# Implicit recurrent networks: A novel approach to stationary input processing with recurrent neural networks in deep learning.

Sebastian Sanokowski,[1*]

[1]Friedrich-Alexander Universität Erlangen-Nürnberg; Department of Physics,
[*] E-mail: sebastian.sanokowski@fau.de

October 22, 2020

The brain cortex, which processes visual, auditory and sensory data in the brain, is known to have many recurrent connections within its layers and from higher to lower layers. But, in the case of machine learning with neural networks, it is generally assumed that strict feed-forward architectures are suitable for static input data, such as images, whereas recurrent networks are required mainly for the processing of sequential input, such as language. However, it is not clear whether also processing of static input data benefits from recurrent connectivity. In this work, we introduce and test a novel implementation of recurrent neural networks with lateral and feed-back connections into deep learning. This departure from the strict feed-forward structure prevents the use of the standard error backpropagation algorithm for training the networks. Therefore we provide an algorithm which implements the backpropagation algorithm on a implicit implementation of recurrent networks, which is different from state-of-the-art implementations of recurrent neural networks. Our method, in contrast to current recurrent neural networks, eliminates the use of long chains of derivatives due to many iterative update steps, which makes learning computationally less costly. It turns out that the presence of recurrent intra-layer connections within a one-layer implicit recurrent network enhances the performance of neural networks considerably: A single-layer implicit recurrent network is able to solve the XOR problem, while a feed-forward network with monotonically increasing activation function fails at this task. Finally, we demonstrate that a two-layer implicit recurrent architecture leads to a better performance in a regression task of physical parameters from the measured trajectory of a damped pendulum.

# 1. Introduction

There is some research, which indicates that human vision for stationary input processing is not merely a feed forward process [1, 2, 3]. Therefore the question arises, whether deep learning may also benefit from recurrent connections when stationary inputs, such as images, are processed. In order to make it plausible, that recurrence does play a role in human vision, we will take the example of the famous illusion "My Wife and My Mother-in-Law", where one can either see a young woman or an old woman in this drawing. It is well known that neurons within the brain or within artificial neural networks act as feature detectors, where the detected features in higher layers are more complex than in lower layers. Therefore, if our brain detects a young woman in this picture, there may be active neurons within lower brain regions, which recognise a necklace or the chin of the young woman. But, if one recognises an old woman, the active features within lower brain regions change and instead other neurons, which may instead detect the nose and the mouth of the old woman, become active. This example illustrates that the final prediction of the human brain influences neural activation in lower brain regions, which can only be explained with recurrent connections. Therefore, by using such recurrent mechanisms in artificial neural networks error correction from higher layers into lower ones may be possible.
In a feed-forward process all of these lower features would have to be active, independent of what the network recognises in the end. While this may be not a problem if this network is merely used for classification, one could assume that incorrectly detected features are a problem in architectures, where lower layers are shared or where lower features are used for some other task. Another advantage of recurrent connections in deep learning may be that, for example, lateral recurrence within a layer would bring some additional interaction within a layer. This may lead to advantages during the training process, where connected neurons may recognise more easily that they are learning similar features and could therefore learn different tasks at an earlier stage during the learning phase. Furthermore, the additional interaction due to recurrence may lead to more computational power of the network. This assumption will be studied in this paper.

# 2. Results

## 2.1. Problems with state-of-the-art recurrent neural networks for stationary input processing

In current state-of-the-art models for recurrent neural networks for stationary input processing, such as images, so called iterative recurrent networks are used. In iterative recurrent networks, like they have been used in O'Reilly et al. [4] and in M. Ernst et al. [5], the recurrent neural activation is updated iteratively over a few time steps.
In order to explain the principle of iterative recurrent networks, we are going to consider the simplest case of a single iterative recurrent neuron $Y$. This neuron receives a stationary input value $X$, which is connected via the weight Q to neuron $Y$. Additionally, $Y$ is recurrently connected to itself through the weight $W$.
The activation of this neuron is then given by

$$Y(t) = f(W \cdot Y(t-1) + Q \cdot X + T),$$

where $T$ is the bias of this neuron and $f(\cdot)$ is the sigmoid activation function. The activation of this neuron is then calculated iteratively by starting at $Y(t=0) = 0$ and calculating the consecutive time steps from there on up to $Y(t=n)$.
Here, we identify the first problem of this method, where it is not clear how many iterative updates should be used. The next problem occurs, when we want to train these recurrent networks with backpropagation. In order to apply gradient descent, it would for example be necessary to calculate the derivative with respect to $W$. In our simple example, we obtain

$$\frac{\partial Y(t)}{\partial W} = f'(\widetilde{X}) \cdot (W \cdot \frac{\partial Y(t-1)}{\partial W} + Y(t-1)),$$

where $\frac{\partial Y(t-1)}{\partial W}$ again depends on $Y(t-2)$ which also depends on $W$. Therefore, in order to obtain the final derivative, the chain rule has to be applied $n$ times up to $Y(t=0)$. Hence, for every iterative time step which is used in this network the derivatives become more complicated and if too many time steps are used, this calculation becomes intractable. That is why in iterative recurrent networks usually only a few iterative updates are used.
However, these iterative networks have an equilibrium, when $\vec{Y}(t) = \vec{Y}(t-1)$, which is often reached after a few iterative updates of the network. But, if the number of neurons within a layer is too large, iterative updates will not converge to this equilibrium.

## 2.2. Intuition of implicit recurrent networks

Instead of using iterative recurrent networks, we use the alternative approach of implicit activation functions with which the equilibrium state can be calculated immediately.
In the simplest case of a single recurrent neuron, the implicit activation function is given by

$$Y = f(W \cdot Y + Q \cdot X + T). \tag{1}$$

Here, in order to obtain the activation of the neuron the equilibrium $Y$ has to be found, where the left and the right side of the equation equal one another.
Next, we will show that this mathematical formulation will make it possible to calculate derivatives directly at the equilibrium state of recurrent neural networks. These derivatives are, in contrast to derivatives of iterative recurrent networks, not computationally expensive.
Now, if we calculate the derivative with respect to $W$, we obtain

$$\frac{\partial Y}{\partial W} = f'(\widetilde{X}) \cdot (W \cdot \frac{\partial Y}{\partial W} + Y),$$

where $\widetilde{X} = W \cdot Y + Q \cdot X + T$ is the overall input of this neuron. Here, the derivative $\frac{\partial Y}{\partial W}$ appears on the right side and on the left side of the equation. Therefore, we obtain a linear equation which has to be solved for $\frac{\partial Y}{\partial W}$, in order to obtain the final derivative. After solving this equation one obtains

$$\frac{\partial Y}{\partial W} = \frac{f'(\widetilde{X}) \cdot Y}{1 - f'(\widetilde{X}) \cdot W}, \tag{2}$$

which fits perfectly to numerical approximations of this derivative, as can be seen in figure 1. Thus, by changing the mathematical formulation of iterative recurrent networks to implicit recurrent networks, it is possible to calculate the derivatives of this neuron in a way which is computationally not expensive.
Such implicit recurrent networks were originally introduced by P. Földiák [6], where they have been used in so called Hebbian neural networks, which are trained with local Hebbian and Anti-Hebbian learning rules. It is well known how to train these networks with Hebbian- and Anti-Hebbian learning rules, as it is shown in [7] or [6], but to our knowledge nobody has formulated a way on how to train these networks with backpropagation. Therefore, in this paper, we find a mathematical formulation for a one-layer implicit recurrent neural network in subsection 3.1, which has lateral connections within the layer. Furthermore, in subsection 3.2 we write down a formulation for a two-layer fully recurrent implicit network which has additional feed-back connections from higher to lower layers.

## 2.3. Tests on the XOR-Problem

It is well known that a single feed-forward neuron, which has a monotonically increasing activation function can not solve the XOR-Problem. Therefore, since a single feed-forward neuron can not solve this problem, a one-layer feed forward network also can not solve this problem because in feed-forward networks there are not interactions between neurons of the same layer. But, in contrast to a feed-forward network a one-layer implicit recurrent network does have interactions within its layer. Therefore, we try to solve the XOR-Problem with a one-layer implicit recurrent

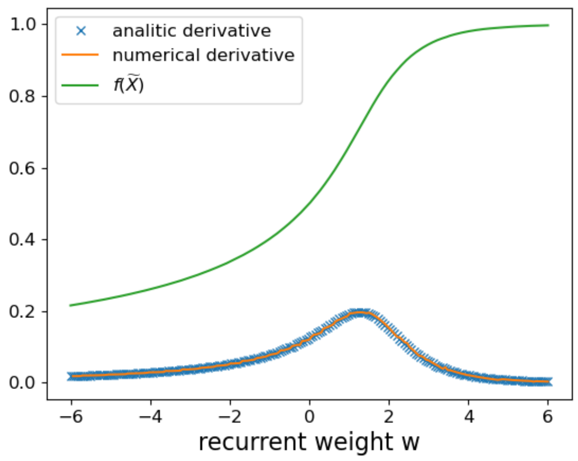
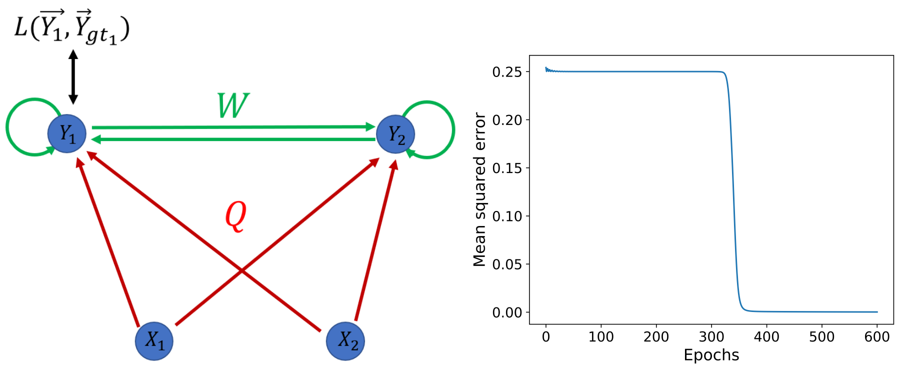


Figure 1.: Comparison of the numerical derivative and the derivative from equation 2. The numerical derivative fits perfectly to the derivative which we have derived. The numerical derivative is calculated by calculating the distance between two neighbouring points of $f(\widetilde{X})$ and dividing it by their distance.



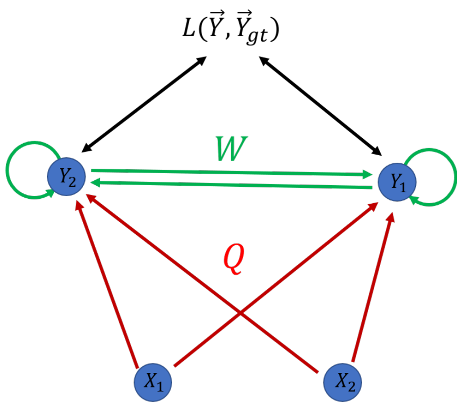
(a) Network architecture

(b) Error over epochs

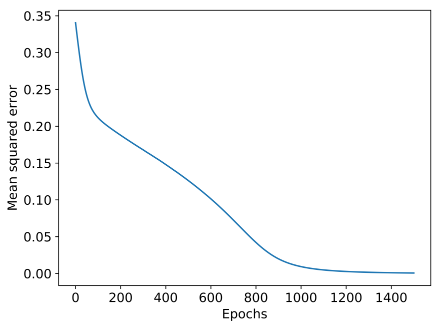
Figure 2.: Training results of a one-layer implicit recurrent network with two neurons. One Neuron $Y_1$ is trained directly on the XOR-Problem. Neuron $Y_2$ is trained indirectly due to recurrence. As can be seen in the error plot over the epochs in sub-figure (b), this network is able to solve this problem with a very low error rate.

| $X_1$ | $X_2$ | $Y_{gt_1}$ | $Y_{gt_2}$ | $Y_1$ | $Y_2$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | Any | 0.01 | 0.98 |
| 0 | 1 | 1 | Any | 0.99 | 0.01 |
| 1 | 0 | 1 | Any | 0.99 | 0.01 |
| 1 | 1 | 0 | Any | 0.01 | 0.00 |

Figure 3.: Tabular with input data, the corresponding ground truth data and the output data after training for the simulation of the one-layer implicit recurrent network on the XOR-Problem. Only the output of neuron $Y_1$ is minimized through the loss function to the ground truth data $Y_{gt_1}$. Here, neuron $Y_2$ is allowed to generate any output. As can be seen neuron $Y_1$ learns an output, which is very close to the ground truth of the XOR-Problem.



(a) Network architecture



(b) Error over epochs

Figure 4.: Training results of a semi-gradient one-layer implicit recurrent network with two neurons. One Neuron $Y_1$ is trained directly on the XOR-Problem and neuron $Y_2$ is trained to a sub-problem. As can be seen in the error plot over the epochs in sub-figure (b), this network is able to solve this problem with a very low error rate.

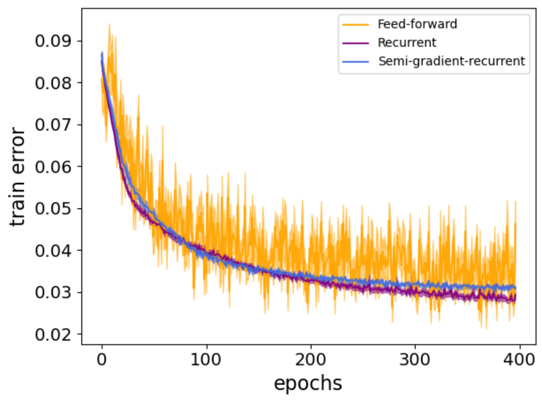
| $X_1$ | $X_2$ | $Y_{gt_1}$ | $Y_{gt_2}$ | $Y_{out_1}$ | $Y_{out_2}$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0.03 | 0.02 |
| 0 | 1 | 1 | 1 | 0.97 | 0.98 |
| 1 | 0 | 1 | 0 | 0.97 | 0.00 |
| 1 | 1 | 0 | 0 | 0.03 | 0.02 |

Figure 5.: Tabular with input data, the corresponding ground truth data and the output data after training for the simulation of the semi-gradient one-layer implicit recurrent network on the XOR-Problem. Here, $Y_1$ is trained directly to the XOR-Problem and $Y_2$ is trained to a simpler logic operation. As can be seen, even with the semi-gradient method neuron $Y_1$ learns an output, which is very close to the ground truth of the XOR-Problem.

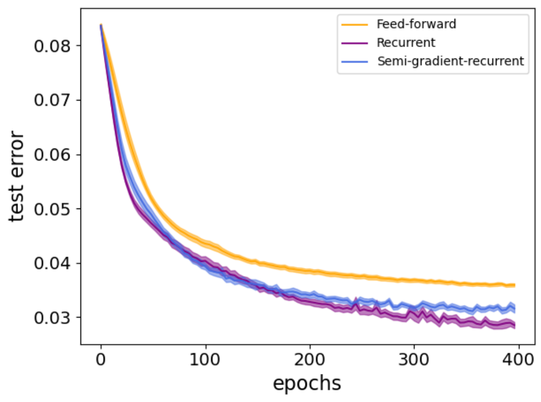
neural network. Here, as depicted in figure 2, we use two neurons and train only neuron $Y_1$ directly on the XOR-Problem through the minimisation of the mean squared error loss function. The other neuron $Y_2$ is trained indirectly due to recurrence. Therefore, the other neuron $Y_2$ simply learns an output, which supports $Y_1$ in solving the XOR-Problem. Results of this simulation are depicted in figure 2 and in tabular 3, where we can see that this network is able to solve the XOR Problem with a very low error. As can be seen in tabular 3, the second neuron $Y_2$ learns the output of a NOR-gate, which is a subgate which supports $Y_1$ in solving the XOR-Problem.
To simplify, we also introduce a semi-gradient method for training such recurrent neural networks, as they are explained in the methods section in 3.3. Due to the approximation, this system loses the ability of indirect training of neurons within the same layer. Therefore, as can be seen in figure 4 and in tabular 5 the output of both neurons have to be trained to a ground truth target. But still, as can be seen on the mean squared error in figure 4 (b) or on the output in tabular 5, the network is still able to solve the XOR-Problem. Thus, despite of the approximation, the network still remains computationally stronger than state-of-the-art feed forward networks.

## 2.4. Performance on a regression problem in physics

Next, we train two-layer feed-forward networks, two-layer implicit recurrent networks and semi-gradient two-layer implicit recurrent networks and compare their performance to one another on a regression problem. Here, we generate the trajectory of a damped oscillator $X(t)$, which can be described by the differential equation



(a) Comparison of the train error

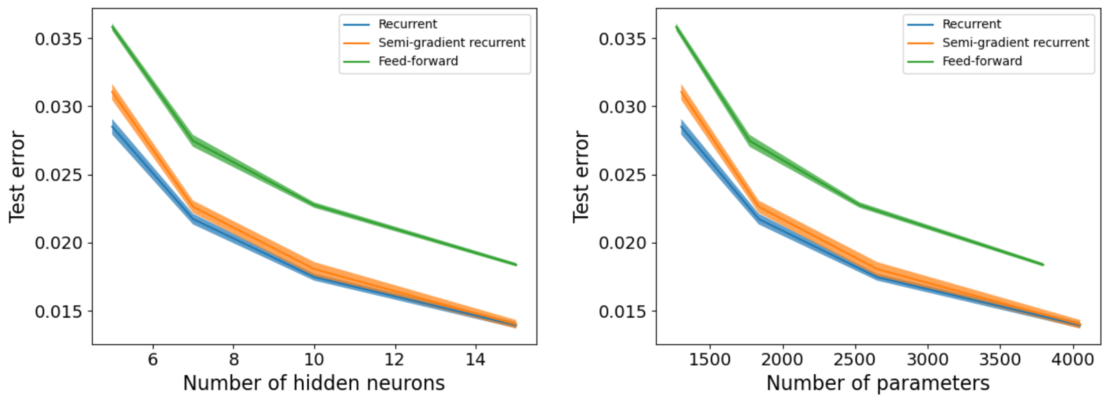


(b) Comparison of the test error

Figure 6.: Comparison of the performance of a two-layer feed-forward network, a semi-gradient two-layer implicit recurrent network and a semi-gradient two-layer implicit recurrent network on the regression problem of a damped pendulum. Results are averaged over 12 runs. The shaded region indicates the standard deviation of the mean.

$$\ddot{x} + 2 \cdot \delta \cdot \dot{x} + \omega_0^2 \cdot x = 0,$$

where $\omega_0$ and $\delta$ are the only parameters that govern this system.
In this problem, we generate for a given pair of $\omega_0$ and $\delta$ the trajectory of this system, which is then put into the network. The network is then supposed to reconstruct $\omega_0$ and $\delta$. In our simulations, we discretize the trajectory with a time step of $\Delta t = 0.1$ and generated the dynamics with random starting positions $x_0$ and starting velocities $v_0$, which were drawn from a normal distribution $N(0,2)$. Additionally, $\omega_0$ is drawn form $[1,2]$ and $\delta$ is drawn form $[0,2]$. We have then separated the dataset with $4 \cdot 10^6$ samples into a training- and test dataset with the ratio of $4:1$. It is made sure, that every combination of $x_0, v_0, \omega_0$ and $\delta$ occurs only once in the whole dataset. In our simulations we perform for every epoch 200 gradient steps over six batches. Every four epochs we calculate the mean squared error on the whole test dataset. All simulations are repeated 12 times over 400 epochs.
The results of a simulation on this problem with a networks with 5 hidden neurons are shown in figure 6. In figure 6 (a), we can see the mean train error performance of each network and the shaded region indicates with the standard deviation of the mean. Here, the two-layer implicit recurrent network has the best performance, followed by the semi-gradient two-layer recurrent network. The feed-forward network has the worst performance and interestingly during training it has much higher variance than its recurrent counterparts. This may indicate that the prediction of implicit recurrent networks is more consistent.
In figure 6 (b), the test error of each network is plotted over the epochs, where again the two-layer implicit recurrent network shows the best performance.
Additionally, we study the dependence of the final test error on the number of hidden neurons and on the number of parameters of each network architecture. Here, we train each network for a given number of hidden neurons over 400 epochs and plot the final test error in figure 7 (a). In figure 7 we take the results from 7 (a), but plot the final test error over the number of parameters. Here, we see that two-layer implicit recurrent networks perform much better than their two-layer feed-forward counterparts. If we compare two-layer implicit recurrent networks to their semi-gradient counterpart, we see that for a small number of hidden neurons the semi-gradient method performs significantly worse. But, as the number of hidden neurons increases the disadvantage of semi-gradient implicit recurrent networks becomes smaller. This may suggest,



(a) Final test error over number of hidden neurons (b) Final test error over number of parameters

Figure 7.: Comparison of the final test error of a two-layer feed-forward network, a semi-gradient two-layer implicit recurrent network and a semi-gradient two-layer implicit recurrent network on the regression problem of a damped pendulum. In (a) the final test error is plotted over the number of hidden neurons of the trained networks and in (b) the final test error is plotted over the number of parameters. Results are averaged over 12 runs. The shaded region indicates the standard deviation of the mean.

that if many neurons are used it is sufficient to use semi-gradient method instead of the using the exact derivatives. Though, we think this has to be studied more rigorously.

## 3. Methods

### 3.1. One-layer implicit recurrent networks

A one-layer implicit recurrent network receives an input $\vec{X}$ which is connected via a weight matrix $Q$ into a layer $\vec{Y}$, which is recurrently connected to itself via recurrent weights $W$.
The activation function of this layer is then given by

$$\vec{Y} = f(W \cdot \vec{Y} + Q \cdot \vec{X} + \vec{T}), \tag{3}$$

where $\vec{Y}$ is solved with the method described in the method section 3.4.
The derivatives with respect to the weights of $W$, which have to be used in order to perform gradient descend are given by

$$J_W = (\mathbb{1} - f' \odot W)^{-1} f' \odot \delta_{\vec{Y}}, \tag{4}$$

where $(J_W)_{ijm} := \frac{\partial Y_i}{\partial W_{jm}}$, $(\delta_{\vec{Y}})_{ijm} := \delta_{ij} \cdot Y_m$ and $(f')_{ij} := \frac{\partial f(\widetilde{X}_i)}{\partial \widetilde{X}_i}$. Here, $\odot$ is the element wise multiplication sign.
The derivatives with respect to the recurrent weights of $Q$ are given by

$$J_Q = (\mathbb{1} - f' \odot W)^{-1} f' \odot \delta_{\vec{X}}, \tag{5}$$

where $(J_Q)_{ijm} = \frac{\partial Y_i}{\partial Q_{jm}}$ and $(\delta_{\vec{X}})_{ijm} = \delta_{ij} \cdot X_m$.
And finally, the derivatives with respect to the biases $T$ are given by

$$J_T = (\mathbb{1} - f' \odot W)^{-1} f' \odot \mathbb{1}, \tag{6}$$

where $(J_T)_{ij} = \frac{\partial Y_i}{\partial T_j}$ and $(\mathbb{1})_{ij} = \delta_{ij}$.

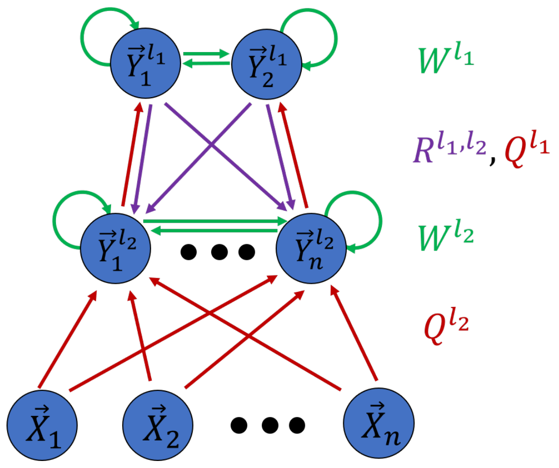


Figure 8.: Depiction of a two-layer implicit recurrent network with lateral connections within layers and feed-back connections from the output into the hidden layer.

A derivation of these derivatives is provided in appendix A. We show additionally in appendix A.2, that iterative recurrent networks which are updated an infinite number of times lead, under certain conditions, to the same derivative.
In our simulations we have initialized the weights of $\vec{T}$ and $Q$ according to a uniform distribution in range of $[-0.5, 0.5]$. The weights of $W$ were initialized to be all zero.
A implicit recurrent network is then trained according to the following pseudocode:

**Algorithm 1.** Training of implicit recurrent networks

```
1: procedure TRAINING STEP IN EPOCH
2:     $D_{\vec{X}}$ ← load batch of input data
3:     $D_{\vec{Y}_{gt}}$ ← load batch of ground truth data
4:     for every input data $\vec{X}$ in $D_{\vec{X}}$ do
5:         Calculate $\vec{Y}$ so that equation 3 is full filled with accuracy $\epsilon$
6:         Calculate derivatives of the implicit recurrent network
7:         Compute and store gradient step for this input output pair
8:     Perform a gradient step of the whole batch according to the ADAM algorithm 3.5
```

## 3.2. Two-layer fully recurrent implicit networks

In this work, we also implement two-layer implicit recurrent layers, where additional feed-back connections from higher to lower layers are used. An example of such a two-layer implicit recurrent network is depicted in figure 8.
The activation of the output layer $l_1$ is then given by

$$\vec{Y}^{l_1} = f(Q^{l_1} \cdot \vec{Y}^{l_2} + W^{l_1} \cdot \vec{Y}^{l_1} + \vec{T}^{l_1}),$$

where $l_n$ denotes, to which layer the weights or the neural activation belongs to.
The activation of the second layer is then given by

$$\vec{Y}^{l_2} = f(Q^{l_2} \cdot \vec{X} + W^{l_2} \cdot \vec{Y}^{l_2} + R^{l_1 l_2} \cdot \vec{Y}^{l_1} + \vec{T}^{l_2}).$$

A derivation of the derivatives of this network is given in the appendix B, where we show how to derive the derivative for the example of the derivative with respect to $W^{l_1}$. In our experiments all recurrent weights $W^{l_1}$, $W^{l_2}$ and $R^{l_1 l_2}$ are initially set to zero. All other weights and biases are initialized uniformly within $[-0.5, 0.5[$.

## 3.3. Semi-gradient implicit recurrent networks

In semi-gradient implicit recurrent networks, we ignore the reappearing dependence of $\vec{Y}$ with respect to the parameters. This approximation is made, so that implicit recurrent networks can be implemented more easily into auto-differentiation modules such as pytorch [8] or tensorflow [9]. The procedure of training the semi-gradient implicit recurrent network remains in principle the same as described in the pseudo-code 1, with the exception that neural activation which appear inside activation functions are treated as a constant.

## 3.4. Calculation of the equilibrium

In order to use implicit activation functions, it is necessary that the equilibrium $\vec{Y}$ is calculated with a sufficiently low error $\epsilon$. In our observation the iterative method of calculating the equilibrium $\vec{Y}$, does in principle work for small networks, but when many hidden neurons are used or the input dimension becomes too large, this method does not work at all. Therefore, we compute the neural activation $\vec{Y}$ as proposed by P. Földiák [6] by solving the differential equation

$$\frac{d\vec{Y}}{dt} = f(Q \cdot \vec{X} + W \cdot \vec{Y} + \vec{T}) - \vec{Y}$$

numerically at equilibrium, where $\frac{d\vec{Y}}{dt} = 0$.
If the Euler method is used in order to solve differential equation numerically, we obtain the update rule

$$\vec{Y}(t+1) = \vec{Y}(t) + h \cdot (f(\widetilde{X}(t)) - \vec{Y}(t)).$$

If a step size of $h = 1$ is used, this method is actually the same as the iterative method where $\vec{Y}(t+1) = f(\widetilde{X}(t))$. But, of course a step size of $h = 1$ is a very rough numerical approximation to the differential equation. This may explain, why the iterative method fails for big networks. Therefore, we use from scipy.integrate.solve_ivp [10] the RK-4 method for 30 iterations in order to solve the equilibrium. Here, we always start at $\vec{Y}(t=0) = 0$ and usually obtain an accuracy of $\|\frac{d\vec{Y}}{dt}\| \approx 10^{-8}$.
In the case of two-layer implicit recurrent networks (see section 3.2) the activation of both layers is coupled to one another.
Therefore, in order to obtain the equilibrium, we have to solve the differential equations

$$\frac{d\vec{Y}^{l_2}}{dt} = f(Q^{l_2} \cdot \vec{X} + W^{l_2} \cdot \vec{Y}^{l_2} + R^{l_1 l_2} \cdot \vec{Y}^{l_1} + \vec{T}^{l_2}) - \vec{Y}^{l_2} \tag{7}$$

and

$$\frac{d\vec{Y}^{l_1}}{dt} = f(Q^{l_1} \cdot \vec{Y}^{l_2} + W^{l_1} \cdot \vec{Y}^{l_1} + \vec{T}^{l_1}) - \vec{Y}^{l_1}. \tag{8}$$

simultaneously.

## 3.5. Learning rate optimizer

We have used ADAM [11] with a learning rate of 0.01 as an optimizer for our networks. Here, we have applied the ADAM algorithm for every weight matrix $Q, W$ and $T$ separately.

# 4. Discussion

In this work, we show that a one-layer implicit recurrent neural network is able to solve the XOR-Problem. Therefore, this indicates that implicit recurrent neural networks are computationally more capable than state-of-the-art feed forward networks. Furthermore, due to these results, we speculate that this indicates that the universal approximation theorem [12] may hold for one-layer

implicit recurrent neural networks. The universal approximation theorem states, that a two-layer feed-forward network can approximate any real valued function, if it has enough hidden neurons. Therefore, a two-layer feed-forward network can in principle solve any problem. But in practise, it often works better to increase the number of layers in order to increase the computational power of the artificial neural network. However, if too many layers are used within a network the problem of the vanishing/exploding gradient occurs, so that training of the weights within the first layers of the network becomes difficult.
Our work indicates that with the use of implicit recurrent neural networks, it is also possible to increase the computational power of neural networks. Thus, we think that implicit recurrent networks may be a new alternative to adding more neurons and more layers into artificial neural networks. Therefore, we suggest to generalize our method to multi-layer recurrent neural networks. Thereafter, one could compare the network performance of feed-forward networks to implicit recurrent networks over the network depth. We speculate, that implicit recurrent networks will reach accuracy's that can not be reached by a feed-forward network architecture.
Interestingly, in our simulations implicit recurrent networks show lower variance in the training error. This suggests, that the network‘s predictions are more consistent than in feed-architectures. Therefore, we suppose that it may be interesting to apply these networks to deep reinforcement learning algorithms, such as [13, 14], which struggle with high variance during training. Additionally, recurrent connections may make it possible to have a correction mechanism from higher to lower layers. Thus, we think it may be interesting to test weather actor-critic architectures with a shared layer architecture may benefit from such recurrent mechanisms. Finally, we suggest to generalize implicit recurrent networks to convolutional networks [15], where recurrent connections between different filters could be added. Additionally, one could try to connect neighbouring kernels with one another.

## References

[1] Kohitij Kar, Jonas Kubilius, Kailyn Schmidt, Elias B Issa, and James J DiCarlo. Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior. *Nature neuroscience*, 22(6):974–983, 2019.

[2] Tim C Kietzmann, Courtney J Spoerer, Lynn KA Sörensen, Radoslaw M Cichy, Olaf Hauk, and Nikolaus Kriegeskorte. Recurrence is required to capture the representational dynamics of the human visual system. *Proceedings of the National Academy of Sciences*, 116(43):21854–21863, 2019.

[3] Jeffrey S Johnson and Bruno A Olshausen. The recognition of partially visible natural objects in the presence and absence of their occluders. *Vision research*, 45(25-26):3262–3276, 2005.

[4] Randall C O'Reilly, Dean Wyatte, Seth Herd, Brian Mingus, and David J Jilk. Recurrent processing during object recognition. *Frontiers in psychology*, 4:124, 2013.

[5] Markus Roland Ernst, Jochen Triesch, and Thomas Burwick. Recurrent connections aid occluded object recognition by discounting occluders. In *International Conference on Artificial Neural Networks*, pages 294–305. Springer, 2019.

[6] Peter Földiak. Forming sparse representations by local anti-hebbian learning. *Biological cybernetics*, 64(2):165–170, 1990.

[7] Felipe Gerhard, Cristina Savin, and Jochen Triesch. A robust biologically plausible implementation of ica-like learning. In *ESANN*, 2009.

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[9] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[10] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

[11] Kingma Da. A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] George Cybenko. Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:183–192, 1989.

[13] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[15] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

## A. Derivation of the derivatives of a one-layer implicit recurrent neuron

In this section, we are going to derive the derivatives of a one-layer implicit recurrent neuron, which are necessary for the backpropagation update rule. First, we are going to derive the derivatives directly from the implicit activation function. Secondly, we will provide an alternative derivation, where we derive the derivative from an infinite chains of iterative network updates.

### A.1. Derivation from the implicit activation function

In this section we consider a single neuron layer $\vec{Y}$, which receives a stationary input vector $\vec{X}$ over the feed-forward weight matrix $Q$ and has a bias vector $\vec{T}$. The neuron layer is additionally recurrently connected to itself over the recurrent weight matrix $W$. The overall input $\widetilde{X} = W \cdot \vec{Y} + Q \cdot \vec{X} + \vec{T}$ is then put into a sigmoid activation function $f(\cdot)$.
Therefore, the activation of this implicit recurrent network is given by

$$\vec{Y} = f(W \cdot \vec{Y} + Q \cdot \vec{X} + \vec{T}). \tag{9}$$

If we now want to calculate the derivative of a specific neuron $Y_i$ in this layer with respect to a specific weight $W_{jm}$, we obtain

$$\begin{aligned} \frac{\partial Y_i}{\partial W_{jm}} &= \frac{\partial f(\widetilde{X}_i)}{\partial \widetilde{X}_i} \cdot \frac{\partial \widetilde{X}_i}{\partial W_{jm}} = \frac{\partial f(\widetilde{X}_i)}{\partial \widetilde{X}_i} \cdot \left( \sum_k \left( W_{ik} \cdot \frac{\partial Y_k}{\partial W_{jm}} + \delta_{ij} \cdot \delta_{km} \cdot Y_k \right) \right) \\ &= \frac{\partial f(\widetilde{X}_i)}{\partial \widetilde{X}_i} \cdot \left( \sum_k \left( W_{ik} \cdot \frac{\partial Y_k}{\partial W_{jm}} \right) + \delta_{ij} \cdot Y_m \right). \end{aligned} \tag{10}$$

In this equation the derivative of $\frac{\partial Y_i}{\partial W_{jm}}$ reappears on the right side of the equation. But, since in the sum over $k$ the derivatives of the other neurons $Y_k$ with respect to the weight $W_{jm}$ occurs as

well, we obtain a set of $y \times y \times y$ linear equations. This set of linear equations has to be solved in order to obtain the derivatives.
With the use of matrix formulation, equation 10 can be rewritten to

$$J_W = f' \odot \left(W \cdot J_W + \delta_{\vec{Y}}\right),$$

where $(J_W)_{ijm} := \frac{\partial Y_i}{\partial W_{jm}}$, $(\delta_{\vec{Y}})_{ijm} := \delta_{ij} \cdot Y_m$ and $(f')_i := \frac{\partial f(\widetilde{X}_i)}{\partial \widetilde{X}_i}$. Here, $\odot$ is simply the element wise multiplication sign between two matrices.
In order to obtain the resulting set of derivatives, one now has to solve this equation for $J_W$, where one obtains

$$J_W = (\mathbb{1} - f' \odot W)^{-1} f' \odot \delta_{\vec{Y}}.$$

Similarly, for the other derivatives with respect to $Q_{jm}$ one obtains

$$J_Q = (\mathbb{1} - f' \odot W)^{-1} f' \odot \delta_{\vec{X}},$$

where $(J_Q)_{ijm} = \frac{\partial Y_i}{\partial Q_{jm}}$ and $(\delta_{\vec{X}})_{ijm} = \delta_{ij} \cdot X_m$.
For the derivatives with respect to $T_j$ one obtains

$$J_T = (\mathbb{1} - f' \odot W)^{-1} f' \odot \mathbb{1},$$

where $(J_T)_{ij} = \frac{\partial Y_i}{\partial T_j}$ and $(\mathbb{1})_{ij} = \delta_{ij}$.
Therefore, we have derived all derivatives, which are necessary for applying the backpropagation algorithm in a one-layer implicit recurrent neural network.

## A.2. Derivation as an infinite chain of iterative recurrent network updates

Alternatively, these derivatives can also be derived by considering an iterative recurrent neural network, which uses infinite iterative update steps. In this derivation, we additionally assume that the iterative update reaches an equilibrium after a finite iterative updates. Furthermore, we assume that $(f' \odot W)^k \approx 0$ for large $k$, where $f'$ is the derivative of the activation function at the equilibrium. This assumption is analogous to the assumption, that the gradient vanishes after many iterative updates.
In order to show that let us consider the iterative update rule for an iterative one-layer recurrent neural network.
The activation function is then given by:

$$\vec{Y}(t) = f(W \cdot \vec{Y}(t-1) + Q \cdot \vec{X} + \vec{T}) \tag{11}$$

In the firs time step we set $\vec{Y}(t=0) = 0$ and this activation function is then updated iteratively up to $\vec{Y}(t=n)$.
If we calculate for example the derivative with respect to $W_{jm}$, we obtain

$$\frac{\partial Y_i(t)}{\partial W_{jm}} = \frac{\partial f(\widetilde{X}_i)}{\partial \widetilde{X}_i} \cdot \left( \sum_k \left( W_{ik} \cdot \frac{\partial Y_k(t-1)}{\partial W_{jm}} \right) + \delta_{ij} \cdot Y_m(t-1) \right), \tag{12}$$

which can again be rewritten into matrix formulation by

$$J_W(t) = f'_{\widetilde{X}(t-1)} \odot \left( W \cdot J_W(t-1) + \delta_{\vec{Y}(t-1)} \right),$$

where $f'_{\widetilde{X}(t-1)}$ is the derivative of the activation function for the overall input at the $(t-1)$ th iterative time step. This calculation has to be applied repeatedly up to the first time step, therefore we obtain for $n >= 2$

$$J_W(t) = \sum_{k=0}^{n} \left[ \left( \prod_{m=1}^{k} f'_{\widetilde{X}(t-m)} \odot W \right) \cdot f'_{\widetilde{X}(t-k-1)} \odot \delta_{\vec{Y}(t-k-1)} \right). \tag{13}$$

Because we apply an infinite amount of iterative time steps $n$ goes to infinity. Furthermore, we assume that this system reaches an equilibrium, where $\vec{Y}(t) = \vec{Y}(t-1)$, after a finite iterative update steps.
Therefore, we can approximate equation 13 to

$$J_W(t) \approx \sum_{k=0}^{\infty} \left( \prod_{m=1}^{k} f' \odot W \right) \cdot f' \odot \delta_{\vec{Y}(t)} = \left( \sum_{k=0}^{\infty} (f' \odot W)^k \right) \cdot f' \odot \delta_{\vec{Y}(t)},$$

where we have used that the last infinite iterative updates are calculated at the equilibrium. Thus, we can approximate $\delta_{\vec{Y}(t-k-1)}$ for these time steps to the equilibrium $\delta_{\vec{Y}(t)}$. Additionally, we can therefore approximate $f'_{\widetilde{X}(t-m)}$ to $f'$ which is the derivative at equilibrium.
Next, in order to further simplify this equation, we can use the Neumann series, which is a generalisation of the geometric series to matrices. Therefore, we additionally assume that $\sum_{k=0}^{\infty}(f' \odot W)^k$ is convergent, from which follows that $\sum_{k=0}^{\infty}(f' \odot W)^k = (\mathbb{1} - f' \odot W)^{-1}$.
We can now use this fact to obtain the final iterative update rule after a infinite amount of update rules, which is then given by

$$J_W = (\mathbb{1} - f' \odot W)^{-1} f' \odot \delta_{\vec{Y}}.$$

This final derivative with respect to $W$ is the same as the derivative for the implicit recurrent network and for the other derivatives with respect to $Q$ and $T$ this can be shown analogously. Therefore the relationship between the derivative of an implicit recurrent network and the derivative of an iterative recurrent network, which is updated an infinite number of times at the equilibrium, has been shown.

## B. Derivation of derivatives of two-layer implicit recurrent neural networks

In this section the derivatives of two-layer implicit recurrent neural networks will be derived. Here, the output layer $\vec{Y}^{l_1}$ is defined as

$$\vec{Y}^{l_1} = f(Q^{l_1} \cdot \vec{Y}^{l_2} + W^{l_1} \cdot \vec{Y}^{l_1} + \vec{T}^{l_1}),$$

where $l_n$ denotes, to which layer the weights or the output belongs to.
The activation of the second layer is then given by

$$\vec{Y}^{l_2} = f(Q^{l_2} \cdot \vec{X} + W^{l_2} \cdot \vec{Y}^{l_2} + R^{l_1 l_2} \cdot \vec{Y}^{l_1} + \vec{T}^{l_2}),$$

where $R^{l_1 l_2}$ are feedback connections from layer $l_1$ to layer $l_2$.
If we calculate the derivative of both layers with respect to $W^{l_1}_{ij}$ and write them down into matrix notation, we obtain for the first layer $\vec{Y}^{l_1}$

$$J^{l_1}_{W^{l_1}} = f'_{l_1} \odot (Q^{l_1} \cdot J^{l_2}_{W^{l_1}} + \delta_{\vec{Y}^{l_1}} + W^{l_1} \cdot J^{l_1}_{W^{l_1}}), \tag{14}$$

where $(J^{l_1}_{W^{l_1}})_{ijm} = \frac{\partial Y^{l_1}_i}{\partial W^{l_1}_{jm}}$, $(J^{l_2}_{W^{l_1}})_{ijm} = \frac{\partial Y^{l_2}_i}{\partial W^{l_1}_{jm}}$, $(\delta_{\vec{Y}^{l_1}})_{ijm} := \delta_{ij} \cdot Y^{l_1}_m$ and $f'_{l_1}$ is simply the derivative of the sigmoid activation functions in layer $l_1$.
The derivatives for layer $\vec{Y}^{l_2}$ are analogously given by

$$J^{l_2}_{W^{l_1}} = f'_{l_2} \odot (R^{l_1 l_2} \cdot J^{l_1}_{W^{l_1}} + W^{l_2} \cdot J^{l_2}_{W^{l_1}}). \tag{15}$$

Therefore, we obtain with equation 14 and 15 two coupled sets of linear equations, which have to be solved in order to obtain $J^{l_1}_{W^{l_1}}$ which can then be used for backpropagation. The first step would be to solve equation 15 for $J^{l_2}_{W^{l_1}}$. Here, we obtain

$$J^{l_2}_{W^{l_1}} = (\mathbb{1} - f'_{l_2} \odot W^{l_2})^{-1} \cdot f'_{l_2} \odot (R^{l_1 l_2} \cdot J^{l_1}_{W^{l_1}}).$$

Then this solution can be used by inserting the solution into equation 14, where we again have to solve for $J^{l_1}_{W^{l_1}}$.
The final solution is then given by

$$J^{l_1}_{W^{l_1}} = (\mathbb{1} - f'_{l_1} \odot W^{l_1} - f'_{l_1} \odot Q^{l_1} \cdot (\mathbb{1} - f'_{l_2} \odot W^{l_2})^{-1} f'_{l_2} \odot R^{l_1 l_2})^{-1} \cdot f'_{l_1} \odot \delta_{\vec{Y}^l_1}.$$

Note that, if we set $R^{l_1 l_2} = 0$ the derivatives simplify to

$$J^{l_1}_{W^{l_1}} = (\mathbb{1} - f'_{l_1} \odot W^{l_1})^{-1} \cdot f'_{l_1} \odot \delta_{\vec{Y}^l_1},$$

which has the same form as the derivatives of a one-layer implicit recurrent network.
The other derivatives with respect to $T_{l_1}$, $Q_{l_1}$, $T_{l_2}$, $R_{l_1 l_2}$, $W_{l_2}$ and $Q_{l_2}$ can be derived analogously. The derivatives are then given by

$$J^{l_1}_{Q^{l_1}} = S \cdot f'_{l_1} \odot \delta_{\vec{Y}^l_2},$$

$$J^{l_1}_{T^{l_1}} = S \cdot f'_{l_1} \odot \mathbb{1},$$

$$J^{l_1}_{Q^{l_2}} = S \cdot f'_{l_1} \odot Q^{l_1} (\mathbb{1} - f'_{l_2} \odot W^{l_2})^{-1} \cdot f'_{l_2} \odot \delta_{\vec{X}}.$$

$$J^{l_1}_{R^{l_1 l_2}} = S \cdot f'_{l_1} \odot Q^{l_1} (\mathbb{1} - f'_{l_2} \odot W^{l_2})^{-1} \cdot f'_{l_2} \odot \delta_{\vec{Y}^{l_1}},$$

$$J^{l_1}_{W^{l_2}} = S \cdot f'_{l_1} \odot Q^{l_1} (\mathbb{1} - f'_{l_2} \odot W^{l_2})^{-1} \cdot f'_{l_2} \odot \delta_{\vec{Y}^{l_2}}$$

and

$$J^{l_1}_{T^{l_2}} = S \cdot f'_{l_1} \odot Q^{l_1} (\mathbb{1} - f'_{l_2} \odot W^{l_2})^{-1} \cdot f'_{l_2} \odot \mathbb{1},$$

where $S := (\mathbb{1} - f'_{l_1} \odot W^{l_1} - f'_{l_1} \odot Q^{l_1} \cdot (\mathbb{1} - f'_{l_2} \odot W^{l_2})^{-1} f'_{l_2} \odot R^{l_1 l_2})^{-1}$.