is the presence of absence of spikes. Thus, a biologically realistic model will implement the inhibition current in a different form.

Excitability. Neurons change their excitability due to an adaptive bias $b_i(t)$ in (29), which increases their firing probability and thus their response to input. The value of this bias at timestep t is computed as

$$b_i(t) = \begin{cases} b_i(t - \Delta t) + 0.05(1 - b_i(t - \Delta t)) & \text{if neuron } i \text{ has spiked at time } t \\ b_i(t - \Delta t)\mathrm{e}^{-\Delta t/\tau_b} & \text{else} \end{cases} \,, \tag{46}$$

with a time constant τ_b for the decay. It follows that value of the bias is bound to $b_i(t) \in [0, 1]$, furthermore, the first spikes evoke the largest increase of $b_i(t)$.

If the neuron is not inhibited, i.e. $G(t) \neq 0$, and $b_i(t)$ assumes large values, then this may itself lead to a high spiking probability even in the absence of any input.

Refractoriness. The neurons in the original model undergo a refractory period Δ_{abs} after emitted a spike. During this phase, their membrane potential is set to zero, which is the initial value at the beginning of simulations and also the value used when Δ_{abs} has passed. During this period, no response to incoming spikes or currents is present, also, the bias current has no effect.

The length of the refractory period is sampled from a uniform distribution $\mathcal{U}(1, 6)$ ms. This prevents neurons from becoming locked in periodic spiking patterns.

If an additional bias current is injected into neurons, a relative refractory period is present in a way since the membrane potential needs some time to reach the steady-state value after it has been reset to zero after Δ_{abs} has ended.

Obviously, this refractory time period and the reset will alter the response to constant currents. The actual mean firing rate will be smaller than $\bar{\rho}_i$ calculated above.

Synaptic Plasticity. The STDP model in the original model is

$$\Delta w_{ij}(t) = \eta \sum_{k:\, t_j^{(k)} < t} \left(\exp\left(\frac{t - t_j^{(k)}}{\tau_+} \right) - A_- \right) \tag{47}$$

where the sum runs over the recent presynaptic spikes, A_- determines the negative offset, and η is a learning rate. This learning rule is adapted from the hard winner-take-all (WTA) rule described in [36] and is sometimes referred to as SEM rule after the spiking expectation maximization performed by neurons using it.

Summary. This neuron model implements a leaky integrator with probabilistic firing depending exponentially on the membrane potential (cf. Section 2.1). Problematic is the implementation of the inhibition, which is modeled in a biologically implausible way. Furthermore, for implementation, the learning rule for synaptic plasticity is problematic, since it differs from the standard form STDP.

Some differences to other neuron simulators may arise from the handling of time. In the simulations performed with the original model, spikes are forced on grid due to the time resolution $\Delta t = 1$ ms. More sophisticated simulation techniques for inhomogeneous Poisson processes may allow for spikes to occur at any time. Furthermore, all firing rates are relative to the time step Δt in [9], so a conversion factor is needed to obtain the actual spike rate in Hz.

A.2 The NEST Neuron Model `pp_psc_delta`

As described above, we consider the case of delta-shaped PSPs. NEST provides a host of neuron models with this behavior indicated by the suffix `_psc_delta` in their name, like the leaky Integrate-and-Fire (LIF) model `iaf_psc_delta`. If the membrane time constant τ_m and the membrane capacity C_m are to appropriate values, the desired behavior of the membrane potential can be achieved with these models, assuming the input terms are the same.

However, the desired behavior for the neurons is not deterministic (like LIF neurons), but rather stochastic. Therefore, we will base our implementation in NEST on the NEST neuron model `pp_psc_delta`, which produces spikes according to a Poisson process. The rate of this process depends on the membrane potential, which behaves according to the desired behavior specified above with respect to input currents.

Firing Rate. Spikes of `pp_psc_delta` neurons are drawn from an inhomogeneous Poisson process with firing rate

$$\rho_i(t) = c_1 \cdot V_i'(t) + c_2 \cdot \mathrm{e}^{c_3 \cdot V_i'(t)} \tag{48}$$

where c_1, c_2, and c_3 are freely adjustable parameters. $V_i'(t)$ called the effective membrane potential and is calculated from

$$V_i'(t) = V_i(t) + b_{\mathrm{sfa}}(t) \ , \tag{49}$$

where $b_{\mathrm{sfa}}(t)$ is an adaptive bias which can be used to model spike-frequency adaptation. The resting membrane potential is fixed to 0 mV for this neuron model.

In the original model, the time step Δt was 1 ms, so at the maximum spiking probability of $u_i(t) \equiv 1$, the neuron would spike at rate of 1/ms if the absolute refractory period is neglected. In NEST, the time resolution is $\Delta t = 0.1$ ms, so spikes may occur more often. Furthermore, in the original model, $u_i(t)$, which decays with τ_m, directly resembles the spike rate, while in NEST model `pp_psc_delta`, the exponential is applied when computing

the instantaneous rate, but the decay is applied to the (distinct) membrane potential. However, this difference only introduces minor chances to the final spiking probability.

With the original model, $u_i(t)$ models the spiking probability directly, while in NEST, it is possible to define a custom mapping between $V_i(t)$ and the instantaneous firing rate $\rho(t)$ of the neuron. For example, to get a firing rate of ρ_1 at the membrane potential of $V_{\rho 1}$, we can set the parameters c_1, c_2, and c_3 for (48) depending on the desired neuron behavior:

- **Linear rate.** We set $c_2 = c_3 = 0$ and $c_1 = \rho_1 / V_{\rho 1}$ to get a linear behavior. For $V_i(t) < 0$, the firing rate will be set to zero.
- **Exponential rate.** The exponential term is always greater than one, thus, $\rho(V = 0) = \rho_0 \neq 0$. To get an exponential mapping of $V_i(t) \in [0, V_{\rho 1}] \to \rho(t) \in [\rho_0, \rho_1]$, we set $c_1 = 0$, $c_2 = \rho_0$, and $c_3 = \frac{1}{V_{\rho 1}} \ln \frac{\rho_1}{\rho_0}$. The value for ρ_0 may be chosen sufficiently small.

Since the neuron we wish to model shows exponential behavior, we will assume $c_1 = 0$ in the next subsections.

Membrane Potential. The membrane potential $V_i(t)$ is computed in NEST using

$$V_i(t) = \mathrm{e}^{-\Delta t/\tau_m} V_i(t - \Delta t) + (1 - \mathrm{e}^{-\Delta t/\tau_m}) \frac{R_m}{1000} \left(I_{i,c}(t - \Delta t) + I_e\right) + I_{i,s}(t) \; . \tag{50}$$

The denominator of 1000 yields from the units used in NEST. This neuron model provides for incoming currents $I_{i,c}(t)$ as well as for currents $I_{i,s}(t)$ evoked by incoming spikes. Note the absence of any scaling factor to the latter, resulting in completely different handling of currents (including the bias) and spikes. Furthermore, it is important to note that the Δt is different to the one in the original model.

The two input currents are given by

$$I_{i,c}(t) = \sum_j w_{ij} I_j(t) \tag{51}$$

$$I_{i,s}(t) = \sum_j w_{ij} z_j(t) \tag{52}$$

where the w terms are weights, $I_j(t)$ is current from current generators, and $z_j(t) = 1$ is a presynaptic neuron has spiked.

To see the behavior of the membrane potential for some time-varying input, we first set $I_{i,c}(t) \stackrel{!}{=} I(t)$ while neglecting I_e and $I_{i,s}(t)$. Thus, we obtain

$$V_i(t) = \mathrm{e}^{-\Delta t/\tau_m} V_i(t - \Delta t) + (1 - \mathrm{e}^{-\Delta t/\tau_m}) \frac{R_m}{1000} I(t) \; . \tag{53}$$

Using the $\mathcal{Z}$-transform with $V_i(t) \overset{\mathcal{Z}}{\longleftrightarrow} \hat{V}(z)$ and $I(t) \overset{\mathcal{Z}}{\longleftrightarrow} \hat{I}(z)$, this equals

$$\hat{V}(z) = (1 - \mathrm{e}^{-\Delta t/\tau_m}) \frac{R_m}{1000} \cdot \frac{z}{z - \mathrm{e}^{-\Delta t/\tau_m}} \cdot \hat{I}(z) \tag{54}$$

in the $\mathcal{Z}$-domain, which corresponds to

$$V_i(t) = (1 - \mathrm{e}^{-\Delta t/\tau_m}) \frac{R_m}{1000} \cdot (k * I)(t) \tag{55}$$

which is a convolution between $I(t)$ and

$$k(t) = \exp\left(-t \frac{\Delta t}{\tau_m}\right) = \exp\left(-\frac{t}{\tau'_m}\right) \tag{56}$$

with a modified time constant $\tau'_m = \tau_m / \Delta t$.

The response to a current elicited by incoming spikes $I_{i,s}(t) \overset{!}{=} J(t)$ similarly is

$$V_i(t) = (k * J)(t) \; , \tag{57}$$

however, the scaling factor is absent.

In summary, we see that the neuron responds to both currents resulting from current generators or the bias as well as from incoming spikes with a leaky integrator response, however, currents and spikes are treated somewhat differently, with no possibility to scale the response to spikes other than applying a uniform scaling to all weights on ingoing synapses.

Response to Current Input. Assuming a constant input current I, which may either be the bias current ($I = I_e$) or some weighted constant external current ($I = wI_{i,c}$), and neglecting all other input, we may compute the steady-state response similar to above by setting $V_i(t) = V_i(t - \Delta t)$ and obtain

$$\lim_{t \to \infty} V_i(t) = \frac{R_m}{1000} I \; . \tag{58}$$

It follows that the steady-state firing rate $\bar{\rho}_i$ elicited by constant current is

$$\bar{\rho}_i = c_2 \exp\left(c_3 \frac{R_m}{1000} I\right) . \tag{59}$$

Similar to the original neuron model, the real firing rate of neurons with a constant current will be smaller than this due to refractory effects.

Response to Spike Input. As we have done above, we compute the instantaneous firing rate at the time of an input spike t_0 neglecting all other influences. We set $I_{i,s}(t_0) = w$ and obtain the membrane potential at t_0

$$V_i(t_0) = w , \tag{60}$$

and thus,

$$\rho_i(t_0) = c_2 \, \mathrm{e}^{c_3 w} . \tag{61}$$

Afterwards, $V_i(t)$ decays exponentially, as shown above.

Excitability. The NEST neuron model `pp_psc_delta` features an adaptive excitability changing mechanism to implement spike-frequency adaptation (SFA) through $b_{\mathrm{sfa}}(t)$ in (49). At each spike, the bias is increased by some fixed quantity q_{sfa}, which may be positive or negative. It then decays with some time constant τ_{sfa}. Since the bias is not bounded in any way, it may easily lead to instabilities, i.e. persistent excitation of the neuron if q_{sfa} is chosen too large. If the values of q_{sfa} and τ_{sfa} are kept in the stable regime, the initial changes in excitability are small after $b_{\mathrm{sfa}}(t)$ was zero. In comparison, the first changes within the original model may rapidly increase the bias due to the value-dependency of the update.

The role of the adaptive excitability in the original model is to lead to a high excitability of individual neurons after they have been active for some time period during the presentation of an input pattern. The adaptive bias may furthermore have a role in learning as it facilitates multiple consecutive spikes of a neuron and therefore weight updates. In comparison to the bounded mechanism, using the SFA model of the `pp_psc_delta` model may prove to be more complicated.

Refractoriness. The neuron model `pp_psc_delta` allows the use of an absolute refractory period Δ_{abs} during which the membrane potential is clamped to zero. Incoming spikes and currents are ignored. The length of

Δ_{abs} can either be fixed or sampled from a Γ-distribution with parameters k and μ. While k is the usual shape parameter (often denoted α), μ relates to the more common parameterization of Γ-distributions using a rate parameter via $\beta = k/\mu$ [67].

The membrane potential after a reset is always set to zero, which is somewhat surprising since when using an exponential rate, depending on the parameters c_2 and c_3, the firing rate may be quite large at this value of $V_i(t)$.

Neural Space Inhibition. In the original model, neural spaces can either be completely inhibited or disinhibited. This is a key operation for the functions of the neural circuit. The disinhibition state is given by the term $G(t) \in \{0, 1\}$ which enters the update equation for the membrane potential (29).

NEST neuron models like `pp_psc_delta` provide no direct mechanism for this operation, however, the inhibition of entire neural spaces can be performed by inducing a strong external bias current into the neurons within the neural space. This current can be provided by a `step_current_generator` which can be activated and deactivated at specific times depending on the desired state of the neural space.

Summary. The NEST neuron model described shows many generic features similar to the neuron model in the original work. Spikes are generated in a stochastic manner with the firing rate depending on the current membrane potential. This membrane potential also behaves as a leaky integrator. At some point, an exponential is applied in the calculation of the firing rate from the inputs.

In comparison to the original neuron model, the design of `pp_psc_delta` differs in several important ways:

- **Exponential decay of membrane potential.** While the membrane potential also decays exponentially, the NEST model more closely follows the target of obtaining a response with $\mathrm{e}^{-t/\tau}$. In the original model, the decay is slightly faster.
- **Firing rate.** In the original model, the membrane potential directly gives the firing rate; this is not the case with the NEST neuron model.
- **Application of exponential.** The original model first applies the exponential to inputs to get the membrane potential. In the NEST model, the membrane potential also decays, but the exponential is only applied when computing the rate as a function of it.
- **Minimum firing rate.** The original model produces spikes at a rate of zero if no input is present due to the offset of -1 to the exponential. This is not present in the second model, thus, even for negative inputs, the firing rate will be non-zero.

- **Treatment of spikes.** The NEST model treats spike complete different from incoming currents.

Furthermore, in the original work, neuron properties are modeled in a rather abstract way, while in NEST, each parameter has precise units assign (e.g. input currents are given in pA).

A.3 Mapping the Neuron Behavior to NEST Neuron Models

Since we wish to replicate the behavior of the model presented in [9], we want to model the individual neurons as closely as possible. This way, we may use the same parameters and hyperparameters to achieve similar functionality. In this section, we investigate the possibility of doing so using the NEST neuron model described in the previous section as a starting point.

One way of attempting to map the behavior of the original model onto a `pp_psc_delta` neuron would be to set the parameters of the latter so that the responses to constant current input and change of instantaneous firing rate are identical. We will investigate this approach in the next section.

Problems of `pp_psc_delta`. Neglecting the different decay of the two models, it would suffice to find parameters which lead to the same steady-state firing rate $\bar{\rho}_i$ for constant input current and the same instantaneous firing rate $\rho_i(t_0)$ at spike arrival. For the sake of clarity, we will denote terms which occur in both the original model and from the NEST neuron model with the subscripts A and B for the original model and the NEST model under investigation, respectively. The steady-state responses for currents were derived above in (42) and (59). We set them to be equal and obtain

$$\underbrace{\frac{1}{\Delta t_A} \cdot (\mathrm{e}^I - 1)}_{\bar{\rho}_A} = \underbrace{c_2 \, \exp\left(c_3 \frac{R_m}{1000} I\right)}_{\bar{\rho}_B} \tag{62}$$

where can we see that identical behavior cannot be reached. If for large I we approximate $(\mathrm{e}^I - 1) \approx \mathrm{e}^I$, we would obtain $c_2 = 1/\Delta t_A$ and $c_3 = 1000/R_m$. However, since we want $\bar{\rho}_B \approx 0$ for small I, we need $c_2 \ll 1$, which is not the case when setting c_2 in this way. Furthermore, we obtain different parameters than these (and the same problem of incoherence) when trying to match the change in instantaneous firing rate described above in (44) and (61) by setting

$$\underbrace{\frac{1}{\tau_m}(\mathrm{e}^w - 1)}_{\rho_A(t_0)} = \underbrace{c_2 \, \mathrm{e}^{c_3 w}}_{\rho_B(t_0)} \; . \tag{63}$$

Therefore, changes to the NEST neuron model are necessary.

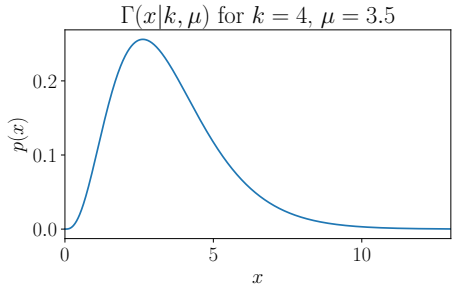

Figure 23: Distribution of Δ_{abs} using parameters which yield a similar value range to the original model.

Description of the New Neuron Model `pp_psc_delta_mod`. To address the problem of the missing negative offset of the exponential term and also the absence of a scaling term for spike responses, we create a new NEST neuron model `pp_psc_delta_mod` based on the model described above, but with (50) – the update equation for $V_i(t)$ – replaced by

$$V_i(t) = \mathrm{e}^{-\Delta t/\tau_m} V_i(t - \Delta t) + (1 - \mathrm{e}^{-\Delta t/\tau_m}) \frac{R_m}{1000} \left(I_{i,c}(t - \Delta t) + I_e\right) + z_{\text{scale}} I_{i,s}(t) \tag{64}$$

and the equation for the calculation of the firing rate (48) replaced by

$$\rho_i(t) = c_1 \cdot V_i'(t) + c_2 \cdot (\mathrm{e}^{c_3 \cdot V_i'(t)} - 1) \; . \tag{65}$$

A complete description of the new neuron model is given in Appendix B.1. With these changes in the model behavior, we still cannot entirely achieve identical behavior, but we can reach somewhat similar functionality. We will next derive the parameters required to reach this goal.

Parameter Selection. First, we set the parameters for the random distribution of the absolute refractory period Δ_{abs} for the NEST neurons, which is drawn from a $\Gamma(k, \mu)$ distribution. To obtain the same mean value $\mathbb{E}\left[\Delta_{\text{abs}}\right]$ as in the original model, we set $k = 4$ and $\mu = 3.5$ ms (Figure 23) which results in roughly the same value range.

As we have done above in Section A.3, we compare the steady-state responses for input currents to derive c_2 and c_3. From

$$\frac{1}{\Delta t_A} \cdot (\mathrm{e}^I - 1) = c_2 \left(\exp\left(c_3 \frac{R_m}{1000} I\right) - 1\right) \tag{66}$$

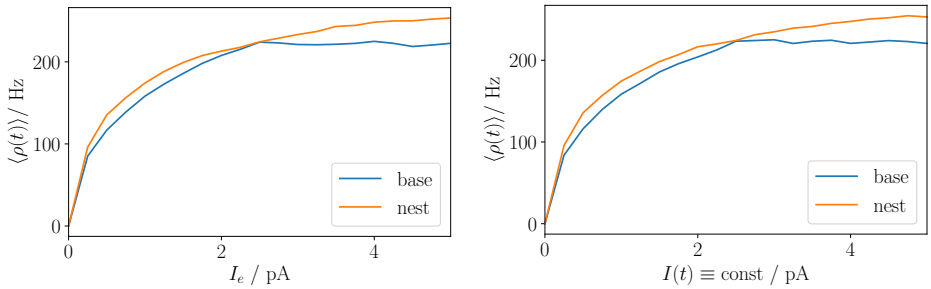

Figure 24: Match of neuron response to input currents. Neurons exhibit absolute refractory periods as described in the text. A: Neuron firing rate as a function of the bias set. B: Neuron firing rate as a function an input current. For the base model, the bias was used to set the current, as the model has no provisions for currents from external sources. Note that the base model clips at $\rho = 222$ Hz since $\mathbb{E}[\Delta_{\text{abs}}] = 3.5$ ms and at maximum one spike can be generated per timestep (with $\Delta t = 1$ ms). NEST runs by default at $\Delta t = 0.1$ ms, thus generating a much smoother curve and allowing for a higher maximum spike rate.

it now directly follows that setting $c_2 = 1/\Delta t_A$ and $c_3 = 1000/R_m$ will lead to identical behavior. The match of neuron spike rate is depicted in Figure 24.

Next, we investigate the response to incoming spikes to find a good value for the newly introduced parameter z_{scale}. As above in Section A.3, we may try to set the increase in instantaneous firing for both models equal. This now yields

$$\frac{1}{\tau_m}(\mathrm{e}^w - 1) = \frac{1}{\Delta t_A}\left(\exp\left(\frac{1000}{R_m} z_{\text{scale}} w\right) - 1\right) \tag{67}$$

where we clearly cannot set z_{scale} so that this equation holds for all values of w. (The problem arises from the different amplitudes of the increments of the firing rate in the original model for currents elicited by input spikes and the bias.) However, since the identity holds for $w = 0$, we may adjust z_{scale} so the response matches some fixed value w_{match}, where one possible choice for w_{match} is the maximum allowed weight in the model. We obtain

$$z_{\text{scale}} = \frac{1}{c_3 w_{\text{match}}} \log\left(\frac{\Delta t_A}{\tau_m}(\mathrm{e}^{w_{\text{match}}} - 1) + 1\right), \tag{68}$$

which for the maximum allowed weight in the original model $w_{\text{match}} = w_{\text{max}} = 0.8$ results in a scaling factor of $z_{\text{scale}} = 1.45 \cdot 10^{-3}$. In practice, however, this does not lead to an accurate matching of f-f curves (neuron spike

rate vs. input spike rate)[10]. We empirically find that using $z_{\text{scale}} = 5 \cdot 10^{-4}$ leads to an acceptable qualitative match (Figure 25).

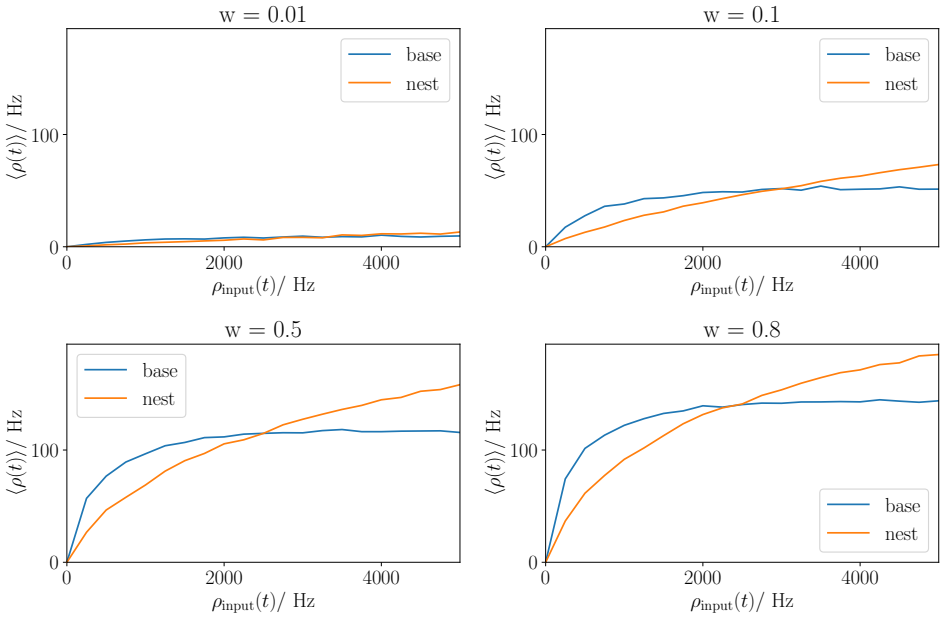

Figure 25: Match of neuron response to input spikes. Displayed is the firing rate of a single neuron as a function of the spike rate of incoming spikes for different weight values. Neurons exhibit absolute refractory periods, all parameters are chosen as described in the text. No bias current is used. The important range for matching is $\rho_{\text{input}} \in [0, 2500]$ Hz since in the original model, 25 input units firing at 100 Hz each are used per pattern.

Synaptic Plasticity. Finally, we consider the synaptic learning rule. Synaptic plasticity in NEST does not depend on the neuron model, but is governed by specific synapse models which can be parameterized and combined in various ways. The standard STDP synapse model provided by NEST follows [34] and uses a learning window of

$$\Delta w(\Delta t) = \begin{cases} A_+ \cdot \mathrm{e}^{-|\Delta t|/\tau_+} & \text{if } \Delta t \geq 0 \\ A_- \cdot \mathrm{e}^{-|\Delta t|/\tau_-} & \text{if } \Delta t < 0 \end{cases} \tag{69}$$

where $\tau_+, \tau_- > 0$, and

[10] Calculating z_{scale} using (68) does not take into account three differences in the models: first, the different expected number of spikes $\int \rho(t)dt$ differs even when the increase at spike arrival is the same as the decays do not match exactly, second, the refractory periods are not identical, and probably most significantly, the way spikes are drawn from the current firing rate does not follow the same procedure.

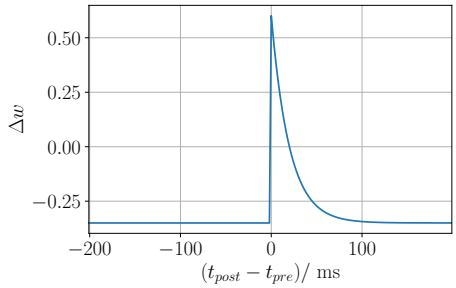

Figure 26: Learning rule measured from NEST using `stdp_synapse_sem` with $\alpha = 0$, $\tau_+ = 20$ ms, and $A_- = 0.35$. This corresponds to the learning rule used in the original model for feed-forward connections.

$$A_+ = \lambda(1 - w)^{\mu_+} \qquad \text{and} \tag{70}$$

$$A_- = -\alpha\lambda w^{\mu_-} \ . \tag{71}$$

Using this parameterization, the behavior of the original model (47) cannot be achieved. In particular, the allowed range for τ_+ and τ_- does not permit a constant negative offset as is achieved by the A_- term in the original.

We therefore implement a new synapse model `stdp_synapse_sem` which performs weight updates according to (47). The resulting learning window of the implementation as measured from NEST is shown in Figure 26. (Implementation details of the synapse model are given in Appendix B.2.)

B Description of NEST Models

B.1 Description of `pp_psc_delta_mod`

Neurons using the `pp_psc_delta` model fire stochastically. Spikes are drawn from an inhomogeneous Poisson process with instantaneous firing rate

$$\rho_i(t) = c_1 \cdot V_i'(t) + c_2 \cdot (\mathrm{e}^{c_3 \cdot V_i'(t)} - 1) \ , \tag{72}$$

where c_1, c_2, and c_3 can be used to get the desired neuron behavior (linear or exponential). The effective membrane potential $V_i'(t)$ is calculated from

$$V_i'(t) = V_i(t) + b_{i,\mathrm{sfa}}(t) \ , \tag{73}$$

where $b_{i,\mathrm{sfa}}(t)$ is an adaptive bias which is usually used to model spike-frequency adaptation (thus, normally $b_{i,\mathrm{sfa}}(t) < 0$), but can also be used to model an adaptive excitability. Its values are calculated by

$$b_{i,\mathrm{sfa}}(t) = \begin{cases} b_{i,\mathrm{sfa}}(t-\Delta t)\mathrm{e}^{-\Delta t/\tau_{\mathrm{sfa}}} + q_{\mathrm{sfa}} & \text{if neuron } i \text{ has spiked at time } t \\ b_{i,\mathrm{sfa}}(t-\Delta t)\mathrm{e}^{-\Delta t/\tau_{\mathrm{sfa}}} & \text{else} \end{cases} . \tag{74}$$

The additive quantity q_{sfa} and the decay time constant τ_{sfa} can be set arbitrarily. It is furthermore possible to use a maximum value $\hat{b}_{i,\mathrm{sfa}}$ which may be greater or less than zero (depending on the sign of q_{sfa}), then, $b_{i,\mathrm{sfa}}$ is clipped to the range $[0, \hat{b}_{i,\mathrm{sfa}}]$ after each update.

The neuron's membrane potential is computed at each time step using

$$\begin{aligned} V_i(t) = \mathrm{e}^{-\Delta t/\tau_m} V_i(t-\Delta t) + (1 - \mathrm{e}^{-\Delta t/\tau_m}) \frac{R_m}{1000} \left(I_{i,c}(t-\Delta t) + I_e\right) \\ + z_{\mathrm{scale}} I_{i,s}(t) \end{aligned} \tag{75}$$

where τ_m is the membrane time constant, R_m is the membrane resistance, I_e is the bias current, z_{scale} is used to scale the effect of incoming spikes, ant Δt is the time step (0.1 ms by default in NEST). The currents $I_{i,c}(t)$ and $I_{i,s}(t)$ result from incoming currents and spikes, respectively, and are calculated via

$$I_{i,c}(t) = \sum_j w_{ij} I_j(t) \tag{76}$$

$$I_{i,s}(t) = \sum_j w_{ij} z_j(t) \tag{77}$$

where $I_j(t)$ are incoming currents, $z_j(t)$ are incoming spikes, and w_{ij} are weights assigned to the specific connection.

Like `pp_psc_delta`, the modified neuron model allows the use of an absolute refractory period Δ_{abs} during which the membrane potential is clamped to zero and both incoming spikes and currents are ignored. The length of Δ_{abs} can either be set to a constant or drawn randomly (once at the creation of the neuron) from a Γ-distribution with parameters shape k and mean μ.

B.2 Description of `stdp_synapse_sem`

The standard STDP synapse in NEST `stdp_synapse` closely follows [34]. For our simulations, this model was modified to create `stdp_synapse_sem` which implements the following equation. Weight updates are performed using

$$\Delta w(\Delta t) = \begin{cases} \eta \cdot \mathrm{e}^{-|\Delta t|/\tau_+} - A_- & \text{if } \Delta t \geq 0 \\ \eta \cdot \alpha \cdot \mathrm{e}^{-|\Delta t|/\tau_-} - A_- & \text{if } \Delta t < 0 \end{cases} \tag{78}$$

where $\tau_+, \tau_- > 0$ are time constants determining the width of the learning window, A_- determines the negative offset, α determines the shape of the depression term in relation to the facilitation term and η is a learning rate. Using $\alpha = 0$, we obtain the plasticity behavior of [9], while $\alpha > 0$ gives the most commonly used learning curve with depression of the postsynaptic neuron fires before the presynaptic one, and facilitation if they fire in the other order. Using $\alpha < 0$ (in particular $\alpha = -1$), we obtain a symmetric, "Hebbian"-like learning window which leads to strong connections and persistent activation with a network. Figure 27 shows the weight updates measured from NEST using $\alpha = 0$ and parameters used in [9]. Figure 28 shows the same curve for $\alpha = -1$.

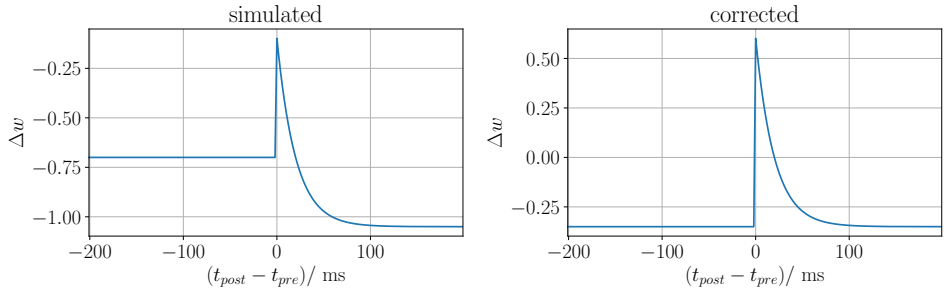

Figure 27: Resulting STDP window for the parameters in the original model ($\tau_+ = 20$ ms and $A_- = -0.35$, the learning rate was set to 1). Synaptic updates in NEST are triggered by presynaptic spikes, therefore, for pre-before-post, we need an additional pre-synaptic spike at the end of the simulation to trigger the weight update of the target pre-post pairing. However, during each weight update depression is performed regardless of whether the postsynaptic neuron has actually fired. This leads to the resulting raw simulation output on the left. On the right, the corrected learning window is shown. In practice, neurons regularly spike, so these issues should not play a significant role.

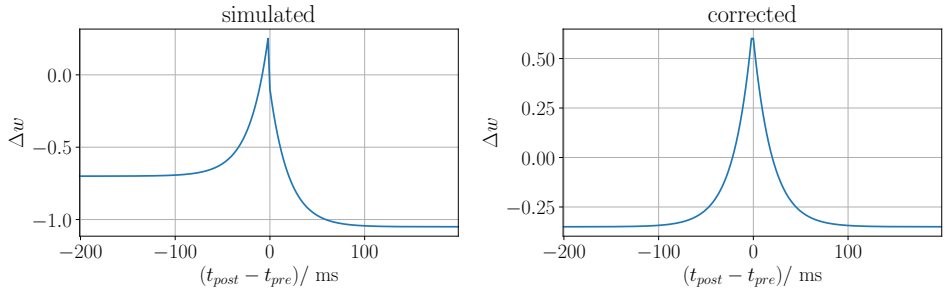

Figure 28: Learning window for `stdp_synapse_sem` using $\tau_+ = \tau_- = 20$ ms, $\alpha = -1$, $A_- = 0.35$, and $\eta = 1$. (See the caption of Figure 27 for details on the recording process of this curve.)

C Parameters for NEST Circuits

C.1 Parameters for NEST SWTA Circuits

The parameters used for the NEST SWTA circuits (as described in Section 3) are given in Tables 6 and 7.

parameter	unit	excitatory	inhibitory
R_m	MΩ	10	10
C_m	pF	1000	1000
τ_m	ms	10	10
c_1	Hz / mV	0	1000
c_2	Hz	1000	0
c_3	1 / mV	100	0
I_e	pA	0.1	0
z_{scale}		$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
q_{sfa}	mV	0	0
Δ_{abs}	ms	$\sim \Gamma(k = 4, \mu = 3.5)$	$\sim \Gamma(k = 4, \mu = 3.5)$

Table 6: Neuron parameters.

parameter	unit	X→E	E→E	E→I	I→E	I→I
p		1	0.1	0.575	0.6	0.55
weight init		$\sim \mathcal{U}(0, 0.8)$	0	17.39	−4.76	−16.67
weight bounds		$[0, 0.8]$	$[0, 0.25]$			
Δ_{syn}	ms	$\sim \mathcal{U}(1, 10)$	1	0.5	0.5	1
STDP		yes	yes	no	no	no
α		0	0			
τ_+	ms	20	20			
τ_-	ms	–	–			
A_-		0.35	0.35			
η		0.01	0.0025			

Table 7: Network parameters including parameters for synaptic plasticity. Since τ_- is not used when $\alpha = 0$, it is not given here.

C.2 Parameters for NEST Neural Spaces

Tables 8 to 11 give the parameters used for experiments in Section 4.

Experiments without neuronal excitability. Each neuron uses the parameters given in Table 6 above, and each space uses the E→I, I→E, and I→I parameters from Table 7. The network parameters are given below for the content space $\mathcal{C}$ and one or more variable spaces.

parameter	unit	$\mathcal{X} \to$ E	$\mathcal{N} \to$ E	E→E
p		1	0.1	0.1
weight init		$\sim \mathcal{U}(0, 0.8)$	$\sim \mathcal{U}(0.22, 0.53)$	0
weight bounds		$[0, 0.8]$	$[0, 1.01]$	$[0, 0.6]$
Δ_{syn}	ms	$\sim \mathcal{U}(1, 10)$	$\sim \mathcal{U}(1, 10)$	1
α		0	0	-1
τ_+	ms	25	13	25
τ_-	ms	43	43	43
A_-		0.4	0.17	0.5
η		0.01	0.009	0.0025

Table 8: Network parameter for content spaces including parameters for synaptic plasticity. The E→I, I→E, and I→I parameters used are given in Table 7.

parameter	unit	$\mathcal{C} \to$ E	E→E
p		0.1	0.1
weight init		$\sim \mathcal{U}(0.44, 0.93)$	$\sim \mathcal{U}(0.39, 0.85)$
weight bounds		$[0, 1.27]$	$[0, 1.26]$
Δ_{syn}	ms	$\sim \mathcal{U}(1, 10)$	1
α		0	-1
τ_+	ms	39	40
τ_-	ms	39	39
A_-		0.10	0.44
η		0.005	0.008

Table 9: Network parameter for variable spaces including parameters for synaptic plasticity. The E→I, I→E, and I→I parameters used are given in Table 7.

Experiments with neuronal excitability. Again, each neuron uses the parameters given in Table 6, and each space uses the E→I, I→E, and I→I parameters from Table 7. Additionally, the neurons in the variable spaces use an adaptive bias increment of $q_{\text{sfa}} = -0.0002$ and are clipped at $\hat{b}_{\text{sfa}} = -0.005$. The network parameters are given below for both the content space $\mathcal{C}$ and one or more variable spaces.

parameter	unit	$\mathcal{X} \to$ E	$\mathcal{N} \to$ E	E→E
p		1	0.1	0.1
weight init		$\sim \mathcal{U}(0, 0.8)$	$\sim \mathcal{U}(0.19, 0.38)$	0
weight bounds		$[0, 0.8]$	$[0, 0.87]$	$[0, 0.6]$
Δ_{syn}	ms	$\sim \mathcal{U}(1, 10)$	$\sim \mathcal{U}(1, 10)$	1
α		0	0	-1
τ_+	ms	25	20	25
τ_-	ms	—	—	40
A_-		0.4	0.47	0.5
η		0.01	0.008	0.0025

Table 10: Network parameter for content spaces including parameters for synaptic plasticity. The E→I, I→E, and I→I parameters used are given in Table 7.

parameter	unit	$\mathcal{C} \to$ E	E→E
p		0.1	0.1
weight init		$\sim \mathcal{U}(0.48, 0.86)$	$\sim \mathcal{U}(0.44, 0.87)$
weight bounds		$[0, 1.33]$	$[0, 1.08]$
Δ_{syn}	ms	$\sim \mathcal{U}(1, 10)$	1
α		0	-1
τ_+	ms	21	37
τ_-	ms	—	49
A_-		0.28	0.52
η		0.004	0.006

Table 11: Network parameter for variable spaces including parameters for synaptic plasticity. The E→I, I→E, and I→I parameters used are given in Table 7.

D Parameters for Optimization Algorithms

These are the parameters used for the results shown in Figures 21 and 22. All algorithms may perform 100 function calls, although this bound could not be enforced on L-BFGS-B.

algorithm	parameters
L-BFGS-B	use gradient approximation
	(otherwise SciPy 0.18.1 defaults)
Simulated Annealing	$T_0 = 1$
(local updates)	$T_{\text{decay}} = 0.99$
	candidates: $\mathbf{x}_{\text{new}} = \mathbf{x} + \mathbf{r}$,
	$r_i \sim \mathcal{U}(-1, 1)\ \forall i$
	iterations $= N$
Simulated Annealing	$T_0 = 1$
(global updates)	$T_{\text{decay}} = 0.99$
	candidates: $\mathbf{x}_{\text{new}} = \mathbf{r}$,
	$r_i \sim \mathcal{U}(x_{\min,i}, x_{\max,i})\ \forall i$
	iterations $= N$
Differential Evolution	population size $S = 5 \cdot D$
	iterations $= N/S$
	(otherwise SciPy 0.18.1 defaults)
$\mathcal{U}$-Decay	$\sigma_0 = 1$
	$\bar{\sigma} = 0.001$
	$p_{\text{m}} = 0.5$
	iterations $= N$

Table 12: Parameters for optimization algorithms.

E References

[1] Christoph Von Der Malsburg. The correlation theory of brain function. In *Models of neural networks*, pages 95–119. Springer, 1994.

[2] Anne Treisman. The binding problem. *Current opinion in neurobiology*, 6(2):171–178, 1996.

[3] Gary Marcus, Adam Marblestone, and Thomas Dean. The atoms of neural computation. *Science*, 346(6209):551–552, 2014.

[4] Rodrigo Quian Quiroga. Neuronal codes for visual perception and memory. *Neuropsychologia*, 83:227–241, 2016.

[5] Donald Olding Hebb. *The organization of behavior: A neuropsychological approach*. John Wiley & Sons, 1949.

[6] Matias J Ison, Rodrigo Quian Quiroga, and Itzhak Fried. Rapid encoding of new memories by individual neurons in the human brain. *Neuron*, 87(1):220–230, 2015.

[7] György Buzsáki. Neural syntax: cell assemblies, synapsembles, and readers. *Neuron*, 68(3):362–385, 2010.

[8] Steven M Frankland and Joshua D Greene. An architecture for encoding sentence meaning in left mid-superior temporal cortex. *Proceedings of the National Academy of Sciences*, 112(37):11732–11737, 2015.

[9] Robert Legenstein, Christos H Papadimitriou, Santosh Vempala, and Wolfgang Maass. Assembly pointers for variable binding in networks of spiking neurons. *arXiv preprint arXiv:1611.03698*, 2016.

[10] Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

[11] Jochen Martin Eppler, Moritz Helias, Eilif Muller, Markus Diesmann, and Marc-Oliver Gewaltig. Pynest: a convenient interface to the nest simulator. *Frontiers in neuroinformatics*, 2, 2008.

[12] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[13] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[15] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[16] Mark F Bear, Barry W Connors, and Michael A Paradiso. *Neuroscience*, volume 2. Lippincott Williams & Wilkins, 2007.

[17] Alan L Hodgkin and Andrew F Huxley. Propagation of electrical signals along giant nerve fibres. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, pages 177–183, 1952.

[18] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[19] WM Kistler, W Gerstner, and J Leo van Hemmen. Reduction of hodgkin-huxley equations to a single-variable threshold model. *Neural Comput.*, 9(LCN-ARTICLE-1997-001):1015–1045, 1997.

[20] Jonathan T Ting and Paul EM Phillips. Neurotransmitter release. *Wiley Encyclopedia of Chemical Biology*, 2008.

[21] Tiago Branco and Kevin Staras. The probability of neurotransmitter release: variability and feedback control at single synapses. *Nature reviews. Neuroscience*, 10(5):373, 2009.

[22] Renaud Jolivet, Alexander Rauch, Hans-Rudolf Lüscher, and Wulfram Gerstner. Predicting spike timing of neocortical pyramidal neurons by simple threshold models. *Journal of computational neuroscience*, 21(1):35–49, 2006.

[23] Romain Brette and Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of neurophysiology*, 94(5):3637–3642, 2005.

[24] Tiago Branco and Michael Häusser. The single dendritic branch as a fundamental functional unit in the nervous system. *Current opinion in neurobiology*, 20(4):494–502, 2010.

[25] George Kastellakis, Denise J Cai, Sara C Mednick, Alcino J Silva, and Panayiota Poirazi. Synaptic clustering within dendrites: an emerging theory of memory formation. *Progress in neurobiology*, 126:19–35, 2015.

[26] Jordan Guergiuev, Timothy P Lillicrap, and Blake A Richards. Towards deep learning with segregated dendrites. *arXiv preprint arXiv:1610.00161*, 2016.

[27] Robert Legenstein and Wolfgang Maass. Branch-specific plasticity enables self-organization of nonlinear computation in single neurons. *Journal of Neuroscience*, 31(30):10787–10802, 2011.

[28] Robert Urbanczik and Walter Senn. Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528, 2014.

[29] Gina Turrigiano. Too many cooks? intrinsic and synaptic homeostatic mechanisms in cortical circuit refinement. *Annual review of neuroscience*, 34:89–103, 2011.

[30] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[31] Yang Dan and Mu-ming Poo. Spike timing-dependent plasticity of neural circuits. *Neuron*, 44(1):23–30, 2004.

[32] Natalia Caporale and Yang Dan. Spike timing–dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.*, 31:25–46, 2008.

[33] Larry F Abbott and Sacha B Nelson. Synaptic plasticity: taming the beast. *Nature neuroscience*, 3(11s):1178, 2000.

[34] Robert Gütig, Ranit Aharonov, Stefan Rotter, and Haim Sompolinsky. Learning input correlations through nonlinear temporally asymmetric hebbian plasticity. *Journal of Neuroscience*, 23(9):3697–3714, 2003.

[35] Bernhard Nessler, Michael Pfeiffer, and Wolfgang Maass. Stdp enables spiking neurons to detect hidden causes of their inputs. In *Advances in neural information processing systems*, pages 1357–1365, 2009.

[36] Bernhard Nessler, Michael Pfeiffer, Lars Buesing, and Wolfgang Maass. Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLOS Computational Biology*, 9(4):1–30, 04 2013.

[37] Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, 4(10):e1000180, 2008.

[38] Björn M Kampa, Johannes J Letzkus, and Greg J Stuart. Dendritic mechanisms controlling spike-timing-dependent synaptic plasticity. *Trends in neurosciences*, 30(9):456–463, 2007.

[39] Srdjan D Antic, Wen-Liang Zhou, Anna R Moore, Shaina M Short, and Katerina D Ikonomu. The decade of the dendritic nmda spike. *Journal of neuroscience research*, 88(14):2991–3001, 2010.

[40] George Kastellakis, Alcino J Silva, and Panayiota Poirazi. Linking memories across time via neuronal and dendritic overlaps in model neurons with active dendrites. *Cell reports*, 17(6):1491–1504, 2016.

[41] Matthew Larkum. A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex. *Trends in neurosciences*, 36(3):141–151, 2013.

[42] Jason J Moore, Pascal M Ravassard, David Ho, Lavanya Acharya, Ashley L Kees, Cliff Vuong, and Mayank R Mehta. Dynamics of cortical dendritic membrane potential and spikes in freely behaving rats. *Science*, 355(6331):eaaj1497, 2017.

[43] Kenneth J Hayworth. Dynamically partitionable autoassociative networks as a solution to the neural binding problem. *Frontiers in computational neuroscience*, 6, 2012.

[44] Tony A Plate. Holographic reduced representations. *IEEE Transactions on Neural networks*, 6(3):623–641, 1995.

[45] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012.

[46] Chris Eliasmith. *How to build a brain: A neural architecture for biological cognition*. Oxford University Press, 2013.

[47] Trenton Kriete, David C Noelle, Jonathan D Cohen, and Randall C O'Reilly. Indirection and symbol-like processing in the prefrontal cortex and basal ganglia. *Proceedings of the National Academy of Sciences*, 110(41):16390–16395, 2013.

[48] Ariel Zylberberg, Luciano Paz, Pieter R Roelfsema, Stanislas Dehaene, and Mariano Sigman. A neuronal device for the control of multi-step computations. *Papers in physics*, 5(2):1–15, 2013.

[49] Frank Van der Velde and Marc De Kamps. Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences*, 29(1):37–70, 2006.

[50] Zeno Jonke, Robert Legenstein, Stefan Habenschuss, and Wolfgang Maass. Feedback inhibition shapes emergent computational properties of cortical microcircuit motifs. *arXiv preprint arXiv:1705.07614*, 2017.

[51] Robert Legenstein, Zeno Jonke, Stefan Habenschuss, and Wolfgang Maass. A probabilistic model for learning in cortical microcircuit motifs with data-based divisive inhibition. *arXiv preprint arXiv:1707.05182*, 2017.

[52] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[53] Michael D McKay, Richard J Beckman, and William J Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[54] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[55] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[56] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[57] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

[58] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

[59] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[60] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, page 101, 2005.

[61] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3), 2007.

[62] Rodney J Douglas and Kevan AC Martin. Neuronal circuits of the neocortex. *Annu. Rev. Neurosci.*, 27:419–451, 2004.

[63] Clayton E Curtis and Mark D'Esposito. Persistent activity in the prefrontal cortex during working memory. *Trends in cognitive sciences*, 7(9):415–423, 2003.

[64] Clayton E Curtis and Daeyeol Lee. Beyond working memory: the role of persistent activity in decision making. *Trends in cognitive sciences*, 14(5):216–222, 2010.

[65] Bruno B Averbeck and Daeyeol Lee. Prefrontal neural correlates of memory for sequences. *Journal of Neuroscience*, 27(9):2204–2211, 2007.

[66] Guangyu Robert Yang, John D Murray, and Xiao-Jing Wang. A dendritic disinhibitory circuit mechanism for pathway-specific gating. *Nature communications*, 7, 2016.

[67] Athanasios Papoulis and S Unnikrishna Pillai. *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 2002.