Int. J. Appl. Math. Comput. Sci., 2021, Vol. 31, No. 4, 577–586
DOI: 10.34768/amcs-2021-0039

A MODIFIED PARTICLE SWARM OPTIMIZATION PROCEDURE FOR TRIGGERING FUZZY FLIP–FLOP NEURAL NETWORKS

PIOTR A. KOWALSKI [a,b,*], TOMASZ SŁOCZYŃSKI [a]

[a] Faculty of Physics and Applied Computer Science
AGH University of Science and Technology
al. A. Mickiewicza 30, 30-059 Cracow, Poland
e-mail: pkowal@agh.edu.pl, sloczynski.tomasz@gmail.com

[b] Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland
e-mail: pakowal@ibspan.waw.pl

The aim of the presented study is to investigate the application of an optimization algorithm based on swarm intelligence to the configuration of a fuzzy flip-flop neural network. Research on solving this problem consists of the following stages. The first one is to analyze the impact of the basic internal parameters of the neural network and the particle swarm optimization (PSO) algorithm. Subsequently, some modifications to the PSO algorithm are investigated. Approximations of trigonometric functions are then adopted as the main task to be performed by the neural network. As a result of the numerical verification of the problem, a set of rules are developed that can be helpful in constructing a fuzzy flip-flop type neural network. The obtained results of the computations significantly simplify the structure of the neural network in relation to similar conditions known from the literature.

Keywords: fuzzy neural network, fuzzy flip-flop neuron, particle swarm optimization, training procedure, regression.

1. Introduction

Modern technologies have enabled the near-instant ability to communicate huge amounts of data. This was possible because of computer algorithms. These have become the basic method of data processing and modelling. Advances in these have resulted in what is termed "neural networks".

Neural networks, the idea of which is to reflect the way the human brain works, dominate the market of machine learning solutions as applied wherever there is a need to predict future behavior based on previous decisions or to classify objects based on known attributes. Thus, they have application in the military, business, computer gaming or medicine. The effectiveness of neural networks depends on the provided appropriate training patterns and on a correctly conducted training process.

The subject of the research described in this article is the implementation and analysis of selected structures of recursive fuzzy neural networks that are built upon the application of a metaheuristic training algorithm. The neural networks selected for the study in this paper are based on J-K flip-flop sequential logic. Recursion is realized therein through the feedback between inputs J and K. The metaheuristic method adopted for the training process is particle swarm optimization.

Fuzzy flip-flop neural networks (FFFNNs) were first described by Professor Hirota and his scientific team in 1989 (Hirota and Ozawa, 1989; Ozawa *et al.*, 1996). Beyond purely theoretical debates, for many years, these networks were not among the most discussed issues in the neural network community. This was largely due to the absence of a convenient teaching algorithm, the causes of which were the lack of fast and efficient computers and imperfection in training algorithms.

The uniqueness and diversity of the activation functions condition the inability to effectively apply classical training methods. The main obstacle to supervised training is the absence of an analytical derivative operator that builds upon the information on

*Corresponding author

the current value of the activation function—as it is the case with the use of neural functions, i.e., linear, sigmoid or hyperbolic tangent. In the literature there are only two known effective learning algorithms for the FFFNN shortly described above. One is the bacterial memetic algorithm (Gal *et al.*, 2008) introduced by the group of Professor Laszlo T. Koczy (Lovassy *et al.*, 2008b). The other one is the training algorithm based upon an evolutionary strategy (Kowalski, 2013). We propose in this paper the application of a third algorithm based on the PSO optimization procedure. On the other hand, the well-known gradient algorithm called the "back-propagation training procedure" (Rutkowski, 2008; Lillicrap *et al.*, 2016) was intended in 2009 to be used to train FFFNNs, but turned out to be characterized by very poor results and weak convergence rates (Gál *et al.*, 2009).

This article is composed as follows. Section 2 introduces the structures of the fuzzy flip-flop (FFF) neuron, as well as the FFFNN. In Section 3, the PSO procedure for the FFFNN training process is applied. Section 4 sets out a description of the extensions of PSO models. In Section 5, the dependencies and the factors influencing the quality of FFFNN training are numerically analyzed. In Section 6, the obtained results are discussed and conclusions are drawn.

2. Fuzzy flip-flop neural network

The J-K flip-flop has been adapted for implementation within a neural network by transforming the binary version into a fuzzy flip-flop. This is achievable by using fuzzy logic operators such as t-norm and co-norm to replace the previously used logical AND, OR and NOT operators (Ozawa *et al.*, 1991). The first unified equation of the J-K flip-flop using fuzzy operators were proposed by Hirota and Pedrycz (1993) and has the form

$$Q_{\text{out}}(t+1) = (Ji\overline{K})u(JiQ)u(\overline{J}u\overline{K}), \tag{1}$$

where J and K constitute J-K flip-flop input signals and Q_{out} is its output signal.

In the fuzzy domain, the functions i and u are the t-norm and co-norm, respectively, while the negation operators are defined as $\overline{K} = 1 - K$ and $J, K, Q \in [0, 1]$. Formula (1) makes the flip-flop response dependent on the inputs J, K and the current state of Q. The discrete time variable has been omitted here for readability. This form is used successively in the rest of the article. The proposed model of the neuron uses the feedback between signals Q and K (see Fig. 1). As a result, the J-K flip-flop is able to create a recursive neuron with the degree of recursion r, where r is the number of recursive loops. In addition, for the degree of recursion $r = 1$, an initial value of Q_0 must be set. It should be mentioned that the input J must be checked and limited to the interval $[0, 1]$. Thus, taking into account the feedback, the output from the FFF neuron can be determined as follows:

$$Q_{\text{out}} = (JuQ)i(JuQ)i(Qu(1-Q)), \tag{2}$$

In equation (2), signal J is the sum of the products of bias and inputs with weight values. Next this value is transformed by the *satlin* activation function

$$J = S\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{3}$$

where

$$S(x) = \begin{cases} 0 & \text{for } x < 0, \\ x & \text{for } x \in [0, 1], \\ 1 & \text{for } x > 0. \end{cases} \tag{4}$$

Table 1 presents the selected t-norm and co-norm, thanks to which the output Q from the J-K flip-flop (Eqn. (1)) can be treated as a quasi-sigmoid neuron activation function. The additional parameters listed in Table 1, i.e., α, γ, s and w, belong to the interval $(0, \infty)$.

Based on the FFF-type neuron described above, an artificial neural network can be created. For interpolation-type problems, the FFF network structure has the form shown in Fig. 2. The neural network topology consists of four layers. As in all models, there are input and output layers. Between them, the so-called "hidden layers" are placed. As shown, each hidden layer has a certain fixed number of neurons. In subsequent layers, these neurons are connected to the previous layer in a "one-to-all" relation. This is a typical connection structure for shallow (Chang, 2015; Bodyanskiy and Tyshchenko, 2019) or fully connected neural networks (Basha *et al.*, 2020). In this type of neural network, the hidden layer neurons are of the FFF type, while in the output layer, classical neurons are found exclusively and have a linear activation function. In the case under consideration, each of the hidden and output neurons has a number of parameters called "weights", as shown in Fig. 1 and Eqn. (3). However, in the case of using the FFFNN for the classification problem (Siminski, 2021), it is advisable to replace the linear neurons in the output layer with FFF neurons.

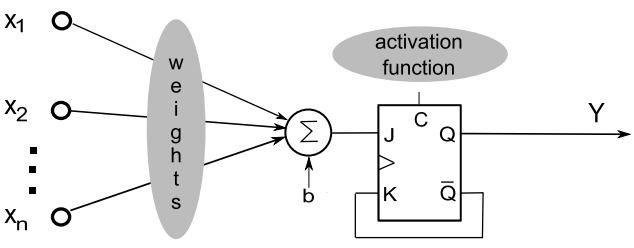

Fig. 1. Fuzzy flip-flop neuron.

Table 1. Selected triangular t-norms and co-norms applied in the FFFNN.

Name	norm(x,y)	co-norm(x,y)
Min-max	$\min(x, y)$	$\max(x, y)$
Algebraic	xy	$x + y - xy$
Łukasiewicz	$\max(0, x + y - 1)$	$\min(x + y, 1)$
Trigonometric	$\frac{2}{\pi}\arcsin(\sin(x\frac{2}{\pi})\sin(y\frac{2}{\pi}))$	$\frac{2}{\pi}\arccos(\cos(x\frac{2}{\pi})\cos(y\frac{2}{\pi}))$
Dombi	$\frac{1}{1+[(1/x-1)^a+(1/b-1)^a]^{1/a}}$	$\frac{1}{1+[(1/x-1)^{-a}+(1/b-1)^{-a}]^{-1/a}}$
Hamacher	$\frac{xy}{\gamma+(1-\gamma)(x+y-xy)}$	$\frac{x+y-(2-\gamma)xy}{1-(1-\gamma)xy}$
Frank	$\log_s(1+\frac{(s^x-1)(s^y-1)}{s-1})$	$1-\log_s(1+\frac{(s^{1-x}-1)(s^{1-y}-1)}{s-1})$
Yager	$1-\min[1,((1-x)^w+(1-y)^w)^{1/w}]$	$\min[1,(x^w+y^w)^{1/w}]$

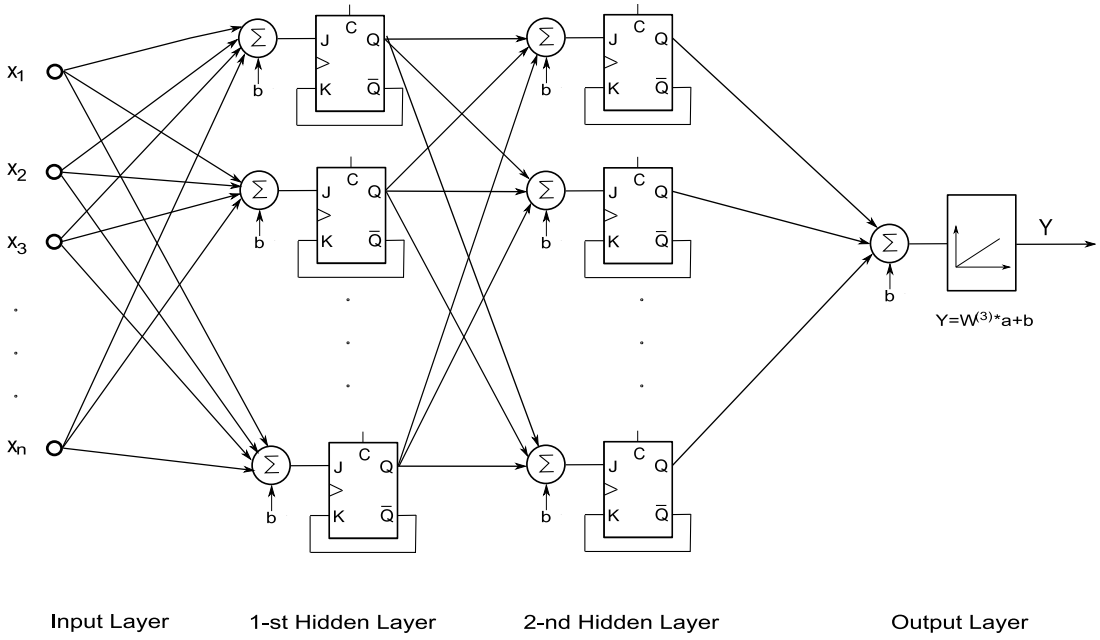

Fig. 2. Topology of the FFFNN.

It should be noted that this type of network and fuzzy logic operators have been implemented in the form of hardware (Lovassy *et al.*, 2010; Zavala *et al.*, 2009; Gniewek and Kluska, 2004).

3. Optimization of the FFFNN with the particle swarm algorithm

The PSO algorithm was first proposed by Kennedy and Eberhart (1995). PSO is a population-based metaheuristic optimization algorithm based on swarm intelligence. The inspiration for its development is the coordinated behavior observed in the nature of a swarm of bees, a flock or swarm of birds and a swarm of fish while searching for food (Talbi, 2009). The assumption of the algorithm is that the swarm consists of S particles searching the N-dimensional space of solutions. Each x particle is an N-dimensional vector that proposes a solution to a given problem (Łukasik and Kowalski, 2014; Tsoulos *et al.*, 2021).

Each swarm element has the following attributes: the vector x defines the particle position; the vector v, referred to as "particle velocity", determines the direction and value of the position change; the value of the $swarmCost$ adjustment function determines the quality of a given solution; the vector x_{LB} is the best (so far) preserved position of a given particle and source of information about neighbors. In this research, the global topology (Kennedy and Eberhart, 1995) is applied, so that each particle has information about every other particle in the swarm. This assumption gives access to the globally best particle of x_{GB}. The last two points are an emanation of the swarm's intelligence and the social behavior of the particles towards each other. In this approach, the $swarmCost$ adjustment function in terms of FFFNN testing MSE errors is considered.

In each time step t, the particle in the swarm moves

in the solution space and updates its position according to

$$x_i(t+1) = x_i(t) + v(t+1) \tag{5}$$

for $i = 1, \ldots, N$, where $x_i(t)$ is the item in the previous step, $v(t+1)$ denotes the speed updated according to

$$\begin{aligned} v_i(t+1) = \omega v_i(t) + c_1 r(t)(x_{GB} - x_i) \\ + c_2 r_2(t)(x_{LB} - x_i(t)), \end{aligned} \tag{6}$$

where ω is the inertia coefficient that determines the effect of the speed in the previous step on the current one, c_1 denotes the coefficient determining the pursuit of the global best particle (position), c_2 is a coefficient determining the pursuit of the best local particle (position), and finally, $r_1(t)$ and $r_2(t)$ are vectors of random values from the interval $[0, 1]$. Figure 3 presents a block diagram of the basic form of the PSO algorithm, which is the starting point for considerations in the sequel. The figure shows that the additional variable of the algorithm is $epochs$—it corresponds to the number of the currently processed epochs—and $maxEpochs$ which delimits the maximum number of epochs in the PSO algorithm. Due to the necessity to adapt PSO to the domain of an artificial neural network, the term "iteration" has been replaced with the term "epoch", which is characteristically employed in a neural networks domain.

The adaptation of PSO to neural network training consists in treating each particle as a set of all the neural network's weights and biases. In order to calculate the cost (fit) of a given solution, the position vector is substituted into the neural network as a set of weights and biases. The training data (the period of a single epoch) are presented in such a configured network, and then, on the basis of the obtained results, the training error (E_{train}) is calculated. The error obtained is the cost $swarmCost_i$ of the given particle in the PSO (Rakitianskaia and Engelbrecht, 2015) algorithm. The effect of finished training is the x_{GB} particle, which has a set of weights and biases belonging to a neural network characterized by generating the smallest training error.

To clarify the facts, it should be pointed out once again that, within the training algorithm, S neural networks are considered in parallel. Each one is represented as a set of weights and biases encoded in the position vector x_i of the i-th particle. At each stage, i.e., the training iteration, the particle labeled x_{GB} contains the set of weights and biases of the neural network characterized by the smallest error obtained on the training data. On the other hand, each particle remembers its best solution, i.e., the best neural network that a given particle represented. This solution is marked as x_{LB}.

In the case of using the PSO algorithm to train the FFF type neural network, it is necessary to determine

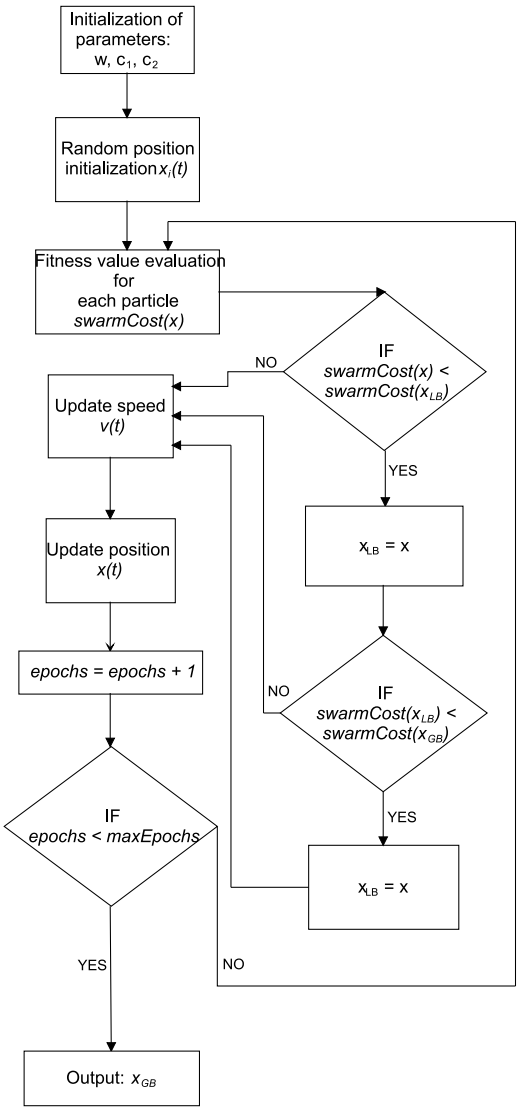

Fig. 3. Flowchart of the PSO procedure.

which internal parameters of both the neural network and the training algorithm are beneficial. The first include the initial value of the Q_0 flip-flop, the type of the triangular norm and co-norm operators, the number of r recursion loops and the network topology. In the second group of parameters related to the PSO algorithm, ω, c_1 and c_2 should be considered. The above will be the subject of the research part of this paper.

4. Extension of the basic PSO algorithm

In this section, modifications of the training algorithm will be presented. The intent of these is to improve the performance of the PSO procedure. As part of the proposed modifications to the training procedure, the PSO algorithm is enriched with regularization control procedures, minimum speed limitations, as well as

geometric determinations of the swarm center and grouping of particles inside the swarm.

4.1. Regularization control. Here regularization constitutes a method aimed at improving the network's problem-solving ability by modifying the cost function and thus forcing weights to minimized it.

Mostly, it is used in the case of overfitting. In this article, it is proposed to apply the regularization to the PSO algorithm by modifying the method of updating items by means of the following formula, where each particle is endowed with a new attribute λ:

$$x_i(t+1) = x_i(t) + v(t+1) - \lambda(t)x_i(t). \tag{7}$$

The coefficient λ in (7) is updated as follows:

$$\lambda_i(t+1) = \begin{cases} \lambda_i(t) + 5 \times 10^{-3} & \text{for } E_i(t+1) < E_i(t), \\ \lambda_i(t) - 5 \times 10^{-3} & \text{for } E_i(t+1) > E_i(t), \end{cases} \tag{8}$$

where $E_i(t+1)$ and $E_i(t)$ are the particle cost values $swarmCost$ at simulation times $t+1$ and t, respectively. In this investigation the initial value $\lambda(0) = 5 \times 10^{-6}$ is applied.

4.2. Minimum speed limitation. In this subsection, the minimum speed limitation of $v_{\min}$ is addressed thanks to an idea taken from Pu *et al.* (2007). When the velocity vector component is lower than the limiting value of $|v_{\min}|$, this component should be changed to the value $\overline{v}$ that is derived from applying

$$\overline{v} = \frac{1}{SN} \sum_{i=1}^{S} \sum_{j=1}^{N} |v_{i,j}|, \tag{9}$$

where S is the number of particles in the swarm and N is the number corresponding to the size of a single particle. Here $|v_{\min}|$ is the average velocity module calculated from all particles in the swarm.

4.3. Geometric center of the swarm. Chen (2008) proposed to extend the standard PSO algorithm to converge to the geometric center of the swarm x_{GC}. In such a case, the velocity updating equation (6) is enriched with an additional part determining the trust in the center of the particle swarm

$$\begin{aligned} v_i(t+1) = {} & \omega v_i(t) + c_1 r(t)(x_{GB}(t) - x_i) \\ & + c_2 r_2(t)(x_{LB}(t) - x_i(t)) \\ & + c_3 r_3(t)(x_{GC}(t) - x_i(t)). \end{aligned} \tag{10}$$

Here, c_3 is the coefficient of striving for the geometric center of the swarm, $r_3 \in [0, 1]$ denotes a random value, and the geometric center of the swarm can be calculated as follows:

$$x_{GC} = \frac{1}{S} \sum_{i=1}^{S} x_{LB}(t). \tag{11}$$

Due to the necessity of additional computational effort, the procedure for determining the geometric center of a swarm is updated every specified number of epochs T.

4.4. Grouping of particles inside a swarm. Chen (2010) recommended to divide the particles in the swarm into three groups. Moreover, their positions are to be updated differently depending on the assigned category. The parameters in (12)–(14) are identical to those in (5)–(6). The particle positions with the best cost value are updated according to

$$x_i(t+1) = \omega x_i(t) + c_1 r(t)(x_{GB}(t) - x_i). \tag{12}$$

The particle positions with an average cost are updated according to

$$\begin{aligned} x_i(t+1) = {} & \omega x_i(t) + c_1 r(t)(x_{GB}(t) - x_i) \\ & + c_2 r_2(t)(x_{LB}(t) - x_i(t)). \end{aligned} \tag{13}$$

Finally, the positions of the particles with the worst cost value change as

$$x_i(t+1) = \omega x_i(t) + c_2 r_2(t)(x_{LB}(t) - x_i(t)). \tag{14}$$

The assignment to particular groups is carried out as follows. Within each epoch, the values of the worst-matched particle ($swarm_{WC}$) and the average cost in the swarm ($swarm_{AC}$) are determined. The mean (avg_1) between the globally best particle $swarm_{BC}$ and $swarm_{AC}$ is then derived. After this, the average (avg_2) between $swarm_{AC}$ and $swarm_{WC}$ is calculated. Based on the parameters generated in this way, the respective ranges are defined as follows:

- the best particles are those belonging to the compartment $[swarm_{BC}, avg_1]$,
- the mean particles are in the range $[avg_1, avg_2]$,
- the worst particles are those in the range $[avg_2, swarm_{WC}]$.

5. Numerical verification of the training procedure

This section is devoted to the study of the impact on the quality of the FFFNN operation of individual parameters of both the neural network itself and the PSO training algorithm. The idea behind this is to extract the attributes and ascertain how a single change (for example, in the Q_0 parameter) affects the solution of the task addressed by

the neural network. The quality of the solution in terms of both training and testing MSE errors is considered. For this purpose, an FFFNN is used to approximate the function of the form $y(x) = 3\sin(-2x) + 1$.

The basic PSO algorithm described in Section 3 is used to solve the given task and to determine the comparative level. The foundation configuration of the network and the parameters of the entire algorithm are decided on the basis of preliminary numerical pilot tests and settings known from the literature. The neural network topology is in the form 1-4-1, i.e., one neuron in the input layer, four FFF neurons in the hidden layer and one linear neuron in the layer output. Additionally, the vector of weights is initialized randomly from the interval $(0, 1)$ using a pseudorandom number generator with uniform distribution, while the velocity vector $v_i(t)$ is initialized with zero. In the J-K flip-flop, a pair of trigonometric norms with the degree of recursion $r = 2$ is used. What is more, the initial value of $Q_0 = 0.25$ is adopted. During training, the following parameter values are employed: the number of epochs = 400, the size of the swarm $S = 20$, the coefficient of inertia $\omega = 0.7298$, the coefficient of striving for the best globally particle $c_1 = 1.496$ and the coefficient of striving for the best position $c_2 = 1.496$. The above parameters form the so-called basic configuration of the main FFFNN training procedure, to which the proposed changes to the algorithm and its parameters will refer.

The parameters ω, c_1 and c_2 are derived from the work of Eberhart and Shi (2000), where the above values are suggested as optimal for achieving correct convergence properties. The data set consists of 100 points where 70% and 30% of all cases belong to the training and testing subsets, respectively. All tables with numerical verification show the results obtained for the normalized data in the following column order: MSE training error (E_{train}) and testing MSE error (E_{test}). The use of such a data set is caused by mass tests and a large number of results that can be easily lost thanks to this solution. In the case of large data sizes, it is possible to take advantage of large supercomputers operating in cloud structures.

In the basic version of the configuration, the following error values are obtained for the training sample: $E_{\text{train}} = 0.0043$ and the MSE test error $E_{\text{test}} = 0.0107$.

Initial value of Q_0. In this section, the influence of the initial value of Q_0 on the correct operation of the neural network is tested. This attribute is the only attribute that is changed in relation to the basic version of the algorithm. In the basic configuration, $Q_0 = 0.25$ and the obtained results are summarized in Table 2.

In the case of the tested data, in the basic case, the best values of training and testing errors and the lowest values of saturation (described by Rakitianskaia and Engelbrecht (2015)) are obtained. It can be seen that for the extreme cases of the signal Q_0, the error values increase significantly.

Type of fuzzy logic operator. In this section, the effect of fuzzy logic operators on approximation quality is tested. In the basic configuration, we used the trigonometric norm and co-norm and the obtained results are presented in Table 3. In the case of applying the Dombi and Yaeger norms, the value of the equation parameter is additionally selected. In (Gál *et al.*, 2010), it is suggested to select $\alpha = 5$ for Dombi and $w = 2.13$ for Yaeger.

In the case under consideration, Yager's and trigonometric norms proved to be the best. However, at this point it is worth emphasizing that the Yager norm is strongly dependent on the internal parameter w. Thus, a kind of compromise is to use an operator without an internal parameter, which is the trigonometric norm.

Degree of recursion. In this part of the paper, the effect of the number of recursion loops on the FFF network quality is tested. In the basic configuration, the degree of recursion is $r = 2$ and the obtained results are presented in Table 4. When analyzing the results listed in Table 4, in the case of training data, the parameter r turns out to be the most favorable for even values. In contrast, for the testing data, the best results were obtained for $r = 4$ and $r = 6$.

Table 2. Effect of parameter Q_0 on the FFF-type neuron.

	E_{train}	E_{test}
$Q_0 = 0.0$	0.0319	0.0346
$Q_0 = 0.25$	0.0043	0.0107
$Q_0 = 0.5$	0.0113	0.0124
$Q_0 = 0.75$	0.0214	0.0214

Table 3. Influence of the adopted triangular norm in the FFF neuron.

	E_{train}	E_{test}
Trigonometric	0.0043	0.0107
Łukasiewicz	0.0373	0.0381
Dombi ($\alpha = 5$)	0.0319	0.0276
Algebraic	0.0245	0.0267
Yaeger ($w = 2.13$)	0.0174	0.0180

Table 4. Degree of recursion r in the FFFNN.

	E_{train}	E_{test}
$r = 2$	0.0043	0.0107
$r = 3$	0.0281	0.0384
$r = 4$	0.006	0.0069
$r = 5$	0.0742	0.0741
$r = 6$	0.008	0.0068

The remaining cases considered here are characterized by an error larger by an order of magnitude.

Particle velocity maximum value limitation. Introducing a speed limit $v_{\max}$ is one idea to stop particles from exploring the solution space too dynamically (Rakitianskaia and Engelbrecht, 2015). The dominant view in most of the works is that velocities are initialized with zero values. However, Carvalho and Ludermir (2006) suggested to use for small interval initialization when testing speed limits. The initial values of $[-v_{\max}, v_{\max}] = [-0.1, 0.1]$ and a smaller range of the interval are considered to be ideal for this purpose.

In the basic case, there is no initialization and no speed limits. The limitation is carried out according to

$$v_i(t) = \begin{cases} v_{\max} & \text{for } v_i(t) \geq v_{\max}, \\ -v_{\max} & \text{for } v_i(t) \leq -v_{\max}, \\ v_i(t) & \text{for } -v_{\max} < v_i(t) < v_{\max}. \end{cases} \quad (15)$$

Table 5 presents the results of the simulation of a neural network operation utilizing the speed limitation version of the PSO training algorithm that incorporates a constraint. Based on the results, it can be stated that limiting the maximum speed to the range $[-1, 1]$ allows reducing the MSE error for the content set to 0.0052. It is worth emphasizing here that increasing the value of the limitation is more advisable than lowering this parameter with respect to the value of 1.

Influence of initialization and position limitation of individual particles. Limiting the range of initialization of weights (similarly to positioning the particles by applying the PSO algorithms) is the simplest and arithmetically most justified move for minimizing MSE error value. It is logical that small weight values imply low net values and, consequently, the potential to enter the saturation regions of the activation function in the hidden layers is lowered. Additionally, as there is a restriction on the solution search space, the weights will be kept at the appropriate level. The introduction of a position restriction is also proposed by Rakitianskaia and Engelbrecht (2015). In this part of the research, two scenarios are tested: the impact of initialization itself and initialization combined with the limitation of the value of $x_i(t)$. We enter the limitations of the $X_{\max}$ position in the same way as in the case of speed (15), with one significant difference, if the particle is outside the area designated for the scales, its velocity is changed to the opposite and then its position is limited.

In the base case, the weights are randomly initialized from the range $[0, 1]$ and are not limited during the algorithm's operation. The impact of initialization and the extended case with position limitation are presented in Table 6.

For the initialization cases considered, the most favorable ranges are $|X_{\max}| = 2$ and the base configuration. In both cases, the testing error is the smallest and amounts to 0.0127 and 0.0107, respectively. The above conclusion is not, however, reflected in the results of the training sample. In this case, the best results for training the neural network are obtained for the base example and the constraints $|X_{\max}| = 0.1$. The last two columns present the results of the constraint of the particle position, which shows that again the interval $|X_{\max}| = 2$ and the base configuration seem to be the least convenient configuration.

Introducing a regularization procedure to the PSO algorithm. In the next step of our research, a regularization component is introduced to the training procedure. This factor is based on the optimization algorithm. The above is to prevent the neural network from being overfitted with the training data. During the research, the base case was compared with the instance in which the regularization is introduced. In the case of the regularization algorithm considered, the neural network recorded the following error values: $E_{\text{train}} = 0.0279$ and $E_{\text{test}} = 0.0285$. In this situation, no improvement in the operation of the neural network is observed, but in separate observations it is seen that the FFF neurons are much less likely to become saturated (Rakitianskaia and Engelbrecht, 2015).

Table 5. Influence of speed limitation and initialization on the performance quality of the FFFNN.

	E_{train}	E_{test}
Base case	0.0043	0.0107
$\lvert v_{\max}\rvert = 10$	0.0236	0.0246
$\lvert v_{\max}\rvert = 2$	0.0226	0.0192
$\lvert v_{\max}\rvert = 1$	0.0054	0.0052
$\lvert v_{\max}\rvert = 0.5$	0.0460	0.0490
$\lvert v_{\max}\rvert = 0.1$	0.0020	0.0030
$\lvert v_{\max}\rvert = 0.01$	0.0843	0.0821
$\lvert v_{\max}\rvert = 0.001$	0.3033	0.3070

Table 6. Impact of position limitation initialization on the FFFNN performance quality.

	Initialization		Position limitation	
	E_{train}	E_{test}	E_{train}	E_{test}
Base case	0.0043	0.0107	0.0043	0.0107
$\lvert X_{\max}\rvert = 10$	0.0844	0.0859	0.0950	0.1026
$\lvert X_{\max}\rvert = 2$	0.0139	0.0127	0.0913	0.0863
$\lvert X_{\max}\rvert = 1$	0.0107	0.0176	0.0815	0.7140
$\lvert X_{\max}\rvert = 0.1$	0.0096	0.0140	0.2545	0.1921

PSO procedure with minimum speed limitation. This part of the article presents the effect of modifying the PSO optimization algorithm by introducing a minimum speed limit $v_{\min}$. This is tested as two variants. The first (A1) is consistent with the description of the algorithm provided in subsection 4.2. The second (A2) consists in substituting the minimum value when the minimum speed is exceeded.

The base case does not include such limitations. The results of both scenarios (A1 and A2) are presented in Table 7.

The algorithm of minimum speed (A2) achieves better results than the A1 procedure in terms of training and testing errors. Still, an acceptable impact is obtained using the mean velocity substitution for the range $|v_{\min}| = 0.001$. In this case, both indicators of the A2 algorithm are similar to the base case. The speed lower limit can be useful if the network is stuck at the local minimum, and by switching to medium speed, there is a chance it will come out of the local extreme.

PSO procedure with a geometric center of a swarm. The next stage of the research is to propose extending the standard PSO algorithm to the geometric center of the swarm through convergence. In this simulation, the time interval T measured in epochs is tested so as to discover how often it is advisable to use this modification of the PSO algorithm. In this case, the parameters (c_1, c_2, c_3) are set based on recommendations of Chen (2008).

On the basis of Table 8 for the parameters $c_1 = c_2 = c_3 = 1.496$, introducing this additional procedure deteriorates the results achieved. The results obtained in this way do not correspond in any way to those obtained in other numerical experiments and in the base example. Moreover, the algorithm did not converge in any case. However, in the suggested configuration of $c_1 = c_2 = c_3 = 1.333$, good results are obtained, especially for the cases $T = 1$ and $T = 2$. For $T = 2$, the performance of the neural network is better for the test data than for the base case.

PSO procedure with grouping particles inside a swarm. The last modification presented in this article is a procedure for changing the position update that is dependent upon grouping the investigated particles within one of three categories. The results obtained on the basis of the algorithm presented in the previous part of the article showed no improvement compared the base example. For the approximation example, the neural network recorded the following error values: $E_{\text{train}} = 0.0824$ and $E_{\text{test}} = 0.1083$. However, it should be emphasized that despite the poor end result, saturation of FFF neurons was observed much less frequently during the training process than in the base example.

6. Conclusions

The results of the analysis from the previous chapters allow for an idea of how many factors depend on the correct configuration of the FFF type neural network and its training algorithm. Over all, it can be said that there must be an appropriate balance between the selection of the parameters of J-K neurons and the PSO algorithm to achieve satisfactory network matching results that incorporate the lowest possible training and testing errors.

The configuration of best neural network is based on the appropriate selection of the initial value of Q_0, the number r of recursion loops and the type of fuzzy logic operation. A base value of $Q_0 = 0.25$ seems to be a preferred starting value for the data under consideration. In the case of choosing a basic form, the choice should be trigonometric, whereby very good results have been achieved for normalized data. Determining the number of recursive loops r was not so unambiguous, because with an increase in the number of loops, training errors decreased. But on the other hand, this was accompanied by a significant extension of the program duration. Thus, the number $r = 2$ is an effective compromise between these factors.

The best configuration of the parameters of the PSO algorithm is built on the appropriate selection of parameters ω c_1 and c_2 as well as the initialization and limitations of the velocity and position vectors. Many variants of the inertia coefficient and confidence coefficients were tested, and a convenient form of the training algorithm was obtained for the base case. For normalized data, the desired behavior is gained when the constraint values were $|v_{\max}| = 0.1$. Limiting the range of weight initialization is the simplest and arithmetically most justified move for minimizing the training error.

Table 7. PSO procedure with minimum particle velocity limitation.

	Variant A1		Variant A2	
	E_{train}	E_{test}	E_{train}	E_{test}
$\lvert v_{\min}\rvert = 0.1$	0.9044	1.5117	0.3695	0.2541
$\lvert v_{\min}\rvert = 0.05$	0.9001	1.1527	0.5841	0.8624
$\lvert v_{\min}\rvert = 0.01$	0.5253	0.5507	0.1524	0.0880
$\lvert v_{\min}\rvert = 0.001$	0.2955	0.2638	0.0621	0.0241

Table 8. Influence of extending the PSO procedure on the swarm's geometric center.

	$c_i = 1.496$		$c_i = 1.333$	
	E_{train}	E_{test}	E_{train}	E_{test}
$T = 1$	9.4571	8.7413	0.0152	0.0202
$T = 4$	28.2332	29.4629	0.0134	0.0103
$T = 7$	19.7599	26.5320	0.0473	0.0479
$T = 10$	15.3353	14.4675	0.0511	0.0508

In this case, the algorithm in which the constraint was represented as $|X_{\max}| = 0.1$ is characterized by providing the best solutions.

As part of this article, extending the PSO algorithm used in the training process of the FFF network was also examined. The addition of a regularization factor did not improve the performance of the training algorithm. Regarding the use of the minimum speed limit, in this case, acceptable results are obtained using the mean velocity substitution for the range $[-0.001, 0.001]$. Moreover, very satisfactory results are gained for application of the grouping of particles inside the swarm. For the case of $c_i = 1.333$ and with the time of launching, this modification being every 4 epochs, the results were very good. In the case of using the grouping procedure with respect to the geometric center of the swarm, no significant enhancement of the training algorithm was noticed.

In conclusion, numerical verification shows the positive property of FFFNNs allied with PSO as a training procedure, albeit with some modifications. It should also be underlined that in the examples of approximation in this study, neural networks with significantly lower complexity (a lower number of layers and neurons) than in the work of Lovassy *et al.* (2008a) were used.

This work does not deal with the issue of dynamic changes in the architecture of the neural network. Modifications of the number of neurons in hidden layers during the training process may complement the algorithms included in this paper and contribute to increasing the efficiency of FFF networks. This issue will be the subject of future research, as it requires either the use of other optimization algorithms (e.g., AG, AE) or a radical change in the structure of the PSO procedure.

In addition, it is worth emphasizing that the solution proposed in this article can be effectively applied to real issues, e.g., data classification. In additional research, FFFNNs, along with a PSO-based training procedure, performed as well as other known neural algorithms.

Acknowledgment

This article is financed with grants for statutory activity of the Faculty of Physics and Applied Computer Science at the AGH University of Science and Technology in Cracow.

The authors would also like to thank the anonymous referees for their careful reading of the paper and their contribution of useful suggestions that helped to improve this article.

References

Basha, S.S., Dubey, S.R., Pulabaigari, V. and Mukherjee, S. (2020). Impact of fully connected layers on performance of convolutional neural networks for image classification, *Neurocomputing* **378**: 112–119.

Bodyanskiy, Y.V. and Tyshchenko, O.K. (2019). A hybrid cascade neuro-fuzzy network with pools of extended neo-fuzzy neurons and its deep learning, *International Journal of Applied Mathematics and Computer Science* **29**(3): 477–488, DOI: 10.2478/amcs-2019-0035.

Carvalho, M. and Ludermir, T.B. (2006). Particle swarm optimization of feed-forward neural networks with weight decay, *6th International Conference on Hybrid Intelligent Systems (HIS'06), Rio de Janeiro, Brazil*, pp. 5–5.

Chang, C.-H. (2015). Deep and shallow architecture of multilayer neural networks, *IEEE Transactions on Neural Networks and Learning Systems* **26**(10): 2477–2486.

Chen, G. (2010). Simplified particle swarm optimization algorithm based on particles classification, *6th International Conference on Natural Computation, Yantai, China*, Vol. 5, pp. 2701–2705.

Chen, M. (2008). Second generation particle swarm optimization, *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), Hong Kong, China*, pp. 90–96.

Eberhart, R.C. and Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization, *Proceedings of the 2000 Congress on Evolutionary Computation, CEC00, La Jolla, USA*, Vol. 1, pp. 84–88.

Gal, L., Botzheim, J. and Koczy, L.T. (2008). Improvements to the bacterial memetic algorithm used for fuzzy rule base extraction, *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications, Istanbul, Turkey*, pp. 38–43.

Gál, L., Botzheim, J., Kóczy, L.T. and Ruano, A.E. (2009). Applying bacterial memetic algorithm for training feedforward and fuzzy flip-flop based neural networks, *Joint 2009 International Fuzzy Systems Association World Congress and the European Society of Fuzzy Logic and Technology Conference, Lisbon, Portugal*, pp. 1833–1838.

Gál, L., Lovassy, R. and Kóczy, L.T. (2010). Function approximation performance of fuzzy neural networks based on frequently used fuzzy operations and a pair of new trigonometric norms, *International Conference on Fuzzy Systems, Barcelona, Spain*, pp. 1–8.

Gniewek, L. and Kluska, J. (2004). Hardware implementation of fuzzy Petri net as a controller, *IEEE Transactions on Systems, Man, and Cybernetics B: Cybernetics* **34**(3): 1315–1324.

Hirota, K. and Ozawa, K. (1989). The concept of fuzzy flip-flop, *IEEE Transactions on Systems, Man, and Cybernetics* **19**(5): 980–997.

Hirota, K. and Pedrycz, W. (1993). Neurocomputations with fuzzy flip-flops, *Proceedings of International Conference on Neural Networks (IJCNN-93), Nagoya, Japan*, Vol. 2, pp. 1867–1870.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization, *Proceedings of the ICNN'95 International Conference on Neural Networks*, *Perth, Australia*, Vol. 4, pp. 1942–1948.

Kowalski, P.A. (2013). Evolutionary strategy for the fuzzy flip-flop neural networks supervised learning procedure, *International Conference on Artificial Intelligence and Soft Computing, Zakopane, Poland*, pp. 294–305.

Lillicrap, T.P., Cownden, D., Tweed, D.B. and Akerman, C.J. (2016). Random synaptic feedback weights support error backpropagation for deep learning, *Nature Communications* **7**(1): 1–10.

Lovassy, R., Kóczy, L.T. and Gál, L. (2008a). Applicability of fuzzy flip-flops in the implementation of neural networks, *9th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics, CINTI 2008, Budapest, Hungary*, pp. 333–344.

Lovassy, R., Koczy, L.T. and Gal, L. (2008b). Multilayer perceptron implemented by fuzzy flip-flops, *IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence), Hong, Kong, China*, pp. 1683–1688.

Lovassy, R., Zavala, A.H., Gál, L., Nieto, O.C., Kóczy, L.T. and Batyrshin, I. (2010). Hardware implementation of fuzzy flip-flops based on Łukasiewicz norms, *9th WSEAS International Conference on Applied Computer and Applied Computational Science, Genova, Italy*, pp. 196–201.

Łukasik, S. and Kowalski, P.A. (2014). Fully informed swarm optimization algorithms: Basic concepts, variants and experimental evaluation, *Federated Conference on Computer Science and Information Systems, Warsaw, Poland*, pp. 155–161.

Ozawa, K., Hirota, K. and Koczy, L.T. (1996). Fuzzy flip-flop, *in* M.Y. Patyra and D.M. Mlynek (Eds), *Fuzzy Logic: Implementation and Applications*, Wiley/ BG Teunbner Publ., pp. 197–236.

Ozawa, K., Hirota, K., Koczy, L.T. and Omori, K. (1991). Algebraic fuzzy flip-flop circuits, *Fuzzy Sets and Systems* **39**(2): 215–226.

Pu, X., Fang, Z. and Liu, Y. (2007). Multilayer perceptron networks training using particle swarm optimization with minimum velocity constraints, *in* D. Liu *et al.* (Eds), *Advances in Neural Networks*, Lecture Notes in Computer Science, Vol. 493, Springer, Berlin/Heidelberg, pp. 237–245.

Rakitianskaia, A. and Engelbrecht, A. (2015). Saturation in PSO neural network training: Good or evil?, *IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan*, pp. 125–132.

Rutkowski, L. (2008). *Computational Intelligence: Methods and Techniques,* Springer, Berlin/Heidelberg.

Siminski, K. (2021). An outlier-robust neuro-fuzzy system for classification and regression, *International Journal of Applied Mathematics and Computer Science* **31**(2): 303–319, DOI: 10.34768/amcs-2021-0021.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*, Wiley, Hoboken.

Tsoulos, I.G., Tzallas, A. and Karvounis, E. (2021). Improving the PSO method for global optimization problems, *Evolving Systems* **12**: 1–9, DOI: 10.1007/s12530-020-09330-9.

Zavala, A.H., Nieto, O.C., Batyrshin, I. and Vargas, L.V. (2009). VLSI implementation of a module for realization of basic t-norms on fuzzy hardware, *IEEE International Conference on Fuzzy Systems, Jeju, South Korea*, pp. 655–659.

Piotr A. Kowalski is an associate professor at the Faculty of Physics and Applied Computer Science, AGH University of Science and Technology, Cracow, Poland, and at the Systems Research Institute of the Polish Academy of Sciences. He received his MSc in teleinformatics (with honors) and automatic control (with honors) from the Cracow University of Technology in 2003 and his PhD on data analysis from the Polish Academy of Sciences in 2009. In 2018, he received his DSc degree from the Systems Research Institute of the Polish Academy of Sciences. His research interests are in the area of information technology and are focused on intelligent methods (neural networks, fuzzy systems and nature-inspired algorithms) with applications to complex systems and knowledge discovery.

Tomasz Słoczyński is an IT specialist working outside academia. He earned his MSc degree in computer science from the Faculty of Physics and Applied Computer Science, AGH University of Science and Technology, Cracow, Poland, in 2020. His research interests include data analysis and computational intelligence methods.

Received: 28 April 2021
Revised: 2 September 2021
Accepted: 7 September 2021