Int. J. Fuzzy Syst.
https://doi.org/10.1007/s40815-024-01697-0

Scikit-ANFIS: A Scikit-Learn Compatible Python Implementation for Adaptive Neuro-Fuzzy Inference System

Dongsong Zhang [1,2] · **Tianhua Chen**[2]

Received: 1 August 2023 / Revised: 5 January 2024 / Accepted: 30 January 2024

Abstract The Adaptative neuro-fuzzy inference system (ANFIS) has shown great potential in processing practical data from control, prediction, and inference applications, reflecting advantages in both high performance and system interpretability as a result of the hybridization of neural networks and fuzzy systems. Matlab has been a prevalent platform that allows to utilize and deploy ANFIS conveniently. On the other hand, due to the recent popularity of machine learning and deep learning, which are predominantly Python-based, implementations of ANFIS in Python have attracted recent attention. Although there are a few Python-based ANFIS implementations, none of them are directly compatible with scikit-learn, one of the most frequently used libraries in machine learning. As such, this paper proposes Scikit-ANFIS, a novel scikit-learn compatible Python implementation for ANFIS by adopting a uniform format such as *fit*() and *predict*() functions to provide the same interface as scikit-learn. Our Scikit-ANFIS is designed in a user-friendly way to not only manually generate a general fuzzy system and train it with the ANFIS method but also to automatically create an ANFIS fuzzy system. We also provide four kinds of representative cases to show that Scikit-ANFIS represents a valuable addition to the scikit-learn compatible Python software that supports ANFIS fuzzy reasoning. Experimental results on four datasets show that our Scikit-ANFIS outperforms recent Python-based implementations while achieving parallel performance to ANFIS in Matlab, a standard implementation officially realized by Matlab, which indicates the performance advantages and application convenience of our software.

Keywords Neuro-fuzzy · Fuzzy system · Anfis · Python · Scikit-learn · PyTorch

1 Introduction

Since the adaptative neuro-fuzzy inference system (ANFIS) [1] was proposed in 1993 as a creative method of combining the advantages of the fuzzy system and neural network, it has been extensively applied in numerous fields. ANFIS is a unique five-layer neural network model that integrates fuzzy sets and logic modeling a fuzzy system. The model features a two-step learning algorithm that comprises a forward pass and a backward pass, which allows for automatic adjustment of the antecedent and consequent parameters by minimizing the error between the actual and target outputs [1]. This approach provides two main benefits, allowing for automatic learning from the data and employing fuzzy if-then rules to explain the model-generated results. By combining the fuzzy system's explainability with the neural network's self-learning ability, this approach delivers unparalleled accuracy and interpretability.

As a result of the above advantages, research and application of ANFIS have drawn widespread attention in many domains. Health and well-being are one of the prioritized applications, due to the interpretability and accuracy typically required in healthcare domain that ANFIS

https://scikit-learn.org/.

✉ Tianhua Chen
T.Chen@hud.ac.uk

[1] School of Big Data and Artificial Intelligence, Xinyang College, Xinyang 464000, Henan, China

[2] School of Computing and Engineering, University of Huddersfield, Huddersfield HD1 3DH, UK

Published online: 03 June 2024

may provide. Researchers have proposed a new approach to clinical decision support that uses data-driven techniques to create interpretable fuzzy rules. This approach combines decision tree learning mechanisms with an ANFIS framework, resulting in a method that outperforms many other popular machine learning techniques in terms of accuracy [2]. Other researchers have used ANFIS optimized through artificial bee colonies to classify heartbeat sounds, aiming at early detection of cardiovascular disease [3]. For the diagnosis process of Alzheimer's disease, some researchers have proposed a technique that first transforms it into a clustering problem, and then uses ANFIS to optimize fuzzy rules, which ultimately improves the accuracy of diagnosis [4]. ANFIS is also widely used in the field of control and engineering. A new combined ANFIS and robust proportional integral derivative control framework are proposed for building structure damping systems, which can effectively ensure the stability and robustness of the controller [5]. Some researchers have proposed using adaptive virtual synchronous generators with ANFIS controllers as inverter controllers in photovoltaic systems, which can enhance the system response in different operating scenarios [6]. ANFIS model has also been utilized to enhance electricity demand forecasting accuracy in a developing country, surpassing prior models and databases [7]. To predict power generation in photovoltaic systems, ANFIS models [8, 9] optimized by genetic algorithm or particle swarm optimization have been developed using Matlab [10] software with sound performance. These applications across a wide range of domains, demonstrate the effectiveness and popularity of ANFIS as a significant data analysis and model construction tool predominantly in decision-making and forecasting tasks, which in turn calls for more efforts in building an accessible development environment to streamline its applications.

At present, Matlab [10] is a widely used platform for convenient utilization and deployment of ANFIS. However, due to the increasing popularity of machine learning and deep learning, which are mainly based on Python, Python-based implementations of ANFIS have gained increasing attention. Despite the availability of some Python-based ANFIS implementations such as ANFIS-PyTorch [11], ANFIS-Numpy [12], and ANFIS-PSO [13], what is lacking in the current research landscape is that none of these implementations are directly compatible with scikit-learn, one of the most commonly used machine learning libraries.

Furthermore, due to emerging advancement in deep learning models, ANFIS has recently undergone new developments, including cascade ANFIS [14, 15], as well as integration with deep learning technology [16]. This has led to the emergence of a popular research area known as deep neural fuzzy system [17], of which ANFIS is an essential component. However, although some researchers have tried combining deep neuro-fuzzy systems (DNFS) [17] created using Python with scikit-learn, such as PyTSK [18], no cases have been found that combine ANFIS with scikit-learn, to the best of our knowledge.

Conventionally, ANFIS application and development are conducted in Matlab. However, with the rapid progress of deep learning and machine learning, which is commonly conducted in a Python environment, it is critical to develop ANFIS in an environment that is directly compatible with Python, Sklearn, and PyTorch [19]. This will facilitate the research and development of ANFIS and ensure compatibility with the latest technologies.

This paper reports a novel implementation of ANFIS, in Python programming language. To ease the use of ANFIS in compatibility with popular machine learning models, which have been realized in the popular scikit-learn library, our implementation, termed Scikit-ANFIS, fully supports interfaces as specified by scikit-learn. Furthermore, our ANFIS implementation, which may be utilized as an optimization method, also supports the training of an existing fuzzy system. Through several case studies and cross-validated experiments, our results demonstrate the superior performance of Scikit-ANFIS software compared to other ANFIS-based or DNFS-based Python software and are parallel to the standard ANFIS implementation by Matlab. Concretely, our contributions can be summarized as:

(1) Our Scikit-ANFIS implementation is fully compatible with commonly used scikit-learn functions such as *fit*() and *predict*() - this enables our development directly applicable in combination with all existing machine learning models and methods as typically conducted through scikit-learn.
(2) Scikit-ANFIS allows the manual generation of a general-purpose Takagi-Sugeno-Kang (TSK) [20] fuzzy system using natural languages. To the best of our knowledge, our method is the only Python-based implementation that supports fuzzy reasoning with complex rules and logical operators of multiple choices.
(3) Scikit-ANFIS utilizes the *scikit_anfis*() class to train a pre-existing TSK fuzzy system and automatically generate an ANFIS fuzzy system based on user-specified input–output data pairs, resulting in an efficient optimized fuzzy system.
(4) The Scikit-ANFIS implementation can automatically save and load the trained ANFIS with the best

performance to/from a local model file, which is not currently available in other Python-based ANFIS implementations.

2 Technical Background on ANFIS

We begin with a brief introduction of the ANFIS fuzzy system [1], as shown in Fig. 1. The basic architecture of ANFIS consists of five layers with the output of the nodes in each respective layer represented by $O_{i,j}$ where i is the *ith* node of layer j.

In the first layer, the nodes of this layer are the membership scores generated based on the values of the fuzzy input variables, defined as:

$$O_i^1(x_1) = \mu_{A_i}(x_1),\ O_i^1(x_2) = \mu_{B_i}(x_2),\ i = 1,2 \tag{1}$$

where x_1, x_2 represent the crisp values of two input variables, and A_i, B_i are the fuzzy set associated with this node, and $\mu_{A_i}(x_1)$, $\mu_{B_i}(x_2)$ denote the membership function of linguistic labels A_i and B_i respectively. Any continuous and piecewise differentiable function such as the commonly used bell-shaped, gaussian, trapezoidal, and triangular membership functions, can be used as a membership function in this layer, and each membership function itself includes a set of parameters. When the values of these parameters change, the membership function also varies, so these parameters in this layer are called premise parameters [1].

In the second layer, each node represents the accumulated firing strength of rule antecedents through a t-norm operator such as the product as:

$$O_i^2 = w_i = \mu_{A_i}(x_1) \times \mu_{B_i}(x_2),\quad i = 1,2 \tag{2}$$

In the third layer, each node outputs a normalized firing strength:

$$O_i^3 = \bar{w}_i = \frac{w_i}{w_1 + w_2},\quad i = 1,2 \tag{3}$$

In the fourth layer, each rule consequent is calculated with associated parameters p_i, q_i, r_i.

$$O_i^4 = \bar{w}_i f_i = \bar{w}_i(p_i x_1 + q_i x_2 + r_i),\quad i = 1,2 \tag{4}$$

When the values of the premise parameters are given, the single node in the fifth layer can be expressed as the sum of the linear combinations of consequent outputs, i.e.,

$$O_1^5 = \sum_i \bar{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i},\quad i = 1,2 \tag{5}$$

It is important to note that ANFIS, unlike neural networks, grows faster in terms of the total number of parameters P_t, which can be calculated as follows [21]:

$$\begin{cases} P_p = in \times MF(\text{input}) \times coef(\text{MF}) \\ P_c = rs \times (in + 1) \times out \\ P_t = P_p + P_c \end{cases} \tag{6}$$

where P_p and P_c denote the number of premise parameters and that of consequent parameters respectively; *in* stands for the number of inputs, *MF*() is the number of membership functions in each input, and *coef*() is the number of coefficients for each membership function; *rs* stands for the number of rules, and *out* is the number of nodes in the fifth layer.

Given the original definition of ANFIS as introduced above, it represents a TSK-type fuzzy system that naturally fits a regression and control problem. Figure 1 shows an ANFIS with two inputs and two rules and one output, where each input has two Gaussian membership functions. The total number of parameters in ANFIS is 14, found by multiplying the coefficient (2) of the Gaussian membership function by the relevant values and adding them up: $P_t = 2 \times 2 \times 2 + 2 \times 3 \times 1 = 14$.

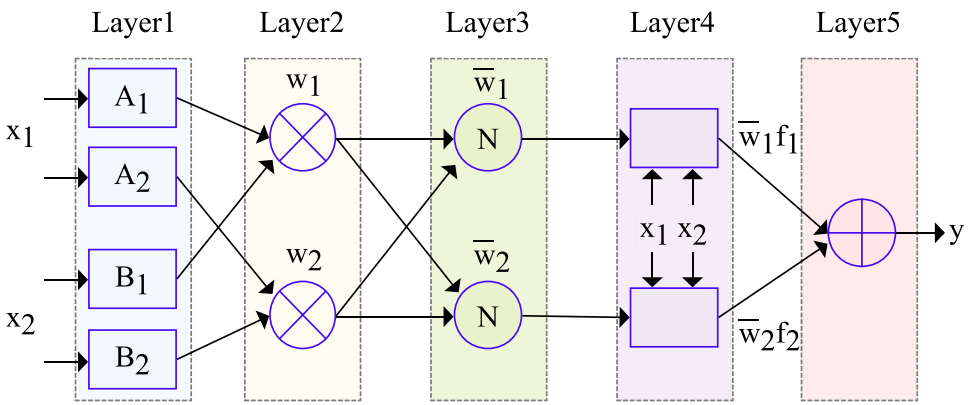


Fig. 1 ANFIS example with two inputs, two membership functions in each input, and two rules

Depending on how the consequent parameters are set and updated, Jang, the inventor of ANFIS [1], proposed two learning algorithms (i.e. training strategies) for the ANFIS model, namely hybrid and online. In hybrid learning, the antecedents are updated by the gradient descent method, while the consequents are calculated by the least squares method after fixing the premise parameters. Meanwhile, in online learning, all parameters are updated by the gradient descent method.

As depicted in Fig. 2, the hybrid learning algorithm comprises two stages: the forward pass and the backward pass. In the forward pass, the functional signal from Layer 1 is passed directly through the ANFIS network to Layer 4, where the consequent parameters are calculated by the least squares estimate (LSE) for input data **X** and target data **Y**. At this point, the premise parameters from the membership functions in Layer 1 remain fixed. The backward pass procedure starts after computing the total root mean square error (RMSE) loss. During this process, the consequent parameters are kept unchanged while the premise parameters are updated using the gradient descent method.

For clarity, the list of terminology abbreviations used in the paper is given in Table 1.

Table 1 The terminology list of abbreviations used in the paper

Abbreviation	Expansion
TSK	Takagi-Sugeno-Kang
ANFIS	Adaptive Neuro-Fuzzy Inference System
DNFS	Deep Neuro-Fuzzy Systems
Sklearn	scikit-learn
N/A	No Answer
Hybrid	gradient descent and least squares estimate
Online	gradient descent only
PSO	Particle Swarm Optimizer
GA	Genetic Algorithm
ABC	Artificial Bee Colony
LSE	Least Squares Estimate
RMSE	Root Mean Square Error
MBGD	Minibatch Gradient Descent
BN	Batch Normalization
UR	Uniform Regularization
LU	Layer Normalization
ReLU	Rectified Linear Unit
MFt	Membership Function types
FIS	Fuzzy Inference System
10-CV	10-fold cross-validation
Acc	Accuracy
n/a	Not applicable

3 Related Work

3.1 Recent Software Development for Fuzzy Systems

Generally speaking, a fuzzy system has a good level of interpretability, due to its knowledge encoding with imprecise knowledge and the intuitive inference mechanism that mimics human reasoning [22, 23]. During the early development of plain fuzzy systems in Python, many Python libraries were moving in the direction of general-purpose fuzzy system applications, such as PyFuzzy [24], Fuzzylab [25], Scikit-Fuzzy [26].

However, many of these tools are outdated or no longer maintained. Recently, an open source software for general fuzzy systems is Simpful [27], which supports the natural language definition of fuzzy variables, fuzzy sets, and fuzzy rules, as well as any order TSK reasoning method. A common limitation is that most of the above software aims to create a general framework, which tends to require the creation of a fuzzy system by hand. The manual creation would become impractical in working with even a moderate-sized data set. Such limitation may also be more obvious in need of an automated optimization of system parameters, which can be dealt with through Matlab, but

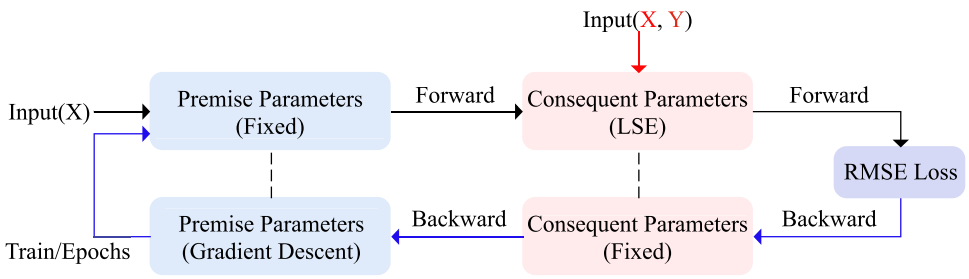


Fig. 2 The hybrid learning process implemented in ANFIS model

existing Python implementations are usually not applicable.

Focusing on the ANFIS framework [1], which has been a very prevalent TSK-type fuzzy system since its inception for a variety of domain problems [28], our Scikit-ANFIS is the first open-source Python tool to combine the creation of a general-purpose TSK fuzzy system embedded with the ANFIS optimization method.

3.2 Brief History of ANFIS Software Development

Since Jang proposed ANFIS, we make a summary of the recent major development of ANFIS software as shown in Table 2. Matlab-ANFIS is one of the most popular tools used to implement the ANFIS model [10], which can not only create the ANFIS model directly to train and test the data set but also utilize ANFIS as an optimization method to train the existing fuzzy system. However, Matlab is commercial software that is not open to the public. Furthermore, an extra installation of the Matlab Engine API for Python is required to access Matlab from Python.

The ANFIS-C [1] and ANFIS-Vignette [21] software written in C and R respectively, are outdated and not regularly updated. Currently, ANFIS software such as ANFIS-PyTorch [11], ANFIS-Numpy [12], and ANFIS-PSO [13] are mostly developed in Python 3 [29]. Out of the above three Python-based software, none supports the scikit-learn interface, and only a limited number of membership function types are supported (5, 3, and 3, respectively). ANFIS-PyTorch is the only software that supports both hybrid and online learning algorithms, while ANFIS-Numpy only supports hybrid learning, and ANFIS-PSO supports the particle swarm optimizer (PSO). On the other hand, our Scikit-ANFIS supports 12 different membership function types, the same as Matlab-ANFIS. Additionally, Scikit-ANFIS fully supports two learning algorithms (Hybrid/Online) for ANFIS training and also supports the scikit-learn interface, which is more user-friendly and has more powerful application capabilities.

3.3 Review on Deep Neuro-fuzzy Systems Framework

Deep neuro-fuzzy systems (DNFS), which present one of the most advanced developments as a combination of deep learning and fuzzy systems, have become a focus in fuzzy logic research [31]. This is because fuzzy systems can not only deal with the widespread inaccuracy and uncertainty in the real world but also potentially enrich the representation of deep models. At the same time, ANFIS can be seen as a simplified representation of DNFS [17], which itself is in principle a fuzzy system whose membership function parameters can be tuned by a five-layer adaptive neural network [1].

We further compare our Scikit-ANFIS implementation with other deep neural fuzzy methods for regression and classification in Table 3. There are several methods available for solving classification tasks, including the Neuro-Fuzzy [32] method based on C language, DNFC [33] based on Matlab, and TSK-MBGD-UR-BN [35] and PyTSK [18] based on Python 3. For regression tasks, there are also various methods available, such as FCM-RDpA [36] developed based on Matlab, MBGD-RDA [34], and HTSK-LN-ReLU [37] developed based on Python 3. However, only our Scikit-ANFIS is capable of solving both classification and regression tasks. It's worth noting that Python 3 has become a popular choice among the fuzzy logic research community, likely due to its widespread use in developing artificial neural networks. Although the methods mentioned above offer practical solutions for their specific tasks and implement different optimization techniques like gradient descent, minibatch gradient descent [38], Adam [39], AdaBound [40], Powerball [41], and AdaBelief [42], they do not utilize the ANFIS architecture or its training methods. By contrast, our Scikit-ANFIS has the ability to not only adopt the ANFIS's five-layered architecture but also adapt to the upcoming requirements of DNFS research for network interpretability and high performance with the assistance of PyTorch and Numpy frameworks.

Table 2 Overview of the software for ANFIS

Name	Language	Library	MFt	Learning strategy	Sklearn	Release
ANFIS-C[1]	C	N/A	4	Hybrid/Online	No	1993
ANFIS-Vignette[21]	R	N/A	4	Hybrid/Online	No	2012
Matlab-ANFIS[10]	Matlab	N/A	12	Hybrid/Online	No	2023
ANFIS-PyTorch[11]	Python 3	PyTorch	5	Hybrid/Online	No	2019
ANFIS-Numpy[12]	Python 3	Numpy[30]	3	Hybrid	No	2020
ANFIS-PSO[13]	Python 3	Numpy	3	PSO	No	2021
Scikit-ANFIS	Python 3	PyTorch+Numpy	12	Hybrid/Online	Yes	2023

Table 3 The difference between Scikit-ANFIS and other deep neural fuzzy methods for two common tasks: regression and classification

Name	Language	MFt	Layers	Optimization method	Tasks	Sklearn	Release
Neuro-Fuzzy[32]	C	1	4	Gradient Descent	Classification	No	1993
DNFC[33]	Matlab	1	8	Gradient Descent	Classification	No	2020
MBGD-RDA[34]	Python 3	1	5	MBGD+AdaBound	Regression	No	2020
TSK-MBGD-UR-BN[35]	Python 3	1	6	MBGD+AdaBound+BN+UR	Classification	No	2020
FCM-RDpA[36]	Matlab	1	5	MBGD+Powerball+AdaBelief	Regression	No	2021
HTSK-LN-ReLU[37]	Python 3	1	7	MBGD+Adam+LU+ReLU	Regression	Yes	2022
PyTSK[18]	Python 3	2	6	MBGD+Adam	Classification	Yes	2022
Scikit-ANFIS	Python 3	12	5	ANFIS	Regression+Classification	Yes	2023

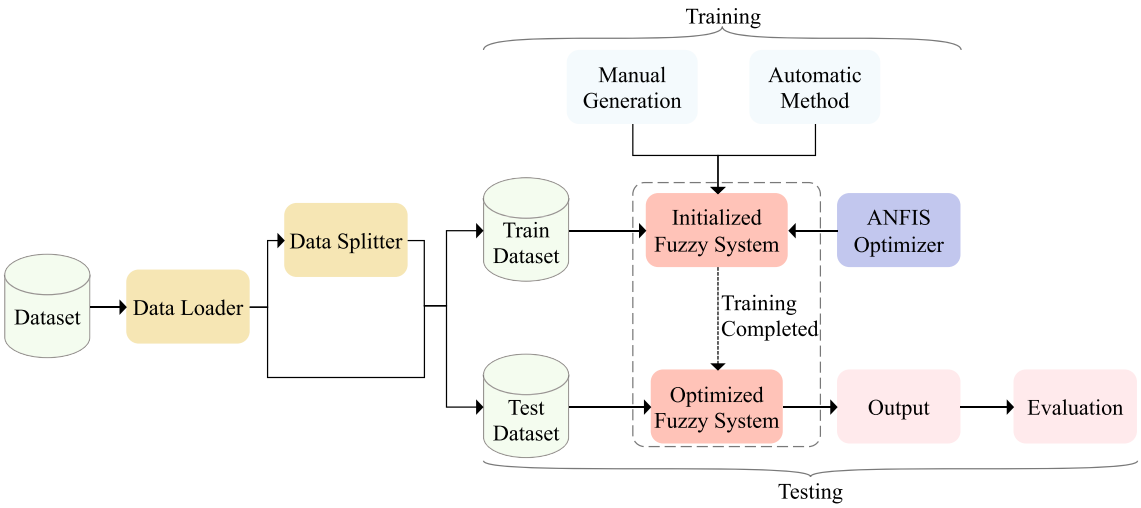


Fig. 3 Overview of Scikit-ANFIS architecture

4 Scikit-ANFIS

4.1 Architecture Overview

The diagram in Fig. 3 illustrates the overall structure of our Scikit-ANFIS. Scikit-ANFIS employs the data loader module to read data from the dataset, which is then divided into train, and test datasets by the data splitter module. These datasets are sent to the generated fuzzy system for training. Alternatively, the initialized fuzzy system can directly read the train, and test data from the dataset by the data loader module and train itself accordingly. To create a fuzzy system for predictive tasks, Scikit-ANFIS provides two options: the manual generation module can be utilized to define and generate a fuzzy system, or the automatic method module can automatically generate an ANFIS fuzzy system by default without definition. For training, Scikit-ANFIS uses the ANFIS optimizer module to train the initialized fuzzy system. Once training is completed, the optimized fuzzy system is selected and tested with data. The evaluation module then examines the test outputs to formulate a report.

Scikit-ANFIS[1] is also implemented in the Python 3 language, which mainly includes two dependencies such as PyTorch [19] and Numpy [30]. Our Scikit-ANFIS currently supports the following primary functions: (i) The twelve types of membership functions such as Gaussian, bell, triangular, and others. (ii) Fuzzy sets written in natural language, and complex fuzzy rules with logical operators AND, OR, and NOT. (iii) Two training strategies of ANFIS, namely hybrid and online. (iv) Automatic

[1] The code for Scikit-ANFIS, the associated cases, and the user guide will be publicly available at https://github.com/hudscomdz/scikit-anfis. Scikit-ANFIS can be installed by using the following command: pip install skanfis.

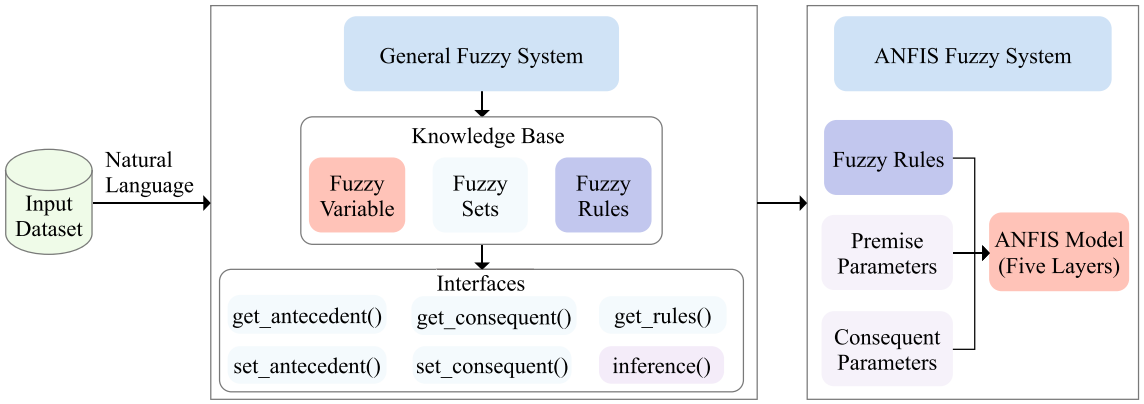


Fig. 4 The illustration of the manual generation method for the general fuzzy system

generation and training of the ANFIS fuzzy system. (v) A uniform structure such as the *fit*() and *predict*() functions to provide the same interface as scikit-learn.

4.2 Implementation Details

4.2.1 Manual Generation of a General Fuzzy System

Considering that Simpful [27] is already open source and general-purpose fuzzy system software developed in Python, our implementation also makes use of some existing components as defined by Simpful for efficient development. As depicted in Fig. 4, the difference between the manual generation method for general fuzzy system and Simpful is mainly that the former can interact with the ANFIS optimizer in Scikit-ANFIS, which can not only realize the ANFIS training of the fuzzy system but also return the trained results to the fuzzy system to generate new output. However, the latter can only generate an output after passing the received input data through the fuzzy knowledge base without any model training operation.

Similarly to Simpful, after receiving the natural language information, our manual generation method automatically parses the fuzzy variables, fuzzy sets, and fuzzy rules, creating Scikit-ANFIS's fuzzy system object. The fuzzy rule uses Takagi and Sugeno's fuzzy if-then rule [1], and its natural language description supports the commonly used fuzzy operators like AND, OR, and NOT, as detailed in [27]. When the input dataset is fed into the fuzzy system object created by the manual generation method, it can conduct fuzzy reasoning through an interface namely *inference*(), and provide output results. Additionally, the object can communicate with the ANFIS model in Scikit-ANFIS through five interfaces: *get_antecedent*(), *get_consequent*(), *get_rules*(), *set_antecedent*(), and *set_consequent*(). By providing the first three interfaces, Scikit-ANFIS can send the antecedent parameters, fuzzy rules, and consequent parameters of the fuzzy system object to the ANFIS model for training. After the training is finished, the last two interfaces accurately return the well-trained ANFIS model's parameters to the fuzzy system object, enabling it to perform precise fuzzy inference.

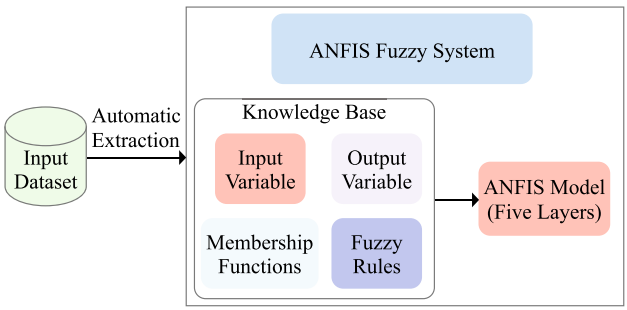


Fig. 5 The illustration of the automatic method for ANFIS fuzzy system

4.2.2 Automated Method to Initialise an ANFIS Fuzzy System

To facilitate users to create the ANFIS model, our Scikit-ANFIS designs and implements an automatic method for ANFIS fuzzy system, which shares the same *scikit_anfis*() class with the ANFIS optimizer module. Figure 5 illustrates the functional diagram of the method, which can automatically generate an ANFIS fuzzy system object including a knowledge base and ANFIS model for the input dataset. The input data set is used to automatically extract a knowledge base consisting of input variables, output variables, membership functions, and fuzzy rules. This leads to

the generation of an ANFIS model that adheres to the strict requirements of the type-3 fuzzy inference system as proposed by the original paper [1]. The rule base follows fuzzy if-then rules and can be effortlessly mapped to an equivalent ANFIS architecture [1]. The resulting ANFIS model comprises a five-layer neural network structure, as illustrated in Fig. 1, and provides a robust fuzzy inference system.

4.2.3 ANFIS as an Optimizer

ANFIS optimizer as an optimization module also utilizes the *scikit_anfis*() class to help the initialized fuzzy system to be trained more efficiently. The ANFIS optimizer takes the training set as input and uses forward propagation and cost function to calculate the total loss of the ANFIS neural network generated. The *forward*() method is used for forward propagation, built on the PyTorch framework. The default training algorithm used is hybrid learning, with online learning available as an alternative. Then, it updates all the antecedent and consequent parameters in the model through the backpropagation process. This entire process is the training process for the five-layer ANFIS model, and the number of times the model is trained is related to the 'epoch' hyperparameter. The optimizer used to update the parameter through backpropagation is usually related to the 'optimizer' hyperparameter of the model. Our Scikit-ANFIS has implemented various optimizers based on gradient descent, including Adam [39], SGD [43], Rprop [44], L-BFGS [45], Adadelta [46], and Adagrad [47], with Adam being the default.

Once the training of the ANFIS model is completed, all parameters of the current model with minimum loss can be saved to the local model file 'tmp.pkl'. This saved model file can be later used to continue training or testing, which can be very useful. To ensure that a manually created general fuzzy system and a trained ANFIS model are consistent with each other, it is possible to transfer the premise and consequent parameters from the ANFIS model to the fuzzy system. This can be done by using two interfaces such as *set_antecedent*() and *set_consequent*(), which are explicitly called as shown in Fig. 4. Moreover, the optimized fuzzy system can give fuzzy inference results based on the test data.

4.2.4 Scikit-ANFIS

The implementation of our Scikit-ANFIS is detailed in Algorithm 1. The input, training, and test datasets are denoted as (X, Y), (X_{train}, Y_{train}), and (X_{test}, Y_{test}) respectively, as illustrated in Fig. 3. The main procedure of Scikit-ANFIS encapsulated between lines 1 and 15, comprises three key elements. Firstly, in line 1, a general fuzzy system object *fs* is manually created. Following that, lines 2 and 3 specify fuzzy sets, input variables, fuzzy rules, and output variables for *fs*. In line 4, we check whether *fs* is empty. If it is not, we use the antecedent and consequent parameters along with rules from *fs* to represent a 5-layer ANFIS model called *anfis_layers* in line 5. At the same time, we create a *scikit_anfis* object based on *anfis_layers* in line 6, also specifying the maximum number of epochs for training (*max_epoch*), the training strategy (*hybrid*), and the task type (*label*), and the *optimizer*. By default, *max_epoch* is set to 10, *hybrid* is set to True, indicating the use of the hybrid training strategy, *label* is set to "r", indicating that the model is intended for regression tasks, and *optimizer* is set to the gradient descent method namely "Adam". Secondly, if *fs* is empty, we automatically create an ANFIS fuzzy system from lines 7 to 10. Line 8 is the 5-layer ANFIS model *anfis_layers* derived from the input dataset (X, Y), and line 9 generates a *scikit_anfis* object and specifies its associated parameters, such as *max_epoch*, *hybrid*, *label*, and *optimizer*. Thirdly, the most crucial step involves the implementation of the ANFIS optimizer from lines 11 to 15. At each epoch in the loop, we feed the training dataset (X_{train}, Y_{train}) into the *scikit_anfis* object to complete the forward pass from layer 1 to layer 5 in its ANFIS model. During this process, we update the consequent parameters and obtain the final output $\hat{Y}$. We then compute the RMSE loss between the predicted $\hat{Y}$ and the training target Y_{train}, followed by updating the premise parameters of the ANFIS model in its backward process.

When the ANFIS optimizer has completed training the model, confidently use *fs* for fuzzy inference of test data from lines 16 to 20. Return the premise and consequent parameters to *fs* using the *set_antecedent*() and *set_consequent*() interfaces. Then, directly call the *inference*() function of *fs* to complete the fuzzy system reasoning. Alternatively, we can opt for the more convenient *scikit_anfis* object in line 21 for fuzzy inference. This method uses the test input data X_{test} to generate the predicted output Y_{pred} through fuzzy reasoning of the ANFIS model. This option has been used in subsequent cases in this paper. It is important to note that during testing, there is no need to provide the test target value Y_{test} to the trained *scikit_anfis* object since all of its parameters are already fixed. Finally, in the last line, we compare and evaluate the fuzzy inference result Y_{pred} with the test target value Y_{test}.

Algorithm 1 Pseudo Code for Scikit-ANFIS Implementation

Require: Input dataset (X, Y), train dataset (X_{train}, Y_{train}), and test dataset (X_{test}, Y_{test}) as in Fig. 3
$fs \Leftarrow FS()$ ▷ create a general fuzzy system manually.
$fs.add$(fuzzy sets), $fs.add$(input variables)
$fs.add$(fuzzy rules), $fs.add$(output variables)
if $fs \neq empty$ **then** ▷ create a scikit-anfis object based on fs.
  $anfis\_layers \Leftarrow \{fs.get\_antecedent(), fs.get\_consequent(), fs.get\_rules()\}$
  $scikit\_anfis \Leftarrow \{anfis\_layers, max\_epoch, hybrid, label, optimizer\}$
else ▷ create an anfis fuzzy system automatically.
  $anfis\_layers \Leftarrow \{(X, Y)\}$
  $scikit\_anfis \Leftarrow \{anfis\_layers, max\_epoch, hybrid, label, optimizer\}$
end if
for $epoch = 1, 2, \cdots, max\_epoch$ **do** ▷ anfis optimizer begins.
  $\hat{Y}$, $update$(consequent parameters) $\Leftarrow scikit\_anfis.forward(X_{train}, Y_{train})$
  $loss \Leftarrow ComputeRMSELoss(\hat{Y}, Y_{train})$
  $update$(premise parameters) $\Leftarrow loss.backward()$
end for ▷ anfis optimizer ends.
if $fs \neq empty$ **then** ▷ fuzzy system reasoning for test data.
  $fs.set\_antecedent$(premise parameters)
  $fs.set\_consequent$(consequent parameters)
  $Y_{pred} \Leftarrow fs.inference(X_{test})$
end if
$Y_{pred} \Leftarrow scikit\_anfis.forward(X_{test})$ ▷ anfis fuzzy reasoning for test data.
$evaluate(Y_{pred}, Y_{test})$ ▷ evaluate fuzzy inference results.

Table 4 Summary of the three regression datasets and one classification dataset

Dataset Name	Abbr	Task type	Features	Number of Samples
Restaurant Tipping Problem[48]	Tip	Regression	2	441
Two-input Nonlinear Function[1]	Sinc	Regression	2	121
Predict Chaotic Dynamics Problem[1]	PCD	Regression	4	1000
Iris Classifier[49]	Iris	Classification	4	150

4.3 Code Overview using Scikit-ANFIS

Our Scikit-ANFIS can be used in two different manners. The first method demonstrates how to optimize a manually created TSK fuzzy system using Scikit-ANFIS in Listing 1. Scikit-ANFIS implements an ANFIS model based on a general TSK fuzzy system object in lines 4 to 5 and then uses the scikit-learn interface to train and test the model in lines 6 to 7. The ANFIS model is trained on the training data using the *fit*() function. Following this, the trained ANFIS model is tested using the *predict*() function with the test data.

Listing 1 The code demo of the Sklearn interface for optimizing the general TSK fuzzy system using Scikit-ANFIS.

```
1 from skanfis.fs import *
2 from skanfis import *
3
4 fs = FS() # Create a general TSK fuzzy system
5 model = scikit_anfis(fs) # Create an ANFIS model
      utilizing fs
6 model.fit(X_train, y_train) # Model training
7 y_pred = model.predict(X_test) # Model testing
```

Scikit-ANFIS can be used to automatically generate and optimize a TSK fuzzy system by creating an ANFIS model, as demonstrated in line 3 of Listing 2. The ANFIS model created is not only a TSK system but also adopts scikit-learn compatible interface design, which can easily complete the training and testing of the TSK fuzzy model by

calling the *fit*() and *predict*() functions in line 4 and 5, respectively.

Listing 2 The code demo of the Sklearn interface for generating and optimizing a TSK fuzzy system using Scikit-ANFIS.

```
1 from skanfis import *
2
3 model = scikit_anfis() # Create an ANFIS model
4 model.fit(train_data) # Model training
5 y_pred = model.predict(X_test) # Model testing
```

5 Experimentation

5.1 Experimental Setup

For a fair comparison and consistency with previous ANFIS implementations (Matlab-ANFIS [10], ANFIS-PyTorch [11], and ANFIS-Numpy [12]), this section reports results and discussions as a result of experimentation over both regression and classification tasks. Thus, the first regression dataset is the restaurant tipping problem [48] taken from the Matlab file repository. The next two regression datasets are from Jang's literature [1], using ANFIS to model a nonlinear sinc equation and predict future values of a chaotic time series, respectively. Finally, we use the ANFIS model to train and test the popular iris benchmark dataset [49], which is essentially to a three-class classification problem. The details regarding task type, features (i.e. inputs), and the number of samples in all four datasets are presented in the following Table 4. It should be noted that we do not pre-process data or conduct any hyperparameter tuning for a fair and straightfoward comparison.

5.2 Case Studies

In this section, we report reports on four studies including three regression problems and one classification problem, with major Python code to demonstrate how Scikit-ANFIS can be applied in practice.

5.2.1 Restaurant Tipping Problem

The restaurant tip problem is to calculate a fair tip ratio of the total bill according to the service and food quality of a restaurant. Listing 3 shows an example of Scikit-ANFIS code to manually define a general TSK fuzzy inference system through natural language, and then train and test the fuzzy inference system (FIS) with a Scikit-ANFIS object, based on the Matlab data file 'data_Tip_441.mat' with two inputs and one output for a total of 441 samples.

Listing 3 A TSK FIS example for restaurant tipping problem, implemented in Scikit-ANFIS

```
1 from skanfis.fs import *
2 from skanfis import *
3 from scipy.io import loadmat
4 from sklearn.model_selection import
      train_test_split
5 from sklearn.metrics import mean_squared_error
6
7 # Load Tip data
8 tip_data = loadmat('data_Tip_441.mat')['Tip_data
      ']
9 # Split the data using the test_size argument in
       training(50%) and test(50%) set
10 train_data, test_data = train_test_split(
      tip_data, test_size=0.5, random_state=42)
11 y_test = test_data[:,-1]
12 X_test = test_data[:,:-1]
13
14 # Create a TSK fuzzy system object by default
15 fs = FS()
16
17 # Define fuzzy sets and linguistic variables
18 S_1 = TriangleFuzzySet(a=0, b=0, c=5, term="poor
      ")
19 S_2 = TriangleFuzzySet(a=0, b=5, c=10, term="
      good")
20 S_3 = TriangleFuzzySet(a=5, b=10, c=10, term="
      excellent")
21 fs.add_linguistic_variable("Service",
      LinguisticVariable([S_1, S_2, S_3]))
22 F_1 = TriangleFuzzySet(a=0, b=0, c=10, term="
      rancid")
23 F_2 = TriangleFuzzySet(a=0, b=10, c=10, term="
      delicious")
24 fs.add_linguistic_variable("Food",
      LinguisticVariable([F_1, F_2]))
25
26 # Define crisp outputs for small and average tip
27 fs.set_crisp_output_value("small", 5)
28 fs.set_crisp_output_value("average", 15)
29 # Define function for generous tip (2*food score
       + 3*service score + 5%)
30 fs.set_output_function("generous", "2*Food+3*
      Service+5")
31
32 # Define fuzzy rules
33 R1 = "IF (Service IS poor) OR (Food IS rancid)
      THEN (Tip IS small)"
34 R2 = "IF (Service IS good) THEN (Tip IS average)
      "
35 R3 = "IF (Service IS excellent) OR (Food IS
      delicious) THEN (Tip IS generous)"
36 fs.add_rules([R1, R2, R3])
37
```

In line 8 of Listing 3, the data 'tip_data' is loaded from the Matlab file using the *loadmat*() command in the scipy.io package. Then in line 10, using the *train_test_split*() command from the sklearn package, the

above 'tip_data' is split in half randomly to get the training data and the test data. Lines 11 and 12 extract the output 'y_test' and input 'X_test' from the test data respectively for performance evaluation after the test results. In line 15 a TSK fuzzy system object is created by default. The language variable 'Service' and its three triangular fuzzy sets, 'poor', 'good', and 'excellent', are defined in lines 18 to 21, with values ranging from 0 to 10. Similarly, lines 22 to 24 define the language variable 'Food', which has two triangular fuzzy sets, 'rancid' and 'delicious'. The output crisp values of 'small' and 'average' are set to 5% and 15%, respectively, in lines 27 and 28. Meanwhile, the output value of a 'generous' tip is defined in line 30 as a linear function that depends on the scores for 'Food' and 'Service'. Once the fuzzy rules are defined in lines 33 to 36, a general TSK fuzzy system is created, and then, as shown in line 39, a Scikit-ANFIS object needs to be generated based on it to implement ANFIS training of this fuzzy system. Line 39 uses the default setting for the epoch number and the training strategy of our Scikit-ANFIS object, that is, the epoch is 10 and the training strategy is hybrid learning. Then, ANFIS model training was conducted on the training data 'train_data' using the *fit*() command in line 40. The *predict*() command is used to perform model prediction on the input 'X_test' in the test data to obtain the prediction result 'y_pred' in line 41. Finally, the root mean square error (RMSE) value between 'y_pred' and the actual result 'y_test' of the test data was calculated in line 42.

5.2.2 Two-Input Nonlinear Function

The two-input nonlinear function is the sinc equation, expressed as follows:

$$z = sinc(x, y) = (sin(x)/x) \times (sin(y)/y) \tag{7}$$

where x and y are two input variables, and z is the output variable. According to [1], x and y were selected from the range of -10 to 10 with an interval of 2, and 121 data pairs could be obtained as training samples. Listing 4 shows how to generate Scikit-ANFIS code to train an ANFIS model with 16 fuzzy rules and 4 bell membership functions assigned to each input variable, as well as the test results.

Listing 4 A TSK FIS example for two-input non-linear function, implemented in Scikit-ANFIS

```
1  from skanfis.fs import *
2  from skanfis import *
3  from pandas import read_csv
4  from sklearn.model_selection import
       train_test_split
5  from sklearn.metrics import mean_squared_error
6
7  # Load Sinc data
8  sinc_data = read_csv("data_sinc_121.csv")
9  # Split the data using the test_size argument in
        training(60%) and test(40%) set
10 train_data, test_data = train_test_split(
       sinc_data, test_size=0.4, random_state=42)
11 y_test = test_data.pop('z')
12
13 # Create a TSK fuzzy system object by default
14 fs = FS()
15
16 # Define fuzzy sets and linguistic variables
17 S_1 = BellFuzzySet(a=3.33330, b=2, c=-10, term="
       mf0")
18 S_2 = BellFuzzySet(a=3.33330, b=2, c=-3.33330,
       term="mf1")
19 S_3 = BellFuzzySet(a=3.33330, b=2, c=3.33330,
       term="mf2")
20 S_4 = BellFuzzySet(a=3.33330, b=2, c=10, term="
       mf3")
21 fs.add_linguistic_variable("x0",
       LinguisticVariable([S_1, S_2, S_3, S_4]))
22 F_1 = BellFuzzySet(a=3.33330, b=2, c=-10, term="
       mf0")
23 F_2 = BellFuzzySet(a=3.33330, b=2, c=-3.33330,
       term="mf1")
24 F_3 = BellFuzzySet(a=3.33330, b=2, c=3.33330,
       term="mf2")
25 F_4 = BellFuzzySet(a=3.33330, b=2, c=10, term="
       mf3")
26 fs.add_linguistic_variable("x1",
       LinguisticVariable([F_1, F_2, F_3, F_4]))
27
28 # Define output functions and crisp value
29 for i in range(15):
30     fs.set_output_function("sinc_x_y"+str(i), "
       2*x0+2*x1+1")
31 fs.set_crisp_output_value("sinc_x_y15", 1)
32
33 # Define fuzzy rules
34 R1 = "IF (x0 IS mf0) AND (x1 IS mf0) THEN (y0 IS
        sinc_x_y0)"
35 # We've shown the full rule list in Listing 3,
       but owing to space limitations, we are
       omitting most rules from here
36 :
37 R15 = "IF (x0 IS mf3) AND (x1 IS mf2) THEN (y0
       IS sinc_x_y14)"
38 R16 = "IF (x0 IS mf3) THEN (y0 IS sinc_x_y15)"
39 fs.add_rules([R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11
       ,R12,R13,R14,R15,R16])
40
41 # Create a Scikit-ANFIS object based on fs
42 model = scikit_anfis(fs, description="Sinc",
       epoch=250, hybrid=True)
43 model.fit(train_data) # Model training
44 y_pred = model.predict(test_data) # Prediction
45 print("Test RMSE: ", mean_squared_error(y_pred,
       y_test, squared=False))
```

Line 8 of Listing 4 shows that users can load data 'sinc_data' from csv format file 'data_sinc_121.csv' using the *read_csv*() command in the pandas package. Similar to Listing 3, the *train_test_split*() command from the sklearn package is used in line 10 to split the above 'sinc_data' randomly in 60:40 proportion into 'train_data' and 'test_data' set. Considering that 'test_data' belongs to the DataFrame type in the pandas package, 'y_test' can be extracted from 'test_data' by *pop*() command in line 11 according to the variable name 'z', and there is no property 'z' in 'test_data' after extraction, only two input variables. Line 14 first creates a TSK fuzzy system object, and then the user defines four bell fuzzy sets for each of the two language variables such as 'x0' and 'x1' from lines 17 to 26. In lines 29 to 30, a total of 15 output variable values from 'sinc_x_y0' to 'sinc_x_y14' are defined as linear functions that depend on the scores of the input language variables 'x0' and 'x1'. The crisp value of the last output variable 'sinc_x_y15' is defined as one in line 31. After the fuzzy rule set is added in lines 34 to 39 (owing to space limitations, the paper are omitting most rules from here), a general-purpose TSK fuzzy system is complete. In line 42, a Scikit-ANFIS object is created based on the above fuzzy system with an epoch of 250 and a learning strategy of hybrid. Next, the ANFIS model is trained on the training data 'train_data' using the *fit*() command in line 43. Then in line 44, the *predict*() command is used to model the test data 'test_data', and the prediction result is 'y_pred'. Finally, the RMSE value between the predicted result 'y_pred' and the actual result 'y_test' is calculated in line 45.

5.2.3 Predict Chaotic Dynamics Problem

The predict chaotic dynamic problem comes cirely from [1], whose goal is to predict the future value of chaotic time series by creating the ANFIS model. In [1], by obtaining time series values at each integer point, the time series values corresponding to four consecutive time points with an interval of six are selected as the input, the value to the fifth point as output, and then 1000 input–output data pairs can be extracted, which are formally expressed as follows:

$$[\text{x0, x1, x2, x3, y0}] = [v(t-18), v(t-12), v(t-6), v(t), v(t+6)] \tag{8}$$

where 'x0', 'x1', 'x2', and 'x3' denote the time series value of the four inputs respectively, 'y0' denotes the time series value of the output, and $v(t)$ denotes the time series value at the time point t.

Listing 5 shows how to manually generate a general fuzzy system with 16 fuzzy rules and 2 Gaussian membership functions for each of 4 input variables, then create a Scikit-ANFIS object based on this, and then complete hybrid training and testing of ANFIS model. In lines 8 and 9, we load all the input data 'X' and the real output data 'y' from the 'data_PCD_1000.txt' data file, respectively, using the *loadtxt*() command with its 'usecols' parameter from the numpy package. As in Listing 3 and 4, the *train_test_split*() command from the sklearn package is also used in line 11 of Listing 5 to randomly split data 'X' into 'X_train' and 'X_test' according to the specified 70:30 proportion, and data 'y' into 'y_train' and 'y_test' in the same proportion. In line 14, the empty TSK fuzzy system object is created for the problem. From lines 17 to 28, the fuzzy sets and Gaussian membership functions of the four input variables such as 'x0', 'x1', 'x2', and 'x3' are defined. Lines 31 to 36 show how the 16 fuzzy rules are defined, and again most of the rules are omitted due to space limitations. Since the antecedents of the first 15 rules use all four input language variables, the loops of lines 39 to 40 define the output variables of the first 15 rules in the form of linear functions. However, line 41 specifically defines the output variable in the last rule, 'R16', as a linear function of zero times the fourth input language variable 'x3', because the antecedent of the last rule uses only those three input language variables. In line 44, based on the newly created fuzzy system 'fs', a Scikit-ANFIS model is generated that initializes 500 epochs and a hybrid training strategy. Line 45 uses the *fit*() command line to feed the input data 'X_train' and output data 'y_train' to the Scikit-ANFIS model for hybrid training. After training, the *predict*() command in line 46 is used to predict the input data 'X_test' in the test set, and the prediction result 'y_pred' is obtained. Finally, the RMSE between 'y_pred' and 'y_test' is calculated in line 47 using the *mean_squared_error*() command from the sklearn package.

Listing 5 A TSK FIS example for predict chaotic dynamics problem, implemented in Scikit-ANFIS

```
from skanfis.fs import *
from skanfis import *
from numpy import loadtxt
from sklearn.model_selection import
    train_test_split
from sklearn.metrics import mean_squared_error

# Load PCD data
X = loadtxt('data_PCD_1000.txt', usecols
    =(0,1,2,3))
y = loadtxt('data_PCD_1000.txt', usecols=(4))
# Split the data using the test_size argument in
     training(70%) and test(30%) set
X_train,X_test,y_train,y_test = train_test_split
    (X,y,test_size=0.3,random_state=42)

# Create a TSK fuzzy system object by default
fs = FS()

# Define fuzzy sets and linguistic variables
S_1 = GaussianFuzzySet(mu=2, sigma=0.4270, term=
    "mf0")
S_2 = GaussianFuzzySet(mu=2, sigma=1.3114, term=
    "mf1")
fs.add_linguistic_variable("x0",
    LinguisticVariable([S_1, S_2]))
S_3 = GaussianFuzzySet(mu=2, sigma=0.4270, term=
    "mf0")
S_4 = GaussianFuzzySet(mu=2, sigma=1.3114, term=
    "mf1")
fs.add_linguistic_variable("x1",
    LinguisticVariable([S_3, S_4]))
F_1 = GaussianFuzzySet(mu=2, sigma=0.4270, term=
    "mf0")
F_2 = GaussianFuzzySet(mu=2, sigma=1.3114, term=
    "mf1")
fs.add_linguistic_variable("x2",
    LinguisticVariable([F_1, F_2]))
F_3 = GaussianFuzzySet(mu=2, sigma=0.4270, term=
    "mf0")
F_4 = GaussianFuzzySet(mu=2, sigma=1.3114, term=
    "mf1")
fs.add_linguistic_variable("x3",
    LinguisticVariable([F_3, F_4]))

# Define fuzzy rules
R1 = "IF (x0 IS mf0) AND (x1 IS mf0) AND (x2 IS
    mf0) AND (x3 IS mf0) THEN (y0 IS chaotic0)"
# We've shown the full rule list in Listing 3,
    but owing to space limitations, we are
    omitting most rules from here
.
.
.
R15 = "IF (x0 IS mf1) AND (x1 IS mf1) AND (x2 IS
     mf1) AND (x3 IS mf0) THEN (y0 IS chaotic14
    )"
R16 = "IF (x0 IS mf1) AND (x1 IS mf1) AND (x2 IS
     mf1) THEN (y0 IS chaotic15)"
fs.add_rules([R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11
    ,R12,R13,R14,R15,R16])

# Define output functions
for i in range(15):
    fs.set_output_function("chaotic"+str(i), "2*
    x0+2*x1+2*x2+2*x3+1")
fs.set_output_function("chaotic15", "2*x0+2*x1
    +2*x2+0*x3+1")

# Create a Scikit-ANFIS fuzzy system object
    based on fs
model = scikit_anfis(fs, description="PCD",
    epoch=500, hybrid=True)
model.fit(X_train, y_train) # Model training
y_pred = model.predict(X_test) # Prediction
print("Test RMSE: ", mean_squared_error(y_pred,
    y_test, squared=False))
```

5.2.4 *Building a Classifier for Iris problem*

The Iris dataset is a classic multi-classification dataset [49] with 150 data samples in 3 classes, each corresponding to one species of iris plant, and 50 samples. Listing 6 below shows how our developed Scikit-ANFIS automatically generates ANFIS model for training and testing on the above iris dataset. First, the *load_iris*() command in scikit-learn is used to quickly load and return the iris dataset without downloading any files from an external website, as shown in line 7 of Listing 6. In line 9, the *train_test_split*() command is used again to randomly split the iris input data 'iris.data' into 'X_train' and 'X_test' in the specified 80:20 ratio, and output target data 'iris.target' into 'y_train' and 'y_test' in the same ratio. The Scikit-ANFIS fuzzy system object is created for the classification problem in line 12, taking 'iris.data' as an argument to automatically extract all four input variables and assign two Gaussian membership functions to each by default, and setting the epoch to 100, and using the hybrid training method (hybrid=True). In particular, the label parameter in line 12 is set to 'c', indicating that this is an ANFIS model to solve the classification problem, while the default value is 'r', indicating the regression model. Line 13 uses the *fit*() command line to feed the input data 'X_train' and output data 'y_train' to the Scikit-ANFIS model for classification training. After the completion of the training, the *predict*() command in line 14 is used to predict the input data 'X_test' in the test set, and the predicted classification result is 'y_pred'. Subsequently, the accuracy of the model between 'y_pred' and 'y_test' is calculated and printed in line 15 using the *accuracy_score*() command from the sklearn package.

Listing 6 A TSK FIS example for iris classifier, implemented in Scikit-ANFIS

```
from skanfis import *
from sklearn.datasets import load_iris
from sklearn.model_selection import
    train_test_split
from sklearn.metrics import accuracy_score

# Load Iris data
iris = load_iris()
# Split the data using the test_size argument in
     training(80%) and test(20%) set
X_train, X_test, y_train, y_test =
    train_test_split(iris.data, iris.target,
    test_size=0.2, random_state=42)

# Create a Scikit-ANFIS fuzzy system object
model = scikit_anfis(iris.data,description="Iris
    ",epoch=100,hybrid=True,label="c")
model.fit(X_train, y_train) # Model training
y_pred = model.predict(X_test) # Prediction
print("Test Accuracy: ", accuracy_score(y_pred,
    y_test))
```

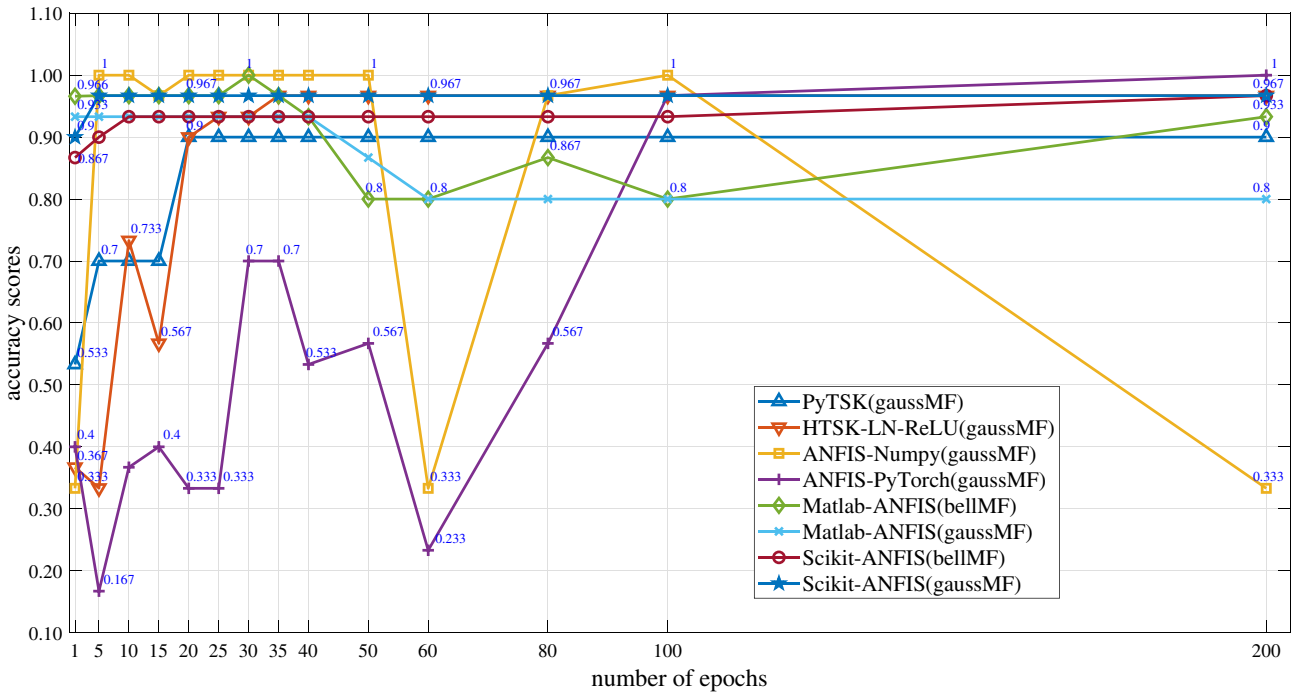


Fig. 6 Comparison of accuracy scores for several software methods in terms of different epochs using the Iris dataset

Subsequently, Fig. 6 shows a comparison of output accuracy scores produced by various software methods such as PyTSK [18], HTSK-LN-ReLU [37], ANFIS-Numpy [12], ANFIS-PyTorch [11], Matlab-ANFIS [10], and our Scikit-ANFIS with different epoch sizes under the same Iris dataset in Table 4. For a fair comparison, all the methods generate fuzzy systems two membership functions for each of the four input variables, resulting in 16 fuzzy rules. The Iris dataset is then divided into a fixed 80:20 proportion for training and testing purposes (refer to Listing 6: lines 7-9). It is important to note that the training set and test set remain the same across all the methods. In addition, in order to explain the influence of the membership function of input variables on a fuzzy system, the text in parentheses after each software name indicates which membership function is used to construct the fuzzy system. For example, the 'PyTSK(guassMF)' entry and 'Matlab-ANFIS(bellMF)' one in the figure indicate that the PyTSK software uses the Gaussian membership function, and the Matlab-ANFIS software uses the bell membership function, respectively. PyTSK and HTSK-LN-ReLU used their own gradient descent methods to train the dataset, while other ANFIS methods used a hybrid approach. The experiments were repeated 10 times per group for accuracy.

As shown in Fig. 6, the accuracy score in the PyTSK increases linearly when the number of epochs increases from one to two hundred, ranging from 0.533 to 0.9. Although the accuracy score of HTSK-LN-ReLU still maintains the overall linear increase from 0.333 to 0.967, the accuracy score increases and decreases with different epochs, which has a slight fluctuation. With the increasing number of epochs, the accuracy scores of the ANFIS-Numpy and ANFIS-PyTorch fluctuate between 0.333 and 1.0 and between 0.167 and 1.0, respectively. In addition, Matlab-ANFIS(bellMF) exhibits a small change from 0.8 to 1.0, and the accuracy of Matlab-ANFIS(gaussMF) decreases linearly from 0.933 to 0.8 as the number of epochs increases. In contrast, the accuracy score of our Scikit-ANFIS increases strictly linearly as the number of epochs grows larger, with minimal variation, where Scikit-ANFIS(bellMF)'s score increases from 0.867 to 0.967 and Scikit-ANFIS(gaussMF)'s from 0.9 to 0.967. The relatively stable accuracy score in our Scikit-ANFIS is particularly beneficial in real scenarios where the ANFIS model is applied. This makes our Scikit-ANFIS more efficient and adaptable to handle different ANFIS training configurations of fuzzy inference systems.

5.3 Using Sklearn-supported Cross-validation for Scikit-ANFIS

To further demonstrate the effectiveness of our Scikit-ANFIS software tool, we completed the 10-fold cross-validation ('10-CV' for short) experimental comparison of various software for the above four datasets using *cross_val_score*() command in Scikit-learn. '10-CV' refers to dividing the data into 10 equal folds of smaller

Table 5 Summary of 10-fold cross-validation ('10-CV' for short) experiments of various software for ANFIS or DNFS under four different datasets at the same setting of 100 epochs and Gaussian membership functions

Methods	Regresssion(RMSE↓)			Classificaion(Acc↑)
	Tip	Sinc	PCD	Iris
Matlab-ANFIS(hybrid)[10]	**8e-7±3e-8**	0.127±0.016	**0.002±5e-5**	0.875±0.018
ANFIS-PyTorch(hybrid)[11]	3e-6±1e-6	0.236±0.082	0.030±0.004	0.933±0.067
ANFIS-Numpy[12]	0.171±0.510	0.283±0.326	0.194±0.200	0.860±0.156
Scikit-ANFIS(hybrid)	1e-6 ± 1e-7	**0.109±0.068**	0.041±0.007	**0.952±0.054**
PyTSK[18]	n/a	n/a	n/a	0.301±0.098
HTSK-LN-ReLU[37]	0.945±0.451	0.859±0.643	1.100±0.065	0.346±0.105
Matlab-ANFIS(online)[10]	0.426±0.013	0.119±0.015	0.103±0.001	**0.968±0.019**
ANFIS-PyTorch(online)[11]	2.043±1.516	0.141±0.075	0.403±0.006	0.493±0.326
Scikit-ANFIS(online)	**0.382±0.187**	**0.101±0.069**	**0.044±0.005**	0.960±0.044

Each value in the experiment is in the form of the mean score ± standard deviation. Except for the Iris data set in the last column of the following table, which is the accuracy (Acc) score computed at each '10-CV' iteration, the data sets in the other three columns such as Tip, Sinc, and PCD are computed for the root mean squared error (RMSE) score. ↓ indicates that the smaller the value is, the better the performance, while ↑ indicates that the larger the value is, the better the performance. 'n/a' indicates 'not applicable'

sets, then training the model using 9 of the folds as training data, and testing the resulting model on the remaining 1 fold of the data for computing a performance measure such as root mean squared error and accuracy. The four datasets in the '10-CV' experiment are from Table 4, which can be divided into three regression sets (Tip, Sinc, PCD) and one classification set (Iris) according to task type. In the experiment, the processing details of regression and classification data sets are different. The main difference lies in the setting of 'scoring' parameters for defining model evaluation rules in the *cross_val_score*() command. As for regression, 'neg_mean_squared_error' has been specified as the 'scoring' parameter shown in line 4 of Listing 7, while for classification, the 'scoring' parameter is set to 'accuracy' shown in line 4 of Listing 8. In addition, line 4 of Listing 7 or 8 aims to conduct '10-CV' experiments on the data set composed of input data 'X' and output data 'y', in which 'model' parameter in *cross_val_score*() command can refer to our Scikit-ANFIS and any ANFIS or DNFS model to be trained and tested. In line 5 of Listing 7 and 8, the mean and standard deviation of the model's evaluation scores in ten folds are calculated and printed in the console.

Listing 7 The code demo of our 10-fold cross-validation experiment for regression dataset.

```
1 from sklearn.model_selection import
      cross_val_score
2 .
  .
  .
3 # Evaluate a score of the model
4 scores = cross_val_score(model,X,y,cv=10,scoring
      ='neg_root_mean_squared_error')
5 print("Mean: %0.3f, Standard Deviation: %0.3f"%(
      scores.mean(), scores.std()))
```

Listing 8 The code demo of our 10-fold cross-validation experiment for classification dataset.

```
1 from sklearn.model_selection import
      cross_val_score
2 .
  .
  .
3 # Evaluate a score of the model
4 scores = cross_val_score(model,X,y,cv=10,scoring
      ='accuracy')
5 print("Mean: %0.3f, Standard Deviation: %0.3f"%(
      scores.mean(), scores.std()))
```

Table 5 summarizes the mean and standard deviation of evaluated scores in each of '10-CV' experiments with various software for ANFIS or DNFS such as Matlab-ANFIS [10], ANFIS-PyTorch [11], ANFIS-Numpy [12], PyTSK [18], HTSK-LN-ReLU [37], and Scikit-ANFIS in four datasets from Table 4. To ensure uniformity in the software builds of fuzzy sets and fuzzy rules for the same data set, we have mandated that every experiment must have an epoch of 100 and an initial step of 0.01, with two Gaussian membership functions designated for each input variable. As illustrated in Listing 3, our experiment comprises two input variables and three fuzzy rules for the Tip. Similarly, as per Listing 4, we have enforced two input variables and 16 fuzzy rules for the Sinc. For the PCD and Iris, we have assigned four input variables and 16 fuzzy rules, as elaborated in Listings 5 and 6. Each experiment was repeated 10 times, and the average was the final result.

In Matlab-ANFIS, we can complete the '10-CV' experiment by calling the *crossvalind*() with the Kfold method and *anfis*() commands in Matlab [10]. In addition, ANFIS-PyTorch can be called by the *cross_val_score*() command through the Numpy wrapper. In contrast, the other four software can be called directly, because all five are based on the Python 3 programming language, and have natural interoperability with the sklearn package. The experimental

results under the two training strategies of the three ANFIS-based software Matlab-ANFIS, ANFIS-PyTorch, and Scikit-ANFIS are distinguished by adding 'hybrid' or 'online' in parentheses, as shown in Table 5. However, there is only a hybrid training method for ANFIS-Numpy, and PyTSK and HTSK-LN-ReLU can be classified as online training methods because they are based on gradient descent to realize the backpropagation update of all parameters in the network. Since PyTSK only applies to classification problems, its experimental results in three regression data sets are denoted as 'n/a'. We have also highlighted the best performance values in bold for both the first four methods under hybrid training strategy and the remaining five methods under online strategy, for each data set.

The findings in Table 5 reveal that the evaluation performance (0.109±0.068 and 0.952±0.054) of our Scikit-ANFIS exceeds all other three software in Sinc and Iris data sets under the 'hybrid' training method, while only the RMSE performance (8e-7±3e-8 and 0.002±5e-5) Matlab-ANFIS exceeds ours in Tip and PCD data sets and ours still exceeds ANFIS-PyTorch and ANFIS-Numpy. On average, our developed Scikit-ANFIS demonstrates performance approximation to commercial software Matlab-ANFIS compared to ANFIS-PyTorch and ANFIS-Numpy. In addition, under the 'online' training method, the RMSE performance (0.382±0.187, 0.101±0.069, and 0.044±0.005) of our Scikit-ANFIS outperforms the other three software such as HTSK-LN-ReLU, Matlab-ANFIS, and ANFIS-PyTorch in three data sets such as Tip, Sinc, and PCD, while the accuracy score 0.968±0.019 of Matlab-ANFIS only slightly outperforms 0.960±0.044 of ours in Iris data set, and ours consistently outperforms the other three software such as PyTSK, HTSK-LN-ReLU, and ANFIS-PyTorch. These results also highlight the faster convergence of our Scikit-ANFIS compared to PyTSK, HTSK-LN-ReLU, Matlab-ANFIS, and ANFIS-PyTorch in updating all the antecedent and consequent parameters of fuzzy systems based on the gradient descent method.

5.4 Discussions and Limitations

Figure 6 and Table 5 yield that Matlab-ANFIS is still the relatively best-performing software among many existing ANFIS-based or DNFS-based software for both regression and classification tasks. However, when faced with a mainstream Python-based machine learning library such as scikit-learn, Matlab-ANFIS is not convenient to use as closed-source commercial software. In contrast, our Scikit-ANFIS software is open source and better suited for the Python platform, outperforming state-of-the-art ANFIS-based or DNFS-based Python software in terms of performance, and is the closest to Matlab-ANFIS. This superiority can be attributed to several key factors. First and foremost, our software benefits from the step size (i.e., learning rate) update method implemented following two heuristic rules [1], which plays an important role in guiding the ANFIS model to accelerate the convergence speed of gradient descent when backpropagating. At the backward stage of Scikit-ANFIS, we apply the adaptive gradient descent method (Adam by default) instead of strict gradient descent to identify the parameters in the ANFIS network. This enables us to obtain an unscaled direct estimation of the parameter's updates, which is well-suited for problems that are large in terms of data or parameters. In addition, although PyTSK and HTSK-LN-ReLU also adopt stochastic gradient descent methods such as Adam, they are different from the ANFIS model in that they completely rely on input–output data pairs to generate corresponding network structure and membership parameters. However, our Scikit-ANFIS relies not only on input and output data, but also on human knowledge to construct fuzzy if-then rules, so the resulting fuzzy system is more consistent with the real data distribution and achieves remarkable results.

Another key contribution of our software is the support of the ANFIS model as an optimization method to directly train existing TSK fuzzy inference systems, allowing users to apply the software more easily. Although Matlab-ANFIS can accomplish the same function, it can only be used for fuzzy systems created using the Matlab language and is not easily compatible with the Python platform.

One of the most significant advantages of our proposed syntax is its high level of consistency with that of scikit-learn: both adopt a universal format that first creates a model, which can then be fed data through *fit*(), before outputting a result through *predict*(). It is also worth noting again that *fit*() and *predict*() functions of our Scikit-ANFIS inherit the same interface provided by scikit-learn, thus facilitating the proposed model to be used efficiently with other available machine learning algorithms. This is also reinforced by the fact that output generated by *predict*() can be directly used to calculate metrics such as RMSE and accuracy (see code example Listing 6: lines 14-15).

Although our Scikit-ANFIS helps users to efficiently combine the ANFIS model with other machine learning algorithms in scikit-learn, it has some limitations. Scikit-ANFIS relies on adaptive gradient descent methods to update antecedent and consequent parameters of a fuzzy system, making it hard to initialize the optimal hyperparameters for the above gradient descent method. Since there is no way to know in advance the optimal value for the hyperparameter, this limitation can be addressed by using the *GridSearchCV*() function in scikit-learn's *model_selection* package to try all possible values to find the optimal one.

During the training process of ANFIS, the optimization methods used play an essential role in obtaining effective results [28]. The commonly used methods are Gradient Descent and Least Squares Estimate, but heuristic

algorithms such as PSO [50] and GA [51] can also be utilized to train the premise and consequence parameters of ANFIS. To further enhance the training method of Scikit-ANFIS, we aim to integrate PSO and GA in a hybrid training method along with Gradient Descent or LSE. This will improve the training process [28] and ultimately lead to better overall performance of our Scikit-ANFIS.

Another limitation of existing Scikit-ANFIS is that the fuzzy system does not directly address variables with missing values, which is also a limitation for some machine learning algorithm as implemented by scikit-learn - a common workaround is to use imputation techniques (e.g., advanced fuzzy interpolation techniques [52] in the context of a fuzzy system) to fill missing values with artificially generated values before training and/or prediction.

One more limitation worth discussion is that Scikit-ANFIS can still suffer from the curse of dimensionality problem, particularly those initialized by the grid partition method. This is because both the number of fuzzy rules and the training time grow exponentially with the number of fuzzy sets for each input variable, which limits the number of input variables and membership functions, resulting in reduced prediction accuracy due to the absence of important characteristic variables [53]. Although the original ANFIS paper [1] does not discuss this limitation possibly due to when data collected by then was relatively small, it's naturally desirable for a model to work with data of high dimensionality to meet the growing trend. While this paper reports only the implementation of original ANFIS, part of future work will concentrate on advanced dimensionality reduction techniques, such as those based on granule computing and rough-set [54–58] for developing novel computational intelligence models and applications in the era of big data.

6 Conclusion

It is common among the research community to apply ANFIS based on the Matlab platform to a variety of regression, classification, process controls, and pattern recognition applications, which makes it difficult for users to combine it with the common scikit-learn library in the Python platform. Hence, in this work, we implement Scikit-ANFIS, a user-friendly, and scikit-learn-compatible Python software using ANFIS architecture specifically designed for training the TSK fuzzy systems. Scikit-ANFIS takes a universal format to create a model and train the model through *fit*() and test it through *predict*(). Our Scikit-ANFIS implementations demonstrate performance gains on four standard data sets compared to existing Python programs that have implemented the ANFIS or DNFS method. Furthermore, our Scikit-ANFIS allows for training an existing TSK fuzzy system directly and automatically generating an ANFIS fuzzy system based on stipulated input–output data pairs.

For future research, we will explore how the Scikit-ANFIS software can be used with deep neuro-fuzzy systems to further strengthen the performance for solving regression and classification problems. Additionally, we will apply it to more complex problem domains such as health and care domain where both model performance and interpretability are usually among the top concerns in medical practice [59].

Acknowledgements This work is partially supported by the Henan Key Research and Development Breakthrough Program of China (No. 222102210191).

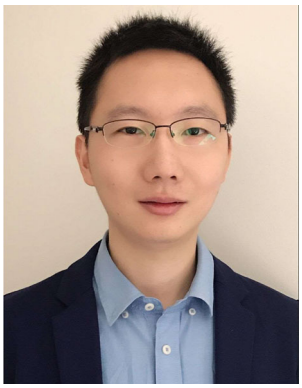
References

1. Jang, J.: Anfis: adaptive-network-based fuzzy inference system. IEEE Trans. Syst. Man Cybern. **23**, 665–685 (1993)
2. Chen, T., et al.: A decision tree-initialised neuro-fuzzy approach for clinical decision support. Artif. Intell. Med. **111**, 101986 (2021)
3. Keikhosrokiani, P., Naidu, A., Anathan, A.B., Iryanti Fadilah, S., Manickam, S., Li, Z.: Heartbeat sound classification using a hybrid adaptive neuro-fuzzy inferences system (anfis) and artificial bee colony. Digital Health **9**, 85 (2023). https://doi.org/10.1177/20552076221150741
4. Chen, T., et al.: A dominant set-informed interpretable fuzzy system for automated diagnosis of dementia. Front. Neurosci. **16**, 867664 (2022)
5. Zand, J.P., Katebi, J., Yaghmaei-Sabegh, S.: A generalized ANFIS controller for vibration mitigation of uncertain building structure. Struct. Eng. Mech. **87**, 231–242 (2023)
6. Osheba, D.S., Osheba, S., Nazih, A., Mansour, A.S.: Performance enhancement of PV system using VSG with ANFIS controller. Electr. Eng. **105**, 2523–2537 (2023)
7. Arévalo, P., Cano, A., Jurado, F.: Large-scale integration of renewable energies by 2050 through demand prediction with ANFIS, ECUADOR case study. Energy **286**, 129446 (2024)
8. Lara-Cerecedo, L., Pitalúa-Díaz, N., Hinojosa-Palafox, J.: Comparative study of the prediction of electrical energy from a photovoltaic system using the intelligent systems ANFIS and ANFIS-GA. Revista Mexicana de Ingeniería Química **22**, 1–16 (2023)
9. Lara-Cerecedo, L.O., Hinojosa, J.F., Pitalúa-Díaz, N., Matsumoto, Y., González-Angeles, A.: Prediction of the electricity generation of a 60-kw photovoltaic system with intelligent

models ANFIS and optimized ANFIS-PSO. Energies **16**, 6050 (2023)
10. MathWorks: neuro-adaptive learning and anfis - r2023a (2023). https://uk.mathworks.com/help/fuzzy/neuro-adaptive-learning-and-anfis.html. Accessed 5 Jan 2024
11. Power, J.: Anfis in pytorch (2019). https://github.com/jfpower/anfis-pytorch. Accessed 5 Jan 2024
12. Meggs, T.: Anfis (2020). https://github.com/twmeggs/anfis. Accessed 5 Jan 2024
13. Gilardi, G.: Anfis (2021). https://github.com/gabrielegilardi/ANFIS. Accessed 5 Jan 2024
14. Rathnayake, N., Dang, T.L., Hoshino, Y.: A novel optimization algorithm: cascaded adaptive neuro-fuzzy inference system. Int. J. Fuzzy Syst. **23**, 1955–1971 (2021)
15. Rathnayake, N., Rathnayake, U., Dang, T.L., Hoshino, Y.: A cascaded adaptive network-based fuzzy inference system for hydropower forecasting. Sensors **22**, 2905 (2022)
16. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
17. Talpur, N., et al.: Deep neuro-fuzzy system application trends, challenges, and future perspectives: a systematic survey. Artif. Intell. Rev. **56**, 865–913 (2023)
18. Cui, Y., Wu, D., Jiang, X., Xu, Y.: Pytsk: a python toolbox for tsk fuzzy systems. arXiv preprint arXiv:2206.03310 (2022). https://doi.org/10.48550/arXiv.2206.03310
19. Ketkar, N., Moolayil, J.: Deep Learning with Python: learn best practices of deep learning models with PyTorch 2 edn Apress. Berkeley, CA (2021)
20. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control. IEEE Trans. Syst. Man Cybern. **SMC–15**, 116–132 (1985)
21. Fresno, C., Fernández, E.A.: Anfis vignette (2012). https://github.com/jfpower/anfis-pytorch/blob/master/Anfis-vignette.pdf. Accessed 5 Jan 2024
22. Chen, T., Shang, C., Su, P., Shen, Q.: Induction of accurate and interpretable fuzzy rules from preliminary crisp representation. Knowl.-Based Syst. **146**, 152–166 (2018)
23. Carter, J., Chiclana, F., Khuman, A.S., Chen, T. (eds.): Fuzzy logic: recent applications and developments, 1st edn. Springer, Switzerland (2021)
24. Liebscher, R.: Pyfuzzy-python fuzzy package (2014). http://pyfuzzy.sourceforge.net/. Accessed 5 Jan 2024
25. Avelar, E., Castillo, O., Soria, J.: Fuzzy logic controller with fuzzylab python library and the robot operating system for autonomous robot navigation: a practical approach. Intuit Type-2 Fuzzy Logic Enhanc. Neural Optim. Algor. Theory Appl. **862**, 355–369 (2020)
26. Scikit-fuzzy (2023). https://pythonhosted.org/scikit-fuzzy/. Accessed 5 Jan 2024
27. Spolaor, S., et al.: Simpful: a user-friendly python library for fuzzy logic. Int. J. Comput. Intell. Syst. **13**, 1687–1698 (2020)
28. Karaboga, D., Kaya, E.: Adaptive network based fuzzy inference system (anfis) training approaches: a comprehensive survey. Artif. Intell. Rev. **52**, 2263–2293 (2019)
29. Oliphant, T.E.: Python for scientific computing. Comput. Sci. Eng. **9**, 10–20 (2007)
30. Oliphant, T.E., et al.: A guide to NumPy, vol. 1. Trelgol Publishing, USA (2006)
31. Zheng, Y., Xu, Z., Wang, X.: The fusion of deep learning and fuzzy systems: a state-of-the-art survey. IEEE Trans. Fuzzy Syst. **30**, 2783–2799 (2022)
32. Sun, C., Jang, J.: A neuro-fuzzy classifier and its applications. In: Proceedings Second IEEE International Conference on Fuzzy Systems (pp. 94-98). IEEE (1993)
33. Talpur, N., Abdulkadir, S.J., Hasan, M.H.: A deep learning based neuro-fuzzy approach for solving classification problems, 167–172 IEEE, (2020)
34. Wu, D., Yuan, Y., Huang, J., Tan, Y.: Optimize tsk fuzzy systems for regression problems: Minibatch gradient descent with regularization, droprule, and adabound (mbgd-rda). IEEE Trans. Fuzzy Syst. **28**, 1003–1015 (2020)
35. Cui, Y., Wu, D., Huang, J.: Optimize tsk fuzzy systems for classification problems: Minibatch gradient descent with uniform regularization and batch normalization. IEEE Trans. Fuzzy Syst. **28**, 3065–3075 (2020)
36. Shi, Z., et al.: Fcm-rdpa: Tsk fuzzy regression model construction using fuzzy c-means clustering, regularization, droprule, and powerball adabelief. Inf. Sci. **574**, 490–504 (2021)
37. Cui, Y., Xu, Y., Peng, R., Wu, D.: Layer normalization for tsk fuzzy system optimization in regression problems. IEEE Trans. Fuzzy Syst. **31**, 254–264 (2022)
38. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: generalization gap and sharp minima. arXiv preprint arXiv:1609.04836 (2016). https://doi.org/10.48550/arXiv.1609.04836
39. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014). https://doi.org/10.48550/arXiv.1412.6980
40. Luo, L., Xiong, Y., Liu, Y., Sun, X.: Adaptive gradient methods with dynamic bound of learning rate. arXiv preprint arXiv:1902.09843 (2019). https://doi.org/10.48550/arXiv.1902.09843
41. Yuan, Y., Li, M., Liu, J., Tomlin, C.: On the powerball method: variants of descent methods for accelerated optimization. IEEE Control Syst. Lett. **3**, 601–606 (2019)
42. Zhuang, J., et al.: Adabelief optimizer: adapting stepsizes by the belief in observed gradients. Adv. Neural. Inf. Process. Syst. **33**, 18795–18806 (2020)
43. Bottou, L.: Large-scale machine learning with stochastic gradient descent, pp. 177–186. Springer, Berlin (2010)
44. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The rprop algorithm. In: IEEE International Conference on Neural Networks, pp. 586–591 IEEE, (1993)
45. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Math. Program. **45**, 503–528 (1989)
46. Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012). https://doi.org/10.48550/arXiv.1212.5701
47. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)
48. Pathak, A.: Restaurant tipping problem using fuzzy logic (2023). https://github.com/ap1904/RTP. Accessed 5 Jan 2024
49. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Ann. Eugen. **7**, 179–188 (1936)
50. Turki, M., Bouzaida, S., Sakly, A., M'Sahli, F.: Adaptive control of nonlinear system using neuro-fuzzy learning by pso algorithm. pp. 519–523 IEEE, (2012)
51. Cárdenas, J.J., García, A., Romeral, J., Kampouropoulos, K.: Evolutive ANFIS training for energy load profile forecast for an IEMS in an automated factory. In: ETFA2011, pp. 1–8 (IEEE, 2011)
52. Chen, T., Shang, C., Yang, J., Li, F., Shen, Q.: A new approach for transformation-based fuzzy rule interpolation. IEEE Trans. Fuzzy Syst. **28**, 3330–3344 (2019)
53. Stathakis, D., Savina, I., Nègrea, T.: Neuro-fuzzy modeling for crop yield prediction. Int. Arch. Photogramm. Remote. Sens. Spat. Inf. Sci. **34**, 1–4 (2006)
54. Li, W., et al.: Feature selection approach based on improved fuzzy c-means with principle of refined justifiable granularity. IEEE Trans. Fuzzy Syst. **31**, 2112–2126 (2023)

55. Su, P., et al.: Corneal nerve tortuosity grading via ordered weighted averaging-based feature extraction. Med. Phys. **47**, 4983–4996 (2020)
56. Li, W., et al.: Double-quantitative feature selection approach for multi-granularity ordered decision systems. IEEE Trans. Artif. Intell. **1**, 1–12 (2023)
57. Mac Parthaláin, N., Jensen, R., Diao, R.: Fuzzy-rough set bireducts for data reduction. IEEE Trans. Fuzzy Syst. **28**, 1840–1850 (2019)
58. Li, W., Zhou, H., Xu, W., Wang, X.-Z., Pedrycz, W.: Interval dominance-based feature selection for interval-valued ordered data. IEEE Trans. Neural Netw. Learn. Syst. **34**, 6898–6912 (2023)
59. Chen, T., Carter, J., Mahmud, M., Khuman, A.S.: Artificial intelligence in healthcare: recent applications and developments, vol. 1. Springer Nature, Singapore (2022)

Dongsong Zhang received the PhD degree in Computer Science and Technology from National University of Defense Technology, China, in 2012. He currently is an assistant professor in School of Big Data and Artificial Intelligence at Xinyang College. His research interests are in the area of soft computing (Neural Network, Fuzzy Logic), Real-Time Systems.



Tianhua Chen received the Ph.D. degree in Computational Intelligence from Aberystwyth University, Aberystwyth, U.K., in 2017. He is currently a Reader (Associate Professor) in Artificial Intelligence with the Department of Computer Science, School of Computing and Engineering, University of Huddersfield, UK. He has published over 60 peer reviewed papers in leading international journals and conferences, including a lead-authored paper selected as IEEE Transactions on Fuzzy System Publication Spotlight Article by IEEE Computational Intelligence Society. His research interests are: Artificial Intelligence for health and wellbeing, Explainable AI, Neuro-Fuzzy Systems. Tianhua is an Editorial Board Member of Artificial Intelligence in Medicine journal (Elsevier), BMC Medical Informatics and Decision Making journal (Springer), and PLOS ONE.