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.

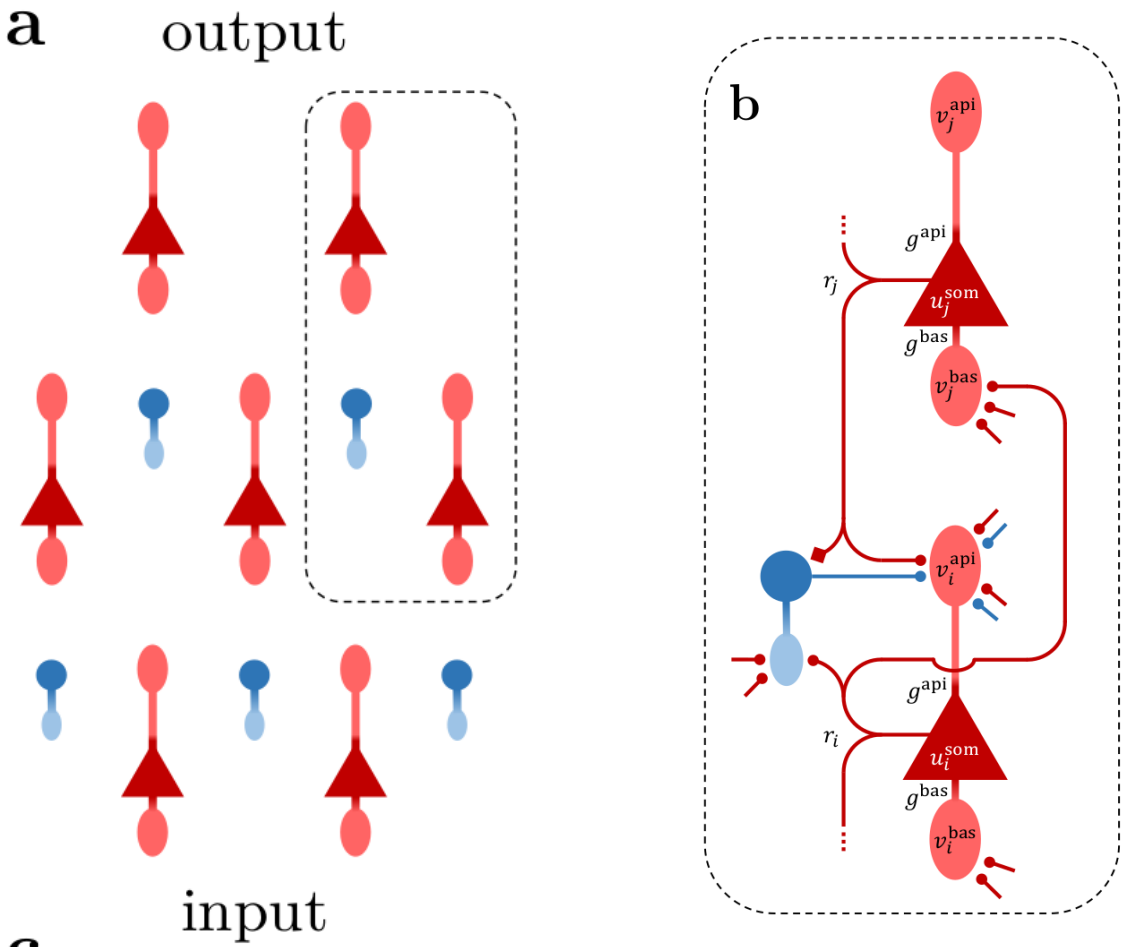

Figure 2.1: Network structure, from Haider et al. (2021). **a:** pyramidal- (red) and interneurons (blue) in a network of three layers. Note the fact that the number interneurons in a layer is equal to the number of pyramidal neurons in the subsequent layer[1]. **b:** connectivity within the highlighted 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 in contrast transmit the neuron activation $\phi(u^{som})$ and fully connect their 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. 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. Note that vector notation is used here, 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.

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

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

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_i^{bas} and v_i^{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 computations, 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.

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

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. The extension in which the somatic potential also affects dendrites, will be discussed in

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 error). Urbanczik and Senn (2014) show, that **TODO: what?**

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

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

Note that pyramidal-to-pyramidal feedback weights w_l^{down} are not plastic in most simulations, and are only listed here for the sake of completeness **TODO: some notes in the appendix to this?**

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 ($w_l^{ip} \approx \omega w_l^{up}$) up to some scaling factor ω. ω 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 plastic-

ity 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 four features:

- The symmetries between feedforward ($w_l^{ip} \approx \omega w_l^{up}$) and feedback ($w_l^{pi} \approx -w_l^{down}$) weights.
- Inactivity in pyramidal neuron apical compartments ($v_l^{api} \approx 0$)
- Equal activations in interneurons and their respective sister neurons ($\phi(u_l^I) \approx \phi(u_{l+1}^P)$)

All of these equations are approximate, since the network does not truly

Note, how all feedback

For feedforward pyramidal-to-pyramidal

In this architecture, plasticity serves two purposes. For pyramidal-to-pyramidal feedforward synapses, it implements error-correcting learning as a time-continuous approximation of BP. For pyramidal-to- interneuron synapses, it drives interneurons to mimic their pyramidal partners in the layers above (see also SI). Thus, in a well-trained network, apical compartments of pyramidal cells are at rest, reflecting zero error, as top-down and lateral inputs cancel out. When an output error propagates through the network, these two inputs can no longer cancel out and their difference represents the local error ei . This architecture does not rely on the transpose of the forward weight matrix, improving viability for implementation in distributed asynchronous systems. Here, we keep feedback weights fixed, realizing a variant of feedback alignment. In principle, these weights could also be learned in order to further improve the local representation of errors Section 7.

[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.

2.4 Training the network

2.5 NEST implementation

One of the key research questions motivating this thesis is whether the proposed architecture would be able to learn successfully under a spiking neuron paradigm.

To calculate a number of spikes from the rate $r = \phi(u)$, a poisson point process was used:

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

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

with number of spikes n and simulation timestep Δt, which will be assumed to be $0.1ms$ from here on out. Within the code, true number of spikes during Δt is drawn from a poisson distribution with parameter $\langle n \rangle$. 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 handles such high rates without issue. Yet for the analysis with regard to biological plausibility, spiking behaviour with refractory periods might be useful. The probability of at least one spike occuring within the next simulation step thus is the inverse probability of no spike occuring. Entering $n = 0$ into Equation 2.17, the probability of eliciting at least one spike within the next simualtion step becomes:

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

After randomly drawing from this probability a spike is sent, and the spiking probability is set to 0 for the duration of the refractory period t_{ref}.

2.6 Event-based Urbanczik-Senn plasticity

One major challenge in implementing this architecture with spiking neurons is the Urbanczik-Senn plasticity. Since it depends on pre- and postsynaptic information for all updates, one of two approaches is typically employed to compute it:

One option is to store all relevant state variables for every timestep in the synapse and update the synaptic weight whenever a spike traverses the synapse. This is efficient in terms of computation time, since fewer computations are required per synapse. The major downside of the approach lies in memory efficiency; Every synapse potentially stores the history over several hundred simulation steps. In addition, every synapse innervating a given neuron would store its membrane potentials redundantly.

Another approach updates synaptic weights at every timestep. This requires very little memory, but substantially increases computation time. Importantly, it is incompatible with asynchronous neuron simulations, as all synapses and their weights need to be updated at every simulation step. In particular for large networks and modern hardware solutions, asynchronous simulations have proven to be more efficient for most use cases **TODO: cite something by abigail**

Thus, a novel approach was developed by Stapmanns et al. (2021) for the NEST simulator, which will be explained in this section. In order to utilize the paralellization capabilities of NEST and spiking neural networks, transmitting the pre- and postsynaptic membrane potential is a poor choice. Thus, the error between somatic and dendritic activation is stored in the `urbanczik_history` of the postsynaptic neuron. Since it applies equally to all incoming synapses, redundant storage is avoided by storing a counter alongside each entry

$$\kappa(t) = H(t)e^{-t/\tau_\kappa} \tag{2.20}$$

$$H(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t \leq 0 \end{cases} \tag{2.21}$$

Antiderivatives:

$$\int_{-\infty}^{x} H(t)dt = tH(t) = max(0,t) \tag{2.22}$$

Convolution:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.23}$$

For functions f, g supported on only $[0, \infty]$ (as one-sided decay kernels and spiketrains are), integration limits can be truncated:

$$(f * g)(t) = \int_{0}^{t} f(\tau)g(t-\tau)d\tau \tag{2.24}$$

$$\tag{2.25}$$

Plasticity:

$$\frac{dW_{ij}}{dt}(t) = F(W_{ij}(t), s_i^*(t), s_j^*(t), V_i^*(t)) \tag{2.26}$$

$$F[s_j^*, V_i^*] = \eta\,\kappa * (V_i^* s_j^*) \tag{2.27}$$

$$\text{with } V_i^* = (s_i - \phi(V_i))h(V_i), \tag{2.28}$$

$$s_j^* = \kappa_s * s_j. \tag{2.29}$$

For an event-based plasticity we need:

$$\Delta W_{ij}(t,T) = \int_t^T dt' F[s_j^*, V_i^*](t') \tag{2.30}$$

$$= \int_t^T dt' \eta\,\kappa * (V_i^* s_j^*) \tag{2.31}$$

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

$$= \eta \int_0^t dt' \int_{t''}^{t'} dt''\ \kappa(t'-t'')V_i^*(t'')s_j^*(t'') \tag{2.33}$$

$$\tag{2.34}$$

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

$$\Delta W_{ij}(0,t) = \eta \int_0^t dt' \int_0^{t'} dt''\ \kappa(t'-t'')V_i^*(t'')s_j^*(t'') \tag{2.35}$$

$$= \eta \int_0^t dt'' \int_{t''}^{t} dt'\ \kappa(t'-t'')V_i^*(t'')s_j^*(t'') \tag{2.36}$$

$$= \eta \int_0^t dt''\ \left[\tilde{\kappa}(t-t'') - \tilde{\kappa}(0)\right] V_i^*(t'')s_j^*(t'') \tag{2.37}$$

$$\tag{2.38}$$

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

$$\kappa(t) = \frac{\delta}{\delta t}\tilde{\kappa}(t) \tag{2.39}$$

$$\tilde{\kappa}(t) = -e^{-\frac{t}{t_\kappa}} \tag{2.40}$$

$$\tag{2.41}$$

The above can be split up into two separate integrals:

$$\Delta W_{ij}(0,t) = \eta\left[-I_2(0,t) + I_1(0,t)\right] \tag{2.42}$$

$$I_1(t_1,t_2) = -\int_{t_1}^{t_2} dt' \;\; \tilde{\kappa}(0) V_i^*(t') s_j^*(t') \tag{2.43}$$

$$I_2(t_1,t_2) = -\int_{t_1}^{t_2} dt' \;\; \tilde{\kappa}(t_2 - t') V_i^*(t') s_j^*(t') \tag{2.44}$$

$$\tag{2.45}$$

Which implies the identities

$$I_1(t_1, t_2 + \Delta t) = I_1(t_1,t_2) + I_1(t_2, t_2 + \Delta t) \tag{2.46}$$

$$I_2(t_1, t_2 + \Delta t) = e^{-\frac{t_2 - t_1}{\tau_\kappa}} I_2(t_1,t_2) + I_2(t_2, t_2 + \Delta t) \tag{2.47}$$

$$I_2(t_1, t_2 + \Delta t) = -\int_{t_1}^{t_2+\Delta t} dt' \;\; \tilde{\kappa}(t_2 + \Delta t - t') V_i^*(t') s_j^*(t') \tag{2.48}$$

$$= -\int_{t_1}^{t_2} dt' \;\; \left[-e^{-\frac{t_2 + \Delta t - t'}{\tau_\kappa}}\right] V_i^*(t') s_j^*(t') - \int_{t_2}^{t_2+\Delta t} dt' \;\; \left[-e^{-\frac{t_2 + \Delta t - t'}{\tau_\kappa}}\right] V_i^*(t') s_j^*(t') \tag{2.49}$$

$$= -e^{-\frac{\Delta t}{\tau_\kappa}} \int_{t_1}^{t_2} dt' \;\; \left[-e^{-\frac{t_2 - t'}{\tau_\kappa}}\right] V_i^*(t') s_j^*(t') - \int_{t_2}^{t_2+\Delta t} dt' \;\; \left[-e^{-\frac{t_2 + \Delta t - t'}{\tau_\kappa}}\right] V_i^*(t') s_j^*(t') \tag{2.50}$$

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

$$\Delta W_{ij}(t,T) = \Delta W_{ij}(0,T) - \Delta W_{ij}(0,t) \tag{2.51}$$

$$= \eta[-I_2(0,T) + I_1(0,T) + I_2(0,t) - I_1(0,t)] \tag{2.52}$$

$$= \eta[I_1(t,T) - I_2(t,T) + I_2(0,t)\left(1 - e^{-\frac{T-t}{\tau_\kappa}}\right)] \tag{2.53}$$

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

$$\frac{dW_{ij}}{dt} = \eta(\phi(u_i) - \phi(\hat{v}_i))\phi(u_j) \tag{2.54}$$

$$\Delta W_{ij}(t,T) = \int_t^T dt' \ \eta \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{2.55}$$

$$\Delta W_{ij}(t,T) = \eta \int_t^T dt' \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{2.56}$$

$$V_i^* = \phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}}) \tag{2.57}$$

$$s_j^* = \kappa_s * s_j \tag{2.58}$$

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?** .

$$\tau_l = \frac{C_m}{g_L} = 10 \tag{2.59}$$

$$\tau_s = 3 \tag{2.60}$$

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

```
1
2 UrbanczikArchivingNode< urbanczik_parameters >::write_urbanczik_history(Time t
    , double V_W, int n_spikes, int comp)
3 {
4   double V_W_star = ( ( E_L * g_L + V_W * g_D ) / ( g_D + g_L ) );
5   double dPI = ( n_spikes - phi( V_W_star ) * Time::get_resolution().get_ms()
    )
6       * h( V_W_star );
7 }
```

I interpret this as:

$$\int_{t_{ls}}^T dt' \ V_i^* = \int_{t_{ls}}^T dt' \ (s_i - \phi(V_i))h(V_i), \tag{2.61}$$

$$\int_{t_{ls}}^T dt' \ V_i^* = \sum_{t=t_{ls}}^T (s_i(t) - \phi(V_i^t)\Delta t)h(V_i^t) \tag{2.62}$$

$$\tag{2.63}$$

```
1 for (t = t_ls; t< T; t = t + delta_t)
2 {
```

```
    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;

```

$$\int_{t_{ls}}^{T} dt' s_j^* = \tilde{\kappa_L}(t') * s_j - \tilde{\kappa_s}(t') * s_j \tag{2.64}$$

I_1 in the code is computed as a sum:

$$I_1(t,T) = \sum_{t'=t}^{T} (s_L^*(t') - s_s^*(t')) * V^*(t') \tag{2.65}$$

2.7 steady-state potentials in Sacramento (2018)

$$u_k^p = \frac{g_B}{g_{lk} + g_B + g_A} v_{B,k}^P + \frac{g_A}{g_{lk} + g_B + g_A} v_{A,k}^P \tag{2.66}$$

$$\hat{v}_{B,k}^P = \frac{g_B}{g_{lk} + g_B + g_A} v_{B,k}^P \tag{2.67}$$

$$\hat{v}_k^I = \frac{g_B}{g_{lk} + g_B} v_k^I \tag{2.68}$$

$$\lambda = \frac{g_{som}}{g_{lk} + g_B + g_{som}} \tag{2.69}$$

2.8 Haider 2021 updates

Besides the using a prediction of future somatic activity for neuronal transfer and plasticity, Haider et al. (2021) also slightly alter the plasticity rule, which turns out to be crucial for the latent equilibrium mechanism. While the simulations by Sacramento et al. (2018) use the equations above, the new implementation fill with neurips link implicitly change them within

its code. 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:

$$\dot{W}_{ij}(t) = \eta(\phi(u_i) - \phi(\alpha v_i^{basal}(t)))\phi(u_j) \tag{2.70}$$

$$\Delta W_{ij}(t,T) = \int_t^T dt' \ \eta \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{2.71}$$

$$\Delta W_{ij}(t,T) = \eta \int_t^T dt' \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{2.72}$$

$$V_i^* = \phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}}) \tag{2.73}$$

$$s_j^* = \kappa_s * s_j \tag{2.74}$$

2.9 Latent equilibrium

$$\phi(V^{som}) \to \phi(\breve{V}^{som}) \tag{2.75}$$

$$\breve{V} := V + \tau^m \dot{V} \tag{2.76}$$

$$\tag{2.77}$$

$$\frac{d}{dt}W_{ba} = \eta(\phi(V_b^{som}) - \phi(\alpha V_b^{dend}))\phi(V_a^{som}) \tag{2.78}$$

$$\frac{d}{dt}W_{ba} = \eta(\phi(\breve{V}_b^{som}) - \phi(\alpha \breve{V}_b^{dend}))\phi(\breve{V}_a^{som}) \tag{2.79}$$

2.10 simulation details/updates

All voltages need to be reset between simulations!

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 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

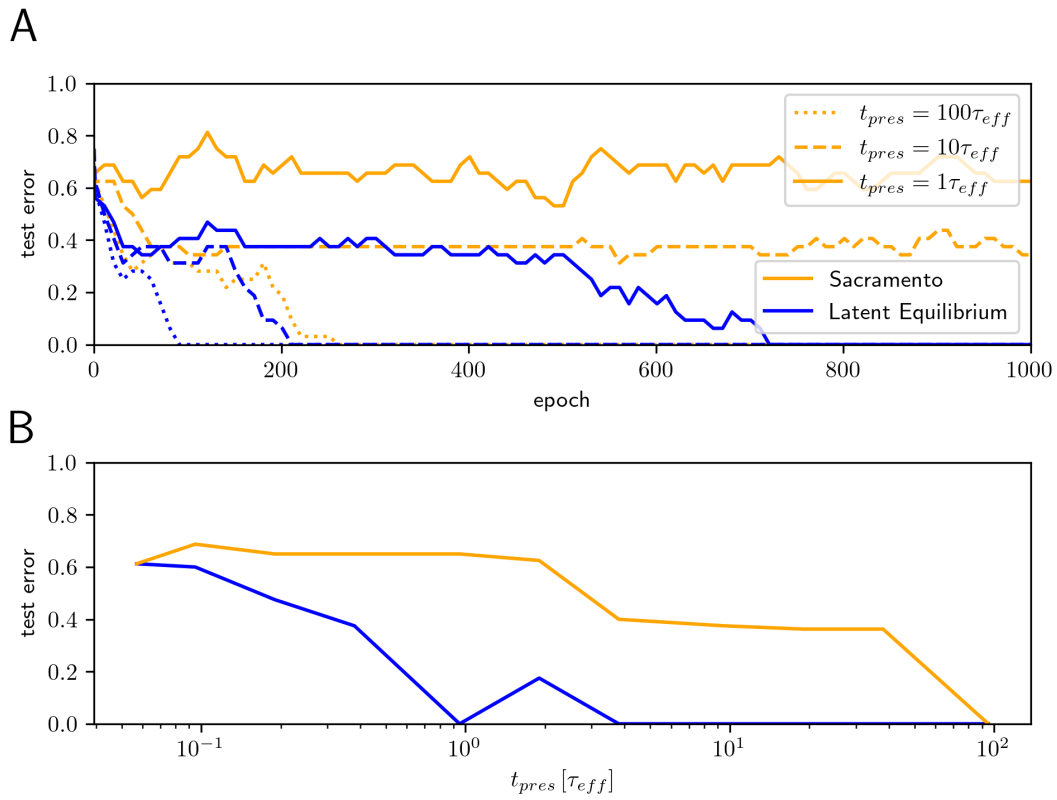

Figure 3.1: Replication of Figure 3 from Haider et al. (2021) using networks of spiking neurons in the NEST simulator. textbfA: Comparison between Original pyramidal microcircuit network by Sacramento et al. (2018) and updated variant from Haider et al. (2021). Shown is the training of a network with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

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

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 successfull 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**

15,006 458

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

6.2 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

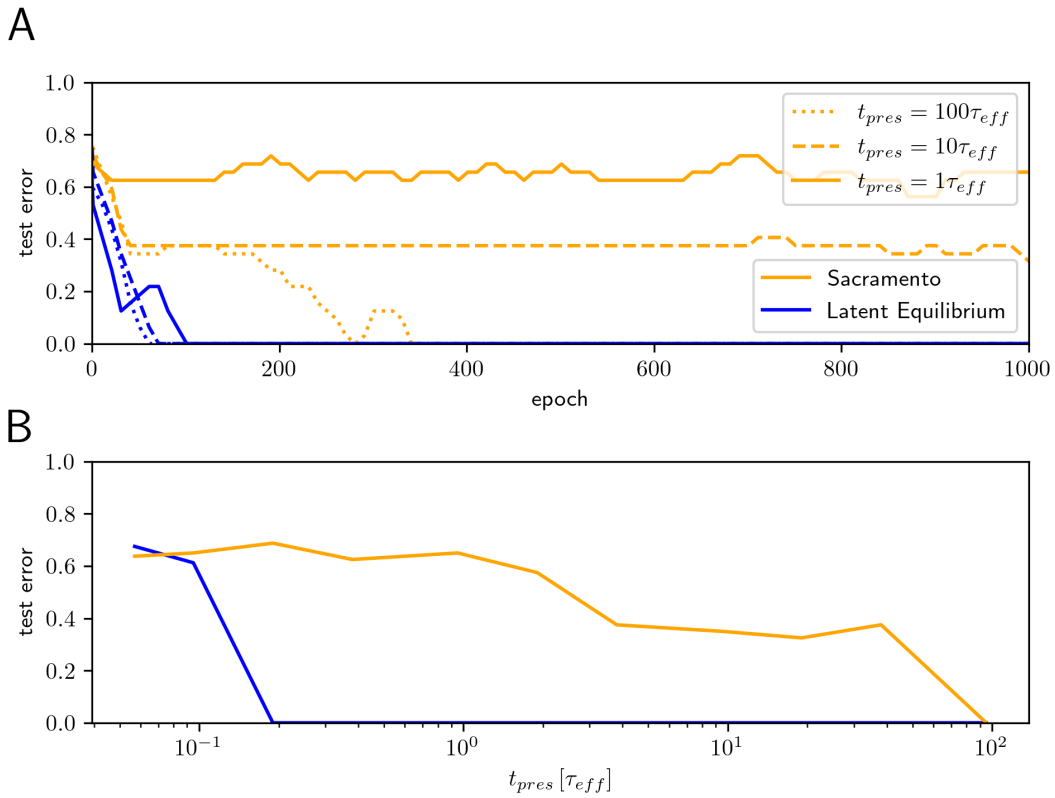

Figure 6.1: Replication of Figure 3.1 using a re-implementation of the Haider et al. (2021) code in python. Resulting performance matches the original results exactly, proving that my replication can serve as a baseline for comparing the NEST implementation

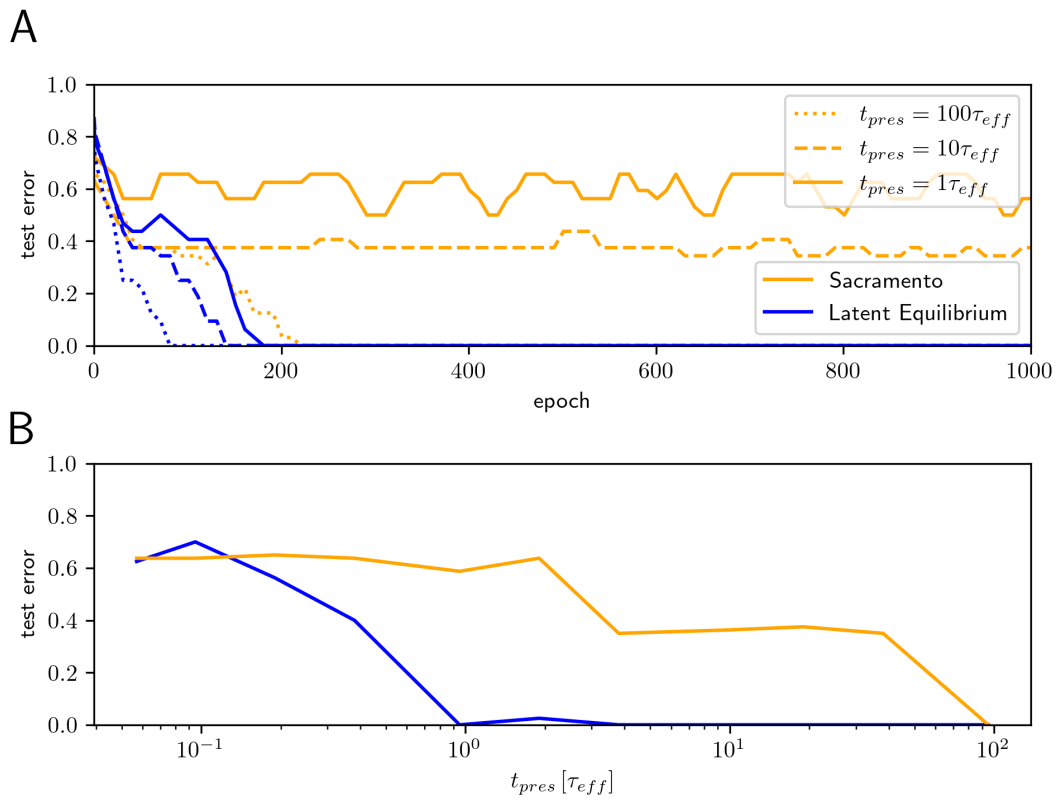

Figure 6.2: Replication of Figure 3.1 using networks of rate neurons in the NEST simulator.

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?

6.3 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

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.

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.

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.