entropy

MDPI

Article

Learning in Feedforward Neural Networks Accelerated by Transfer Entropy

Adrian Moldovan [1,2], Angel Cataron [1,2,*] and Răzvan Andonie [3]

[1] Department of Electronics and Computers, Transilvania University, 500024 Braşov, Romania; adrian.moldovan@gmail.com
[2] Corporate Technology, Siemens SRL, 500007 Braşov, Romania
[3] Department of Computer Science, Central Washington University, Ellensburg, WA 98926, USA; razvan.andonie@cwu.edu
* Correspondence: cataron@unitbv.ro; Tel.: +40-268-413000

Received: 1 December 2019; Accepted: 11 January 2020; Published: 16 January 2020

Abstract: Current neural networks architectures are many times harder to train because of the increasing size and complexity of the used datasets. Our objective is to design more efficient training algorithms utilizing causal relationships inferred from neural networks. The transfer entropy (TE) was initially introduced as an information transfer measure used to quantify the statistical coherence between events (time series). Later, it was related to causality, even if they are not the same. There are only few papers reporting applications of causality or TE in neural networks. Our contribution is an information-theoretical method for analyzing information transfer between the nodes of feedforward neural networks. The information transfer is measured by the TE of feedback neural connections. Intuitively, TE measures the relevance of a connection in the network and the feedback amplifies this connection. We introduce a backpropagation type training algorithm that uses TE feedback connections to improve its performance.

Keywords: transfer entropy; causality; neural network; backpropagation; gradient descent; deep learning

1. Introduction and Related Work

We generally differentiate between statistical correlation and causality. Often, when correlation is observed, causality is wrongly inferred and we are tempted to identify causality through correlation. This is because of the inability to detect a time lag between a cause and effect, which is a prerequisite for causality [1].

Following Shadish et al. [2], the three key criteria for inferring a cause and effect relationship are (1) the cause preceded the effect, (2) the cause was related to the effect, and (3) we can find no plausible alternative explanation for the effect other than the cause.

According to [3], there is an important distinction between the "intervention-based causality" and "statistical causality". The first concept, introduced by Pearl [4], combines statistical and non-statistical data and allows one to answer questions, like "if we give a drug to a patient, i.e., intervene, will their chances of survival increase?" Statistical causality does not answer such questions, because it does not operate on the concept of intervention and only involves tools of data analysis. The causality in a statistical sense is a type of dependence, where we infer direction as a result of the knowledge of temporal structure and the notion that the cause has to precede the effect. We will focus here only on statistical causality measured by the information transfer approach.

A relatively recent information transfer measure is the transfer entropy (TE). The TE was introduced by Schreiber [5] not as a causality indicator, but as an information transfer measure

used to quantify the statistical coherence between events (time series). For a comprehensive discussion of the TE vs. causality paradigms we refer to the work in [6]. In our previous work, we introduced the Transfer Information Energy (TIE) [7,8] as an alternative to the TE. Whereas the TE can be used as a measure of the reduction in uncertainty about one event given another, the TIE measures the increase in certainty about one event given another.

Causality and information transfer are not exactly the same. Causality is typically related to whether interventions on a source have an effect on the target. Information transfer measures how observations of the source can predict transitions of the target. Causal information flow describes the causal structure of a system, whereas information transfer can then be used to describe the emergent computation on that causal structure [6].

The directivity of information flow through a channel was defined by Massey [9] in the form of directed information. The author shows that in the presence of feedback, this is a more useful quantity than the traditional mutual information. From a similar perspective, the TE measures the information flow from one process to another by quantifying the deviation from the generalized Markov property as a Kullback–Leibler distance, thus both TE and the directed information can be used to estimate the directional informational interaction between two random variables.

TE is a directional, dynamic measure of predictive information, rather than a measure of the causal information flow from a source and to a destination. To be interpreted as information transfer, the TE should only be applied to causal information sources for the given destination [6]. We will use the information transfer measured by the TE to establish the presence of and quantify causal relationships between the nodes (neurons) of neural networks.

In the current deep learning era, neural architectures are many times hard to train because of the increasing size and complexity of the used datasets. Our main question is how causal relationships can be inferred from neural networks. Using such relationships, can we define better training algorithms? There are very few results reporting applications of causality or transfer information in neural networks. We will refer to them in the following.

TE has been used for the quantification of effective connectivity between neurons [10–13]. To the extent of our knowledge, the work in [14,15] represent the only attempts to use TE for improving the learning capability of neural networks.

The reservoir adaptation method in [14] optimizes the TE at each individual unit, dependent on properties of the information transfer between input and output of the system. It improves the performance of online echo state learning and recursive least squares online learning.

Causal relationships within a neural network are defined by Féraud et al. in [16]. To explain the classification obtained by a multilayer perceptron, Féraud et al. introduced the concept of "causal importance" and defined a saliency measurement allowing the selection of relevant variables. Combining the saliency and the causal importance allowed them an interpretation of the trained neural network.

Herzog et al. [15] used feedforward TE between neurons to structure neural feedback connectivity. These feedback connections are then used to improve the training algorithm in a convolutional neural network (CNN) [17]. In deep learning, a CNN is a class of deep neural networks, most commonly applied to image analysis. Intuitively, a CNN is a multilayered neural network that uses convolution in place of general matrix multiplication in at least one of its layers. Herzog et al. averaged (by layer and class of the training sample) the calculation of TE gathered from directly or indirectly connected neurons, using thresholded activation values to obtain the required time series. The averaged TEs are indirectly implied in the subsequent neuron's activations as part of the training with feedback. They are potentiated with a layer distance amplifier and the new value is summed to the input of the activation function. As a result, only one TE derived value is used for each of the layers. Herzog et al. made two interesting observations about why using TE for defining TE feedback in CNN networks:

- There is a decreasing feedforward convergence towards higher layers.

- The TE is in general lower between nodes with larger layer distances than between neighbors. This is caused by the fact that long-range TE is calculated by conditioning on the intermediate layers. Thus, there is a higher probability to form long-range as compared to short-range feedback connections.

Our contribution is a novel information-theoretical approach for analyzing information transfer (measured by TE) between the nodes of neural networks. We use the information transfer to establish the presence of relationships and the quantification of these between neurons. Intuitively, TE measures the relevance of a connection in the network and the feedback amplifies this connection. We introduce a backpropagation-type training algorithm which uses TE feedback connections to improve its performance.

The paper has the following structure. In Section 2, we introduce the formal definition of the TE and enumerate some of its applications. Section 3 describes how the feedback TE can be numerically approximated during the training of a feedforward neural network. In Section 4, we present our approach for integrating the TE as a feedback in the training algorithm of a neural classifier. The closest related work is Herzog's et al. paper [15], and we will explain the differences between the two approaches. Section 5 presents several experiments performed on a toy example and on standard benchmarks. Section 6 analyzes the results of the numerical experiments. Section 7 contains final remarks. The Appendix A presents further details of our experiments.

2. Background: Transfer Information Entropy

We start by introducing the formal definition of TE. A detailed presentation can be found in Bossomaier et al. [18]. The connection between TE and causality in time series analysis is discussed in [19].

TE measures the directionality of a variable with respect to time based on the probability density function. For two discrete stationary processes I and J, TE relates k previous samples of process I and l previous samples of process J and is defined as follows [5,20],

$$TE_{J\to I} = \sum_{t=1}^{n-1} p(i_{t+1}, i_t^{(k)}, j_t^{(l)})\, log \frac{p(i_{t+1}|i_t^{(k)}, j_t^{(l)})}{p(i_{t+1}|i_t^{(k)})}, \tag{1}$$

where i_t and j_t are the discrete states at time t of I and J, respectively, and $i_t^{(k)}$ and $j_t^{(l)}$ are the k and l dimensional delay vectors of time series I and J, respectively. The three symbols $i_{t+1}, i_t^{(k)}, j_t^{(l)}$ for computing probabilities are sequences of time series symbols.

$TE_{J\to I}$ measures the extend to which time series J influences time series I. The TE is asymmetric under the exchange of i_t and j_t, and provides information regarding the direction of interaction between the two time series. With respect to mutual information, the TE can be interpreted as being equivalent to the conditional mutual information [19].

The accurate estimation of entropy-based measures is generally difficult. There is no consensus on an optimal way for estimating TE from a dataset [21]. Schreiber proposed the TE using correlation integrals [5]. The most common TE estimation approach is histogram estimation with fixed partitioning. This simple method is not scalable for more than three scalars. Moreover, it is sensitive to the size of bins used. Other nonparametric entropy estimation methods have been also used for computing the TE [21–23]: kernel density estimation methods, nearest-neighbor, Parzen, neural networks, etc.

The best known applications of TE are in financial time series analysis. TE was used to compute the information flow between stock markets or to determine relationships between stocks, indexes, or markets [24,25]. Other application areas are neuroscience, bioinformatics, artificial life, and climate science [18].

3. How to Compute the Feedback Transfer Entropy in a Neural Network

An interesting question is how to compute the TE in a neural network, as the TE was originally defined for time series. We describe in the following how the TE can be defined and computed in a feedforward neural network as a feedback measure.

A feedforward neural network is the simplest artificial neural network architecture. The information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles in the network. The most common training method of such a network is to use the output result backwards, to adjust the weights of the connections between nodes. In our case, we will adjust the weights also considering the TE feedback measure.

We denote by *FF+FB* the network with a TE feedback, whereas a network without feedback is named *FF* (feedforward only). R is the number of epochs used in the training algorithm for both the *FF+FB* and *FF* networks, whereas r is the index of a current epoch, $r \in \{1, \ldots, R\}$.

The information transfer between two neurons can be computed if the outputs of the neurons are logged along the training process. After p consecutive training steps in *FF+FB*, we obtain two time series, of length p each, with the observations aligned in time by the index of the training step and sample position. For the *FF+FB* network trained with the backpropagation algorithm, we denote by $o_i^{r,n}$ the output of neuron i in layer $l-1$ when processing the n^{th} training sample during epoch r. Similarly, $o_j^{r,n}$ is the output of neuron j in layer l. The layer index is not required here since we compute the information transfer only between pairs of neurons from adjacent layers.

The time series are obtained by discretization of the continuous values $o_i^{r,n}$ and $o_j^{r,n}$ (we use the sigmoid activation function). The continuous sub-intervals are mapped to discrete values using binning: $s_i^{r,n} = 1$ for $o_i^{r,n} > g$ and $s_i^{r,n} = 0$ for $o_i^{r,n} \leq g$, where s stands for time *series*. We record the time series only after the first 10 training patterns were processed. This value was obtained experimentally, optimized for smaller training sets.

We estimate the TE for the two generated time series by approximating the probabilities with relative frequencies. A higher number of discrete levels gives a better approximation of the TE, but also requires longer time series. Obviously, a reduced number of discrete values is computationally more efficient. We illustrate in Figure 1 the computational pipeline that (1) collects process values from the data flow that goes through two connected neurons and (2) computes the TE between the two discrete time series produced from the output of each neuron. In the following, we denote by te the computed (approximated) TE value.

The TE is in general non-negative. However, at certain steps during training, some local TE values can be negative due to noisy inputs resulted from neuron's outputs, and also due to finite training samples available [26].

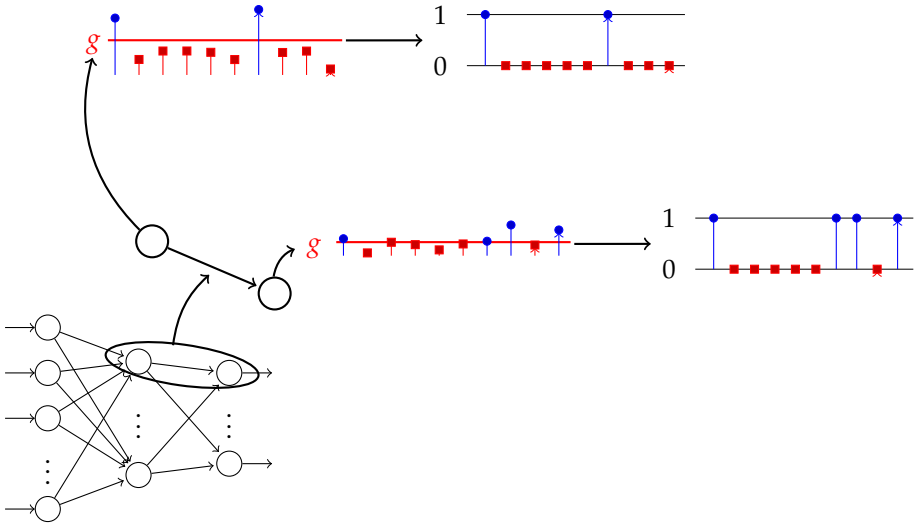

Figure 1. This illustrates how the two neurons with indices i and j from a network produce a series of activations. The g threshold is the red line that splits these activations into two groups: the ones above the threshold (blue) and the ones below the threshold (red). They correspond to the $o_i^{r,n}$ and $o_j^{r,n}$ time series, which produce the time series of binary values $s_i^{r,n}$ and $s_j^{r,n}$ used to calculate the TE. The process is applied to all pairs of connected neurons.

4. Integrate the TE in the Training of Neural Networks

In the following we explain how we integrate the feedback TE in the training algorithm of a feedforward neural network. This is the key concept of our approach. As a first step, we focus on a one hidden layer perceptron architecture trained with backpropagation gradient descent for classification tasks. Our results can be extended for any number of layers.

Backpropagation [27] is an algorithm for supervised training of artificial neural networks using gradient descent. It stands for "backward propagation of errors". Using the gradient of an error function with respect to the network's weights, we calculate this gradient backwards through the network's layers, from the output layer back to the input layer. This process presents all the training set items to the network's input, while iteratively updating the weights and calculating the *te* values that we use during the weights updates step as shown in the following sections.

The *FF+FB* training algorithm uses the discretized outputs of the neurons from adjacent layers to construct the time series needed for the TE computation. Once obtained, we use it in the backpropagation weight update process. The weights are updated incrementally (online), after processing each input pattern.

The *FF+FB* requires two training stages. Both stages are using the standard backpropagation algorithm with the modifications we describe below. In Stage I, we train the *FF+FB* network with all training samples while re-evaluating and using TE after each sample. At the end of the Stage I training, we store the *te* values for all the neuron pairs. In Stage II, we train the same network, using the *te* values computed in Stage I. In summary:

(I) Train with the TE feedback. During this stage, we apply the calculated TE values after processing each input pattern.
(II) Train with the TE feedback using the TE values calculated in Stage I, for R epochs.

The *FF+FB* network requires an additional hyperparameter g, the threshold used to bin the output of the neurons when generating the time series that is also obtained using grid search. A unique value of the threshold g is used for all neurons.

The value $te_{j,i}^{r,n}$ is calculated using neurons j and i located in layer l, respectively, $l-1$, for the n^{th} sample at epoch r. Using this notation and lag one in the definition of TE (see eq (1)), we obtain the TE measure of interneuron connections:

$$te_{j,i}^{r,n} = \sum_{s_i^{r,n+1}, s_i^{r,n}, s_j^{r,n}} p(s_i^{r,n+1}, s_i^{r,n}, s_j^{r,n}) \, log \frac{p(s_i^{r,n+1}, s_i^{r,n}, s_j^{r,n}) \, p(s_i^{r,n})}{p(s_i^{r,n+1}, s_i^{r,n}) \, p(s_i^{r,n}, s_j^{r,n})} \tag{2}$$

where $s_i^{r,n}$ and $s_j^{r,n}$ are the time series obtained from the outputs of neurons i and j, located in layers $l-1$, l, using the n^{th} sample at epoch r.

Once Stage I is completed, we hold only the most recent $te_{j,i}^{r,n}$ values for updating the weights in Stage II. For given values of r and n, we simplify the notation by removing indices r and k and introduce the layer index l. Instead of $te_{j,i}^{r,n}$, we use $te_{j,i}^{l}$, which aligns with the standard backpropagation notations. We update the weights by a modified gradient descent:

$$\Delta w_{ij}^{l} = -\eta \frac{\partial C}{\partial w_{ij}^{l}} (1 - te_{j,i}^{l}) \tag{3}$$

where C is the loss function. This is one of the changes we have made to the backpropagation algorithm, the addition of $1 - te_{j,i}^{l}$ term in the calculation of the network's weights updates. There are many possible feedback loop modifications of the backpropagation algorithm, but ours has been experimentally proven to yield positive results compared with the standard algorithm.

We include these modifications in the standard backpropagation gradient descent algorithm [27], and obtain Algorithm 1, used to train the *FF+FB*. This represents the standard backpropagation algorithm for a single hidden layer perceptron network; our additions are at lines 13, 14, 23–26.

As mentioned before, the closest related work is Herzog's et al. paper [15]. The differences are as follows.

- In contrast to our method, in [15] the computed TE values are used only in their last training step as an input in the activation function; the activation function is $\tilde{g}(x_i) = g(x_i + \sum_j f_{j \to i})$, where $f_{j \to i}$ is $(w_{min}|\beta - \alpha|)$ divided by the layer count (β and α are layer indices, $\beta > \alpha$, and w_{min} is the smallest weight value in their network determined in the first training step). The $f_{j \to i}$ feedback is only used if the averaged by class TE value is below a threshold Φ.
- In our approach, the distance between layers is not required since we consider only pairs of neurons from neighboring layers.
- In [15], only larger *te* values are used in the training stage. We use all computed *te* values. This adaptation helps us to obtain a longer series of events, which experimentally showed to improve the training process.

Algorithm 1: Backpropagation using transfer entropy

begin
- **input :** $\boldsymbol{y}$ true class labels vector, $\boldsymbol{x}$ input vector, N number of training samples, R number of epochs, g the threshold rate
- **input :** *TrainStage* takes the following values: *I*, *II*
- initialize all $\boldsymbol{W}$ weights with random samples between 0 and 1, drawn from a 0 centered, 0.1 width, normal distribution
- initialize all biases $\boldsymbol{b}$ and activations $\boldsymbol{A}$ with 0.0
- initialize $s_{i,j}^{l,r,n}$, i and j are the neuron indexes of the $l-1^{th}$, l^{th} layer respectively, and k is the sample's index in the training set, r - current epoch

foreach epoch $r = 0$ **to** R **do**
- randomize training set

foreach sample k *in* training set **do**

foreach l layer **do**

- compute input layer outputs $z^{l,k} = \boldsymbol{x} \odot \boldsymbol{W}^{(in)} + b^l$
- compute $z^{l,n} = w^l \sigma(z^{l-1}) + b^l$ for hidden and output layers, where $\sigma(x) = \frac{1}{1+e^{-x}}$

if Training Stage I **and** $z^{l,n} < g$ **then** $s_{i,j}^{l,r,n} = 0$

else $s_{i,j}^{l,r,n} = 1$

if l *is* hidden layer **then**

- compute output error vector $\boldsymbol{\delta}^{(hidden)} = \boldsymbol{\delta}^{(out)} \boldsymbol{W}^{(out)^T} \odot \frac{\partial \sigma(z^{(hidden)})}{\partial z^{(hidden)}}$;
  $// \frac{\partial \sigma(z^{(hidden)})}{\partial z^{(hidden)}} = \sigma(z^{(hidden)}) \odot (1 - \sigma(z^{(hidden)}))$ `the derivative of the activation function`
- compute derivation of the $J(\boldsymbol{W})$ function $\frac{\partial}{\partial w_{i,j}^{(hidden)}} J(\boldsymbol{W}) = \sigma(z_j^{(input)}) \delta_i^{(out)}$, vectorized as: $\Delta^{(hidden)} = \Delta^{(hidden)} + (\boldsymbol{A}^{(in)})^T \delta^{(hidden)}$

else if l *is* output layer **then**

- compute output error vector $\boldsymbol{\delta}^{(out)} = z^{l,n^{(out)}} - \mathbf{y}$
- compute derivation of the $J(\boldsymbol{W})$ function $\frac{\partial}{\partial w_{i,j}^{(out)}} J(\boldsymbol{W}) = \sigma(z_j^{(hidden)}) \delta_i^{(out)}$, vectorized as: $\Delta^{(out)} = \Delta^{(out)} + (\boldsymbol{A}^{(hidden)})^T \delta^{(out)}$

end

if Training Stage I **and** $r = R$ **and** $k = N$ **then**

- compute $te_{i,j}^l$ using $s_{i,j}^{l,r,n}$ according to (2)

if Training Stage I **or** Training Stage II **then**

- $\boldsymbol{W}^l := \boldsymbol{W}^l - \eta \Delta^l (1 - \boldsymbol{te}^l)$

end

end

end

end

5. Experimental Results

The main question is if the addition of the TE factor improves learning and creates any benefits. To answer it, we compare the *FF+FB* and *FF* algorithms using a target accuracy.

The experimental set-up is the following. We define a fixed target validation accuracy that both networks have to reach in an also fixed maximum number of epochs. The learning process ends when either of the target validation accuracy or the maximum number of epochs is reached. The hyperparameters used are the standard ones in any multilayer perceptron (MLP): learning rate η, number of epochs R, and number of neurons in each layer. To make comparisons easier, we use the same learning rate η for *FF+FB* and *FF*, determined by grid search. In addition, we use the target accuracy (which is the early training stop limit). For the *FF+FB* we also use the binning threshold hyperparameter g.

First, we train the *FF+FB* network on a toy example: the XOR (or "exclusive or") problem. The XOR problem is a classic benchmark in neural network research. It is the problem of using a neural network to predict the outputs of XOR logic gates given two (or more) binary inputs. An XOR function should return a true value if the inputs are not equal and a false value if they are equal. We use the most simple XOR problem, with two inputs.

The accuracy measured is here the training accuracy—the one obtained on the training set. Therefore, for now we focus only on the learning cycles, disregarding the overfitting/generalization aspect. We train up to saturation, or to 100% training accuracy the *FF+FB* on the XOR dataset (Table 1), observing the number of epochs required to reach this accuracy, in comparison to the *FF* network.

Table 1. The XOR dataset. A training epoch consists of 200 vectors, randomly selected from this dataset.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

The number of epochs for training the *FF+FB* network to reach the target accuracy is 7–10 times less than for *FF*. We average 10 runs, taking $g = 0.7$ and $\eta = 0.025$ for both networks. The number of epochs is capped to 300 epochs. With these constraints, η is optimized (by grid search) for the smallest number of epochs of the two networks that reached 100% training accuracy in 10 runs. The network for the XOR dataset has $2 - 2 - 1$ (input–hidden–output) nodes, as depicted in Figure 2.

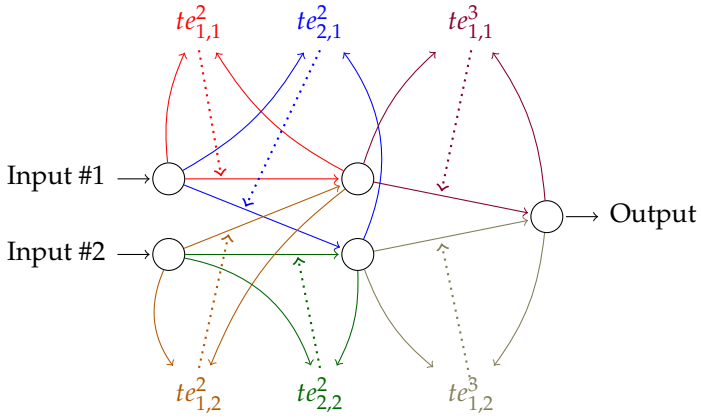

Figure 2. The *FF+FB* architecture for the XOR problem. The $te^l_{j,i}$ values are calculated between neurons j and i. Neuron i is in layer $l-1$, neuron j is in layer l. The colored arrows from the neurons show how the outputs from the neurons are used to calculate the *te* values. The same color in a dotted line arrow shows to which weight the *te* is applied (see Equation 3). The bias units are implemented but not shown here since they do not use the *te* values in the algorithm.

For the XOR problem, the *FF+FB* network is more efficient for a relatively small η (between 0.022 and 0.045) and $g = 0.7$. However, these η values are not also optimal for the *FF* network. For a larger learning constant ($\eta \approx 0.1$), *FF* generally needs less epochs (i.e., converges faster). For *FF+FB*, a small η ($\eta < 0.45$) paired with a small g ($g < 0.25$) or a large g ($g > 0.7$) threshold destabilizes *FF+FB* training. Figure 3 and Table 2 depicts the evolution of XOR learning for each run.

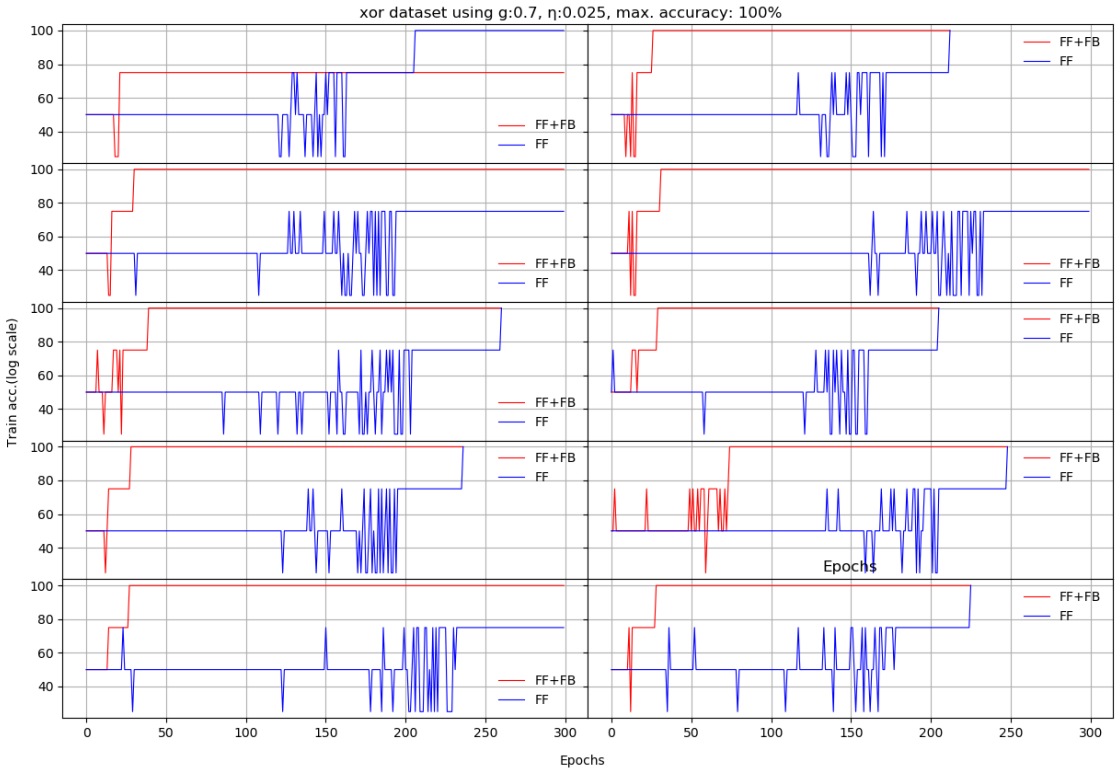

Figure 3. Ten runs on XOR dataset. Each x axis finishes when the last of the *FF+FB* or *FF* reaches either the maximum number of epochs or 100% training accuracy (log scale).

Table 2. Comparison between the number of epochs required by *FF+FB* and *FF* to reach 100% training accuracy on the XOR dataset in 10 runs. The networks did not successfully reach the target accuracy on runs 1 for *FF+FB* and runs 2,5,7 for *FF*, showing the maximum number of epochs (300) the training was limited to.

Run	*FF+FB* Epochs at 100% acc.	*FF* Network Epochs at 100% acc.
1	300	207
2	32	300
3	40	261
4	75	249
5	28	300
6	27	213
7	31	300
8	30	206
9	29	237
10	29	226
Average	**62.2**	**349.9**

Next, we train the two networks on ten standard datasets [28]: *abalone*, *car*, *chess*, *glass*, *ionosphere*, *iris*, *liver*, *redwine*, *seeds*, and *divorce*. For these set of experiments, the accuracy is measured on independent test sets, as usual. Each of the datasets had specific target accuracies and number of epochs.

The question is which of the two networks reaches the target accuracy in less epochs. We obtain on almost all datasets increased accuracies in a 10 run average, and for most of them, *FF+FB* reaches the target accuracies in less epochs than the *FF* network.

We noticed that using an unoptimized hidden layer size negatively impacts *FF* network's training. The *FF+FB* is less sensitive to this aspect (up to certain thresholds), and can successfully converge to the target accuracy; however, for some datasets, it requires more epochs.

Appendix A depicts the evolution of the learning process for all 10 datasets per each run. Our proposed solution (the *FF+FB* model) is generally more stable and reaches the accuracy target in less epochs than the *FF* network.

We also performed several control experiments to assess the significance of these results. We verified if variations of our modified backpropagation, including detrimental and misuse of the *te*, could produce different or similar results. In Section 6 we discuss some of these results. We explored the following.

- Set a fixed *te* value for all feedbacks.
- Strengthen/weaken the $te^l_{j,i}$ values by layer index.
- Replace all weights with fixed values and use *te* as feedback.
- Scale the $te^l_{j,i}$ values to $[0, 1]$.

6. Discussion

Compared to *FF*, the *FF+FB* training algorithm has an overhead needed to compute the *te* values in Stage I. In Stage II, these values are only used and there is no additional overhead.

According to our experiments, adding the TE feedback parameter (Stage II in *FF+FB*) brings two benefits: (a) it accelerates the training process—in general, less epochs are needed, and (b) we generally achieve a better test set accuracy. For the plots shown in Appendix A, Table 3 summarizes the obtained average accuracies and the differences between *FF+FB* and *FF*.

Table 3. Comparison of various between *FF+FB* and *FF* for the target validation accuracies on specified datasets (average of 10 runs). Whenever the networks did not successfully reach the targets, we used the maximum number of epochs and the last recorded accuracy to calculate the averages.

Dataset	Target Accuracy	Avg. *FF+FB* Accuracy	*FF+FB* Avg. Epochs	Avg. *FF* Accuracy	*FF* Avg. Epochs	Accuracy Difference	Max Epochs
abalone	52%	53.01	6.2	52.16	37.5	0.84%	50
car	73%	72.21	163.5	73.14	184.2	−0.92%	300
chess	96%	96.20	19.0	95.41	38.3	0.79%	40
glass	52%	52.46	154.6	35.84	294.4	16.61%	300
ionosphere	92%	92.17	16.4	92.26	22.5	−0.09%	60
iris	92%	95.11	13.8	96.22	24.8	−1.11%	100
liver	70%	68.46	212.1	61.82	294.2	6.63%	300
redwine	52%	50.18	134.6	49.89	171.8	0.29%	200
seeds	85%	87.46	41.3	87.14	136.2	0.31%	200
divorce	98%	98.03	6.9	98.62	7.4	−0.58%	20

We observed that the optimal η values for the *FF+FB* and *FF* networks are different. The *FF+FB* network usually needs a slightly smaller learning rate. As we empirically observed that a smaller learning rate value is more beneficial for the *FF+FB*, we can conclude that *FF+FB* learns in smaller steps. The direction of these steps is in general more targeted, given the smoothness of the accuracy curve for most datasets (see Table 3 and Appendix A).

Small η and g values can make the *FF+FB* network get stuck in local minima. For an unoptimized η, the *FF+FB* network uses the $te^l_{j,i}$ values to compensate for a poor η choice. A good choice of the g threshold becomes more important in this case. Threshold g was determined for each dataset using grid search, after η was selected. As we use the sigmoid activation function (with values less than 1), a $g = 0.9$ value would mean that only significant activations will be used in the TE computation. The learning rate η was selected for each dataset targeting the best results in 10 runs of the *FF* network and the same η has been used for *FF+FB*.

Constructing the time series, per epoch, for each training sample independently, does not produce good results: the obtained *te* values (scaled or not) were very small. Weighting the *te* values in this scenario, was also not a good approach.

The *te* values are tuned during Stage I. Calculating *te* after all training samples were processed proved to be a bad alternative.

Examining the *te* values raised new questions and we performed additional experiments to alleviate any possible bias in the results. The investigation was motivated by the negative and large *te* values (observed in most datasets) and their association with increased scores. Using these values in Equation (3) is not consistent with Equation (2), as negative values would mean that the source misinforms the target's next state. By subtracting *te* from 1, we revert the negative *te* values. Since the *te* values rarely exceed values like $+/-6$, this operation is also favorable for the positive extreme values.

For the *car* and *glass* datasets, where learning was slow or capped to inappropriate margins, we used the Weka package (version 3.9.3) with the MultiLayerPerceptron function, with identical or different hyperparameters, as needed, and, where required, with a lot more iterations, to validate our implementation's behavior for an established maximum accuracy. Our implementations of *FF+FB* and *FF* performed at least as good as the ones in Weka.

Computing $te^l_{j,i}$ for all neuron pairs is prohibitive even for shallow networks, especially for training sets with more than 10^5 samples. However, these are related only to TE computation and occur only during training Stage I. In practice, the trained weights of a neural network can be stored (*te* values are embedded in these weights). Therefore, in real-world applications, any inference tasks would not be affected by the increased training computational cost for *FF+FB*. Additionally, having the *te* values obtained in training Stage I stored, they can be reused as needed in training Stage II without

any computational overhead. Alternative TE estimation techniques may be considered for such cases [8].

7. Conclusions

We introduced *FF+FB*, a neural training algorithm which uses information transfer to quantify relationships between neurons and uses the TE feedback to enhance certain neural connections. Our method generally uses less training epochs and achieves higher accuracy compared to the *FF* network. In addition, it is more stable during training, as it can be observed from the plots in Appendix A, and less sensitive to local minima.

Using the TE feedback can reduce the effort for optimizing hyperparameters like η and number of hidden neurons in the hidden layer. According to our experiments, choosing an optimized value for the threshold parameter g can decrease the importance of other hyperparameters (e.g., η and the number of hidden nodes).

Our approach could facilitate the extraction of knowledge (and explanations) from the trained networks using the causality paradigm. This is left as an open problem.

Author Contributions: A.M., A.C. and R.A. equally contributed to the published work. All authors have read and agreed to the published version of the manuscript.

Funding: The costs to publish in open access were covered by Siemens SRL.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

te	transfer entropy value
g	the binning threshold
$te^{l}_{j,i}$	obtained te calculated using the time series produced by binning the outputs of neurons' index i from layer $l-1$ and index j from layer l
$te^{r,n}_{j,i}$	obtained te using the time series from outputs of neurons' index i from layer $l-1$ and index j from layer l, at epoch r and sample n
$o^{r,n}_{i}$	output of neuron having index i—that is layer $l-1$, at epoch r using sample n
$o^{r,n}_{j}$	output of neuron having index j—that is layer l, at epoch r using sample n
$s^{r,n}_{i}$	binning the output of neuron index i—layer $l-1$, at epoch r using sample n, with threshold g
$s^{r,n}_{j}$	binning the output of neuron index j—layer l, at epoch r using sample n, with threshold g
CNN	Convolutional Neural Network
FF+FB	Feedback Transfer Entropy—our proposed method
MLP	Multi-layer perceptron
FF	Non Feedback network, regular MLP architecture and algorithm
TE	Transfer Entropy

Appendix A. Validation Accuracy Evolution during Stage II

We present here the results of 10 runs on 10 standard benchmarks by illustrating the evolution of Stage II. We compare the dynamic behavior of *FF+FB*(in red) and *FF*(in blue) for a given target accuracy, with respect to the number of epochs required to reach that target. For most datasets, *FF+FB* improves its accuracy earlier and with a faster rate than *FF*.

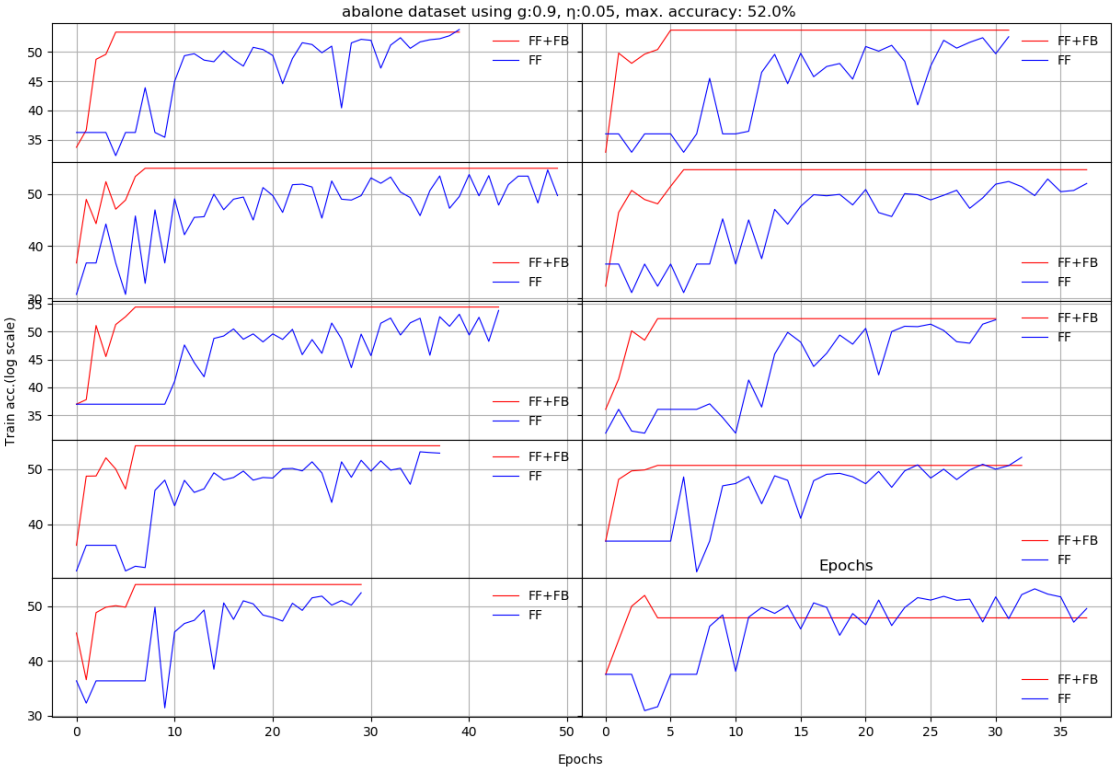

Figure A1. Ten runs of Abalone dataset. Each X axis finishes when the last of the *FF+FB* or *FF* reaches either the maximum number of epochs or the maximum set validation accuracy (log scale).

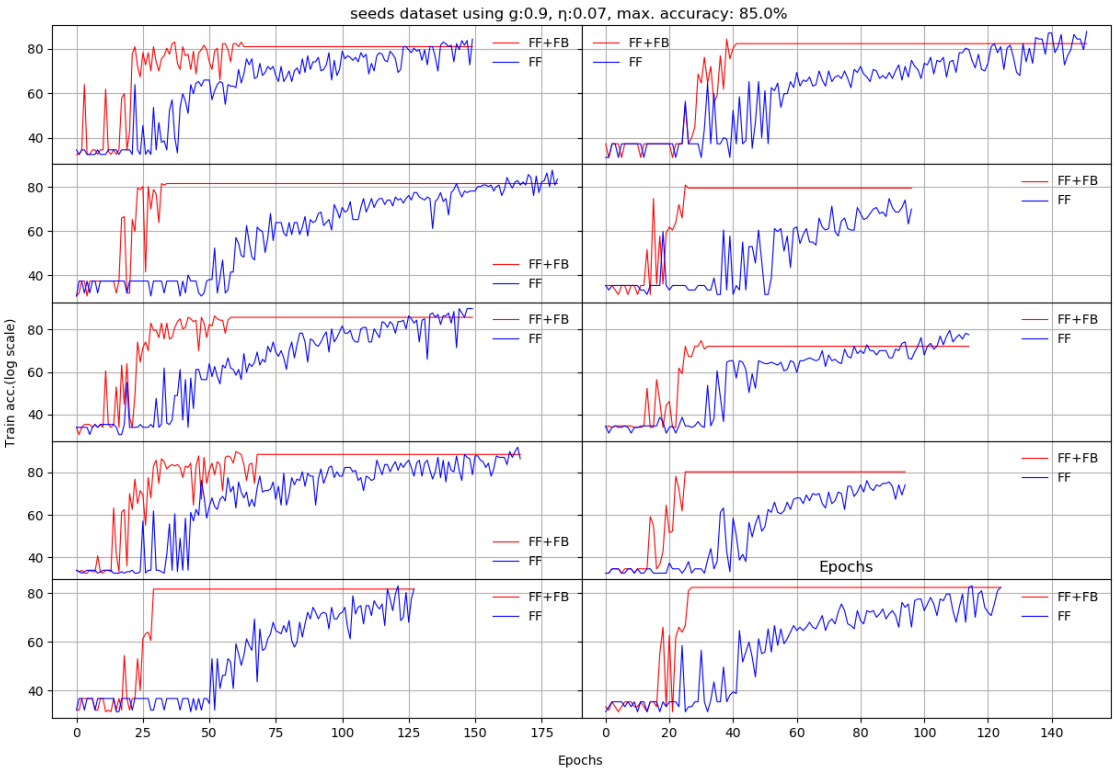

Figure A2. *Cont.*

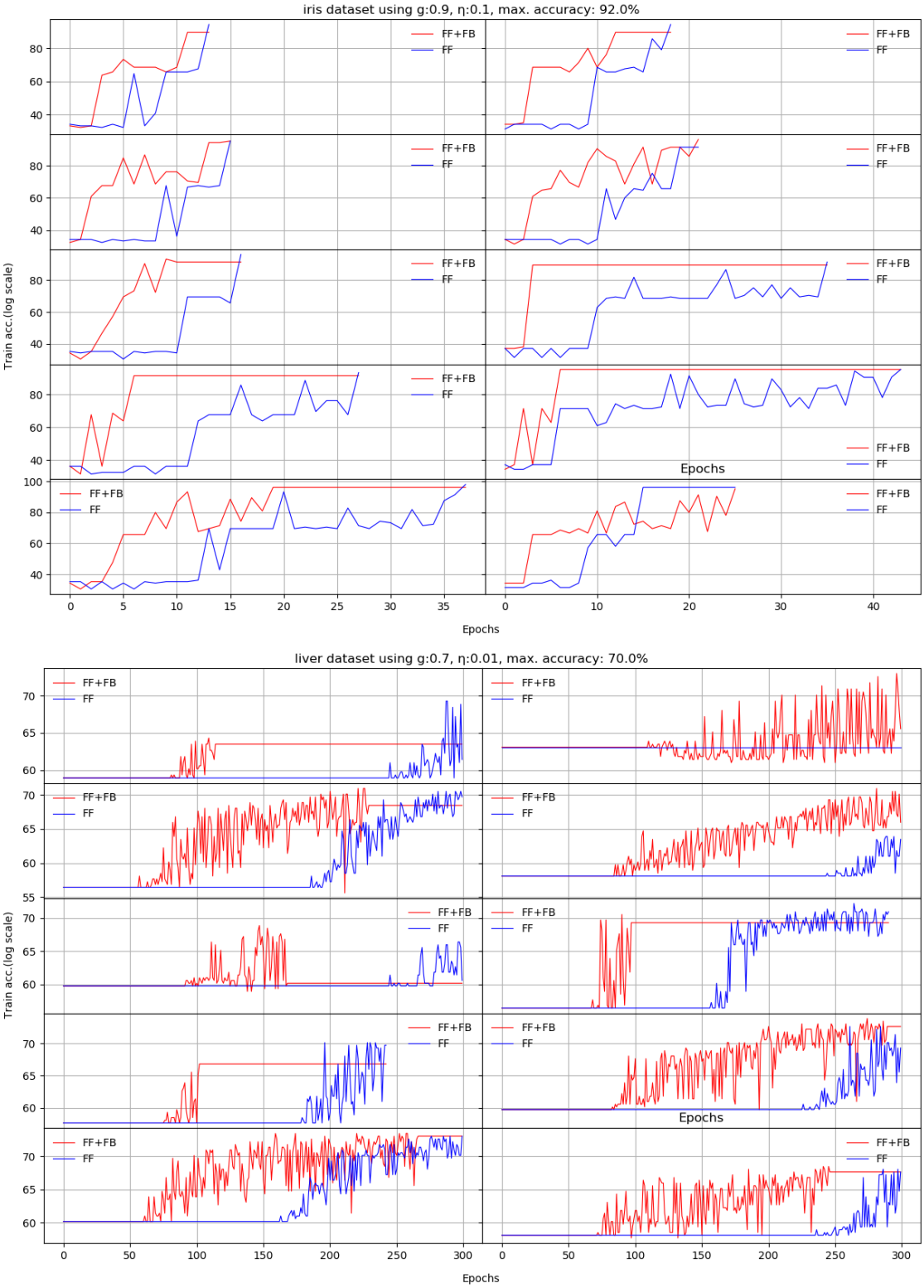

Figure A2. *Cont.*

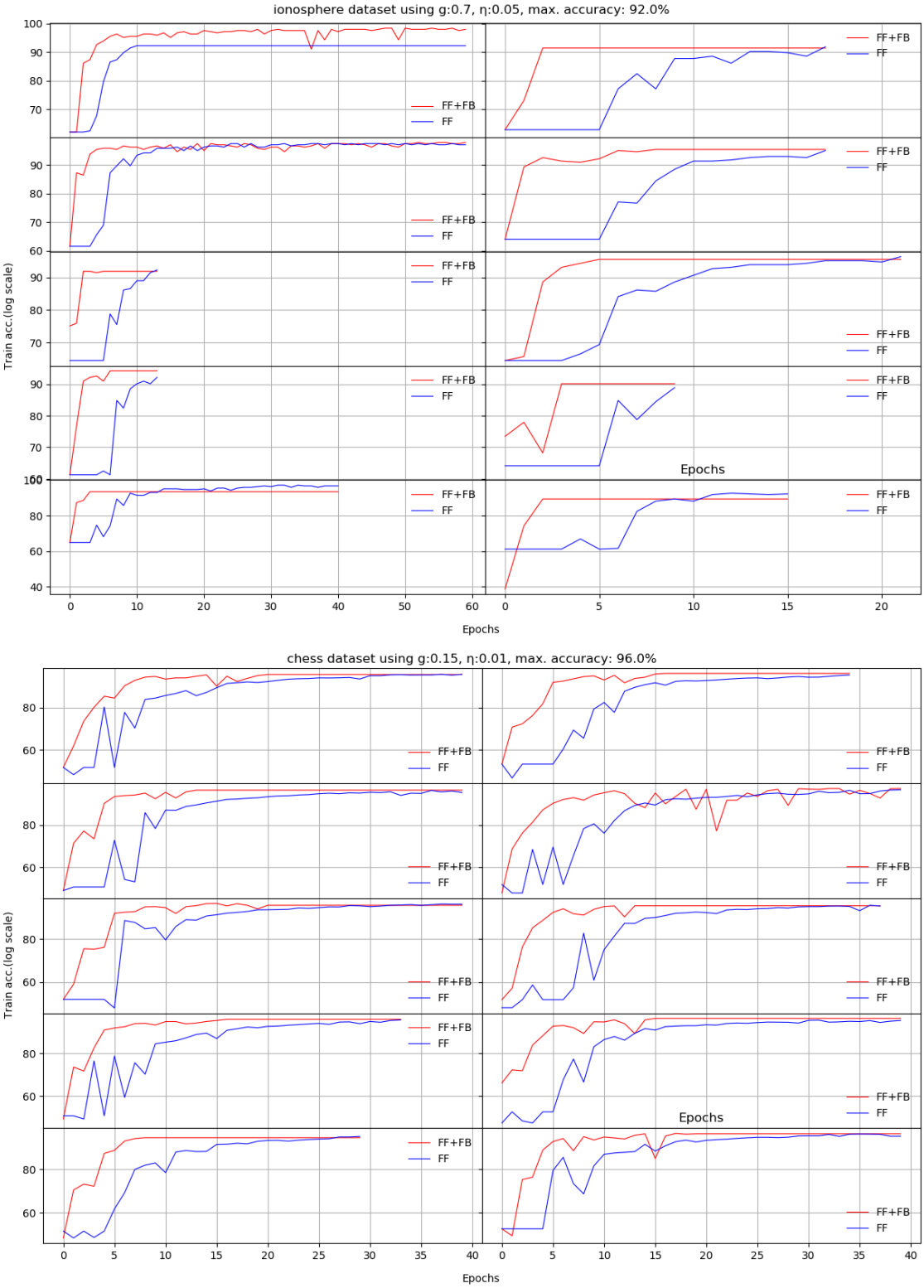

Figure A2. *Cont.*

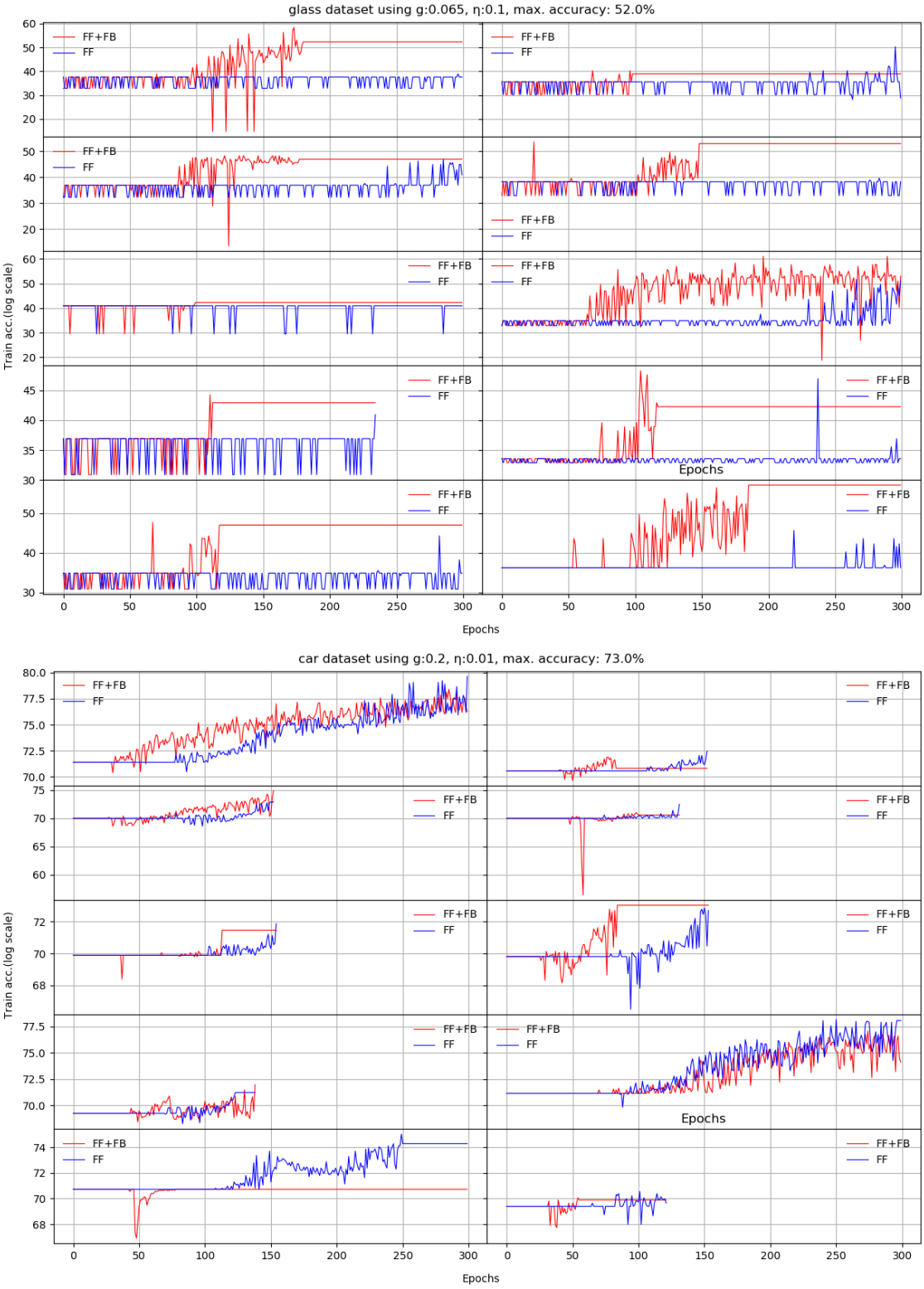

Figure A2. We tried several η values for this benchmark. To improve stability of both models, we selected a smaller value. It can be observed that *FF+FB* failed to converge on the 9th run and even to properly learn on other runs.

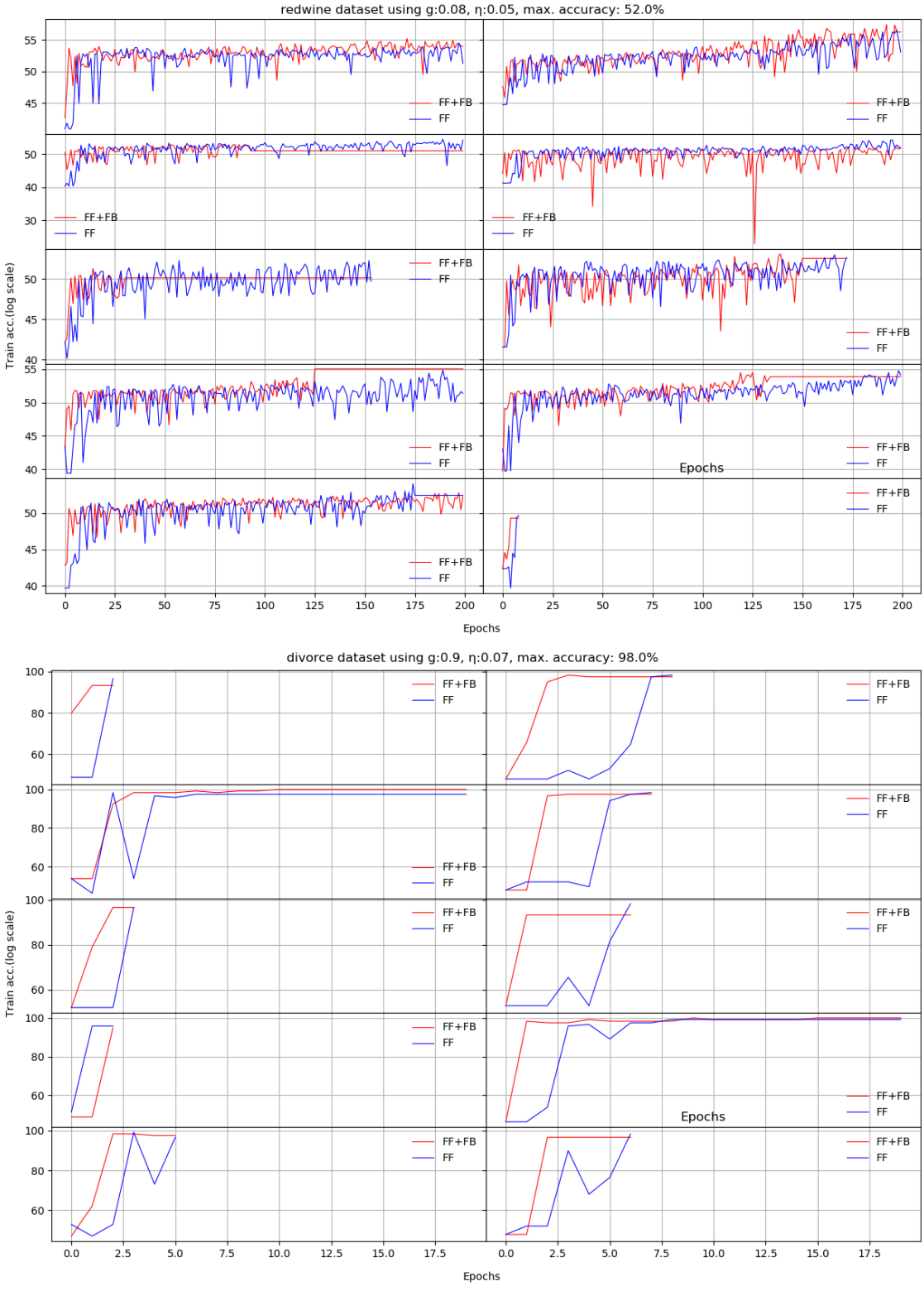

Figure A3. The *divorce* dataset constantly requires only a few epochs to reach the target accuracy.

References

1. Marwala, T. *Causality, Correlation and Artificial Intelligence for Rational Decision Making*; World Scientific: Singapore, 2015; doi:10.1142/9356. [CrossRef]
2. Shadish, W.; Cook, T.; Campbell, D. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*; Houghton Mifflin: Boston, MA, USA, 2001.

3. Zaremba, A.; Aste, T. Measures of Causality in Complex Datasets with Application to Financial Data. *Entropy* **2014**, *16*, 2309–2349. doi:10.3390/e16042309. [CrossRef]
4. Pearl, J. *Causality: Models, Reasoning and Inference*, 2nd ed.; Cambridge University Press: New York, NY, USA, 2009.
5. Schreiber, T. Measuring Information Transfer. *Phys. Rev. Lett.* **2000**, *85*, 461–464. [CrossRef] [PubMed]
6. Lizier, J.T.; Prokopenko, M. Differentiating information transfer and causal effect. *Eur. Phys. J. B* **2010**, *73*, 605–615. [CrossRef]
7. Cataron, A.; Andonie, R. Transfer Information Energy: A Quantitative Causality Indicator Between Time Series. In Proceedings of the Artificial Neural Networks and Machine Learning—ICANN 2017—26th International Conference on Artificial Neural Networks, Alghero, Italy, 11–14 September 2017; Springer: Cham, Switzerland, 2017; pp. 512–519. doi:10.1007/978-3-319-68612-7_58. [CrossRef]
8. Cataron, A.; Andonie, R. Transfer Information Energy: A Quantitative Indicator of Information Transfer between Time Series. *Entropy* **2018**, *20*. doi:10.3390/e20050323. [CrossRef]
9. Massey, J.L. Causality, feedback and directed information. In Proceedings of the 1990 International Symp. on Information Theory and Its Applications, Hawaii, USA, 27–30 November 1990, pp. 303–305.
10. Lizier, J.T.; Heinzle, J.; Horstmann, A.; Haynes, J.D.; Prokopenko, M. Multivariate information-theoretic measures reveal directed information structure and task relevant changes in fMRI connectivity. *J. Comput. Neurosci.* **2011**, *30*, 85–107. doi:10.1007/s10827-010-0271-2. [CrossRef] [PubMed]
11. Vicente, R.; Wibral, M.; Lindner, M.; Pipa, G. Transfer entropy—A model-free measure of effective connectivity for the neurosciences. *J. Comput. Neurosci.* **2011**, *30*, 45–67. doi:10.1007/s10827-010-0262-3. [CrossRef] [PubMed]
12. Shimono, M.; Beggs, J.M. Functional Clusters, Hubs, and Communities in the Cortical Microconnectome. *Cerebral Cortex* **2015**, *25*, 3743–3757. [CrossRef] [PubMed]
13. Fang, H.; Wang, V.; Yamaguchi, M. Dissecting Deep Learning Networks—Visualizing Mutual Information. *Entropy* **2018**, *20*. doi:10.3390/e20110823. [CrossRef]
14. Obst, O.; Boedecker, J.; Asada, M. Improving Recurrent Neural Network Performance Using Transfer Entropy. In Proceedings of the 17th International Conference on Neural Information Processing: Models and Applications (ICONIP 2010), Sydney, Australia, 22–25 November 2010; Springer: Berlin, Heidelberg, 2010; Volume II, pp. 193–200.
15. Herzog, S.; Tetzlaff, C.; Wörgötter, F. Transfer entropy-based feedback improves performance in artificial neural networks. *arXiv* **2017**, arXiv:1706.04265.
16. Féraud, R.; Clérot, F. A methodology to explain neural network classification. *Neur. Netw.* **2002**, *15*, 237–246. [CrossRef]
17. Patterson, J.; Gibson, A. *Deep Learning: A Practitioner's Approach*, 1st ed.; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2017.
18. Bossomaier, T.; Barnett, L.; Harré, M.; Lizier, J.T. *An Introduction to Transfer Entropy. Information Flow in Complex Systems*; Springer: Berlin, Germany, 2016.
19. Hlaváčková-Schindler, K.; Paluš, M.; Vejmelka, M.; Bhattacharya, J. Causality detection based on information-theoretic approaches in time series analysis. *Phys. Rep.* **2007**, *441*, 1–46. [CrossRef]
20. Kaiser, A.; Schreiber, T. Information transfer in continuous processes. *Phys. D* **2002**, *166*, 43–62. [CrossRef]
21. Gencaga, D.; Knuth, K.H.; Rossow, W.B. A Recipe for the Estimation of Information Flow in a Dynamical System. *Entropy* **2015**, *17*, 438–470. [CrossRef]
22. Hlaváčková-Schindler, K. Causality in Time Series: Its Detection and Quantification by Means of Information Theory. In *Information Theory and Statistical Learning*; Emmert-Streib, F.; Dehmer, M., Eds.; Springer: Boston, MA, USA, 2009; pp. 183–207. doi:10.1007/978-0-387-84816-7_8. [CrossRef]
23. Zhu, J.; Bellanger, J.J.; Shu, H.; Le Bouquin Jeannès, R. Contribution to Transfer Entropy Estimation via the k-Nearest-Neighbors Approach. *Entropy* **2015**, *17*, 4173–4201. doi:10.3390/e17064173. [CrossRef]
24. Kwon, O.; Yang, J.S. Information flow between stock indices. *EPL (Europhys. Lett.)* **2008**, *82*, 68003. [CrossRef]
25. Sandoval, L. Structure of a Global Network of Financial Companies Based on Transfer Entropy. *Entropy* **2014**, *16*, 4443–4482. doi:10.3390/e16084443. [CrossRef]
26. Prokopenko, M.; Lizier, J.T.; Price, D.C. On Thermodynamic Interpretation of Transfer Entropy. *Entropy* **2013**, *15*, 524–543. doi:10.3390/e15020524. [CrossRef]

27. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*; MIT Press: Cambridge, MA, USA, 1986; Volume 1, pp. 318–362.
28. Dua, D.; Graff, C. UCI Machine Learning Repository. Available online: https://archive.ics.uci.edu/ml/index.php (accessed on 25 November 2019).