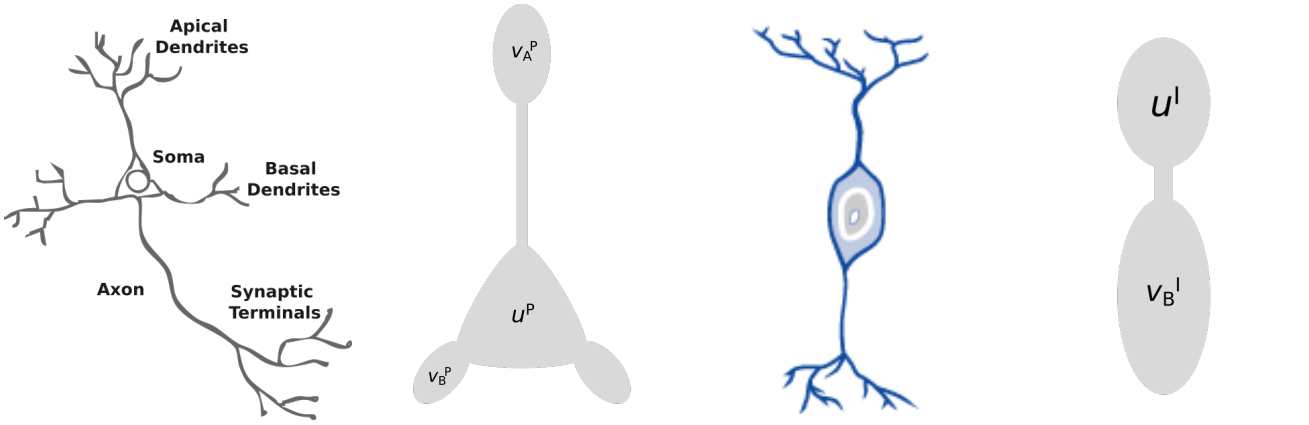
Neuro-Backprop

Paul Hill

August 2020

Abstract

In the following I report my implementation of "Dendritic error backpropagation in deep cortical microcircuits" [2]. The code can be found online [1].



(a) Draft of a biological pyramidal neuron.

(b) Abstracted and simplified computational model of a pyramidal neuron.

(c) Draft of a biological interneuron.

(d) Abstracted and simplified computational model of an interneuron.

Figure 1: Different neuron types.

1 Introduction

1.1 Multicompartment-Neurons

Real nerve-cells usually have a spatial extension much larger than their actual cell-bodies (somas). Since sources and drains of current are distributed all over the cell and electric signals propagate with finite speed, a spatial variation of the membrane potentials (specifically over the cell's dendrites) builds up. To model this effect one introduces several distinct compartments that are coupled to each other. The potential of the soma-compartment is then given by

$$\dot{u} = -g_l u + \sum_x g_x(v_x - u) + I_{syn}, \tag{1}$$

where g_l is the leakage conductance of the soma, x runs over all connected compartments of the cell and g_x is the respective conductance coupling x to the soma. I_{syn} accounts for an additional

current input coupled directly into the soma. In our case we will always have $I_{syn} = g_{som}(u_{ex} - u)$, i.e. COBA synaptic inputs are assumed. Here u_{ex} is a so far arbitrary external potential.
Note that (1) behaves as a low-pass filter for the potentials v_x and u_{ex} with time constant $\tau = \frac{1}{g_{tot}} = \frac{1}{g_l + \sum_x g_x + g_{som}}$. For imprinted signals with time scale $T \gg \tau$ the soma potentials takes on the form (steady-state solution)

$$\tilde{u} = \frac{1}{g_{tot}} \left(\sum_x g_x u_x + g_{som} u_{ex} \right). \tag{2}$$

Furthermore we restrict our model to the rate-based case in which synaptic inputs are given in terms of the firing rates of their presynaptic partners r_y

$$v_x = \sum_y W_{xy} r_y = \sum_y W_{xy} \phi(u_y). \tag{3}$$

Here the vector $\boldsymbol{W}_x$ denotes the coupling between the the presynaptic partners and the x-compartment and firing rates are given in terms of the corresponding soma potentials u_y via the activation function $\phi(u_y)$. Since the activation function essentially gives the rate at which a neuron with average soma potential u spikes, it is required to be monotonic and positive. We will work with either a soft-ReLU ($\phi(u) = \ln(1 + \exp(u))$) or a sigmoid activation function ($\phi(u) = 1/(1 + \exp(-u))$).

1.2 PyraL-Network (PyraLNet)

The network approximating the backpropagation-algorithm is built upon two kind of neurons, pyrimidial- and lateral inter-neurons. The first one consists of two compartments beside the soma, the basal and apical dendrites. (1) takes on the form

$$\dot{u}^P = -g_l u^P + g_B(v_B^P - u^P) + g_A(v_A^P - u^P), \tag{4}$$

where B denotes the basal and A the apical compartment. The inter-neurons on the other hand are governed by

$$\dot{u}^I = -g_l u^I + g_D(v_D^I - u^I) + g_{som}(u_{ex} - u^I), \tag{5}$$

where D denotes the dendritic compartment and we have an external potential u_{ex} directly coupled into the soma. Fig. 1 depicts both neuron types in their biological manifestation and abstract description, respectively.
With these two types of neurons one is now able to go on and build larger networks as the one depicted in Fig. 2.
The network dynamics for N layers ($N - 2$ hidden layers) are governed by the following set of equations (for $0 < k < N$)

$$\dot{\boldsymbol{u}}_k^P = -g_l \boldsymbol{u}_k^P + g_B(\boldsymbol{v}_{B,k}^P - \boldsymbol{u}_k^P) + g_A(\boldsymbol{v}_{A,k}^P - \boldsymbol{u}_k^P) \tag{6}$$

$$\dot{\boldsymbol{u}}_k^I = -g_l \boldsymbol{u}_k^I + g_D(\boldsymbol{v}_{D,k}^I - \boldsymbol{u}_k^I) + g_{som}(\boldsymbol{u}_{k+1}^P - \boldsymbol{u}_k^I) \tag{7}$$

$$\boldsymbol{v}_{B,k}^P = \boldsymbol{W}_k^{up} \phi(\boldsymbol{u}_{k-1}^P) \tag{8}$$

$$\boldsymbol{v}_{A,k}^P = \boldsymbol{W}_k^{pi} \phi(\boldsymbol{u}_k^I) + \boldsymbol{W}_k^{down} \phi(\boldsymbol{u}_{k+1}^P) \tag{9}$$

$$\boldsymbol{v}_{D,k}^I = \boldsymbol{W}_k^{ip} \phi(\boldsymbol{u}_k^P), \tag{10}$$

where $\boldsymbol{W}$ are now matrices, $\boldsymbol{v}, \boldsymbol{u}$ are vectors and k is the layer index ($k = 0$ is the input layer).
The output neurons look slightly different,

$$\dot{\boldsymbol{u}}_N^P = -g_l \boldsymbol{u}_N^P + g_B(\boldsymbol{v}_{B,N}^P - \boldsymbol{u}_N^P) + g_{som}(\boldsymbol{u}^{target} - \boldsymbol{u}_N^P) \tag{11}$$

$$\boldsymbol{v}_{B,N}^P = \boldsymbol{W}_N^{up} \phi(\boldsymbol{u}_{N-1}^P). \tag{12}$$

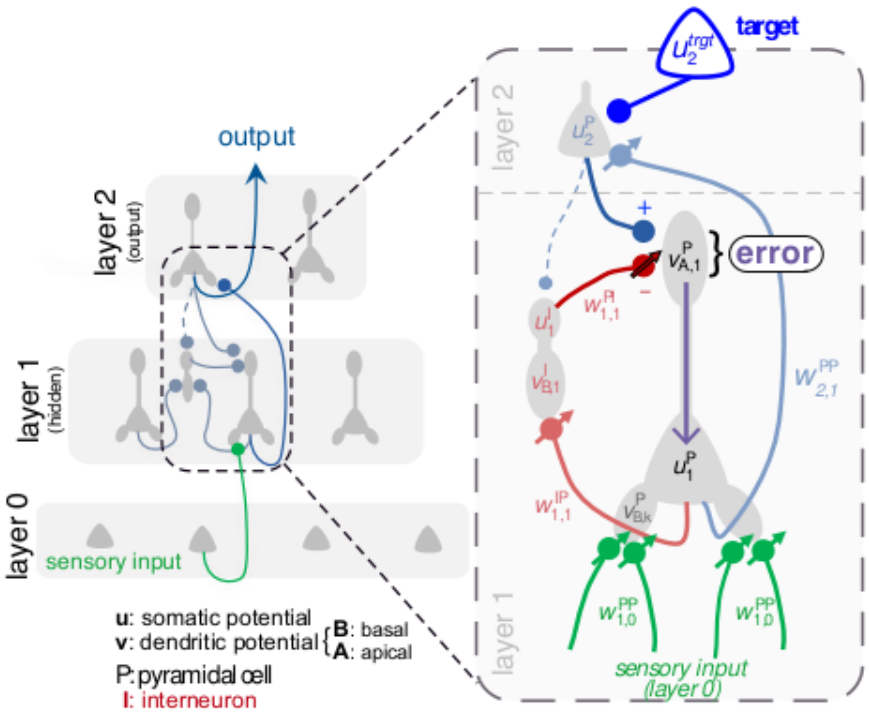


Figure 2: An abstracted cortical microcircuit capable of approximating the backpropagation algorithm. Note that the notation differs from the one used in the main text, namely $W^{PP}_{k,k-1} \equiv W^{up}_k$ and $W^{PP}_{k+1,k} \equiv W^{down}_k$.

Note that, provided the network reaches a (unique) fix-point (also called steady-state), i.e. a configuration in which all derivatives of the soma potentials vanish simultaneously, the network essentially maps the input rates (or input potentials) of the 0-th layer to the output rates obtained from the N-th layer. It is this non-linear mapping between input and output rates that eventually enables the network to solve complex classification and regression tasks by adjusting the strength of its synaptic connections $\boldsymbol{W}$. However, the existence (and uniqueness) of a fix-point of the underlaying network dynamics depends non-trivially on the weights $\boldsymbol{W}$ and conductances g.

1.3 Learning-Rules

Assuming the network indeed converges to a stable fix-point for the given input rates, there is yet no mechanism that makes the output potentials reproduce a given target signal $\boldsymbol{u}^{target}$. To solve this, the concept of dendritic prediction of somatic spiking rates is introduced. Consider therefore a pyramidal neuron in the output layer when teaching is disabled ($g_{som} = 0$). The soma potential is then, in steady-state, given by

$$\hat{\boldsymbol{v}}^P_{B,N} \equiv \tilde{\boldsymbol{u}}^P = \frac{g_B}{g_l + g_B} \boldsymbol{W} r^P_{N-1}. \tag{13}$$

When teaching is enabled, $\tilde{u}^P$ is nudged away from the dendritic prediction $\hat{v}^P$ towards

$$\tilde{\boldsymbol{u}}^P_N = (1 - \lambda_{out}) \hat{\boldsymbol{v}}^P_{B,N} + \lambda_{out} \boldsymbol{u}^{target}, \tag{14}$$

with $\lambda_{out} = \frac{g_{som}}{g_l + g_B + g_{som}}$. The difference in predicted and actual firing rate $\phi(u^P) - \phi(\hat{v}^P)$ can now be used to update the synaptic strengths such that $\hat{v}^P$ is itself nudged towards u^{target} and eventually

$\tilde{u}^P = \hat{v}^p$ holds again.
Similar equations hold for hidden pyramidal neurons and inter-neurons as well

$$\tilde{u}_k^P = \hat{\boldsymbol{v}}_{B,k}^P + \lambda_{hid} \boldsymbol{v}_{A,k}^P \tag{15}$$
$$\lambda_{hid} = \frac{g_A}{g_l + g_B + g_A} \tag{16}$$
$$\hat{\boldsymbol{v}}_{B,k}^P = \frac{g_B}{g_l + g_B + g_A} \boldsymbol{W}_k^{up} r_{k-1}^P \tag{17}$$
$$\tilde{u}_k^I = (1 - \lambda_I)\hat{v}_{D,k}^I + \lambda_I u_{k+1}^P \tag{18}$$
$$\lambda_I = \frac{g_{som}}{g_l + g_D + g_{som}} \tag{19}$$
$$\hat{\boldsymbol{v}}_{D,k}^I = \frac{g_D}{g_l + g_D} \boldsymbol{W}_k^{ip} r_k^P. \tag{20}$$

Finally, updating of the weights is realized by first low-pass filtering the dendritic prediction error with time constant τ_w [1]

$$\tau_w \dot{\boldsymbol{\Delta}}_k^{up} = \left(\phi(\boldsymbol{u}_k^P - \phi(\hat{\boldsymbol{v}}_{B,k}^P)\right) (r_{k-1}^P)^T \tag{21}$$
$$\tau_w \dot{\boldsymbol{\Delta}}_k^{ip} = \left(\phi(\boldsymbol{u}_k^I - \phi(\hat{\boldsymbol{v}}_{D,k}^I)\right) (r_k^P)^T \tag{22}$$
$$\tau_w \dot{\boldsymbol{\Delta}}_k^{pi} = -\boldsymbol{v}_A^P (r_k^I)^T \tag{23}$$

and then integrating

$$\frac{d\boldsymbol{W}_k^{up}}{dt} = \eta_k^{up} \boldsymbol{\Delta}_k^{up} \tag{24}$$
$$\frac{d\boldsymbol{W}_k^{ip}}{dt} = \eta_k^{ip} \boldsymbol{\Delta}_k^{ip} \tag{25}$$
$$\frac{d\boldsymbol{W}_k^{pi}}{dt} = \eta_k^{pi} \boldsymbol{\Delta}_k^{pi}. \tag{26}$$

Note that, based on eq. (21) - (23), output neurons change to predict the target signal, inter-neurons change to predict the pyramidal neurons of the next layer, hidden pyramidal neurons are driven by their apical compartment and finally the apical compartment adjusts the lateral $\boldsymbol{W}^{pi}$ weights such that the top-down (see eq. (9)) is canceled and the apical potential becomes zero.

1.3.1 Self-predicting state

Especially when no target signal is applied this mechanism drives the network in what is called the self-predicting state (strictly speaking it is necessary that the initial (random) state of the network allows for sufficiently well-behaved, i.e. stable, dynamics. Usually one adjusts the random distributions, the initial weights are drawn from, too produce sufficiently small weights such that the network is well-behaved). It is characterized by the cancellation of all apical potentials and when inter-neuron soma potentials match the pyramidal soma potentials of the next layer, i.e.

$$\boldsymbol{v}_A^P = 0 \tag{27}$$
$$\hat{\boldsymbol{v}}_{D,k}^I = \hat{\boldsymbol{v}}_{B,k+1}^P, \tag{28}$$

[1] While this procedure is not directly documented in the original paper it is suggested in the supplementary material. The implementation of the low-pass filtering is motivated by [3].

which is equivalently expressed as

$$\boldsymbol{W}_k^{pi} = -\boldsymbol{W}_k^{down} \tag{29}$$

$$\boldsymbol{W}_k^{ip} = \frac{g_l + g_D}{g_l + g_B + g_A}\frac{g_B}{g_D}\boldsymbol{W}_{k+1}^{up}, \tag{30}$$

with $g_A = 0$ for the output layer. Since in the self-predicting state all apical compartments vanish the network is guaranteed to have a fix-point and behaves (after settling) like a artificial neural network with same forward weights ($\sim \boldsymbol{W}^{up}$) and activation function. Note that the self-predicting state usually also emerges during learning of a task, i.e. when a target signal is applied right from the beginning.

1.4 Approximation of the Error-Backpropagation Algorithm [2]

Starting from the the self-predicting state we now switch on the teaching signal $\boldsymbol{u}^{target}$, where throughout small couplings $\lambda = \lambda_{out} = \lambda_I = \lambda_{hid} \ll 1$ are assumed.
Doing so nudges the output potential away from $\hat{\boldsymbol{v}}_{B,N}^P = \tilde{\boldsymbol{v}}_{D,N-1}^I$, resulting in a change of the apical potential in the previous layer as

$$\begin{aligned}\boldsymbol{v}_{A,N-1}^P &= \boldsymbol{W}_{N-1}^{down}\underbrace{\left(\phi(\tilde{\boldsymbol{u}}_N^P) - \phi(\hat{\boldsymbol{v}}_{B,N}^P + \mathcal{O}(\lambda))\right)}_{\boldsymbol{e}_N} \approx \lambda \boldsymbol{W}_{N-1}^{down}\phi'(\hat{\boldsymbol{v}}_{B,N}^P)\left(\boldsymbol{u}^{target} - \hat{\boldsymbol{v}}_N^P\right) \\ &\equiv \lambda \boldsymbol{W}_{N-1}^{down}\boldsymbol{D}_N\left(\boldsymbol{u}^{target} - \hat{\boldsymbol{v}}_N^P\right),\end{aligned} \tag{31}$$

where the prediction error $\boldsymbol{e}_N \equiv \phi(\hat{\boldsymbol{v}}_{B,N}^P)$ was introduced. The $\mathcal{O}(\lambda)$ term appears since also the interneurons are nudged away from their dendritic prediction $\hat{\boldsymbol{v}}_{D,N-1}^I = \tilde{\boldsymbol{v}}_{B,N}^P$ (including again $\mathcal{O}(\lambda)$ corrections from the pyramidal neurons due to apical nudging). This error propagates further down the network to the interneurons and pyramidals of the next layer. For the pyramidals one finds

$$\begin{aligned}\boldsymbol{e}_{N-1} &= \phi(\tilde{\boldsymbol{u}}_{N-1}^P) - \phi(\hat{\boldsymbol{v}}_{B,N-1}^P) \approx \phi'(\hat{\boldsymbol{v}}_{B,N-1}^P)\lambda \boldsymbol{W}_{N-1}^{down}\boldsymbol{e}_N \\ &= \lambda^2 \boldsymbol{D}_{N-1}\boldsymbol{W}_{N-1}^{down}\boldsymbol{D}_N\left(\boldsymbol{u}^{target} - \hat{\boldsymbol{v}}_N^P\right).\end{aligned} \tag{32}$$

By means of this mechanism the pyramidals of the hidden layer can support learning in the output layer by trying to predict away the local error $\boldsymbol{e}_{N-1}$.

1.5 Steady-State-Approximation of the Network Dynamics (SteadNet)

In a typical application input rates and corresponding target signals are presented to the network for a fixed duration $t_{pattern}$, i.e. every $t_{pattern}$ a new input-output pair ($\boldsymbol{r}_{in}$, $\boldsymbol{u}^{target}$) is presented. Since typically $t_{pattern} \gg \tau$, where τ is the total neuron time constant, numerical simulation of the network is quite time intensive. To speed up the learning process a few approximations can be made. Since eventually one is interested in the fix-point the network will settle in once a new pattern is applied, the idea is to replace the exact settling dynamics by an approximate algorithm. Therefore the input signal is propagated upwards by means of the steady-state solution eq. (17) and (20) (pyramidal before inter-neurons) up to the output layer where finally the basal prediction is mixed with the target signal according to eq. (14). Then the output potentials are propagated backwards by means of the full steady-state solution eq. (2) (interneurons before pyramidals). This

[2] A complete derivation can be found in the original paper.

up- and downwards passing can be repeated several times to increase convergence.
Updating of the weights matrices is done separately by solving eq. (21) - (26) numerically via the explicit Euler method with much larger timestep dT than the usual simulation time step dt (up to $dT = t_{pattern}$). After each timestep the potentials are updated by applying the above procedure.

2 Experimental Results

2.1 Basics

2.1.1 Stability and Fix-Points

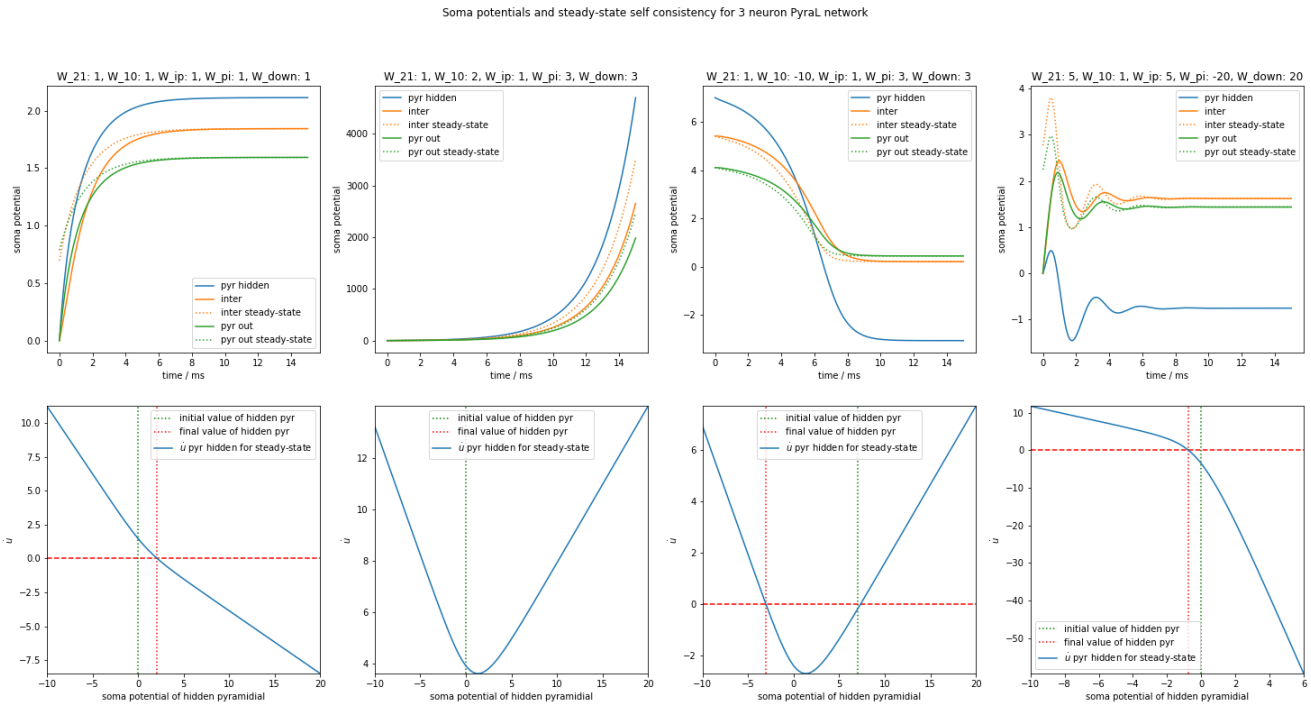


Figure 3: **Upper row**: Evolution of soma potentials of a simple 3-layer network with three neurons in total. Columns represent different network configurations and initial conditions and weights are kept fixed during the simulation. **Lower row**: Derivative of the hidden pyramidal soma potential u^P_{hid} fully expressed through u^P_{hid} via the steady-state solution of the other soma potential (see main text). **Details:** $g_A = g_{som} = 0.8$, $g_D = g_B = 1$, $g_l = 0.1$, $r_{in} = u^{target} = 1$, $dt = 0.001$

Let us first investigate the network dynamics eq. (6) - (12), i.e. with fixed weights, regarding stability. To do so we set up a very simple toy model with one hidden and output layer, each consisting of one pyramidal and one interneuron, one pyramidal output-neuron, respectively. A soft ReLU activation function ($\phi(u) = \ln(1 + \exp(u))$) is used since it is more prone to instabilities due to its unbound nature. Fig 3 shows the behavior of the network for four different configurations of the network, where synaptic strengths and initial conditions change. The input rate r_{in} and the teaching potential u_{target} are equal for all cases. In the upper row the soma potentials of the three neurons are plotted, as well as the steady state solutions eq. (2) for the interneuron and the output-neuron given the current potential of the hidden pyramidal neuron. In order for the system to posses a fix-point the left hand side of eq. (6) - (12) has to vanish simultaneously, i.e. at the

fix-point

$$u^P_{out} = \frac{1}{g^{out}_{tot}} \left(g_B W_{21} \phi(u^P_{hid}) + g_{som} u^{target} \right) \tag{33}$$

$$u^I_{hid} = \frac{1}{g^I_{tot}} \left(g_D W_{ip} \phi(u^P_{hid}) + g_{som} u^P_{out} \right) \tag{34}$$

$$\dot{u}^P_{hid} = -g^P_{tot} u^P_{hid} + g_B W_{10} r_{in} + g_A \left(W_{pi} \phi(u^I_{hid}) + W_{down} \phi(u^P_{out}) \right) = 0 \tag{35}$$

holds. Plugging (33), (34) into (35) yields the self-consistency condition

$$\begin{aligned} 0 = \dot{u}^P_{hid} \equiv \dot{u}(u^P_{hid}) = \; & g^P_{tot} u^P_{hid} + g_B W_{10} r_{in} \\ & + g_A W_{pi} \phi \left[\frac{1}{g^I_{tot}} \left(g_D W_{ip} \phi(u^P_{hid}) + \frac{g_{som}}{g^{out}_{tot}} \left(g_B W_{21} \phi(u^P_{hid}) + g_{som} u^{target} \right) \right) \right] \\ & + g_A W_{down} \phi \left[\frac{1}{g^{out}_{tot}} \left(g_B W_{21} \phi(u^P_{hid}) + g_{som} u^{target} \right) \right] \end{aligned} \tag{36}$$

which is depicted in the lower row in Fig. 3. For the example configuration shown there, one finds a stable fix-point (first and last column), a stable and an unstable fix-point (third column) and no fix-point at all (second column). For the latter case the network behaves completely unstable and no mapping between in and output signals in the sense of the usual artificial neural networks exist. Also interesting is the last column in Fig. 3 where an oscillatory behavior can be observed.
In practice it is difficult to predict if a given set of weights will lead to unstable behavior and usually such instabilities can be cured by changing to smaller initial weight matrices or conductances. Nevertheless, one should be aware that the network can at some point become unstable and especially when unbounded activation functions are used, care has to be taken.

2.1.2 Basic Regression Task and Emergence of Self-Predicting-State

As a first test setup we consider a simple "mimic-task" for a small network. We therefore devise a teaching network with only forwards weights as

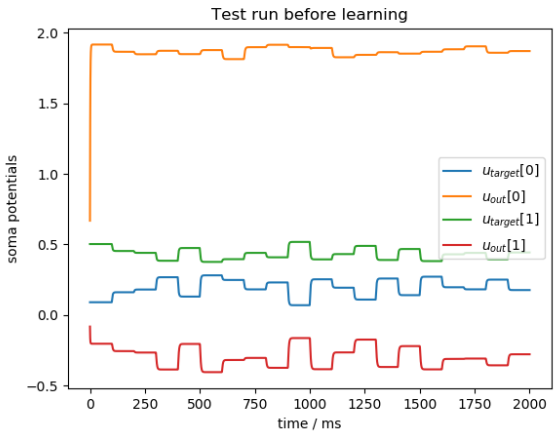
$$\boldsymbol{u}^{target} = \frac{g_B}{g_l + g_B} \boldsymbol{W}_{21} \phi \left(\frac{g_B}{g_l + g_B + g_A} \boldsymbol{W}_{10} \boldsymbol{r}_{in} \right), \tag{37}$$

where the gs are the conductances of the network we want to train, such that both networks are guaranteed to produce the same outputs once the self-predicting state has emerged and the forward weights of both networks equal (it's not clear if the opposite holds, i.e. if there exist different forward weights [3] that can exactly mimic the teaching network). The learning network is set up with the following parameters in the PyraLNet-code [1]:

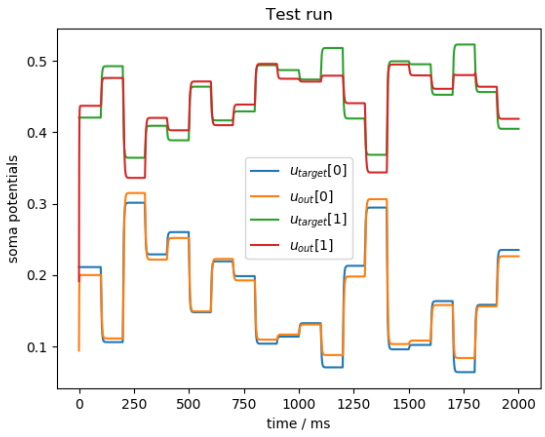
```
{"dims": [2, 3, 2], "dt": 0.1, "gl": 0.1, "gb": 1.0, "ga": 0.8, "gd": 1.0,
 "gsom": 0.8, "eta": {"up": [0.01, 0.005], "pi": [0.01, 0], "ip": [0.01, 0]},
 "bias": {"on": False, "val": 0.0},
 "init_weights": {"up": 1, "down": 1, "pi": 1, "ip": 1}, "tau_w": 30, "noise": 0,
 "t_pattern": 100, "out_lag": 3 * 10, "tau_0": 3, "learning_lag": 0,
 "reset_deltas": False}
```

In the following the parameters will be explained one after the other. First the network dimensions are specified as input layer, hidden layer, ..., output layer dimension. Then we have the simulation

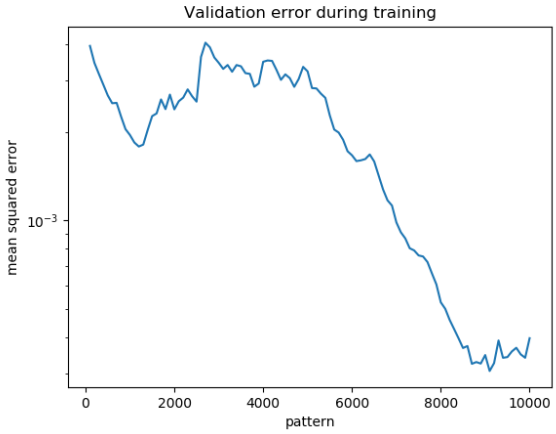
[3] "Different" is understood such that the two matrices don't arise from just permuting neurons in the same layer



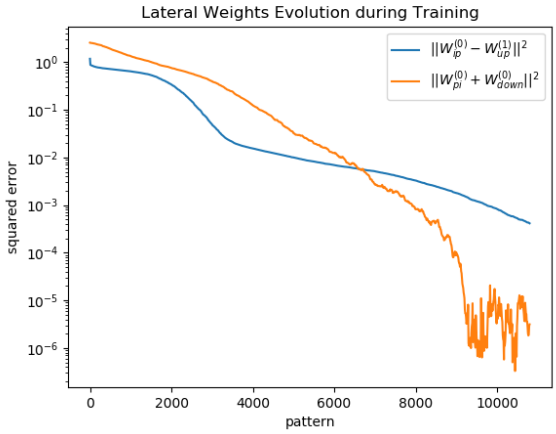
(a) Output layer soma potentials and target potentials before training.



(b) Output layer soma potentials and target potentials after training.



(c) MSE validation error during training.



(d) Emergence of self-predicting-state during training starting from randomly initialized connections.

Figure 4: The networks solves a basic regression task in which it has to mimic a randomly initialized (forward-)teaching network.

parameter *dt* used in the explicit Euler method for solving the network dynamics, followed by the conductances and the learning rates η (*eta*; order: first hidden layer to output layer). Next, one can specify whether a bias neuron, with fixed value *val*, for the input and hidden layers should be added.

When the network is initialized all weights are drawn from a uniform distribution $U(-a, a)$, where a can be specified per weight type in the next element. *tau_w* and *t_pattern* account for the plasticity time constant τ_w and the duration for which each input-output pair $(\boldsymbol{r}_{in}, \boldsymbol{u}^{target})$ is presented, respectively. Furthermore these input-output signals are transitioned smoothly across patterns by applying a low-pass filter with time constant τ_0 (*tau_0*). The noise parameter specifies the amount of Gaussian noise to add to the soma potential dynamics (this term is neglected in all of the above equations, see the original paper).

More interesting again are the remaining three parameters. *out_lag* specifies the amount of time to wait after each new input pattern, before the output potentials are averaged over the remaining pattern-time to give a single (time-independent) output vector associated to the presented input $\boldsymbol{r}_{in}$. Furthermore, it was observed, that the settling phase of the network contributes significantly

to the overall prediction error during one pattern (cf. Fig. 3 which shows a deviation of the actual soma potentials from the dendritic prediction during settling). This basically chokes learning of the problem beyond a certain threshold MSE of the potential deviations in the settled state (corresponding to how well the target signal was predicted by the network), when the settling prediction error becomes the dominant source of plasticity. In order to overcome this a so called learning-lag (*learning_lag*) is introduced, which specifies the time to wait after a new pattern before plasticity is enabled, by means of eq. (21) - (26) (these equations are not invoked during that time). Additionally one can choose to set all updates Δ to zero after each pattern by specifying *reset_deltas* accordingly.

Fig. 4 presents a network with the above parameters and a soft ReLU activation function that learned to mimic a teacher-network on a training set of 1000 points for 10 epochs. The upper row shows the output and target potentials before and after learning. The lower panel shows the mean squared error on validation samples (during which teaching is disabled) on the left. On the bottom right one can observe the emergence of the self-predicting state during training as defined by eq. (29) and (30), which leads to a drop in validation error.

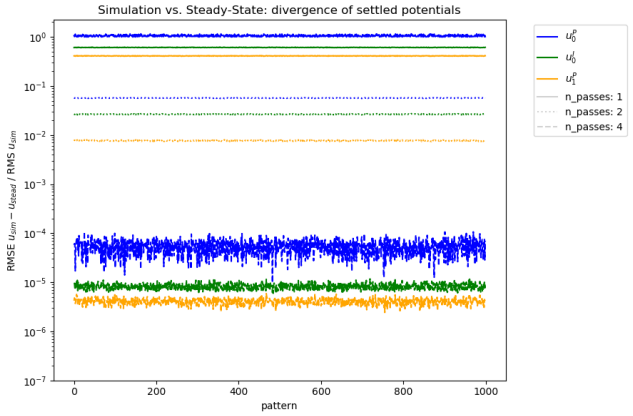
2.1.3 Steady-State-Approximation

We now turn to evaluate the validity of the steady-state approximation approach. To this end we run a 4x120x3 sigmoid-network on the YinYang-training data [1] (with nudging enabled) with the following parameters:

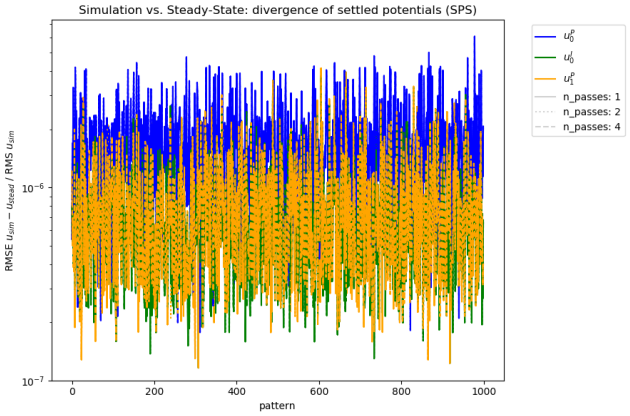
```
{"dims": [4, 120, 3], "dt": 0.1, "gl": 0.1, "gb": 1.0, "ga": 0.8, "gd": 1.0,
 "gsom": 0.8, "eta": {"up": [0.1, 0.01], "pi": [0.01, 0], "ip": [0.02, 0]},
 "bias": {"on": False, "val": 0.5},
 "init_weights": {"up": 0.5, "down": 0.5, "pi": 0.5, "ip": 0.5}, "tau_w": 30,
 "noise": 0, "t_pattern": 100, "out_lag": 80, "tau_0": 3, "learning_lag": 0,
 "reset_deltas": False}
```

The upper left panel in Fig. 5 shows the mean deviation of the soma potentials of the simulated and the steady-state network (for all three neuron types present) when plasticity is turned off (all ηs are zero). The experiment is repeated for three different values of the SteadNet parameter *n_passes*, which gives the number the cycles in the steady-state approximation algorithm (cf. chapter 1.5). As one would expect, a higher value leads to a better approximation. Further note that no additional error builds up when multiple patterns are applied. In the upper right panel we see the same quantities but for a network initialized in the self-predicting state. Here increasing the parameter *n_passes* doesn't lead to a significant decrease of the error. This can be anticipated since in the self-predicting state feeback contributions, except for the contribution of the target signal, mostly cancel.
For the lower panels plasticity was switched on. One can observe how the SteadNet parameter *n_exposures* ($= t_{pattern}/dT$) leads to decreasing deviation between the weights of the simulation and the steady-state approximation. There is, however, little difference between a completely random and a self-predicting initial state of the network. In contrast to the soma potentials, the weight-error increasing with the number of patterns presented to the network.
Finally, note that the presented behavior strongly depends on the network parameters and might

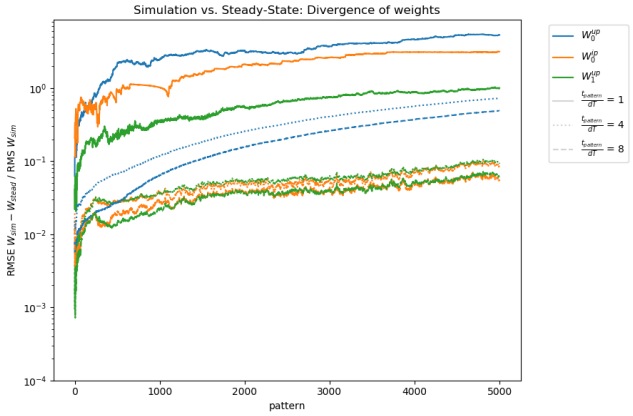
[5]The deviation of the two vector or matrix quantities $\boldsymbol{a}$ and $\boldsymbol{b}$ w.r.t. $\boldsymbol{a}$ is defined as $\mathrm{dev}(n) = \sqrt{\sum_i [(a_i(n) - b_i(n))^2 / \sum_n a_i(n)^2]}$, where i runs over the indices and n is the pattern index. For the simulation-quantities (with time-dependence within one pattern) n is associated with the last value per pattern.



(a) Deviation[5] of the simulated from the approximation soma potentials, when plasticity is switched off.



(b) Deviation of the simulated from the approximation soma potentials, when plasticity is switched off, but the network is initialized in the self-predicting state.



(c) Deviation of the simulated from the approximation weights, when plasticity is switched on.

(d) Deviation of the simulated from the approximation weights, when plasticity is switched on, but the network is initialized in the self-predicting state.

Figure 5: Validation of the steady-state network approximation.

be significantly worse for larger weights or feedback conductances. Also a ReLU-like activation function would lead to worse deviations in most cases.
These results suggest that, since weight deviation grows with the amount of training samples, the steady-state network is not capable of replacing the full simulation. However, since its training process is sped up tremendously, the model can still become extremely helpful when it comes to initial hyperparameter search. As will be explored below, under certain circumstances, the steady-state network is capable of reproducing the achieved accuracy of the full simulation with an absolute error below 1%.

2.2 YinYang-Task

For the YinYang-task it was found that the plasticity induced during the settling phase of the network restricted the performance of the network significantly. With a learning-lag of 20ms the best set of parameters found was

```
{"dims": [4, 120, 3], "dt": 0.1, "gl": 0.1, "gb": 1.0, "ga": 0.28, "gd": 1.0,
 "gsom": 0.34, "eta": {"up": [6.1, 0.00012], "pi": [0, 0], "ip": [0.00024, 0]},
```

```
"bias": {"on": True, "val": 0.5},
"init_weights": {"up": 0.1, "down": 1, "pi": 1, "ip": 0.1}, "tau_w": 30, "noise": 0,
"t_pattern": 100, "out_lag": 80, "tau_0": 3, "learning_lag": 20,
"reset_deltas": False},
```

where the parameters $g_{som}, g_A, \eta_1^{up}$ and η_2^{up} where optimized ($\eta^{ip} = 2\eta_2^{up}$). For simplicity the network was initialized in the self-predicting state and lateral plasticity of the inter- to pyramidal neuron connections was switched off (this condition will be relaxed eventually). The network was then trained on the YinYang-dataset (with normal and inverted coordinate inputs), where the labels where one-hot coded to the three output neurons of the network (logical zero was mapped to 0.1 and logical one to 1.0). A training set of 6000 samples was used and the network was trained for 45 epochs. After each epoch the progress was validated on a separate validation set (choose 100 samples at random from the validation set of size 600). With statistics taken from 10 runs with different seeds the network achieved an accuracy of

$$(96.7 \pm 0.8)\%^6.$$

To further investigate the impact of the learning-lag the runs were repeated for different learning-lags and pattern durations $t_{pattern}$ (therefore also *out_lag* was adapted). The upper panels in Fig. 6 show the results for *reset_deltas* (determines whether to reset the weight updates Δ after each pattern or not) turned on and off, respectively. Most importantly this results indicate that simply increasing $t_{pattern}$ does not suffice to remove the unwanted plasticity induced during settling. One might argue that a simultaneous decrease of the learning-rates would lead to a significant improvement, however this comes at the cost of training-time. Further note that increasing learning-lag leads to a, so far unexplainable, dip in performance around 5 ms. For low $t_{pattern}$ this effect is overcome by resetting the Δs after each pattern, as can be observed in the left upper panel.
Furthermore, the impact of the effect of the parameters τ_0 and τ_w was investigated. The results are presented in Fig. 6 in the lower panels. For the pattern transition time constant τ_0 only a small range was found to produce sensible accuracy. In this range, however, the accuracy seems to be hardly affected by the varying parameter. For the plasticity time constant τ_w hardly any impact on the accuracy was found over the whole parameter range investigated. Of course this results are strictly only valid in the context of the remaining parameters and the behavior may change drastically for another point in the parameter space.
Finally, Fig. 7 demonstrates that learning is still possible if the network is not initialized in the self-predicting state. To obtain this results the parameter η^{pi} was optimized, which yielded $\eta^{pi} = 4.22 \cdot 10^{-4}$. Again statistics over 10 different seeds gives an achieved accuracy of

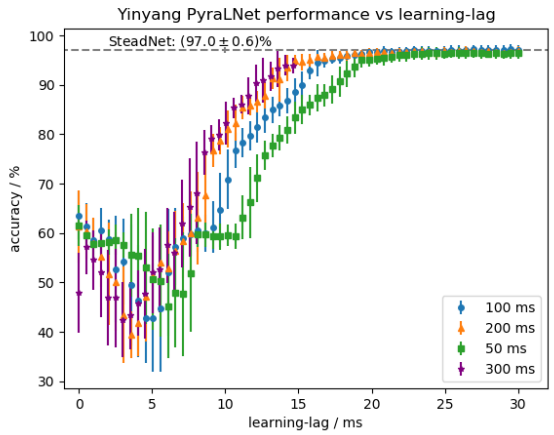
$$(97.2 \pm 1.1)\%.^7$$

3 Conclusion

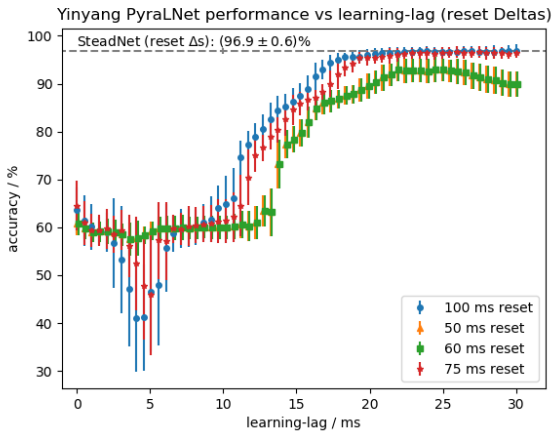
The main result is the implementation (PyraLNet) of a full simulation of the network proposed in [2]. Furthermore, a steady-state approximation (SteadNet) of the network was also implemented.

[6] more precisely this is the result for *learning_lag* = 20.34 ms

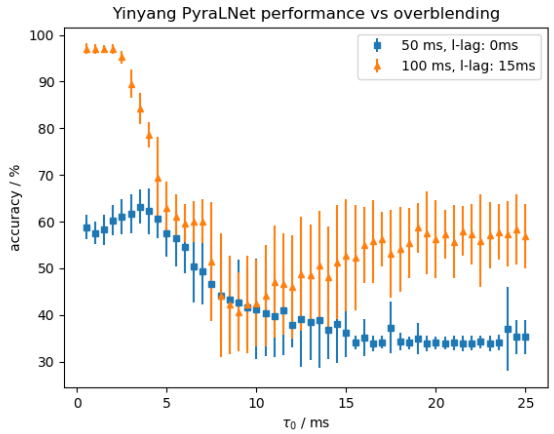
[7] This is a 0.5% increase as compared to the previous case with self-predicting initialization. However, note that training was also extended by 10 epochs.



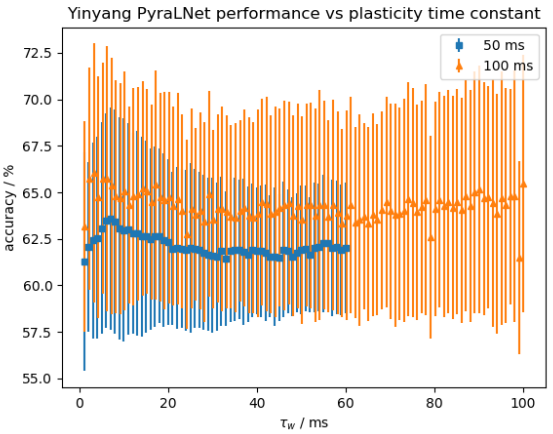
(a) Test set accuracy versus learning-lag for different $t_{pattern}$. See main text for parameter details.



(b) Test set accuracy versus learning-lag for different $t_{pattern}$, but with *reset_deltas* switched on.



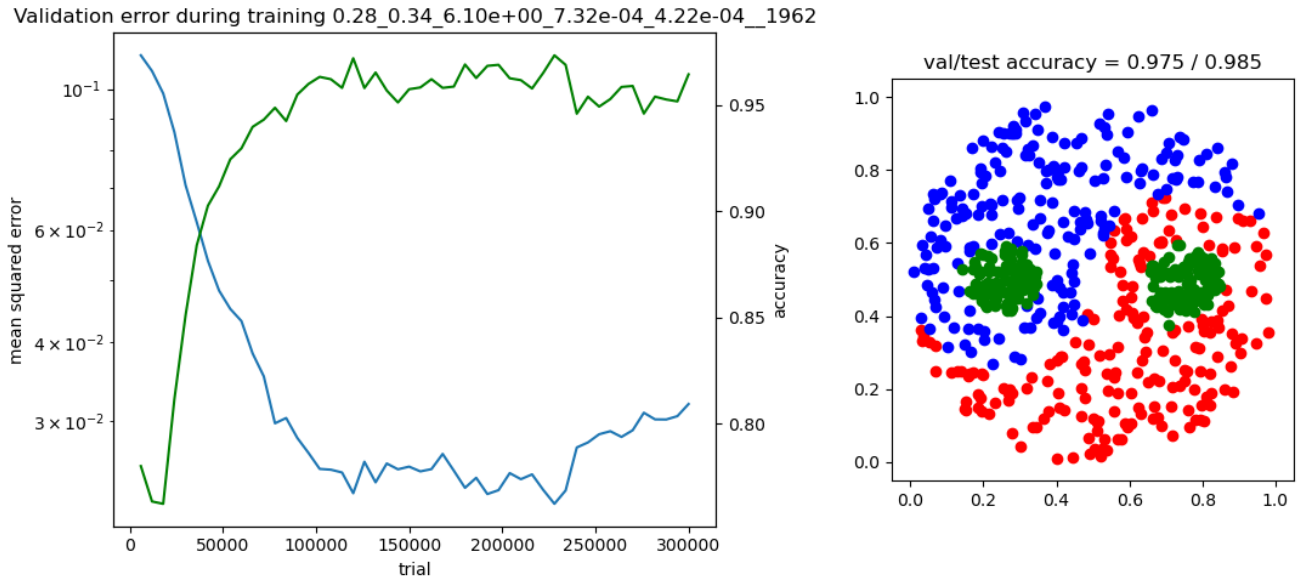
(c) Test set accuracy versus pattern transition time constant τ_0.



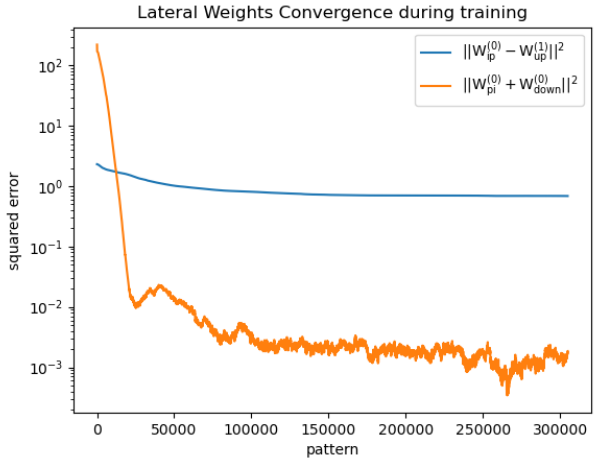
(d) Test set accuracy versus plasticity time constant τ_w.

Figure 6: Impact of different parameters on classification performance for the YinYang-task.

Both implementations with examples and all the code used to generate the plots for this report are available under [1].
By first solving a simple regression task and then solving the YinYang-task the capability of the network to perform deep learning has been demonstrated. Since, the full simulation leads to rather slow training the steady-state approximation proved to be handy. While it is not capable of predicting the weight evolution of the full simulation on the element level, it still indicates if the chosen network architecture and parameters are able to solve the problem at hand effectively. Due to the significant boost in training speed it is perfectly useful for hyper-parameter search. Finally, when a learning-lag (plasticity is paused for certain time after each new pattern) is introduced in the full simulation, both methods, PyraLNet and SteadNet, achieve matching results with less than 1% deviation (on the YinYang-dataset). Since identifying a parameter set for which the YinYang-task is solved efficiently was not successful without introducing learning-lag, the significance of this very parameter may not be underestimated. It is therefore not clear if results achieved with the steady-state approximation, e.g. on MNIST, can be reproduced with the full simulation in its bare formulation. Introducing different learning phases may therefore be unavoidable, coming at the cost of reducing the bio-plausibility of the network architecture.



(a) Validation output potential MSE and classification accuracy during training on the left. The right side shows the accuracy achieved on the validation and test dataset. The picture is obtained on the test set. Note that training lasted for 55 epochs.



(b) Evolution of the self-predicting state during training.

Figure 7: Solving the YinYang-task from a randomly initialized network.

References

[1] Paul Hill. implementation source-files: some-site.html, 2020.

[2] João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic error backpropagation in deep cortical microcircuits. 2017.

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