On the Performance of the Nonsynaptic Backpropagation for Training Long-term Cognitive Networks
Nápoles, Gonzalo; Grau, ISEL; Concepcion, Leonardo; Salgueiro, Yamisleydi

Published in:
Proceedings of the 11th International Conference on Pattern Recognition Systems

DOI:
10.1049/icp.2021.1434

Publication date:
2021

Document Version:
Accepted author manuscript

Citation for published version (APA):

Copyright
No part of this publication may be reproduced or transmitted in any form, without the prior written permission of the author(s) or other rights holders to whom publication rights have been transferred, unless permitted by a license attached to the publication (a Creative Commons license or other), or unless exceptions to copyright law apply.

Take down policy
If you believe that this document infringes your copyright or other rights, please contact openaccess@vub.be, with details of the nature of the infringement. We will investigate the claim and if justified, we will take the appropriate steps.
On the Performance of the Nonsynaptic Backpropagation for Training Long-term Cognitive Networks

Gonzalo Nápoles¹, Isel Grau², Leonardo Concepción³, Yamisleydi Salgueiro⁴

¹Department of Cognitive Science & Artificial Intelligence, Tilburg University, The Netherlands
²Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Belgium
³Faculty of Business Economics, Hasselt University, Belgium
⁴Department of Computer Science, Faculty of Engineering, Universidad de Talca, Campus Curicó, Chile

Abstract

Long-term Cognitive Networks (LTCNs) are recurrent neural networks for modeling and simulation. Such networks can be trained in a synaptic or nonsynaptic mode according to their goal. Nonsynaptic learning refers to adjusting the transfer function parameters while preserving the weights connecting the neurons. In that regard, the Nonsynaptic Backpropagation (NSBP) algorithm has proven successful in training LTCN-based models. Despite NSBP’s success, a question worthy of investigation is whether the backpropagation process is necessary when training these recurrent neural networks. This paper investigates this issue and presents three nonsynaptic learning methods that modify the original algorithm. In addition, we perform a sensitivity analysis of both the NSBP’s hyperparameters and the LTCNs’ learnable parameters. The main conclusions of our study are i) the backward process attached to the NSBP algorithm is not necessary to train these recurrent neural systems, and ii) there is a nonsynaptic learnable parameter that does not contribute significantly to the LTCNs’ performance.

Keywords: neural cognitive mapping, long-term cognitive networks, nonsynaptic learning.

1 Introduction

The modeling and simulation of complex systems allow experts to explore the outcome of hypothetical scenarios before they are observed. These systems often involve many components that may interact with each other. The behavior of complex systems is intrinsically challenging to model due to the dependencies, nonlinearity, and feedback loops among their parts.

Fuzzy Cognitive Maps (FCMs) have been used to model these problems since they allow for human-machine interaction. This means that we could enhance the original model with the aid of historical data. However, in practical terms, their learning algorithms either produce poor results or overwrite the knowledge provided by the expert during the modeling phase. Hence, the main challenge of these models is how to train the network using both expert knowledge and historical data.

To overcome the FCMs’ drawbacks, Nápoles et al. introduced the Short-term Cognitive Networks and a learning principle referred to as nonsynaptic learning. Instead of adjusting the weight matrix, the nonsynaptic learning focuses on the transfer function parameters controlling neurons’ excitation degree. This principle allows experts to inject domain knowledge directly into the network in the form of weights or constraints on the neurons’ activation values, yet the model is able to produce low simulation errors. Recently, the authors in presented a new method termed Long-term Cognitive Networks (LTCNs) to handle longer dependencies among the problem variables. In this neural cognitive model, each neuron’s transfer function parameters are learned through a Nonsynaptic Backpropagation (NSBP) algorithm.

However, similarly to other backpropagation learning algorithms, NSBP might fail when dealing with very long dependencies since the error signal that arrives at the first abstract layers is weak. If that happens, the network will stop learning. Interesting results reported in and further explored in have shown that propagating the error backward through a random channel or suppressing the deep backward pass are suitable options to convey the error signal to the first layers.

In this paper, we study the performance of the NSBP learning algorithm for LTCN-based models under the hypothesis that the backward pass might not be needed at all. Overall, our paper makes two main contributions. Firstly, we propose three variants of the NSBP algorithm, which modify the error backpropagation process to obtain lower simulation errors. Secondly, we conduct several simulations using 35 pattern completion problems and conclude that not all trainable parameters contribute equally to the algorithm’s performance.

2 Long-term Cognitive Networks

In this section, we elaborate on the construction process and the reasoning process of LTCN-based models. Besides, we describe the NSBP algorithm.

2.1 Network construction

In this model, each neural concept  denotes a domain variable while weights represent the rate of change in the conditional mean of with respect to , assuming that the activation values of the remaining neurons impacting are
This function is defined as follows: such that $f$ is impacted by other neural concepts, we can compute a global $i$ the $k$-th variable for the $i$-th neuron in each iteration for a given initial stimulus, which is one of the goals of explainable artificial intelligence.

Another approach is to discover the network structure from data by employing an automated or semi-automated construction procedure. For example, the rate of change in the condition procedure. For example, the rate of change in the condition

$$\begin{align}
  w_{ji} &= \frac{K \sum_k x_i(k) x_j(k) - \sum_k x_i(k) \sum_k x_j(k)}{K \left( \sum_k x_i(k)^2 \right) - \left( \sum_k x_j(k)^2 \right)^2} \\
  r_{ji} &= \frac{\sum_k x_i(k)^2 \sum_k x_j(k) - \sum_k x_i(k) x_j(k) \sum_k x_j(k)}{K \left( \sum_k x_i(k)^2 \right) - \left( \sum_k x_j(k)^2 \right)^2}
\end{align}$$

while $r_{ji}$ is given by

where $K$ is the number of instances and $x_i(k)$ is the value of the $i$-th variable for the $k$-th instance.

In this formulation, $r_{ji}$ is the local bias associated with the $j$-th incoming connection. Since each neuron $C_l$ will likely be impacted by other neural concepts, we can compute a global bias $b_l$ as the aggregation of local ones. For example, $b_l = U_l$ if $\sum_{j=1}^M r_{ji} > U_l$; $b_l = -U_l$ if $\sum_{j=1}^M r_{ji} < U_l$; or $b_l = \sum_{j=1}^M r_{ji} | r_{ji} | \leq U_l$, where $U_l$ denotes the maximal activation value allowed for the $i$-th neuron, which should be defined by domain experts.

### 2.2 Reasoning mechanism

The recurrent reasoning process of LTCNs makes it possible to associate each iteration with an abstract layer having exactly $M$ abstract hidden neurons. This means that LTCNs can be seen as feed-forward neural networks with width $M$ and length $T$, where $T$ is the number of iterations. However, the layers are not necessarily fully connected, while the weights do not change from an abstract layer to another. Equation [3] displays how to compute the activation value $A_i(k)$ of the $i$-th neural concept in each iteration for a given initial stimulus.

$$A_i^{(t+1)}(k) = f_i^{(t+1)} \left( \sum_{j=1}^M w_{ji} A_j^{(t)}(k) \right)$$

such that $f_i^{(t+1)}$ is the transfer function adopted to confine the activation value of each neuron to the desired interval $[L_i, U_i]$. This function is defined as follows:

$$f_i^{(t)}(x) = L_i + \frac{U_i - L_i}{1 + q_i^{(t)} e^{-\lambda_i^{(t)}(x-h_i^{(t)})}}/v_i^{(t)}.$$ where $q_i > 0$, $\lambda_i > 0$, $h_i \in \mathbb{R}$ and $v_i > 0$ are parameters to be adjusted during the nonsynaptic learning phase. In this function, $q_i$ is a parameter related to the initial condition, $\lambda_i$ is the function slope, $h_i$ stands for the sigmoid offset and $v_i$ regulates toward which asymptote the maximum growth occurs. According with the construction method described earlier, $h_i^{(0)} = b_i$, $\forall i$.

### 2.3 Nonsynaptic learning

The NSBP is devoted to optimizing the parameters $\lambda_i^{(t)}$, $h_i^{(t)}$, $q_i^{(t)}$ and $v_i^{(t)}$, whereas the weights $w_{ij}$ and bounds $L_i$ and $U_i$ remain unaltered. Overall, the learning task consists of adjusting the shape of the transfer function associated with the $i$-th neuron in each iteration. This can be done by computing the parameter set:

$$\Theta = \left\{ \theta_i^{(t)} = \left( \lambda_i^{(t)}, h_i^{(t)}, q_i^{(t)}, v_i^{(t)} \right) \right\}.$$ When computing the partial derivative of the network error with respect to $A_i^{(t)}$, we need to determine whether the iteration $t$ is the final one or not.

**Case 1.** When $t = T$, the partial derivative of the global error $E$ is computed as follows:

$$E = \sum_{i=1}^M \frac{(Y_i(k) - A_i^{(t)})^2}{2}$$

$$\frac{\partial E}{\partial A_i^{(t)}} = -(Y_i(k) - A_i^{(t)})$$

where $Y_i(k)$ is the actual value of the $i$-th variable according to the $k$-th training example, while $A_i^{(t)}$ is the activation value of the $i$-th neuron in the current iteration.

**Case 2.** When $1 < t < T$, the partial derivative of the global error $E$ is calculated as follows:

$$\frac{\partial E}{\partial A_i^{(t)}} = \sum_{j=1}^M \frac{\partial E}{\partial A_j^{(t+1)}} \times \frac{\partial A_j^{(t+1)}}{\partial A_i^{(t)}} \times w_{ij}, \quad i \neq j$$

$$= \sum_{j=1}^M \frac{\partial E}{\partial A_j^{(t+1)}} \times \frac{\partial A_j^{(t+1)}}{\partial A_i^{(t+1)}} \times \frac{\partial A_i^{(t+1)}}{\partial A_i^{(t)}}$$

$$= \sum_{j=1}^M \frac{\partial E}{\partial A_j^{(t+1)}} \times \frac{\partial A_j^{(t+1)}}{\partial A_i^{(t+1)}} \times w_{ij}$$

where $A_j^{(t+1)}$ is the raw value of the $j$-th neuron,

$$\frac{\partial A_j^{(t+1)}}{\partial A_i^{(t+1)}} = \frac{(U_j - L_j) \lambda_j^{(t+1)} \Gamma_j^{(t+1)}}{v_j^{(t+1)} \left( 1 + \Gamma_j^{(t+1)} \right)^{t+1/v_j^{(t+1)}}}$$

and

$$\Gamma_j^{(t+1)} = q_j^{(t)} e^{\lambda_j^{(t+1)}(-A_j^{(t+1)} + h_j^{(t+1)}).}$$
Once \( \partial E/\partial A_i^{(t)}(k) \) have been calculated, we need to obtain the partial derivatives of the global error with respect to the target parameters \( \theta_i^{(t)}(p) \in \Theta \),

\[
\frac{\partial E}{\partial \theta_i^{(t)}(p)} = \frac{\partial E}{\partial A_i^{(t)}(k)} \times \frac{\partial A_i^{(t)}(k)}{\partial \theta_i^{(t)}(p)},
\]

such that \( p \) is the index of the parameter of the \( i \)-th sigmoid function in the current abstract layer.

Equations (12) to (15) portray the partial derivatives of the neuron’s activation value with respect to each sigmoid function parameter in the current iteration,

\[
\begin{align*}
\frac{\partial A_i^{(t)}}{\partial x_i^{(t)}} &= \frac{(U_i - L_i) \Gamma_i^{(t)} \left( \tilde{A}_i^{(t)} - h_i^{(t)} \right)}{v_i^{(t)} \left( 1 + \Gamma_i^{(t)} \right)^{1+1/v_i^{(t)}},} \\
\frac{\partial A_i^{(t)}}{\partial h_i^{(t)}} &= -\frac{(U_i - L_i) \Gamma_i^{(t)} \lambda_i^{(t)}}{v_i^{(t)} \left( 1 + \Gamma_i^{(t)} \right)^{1+1/v_i^{(t)}},} \\
\frac{\partial A_i^{(t)}}{\partial q_i^{(t)}} &= -\frac{(U_i - L_i) \Gamma_i^{(t)} v_i^{(t)} \left( 1 + \Gamma_i^{(t)} \right)^{1+1/v_i^{(t)}},} \\
\frac{\partial A_i^{(t)}}{\partial \theta_i^{(t)}} &= \frac{(U_i - L_i) \log \left( 1 + \Gamma_i^{(t)} \right)}{\left( v_i^{(t)} \right)^2 \left( 1 + \Gamma_i^{(t)} \right)^{1/v_i^{(t)}},}
\end{align*}
\]

Equation (16) illustrates the update to be performed to the sigmoid parameters \( \lambda_i^{(t)}, h_i^{(t)}, q_i^{(t)} \) and \( v_i^{(t)} \) associated with the \( C_i \) neuron in the current iteration, where \( \beta \) represents the momentum and \( \eta \) the learning rate,

\[
\Delta \theta_i^{(t)}(p) = \beta \left( \Delta \theta_i^{(t)}(p) \right) - \eta \times \frac{\partial E}{\partial \theta_i^{(t)}(p)}.
\]

The parameters are iteratively updated until a maximal number of epochs \( L \) is reached. Observe that the learning capacity of LTCNs trained with the NSBP algorithm depends on \( 4 \times T \times M \) parameters, however, we might compute similar results using fewer parameters.

3 Nonsynaptic learning variants

A pivotal step in the NSBP learning algorithm refers to propagating the error signal backward from the abstract output layer to the abstract hidden ones.

Figure 1 shows the error flow in the backward pass for an unfolded LTCN model comprised of two neurons \( C_1 \) and \( C_2 \), while \( E_1 \) and \( E_2 \) denote their errors in the last abstract layer, respectively. The error signal travels through the connections and neurons such that we can use it to update the nonsynaptic parameters.

The weights used during the forward pass are the same as the ones used for the backward pass. However, results reported in [3] and further explored in [6] concluded that such symmetry might not be required to train deep neural networks. This leads to the idea of replacing the weights used in the backward pass with random weights. Random backpropagation shows that the information on the upper weights might not be necessary at all.

This section further elaborates on this idea and describes several nonsynaptic learning algorithms that modify the NSBP’s backward pass. These variants differ from each other in the approach we use to compute the partial derivatives of the error with respect to the activation values of abstract hidden neurons.

3.1 Random NSBP (R-NSBP) algorithm

In this algorithm, the weight matrix used in the backward pass is replaced with a matrix comprised of normally distributed random numbers. The key difference with the approach in [6] is that such numbers are derived based on the weights estimated during the network construction phase. In short, the mean of the probability distribution of each Gaussian is initialized with the weights used during the forward pass. Equation (17) formalizes how to compute \( \partial E/\partial A_i^{(t)} \) in this case,

\[
\frac{\partial E}{\partial A_i^{(t)}} = \sum_{j=1}^{M} \frac{\partial E}{\partial A_j^{(t+1)}} \times \frac{\partial A_j^{(t+1)}}{\partial A_i^{(t+1)}} \times \tilde{w}_{ij}
\]

where \( \tilde{w}_{ij} \) is the Gaussian random number generated with the following probability distribution function:

\[
f(x \mid \mu_{ij}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu_{ij})^2}{2\sigma^2}}
\]

such as \( \mu_{ij} = w_{ij} \) denotes the mean while \( \sigma^2 = 0.2 \) represents the variance. If desired, we can force such random weights to share the same sign as the weights defined during the network construction step.

3.2 Skipped NSBP (S-NSBP) algorithm

Another alternative when updating the parameters in the \( t \)-th abstract hidden layer is to use the partial derivative of the total error with respect to the neuron’s output after \( T \) iterations. This suggests that we **skip** \( T - t \) abstract layers in the backward process such that we can deliver the error signal directly.
to the current abstract hidden layer. In this algorithm, the partial derivative of the global error with respect to the neuron’s output in the current abstract layer can be computed as follows:

$$\frac{\partial E}{\partial A_i^{(t)}} = \sum_{j=1}^{M} - (Y_j(k) - A_j^{(T)}) \times w_{ij}.$$  \hspace{1cm} (19)

This method is deemed convenient when learning very long dependencies as the gradient tends to vanish/explode, so the error signal that reaches the first abstract layers is not informative enough. Furthermore, the S-NSBP learning algorithm does reflect the recurrent reasoning process performed during the forward pass.

3.3 Random-Skipped NSBP (RS-NSBP) algorithm

This algorithm combines the strategies explained above. Being more explicit, during the backward pass, we replace the weights by normally distributed random numbers with mean $\mu_{ij} = w_{ij}$ and variance $\sigma^2 = 0.2$, while allowing for the skipping operation. The partial derivative related to the $i$-th neuron is computed as follows:

$$\frac{\partial E}{\partial A_i^{(t)}} = \sum_{j=1}^{M} - (Y_j(k) - A_j^{(T)}) \times \tilde{w}_{ij}.$$  \hspace{1cm} (20)

It is worth mentioning that, in all random variants, we have adopted a relatively small variance to prevent weights in the backward process to be excessively different from the ones used during the forward pass.

4 Numerical simulations

The experiments in this section are oriented to determining whether the backward step is actually necessary and investigating the effect of both nonsynaptic parameters on the algorithm’s performance.

4.1 Benchmark problems

Aiming at conducting the simulations, we use the same datasets used in [3]. These datasets are traditional pattern classification problems adapted to the simulation problem where we need to predict the values of some variables from the values of others. The number of variables ranges from 3 to 22, whereas the number of instances goes from 106 to 625. These datasets are relatively small, as usually happens in real-world scenarios where experts can provide a limited number of training examples.

It is relevant to mention that each variable in the original dataset has a probability of 0.2 of being an output variable. The output variables are determined instance-wise. Thus, the LTCN-based models are fed with altered instances where the values of output variables are replaced with zeros. The original patterns are preserved aside for comparison purposes. In a nutshell, the learning problem narrows down to reconstruct the altered instances. The mean squared error (MSE) is used as the performance metric. All simulation errors reported in this paper are calculated across a 10-fold cross-validation process to explore the generalization capability of each model. Moreover, all random numbers were generated with the same seed to ensure reproducible experiments.

4.2 Sensitivity analysis

Before drawing any conclusion about the proposed nonsynaptic learning variants, it would be convenient to study the effect of hyperparameters on the algorithms’ performance. The hyperparameters to be explored are the learning rate $\eta$ and the momentum $\beta$ of the stochastic gradient powering the NSBP algorithm.

Figure 2 shows the average MSE values across datasets obtained with different learning algorithms. The learning rate ranges from 0.001 to 0.01, with step 0.001, whereas the momentum goes from 0.8 to 0.95, with step 0.01. The remaining hyperparameters are initialized as follows: $\lambda_i = e_i = q_i = 1$, $h_i$ is computed as explained in Section 2.1, the number of training epochs is set to 200 while the number of abstract layers $T$ is set to five.

Figure 2: Heat maps for the LTCN model using different NSBP variants when changing the learning rate and the momentum. The color scale moves from blue to red such that the former indicates smaller average error values on the 35 datasets.

The results show that NSBP, R-NSBP, and RS-NSBP report the lowest errors when using small momentum and learning rate values. However, in the R-NSBP algorithm, the lowest simulation errors are obtained when using relatively small momentum values and rather large learning rate values. This seems related to the fact that this algorithm does not propagate the error backward through all abstract layers.
4.3 Nonsynaptic learning variants

In this subsection, we compare the nonsynaptic learning variants in terms of simulation error. In order to do that, we will compute the median and the best simulation error obtained by each algorithm across all parameter settings explored in the previous subsection. Figure 3 portrays the MSE values reported in each case.

Next, we adopt the lowest simulation errors to conduct the statistical analysis. The Friedman test \([8]\) fails to reject the \(H_0\) hypothesis \((p-value = 2.61E-11 < 0.05)\) for a 95% confidence interval. This suggests that these methods perform comparably in terms of error. Moreover, the pairwise analysis using the Wilcoxon signed-rank test \([8]\) and the corrected \(p\)-values reported in Table 2 confirm that there are no significant differences between the NSBP algorithm and the proposed learning methods. The negative ranks \((R^-)\) show that the methods implementing the skipping operation perform slightly better.

Table 1: Pairwise analysis for learning algorithms with NSBP being the control method.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(p)-value</th>
<th>(R^-)</th>
<th>(R^+)</th>
<th>Holm</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-NSBP</td>
<td>1.69E-01</td>
<td>14</td>
<td>3.07E-01</td>
<td>Fail to reject</td>
<td></td>
</tr>
<tr>
<td>RS-NSBP</td>
<td>1.96E-01</td>
<td>13</td>
<td>5.07E-01</td>
<td>Fail to reject</td>
<td></td>
</tr>
<tr>
<td>R-NSBP</td>
<td>8.96E-01</td>
<td>18</td>
<td>8.96E-01</td>
<td>Fail to reject</td>
<td></td>
</tr>
</tbody>
</table>

The simulations support the main hypothesis of this research: backpropagating the error signal through the inner (abstract) layers is not necessary to convey the error to the first hidden layers. This finding opens new research avenues toward training recurrent neural systems with less effort.

4.4 Sigmoid function parameters

The last experiment of this paper is devoted to studying the contribution of each sigmoid function parameter to the learning capability of LTCN models when trained in a nonsynaptic fashion. This experiment will adopt the S-NSBP algorithm to train the models since it reported the lowest simulation errors.

Figure 4 shows the average MSE over the 35 datasets when learning one, two, three, and all sigmoid function parameters. In this simulation, the baseline refers to optimizing all parameters. The reader can observe that the model reports higher errors than the baseline if only one parameter is adjusted. The simulation errors are quite comparable to the baseline when optimizing the function slope (denoted with \(l\)) and the function offset (denoted with \(h\)). Finally, we reproduced the results obtained with the baseline if the parameter related to the initial condition \((q)\) is excluded from the learning process while retaining the remaining ones.

The Friedman test shows significant performance differences \((p-value = 2.61E-11 < 0.05)\) when optimizing one parameter at a time. The corrected \(p\)-values and the \(R^-\) values associated with the Wilcoxon test in Table 2 show that the LTCNs underperform when adjusting a single parameter.

Table 2: Pairwise analysis for variants that optimize one parameter with LTCN-all being the control method.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(p)-value</th>
<th>(R^-)</th>
<th>(R^+)</th>
<th>Holm</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTCN-t</td>
<td>4.95E-07</td>
<td>33</td>
<td>2</td>
<td>1.98E-06</td>
<td>Reject</td>
</tr>
<tr>
<td>LTCN-q</td>
<td>5.86E-07</td>
<td>31</td>
<td>4</td>
<td>1.98E-06</td>
<td>Reject</td>
</tr>
<tr>
<td>LTCN-h</td>
<td>2.59E-06</td>
<td>29</td>
<td>6</td>
<td>5.18E-06</td>
<td>Reject</td>
</tr>
<tr>
<td>LTCN-v</td>
<td>3.29E-06</td>
<td>30</td>
<td>5</td>
<td>5.18E-06</td>
<td>Reject</td>
</tr>
</tbody>
</table>

The Friedman test indicates significant differences \((p-value = 1.09E-14 < 0.05)\) among these learning configurations when optimizing two parameters. The corrected \(p\)-values for the Wilcoxon test in Table 3 still report significant performance differences between the control method and the variant that optimizes the function offset and the function slope together.

Table 3: Pairwise analysis for variants that optimize two parameters with LTCN-all being the control method.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(p)-value</th>
<th>(R^-)</th>
<th>(R^+)</th>
<th>Holm</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTCN-hq</td>
<td>8.94E-07</td>
<td>32</td>
<td>3</td>
<td>3.36E-06</td>
<td>Reject</td>
</tr>
<tr>
<td>LTCN-hv</td>
<td>2.21E-06</td>
<td>29</td>
<td>6</td>
<td>1.10E-05</td>
<td>Reject</td>
</tr>
<tr>
<td>LTCN-qv</td>
<td>3.56E-06</td>
<td>29</td>
<td>6</td>
<td>1.43E-05</td>
<td>Reject</td>
</tr>
<tr>
<td>LTCN-lv</td>
<td>9.05E-06</td>
<td>28</td>
<td>7</td>
<td>2.71E-05</td>
<td>Reject</td>
</tr>
<tr>
<td>LTCN-lq</td>
<td>1.91E-05</td>
<td>28</td>
<td>7</td>
<td>3.82E-05</td>
<td>Reject</td>
</tr>
<tr>
<td>LTCN-hl</td>
<td>6.23E-03</td>
<td>22</td>
<td>13</td>
<td>6.23E-03</td>
<td>Reject</td>
</tr>
</tbody>
</table>

In the simulations with three parameters, the Friedman test suggests rejecting the null hypothesis \(H_0\) \((p-value = 2.39E-10 < 0.05)\). The corrected \(p\)-values in Table 4 reveals that LTCN-hlv is the only configuration reporting no significant differences in performance.
The simulation results have shown that the $q_i$ parameter does not contribute significantly to the algorithm’s performance. This happens because parameters $h_i$ and $q_i$ are not independent of each other. Consequently, it holds that $q_i e^{-\lambda_i(x-h_i)} = e^{ln(q_i)} e^{-\lambda_i(x-h_i)} = e^{-\lambda_i(x-h_i)+ln(q_i)} = e^{-\lambda_i(x-h_i)} + ln(q_i) = h_i$. This means that we can safely remove the $q_i$ parameter from nonsynaptic learning algorithms to lighten their formulation.

5 Concluding remarks

This paper has presented three nonsynaptic learning methods to train LTCN-based models devoted to solving simulation problems. The empirical evidence supports the hypothesis that the backward pass is unnecessary to train a recurrent neural network even in the nonsynaptic context. Instead, we can bring the error from the last abstract layer to the one being processed. Likewise, the simulation results have shown that the sigmoid function parameter related to the initial condition does not contribute significantly to the algorithm’s performance. The results also show that the most critical parameters are the function slope and its offset. It would be interesting to extend this investigation to other types of recurrent neural networks as future work.

Acknowledgements

This paper was partially supported by CONICYT Program FONDECYT de Postdoctorado through project 3200284 and the Artificial Intelligence Research Program of the Flemish Government, Belgium.

References


