An Approach to ANFIS Performance

Stepan Dalecky and Frantisek V. Zboril

Faculty of Information Technology
Brno University of Technology
Brno, Czech republic
`idalecky@fit.vutbr.cz, zboril@fit.vutbr.cz`

Abstract. The paper deals with Adaptive neuro-fuzzy inference system (ANFIS) and its performance. Firstly, ANFIS is described as a hybrid system based on fuzzy logic/sets and artificial neural networks. Subsequently, modifications of ANFIS are proposed. The aim of these modifications is to improve performance, accuracy or reduce computational time. Finally, experiments are presented and findings are assessed.

Keywords: ANFIS, artificial neural network, performance, fuzzy sets, fuzzy logic

1 Introduction

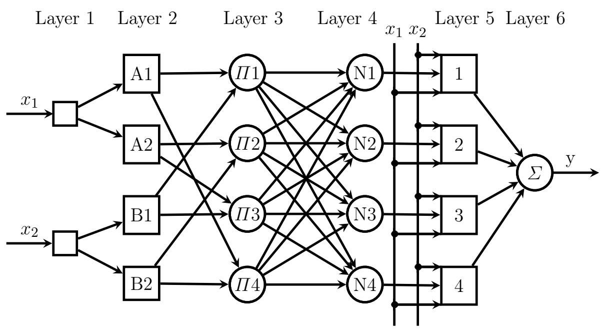
Many problems can be successfully solved using some combination of fuzzy logic [6, 7] and artificial neural networks [6, 1, 8]. Each of these two theories has its pros and cons. That is the reason why it is worth to develop hybrid system which takes advantages of both. One well known example of such system is the Adaptive neuro-fuzzy inference system (ANFIS) developed by Jang [3, 6, 4]. ANFIS has borrowed vagueness and fuzziness from fuzzy sets and learning capability from artificial neural networks. This system has been used for solving many problems, e.g. controlling [5], prediction [2] and classification problems. This paper describes ANFIS and proposed modification of ANFIS that leads to better performance, accuracy and/or less computational time.

2 ANFIS

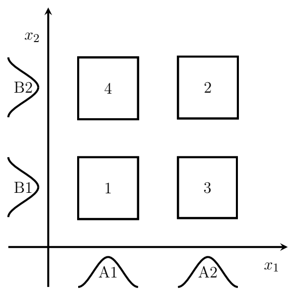
From fuzzy sets point of view, ANFIS represents Sugeno model of the first order. On the other hand, from artificial neural networks point of view, ANFIS represents six layer feed-forward neural network. Architecture of ANFIS is shown in Fig. 1a.

It is obvious that ANFIS shown in this figure divides an input space as it is shown in Fig. 1b and next four rules can be derived from it:

Parameters k_{ij} are specific constants for each rule and their settings will be described later.



(a) Architecture



(b) Division of the input space.

Fig. 1. ANFIS

Rule 1:
IF x_1 is A_1
AND x_2 is B_1
THEN $y = k_{10} + k_{11}x_1 + k_{12}x_2$

Rule 2:
IF x_1 is A_2
AND x_2 is B_2
THEN $y = k_{20} + k_{21}x_1 + k_{22}x_2$

Rule 3:
IF x_1 is A_2
AND x_2 is B_1
THEN $y = k_{30} + k_{31}x_1 + k_{32}x_2$

Rule 4:
IF x_1 is A_1
AND x_2 is B_2
THEN $y = k_{40} + k_{41}x_1 + k_{42}x_2$

2.1 Description of Layers

In this subsection, functions of all layers with respect to ANFIS [3, 6] architecture in Fig. 1a are explained into more details. Symbols $x_{ij}^{(l)}$ and $y_i^{(l)}$ denote j-th input and output of i-th neuron in l-th layer, respectively. If i-th neuron has one input only then this input is denoted simply as $x_i^{(l)}$.

Layer 1 – Input Layer The first layer only distributes input values to the second layer using Eq. (1) as follows.

$$y_i^{(1)} = x_i^{(1)} \tag{1}$$

Layer 2 – Fuzzification Layer The second layer accomplish fuzzification of inputs using bell membership function Eq. (2) as follows.

$$y_i^{(2)} = bell(x_i^{(2)}; a_i, b_i, c_i) = \frac{1}{1 + \left(\frac{x_i^{(2)} - a_i}{c_i}\right)^{2b_i}} \tag{2}$$

It is obvious that parameters a_i, b_i, c_i determine shape of the bell function: centre, width, and slope.

Layer 3 – Rule Layer The third layer corresponds to the rules. Each neuron in this layer represents one rule. Inputs of this layer are membership degrees (outputs of the previous layer), outputs are strength of the rules computed using Eq. (3) as products of all inputs. $C(i)$ symbol denotes the set of neurons of the second layer which are connected to i-th neuron in the third layer.

$$y_i^{(3)} = \prod_{j \in C(i)} x_j^{(3)} = \mu_i \tag{3}$$

Layer 4 – Normalization Layer The fourth layer is used for normalization of rule strength. Inputs of all neurons are corresponding outputs of n neurons of the previous layer. Normalized strengths of corresponding rules are computed using Eq. (4).

$$y_i^{(4)} = \frac{x_i^{(4)}}{\sum\limits_{j=1}^{n} x_j^{(4)}} = \frac{\mu_i}{\sum\limits_{j=1}^{n} \mu_j} = \bar{\mu}_i \tag{4}$$

Layer 5 – Defuzzification Layer The fifth layer uses Eq. (5) to compute consequent (THEN part) strengths of the rule according to the first order Sugeno model [6, 4]. Inputs of this layer are outputs of the previous layer and input variables x_1 and x_2.

$$y_i^{(5)} = x_i^{(5)} \left[k_{i0} + k_{i1}x_1 + k_{i2}x_2\right] = \bar{\mu}_i \left[k_{i0} + k_{i1}x_1 + k_{i2}x_2\right] \tag{5}$$

Layer 6 – Sum Layer The sixth layer computes output of whole network as sum of layer inputs using Eq. (6) as follows.

$$y = y^{(6)} = \sum_{i=1}^{n} x_i^{(6)} = \sum_{i=1}^{n} \bar{\mu}_i \left[k_{i0} + k_{i1}x_1 + k_{i2}x_2\right] \tag{6}$$

2.2 Parameters and Learning

Good performance of proposed model considerably depends on values of parameters a_i, b_i, c_i in the second layer and k_{ij} in the fifth layer. It is impossible to determine optimal value of these parameters directly so we have to estimate them and then tune them to get better performance.

Process of tuning parameters is called learning of ANFIS. Parameters that have to be tuned are divided into two groups:

- parameters that are linear from the output point of view (parameters k_{ij}),
- parameters that are non-linear from the output point of view (parameters a_i, b_i, c_i).

Each group of parameters is tuned separately. One learning step consists of forward pass and backward pass.

Forward Pass Vector $\boldsymbol{k}$ of parameters k_{ij} is tuned during this pass. Suppose m inputs of ANFIS and n neurons in the third layer (number of rules) now, instead 2 inputs and 4 rules shown in Fig. 1a. Then vector $\boldsymbol{k}$ is $n(1+m)$ vector of parameters of the fifth layer.

$$\boldsymbol{k} = \left[k_{10}\ k_{11}\ k_{12} \ldots k_{1m}\ k_{20}\ k_{21}\ k_{22} \ldots k_{2m} \ldots k_{n0}\ k_{n1}\ k_{n2} \ldots k_{nm}\right]^T$$

Let P be number of training vectors. Then outputs of network create vector $\boldsymbol{y}$ of P elements - each row of $\boldsymbol{y}$ is response of ANFIS to the one input vector. We can write

$$\boldsymbol{y} = \mathbf{A}\boldsymbol{k} \tag{7}$$

where $\mathbf{A}$ is $P \times n(1+m)$ dimensional matrix which holds network state after the computation of the fourth layer output.

$$\mathbf{A} = \begin{bmatrix} \bar{\mu}_1(1) & \bar{\mu}_1(1)x_1(1) & \cdots & \bar{\mu}_1(1)x_m(1) & \cdots & \bar{\mu}_n(1) & \bar{\mu}_n(1)x_1(1) & \cdots & \bar{\mu}_n(1)x_m(1) \\ \bar{\mu}_1(2) & \bar{\mu}_1(2)x_1(2) & \cdots & \bar{\mu}_1(2)x_m(2) & \cdots & \bar{\mu}_n(2) & \bar{\mu}_n(2)x_1(2) & \cdots & \bar{\mu}_n(2)x_m(2) \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \bar{\mu}_1(p) & \bar{\mu}_1(p)x_1(p) & \cdots & \bar{\mu}_1(p)x_m(p) & \cdots & \bar{\mu}_n(p) & \bar{\mu}_n(p)x_1(p) & \cdots & \bar{\mu}_n(p)x_m(p) \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \bar{\mu}_1(P) & \bar{\mu}_1(P)x_1(P) & \cdots & \bar{\mu}_1(P)x_m(P) & \cdots & \bar{\mu}_n(P) & \bar{\mu}_n(P)x_1(P) & \cdots & \bar{\mu}_n(P)x_m(P) \end{bmatrix}$$

Equation Eq. (7) is a matrix notation of Eq. (6) and Eq. (5) for set of training vectors.

Let $\boldsymbol{y}_d$ be $P \times 1$ vector of desired outputs.

$$\boldsymbol{y}_d = \left[y_d(1)\ y_d(2) \ldots y_d(p) \ldots y_d(P)\right]^T$$

Then error vector $\boldsymbol{e}$ of the network can be defined as follows

$$\boldsymbol{e} = \boldsymbol{y}_d - \boldsymbol{y} \tag{8}$$

Goal of the learning is to find such vector $\boldsymbol{k}$, respectively its estimate $\boldsymbol{k}^*$, for which mean square error defined by Eq. (9) is minimal (zero).

$$\|\boldsymbol{e}\|^2 = \|\boldsymbol{y}_d - \boldsymbol{y}\|^2 = \|\boldsymbol{y}_d - \mathbf{A}\boldsymbol{k}\|^2 \tag{9}$$

Vector $\boldsymbol{k}$ can be computed using Eq. (10) as follows (with respect to Eq. (7)).

$$\boldsymbol{k} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{y}_d \tag{10}$$

However inverse matrix $(\mathbf{A}^T\mathbf{A})^{-1}$ can be calculated if and only if the matrix $(\mathbf{A}^T\mathbf{A})$ is regular. Unfortunately there is no guarantee of this and that is why we have to compute estimate of vector $\boldsymbol{k}$, denoted $\boldsymbol{k}^*$, using pseudoinverse matrix Eq. (11).

$$\boldsymbol{k}^* = \mathbf{A}^+\boldsymbol{y}_d \tag{11}$$

Symbol $\mathbf{A}^+$ denotes pseudoinverse of matrix $\mathbf{A}$. Finally, vector $\boldsymbol{k}^*$ is vector of estimated parameters of the fifth layer.

Backward Pass During this pass, parameters a_i and c_i are tuned (b_i remains constant). Method is similar to well-known backpropagation method - it uses backward propagation of errors. Error is computed using Eq. (12), where y_d and y denote desired and real output of the network, respectively.

$$E = \frac{1}{2}\left(y_d - y\right)^2 \tag{12}$$

Goal of learning is to minimize error E by changing values of parameters in the second layer. Assume that α_m denotes general parameter of the m-th neuron in the second layer. Then minimization is done according to Eq. (13) by tuning α_m. E should decreases if partial derivative of E with respect to α_m is negative. Symbol κ denotes a learning rate.

$$\Delta\alpha_m = -\kappa \frac{\partial E}{\partial \alpha_m} \tag{13}$$

Using the chain rule and Eq. (13) we can derive Eq. (14), where $n^{(5)}$ is number of neurons in the fifth layer (number of rules), $C^{-1}(m)$ is a set of neurons of the third layer which are connected with m-th neuron in the second layer and $\frac{\partial y_m^{(2)}}{\partial \alpha_m}$ is partial derivative of bell membership function with respect to α_m.

$$\Delta\alpha_m = \kappa \frac{y_d - y^{(6)}}{y_m^{(2)}} \left(\sum_{j\in C^{-1}(m)} \left(y_j^{(5)}\right) - \sum_{j=1}^{n^{(5)}} \left(\frac{y_j^{(5)}}{y_j^{(3)}} y_j^{(4)} \right) \sum_{l \in C^{-1}(m)} \left(y_l^{(3)}\right) \right) \frac{\partial y_m^{(2)}}{\partial \alpha_m} \tag{14}$$

Partial derivative of bell membership function with respect to a_i and c_i are computed using Eq. (15) and Eq. (16).

$$\frac{\partial y_i^{(2)}}{\partial a_i} = y_i^{(2)} \frac{2b_i}{c_i} \left(\frac{y_i^{(1)} - a_i}{c_i} \right)^{2b_i - 1} \tag{15}$$

$$\frac{\partial y_i^{(2)}}{\partial c_i} = \frac{2b_i \left(\frac{y_i^{(1)} - a_i}{c_i} \right)^{2b_i}}{c_i \left(\left(\frac{y_i^{(1)} - a_i}{c_i} \right)^{2b_i} + 1 \right)^2} \tag{16}$$

Finally, values of Δa_i and Δc_i are computed by substitution Eq. (15) and Eq. (16) into Eq. (14).

3 Proposed Modifications of ANFIS

This section describes proposed modifications of ANFIS. After each modification is presented their impact on performance or accuracy is discussed.

3.1 Different Number of Fuzzy Sets for Each Input

Original ANFIS uses the same number of fuzzy sets for each input variable regardless of variable properties. For example if one variable does not change much (e.g. affects output only linearly or similarly) there is no need to has as much fuzzy sets as must have a variable which changes a lot (affects outputs non-linear, high frequency etc.). We can distribute the same amount of fuzzy sets more precisely and put it where they improve accuracy and performance with the same or even less computational time.

This modification brings performance and/or accuracy in comparison with the original ANFIS, depending on the use. The worst case that should happened is that all inputs have the same number of fuzzy sets and it can be easily reached even with this modification.

3.2 Data Normalization

Great method for improve ANFIS performance is data pre-processing. Z-score normalization Eq. (17) is used for this purpose.

$$z = \frac{x - \mu}{\sigma} \tag{17}$$

Symbol x denotes input/output variable, μ denotes mean and σ is standard deviation. Such normalization can be used for every input/output variable but we have to save mean and standard deviation values for each variable to be able to normalize training vectors, testing vectors etc. in the same way and also for reconstructing original value of variable.

Benefit of this modification heavily depends on data properties and it will be discussed in the experiment section.

3.3 Fuzzy Sets Initialization

Parameters a_i, b_i and c_i have to be properly initialized. Good approach is to analyse training data and estimate these parameters. After the z-score normalization, minimum and maximum of each input variable is computed, desired fuzzy count for variable is divided uniformly from minimum to maximum. Parameter b_i is a constant (e.g. $b_i = 2$ for reasonable membership function shape), parameters a_i and c_i are computed using Eq. (18).

$$step = \tfrac{max - min}{n} \qquad a_i = min + \tfrac{step}{2} + i \cdot step \qquad c_i = \tfrac{step}{2} \tag{18}$$

Symbols min and max denote minimum and maximum of input values respectively, n is desired number of fuzzy sets.

This modification has main impact on performance. Proper fuzzy sets initialization can significantly decrease learning time on the other hand bad fuzzy sets initialization increases learning time because learning process start from point far from the solution. Accuracy is affected negligibly because it should converges to the almost same solution as well but it takes more time.

3.4 Changing Learning Rate κ

We borrowed this modification from article [3]. Idea is to modify learning rate in order to improve performance. Symbol κ in Eq. (14) is computed using Eq. (19).

$$\kappa = \frac{\eta}{\sqrt{\left(\frac{\partial E}{\partial a_i}\right)^2 + \left(\frac{\partial E}{\partial c_i}\right)^2}} \tag{19}$$

Symbol κ denotes new learning rate, η is old learning rate. η is usually initialized from interval 0.001 to 0.1. Main advantage is that new learning rate doesn't depend on input data as much as original κ. This modification also brings possibility to tune η depending on success of error minimization. If network error decreases then learning rate increases, on the other hand if error oscillates learning rate decreases, finally if error increases there is no change of learning rate. η is increased by 5 % after 4 consecutive iterations which decrease error. If error oscillates (up, down, up, down) during 4 consecutive iterations η is decreased by 10 %.

Described modification may slightly decrease accuracy in favour to significant performance improvement.

4 Experiments

Two problems has been chosen to demonstrate how proposed system works - function approximation and controlling of a system.

At the first, some metrics to measure performance and accuracy are described.

Performance can be divided into two aspects:

– number of learning iterations (forward and backward pass),
– time that ANFIS needs to converge (to stop learning).

Accuracy is measured as difference between ANFIS output $y(p)$ and desired output $y_d(p)$ for training vector $p \in P$ using E or EA from Eq. (20).

$$E(P) = \frac{\sum_{p \in P} (y(p) - y_d(p))^2}{|P|} \qquad EA(P) = \frac{\sum_{p \in P} |y(p) - y_d(p)|}{|P|} \tag{20}$$

4.1 Function Approximation

Two non-linear functions and their modifications have been chosen to demonstrate performance and accuracy of proposed ANFIS. Both functions are uniformly sampled in each axis in interval $< -5, 5 >$ with 0.1 steps.

The first function is two variables *sinc* function Eq. (21). This function is symmetric in terms of input variable so it is appropriate to have same number of fuzzy sets for each input. In this experiment 3 fuzzy sets are used. To demonstrate efficiency of different number of fuzzy sets for each input, function *sincm* Eq. (21) is chosen and 4 fuzzy sets are used for input variable x, only 3 fuzzy sets are used for y.

$$sinc\,(x,y) = \frac{\sin x}{x} + \frac{\sin y}{y} \qquad sincm\,(x,y) = \frac{\sin 2x}{x} + \frac{\sin y}{y} \tag{21}$$

To verify and confirm results another function f and its modified version fm have been chosen Eq. (22). In case of f, 3 fuzzy sets are used for x and y while in case of fm, 4 fuzzy sets are used for x and 3 for y only.

$$f\,(x,y) = \sin^2 x \cdot \cos y \qquad fm\,(x,y) = \sin^2 x \cdot \cos \tfrac{y}{2} \tag{22}$$

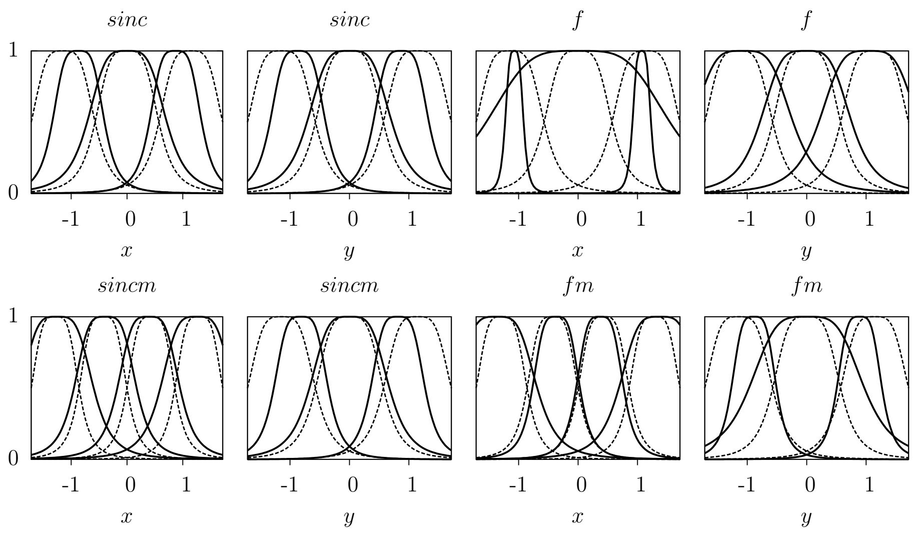


Fig. 2. Membership function after initialization and after learning.

Fuzzy sets after initialization are shown[1] in Fig. 2 by dashed line and solid line is used to represent fuzzy sets after learning. Corresponding sets have same colours (shade of black). It can be seen that there are noticeable differences between fuzzy sets of original (*sinc*, f) and modified (*sincm*, fm) functions.

Performance and accuracy of this experiments are shown in Tab. 1. To avoid impact of initial learning rate several η are chosen $\{\eta_1 = 5 \cdot 10^{-4}, \eta_2 = 1 \cdot 10^{-3}, \eta_3 = 5 \cdot 10^{-3}, \eta_4 = 1 \cdot 10^{-2}, \eta_5 = 3 \cdot 10^{-2}, \eta_6 = 5 \cdot 10^{-2}\}$. Table is organized

[1] Values on both axes are normalized by z-score. So they don't directly correspond to the values of *sinc*.

as follows: Each experiment is on two rows. The first one contains used ANFIS configuration[2] and number of iterations for each η while the second row contains accuracy[3] and time in seconds[4].

It can be seen in Tab. 1 that modified ANFIS works well. Function *sinc* Tab. 2a is learned in about 76.1% less iterations and 76.8% less time with error below 10^{-3} in average, about 50% of iterations and time are saved with error below 10^{-4}. With *sincm* Tab. 2c modified ANFIS uses only about 6.7% less iterations but time is reduced about 50.3% with the same accuracy $2 \cdot 10^{-4}$. Function f Tab. 2b, ANFIS used in about 60.4% less iterations and 62.6% less time with same accuracy. Finally, using fm iterations are about 11.6% more but time is reduced about 21.0%. So proposed modifications save time and iterations from 7% to nearly 80%.

Table 1. Number of learning iterations and consumed time

	η_1	η_2	η_3	η_4	η_5	η_6	Σ
Orig33	210	159	65	39	16	11	**500**
10^{-3}	4.72	3.43	1.47	0.90	0.42	0.41	**11.34**
Mod33	134	92	29	17	7	5	**284**
10^{-3}	3.03	2.06	0.65	0.38	0.17	0.12	**6.42**
Orig33	266	212	103	66	37	41	**725**
10^{-4}	6.48	4.59	2.28	1.46	0.86	1.04	**16.71**
Mod33	184	135	51	38	50	28	**486**
10^{-4}	4.11	2.96	1.14	0.86	1.12	0.62	**10.79**

(a) sinc

	η_1	η_2	η_3	η_4	η_5	η_6	Σ
Orig33	292	237	122	82	38	26	**797**
$4 \cdot 10^{-3}$	6.98	5.51	2.85	1.88	0.94	0.84	**19.00**
Mod33	208	158	64	39	17	11	**497**
$4 \cdot 10^{-3}$	4.94	3.65	1.50	0.92	0.41	0.26	**11.68**

(b) f

	η_1	η_2	η_3	η_4	η_5	η_6	Σ
Orig33	177	129	47	28	11	7	**399**
$5 \cdot 10^{-2}$	4.09	2.81	1.15	0.67	0.32	0.25	**9.30**
Orig44	254	201	103	81	63	65	**767**
$2 \cdot 10^{-4}$	12.64	9.36	4.74	3.69	2.87	2.99	**36.29**
Mod43	192	143	83	87	99	115	**719**
$2 \cdot 10^{-4}$	6.51	4.79	2.76	2.87	3.32	3.89	**24.15**

(c) sincm

	η_1	η_2	η_3	η_4	η_5	η_6	Σ
Orig33	251	198	91	58	25	16	**639**
$5 \cdot 10^{-2}$	5.79	4.24	2.02	1.38	0.61	0.53	**14.58**
Mod34	168	121	43	25	10	7	**374**
$2 \cdot 10^{-4}$	5.49	3.95	1.42	0.86	0.36	0.26	**12.33**
Orig44	203	153	61	37	15	10	**479**
$2 \cdot 10^{-4}$	9.83	6.88	2.75	1.67	0.71	0.48	**22.31**
Mod43	221	169	72	45	21	14	**542**
$2 \cdot 10^{-4}$	7.20	6.25	2.37	1.46	0.69	0.48	**18.45**

(d) fm

4.2 Controlling Discrete System

The second experiment is controlling discrete system Eq. (23) which has been presented in literature [4]. This system has one state variable y and one input u.

$$y(k+1) = \frac{y(k) \cdot u(k)}{1 + y(k)^2} - \tan(u(k)) \tag{23}$$

[2] OrigAB is used for original ANFIS with A fuzzy sets for x and B for y while ModAB means our modified ANFIS with A fuzzy sets for x and B for y.

[3] Accuracy means maximal error computed using E from Eq. (20).

[4] Time is measured on our testing machine with Intel Core i5-2540M.

Goal of controlling is to produce such input u that force state variable y (in this case the same variable as output variable) to track desired trajectory given by Eq. (24).

$$y_d\left(k\right) = 0.6 \sin\left(\frac{2\pi k}{250}\right) + 0.2 \sin\left(\frac{2\pi k}{50}\right) \tag{24}$$

Inverse learning [4] is used to generate training data and control the system. Size of training vector is 100 samples, initial learning rate $\eta = 5 \cdot 10^{-3}$. Training environment consists of follows: $y(0) = 0$ and action $u(k)$ is generated randomly each step from range $< -1, 1 >$ using uniform distribution. Training vectors are $[y\left(k\right), y\left(k+1\right); u\left(k\right)]$.

Accuracy of controlling is shown in Fig. 3. It can be seen that error of proposed ANFIS is slightly lower than error of original ANFIS. Main improvement comes from possibility to have different number of fuzzy sets and make ANFIS to fit your needs - accuracy vs. performance. Exact results can bee seen in Tab. 2 computed using EA from Eq. (20).

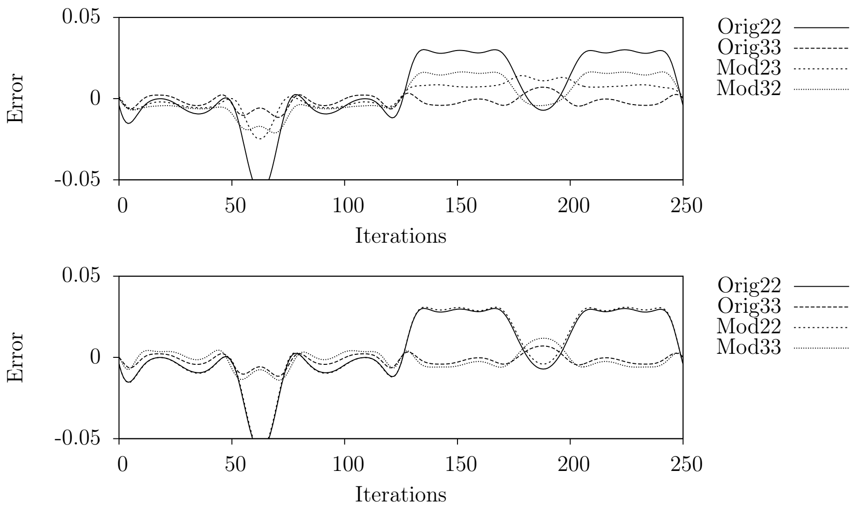


Fig. 3. Comparison of original and proposed ANFIS.

Table 2. Average error

	Orig22	Orig33	Mod22	Mod23	Mod32	Mod33
Average error	0.0161	0.0023	0.0162	0.0058	0.0097	0.0021

5 Conclusion

This paper presented modification of ANFIS. Main goal was to improve performance and to get more accurate results in reasonable time and it has been reached. Proposed modification relies on these changes: number of fuzzy sets for each input may differ, data normalization is added and intelligent fuzzy sets initialization is used, also changing learning rate from original Jang article [3] is borrowed.

To verify benefits of modification two experiments were done. The first one was function approximation and the second one was controlling of a discrete dynamic system. In both cases modified ANFIS had better performance and/or accuracy. Accuracy improvement is not significant in all experiments but the lower number of iterations and less computational time also have to be counted in. Improvement of accuracy and/or performance heavily depends on solved problem and training data. These experiments verified that proposed modification can be successfully used to improve ANFIS performance and/or accuracy.

Further research could be done in way of tuning parameters. For example extended Kalman filter[9] can be used to speed up learning.

Acknowledgement

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070, by the Reliability and Security in IT project FIT-S-14-2486 and by the IGA BUT project FIT-S-14-2486.

References

1. R. A. Aliev and R. R. Aliev. *Soft Computing and its Applications.* World Scientific Publishing Co. Pte. Ltd., 2001.
2. Melek Acar Boyacioglu and Derya Avci. An adaptive network-based fuzzy inference system (anfis) for the prediction of stock market return: The case of the istanbul stock exchange. *Expert Systems with Applications*, 37(12):7908 – 7912, 2010.
3. Jyh-Shing Roger Jang. Adaptive-network-based fuzzy inference system. [online], 1993.
4. Jyh-Shing Roger Jang, C.-T. Sun, and E. Mizutani. *Neuro-Fuzzy and Soft Computing: a computational approach to learning and machine intelligence.* Prentice Hall, 1997.
5. Yanmei Liu, Zhen Chen, Dingyu Xue, and Xinhe Xu. Real-time controlling of inverted pendulum by fuzzy logic. In *Automation and Logistics, 2009. ICAL '09. IEEE International Conference on*, pages 1180–1183, Aug 2009.
6. Michael Negnevitsky. *Artificial Intelligence: A Guide to Intelligent Systems.* Addison-Wesley, 2002.
7. Witold Pedrycz and Fernando Gomide. *An Introduction to Fuzzy Sets: Analysis and Design.* The MIT Press, 1998.
8. Sturart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson, 2010.
9. Dan Simon. Training fuzzy systems with the extended kalman filter. *Fuzzy Sets and Systems*, 2002. `http://academic.csuohio.edu/simond/fuzzyopt/fss.pdf`.