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

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):

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.**

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** .

Neuron models

While the fundamental computational unit of ANNs is called a neuron, it 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. In the case of backprop, it is even unclear how a derivative of activity can be computed from a spiketrain.

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.

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"*.

TODO: discuss supervisor issue?

1.3 Approximating Backprop

Yet, despite these mechanisms being at odds with biology, 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).

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).

2.2 Pyramidal and interneurons

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 compartment voltages, which leak into the soma with specific conductances. The activation r_l^P of a pyramidal neuron at layer l is given by applying the synaptic transfer function ϕ to to its somatic potential 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 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 potential of a pyramidal neuron in layer l 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. An interneuron integrates synaptic information by the same principle, but instead of feedback information arriving at the apical compartment, it is integrated directly into the soma[1]:

$$C_m \dot{u}_l^I = -g_l u_l^I + g^{bas} v_l^{bas} + i^{nudge,I} \tag{2.4}$$
$$i^{nudge,I} = g^{nudge,I} u_{l+1}^P \tag{2.5}$$

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. **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:

$$C_m \dot{u}_N^P = -g_l u_N^P + g^{bas} v_N^{bas} + i^{nudge,tgt} \tag{2.6}$$
$$i^{nudge,tgt} = g^{nudge,tgt} u^{tgt} \tag{2.7}$$

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?**

2.3 Urbanczik-Senn Plasticity

The synapses in the network are all modulated through variations of the "Urbanczik-Senn" plasticity rule (Urbanczik and Senn, 2014), which will be discussed in this section.

The plasticity rule requires the postsynaptic neuron to have one somatic and at least one dendritic compartment, to the latter of which the synapse connects. Functionally, the plasticity rule seeks to minimize discrepancies between somatic and dendritic potential. If a current is injected into the soma, it creates a difference between somatic firing rate and dendritic prediction (i.e. a prediction error). The change in weight for a synapse fron neuron j to neuron i is thus given by:

[1] Note, that Haider et al. (2021) differentiates between pyramidal neuron conductances g^{bas} and interneuron dendritic conductances g^{dend}. Since they share the same value in all relevant simulations at hand, this separation has been omitted for the sake of brevity

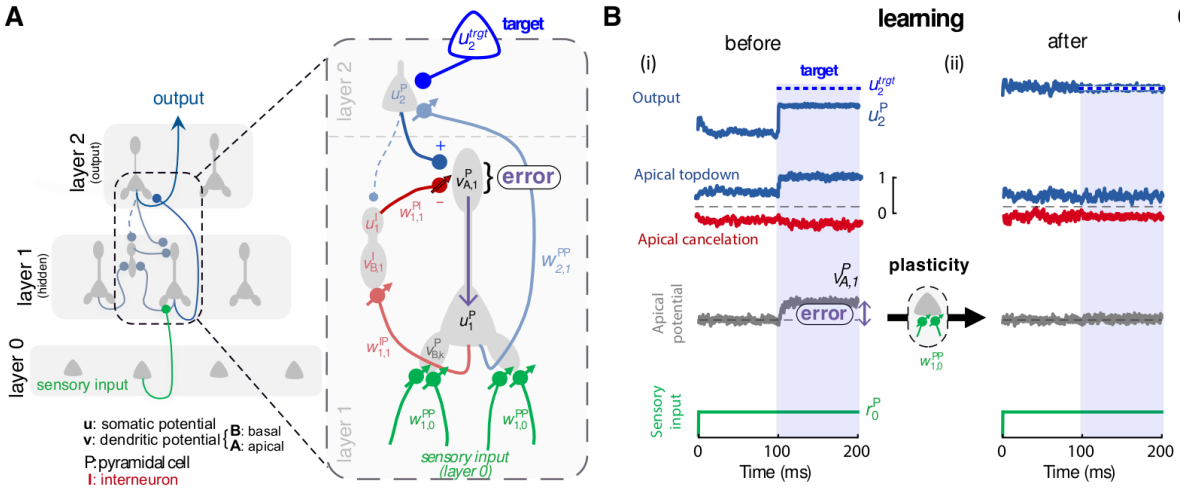

Figure 2.1: Self-predicting initialization without plasticity

$$\dot{w}_{ij} = \eta \ \phi(u_i^{som}) - \phi(\hat{v}_i^{bas}) \ \phi(u_j^{som}) \tag{2.8}$$

$$\hat{v}_i^{bas} = \alpha \ v_i^{bas} \tag{2.9}$$

with learning rate η and prediction of somatic voltage at the basal dendrite $\hat{v}_i^{bas}$. 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 $\alpha = \frac{g^{bas}}{g_l + g^{bas} + g^{api}}$.

from Haider et al. (2021):

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.

One-sided exponential decay kernel

$$\kappa(t) = H(t)e^{-t/\tau_\kappa} \tag{2.10}$$

$$H(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t \leq 0 \end{cases} \tag{2.11}$$

Antiderivatives:

$$\int_{-\infty}^{x} H(t)dt = tH(t) = max(0,t) \tag{2.12}$$

Convolution:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.13}$$

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

$$(f * g)(t) = \int_{0}^{t} f(\tau)g(t-\tau)d\tau \tag{2.14}$$

$$\tag{2.15}$$

Plasticity:

$$\frac{dW_{ij}}{dt}(t) = F(W_{ij}(t), s_i^*(t), s_j^*(t), V_i^*(t)) \tag{2.16}$$

$$F[s_j^*, V_i^*] = \eta\,\kappa * (V_i^* s_j^*) \tag{2.17}$$

$$\text{with } V_i^* = (s_i - \phi(V_i))h(V_i), \tag{2.18}$$

$$s_j^* = \kappa_s * s_j. \tag{2.19}$$

For an event-based plasticity we need:

$$\Delta W_{ij}(t,T) = \int_t^T dt' F[s_j^*, V_i^*](t') \tag{2.20}$$
$$= \int_t^T dt' \eta \kappa * (V_i^* s_j^*) \tag{2.21}$$
$$= \eta \int_t^T dt' \int_0^{t'} dt'' \ \kappa(t'-t'') V_i^*(t'') s_j^*(t'') \tag{2.22}$$
$$= \eta \int_0^t dt' \int_{t''}^{t'} dt'' \ \kappa(t'-t'') V_i^*(t'') s_j^*(t'') \tag{2.23}$$
$$\tag{2.24}$$

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

$$\Delta W_{ij}(0,t) = \eta \int_0^t dt' \int_0^{t'} dt'' \ \kappa(t'-t'') V_i^*(t'') s_j^*(t'') \tag{2.25}$$
$$= \eta \int_0^t dt'' \int_{t''}^{t} dt' \ \kappa(t'-t'') V_i^*(t'') s_j^*(t'') \tag{2.26}$$
$$= \eta \int_0^t dt'' \ \left[\tilde{\kappa}(t-t'') - \tilde{\kappa}(0)\right] V_i^*(t'') s_j^*(t'') \tag{2.27}$$
$$\tag{2.28}$$

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

$$\kappa(t) = \frac{\delta}{\delta t} \tilde{\kappa}(t) \tag{2.29}$$
$$\tilde{\kappa}(t) = -e^{-\frac{t}{t_\kappa}} \tag{2.30}$$
$$\tag{2.31}$$

The above can be split up into two separate integrals:

$$\Delta W_{ij}(0,t) = \eta \left[-I_2(0,t) + I_1(0,t)\right] \tag{2.32}$$
$$I_1(t_1,t_2) = -\int_{t_1}^{t_2} dt' \ \tilde{\kappa}(0) V_i^*(t') s_j^*(t') \tag{2.33}$$
$$I_2(t_1,t_2) = -\int_{t_1}^{t_2} dt' \ \tilde{\kappa}(t_2-t') V_i^*(t') s_j^*(t') \tag{2.34}$$
$$\tag{2.35}$$

Which implies the identities

$$I_1(t_1, t_2 + \Delta t) = I_1(t_1, t_2) + I_1(t_2, t_2 + \Delta t) \tag{2.36}$$

$$I_2(t_1, t_2 + \Delta t) = e^{-\frac{t_2 - t_1}{\tau_\kappa}} I_2(t_1, t_2) + I_2(t_2, t_2 + \Delta t) \tag{2.37}$$

$$I_2(t_1, t_2 + \Delta t) = -\int_{t_1}^{t_2 + \Delta t} dt' \ \tilde{\kappa}(t_2 + \Delta t - t') V_i^*(t') s_j^*(t') \tag{2.38}$$

$$= -\int_{t_1}^{t_2} dt' \ \left[-e^{-\frac{t_2 + \Delta t - t'}{\tau_\kappa}} \right] V_i^*(t') s_j^*(t') - \int_{t_2}^{t_2 + \Delta t} dt' \ \left[-e^{-\frac{t_2 + \Delta t - t'}{\tau_\kappa}} \right] V_i^*(t') s_j^*(t') \tag{2.39}$$

$$= -e^{-\frac{\Delta t}{\tau_\kappa}} \int_{t_1}^{t_2} dt' \ \left[-e^{-\frac{t_2 - t'}{\tau_\kappa}} \right] V_i^*(t') s_j^*(t') - \int_{t_2}^{t_2 + \Delta t} dt' \ \left[-e^{-\frac{t_2 + \Delta t - t'}{\tau_\kappa}} \right] V_i^*(t') s_j^*(t') \tag{2.40}$$

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

$$\Delta W_{ij}(t, T) = \Delta W_{ij}(0, T) - \Delta W_{ij}(0, t) \tag{2.41}$$

$$= \eta [-I_2(0, T) + I_1(0, T) + I_2(0, t) - I_1(0, t)] \tag{2.42}$$

$$= \eta [I_1(t, T) - I_2(t, T) + I_2(0, t) \left(1 - e^{-\frac{T - t}{\tau_\kappa}} \right)] \tag{2.43}$$

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

$$\frac{dW_{ij}}{dt} = \eta (\phi(u_i) - \phi(\hat{v}_i)) \phi(u_j) \tag{2.44}$$

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

$$\Delta W_{ij}(t, T) = \eta \int_t^T dt' \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{2.46}$$

$$V_i^* = \phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}}) \tag{2.47}$$

$$s_j^* = \kappa_s * s_j \tag{2.48}$$

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?** .

$$\tau_l = \frac{C_m}{g_L} = 10 \tag{2.49}$$

$$\tau_s = 3 \tag{2.50}$$

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:

$$\int_{t_{ls}}^{T} dt'\ V_i^* = \int_{t_{ls}}^{T} dt'\ (s_i - \phi(V_i))h(V_i), \tag{2.51}$$

$$\int_{t_{ls}}^{T} dt'\ V_i^* = \sum_{t=t_{ls}}^{T} (s_i(t) - \phi(V_i^t)\Delta t)h(V_i^t) \tag{2.52}$$

$$\tag{2.53}$$

```
1 for (t = t_ls; t< T; t = t + delta_t)
2 {
3     minus_delta_t = t_ls - t;
4     minus_t_down = t - T;
5     PI = ( kappa_l * exp( minus_delta_t / tau_L ) - kappa_s * exp(
    minus_delta_t / tau_s ) ) * V_star(t);
6     PI_integral_ += PI;
7     dPI_exp_integral += exp( minus_t_down / tau_Delta_ ) * PI;
8 }
9 // I_2 (t,T) = I_2(0,t) * exp(-(T-t)/tau) + I_2(t,T)
10 PI_exp_integral_ = (exp((t_ls-T)/tau_Delta_) * PI_exp_integral_ +
    dPI_exp_integral);
11 W_ji = PI_integral_ - PI_exp_integral_;
12 W_ji = init_weight_ + W_ji * 15.0 * C_m * tau_s * eta_ / ( g_L * ( tau_L -
    tau_s ) );
13
14 kappa_l = kappa_l * exp((t_ls - T)/tau_L) + 1.0;
15 kappa_s = kappa_s * exp((t_ls - T)/tau_s) + 1.0;
16
```

$$\int_{t_{ls}}^{T} dt' s_j^* = \tilde{\kappa_L}(t') * s_j - \tilde{\kappa_s}(t') * s_j \tag{2.54}$$

I_1 in the code is computed as a sum:

$$I_1(t,T) = \sum_{t'=t}^{T} (s_L^*(t') - s_s^*(t')) * V^*(t') \tag{2.55}$$

2.4 steady-state potentials in Sacramento (2018)

$$u_k^p = \frac{g_B}{g_{lk} + g_B + g_A} v_{B,k}^P + \frac{g_A}{g_{lk} + g_B + g_A} v_{A,k}^P \tag{2.56}$$

$$\hat{v}_{B,k}^P = \frac{g_B}{g_{lk} + g_B + g_A} v_{B,k}^P \tag{2.57}$$

$$\hat{v}_k^I = \frac{g_B}{g_{lk} + g_B} v_k^I \tag{2.58}$$

$$\lambda = \frac{g_{som}}{g_{lk} + g_B + g_{som}} \tag{2.59}$$

2.5 Multi-compartment neuron models

2.6 Cortical microcircuits

2.7 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:

$$\dot{W}_{ij}(t) = \eta(\phi(u_i) - \phi(\alpha v_i^{basal}(t)))\phi(u_j) \tag{2.60}$$

$$\Delta W_{ij}(t,T) = \int_t^T dt' \ \eta \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{2.61}$$

$$\Delta W_{ij}(t,T) = \eta \int_t^T dt' \ (\phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}})) \ \phi(u_j^{t'}) \tag{2.62}$$

$$V_i^* = \phi(u_i^{t'}) - \phi(\widehat{v_i^{t'}}) \tag{2.63}$$

$$s_j^* = \kappa_s * s_j \tag{2.64}$$

2.8 Latent equilibrium

$$\phi(V^{som}) \to \phi(\breve{V}^{som}) \tag{2.65}$$

$$\breve{V} := V + \tau^m \dot{V} \tag{2.66}$$

$$\tag{2.67}$$

$$\frac{d}{dt} W_{ba} = \eta(\phi(V_b^{som}) - \phi(\alpha V_b^{dend}))\phi(V_a^{som}) \tag{2.68}$$

$$\frac{d}{dt} W_{ba} = \eta(\phi(\breve{V}_b^{som}) - \phi(\alpha \breve{V}_b^{dend}))\phi(\breve{V}_a^{som}) \tag{2.69}$$

2.9 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?

Chapter 4

Discussion

4.1 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

4.2 Outlook

reward-modulated urbanczik-senn plasticity?

Chapter 5

Appendix

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?

5.1 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 6

Preliminary structural components

6.1 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.

6.2 Literature review - Backpropagation in SNN

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

6.3 NEST Urbanczik-senn implementation

6.4 My neuron model

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

6.4.1 NEST rate neuron shenanigans

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

6.5 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.

6.6 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.

6.6.1 Interneurons and their jobs

6.7 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.

6.8 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.

6.9 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 7

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

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.

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.

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

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.