Philipps-Universität Marburg

Johannes Gille

Fachbereich 17
AG Allgemeine und Biologische Psychologie
AE Theoretische Kognitionswissenschaft

Learning in cortical microcircuits with multi-compartment pyramidal neurons

Supervisors:

Prof. Dr. Dominik Endres, Philipps-Universität Marburg

Dr. Johan Kwisthout, Radboud University

Contents

Chapter 1

Introduction

1.1 Motivation

TODO: fill with citations?

The outstanding learning capabilities of the human brain have been found to be elusive and as of yet impossible to replicate in silicio. While the power and utility of classical Machine-learning solutions has improved greatly in recent years, these approaches can not serve as an adequate model of human cognition. The sheer number of neurons and synapses in the brain makes simulations of an entire brain impossible with current hardware constraints. In fact, it has been found to be a substantial challenging to create artificial neural networks that simulate even parts of human neurophysiology while simultaneously being able to learn in a goal-oriented way.

The literature entails numerous approaches to adress this challenges, with varying degrees of success. In this thesis, I will investigate one such approach, and attempt to modify it in a way that it more closely resembles properties exhibited by the human neocortex.

TODO: eigentvalue of the hessian matrix. if they have different signs, something might be off

1.2 The Backpropagation of errors algorithm

The Backpropagation of errors algorithm (henceforth referred as "Backprop") forms the backbone of modern machine learning. **TODO: cite**It is as of yet unmatched with regard to training in deep neural networks due to its unique capability to attribute errors in the output of a network to activations of specific neurons within its hidden layers and adapt incoming weights in order to improve network performance. This property also forms the basis of the algorithm's name; After an initial forward pass to form a prediction about the nature of a given input, a separate backward pass propagates the arising error through all layers in reverse order. During this second network traversal, local error gradients dictate, to what extent a

given weight needs to be altered so that the next presentation of the same sample would elicit a lower error in the output layer.

While Backprop continues to prove exceptionally useful in conventional machine learning systems, attempts use it to explain the exceptional learning capabilities of the human brain have so far not been successful **phrasing**. In fact, Backprop as a mechanism for synaptic plasticity in the brain is dismissed by many neuroscientists as biologically implausible. This dismissal is often focussed on three mechanisms that are instrumental for Backprop (Whittington and Bogacz, 2019; Crick, 1989; Bengio et al., 2015):

1.2.1 Local error representation

Within conventional artificial neural networks (ANNs), the neurons only serve the purpose of transmitting signals in a feedforward fashion. The errors on the other hand are computed and propagated by a completely separate algorithm which can access the entirity of the network state. The algorithm requires the activation of all downstream neurons in order to compute the weight changes of a given layer. Since plasticity in biological neurons is only dependent on factors that are local to the synapse, these errors would need to be represented within the neurons of each layer. Several mechanisms have been proposed to do this in a biologically plausible way, which will be discussed **TODO: in a later chapter.**

1.2.2 The weight transport problem

During the weight update stage of Backprop, errors are transmitted between layers with the same weights that are used in the forward pass. In other words, the magnitude of a neuron-specific error that is propagated through a given connection should be proportional to its impact on output loss during the forward pass. For this to work, a neuronal network would require feedback connections that mirror both the network structure and synaptic weights exhibited by the original network. It was long assumed that the feedback weights are required to be an exact match, but Liao et al. (2016) showed, that this constraint can be relaxed somewhat to a concordance of weight signs.

Bidirectional connections are common in the cortex, yet it is unclear by which mechanism pairs of synapses would be able to align. This issue becomes particularly apparent when considering long-range pyramidal projections, in which feedforward and feedback synapses would be separated by a considerable distance.

Several theories have been proposed as to how biological neural networks could alleviate this issue, which will be discussed **TODO: in a later section**.

1.2.3 Neuron models

While the fundamental computational unit of ANNs is called a neuron, it usually shares little resemblance to biological neurons. Most notably, these types of artificial neurons transmit a

continuous activation. In theory, these activations correspond to the firing rate of a spiking neuron. Yet particularly with regard to synaptic plasticity, spike based communication poses a substantial challenge. Plasticity rules derived for rate neurons do not necessarily have an easily derived counterpart for spiking neurons. A notable example for this issue is Backprop itself; The local error gradient $\frac{\delta E}{\delta \phi(u_i)}$ **TODO: cleanup** requires a method of filtering the spiketrain which itself has no derivative **phrasing**...

To differentiate the two, I will be following Haider et al. (2021) in this Thesis; When referring to biologically plausible, leaky neurons I will be using the term *"neuronal"*. Respectively, when discussing abstract neurons with instantaneous response, I will be using *"neural"*.

Furthermore, a given neuron's activation is computed from a simple weighted sum of all inputs. This fails to capture the complex nonlinearities of synaptic connections that appear to be critical for neuronal computation. Finally, these abstract neurons - at least in classical Backprop - have no persistence through time. Thus, their activation is dictated strictly by the presentation of a single stimulus, in contrast to the leaky membrane dynamics exhibited by biological neurons. **TODO: talk about multiplicity of possible neuron models?**

Transitioning to spiking neurons

It should also be noted that neuroscience has produced a vast amount of results which imply that the brain relies on precise spike timing for its plasticity **TODO: cite**. Thus, one might expect rate neuron approximations to underperform compared to spiking neurons, since a spiketrain contains more information than a rate approximation. However, several studies describe the opposite effect: Spiking neurons often perform substantially worse on learning tasks when compared to rate neurons with similar plasticity rules (Stapmanns et al., 2021) **TODO: cite more**. Given this data I assume that my implementation will also perform slightly worse than the existing rate implementations.

TODO: discuss supervisor issue?

1.3 Approximations of the Backprop algorithm

These results have largely led neuroscience to dismiss Backprop as a plausible learning mechanism for biological brains, and focus different learning mechanisms **TODO: do some reading and expand this**. Yet, Backprop has remained the gold standard against which all supervised learning mechanisms eventually have to compare.

And despite its apparent biological implausibility, it does share some notable parallels to learning in the brain: When training on real-world data, artificial neural networks have been shown to learn similar representations as those found in brain areas responsible for comparable tasks Whittington and Bogacz (2019); Yamins and DiCarlo (2016). **TODO: talk about ideal observer?**

Thus, several attempts have been made to find biologically more plausible approximations of Backprop, which will be discussed in this section.

1.4 Cortical microcircuits

Chapter 2

Methods

2.1 Neuron and network model

This section will go into detail about the computations behind the network which was developed by Sacramento et al. (2018) and expanded by Haider et al. (2021).

2.1.1 Network architecture

The model which was original described by Sacramento et al. (2018) contains a strongly recurrent network structure, which will be explained in this section. It can be functionally separated into layers, with information flowing from the input layer through one or more hidden layers to the output layer. Neurons at the input layer receive no feedback connections and serve primarily to apply a low-pass filter over the input sequence injected into their membrane. Output layers have no interneurons, and are usually modeled as pyramidal neurons without an apical compartment. Hidden layers consist of a pyramidal- and an interneuron population, which are fully connected to each other in both directions (see Figure 2.1). Feedforward connections between layers are facilitated by all-to-all connections between their respective pyramidal neurons and innervate the basal compartments. Feedback connections from superficial layers and interneurons on the other hand arrive at apical compartments. Interneurons receive lateral input from all pyramidal neurons of their layer, as well as feedback information from superficial pyramidal neurons. These feedback connections are special, since they connect one pyramidal neuron to exactly one interneuron. Instead of transmitting neuronal activation, this connection relays somatic voltage directly. The resulting dynamics share features of electrical synapses, which will be discussed in Section 3.1. To understand what purpose this connectivity serves in our model, neuron and plasticity models require some elaboration.

[1]Note that the input layer is displayed as having interneurons here. This appears to be a mistake in the graphic. In the implementation, interneurons are only modelled in hidden layers.

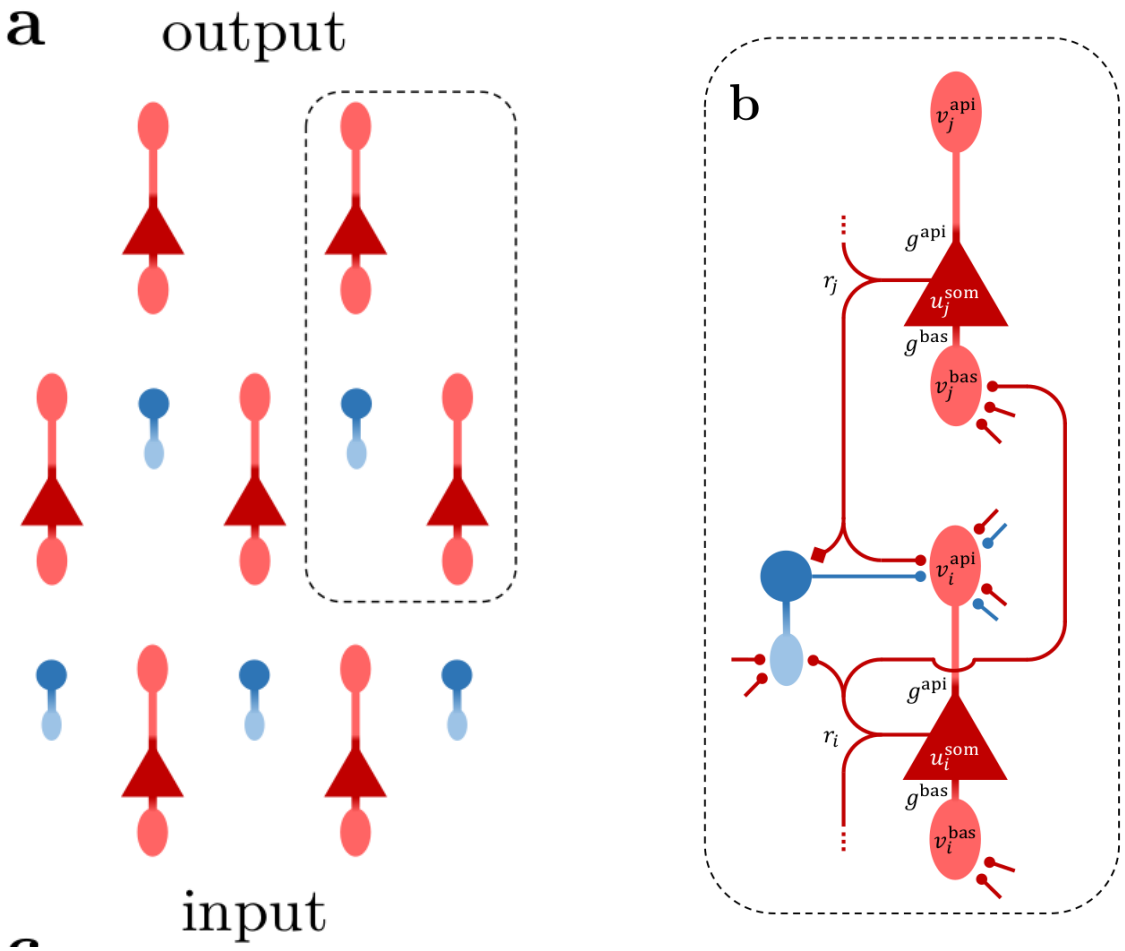

Figure 2.1: Network structure, from Haider et al. (2021). **a:** pyramidal- (red) and interneurons (blue) in a network of three layers. Note the fact that the number interneurons in a layer is equal to the number of pyramidal neurons in the subsequent layer[1]. **b:** connectivity within the highlighted section. Feedback pyramidal-to-interneuron connections (displayed with rectangular synapse) transmit pyramidal somatic potential directly and connect to a single interneuron. This enables these interneurons to learn to match their corresponding next-layer pyramidal neurons. All other synapses (circles) transmit the neuron somatic activation $\phi(u^{som})$ and fully connect their origin and target populations.

2.1.2 Neuron models

The network contains two types of multi-compartment neurons; Pyramidal neurons with three compartments each, and interneurons with two compartments each. They integrate synaptic inputs into dendritic potentials, which leak into the soma with specific conductances. Note that vector notation will be used throughout this section, and u_l^P and u_l^I denote the column vectors of pyramidal- and interneuron somatic voltages respectively. Synaptic weights W are likewise assumed matrices.The activation r_l^P of pyramidal neurons at layer l is given by applying the synaptic transfer function ϕ to to their somatic potentials u_l^P:

$$r_l^P = \phi(u_l^P) \tag{2.1}$$

$$\phi(x) = \begin{cases} 0 & \text{if } x < -\varepsilon \\ \gamma \, log(1 + e^{\beta(x-\theta)}) & \text{if } x < \varepsilon \\ \gamma \, x & \text{otherwise} \end{cases} \tag{2.2}$$

where ϕ acts componentwise on u and can be interpreted as a smoothed variant of ReLu **TODO: cite** with scaling factors γ, β, θ and threshold parameter ε. The derivative somatic membrane potentials of layer l pyramidal neurons is given by:

$$C_m \dot{u}_l^P = -g_l u_l^{som} + g^{bas} v_l^{bas} + g^{api} v_l^{api} \tag{2.3}$$

where v_l^{bas} and v_l^{api} are the membrane potentials of basal and apical dendrites respectively, and g^{bas} and g^{api} their corresponding coupling conductances. The membrane capacitance C_m is 1 in all relevant simulations, and will be omitted from all Equations from here on out.

In the original model, dendritic compartments have no persistence between simulation steps. Thus, they are defined at every timestep t through incoming weight matrices and presynaptic activities:

$$v_l^{bas}(t) = W_l^{up} \; \phi(u_{l-1}^P(t)) \tag{2.4}$$

$$v_l^{api}(t) = W_l^{pi} \; \phi(u_l^I(t)) \; + \; W_l^{down} \; \phi(u_{l+1}^P(t)) \tag{2.5}$$

Weight nomenclature conforms to the python implementation of the network by Haider et al. (2021); Weights are indexed by the layer in which the target neuron is located and belong to one of four populations. Feedforward and feedback pyramidal-to-pyramidal connections arriving at layer l are denoted W_l^{up} and W_l^{down} respectively. Lateral pyramidal-to-interneuron connections are denoted with W_l^{ip} and their corresponding feedback connections with W_l^{pi}. Since the input layer contains no synapses, indexing begins with 0 for the first hidden layer.

Interneurons integrate synaptic information by largely the same principle, but instead of feedback information arriving at the apical compartment, it is integrated directly into the soma. Through this, interneurons functionally resemble the original neuron from Urbanczik and Senn (2014) most closely.

$$C_m \dot{u}_l^I = -g_l u_l^I + g^{dend} v_l^{dend} + i^{nudge,I} \tag{2.6}$$
$$i^{nudge,I} = g^{nudge,I} u_{l+1}^P \tag{2.7}$$
$$v_l^{dend} = W_l^{ip} \ \phi(u_l^P) \tag{2.8}$$

where $g^{nudge,I}$ is the interneuron nudging conductance, and u_{l+1}^P is the somatic voltage of a corresponding pyramidal neuron in the next layer. Since each interneuron is innervated by exactly one pyramidal neuron and vice versa, they will be called each others **sister neurons** from here on.**TODO: discuss whether this 1 to 1 style connection is plausible!**. Pyramidal neurons in the output layer N effectively behave like interneurons, as they receive no input to their apical compartment. Instead, the target activation u^{tgt} is injected into their soma:

$$C_m \dot{u}_N^P = -g_l u_N^P + g^{bas} v_N^{bas} + i^{nudge,tgt} \tag{2.9}$$
$$i^{nudge,tgt} = g^{nudge,tgt} u^{tgt} \tag{2.10}$$

These neuron dynamics correspond closely to those by Urbanczik and Senn (2014), including the extension to more than two compartments which was proposed in the paper. It should be noted however, that they are simplified in some key ways. Primarily, dendritic couplings and nudging are not relative to the somatic or some reversal potential (i.e. $i^{nudge,tgt} = g^{nudge,tgt}(u^{tgt} - u_N^P)$), but are simplified to absolute values. **TODO: Figure out if this makes a difference. maybe create a second branch to try them out?**

2.2 Urbanczik-Senn Plasticity

The synapses in the network are all modulated according to variations of the "Urbanczik-Senn" plasticity rule (Urbanczik and Senn, 2014), which will be discussed in this section. **TODO: add paragraph about relevance of urbanczik-senn**

The plasticity rule requires the postsynaptic neuron to have one somatic and at least one dendritic compartment, to the latter of which synapses connect. The membrane potential of the dendritic compartment leaks into the somatic compartment as described in Equation 2.6.

Functionally, the plasticity rule changes the synaptic weight in such a way, as to minimize discrepancies between somatic and dendritic potential. The change in weight for a synapse

fron neuron j to neuron i is thus given by:

$$\dot{w}_{ij} = \eta \ (\phi(u_i^{som}) - \phi(\hat{v}_i^{bas})) \ \phi(u_j^{som})^T \tag{2.11}$$
$$\hat{v}_i^{bas} = \alpha \ v_i^{bas} \tag{2.12}$$

with learning rate η and u^T denoting the transposition of the vector u (which is by default assumed to be a column vector). The dendritic prediction is simply a scaled version of the dendritic potential by the constant factor α, which is calculated from coupling and leakage conductances. For example, basal dendrites of pyramidal neurons in Sacramento et al. (2018) are attenuated by $\alpha = \frac{g^{bas}}{g_l + g^{bas} + g^{api}}$. If a current is injected into the soma, it creates a difference between somatic firing rate and dendritic prediction (referred to as a dendritic error from here on). Urbanczik and Senn (2014) show, that **TODO: what?**

Weight changes for the synapses in a hidden layer l are thus given by:

$$\dot{w}_l^{up} = \eta_l^{up} \ (\phi(u_l^P) - \phi(\hat{v}_l^{bas})) \ \phi(u_{l-1}^P)^T \tag{2.13}$$
$$\dot{w}_l^{ip} = \eta_l^{ip} \ (\phi(u_l^I) - \phi(\hat{v}_l^{dend})) \ \phi(u_l^P)^T \tag{2.14}$$
$$\dot{w}_l^{pi} = \eta_l^{pi} \ - v_l^{api} \ \phi(u_l^I)^T \tag{2.15}$$
$$\dot{w}_l^{down} = \eta_l^{down} \ (\phi(u_l^P) - \phi(w_l^{down} r_{l+1}^P)) \ \phi(u_{l+1}^P)^T \tag{2.16}$$

Note that pyramidal-to-pyramidal feedback weights w_l^{down} are not plastic in most simulations, and will be discussed in Section 6.4.

2.3 The self-predicting network state

In this model, euron dynamics, plasticity rules and network architecture form an elegant interplay, which I expand on in this section. We will

Since each interneuron receives a somatic nudging signal from its corresponding next-layer pyramidal neuron, incoming synapses from lateral pyramidal neurons adapt their weights to match feedforward pyramidal-to-pyramidal weights. In intuitive terms; Feedforward pyramidal-to-pyramidal weights elicit a certain activation in the subsequent layer, which is fed back into corresponding interneurons. In order to minimize the dendritic error term in Equation 2.14, pyramidal-to-interneuron weight matrices at every layer must match these forward weights ($w_l^{ip} \approx \rho w_l^{up}$) up to some scaling factor ρ. ρ depends on the difference in parameters between pyramidal- and interneurons, and is close to 1 for most of the simulations considered here. As long as no feedback information arrives at the pyramidal neurons, the Urbanczik-Senn plasticity drives synaptic weight to fulfill this constraint. Note, that this alignment of two separate sets of outgoing weights is acheived with only local information. Therefore this mechanism

could plausibly align the weights of biological synapses that are physically separated by long distances.

Next, consider the special case for interneuron-to-pyramidal weights in Equation 2.15, in which plasticity does not serve to reduce discrepancies between dendritic and somatic potential. The error term is instead defined solely by the apical compartment voltage[2]. Thus, plasticity in these synapses works towards silencing the apical compartment. The apical compartments also receive feedback from superficial pyramidal neurons, whose synapses will be considered non-plastic for now. As shown above, interneurons each learn to match their respective sister neuron activity. Thus, silencing of apical compartments can only be acheived by mirroring the pyramidal-to-pyramidal feedback weights ($w_l^{pi} \approx -w_l^{down}$).

When enabling plasticity in only these two synapse types, the network converges on the "**self-predicting state**" (Sacramento et al., 2018). This state is defined by a minimization of four error metrics at each hidden layer l:

- The symmetries between feedforward ($w_l^{ip} \approx \rho w_l^{up}$) and feedback ($w_l^{pi} \approx -w_l^{down}$) weights. Mean squared error between these pairs of matrices will be called **Feedforward -** and **Feedback weight error** respectively.
- Silencing of pyramidal neuron apical compartments ($v_l^{api} \approx 0$). The Frobenius norm **TODO: cite**of apical compartment voltages within a layer is called the **Apical error**.
- Equal activations in interneurons and their respective sister neurons ($\phi(u_l^I) \approx \phi(u_{l+1}^P)$). The mean squared error over these vectors is called the **Interneuron error**.

is there a mathematical symbol for this type of convergence?

All of these equalities are approximate, since the network does not ever reach a state in which all of these values are zero. In the original implementation, these deviations are minute and can likely be explained with floating point conversions. Since it is impossible to replicate the timing of the original precisely within NEST, the NEST simulations deviate more strongly from this ideal. The key insight here is, that this state is not clearly defined by absolute error thresholds, but is rather flexible. Thus, networks are able to learn successfully even when their weights are initialized imperfectly. **TODO: or not at all? some science is required here**

An analysis of the equations describing the network reveals that the self-predicting state is stable **phrasing**. When Interneuron error is zero, dendritic and somatic compartments of all interneurons are equal, thus effectively disabling plasticity in incoming synapses. Likewise, a silenced apical compartment will disable plasticity in all incoming synapses. Synapses from lateral interneurons (Equation 2.15) only depend on the apical compartment itself, thus can

[2]In strict terms, it is defined by the deviation of the dendritic potential from its specific reversal potential. Since that potential is zero throughout, $-v_l^{api}$ remains as the error term.

not change in this state. Similarly, the apical compartment is also the driving factor for the dendritic error of feedforward synapses (Equation 2.13), since it affects somatic activity when active[3]. Thus, all plasticiy in the network is disabled, and remains in the self-predicting state regardless of the kind of stimulus injected into the input layer. This fact highlights another important property of networks in this state. Notice, how information flows backwards through the network; All feedback signals between layers ultimately pass through the apical compartments of pyramidal neurons. Thus, successful silencing of all apical compartments implies that no information can travel backwards between layers (with the exception of interneurons receiving top down signals). As a result, the network behaves strictly like a fully connected feedforward network consisting only of pyramidal neurons. The recurrence within this network is in balance, and completely cancels out its own effects. This holds true as long as all conditions for the self-predicting state are fulfilled, and the network only receives external stimulation at the input layer. One interpretation of this is, that the network has learned to predict its own top-down input. A failure by interneurons to fully explain (i.e. cancel out) top-down input thus results in a prediction error, encoded in deviation of apical dendrite potentials from their resting state. This prediction error in turn elicits plasticity in all synapses connecting to its pyramidal neuron, which drives the network towards a self-predicting state that is congruent with the novel top-down signal. Therefore, these neuron- specific prediction errors are the driving force of supervised learning in these networks

2.4 Training the network

Starting with a network in the self-predicting state, performing time-continuous supervised learning then requires an injection of a target activation into the network's output layer alongside with a stimulus at the input layer. Since output layer neurons feed back into both interneurons and pyramidal neurons of the previous layer, local prediction errors arise. Synapses activatee and drive to minimize the prediction errors, which requires the network to replicate the target activation from activations and weights of the last hidden layer.

Note, that this mechanism is not exclusive to the last two layers. Any Apical errors cause a change in somatic activity, which previous layers will fail to predict. Thus, errors are propagated backwards through the entire network, causing error minimization at every layer. See the Supplementary analysis of Sacramento et al. (2018) for a rigorous proof that this type of network does indeed approximate the Backpropagation algorithm **what does the O mean?** .

Classical backpropagation relies on a strict separation of a forward pass of some trainig

[3]This feature is actually rather important when contemplating biological neurons using the Urbanczik-Senn plasticity. In the original paper, currents were injected directly into the soma to change the error term. The the introduction of a second dendrite which performs that very task is much more useful, as originally described by the authors. Whether interneurons could be modeled by the same principles will be discussed in Section **TODO: talk about interneuron dendritic trees**

pattern and a backwards pass dependent on the arising loss at the output layer. Since the present network is time-continuous, stimulus and target activation are injected into the network simultaneously. These injections are maintained for a given presentation time t_{pres}, in order to allow the network to calculate errors through its recurrent connections before slowly adapting weights. Particluarly for deep networks, signals travelling from both the input and output layer require some time to balance out and elicit the correct dendritic error terms. This property poses the most significatn drawback of this type of time-continuous approximation of Backpropagation: The network tends to overshoot activations in some neurons, which in turn causes an imbalance between dendritic and somatic compartments. This effect causes the network to change synaptic weights away from the desired state during the first few miliseconds of a stimulus presentation. A possible approach to alleviate this issue is to disable plasticity for the first few milliseconds of stimulus presentation. After this initial phase, the network has reached a state in which local error terms are correct, and the plasticity rules produce useful weight changes. Yet a mechanism by which neurons could implement this style of phased plasticity is yet to be found, making this approach questionable in terms of biological plausibility. The solution Sacramento et al. instead found for this issue was to drastically reduce learning rates, while increasing stimulus presentation time. This solution is sufficient to prove that plasticity in this kind of network is able to perform error propagation, but still has some issues. Most notably, training is highly inefficient and computationally intensive. A closer investigation of the issue and a possible solution will be discussed in Section 2.9.

2.5 The NEST simulator

One of the key research questions motivating this thesis is whether the proposed architecture would be able to learn successfully when employing spike-based communication instead of the rate connections for which the original implementation was developed. As a framework for my spike-based implementation two options were considered: The first one was to use the existing implementation of the network within PyTorch **cite this or nah?** and expand it to employ spiking neurons. PyTorch does in principle support spiking communication between layers, but is streamlined for implementing less recurrent and less complex network and neuron models. A more pressing issue yet is efficiency; PyTorch is very well optimized for efficiently computing matrix operations on dedicated hardware. This makes it a prime choice for simulating large networks of rate neurons, which transmit all of their activations between layers at every simulation step. Spiking communication between leaky neurons is almost antithetical to this design philosophy and thus can be expected to perform comparatively poor when using this backend.

The second option was to use the NEST simulator **TODO: cite**, which was developed with highly parallel simulations of large spiking neural networks in mind. It is written in C++ and uses the *Message Passing Interface* to efficiently communicate payloads across both

threads and compute nodes. One design pillar of the simulator that is particularly relevant for this project, is the event-based communication scheme that underpins all simulated nodes. It ensures, that communication bandwidth at every simulation step is only used by the subset of nodes which transmit spikes or other signals at that time step. The simulator also increases computational efficiency by enforcing a transmission delay of at least one simulation step for all connections. This allows Events[4] to be created out of order, and processed at the postsynaptic neuron during the next simulation step. While this feature is critical for efficiency in highly parallel environments, it inflicts some limitations on the simulation which will be discussed in **TODO: create section**. Yet the most notable advantage of the NEST simulator is, that an event-based implementation of the Urbanczik-Senn plasticity alongside a corresponding neuron model had already been developed for it. Combined with the fact that I had more prior experience with NEST over PyTorch, I decided to implement the neuron model in the NEST simulator.

Early experiments showed that implementing spiking neuron and synapse models that behave similarly enough to the original python implementation would be more challenging than expected. The interplay between neuron activity and synaptic plasticity proved to be very sensitive to minor differences in parametrization and simulator setup. It was also uncertain if spiking neurons would be able to perform the desired learning tasks at all. Finally, the NEST simulator itself imposes some limitations on the simulations, which could not be ruled out as being responsible for the failure of these early experiments. Therefore, I decided to recreate the existing rate neuron model within NEST as well as the spiking implementation. This substantially increased the complexity of the NEST implementation, but turned out to be a critical step towards successful learning using spiking neurons. It allowed me to reliably identify whether errors arose from using a different simulation backend, or from the transition from rate-based to spiking neurons.

2.6 Neuron model implementation

The neuron model relies heavily on the one developed by Stapmanns et al. (2021), who showed that spiking neurons are able to learn tasks that were originally implemented for the rate neurons and synapses developed by Urbanczik and Senn (2014). The neuron model (named `pp_cond_exp_mc_urbanczik` within NEST) is an exact replication of the Urbanczik-Senn neuron in terms of membrane dynamics. In order to reduce the number of parameters, reversal potentials as well as the flow of currents from the soma to the dendrites were omitted in my model. Furthermore, the implementation by Sacramento et al. (2018) requires synapses to be able change the sign of their weight through plasticity, which is not permitted in the original

[4] An Event in NEST is an abstract C++ Class that is created by neurons, and transmitted across threads and compute nodes by the Simulator. A Multitude of Event types are provided (i.e. `SpikeEvent`, `CurrentEvent`, `RateEvent`), each able to carry specific types of payload and being processed differently by the postsynaptic neuron.

model. For this reason, the strict separation of excitatory and inhibitory synapses had to be relaxed aswell. The resulting neuron dynamics closely resemble those described in Equation 2.3.

Instead of transmitting their activation $r = \phi(u)$ at every time step, these neurons use a poisson point process to determine the number of spikes to be sent during a simulation time step of duration Δt, which will be assumed to be $0.1ms$ from here on out. The spiking probability uses the average firing rate as a parameter for the poisson distribution:

$$P\{n \text{ spikes during } \Delta t\} = e^{-r\Delta t}\frac{(r\ \Delta t)^n}{n!} \tag{2.17}$$

$$\langle n \rangle = r\ \Delta t \tag{2.18}$$

TODO: find a citation for this

with n denoting the number of spikes to be sent. Within the code, the number of spikes during Δt is then drawn from a random number generator. Note that this mechanism makes the assumption that more than one spike can occur per simulation step. NEST was developed with this possibility in mind and provides a *multiplicity* parameter for Spike Events, which is processed at the postsynaptic neuron.

The model is also capable of simulating a refractory period after spiking, which is a well documented property of biological neurons. The probability of at least one spike occuring within the next simulation step is the inverse probability of no spike occuring. Thus, when inserting $n = 0$ into Equation 2.17, the probability of eliciting at least one spike within the next simualtion step becomes:

$$P\{n \geq 1\} = 1 - e^{-r\Delta t} \tag{2.19}$$

Drawing from this probability then determines whether or not a spike is sent during that step, denoted with the function $s(t)$, which outputs 1 if a spike is sent during the interval $[t, t + \Delta t]$ **TODO: Notation correct?**, and 0 otherwise. If the neuron sends a spike, the spiking probability is set to 0 for the duration of the refractory period t_{ref}.

Since the Urbanczik-Senn plasticity relies on dendritic and somatic voltage differences at every timestep, a more sophisticated model of dendritic voltage is required. The approach for rate neurons described in Equations 2.4 and 2.5 would cause dendritic voltages to be at rest most of the time and reach extreme values when spikes arrive through the synapses. Dendritic compartments therefore need to be modeled with leaky dynamics similar to those described for the somatic compartment. Thus, the basal dendritic compartment of neuron j evolves according to: **TODO: do we need the same equations for the soma?**

$$C_m^{bas}\dot{v}_j^{bas} = -g_l^{bas}\ v_j^{bas} + \sum_i W_{ji}s_i(t) \tag{2.20}$$

$$\tag{2.21}$$

with membrane capacitance C_m^{bas}, leakage conductance g_l^{bas} being specific to the basal dendrite and independent from other compartments. Note that these equations are calculated individually for each neuron and do not employ the matrix notation used for layers of rate neurons. Apical compartments are modeled by the same principle and with largely the same parameters.

2.7 Event-based Urbanczik-Senn plasticity

One major challenge in implementing this architecture with spiking neurons is the Urbanczik-Senn plasticity introduced in Section 2.2. Since the plasticity rule is defined for rate neurons which communicate their activation at every simulation step, computing the updates efficiently requires more complex computations. Fortunately, this problem has been successfully solved in NEST by Stapmanns et al. (2021) for the two-compartment neurons described in Urbanczik and Senn (2014). This Section will discuss its algorithm and implementation.

Since NEST is an event-based simulator, most of the plasticity mechanisms developed for it compute weight changes at the location (i.e. thread and compute node) of the postsynaptic neuron whenever an Event is received. This has several advantages; It allows the thread that created the Event to continue processing neuron updates instead of having to synchronize with the thread that is responsible for the postsynaptic neuron. With regard to spiking neuron simulations this could be considered a 'fire-and-forget' approach to paralellization. More importantly, this feature mirrors the local properties of biological synaptic plasticity models. STDP for example can be performed efficiently by updating the presynaptic trace when a spike traverses the synapse, and retreiving a trace of preceding postsynaptic activity the neuron (Morrison et al., 2008). This retrieval requires less communication, as neuron and synapse are stored on the same thread. For a spiking implementation of the Urbanczik-Senn plasticity, weight updates require the dendritic error at every time step instead of just a scalar trace at the time of a spike. Thus, a more complex mechanism for weight updates was requred, for which two basic possibilities were considered:

In a **Time-driven scheme**, dendritic errors are made available to synapses at every timestep, and weight changes are applied instantaneously. This approach requires very little memory, as no history of dendritic errors needs to be stored. It does come at the cost of computational efficiency, as calls to the synaptic update function are as frequent as neuron updates - for all synapses. Particularly for large numbers of incoming synapses, as is common for

simulations of cortical pyramidal neurons (Potjans and Diesmann, 2014), this entails numerous function calls per time step. It also implies, that weight changes need to be computed at time steps where they are not immediately required (i.e. when there is no presynaptic activity), or when no weight change is necessary. Therefore, this approach proved costly in terms of Computational resources.

An **Event-driven scheme** on the other hand, updates synaptic weights only when a spike is sent through the synapse. A history of the dendritic error **introduce term?**is stored at the postsynaptic neuron, which is read by each synapse when a spike is transmitted in order to compute weight changes. Note, how the history of dendritic error applies equally for all incoming connections, and therefore does only needs to be recorded once. Alongside each entry in the history, a counter is stored and incremented whenever a synapse has read the history at that time step. Thus, when all synapses have read out an entry, it can be deleted. Thus, the history dynamically grows and shrinks during simulation and is only ever as long as the largest inter-spike interval (ISI) of all presynaptic neurons. This approach proves to be more efficient in terms of computation time, since fewer calls to the update function are required per synapse. It does come at the cost of memory efficiency, as the history can grow particularly large for simulations with low in-degrees or large ISI[5]. Despite these drawbacks, an Event-based proved most efficient when simulating spiking neural networks on a distributed system. The following section will explain how the dendritic error history is integrated at the synapse under a spiking neuron paradigm.

2.7.1 Integrating weight changes

Stapmanns et al. describes the Urbanczik-Senn plasticity rule in the form

$$\dot{w}_{ij}(t) = F(s_j^*(t), V_i^*(t)) \tag{2.22}$$

where the change in weight $\dot{w}_{ij}$ of a synapse from neuron j to neuron i at time t is given a function F that depends on the dendritic error of the postsynaptic neuron V_i^* and the presynaptic spiketrain s_j^*. The $*$ operator denotes a causal function, indicating that a value $V_i^*(t)$ potentially depends on all previous values of $V_i(t' < t)$. One can formally integrate Equation 2.22 in order to obtain the weight change between two arbitrary time points t and T:

[5]It should also be noted that in this approach requires redundant integration of the history by every synapse. Stapmanns et al. propose a third solution, in which this integration is performed once whenever a spike is received, with the resulting weight change being applied to all synapses immediately. This approach proved to be even more efficient for some network configurations, but is incompatible with simulations where incoming synapses have heterogeneous synaptic delays due to the way that these delays are processed by the NEST simulator. See Section 3.1.3 in Stapmanns et al. (2021) for a detailed explanation.

$$\Delta w_{ij}(t,T) = \int_t^T dt' F[s_j^*, V_i^*](t') \tag{2.23}$$

This integral is of fundamental importance for a spike-based implementation under an Event-driven scheme, as it is needed to calculate the change in weight between two arriving spikes. If s_j^* were the activation of a rate neuron model, this integration would be fairly straightforward. Yet for spiking neurons, it is necessary to approximate the presynaptic activation ($\phi(u_j)$) in order to achieve similar weight changes as would be evoked by a rate model. For this, a well established solution is to transform the spiketrain s_j into a decaying trace using an exponential filter kernel κ:

$$\kappa(t) = H(t)\frac{1}{t}e^{\frac{-t}{\tau_\kappa}} \tag{2.24}$$

$$H(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t \leq 0 \end{cases} \tag{2.25}$$

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t')g(t-t')dt' \tag{2.26}$$

$$s_j^* = \kappa_s * s_j. \tag{2.27}$$

with filter time constant τ_κ. To obtain the trace of a spiketrain, it is convolved (Equation 2.26) with the exponential filter kernel κ. The filter uses the Heaviside step function $H(t)$, and is therefore only greater 0 for positive values of t (also called a one-sided exponential decay kernel). This property is important, as integration limits of the convolution can be truncated when f and g are only supported on $[0,\infty)$:

$$(f * g)(t) = \int_0^t f(t')g(t-t')dt' \tag{2.28}$$

Since spiketrains are naturally only supported for $t > 0$, this simplified integral allows for an exact and efficient computation of the spike train. In this particular case, the Function F on the right hand side of Equation 2.22 is defined as:

$$F[s_j^*, V_i^*] = \eta\,\kappa * (V_i^* s_j^*) \tag{2.29}$$

$$V_i^* = (\phi(u_i^{som}) - \phi(\hat{v}_i^{dend})) \tag{2.30}$$

this notation seems slightly abusive but is taken precisely from Stapmanns et al. (2021) with learning rate η. V_i^* then is the dendritic error of the dendrite that the synapse

between j and i is located at[6]. Writing out the convolutions in Equation 2.23 explicitly, we obtain

$$\Delta w_{ij}(t,T) = \int_t^T dt' F[s_j^*, V_i^*](t') \tag{2.31}$$

$$= \int_t^T dt' \ \eta \int_0^{t'} dt'' \ \kappa(t'-t'') V_i^*(t'') s_j^*(t'') \tag{2.32}$$

Computing this Equation directly is computationally inefficient due to the nested integrals. Yet, it is possible to break up the integrals into two simpler computations and rewrite the weight change as:

$$\Delta W_{ij}(t,T) = \eta \left[I_1(t,T) - I_2(t,T) + I_2(0,t) \left(1 - e^{-\frac{T-t}{\tau_\kappa}}\right)\right] \tag{2.33}$$

$$I_1(a,b) = \int_a^b dt \ V_i^*(t) s_j^*(t) \tag{2.34}$$

$$I_2(a,b) = \int_a^b dt \ e^{-\frac{b-t}{\tau_\kappa}} V_i^*(t) s_j^*(t) \tag{2.35}$$

$$\tag{2.36}$$

See Section **TODO: ref** for a rigorous proof of this equation. These integrals can be solved analytically in reasonable time, particularly since some of the components are shared between I_1 and I_2, which allows for a streamlined computation of both simultaneously. Furthermore, when a spike at time T is processed, $I_2(0,T) = I_2(0,t) + I_2(t,T)$ can trivially be computed and stored in preparation for the next spike, since all of its components are required anyway. All of this results in a rather efficient integration scheme given the complexity of the integral described in Equation 2.32.

TODO: any more shit about this integral?

2.8 Neuron model adaptations

- additional compartments
- top-down plasticity changes
- leaky dendrites

[6]The dendritic error here is defined as the difference between two hypothetical rates. The original implementation uses the difference between the actual postsynaptic spiketrain and the dendritic rate ($V_i^* = (s_i - \phi(\hat{v}_i^{dend}))$). Furthermore, Stapmanns et al. show that stochastically generating a spiketrain from the dendritic potential ($V_i^* = (s_i - s_i^{dend})$) also results in successful learning, although at the cost of additional training time. This variant was chosen in order to match the rate implementation as closely as possible.

- pyr and intn differences

$$u_k^p = \frac{g_B}{g_{lk} + g_B + g_A} v_{B,k}^P + \frac{g_A}{g_{lk} + g_B + g_A} v_{A,k}^P \tag{2.37}$$

$$\hat{v}_{B,k}^P = \frac{g_B}{g_{lk} + g_B + g_A} v_{B,k}^P \tag{2.38}$$

$$\hat{v}_k^I = \frac{g_B}{g_{lk} + g_B} v_k^I \tag{2.39}$$

$$\lambda = \frac{g_{som}}{g_{lk} + g_B + g_{som}} \tag{2.40}$$

2.9 Latent Equilibrium

The most significant drawback of the Sacramento model is the previously mentioned requirement for long stimulus presentation times and appropriately low learning rates. This makes the network prohibitively inefficient for the large networks required for complex learning tasks. This

Sacramento et al. developed a steady-state approximation of their network which does not model the full neuron dynamics. It does not suffer from these issues and shows that their model can in principle solve more demanding learning tasks such as MNIST. Yet these types of approximation are much further detached from biological neurons than the original model and thus do not lend themselves to an investigation of biological plausibility. Furthermore, they are incompatible with spike-based communication between neurons. Thus, neither the fully modeled neuron dynamics nor the steady-state approximation are fully suited for this thesis A substantial improvement to the model that solves this dilemma was developed by Haider et al. (2021), and will be discussed here.

When not relying on a steady-state approximation of neuron dynamics, the Sacramento network is held back by the slow development of leaky neuron dynamics. When a stimulus is presented at the input layer, input neurons slowly adapt their somatic potential to the target voltage, while gradually increasing their activation. Their output in turn slowly changes the activation of subsequent layers, all the way to the output layer. Given a membrane time constant τ_m, a feedforward network with N layers of leaky neurons thus has a relaxation time constant of $N\tau_m$. Yet in this network, a target activation simultaneously injected into the output neurons slowly propagates backwards through the network. Errors at early layers require subsequent layers to be fully relaxed in order to correctly compute their local error terms, effectively being dependent on two network passes. Haider et al. state that this kind of network therefore requires $2N\tau_m$ before correct error terms are computed for a given input-output pairing.

This is a major issue, as plasticity during the first few miliseconds of a stimulus presentation

is driven by faulty error terms. The network thus tends to 'overshoot', and needs to undo the synaptic changes made in this initial phase during the later phase of a stimulus presentation in order to make progress on the learning task. Haider et al. call this issue the "relaxation problem" and suggest that it might be inherent to most established attempts at biologically plausible Backpropagation algorithms (Whittington and Bogacz, 2017; Guerguiev et al., 2017; Sacramento et al., 2018; Millidge et al., 2020).

The approach of increasing presentation time therefore is somewhat problematic. It implicitly tolerates adverse synaptic changes at all levels of the network. Synaptic changes that are meant to immediately be undone are of course an inefficient use of resources, which is undesirable for a biological system. More importantly, this kind of slow network dynamics is likely too slow for the quick responses required for perception and aciton in the real world (Bartunov et al., 2018).

Ideally, this initial phase would be skipped or shortened, in order to reduce erroneous plasticity. This would allow for a relaxation of the constraints put on presentation time and learning rates, thus increasing computational efficiency. The approach proposed by Haider et al. is to change the parameter of the activation function ϕ, called *Latent Equilibrium*. Neurons in the original implementation (henceforth called *Sacramento neurons*) transmit a function of their somatic potential u_i, which is updated through euler integration at every simulation step (Equation 2.41). In contrast, neurons using Latent Equilibrium (henceforth called *LE neurons*) transmit a function of what the somatic potential is expected to be in the future. To calculate this expected future somatic potential $\breve{u}$, the integration is performed with a larger euler step (Equation 2.42).

$$u_i(t + \Delta t) = u_i(t) + \dot{u}_i(t)\ \Delta t \tag{2.41}$$

$$\breve{u}_i(t + \Delta t) = u_i(t) + \dot{u}_i(t)\ \tau_{eff} \tag{2.42}$$

Instead of broadcasting their rate based on the current somatic potential ($r_i(t) = \phi(u_i(t))$), LE neurons send their predicted future activation, denoted as $\breve{r}_i(t) = \phi(\breve{u}_i(t))$. The degree to wich LE neurons look ahead is determined by the *effective membrane time constant* $\tau_{eff} = \frac{C_m}{g_l + g^{bas} + g^{api}}$. This time constant takes into account the conductance with which dendritic compartments leak into the soma, which is a key driving factor for the speed at which the network balances out, as noted previously. When employing the default parametrization suggested by Haider et al. **TODO: reference parameter table**, it is slightly lower than reported pyramidal neuron time constants (McCormick et al., 1985) at approximately $5.26ms$. While employing LE neurons does not alter the dynamics at the input layer, neurons at subsequent layers approach their steady state more quickly, as depicted in Figure 2.2. This property extends to the local error terms of pyramidal- and interneurons, which on average are much

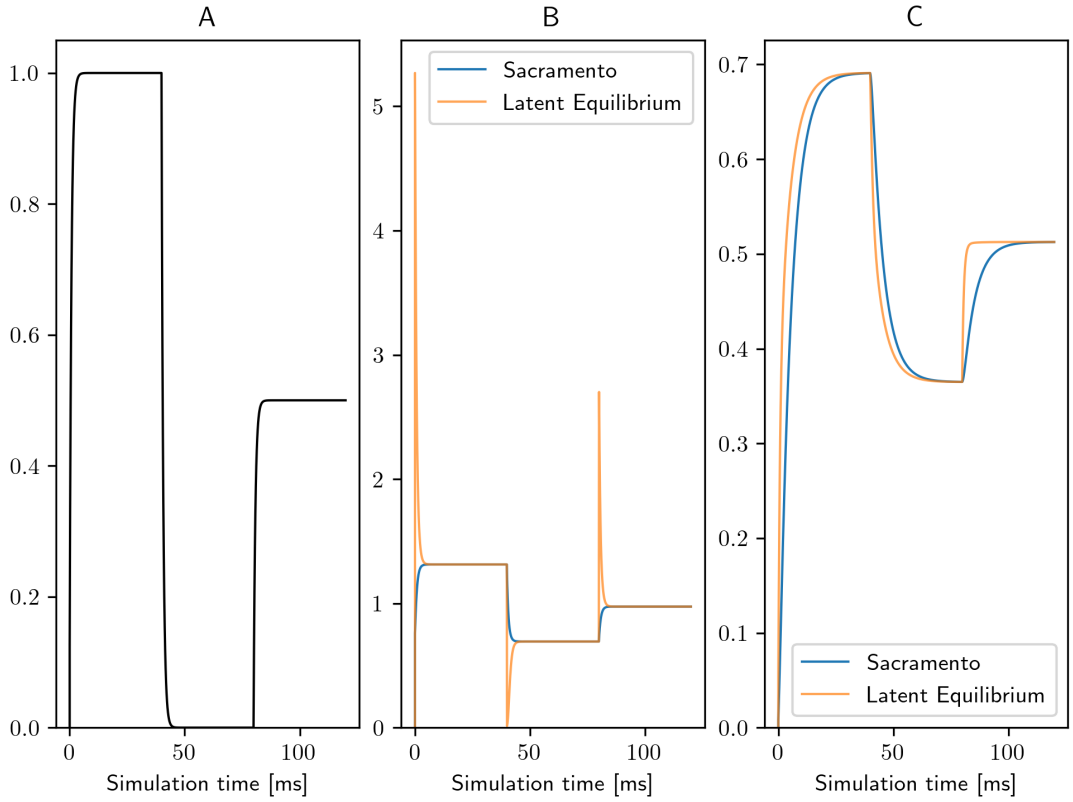

Figure 2.2: Comparison of signal transmission between Sacramento model and a neuron using Latent Equilibrium. **A:** Somatic voltage of a single input neuron. Three different currents are injected for $40ms$ each. The input neuron x acts as a low pass filter on these currents with time constant $\tau_x = 0.8ms$. **B:** Activation of the input neuron using Sacramento dynamics $\phi(u_x)$ (blue), and Latent equilibrium $\phi(\breve{u}_x)$ (orange). Note how strongly the LE neuron reacts to changes in somatic voltage, leading to spikes in activation. After the input neuron has reached its steady-state ($\dot{u}_x = 0$), both types evoke the same activation. **C:** Somatic potential of a simplified pyramidal neuron responding to signals sent from the input neuron (color scheme as in B). When a presynaptic neuron employs LE dynamics, postsynaptic voltage responds more quickly to changes in the input, leading to a faster convergence on a steady state.

lower for LE networks, as shown in Figure 2.3. These results highlight the superiority of LE for learning, as a fully trained network elicits almost no plasticity in its synapses. In contrast, the error terms in a Sacramento network drive synaptic plasticity even when the network has reached a state that exhibits low output loss and in which it is self-predicting. Thus, both the issue of inefficient and redundant plasticity, as well as the concern over response and learning speed can be solved by LE.

In addition to using the prospective **TODO: introduce term** somatic potential for the neuronal transfer function, it is also used in the plasticity rule. The Urbanczik-Senn plasticity is therefore updated to compute dendritic error from the prospective somatic potential $\dot{w}_{ij} = \eta \ (\phi(\breve{u}_i^{som}) - \phi(\hat{v}_i^{bas})) \ \phi(\breve{u}_j^{som})^T$. Much like for the transfer function, this change serves to increase the responsiveness of the network to input changes.

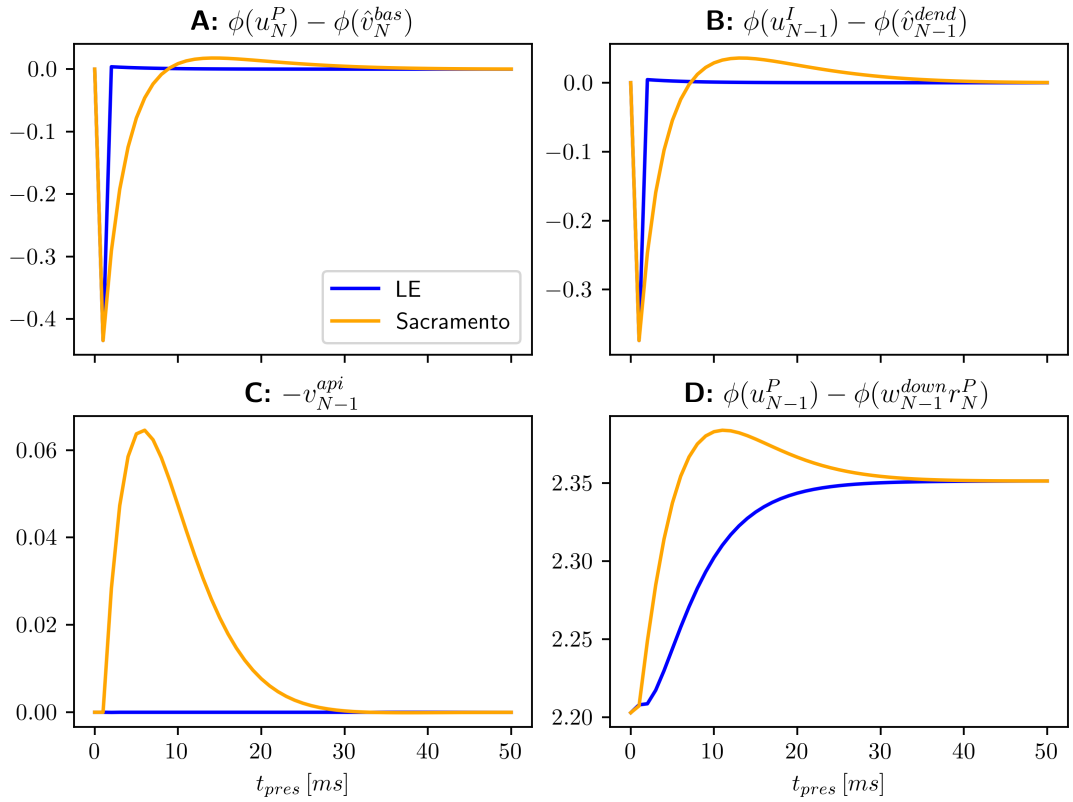

Figure 2.3: Comparison of stimulus presentation effects on the local error terms from Equations 2.13-2.16. Depicted are error terms for individual neurons of a network with one hidden layer ($N = 3$) that was fully trained on the Bars dataset **TODO: ref**. When employing Sacramento dynamics (orange), dendritic errors exhibit longer and more intense deviations, while errors in an identical LE network (blue) are cancelled out much sooner. **A:** Basal dendritic error for a pyramidal neuron at the output layer. **B:** Dendritic error for a hidden layer Interneuron. **C:** Proximal apical error for a hidden layer Pyramidal neuron. For this case the difference between the two networks is most striking, as errors for the LE network exhibit almost no deviation from rest. **D:** Distal apical error for the same pyramidal neuron. Note that this error term does not converge to zero for either network even after training is complete, which will be discussed **TODO: later**.

TODO: keep the following section? Besides the using a prediction of future somatic activity for neuronal transfer and plasticity, Haider et al. (2021) further alter the plasticity rule by means of their implementation. While the simulations by Sacramento et al. (2018) strictly conforms to the equations above, the new implementation (GitHub) . The fundamental building block of a network here is a layer, which is represented in code as an instance of the class `Layer`. Each instances holds information about its corresponding pyramidal- and interneurons. It also holds information about the synaptic connections between the two populations, as well as all incoming feedforward and feedback pyramidal synapses. A layer has two class methods that are fundamental to its computation; `update()` computes membrane potential derivatives and synaptic weight changes given pyramidal neuron activations from the previous and subsequent layers. `apply()` updates weights and membrane potentials from the previously calculated changes. Layers are processed in order from input to output layer, where all of them receive the `update()` signal first, before `apply()` is called on all of them. This ordering ensures, that changes in activation do not cascade through the layers and lead to excessive activations at the output. Yet, since next layer activations at time t have not been computed, top down information is always delayed by one timestep. **TODO: evaluate importance of this**

Thus, the equations for membrane updates change slightly:

$$\dot{W}_{ij}(t) = \eta(\phi(u_i) - \phi(\alpha v_i^{basal}(t)))\phi(u_j) \tag{2.43}$$

$$\Delta W_{ij}(t,T) = \int_t^T dt' \ \eta \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{2.44}$$

$$\Delta W_{ij}(t,T) = \eta \int_t^T dt' \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{2.45}$$

$$V_i^* = \phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}}) \tag{2.46}$$

$$s_j^* = \kappa_s * s_j \tag{2.47}$$

$$\phi(V^{som}) \to \phi(\breve{V}^{som}) \tag{2.48}$$

$$\breve{V} := V + \tau^m \dot{V} \tag{2.49}$$

$$\tag{2.50}$$

$$\frac{d}{dt}W_{ba} = \eta(\phi(V_b^{som}) - \phi(\alpha V_b^{dend}))\phi(V_a^{som}) \tag{2.51}$$

$$\frac{d}{dt}W_{ba} = \eta(\phi(\breve{V}_b^{som}) - \phi(\alpha \breve{V}_b^{dend}))\phi(\breve{V}_a^{som}) \tag{2.52}$$

2.10 rate neurons in NEST

2.11 simulation details/updates

All voltages need to be reset between simulations!

- injections into the network
- output layer readout
- interoperability of networks
-

Chapter 3

Results

TODO: Today, I changed the test criterion from the output neuron voltage at time t to a mean over a sample of the last $20ms$, network performance improved tremendously. Intuitively makes sense, but in particular it makes NEST networks diverge less after peak performance is reached. Are fluctuations in output layer activity increasing during late stages of training?

TODO: it might be worth experimenting with different synaptic delays in NEST in order to evaluate learning performance under biologically plausible transmission times. How easy this will assumably be to implement in NEST deserves note at this point.

TODO: talk about the fact that NEST synapses are updated, and SpikeEvents stored to ring buffers to be integrated into u_som after the synaptic delay. How much of physiological synaptic delays occurs pre- and postsynaptically in pyramidal neurons?

3.1 direct feedback connections to interneurons

Vaughn and Haas (2022); Mancilla et al. (2007)

3.2 Delayed presentation of instructive signals

A key concern of biologically plausible learning is the issue of when an instructive signal becomes available for training the network **TODO: cite or nah?**. When an organism is presented with a stimulus that requires a specific action, the organism responds to the best of its ability. Yet, the consequences of that response are likely to be delayed. **TODO: do we want this section?**

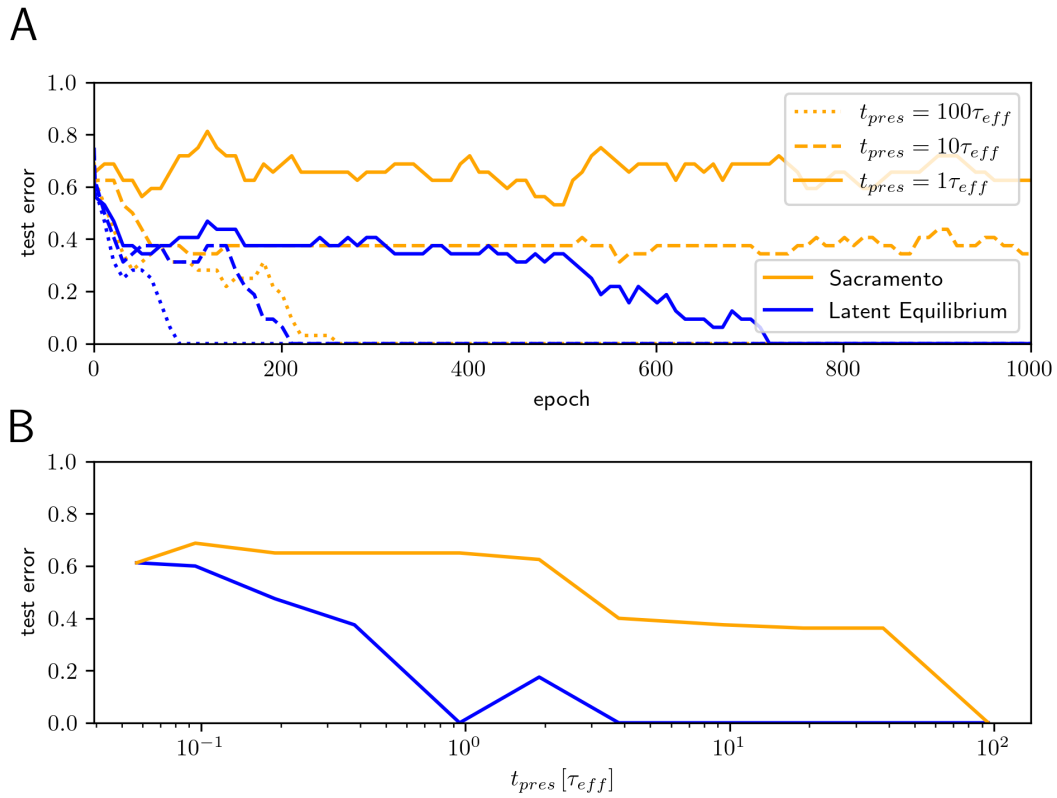

Figure 3.1: Replication of Figure 3 from Haider et al. (2021) using networks of spiking neurons in the NEST simulator. **A:** Comparison between Original pyramidal microcircuit network by Sacramento et al. (2018) and Latent equilibrium variant from Haider et al. (2021). Shown is the training of a network with 9-30-3 neurons on the 'Bars' Dataset from **TODO: describe it** with three different stimulus presentation times. **B:** Test performance after 1000 Epochs as a function of stimulus presentation time.

3.3 Presentation times with latent equilibrium

In order to validate the performance of my implementations, I replicated a parameter study from Haider et al. (2021)[Fig. 3]. The results for the NEST network using spiking neurons with default parameters **TODO: elaborate on this** are shown in Figure 3.1. A

Chapter 4

room for random observations

- When reducing the `weight_scale` parameter weights converge to lower absolute levels. I.e. mean abs weight drops from 0.5 to approx 0.1
- When reducing the size of the bar dataset by just 1 element, much simpler networks are capable of learning the task at lower t_pres. Failure to learn shows strange behaviour: the network fails to predict any sample of one group, with the group which it fails to represent switching every now and then.
- increasing apical C_m for spiking neurons was a straight banger.
- I switched input neurons for poisson_generators. This is faster, but will fuck up any simulations that rely on precise spike timing, since the generator redraws spikes for every target nrn.

Chapter 5

Discussion

5.1 Limitations of the implementation

Network needs to be reset between stimuli original does not do that, in NEST it's kindof a big deal.

exposure time and training set still quite large

non-resetting, non-refractory

5.2 Whittington and Bogacz criteria

In which we discuss, to what extent the network conforms to the criteria for biologically plausible learning rules introduced by Whittington and Bogacz (2017):

1. Local computation. A neuron performs computation only on the basis of the inputs it receives from other neurons weighted by the strengths of its synaptic connections.

2. Local plasticity. The amount of synaptic weight modification is dependent on only the activity of the two neurons the synapse connects (and possibly a neuromodulator).

3. Minimal external control. The neurons perform the computation autonomously with as little external control routing information in different ways at different times as possible.

4. Plausible architecture. The connectivity patterns in the model should be consistent with basic constraints of connectivity in neocortex.

5.3 Should it be considered pre-training?

TODO: someone said this network needs pre-training and that made me sad :(

5.4 Relation to energy minimization

5.5 Outlook

This would likely be super efficient on neuromorphics!

I am not going to try plasticity with spike-spike or spike-rate dendritic errors

Training on ImageNet

Making this shit faster

reward-modulated urbanczik-senn plasticity?

Chapter 6

Appendix

6.1 Somato-dendritic coupling

Urbanczik and Senn (2014) discuss a possible extension to their neuron- and plasticity model, in which the dendro-somatic coupling transmits voltages in both directions. They show that the plasticity rule requires only minor adaptations for successful learning under this paradigm. Yet, as described by passive cable theory, the flow between neuronal compartments is dictated by their respective membrane capacitances. These are calculated from their membrane areas, which vastly differ in the case of pyramidal neurons. **TODO: find a nice citation for this**

15,006 458

will not be considered here. The motivation is, that dendritic membrane area is

6.2 Integration of the spike-based Urbanczik-Senn plasticity

Starting with the complete Integral from $t = 0$.

$$\Delta W_{ij}(0,t) = \eta \int_0^t dt' \int_0^{t'} dt'' \ \kappa(t'-t'')V_i^*(t'')s_j^*(t'') \tag{6.1}$$

$$= \eta \int_0^t dt'' \int_{t''}^t dt' \ \kappa(t'-t'')V_i^*(t'')s_j^*(t'') \tag{6.2}$$

$$= \eta \int_0^t dt'' \ \left[\tilde{\kappa}(t-t'') - \tilde{\kappa}(0)\right] V_i^*(t'')s_j^*(t'') \tag{6.3}$$

$$\tag{6.4}$$

With $\tilde{\kappa}$ being the antiderivative of κ:

$$\kappa(t) = \frac{\delta}{\delta t}\tilde{\kappa}(t) \tag{6.5}$$

$$\tilde{\kappa}(t) = -e^{-\frac{t}{t_\kappa}} \tag{6.6}$$

$$\tag{6.7}$$

The above can be split up into two separate integrals:
Which implies the identities

$$I_1(t_1, t_2 + \Delta t) = I_1(t_1, t_2) + I_1(t_2, t_2 + \Delta t) \tag{6.8}$$

$$I_2(t_1, t_2 + \Delta t) = e^{-\frac{t_2 - t_1}{\tau_\kappa}} I_2(t_1, t_2) + I_2(t_2, t_2 + \Delta t) \tag{6.9}$$

$$I_2(t_1, t_2 + \Delta t) = -\int_{t_1}^{t_2+\Delta t} dt' \ \tilde{\kappa}(t_2 + \Delta t - t') V_i^*(t') s_j^*(t') \tag{6.10}$$

$$= -\int_{t_1}^{t_2} dt' \ \left[-e^{-\frac{t_2+\Delta t-t'}{\tau_\kappa}}\right] V_i^*(t') s_j^*(t') - \int_{t_2}^{t_2+\Delta t} dt' \ \left[-e^{-\frac{t_2+\Delta t-t'}{\tau_\kappa}}\right] V_i^*(t') s_j^*(t') \tag{6.11}$$

$$= -e^{-\frac{\Delta t}{\tau_\kappa}} \int_{t_1}^{t_2} dt' \ \left[-e^{-\frac{t_2-t'}{\tau_\kappa}}\right] V_i^*(t') s_j^*(t') - \int_{t_2}^{t_2+\Delta t} dt' \ \left[-e^{-\frac{t_2+\Delta t-t'}{\tau_\kappa}}\right] V_i^*(t') s_j^*(t') \tag{6.12}$$

Using this we can rewrite the weight change from t to T as:

$$\Delta W_{ij}(t, T) = \Delta W_{ij}(0, T) - \Delta W_{ij}(0, t) \tag{6.13}$$

$$= \eta[-I_2(0, T) + I_1(0, T) + I_2(0, t) - I_1(0, t)] \tag{6.14}$$

$$= \eta[I_1(t, T) - I_2(t, T) + I_2(0, t)\left(1 - e^{-\frac{T-t}{\tau_\kappa}}\right)] \tag{6.15}$$

The simplified Sacramento et al. (2018) case would be:

$$\frac{dW_{ij}}{dt} = \eta(\phi(u_i) - \phi(\hat{v}_i))\phi(u_j) \tag{6.16}$$

$$\Delta W_{ij}(t,T) = \int_t^T dt' \ \eta \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{6.17}$$

$$\Delta W_{ij}(t,T) = \eta \int_t^T dt' \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{6.18}$$

$$V_i^* = \phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}}) \tag{6.19}$$

$$s_j^* = \kappa_s * s_j \tag{6.20}$$

Where s_i is the postsynaptic spiketrain and V_i^* is the error between dendritic prediction and somatic rate and $h(u)$. The additional nonlinearity $h(u) = \frac{d}{du} ln \ \phi(u)$ is ommited in our model **TODO: should it though?**.

Antiderivatives:

$$\int_{-\infty}^{x} H(t)dt = tH(t) = max(0,t) \tag{6.21}$$

$$\tau_l = \frac{C_m}{g_L} = 10 \tag{6.22}$$

$$\tau_s = 3 \tag{6.23}$$

Writing membrane potential to history (happens at every update step of the postsynaptic neuron):

```

UrbanczikArchivingNode< urbanczik_parameters >::write_urbanczik_history(Time t
    , double V_W, int n_spikes, int comp)
{
  double V_W_star = ( ( E_L * g_L + V_W * g_D ) / ( g_D + g_L ) );
  double dPI = ( n_spikes - phi( V_W_star ) * Time::get_resolution().get_ms()
    )
      * h( V_W_star );
}
```

I interpret this as:

$$\int_{t_{ls}}^{T} dt' \; V_i^* = \int_{t_{ls}}^{T} dt' \; (s_i - \phi(V_i))h(V_i), \tag{6.24}$$

$$\int_{t_{ls}}^{T} dt' \; V_i^* = \sum_{t=t_{ls}}^{T} (s_i(t) - \phi(V_i^t)\Delta t)h(V_i^t) \tag{6.25}$$

$$\tag{6.26}$$

```
for (t = t_ls; t< T; t = t + delta_t)
{
    minus_delta_t = t_ls - t;
    minus_t_down = t - T;
    PI = ( kappa_l * exp( minus_delta_t / tau_L ) - kappa_s * exp(
    minus_delta_t / tau_s ) ) * V_star(t);
    PI_integral_ += PI;
    dPI_exp_integral += exp( minus_t_down / tau_Delta_ ) * PI;
}
// I_2 (t,T) = I_2(0,t) * exp(-(T-t)/tau) + I_2(t,T)
PI_exp_integral_ = (exp((t_ls-T)/tau_Delta_) * PI_exp_integral_ +
    dPI_exp_integral);
W_ji = PI_integral_ - PI_exp_integral_;
W_ji = init_weight_ + W_ji * 15.0 * C_m * tau_s * eta_ / ( g_L * ( tau_L -
    tau_s ) );

kappa_l = kappa_l * exp((t_ls - T)/tau_L) + 1.0;
kappa_s = kappa_s * exp((t_ls - T)/tau_s) + 1.0;

```

$$\int_{t_{ls}}^{T} dt' s_j^* = \tilde{\kappa_L}(t') * s_j - \tilde{\kappa_s}(t') * s_j \tag{6.27}$$

I_1 in the code is computed as a sum:

$$I_1(t,T) = \sum_{t'=t}^{T} (s_L^*(t') - s_s^*(t')) * V^*(t') \tag{6.28}$$

6.3 Dendritic leakage conductance

In order to match neuron dynamics between the rate and spiking neuron implementations as closely as possible, a suitable leakage conductance for dendritic compartments was required. For simplicity, we now consider a single synaptic connection from neuron i to neuron j with weight W_{ji} and attempt to identify an appropriate parameter. As described in Equation 2.20,

a dendritic compartment evolves according to: **TODO: expand to multiple presynaptic neurons?**

$$C_m^{dend} \dot{v}_j^{dend} = -g_l^{dend} \; v_j^{dend} + W_{ji} \; \langle n_i \rangle \tag{6.29}$$

Under the assumption that the activation of neuron i remains static over time, we can replace the spontaneous activation $s_i(t)$ with the expected number of spikes per simulation step $\langle n_i \rangle = r_i \; \Delta t$. Next, in order to find the convergence point of the ODE, we set the left side of the equation to 0 and attempt to solve it:

$$0 = -g_l^{dend} \; v_j^{dend} + W_{ji} \; r_i \; \Delta t \tag{6.30}$$

$$g_l^{dend} \; v_j^{dend} = W_{ji} \; r_i \; \Delta t \tag{6.31}$$

The desired effect of presynaptic activation on the postsynaptic dendritic potential is described in Equation 2.4 ($v_j^{dend} = W_{ji} \; r_i$), which occurs on both sides of the above equation. Assuming the desired equalitiy, both terms to drop out from the equation. Thus, the correct parametrization for the dendritic leakage conductance is:

$$g_l^{dend} = \Delta t \tag{6.32}$$

It was shown experimentally, that for hight spike frequencies, this parameterization leads to an exact match of dendritic potentials between neuron models. This value for the dendritic leakage conductance will be assumed to be the default throughout all experiments where spiking neurons are used.

In order to keep the NEST models as similar as possible, rate neurons evolve according to the same dynamics, with the difference that they transmit their activation at every timestep. Like the original implementation, dendrites of rate neurons do not evolve gradually, but are fully defined by presynaptic at time t [1]. This behaviour is acheived by setting the leakage conductance to 1 for all dendritic compartments. During network initialization, dendritic leakage conductances are set to either one of these values depending on the type of network to be trained.

[1] Due to the synaptic delay enforced by the architecture of NEST, this is not entirely correct. Dendritic potentials at time t are computed from presynaptic activity at time $t - \Delta t$, i.e. the previous simulation step.

6.4 Plasticity in feedback connections

In the model by Sacramento et al. (2018), Pyramidal-to-pyramidal feedback weights evolve according to:

$$\dot{w}_l^{down} = \eta_l^{down} \left(\phi(u_l^P) - \phi(w_l^{down} r_{l+1}^P)\right) \phi(u_{l+1}^P)^T \tag{6.33}$$

The error term in this case on first sight differs greatly from the original Urbanczik-Senn plasticity rule. Yet it serves a purpose for learning and could arguably be implemented by biological neurons. An intuitive way to interpret the error term is as the difference between somatic activity and the apical activity that can be traced back to superficial pyramidal neurons. In a biological neuron, this second part could plausibly be encoded by the voltage of a distal apical compartment which is innervated only by top-down connections and leaks into the proximal apical compartment (v^{api}) with a conductance of 1. The separation of pyramidal neuron apical dendrites into a proximal and a distal tree is well documented **TODO: cite**. A difference between plasticity mechanisms for synapses arriving at these two integration zones is plausible, although I was unable to find prior research supporting this type of plasticity **TODO: cite**.

In the NEST implementation, the neuron model contains a distal apical compartment which is innervated exclusively by top-down feedback connections. For simplicity, this compartment has no complex connection to the other compartments. Instead, all events sent to it are duplicated and sent to the proximal apical compartment as well. This allows for convenient readout of the aforementioned error term for plasticity without affecting the neuron dynamics for simualtions where this compartment serves no purpose. A more sophisticated model of the apical tree which resembles pyramidal neurons more closely could be a desirable extension to this network.

6.5 Presentation times and Latent Equilibrium

Exactly matching parameters and the training environment to those of existing implementations turned out to be a significant challenge. Particularly the way NEST handles signal transmissions made and exact numerical replication of results impossible, as discussed in Section **TODO: talk about timing differences**. In order to validate, that

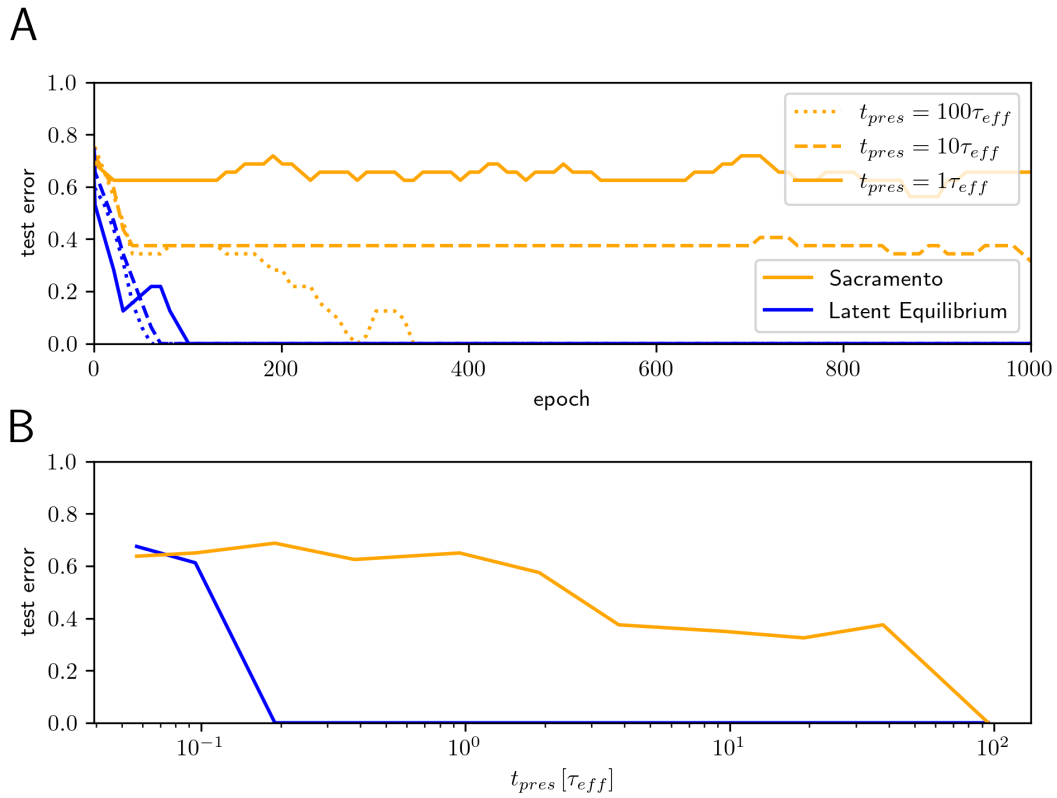

Figure 6.1: Replication of Figure 3.1 using a re-implementation of the Haider et al. (2021) code in python. Resulting performance matches the original results exactly, proving that my replication can serve as a baseline for comparing the NEST implementation

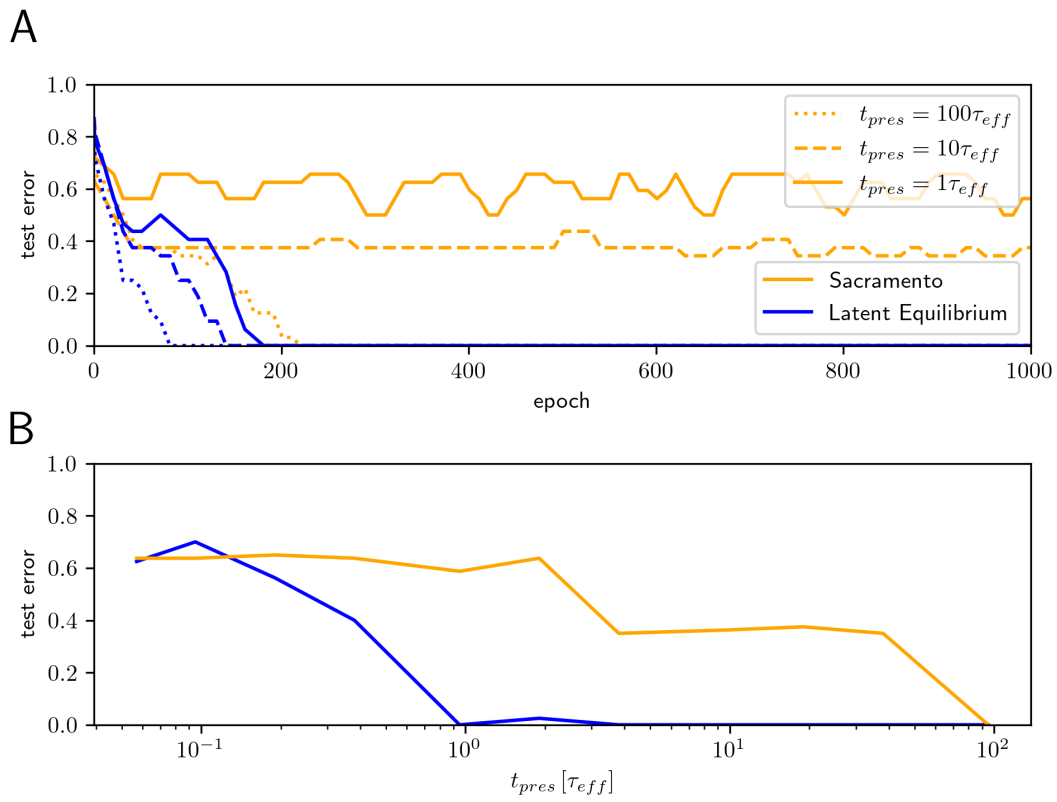

Figure 6.2: Replication of Figure 3.1 using networks of rate neurons in the NEST simulator.

1. In the torch implementation, there no persistence between timesteps at all. Input is fed into the network and processed feedforward and feedback. Output is read and weights (+biases) are updated. Rinse and repeat.
2. to what extent should dendritic and somatic compartments decay?
3. Can (should) we transfer the learned bias from the torch model?
4. I can "cheat" the apical voltage constraint for self prediction by increasing apical leakage conductance. How does this influence my model?
5. Is there some analytical approach to identifying why synaptic weights deviate from their intended targets?
6. I think that lambda needs to be scaled in dependence on g_{lk}, such that current inputs, spike inputs and leakage cancel each other out.
7. How do we deal with population size dependence?

6.6 Parameter study

- Transfer function ϕ
- interneuron mixing factor λ
- injected current I_e
- dendritic leakage conductance $g_{lk,d}$
- somatic leakage conductance $g_{lk,s}$
- Learning rate η
- synaptic time constants τ_{delta}
- noise level σ
- Simulation time $\mathbb{T}$
- plasticity onset after the network relaxes
- compartment current decay τ_{syn}

Observations

- In self-predicting paradigm, Apical errors stay constant, despite interneuron error steadily increasing.
- Interneuron error (between neuron pairs) is proportional to absolute somatic voltage in self-predicting paradigm.
- abs interneuron voltage is always higher than abs pyramidal voltage. This kind of makes sense, as interneurons receive direct current input proportional to pyramidal voltage in addition to feedforward input. This discrepancy disappears when setting λ to 0 as expected.
- When plasticity is enabled from a random starting configuration, apical error **sometimes** converges to better values than can be achieved in both self-predicting paradigms. I believe this to be a huge issue: the self-predicting state does not cause minimal apical voltage, and completely decayed feedback weights are preferable to perfectly counteracting feedback weights.
- feedforward weights tend to increase absolutely, i.e. drift towards the closest extreme. *This only happens since I re-implemented the second exponential term in the pyr_synapse.* Yet they do not simply explode to the nearest extremum, but will traverse a zero weight to reach the maximum with equal sign as the weight they are supposed to match.
- feedback weights tend to decay to around zero. Yet they appear to remain close to zero in the direction they are supposed to be.
- Idea: I think that the somatic nudging is handled as straight currents being sent to the neuron, instead of the difference between actual and desired somatic voltage.
- In the paper and Mathematica code, Feedback learning rate is 2-5 times higher than feedforward lr. In my model, for learning to happen on similar time scales, feedback lr has to be 100 times lower than feedforward lr. An indicator that my plasticity is messed up.
- The simulation is likely producing way too few spikes (5-20 per 1000ms iteration). Could adapting the activation function yield better results?
- In the Mathematica solution, leakage conductance is greater than 1! ($\delta U_i = -(g_L + g_D + g_{SI})U_i + g_D V_{BI} + g_{SI} U_Y$) with $g_L + g_D + g_{SI} = 1.9$

Chapter 7

Preliminary structural components

7.1 Synaptic delays

Where I will inspect the implications of synaptic delays inherent to the NEST simulations on the model and plasticity rule. In particular, I will look at the biological necessity for this type of delay and discuss why any model attempting to replicate neuronal processes must be resilient to these delays.

7.2 Literature review - Backpropagation in SNN

Where I will review other attempts at implementing biologically plausible Backpropagation alternatives and contrast them to the current model.

7.3 NEST Urbanczik-senn implementation

7.4 My neuron model

- Low pass filtering
- multi-compartment computation
- Imprecision of the ODE
- abuse of the somatic conductance

7.4.1 NEST rate neuron shenanigans

Given how long I worked on a rate neuron implementation in NEST, some pages should be devoted to this effort.

7.5 My synapse model

Where I discuss the synapse implementation with regard to multi-compartment neurons, urbanczik-archiving and in particular the issues with timing that arise from NEST delays.

7.6 The relation between the pyramidal microcircuit and actual microcircuits

Where I can finally use the shit that has been on my whiteboard for half a year...

This will also serve as valuable insight into how plausible this microcircuit actually is, and might give some insight into possible model extensions.

7.6.1 Interneurons and their jobs

7.7 Does it have to be backprop?

Where I will explain my concerns regarding the usefulness of approximating backpropagation in light of the substantial one-shot learning capability of the brain and the active inference model.

7.8 Discrepancies between mathematica and NEST model

1. Weights deviate slightly. This difference can be alleviated by exposing a single stimulus for a longer duration before switching.

7.9 Transfer functions

Where I will discuss the sensitivity of this entire simulation to minor changes in the parametrization and style of transfer function being used.

Chapter 8

The weight-leakage tradeoff

Where I will discuss the issue, that decreasing both synaptic weights and dendritic leakage conductance lead to more stability in the dendritic voltage, while at the same time requiring longer exposure per iteration.

TODOs

- Prove that the network is stable in the self-predicting state and at the end of learning
- Show the limits of learning capability (i.e. how big of a network it can match)
- Test the network on a real-world dataset (mnist)
- prove/find literature on why the poisson process is a rate neuron in the limit.
- Does the network still learn when neurons have a refractory period?
- Comparison to other spiking backprops
- what can we learn from this? does it describe part of the brain

Chapter 9

Open Questions

- Any tips for transitioning to large simulation? also regarding the threadripper
- Is refractoryness interesting to us or more of a sidenote?
- Neuron dropout?
- How does one prove that the network is converged and will not diverge again.
- randomized/longer synaptic delays?
- As a follow up of dropout, maybe even neurogenesis?
- Should I look at delaying injection of the target activation?
- more ways in which this is biologically implausible?

$t_{pres} 10 - 50\tau$

Bibliography

Bartunov, S., Santoro, A., Richards, B., Marris, L., Hinton, G. E., and Lillicrap, T. (2018). Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *Advances in neural information processing systems*, 31.

Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., and Lin, Z. (2015). Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*.

Crick, F. (1989). The recent excitement about neural networks. *Nature*, 337(6203):129–132.

Guerguiev, J., Lillicrap, T. P., and Richards, B. A. (2017). Towards deep learning with segregated dendrites. *ELife*, 6:e22901.

Haider, P., Ellenberger, B., Kriener, L., Jordan, J., Senn, W., and Petrovici, M. A. (2021). Latent equilibrium: A unified learning theory for arbitrarily fast computation with arbitrarily slow neurons. *Advances in Neural Information Processing Systems*, 34:17839–17851.

Liao, Q., Leibo, J., and Poggio, T. (2016). How important is weight symmetry in backpropagation? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Mancilla, J. G., Lewis, T. J., Pinto, D. J., Rinzel, J., and Connors, B. W. (2007). Synchronization of electrically coupled pairs of inhibitory interneurons in neocortex. *Journal of Neuroscience*, 27(8):2058–2073.

McCormick, D. A., Connors, B. W., Lighthall, J. W., and Prince, D. A. (1985). Comparative electrophysiology of pyramidal and sparsely spiny stellate neurons of the neocortex. *Journal of neurophysiology*, 54(4):782–806.

Millidge, B., Tschantz, A., Seth, A. K., and Buckley, C. L. (2020). Activation relaxation: A local dynamical approximation to backpropagation in the brain. *arXiv preprint arXiv:2009.05359*.

Morrison, A., Diesmann, M., and Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biological cybernetics*, 98(6):459–478.

Potjans, T. C. and Diesmann, M. (2014). The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. *Cerebral cortex*, 24(3):785–806.

Sacramento, J., Ponte Costa, R., Bengio, Y., and Senn, W. (2018). Dendritic cortical microcircuits approximate the backpropagation algorithm. *Advances in neural information processing systems*, 31.

Stapmanns, J., Hahne, J., Helias, M., Bolten, M., Diesmann, M., and Dahmen, D. (2021). Event-based update of synapses in voltage-based learning rules. *Frontiers in neuroinformatics*, page 15.

Urbanczik, R. and Senn, W. (2014). Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528.

Vaughn, M. J. and Haas, J. S. (2022). On the diverse functions of electrical synapses. *Frontiers in Cellular Neuroscience*, 16.

Whittington, J. C. and Bogacz, R. (2017). An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural computation*, 29(5):1229–1262.

Whittington, J. C. and Bogacz, R. (2019). Theories of error back-propagation in the brain. *Trends in cognitive sciences*, 23(3):235–250.

Yamins, D. L. and DiCarlo, J. J. (2016). Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365.