TUNING THE POSITION OF A FUZZY COGNITIVE MAP ATTRACTOR USING BACKPROPAGATION THROUGH TIME

Michal Gregor[(a)], **Peter P. Groumpos**[(b)]

[(a)]Department of Control and Information Systems, Faculty of Electrical Engineering, University of Žilina
[(b)]Laboratory for Automation and Robotics, Department of Electrical and Computer Engineering, University of Patras

[(a)]michal@gregor.sk, [(b)]groumpos@ece.upatras.gr

ABSTRACT
The paper proposes a new learning method for fuzzy cognitive maps, which makes it possible to encode an attractor into the map. The method is based on the principle of backpropagation through time known from the theory of artificial neural networks. Simulation results are presented to show how well the method performs. It is shown that the results are superior to those achieved using Hebbian learning approaches such as nonlinear Hebbian learning. Some lines for possible future research and development are given.

Keywords: fuzzy cognitive maps, learning, backpropagation.

1. INTRODUCTION
Fuzzy cognitive maps (FCMs) represent a well recognized method in the theory of soft computation. They exhibit several noticeable traits, which make them similar to artificial neural networks (ANNs). More specifically the FCM can be considered as a distinct type of a single-layer recurrent neural network with synchronous activation of units.

Some learning methods from the theory of ANNs have been introduced into the theory of FCMs before. Most notable among these approaches is a family of methods based on Hebbian learning.

The paper will present a new method for encoding an attractor state into an FCM using the principle of backpropagation through time. The theoretical background of the method will be outlined hereinafter, and results of several simulation experiments will be presented as well as their evaluation and discussion of possible future lines of research.

2. DELTA RULE AND BACKPROPAGATION
Let us in this section cover some of the basic theory of supervised error correction learning in artificial neural networks (ANNs). It will later be shown how ANNs are related to fuzzy cognitive maps (FCMs), and how this theory may find applications there.

2.1. Artificial Neuron
In order to set down how the learning method works, let us first with all due brevity give a basic definition of what we mean by the artificial neuron, and by artificial neural networks in general.

An artificial neuron is a structure, which has (a) its set of inputs $X=\{x_1, x_2, \dots, x_n\}$, (b) a set of weights corresponding to the inputs $W=\{w_1, w_2, \dots, w_n\}$, (c) its threshold Θ, its squashing function f (which is perhaps most often the sigmoid function, or the hyperbolic tangent) (Krose and Smagt 1996). The illustration is given in Figure 1.

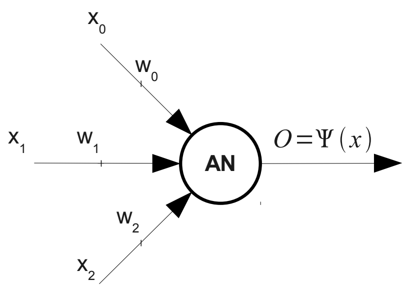

Figure 1: Unwrapping a recurrent network

The output of the neuron is determined in the following fashion (Krose and Smagt 1996):

$$O=f(u-\Theta)=f\left(\sum_{i=1}^{n}(w_i x_i)-\Theta\right), \tag{1}$$

where u is the inner potential of the neuron.

Furthermore, in order to simplify derivation of the learning rules, the threshold value Θ is often interpreted as a bias received from a neuron with the constant output of 1 – thus the threshold value does not have to be considered separately, and its value can be learned in the same way as the weights of the inputs.

By an artificial neural network (ANN) we simply understand a collection of mutually interconnected artificial neural networks.

2.2. The Delta Rule
The delta rule is probably the best known approach (based on error correction) to learning weights of an artificial neuron. It has been designed for supervised learning – that is to say learning from a dataset

consisting of pairs of the following form: *(input, desired output)*. In other words, for every sample's input in the dataset, its corresponding desired output is specified as well.

This allows formulation of an error function (Krose and Smagt 1996):

$$E(W)=\sum_p E^p(W)=\frac{1}{2}\sum_p (D^p-O^p)^2 , \tag{2}$$

where D^p and O^p denote the desired and the real output for input pattern p .

Once such error function has been formed, the idea behind delta rule is to do gradient descent minimization with respect to the weights, thus bringing the real output as close to the desired output as possible. For linear neurons (neurons with no squashing function) the following rule – known as the delta rule – has been derived (Krose and Smagt 1996):

$$\Delta_p w_j=\gamma\delta^p x_j , \tag{3}$$

where $\Delta_p w_j$ denotes the prescribed change of weight w_j due to pattern p , γ is the learning rate, and:

$$\delta^p=D^p-O^p . \tag{4}$$

The delta rule can be generalized for non-linear neurons as well, in which case it contains the derivative of the squashing function $f^{'}$ (Krose and Smagt 1996):

$$\delta^p=(D^p-O^p)f^{'}(u^p) , \tag{5}$$

where u^p is the inner potential of the neuron with pattern p at the input.

2.3. The Backpropagation Principle

The delta rule cannot by itself be used for learning in multi-layer networks, because only the errors of the output neurons can be computed directly using (3) or (4) – desired outputs of hidden neurons remain unspecified.

However, the delta rule can be generalized to multi-layer networks using the backpropagation principle.

In this case the error is propagated back from the output layer to hidden layers. Again, the full derivation of the rule can be found in (Krose and Smagt 1996). The rule itself can be stated as follows:

$$\delta_h^p=F^{'}(u_h^p)\sum_{o=1}^{N_o}\delta_o^p w_{ho} , \tag{6}$$

where h refers to a neuron of the hidden layer, and o refers to neurons of the output layer. N_o is the number of neurons in the output layer. If there are several hidden layers, the principle can be applied recursively.

2.4. Backpropagation Through Time

Backpropagation through time (BPTT) is a further extension of the principle to recurrent neural networks. Recurrent neural networks (RNNs) are networks in which signals may propagate from one time step to another (as opposed to feed-forward neural networks, which only propagate inputs from the current step and have no memory).

The idea is that an RNN can be unwrapped in time into a feedforward neural network, and then trained using backpropagation.

Figure 2 shows an example of unwrapping a single-layer recurrent neural network in 3 time steps, resulting in a feedforward neural network with 3 layers.

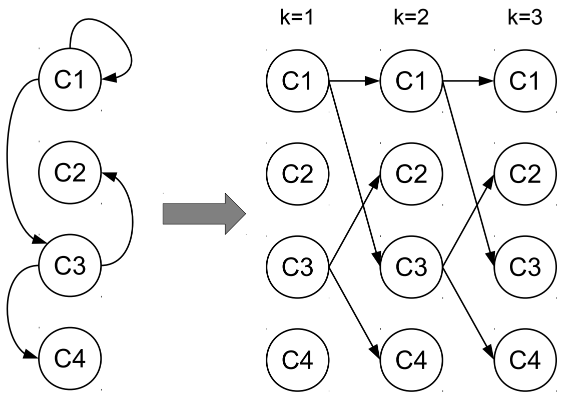

Figure 2: Unwrapping a recurrent network

For additional details and precise mathematical and algorithmic formulations the reader may refer to Werbos (1990). For the present purpose, what has already been said concerning backpropagation itself should suffice.

2.5. Vanishing/Exploding Gradients

The backpropagation principle faces significant problems when applied to deep neural networks. The same kind of problem is usually associated with RNNs using BPTT – because RNNs become deep themselves when unwrapped in time.

The problem that occurs is referred to as the vanishing/exploding gradients problem. When the errors are propagated back through the network, unlike the forward pass no squashing functions are applied. Depending on the values of the weights, the gradients tend to grow very small (*vanishing gradient*), or very large (*exploding gradient*) after they have been propagated through a number of layers (Sutskever, Martens, Hinton 2011).

However, lately some considerable advances in deep learning have been introduced through the work of Hinton, and Salakhutdinov (2006), and Bengio et al. (2007). These mainly advocate careful initialization of the weight matrix based on unsupervised pre-training.

An important breakthrough in the theory of deep learning has been marked by Martens (2010). The author has developed an application of Hessian-free optimization method to learning in deep networks. It has later been shown that this approach can indeed be adapted for recurrent neural networks as well (Martens, and Sutskever 2011). Applications of the method have since begun to appear (Sutskever, Martens, Hinton 2011).

In consequence of these developments, the backpropagation principle has lately been experiencing a renaissance. Backpropagation coupled with Hessian-free optimization is now counted among the most promising tools for learning in both – deep neural networks and recurrent networks.

3. FUZZY COGNITIVE MAPS

Fuzzy cognitive maps (FCMs) are a symbolic representation for the description and modeling of complex systems (Groumpos 2010). They can be expressed and visualized using a weighted directed graph such as that shown in Figure 3.

The nodes of such graph represent the concepts associated with the system being modeled. The number and kind of concepts that form any particular FCM is determined by experts from the corresponding field of knowledge (Groumpos 2010). Every concepts C_i is associated with its activation value A_i. The activation values are most often taken from the interval $[0,1]$, or $[-1,1]$ (this depends on the particular squashing function used – see equation (7) for the context).

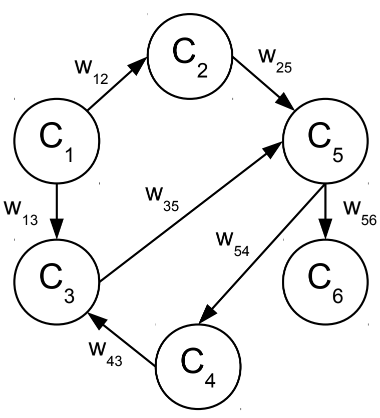

Figure 3: Fuzzy cognitive map – an example

The edges in the graph are directed and weighted. They express causal relationship between the concepts. An edge directed from concept C_i to concept C_j represents the knowledge that there is a causal link between C_i and C_j.

The weight of the edge going from C_i to C_j is denoted $w_{ij} \in [-1,1]$, and it specifies the strength of the causal link. If $w_{ij} > 0$, we can say that concept C_i causes C_j to some extent – it contributes positively to its activation value. If $w_{ij} < 0$, concept C_i has negative influence on the activation value of C_j. If $w_{ij} = 0$, there is no causal link at all (such edges are usually not drawn when the FCM is visualized).

At every time step the activation values of the concepts are updated. The update is synchronous. The update rule has several distinct forms. First of all there is the rule proposed in Kosko (1993) [the notation has been modified for the sake of consistency]:

$$A_i^{(k+1)} = f\left(\sum_{j=1}^{N} A_j^{(k)} w_{ji}\right), \tag{7}$$

where N is the number of concepts, $A_i^{(k)}$ is the activation value of concept C_i at time step k. f is the squashing function, which squashes the dot product $\sum_j A_j^{(k)} w_{ji}$ into some convenient interval.

Most often, f is either the sigmoid function, which squashes the dot product into interval $[0,1]$:

$$S(x) = \frac{1}{1+e^{-x}}, \tag{8}$$

or the hyperbolic tangent, which yields the interval $[-1,1]$.

There are other forms of the rule. In some works, such as Groumpos and Stylios (2000) the feedback links from the concept to itself are removed:

$$A_i^{(k+1)} = f\left(\sum_{\substack{j=1\\ j\neq i}}^{N} A_j^{(k)} w_{ji}\right). \tag{9}$$

In other versions, the feedback link is reintroduced, but the same weight is shared by all concepts (Groumpos and Stylios 2000):

$$A_i^{(k+1)} = f\left(k_1 \sum_{\substack{j=1\\ j\neq i}}^{N} A_j^{(k)} w_{ji} + k_2 A_i^{(k)}\right), \tag{10}$$

where k_1 and k_2 are constants, such that $0 \leq k_1, k_2 \leq 1$.

Unless specified otherwise, we will adhere to the rule given in equation (7) in this work, because that is the most general one.

The weight matrix of the FCM is usually constructed by experts. There are several approaches which make the task easier and more reliable – for discussion of these, the reader may refer to Groumpos (2010), Stach, Kurgan, Pedrycz (2005a), or Stylios, Christova, and Groumpos (2002) for an instance.

It should also be noted that the FCM need not be acyclic. In the presence of cycles, the issue of stability naturally comes into mind. As mentioned in Dickerson, and Kosko (1993), the FCM will quickly settle down to a fixed point, to a limit cycle, or to a chaotic attractor (for illustration, see Figure 4).

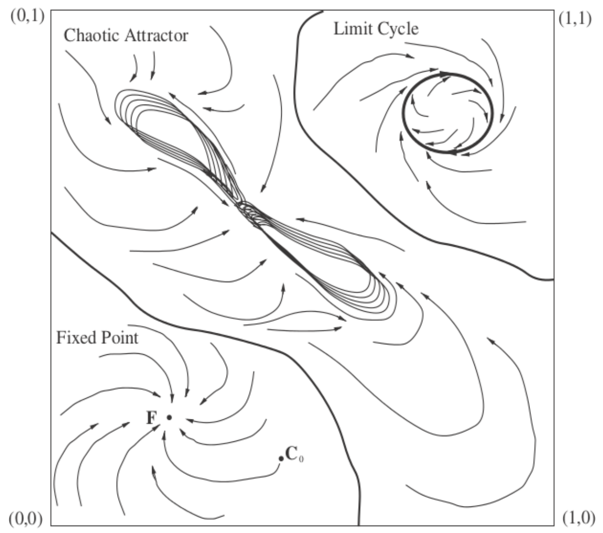

Figure 4: FCM attractors: fixed-point, limit-cycle, chaotic (Dickerson, and Kosko 1993)

3.1. The Relation between FCMs and Artificial Neural Networks

It is obvious that there is a close resemblance between the FCM model as described in equation (7), and the ANN model as shown in equation (1). More specifically, an FCM can be considered as a distinct type of a single-layer recurrent neural network with sigmoid squashing functions and synchronous activation of units.

The FCM can also be viewed as an extension of the concept of a Hopfield network to which its architecture bears several similarities. However, the analogy is not complete – there are several traits in which these differ. Most notably perhaps, the Hopfield network has been designed for binary (0 and 1) or bipolar (-1 and 1) activations, it uses the sign function as the squashing function, and its units are activated asynchronously. Also, the theory of learning developed for Hopfield is based on the concept of an energy function. The energy function derived for Hopfield networks requires that the weight matrix be symmetric (Krose and Smagt 1996), (Rojas 1996).

ANNs are notorious for their lack of interpretability. The knowledge they contain is implicit – it is sometimes called *procedural knowledge* fro this reason.

In spite of their close relationship with ANNs, FCMs do not generally suffer from the same problem, because the knowledge they represent is modelled very explicitly. If the FCM is constructed by experts then it naturally follows that its dynamics are (by definition) well understood. Even in cases when one learning method or other is used to modify the weight matrix or to learn it from scratch, the inner workings of the resulting model typically remain transparent, due to the fact that every node is linked with a distinct concept.

Naturally, this feature is paid for by the fact that the FCM is not a universal approximator. To put this more precisely – if the concepts are linked by a more complex relationship, which cannot be adequately represented using a single connection, the expert must explicitly provide all the auxiliary concepts required to express such relationship. In multi-layer artificial neural networks, neurons are not associated with particular concepts, and thus any of them is free to participate in modelling the relationship.

It may also be noted that – in contrast to ANNs (1), the FCM model (7) does not contain the threshold term. Thus, the absolute term of the linear relationship expressing the inner potential u is missing – if any given concept is to have some non-zero potential even though all of its input concepts have the activation value of zero, this is not possible to achieve unless an additional concept is added for that express purpose.

In any case, it should be obvious that there is a trade-off between the excellent interpretability of FCMs on one hand, and the ability of multi-layer ANNs to do universal approximation. Some work has been invested into creating hybrid models, which offer trade-off points closer to the conventional ANNs. Among these the fuzzy cognitive network of Kottas, Boutalis, and Christodolou (2007) can be mentioned, or the fuzzy neural network of Wang and Wang (2013) can be mentioned.

Furthermore, even though the concepts and weights of a conventional FCM are usually determined by experts, the close connection between the theory of FCMs and the theory of ANNs has been fuelling a promising line of research, which strives to bring some of the learning methods from the theory of ANNs to perform learning in FCMs. The next section covers some of the existing approaches.

3.2. Fuzzy Cognitive Maps and Learning

We have now covered some of the basic ideas and concepts associated with FCMs and their close relation to ANNs. In this section, we will consider the problem of learning the weight matrix of an FCM.

We note that there are two main classes of problems in the theory of FCMs, to which the existing learning methods apply: (a) *the regression problem*, that is to say how a fuzzy cognitive map can be trained as a regression model for a given dataset; (b) *the attractor problem*, that is to say given an initial fuzzy cognitive map, how can we shift a fixed-point attractor to a desired point, to encode a given limit-cycle, etc.

The most notable among the existing approaches inspired by the theory of ANNs are those based on Hebbian learning – to the description of these we now turn.

3.2.1. Differential Hebbian Learning

The first among the methods inspired by Hebbian learning is the so-called *differential Hebbian learning* (DHL) proposed by Dickerson and Kosko (1993). It introduces the following rule (notation modified for the sake of consistency):

$$\Delta w_{ij} = \begin{cases} c_t \left[\Delta A_i \Delta A_j - w_{ij}\right] & \text{if } \Delta A_i \neq 0 \\ 0 & \text{else} \end{cases}, \quad (11)$$

where $\Delta A_i(t) = A_i(t) - A_i(t-1)$, and c_t is the learning rate, which decreases in time. Dickerson and Kosko (1993) propose the following rule for c_t:

$$c_t(t_k) = 0.1\left[1 - \frac{t_k}{1.1\,N}\right]. \quad (12)$$

A follow-up on this method can be found in Huerga (2002). The author shows that rule (11) has a considerable weakness in that it cannot encode some types of sequences correctly. If, for example, concept 4 should change its activation to 1 after concepts 1, 2, and 3 change their activations to 1, after learning is performed using DHL, concept 4 will react to any of the concepts, and not only to their simultaneous activation (Huerga 2002). The approach know as *balanced DHL*, is proposed instead in Huerga 2002.

Both of these approaches – DHL and balanced DHL – are able to encode attractors into the FCM. However, both of them also share a common disadvantage: the authors only consider encoding binary (or bipolar) attractors (as opposed to real-valued attractors). More importantly, even for such cases, the rules are unable to encode an arbitrary sequence.

3.2.2. Active Hebbian Learning

In Papageorgiou, Stylios, and Groumpos (2004b) the authors propose another learning method based on Hebbian learning – the *active Hebbian learning* (AHL) method.

AHL introduces the idea of the sequence of activation of the concepts. The expert specifies the sequence and manner (synchronous, asynchronous) in which the concepts are activated. The process starts from a concept which activates concepts linked to it, and thus the activation propagates until all the concepts have been activated at which point the simulation cycle stops and a new one starts.

Learning is based on the following rule:

$$w_{ji}(k) = (1-\gamma)\, w_{ji}(k-1) + \eta\, A_j^{act}(k-1)\, A_i(k-1), \quad (13)$$

where γ, and η are tunable parameters, and A_j^{act} is the value of the activation concept. In addition to this Papageorgiou, Stylios, and Groumpos (2004b) also suggest normalization of the weight matrix to size 1 after every step, so as to prevent their growing indefinitely.

3.2.3. Nonlinear Hebbian Learning

Finally, there are several papers (e.g. Papageorgiou, Stylios, and Groumpos 2003; Papageorgiou, and Groumpos 2005a; Papageorgiou, Stylios, and Groumpos 2006; Stach, Kurgan, and Pedrycz 2008) discussing the so-called *nonlinear Hebbian learning* (NHL). The learning rule used in this approach is based on the Oja rule, and it has the following form:

$$\Delta w_{ji} = \eta\, A_j\left(A_i - A_j\, w_{ji}\right). \quad (14)$$

In Papageorgiou, and Groumpos (2005a) the rule is further extended into the following:

$$\begin{aligned} w_{ji}^{(k)} &= \gamma\, w_{ji}^{(k-1)} + \eta\, A_j^{(k-1)}\Big(A_i^{(k-1)} \\ &\quad - sgn\left(w_{ji}^{(k-1)}\right) A_j^{(k-1)}\, w_{ji}^{(k-1)}\Big) \end{aligned}. \quad (15)$$

This introduces the weight decay parameter γ, and term $sgn\left(w_{ji}^{(k-1)}\right)$, which is supposed to maintain the sign of the corresponding weight (thus keeping the physical meaning of the corresponding relationship between the concepts) (Papageorgiou, and Groumpos 2005a).

It should also be noted that the mode in which in these works NHL rule is coupled with the execution of the FCM is what makes the method quite distinct from the other learning methods discussed so far.

An initial FCM is constructed by the experts. This is run in the standard way using one of the equations (7), (9), (10). But in addition to this at every step, rule (14) or (15) is applied to the FCM using the values of concepts computed in that step. Thus the NHL method does not simply perform learning – it is used online to help drive the process, and to facilitate convergence to a target attractor. The method is not simply meant to train the FCM to perform a given task, or to encode a given attractor. Rather, it actively participates in the *execution* of the FCM – it adjusts the FCM cause-effect relationships and controls the system's output concepts (Papageorgiou, Stylios, and Groumpos 2006).

A more traditional application of the NHL rule is proposed by Stach, Kurgan, and Pedrycz (2008). In this case, NHL is used to make an FCM with a randomly initialized weight matrix learn the cause-effect relationships from historical data. In other words – the FCM learns to approximate a given data sequence. The authors use another randomly generated FCM to produce the data sequence, but the data could, naturally, come from the real system. The authors call this approach *data-driven NHL* (DD-NHL).

4. THE PROPOSED APPROACH

In this paper, we aim to approach the attractor problem using the concept of backpropagation through time. We refer to it as *BPTT-based attractor setting* (BAS). That is to say, given a specification of what the steady state should look like, and how quickly the model should converge to it, we use BPTT to encode this information into the FCM.

4.1. Using BPTT to Set the Attractor

Using BPTT we can train the FCM to give the desired response to a desired input. If we present it with an input, and run it for k steps, we can compare the real output with the desired output, compute the error and backpropagate it through time in order to learn the weights.

Thus, if our intention is to set the attractor of the FCM, we may create an FCM with a randomly initialized weight matrix, run it for k steps from a number of randomly-generated initial states, and do backpropagation. The learning method should be able to generalize, and so when we run the FCM from an initial state for which it has not been trained, it should still end up in the same attractor.

In theory it should certainly possible to encode limit cycles as well as fixed-point attractors, nevertheless in this paper we will focus on fixed-point attractors only. It is obvious that in addition to coming to the desired state after k steps, the FCM should also remain in that state, if it is supposed to be an attractor.

To ensure this, we propose to specify that the FCM should be in the desired state after k steps (*BPTT steps*), and should retain the same for m further steps (*stabilising steps*). If m is large enough, this should ensure that the FCM remains in the desired state indefinitely.

It should also be noted, that for the m steps, no BPTT needs to be done, because having the desired activation value of every concept, we can compute the error in every step precisely. BPTT is thus only done for the k steps.

However, backpropagation and BPTT could be useful in such cases, when the entire attractor is not specified, but rather only the values of certain concepts are available. In such case, we can use backpropagation and BPTT to compute errors for the missing concepts.

4.2. Vanishing/Exploding Gradients in FCMs

The vanishing/exploding gradient problem has been mentioned in connection with backpropagation learning in ANNs. Naturally the same problem may occur in FCMs. We conclude, however, that – probably owing to their limited size (according to Stach et al. (2005b) in practice FCMs are relatively small and most of them consist of 5-10 concepts), and also due to the fact that they are single-layer themselves – we did not find the gradient explosion/vanishing problem to hinder learning considerably.

The problem does occur in some cases (which is to be expected – especially when using naïve initialization methods), but these are not very frequent and it is possible to work around the problem by selecting a different initial point in the weight space and starting learning anew from there.

Furthermore, preliminary tests seem to indicate that if learning starts from an FCM constructed by the expert, the problem disappears altogether. Application of BPTT together with learning based on hessian-free optimization is nevertheless of considerable interest – especially if we are to encode chaotic attractors or limit cycles into the FCM (as opposed to simple fixed-point attractors), such more reliable learning methods may be called for.

5. SIMULATION EXPERIMENTS

In this section, we will present the results achieved using BPTT-based attractor setting. The first part will cover experiments starting from an FCM with a randomly-generated initial weight matrix, into which a randomly selected fixed-point attractor is encoded. The accuracy of the resulting model is reported. The second part provides a comparison with results reported for AHL and NHL in Papageorgiou, Stylios, and Groumpos (2006). There is also a third section, which offers some 2-dimensional visualizations.

5.1. Encoding Attractors into FCMs

In every experiment we do BPTT from a number of randomly selected initial points. It is obvious that this will make the process highly stochastic, and in consequence we can expect the results to have considerable variance. Therefore, unless otherwise specified, all experiments were carried out 100 times, and their results were averaged. The learning rate is set to 0.2 unless said otherwise.

For computing the attractor, 10 initial points are randomly selected, and the FCM is run for the maximum number of 1000 steps, or until the change of the activation values of the concepts from one step to another is less than 1E-30 in terms of the mean squared error (MSE). Afterwards, we compare the real attractor of the FCM as computed with the target attractor (again using MSE).

In some cases the gradient explosion problem makes the weights grow indefinitely so that they become NaN (not-a-number). We detect these cases and report their number. Naturally, MSE cannot be computed in these cases and thus these runs are not included in the average. Gradient vanishing probably occurs in some cases as well, but we make no special effort to detect those cases. If we did exclude those cases from the MSE, the results might be somewhat improved.

5.1.1. Number of Initial Points

As mentioned, in every experiment, BPTT starts from a number of randomly generated initial points. The first collection of results shows the effect of that the number of said initial points has on the accuracy (in terms of MSE) and on the number of cases in which gradient explosion occurs.

Table 1: Number of gradient explosions, and the MSE for various numbers of stabilising steps

# of init points:	**10**	**30**	**50**	**70**
MSE	2.827 **E-3**	2.745 **E-3**	1.606 **E-3**	1.403 **E-5**
Grad. expl. #	2	4	8	14

The results are for an FCM with 5 concepts, the number of BPTT steps $k=5$, and $m=30$ stabilisation steps. 10 random initial vectors are used for training, and 5 training steps are applied for each.

It is obvious from Table 1 that with increasing the number of initial points, the MSE improves. This can be accounted for by the fact that the amount of training data increases. This should also improve generalization.

5.1.2. Number of Stabilising Steps

The following table shows the effect that the number m of stabilising steps has on accuracy and on the tendency of the algorithm to exhibit gradient explosion. The results are for an FCM with 5 concepts, the number of BPTT steps $k=5$, 10 random initial vectors are used for training, and 5 training steps are applied for each.

Table 2: Number of gradient explosions, and the MSE for various numbers of stabilising steps

m:	**10**	**30**	**50**
MSE	1.178 **E-2**	2.827 **E-3**	3.356 **E-4**
Grad. expl. #	0	2	4

As shown in Table 2, there is a correlation between the number of stabilising steps and the accuracy. Accuracy improves with increasing the number of stabilising steps, but the number of gradient explosions increases with it as well.

5.2. Visualization

Figures 5 and 6 show examples of randomly-generated attractor states encoded into an FCM. The FCM is run from a number of randomly chosen initial points to show how the FCM will eventually converge into the attractor state in every case.

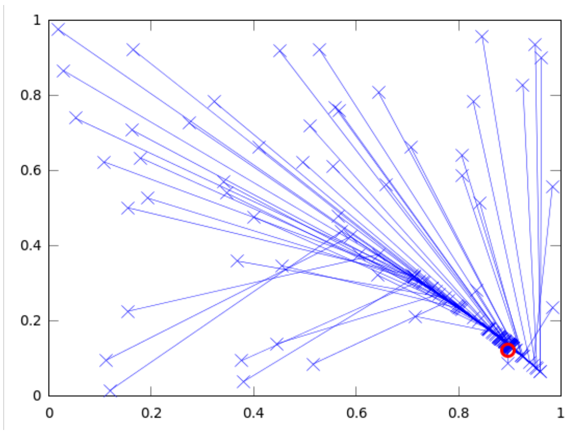

Figure 5: An example of a randomly generated attractor state encoded into an FCM; convergence to the state from randomly selected starting points shown

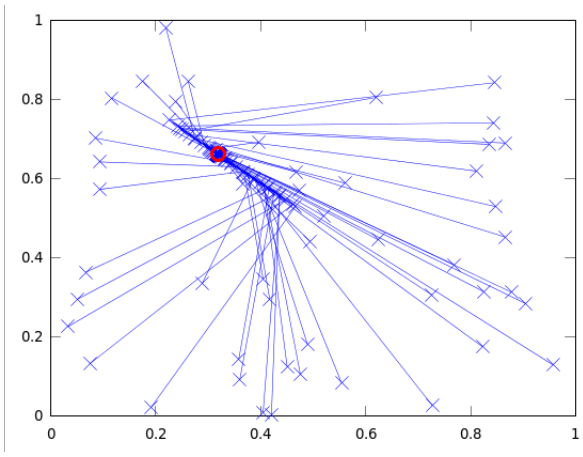

Figure 6: An example of a randomly generated attractor state encoded into an FCM; convergence to the state from randomly selected starting points shown

5.3. Comparison with Hebbian Learning

Let us now present a preliminary comparison with approaches based on Hebbian learning – in particular a cooperation with results achieved using AHL and NHL as reported in Papageorgiou, Stylios, and Groumpos (2006).

The authors present a FCM constructed by experts. Furthermore, the experts specify regions of acceptable values for selected concepts. If the experts specify that the activation value of concept C_j should fall in interval $[T_j^{min}, T_j^{max}]$, the learning goal is given by the following (Papageorgiou, Stylios, and Groumpos 2006):

$$T_j = \frac{T_j^{min} + T_j^{max}}{2} \, . \tag{16}$$

The results compared are those for the first case study. They follow in Table 3. In each case the steady-state difference from the target activation values (defined in terms of equation (16)) are computed, and the mean square error is reported.

The results reported for the BAS approach were achieved using the learning rate of 0.2, BPTT was started from 100 random initial positions, the number of steps after which the FCM should reach the attractor state was set to 5, and the additional number of steps for which was trained was set to 45. In this case, no gradient explosion problem occurred – having a weight matrix specified by experts clearly gives learning some advantage.

There are two columns in the table for the NHL method. This is because Papageorgiou, Stylios, and Groumpos (2006) report two results. The first one is for the steady-state starting from the initial weight matrix, and using NHL to modify weights, and control the process. NHL W refers to the steady-state reached when using the weight matrix learned using NHL without doing further learning. (There is no considerable difference between the errors though.)

Table 3: Comparison of results using AHL, NHL, and BPTT-based attractor setting (BAS)

	AHL	NHL	NHL W	BAS
MSE	3.394 **E-04**	1.312 **E-3**	1.405 **E-3**	1.322 **E-12**

It should be noted, that the comparison presented here is not in any sense definitive. The reason for this is that this paper does as of yet provide no tests for encoding attractors for which only the activation values of several concepts are specified. In this case we work around the problem by arbitrarily fixing the values of the other concepts to 0.5. It should, however, be possible to instead use backpropagation to propagate errors to the concepts with unspecified values. This issue remains for future investigation.

Also, we do not fix values of any of the weights – not even of the zero ones. All weights are allowed to change in this version of the algorithm. This may provide our approach with an unfair advantage in this case. This will also require further investigation, but existing research indicates that backpropagation can indeed work well even in cases when some or most of the weights are fixed – in fact this property is already being exploited by some types of ANNs, such as the echo state network.

We conclude that the error margin of the approach seems large enough to make the approach competitive even if the aforementioned restrictions were to hurt its performance to a certain extent.

6. FURTHER WORK

As a part of future work, the algorithm should be extended with the state-of-the-art learning methods based on Hessian-free optimization. This should effectively prevent the vanishing/exploding gradient problem, and make learning faster in general.

The algorithm should be tested with real models. It should be ascertained how well it can be used in scenarios where the training is only allowed to change some weights, while other remain fixed.

Backpropagation could be used to learn attractors in which only values of several concepts are specified. It should be investigated how such approach to work, and whether the values of the other concepts would stabilize as well, and also whether they would always stabilize at the same (though unspecified) values.

7. CONCLUSION

It has been shown in the past that the similarity between artificial neural networks allow for transfer of learning methods from one to another. Most notable among the related approaches have been those based on Hebbian learning.

In this paper we have contributed a new method for encoding attractors into fuzzy cognitive maps. The approach, as shown, is based on the notion of backpropagation through time known from the theory of recurrent neural networks.

The application of the method has been presented and accompanied with simulation results. It has been shown that the method perform favourably, and preliminary comparisons show that its accuracy surpasses that of the approaches based on Hebbian learning. However, further work is necessary at this point.

It has been shown that increasing the number of stabilising steps has a positive influence on accuracy, but also increases the chance of gradient explosions. A similar relationship has been shown for the number of initial point from which BPTT learning is done.

Further study of some aspects of the approach is still to be carried out. It remains to be shown how well the method will be able to perform when target activation values are only known for certain concepts. It is also of much interest to investigate how well the method will perform when some of the weights will be fixed to their initial values. Also, some learning situations may require the introduction of more powerful optimization methods such as Hessian-free optimization.

REFERENCES

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H., 2007. Greedy layer-wise training of deep networks. *Advances in neural information processing systems* 19.

Dickerson, J.A., and Kosko, B., 1993. Virtual worlds as fuzzy cognitive maps. *Virtual Reality Annual International Symposium*, 471–477.

Groumpos, P.P., and Stylios, C.D., 2000. Modelling supervisory control systems using fuzzy cognitive maps. *Chaos, Solitons and Fractals* 11 (1), 329–336.

Groumpos, P.P., 2010. Fuzzy cognitive maps: basic theories and their application to complex systems. In: Glykas, M., ed. *Fuzzy Cognitive Maps*. Berlin: Springer-Verlag, 1–22.

Hinton, G.E., and Salakhutdinov, R.R., 2006. Reducing the dimensionality of data with neural networks. *Science* 313 (5786), 504–507.

Huerga, A.V., 2002. A balanced differential learning algorithm in fuzzy cognitive maps. *Proceedings of the Sixteenth International Workshop on Qualitative Reasoning*, 10–12.

Kottas, T.L., Boutalis, Y.S., and Christodoulou, M.A., 2007. Fuzzy cognitive network: A general framework. *Intelligent Decision Technologies* 1 (4), 183–196.

Krose, B., and Smagt, P.V.D., 1996. *An Introduction to Neural Networks*. University of Amsterdam. Available from: http://www.cs.unibo.it/babaoglu/ courses/ cas/resources/tutorials/Neural_Nets.pdf [Accessed 10 May 2013].

Martens, J., 2010. Deep learning via Hessian-free optimization. *Proceedings of the 27th International Conference on Machine Learning*.

Martens, J., and Sutskever, I., 2011. Learning recurrent neural networks with Hessian-free optimization. *Proceedings of the 28th International Conference on Machine* Learning 46.

Papageorgiou, E.I., Stylios, C.D., and Groumpos, P.P., 2003. Fuzzy cognitive map learning based on nonlinear Hebbian rule. *AI 2003: Advances in Artificial Intelligence*, 256–268.

Papageorgiou, E.I., and Groumpos, P.P., 2004a. A new hybrid method using evolutionary algorithms to train fuzzy cognitive maps. *Applied Soft Computing* 5, 409–431.

Papageorgiou, E.I., Stylios, C.D., and Groumpos, P.P., 2004b. Active Hebbian learning algorithm to train fuzzy cognitive maps. *International Journal of Approximate Reasoning,* 37 (3), 219–249.

Papageorgiou, E.I., and Groumpos, P.P., 2005a. A weight adaptation method for fuzzy cognitive map learning. *Soft Computing,* 9 (11), 846–857.

Papageorgiou, E.I., Parsopoulos , K. E., Stylios, C.S., Groumpos, P.P., and Vrahatis, M.N., 2005b. Fuzzy Cognitive Maps Learning Using Particle Swarm Optimization. *Journal of Intelligent Information Systems* 25 (1), 95–121.

Papageorgiou, E.I., Stylios, C.D., and Groumpos, P.P., 2006. Unsupervised learning techniques for fine-tuning fuzzy cognitive map causal links. *International Journal of Human-Computer Studies* 64 (8), 727–743.

Parsopoulos, K.E., Papageorgiou, E.I., Groumpos, P.P., and Vrahatis, M.N., 2003. A first study of fuzzy cognitive maps learning using particle swarm optimization. *The 2003 Congress on Evolutionary Computation, 2003. CEC'03*, 2, 1440–1447.

Rojas, R., 1996. *Neural networks: a systematic introduction*. Berlin: Springer-Verlag.

Stach, W., Kurgan, L., and Pedrycz. W., 2005a. A survey of fuzzy cognitive map learning methods. *Issues in soft computing: theory and applications*, 71–84.

Stach, W., Kurgan, L., Pedrycz., W., and Reformat, M., 2005b. Genetic learning of fuzzy cognitive maps. *Fuzzy Sets and Systems*, 153 (3), 371–401.

Stach, W., Kurgan, L., and Pedrycz, W., 2008. Data-driven nonlinear Hebbian learning method for fuzzy cognitive maps. *IEEE International Conference on Fuzzy Systems. FUZZ-IEEE 2008 (IEEE World Congress on Computational Intelligence)*, 1975–1981.

Stylios, C.D., Christova, N., and Groumpos, P.P., 2002. A hierarchical modeling technique of industrial plants using multimodel approach. *Proceeding of 10th IEEE Mediterranean conference on Control and Automation, Lisbon, Portugal*.

Sutskever, I., Martens, J., Hinton, G., 2011. Generating Text with Recurrent Neural Networks. *Proceedings of the 28th International Conference on Machine Learning*.

Wang, H., and Wang, L., 2013. Application of Improved Fuzzy Cognitive Map Based on Fuzzy Neural Network in Intrusion Detection. *Journal of Information & Computational Science* 10 (1), 271–278.

Werbos, P.J., 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* 78 (10), 1550–1560.