Fuzzy Controller Inference via Gradient Descent to Model the Longitudinal Behavior on Real Drivers

Alberto Díaz-Álvarez[1], Francisco Serradilla-García[1], Felipe Jiménez-Alonso[2], Edgar Talavera-Muñoz[3]and Cristina Olaverri-Monreal[4]

Abstract— **This paper introduces a method to represent Takagi-Sugeno Fuzzy Control System (FCS) as computational graphs, so they can be adjusted through a supervised training process based on gradient descent. It has been tested both with values extracted from a fuzzy controller created for the test and also for a set of real data extracted from real drivers. Obtained results show high conformance to synthetic data, and seem to describe a car-following behavior with quite good precision, which suggests that it is possible to model the behavior of conductors in a longitudinal model based on `if-then` type rules.**

I. INTRODUCTION

The fuzzy systems are an application of fuzzy logic in the area of Computational Intelligence that have been successfully applied over a great variety of control problems [1], [2], [3], whose potential values are what are known as *fuzzy sets*, that is, concepts defined in a vague way. In this way we are able to embrace uncertainty within our system, which is more robust.

In systems of which we know their inner workings, we can have a domain expert to help us to define the problem and, if possible, to solve it based on if-then rules based on its inputs and outputs variables [4]. In systems that are unknown to us, however, we do not have these experts [5]. Therefore, other techniques are needed for their modeling.

One of the most widely used techniques is undoubtedly supervised training on neural networks. In this way, having a set of inputs and their corresponding outputs, the Artificial Neural Network (ANN) is modelled in such a way that approximates the inputs to the expected outputs, even for unknown cases. The less stochastic the system is depending on its inputs, the better it is modeled by a neural network. But there is a critical problem with this kind of approach: we can't explain why ANNs make the decisions they do. Therefore, the adjustment of a FCS based on a supervised scheme is relevant, since as well as neural networks, FCSs are also universal approximators. [6].

The manual adjustment process in this kind of controllers is a two-step process consisting on:

[1]Dpto. de Inteligencia Artificial, ETSI de Sistemas Informáticos, Universidad Politécnica de Madrid, Spain `alberto.diaz at upm.es`, `francisco.serradilla at upm.es`

[2]University Institute of Automobile Research (INSIA), Universidad Politécnica de Madrid, Spain `felipe.jimenez at upm.es`

[3]Dpto. Sistemas Informáticos, ETSI de Sistemas Informáticos, Universidad Politécnica de Madrid, Spain `e.talavera at upm.es`

[4]Chair for Sustainable Transport Logistics 4.0, Johannes Kepler University, Linz, Austria `cristina.olaverri-monreal at jku.at`

1) **Fuzzy partition definition** where, for each input variable, a linguistic variable is defined in the form of a partition of fuzzy sets.
2) **Fuzzy rules definition**, where the knowledge of the problem is codified as a sequence of `if-then` rules, to be used later by the inference component.

The self-adjusting techniques available in the literature for this type of controllers are basically focused on one of the two steps: either adjusting fuzzy partitions fixing the rule-base, or adjusting the rules fixing t he s tructure of the linguistic variables partitions. In both cases, the more usual methods are those based on evolutionary computation [7], [8], gradient descent [9], [10] (in the case of rules, after representing them according to their error) or hybrid approaches between these and other techniques [11]. However, the problem is as follows: there are techniques for the adjustment of existing controllers, of rules based on their already fixed partitions and vice versa, and when the approaches try to adjust the FCS as a whole, they rely on separate space state search techniques (such as meta-heuristics, has seen in [12]). However, there seems to be no effective techniques for the efficient generation of controllers from scratch. There are approaches with different techniques of representation of controllers [13], [14], but usually suffer from limitations in representation capacity or learning speed.

In this paper we propose a compact representation of FCS based on computational graphs, in such a way that (i) the operation is quite fast, especially when inferring many values at once, (ii) it is possible to use the gradient descent following a supervised training scheme, and (iii) it is possible to explain why the system makes decisions at each moment.

The system will be tested with a random sample of values taken from a known FCS, where we will see that the fuzzy surface of the inferred system approximates the fuzzy surface of the true system. Later, we will use the same technique on an unknown system, the longitudinal driver's behavior based on some variables. We will see that a relatively high degree of precision is achieved, which indicates that it is possible to be modeled.

II. COMPUTATIONAL GRAPH REPRESENTATION

Depending on the author, a Fuzzy Control System can be depicted into a different number of components. In our case, we break it down into three main components, each of them responsible of one operation: (i) *fuzzification*, (ii) *inference* (where we include the `if-then` rules), and (iii) *defuzzification*.

$$\begin{pmatrix} x_{V_1}^1 & x_{V_2}^1 & \dots \\ x_{V_1}^2 & x_{V_2}^2 & \dots \\ x_{V_1}^3 & x_{V_2}^3 & \dots \\ x_{V_1}^4 & x_{V_2}^4 & \dots \end{pmatrix} \rightarrow \begin{pmatrix} \mu_{V_1}^1(x_{V_1}^1) & \mu_{V_1}^2(x_{V_1}^1) & \mu_{V_2}^1(x_{V_2}^1) & \mu_{V_2}^2(x_{V_2}^1) & \dots \\ \mu_{V_1}^1(x_{V_1}^2) & \mu_{V_1}^2(x_{V_1}^2) & \mu_{V_2}^1(x_{V_2}^2) & \mu_{V_2}^2(x_{V_2}^2) & \dots \\ \mu_{V_1}^1(x_{V_1}^3) & \mu_{V_1}^2(x_{V_1}^3) & \mu_{V_2}^1(x_{V_2}^3) & \mu_{V_2}^2(x_{V_2}^3) & \dots \\ \mu_{V_1}^1(x_{V_1}^4) & \mu_{V_1}^2(x_{V_1}^4) & \mu_{V_2}^1(x_{V_2}^4) & \mu_{V_2}^2(x_{V_2}^4) & \dots \end{pmatrix}$$

Fig. 1: The fuzzification component will transform the crisp input values into fuzzy input ones.

Our approach will be representing the FCS as a computational graph in such a way that their parameters are adjusted through a gradient descent of the space they define.

For this purpose, a number of design decisions have been taken to simplify the development of the controller. These assumptions are relatively easy to modify without altering the representation greatly:

- Each linguistic variable will contain only fuzzy sets defined one line-descending, one line-ascending and n trapezoidal membership functions.
- The t-norm and s-norm operations will be $\min$ and $\max$ functions respectively. The $\max$ function will be also used as the accumulation operation.
- The FCS will be an order 0 Takagi-Sugeno controller. That is, the output will be constant values, and therefore will be represented as a singleton fuzzy set. The defuzzification method will be the operation CoGS (Center of Gravity for Singletons), defined as follows:

$$CoGS = \sum_{i=1}^{n} x_i \cdot \mu_i$$

  Being x_i the characteristic point of the singleton function and μ_i the membership of the output value for this point.
- The fuzzy partitions size will be an algorithm metaparameter, and will not vary throughout the adjustment process.
- The number of output variables will be 1.

We now move on to the controller representation. Let $V = V_1, V_2, \dots, V_n$ and O the ordered set of input and the output linguistic variables of our FCS to be modelled respectively. Each of them will contain a pre-defined number of fuzzy sets $N_{V_1}, \dots, N_{V_n}$ for the input variables and N_O for the output one.

The FCS will receive as an input an (m, n) matrix and will return a (m) vector, where m is the number of examples to be inferred. The development process will be as follows:

1) Construction of each fundamental component of the FCS as an independent computational graph.
2) Construction of the FCS as a computational graph composed by each of the sub-graphs developed in the previous step.

A. *Fuzzification component*

The component will take the (m, n) input matrix which will contain, for each of the m examples to infer, the n values that each of the V_i input variables will take. The output of this graph will be a $(m, \sum_{i=1}^{n} N_{V_i})$ matrix where all the member, where the membership degrees of those values will be stored for each fuzzy set of their corresponding variable (Figure 1).

To do this, as many operations (membership functions) as fuzzy sets are present will be applied to each column of the input matrix. Specifically, line-descending, trapezoidal and line-ascending functions will be applied.

The line-ascending and line-descending computational graphs have a similar shape. Their characteristic equations would be those shown in Equations 1 and 2 respectively:

$$\mu_{asc}(X) = \min(\max(\frac{X - a}{\delta b}, 0), 1) \tag{1}$$

$$\mu_{desc}(X) = \min(\max(\frac{a - X}{\delta b + 1}, 0), 1) \tag{2}$$

The network associated with the line-ascending formula is shown in Figure 2, which is the one that would correspond to the last set of a fuzzy partition.

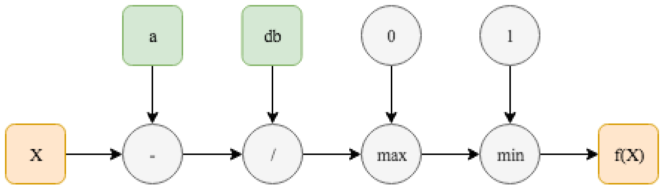


Fig. 2: Computational graph corresponding to the line-ascending membership function.

The computational graph for the line-descending function is quite similar to the one in the figure.

It can be seen that the only adjustable variables are a and δb, which define the interval $(a, a + \delta b) \subset \mathbb{R}$.

The trapezoid membership function is defined in a similar way according to the parameters $(a, \delta b, \delta c, \delta d)$, which defines the intervals $I_1 = [a, a + \delta b)$, $I_2 = [a + \delta b, a + \delta b + \delta c)$ and $I_3 = [a + \delta b + \delta c, a + \delta b + \delta c + \delta d)$. It can be defined as a combination between the line-ascending and line-descending computational graphs, as is shown in Equation 3 3.

$$\mu_{trap}(X) = \min(\max(\min(\mu_{asc}(X), \mu_{desc}(X)), 1), 0) \tag{3}$$

The associated graph is described in Figure 3.

Knowing the computational graphs for each membership function, we can define the graph associated with the fuzzy partition of a linguistic variable. Assuming that the variable V_i is divided into N_{V_i} distinct fuzzy sets, the partition will be composed of:

- A first fuzzy set defined by a descending slope.

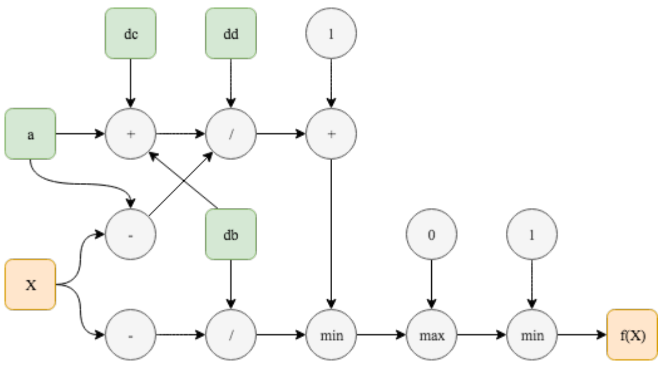


Fig. 3: Computational graph corresponding to the trapezoidal membership function.

- N_{V_i} fuzzy sets defined by trapezoid membership functions.
- A last fuzzy set defined by a line-ascending membership function.

The adjustment process will try to approximate the values of the variables in this graph to minimize an error. The values will correspond to the points of the membership functions, as shown in Figure 4. Thus, besides a standardized and complete fuzzy partition for the linguistic variable, each small variation in the gradient has the potential to cause a variation in all the related variables.

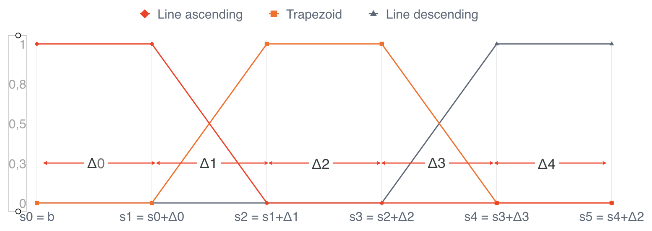


Fig. 4: Relationship between the variables to be adjusted and the membership functions that define.

Our fuzzification graph is defined in such a way that, for an input matrix $m \times l$ being m each tuple of values to be inferred and l each of the linguistic variables, it will generate an array of the form $m \times \sum_{i=1}^{l} |l_i|$, being $|S|$ the number of fuzzy sets contained in the linguistic variable.

B. *Inference component*

This block will take as input a $(m, \sum_{i=1}^{n} N_i)$ matrix, i.e. the output matrix of the fuzz block, and will generate a (m, N_O) matrix containing the fuzzy output (one output for each output fuzzy set). To do this, it will make use of a set of rules upon which it will base its inference.

This set of rules is what we will try to adjust. The representation will be that of a (v_i+1)-dimensional matrix, being $v_i = \sum_{i=1}^{n} l_i$ the total number of fuzzy inputs that come to the inference block. The additional dimension corresponds to the output linguistic variable. In other words, each possible fuzzy output set (each value within the axis corresponding to the output) will correspond to a v_i-dimensional matrix

$$\begin{pmatrix} \mu_1^A \\ \mu_2^A \end{pmatrix} \times \begin{pmatrix} \mu_1^B \\ \mu_2^B \\ \mu_3^B \end{pmatrix} = \begin{pmatrix} \top(\mu_1^A, \mu_1^B) \\ \top(\mu_1^A, \mu_2^B) \\ \top(\mu_1^A, \mu_3^B) \\ \top(\mu_2^A, \mu_1^B) \\ \top(\mu_2^A, \mu_2^B) \\ \top(\mu_2^A, \mu_3^B) \end{pmatrix}$$

Fig. 5: The cartesian product of the input fuzzy variables generates all possible rules that can be defined in a fuzzy controller.

resulting from the cartesian product of the input variables. That is, each one of the possible combinations of rules, to which we can associate a value. In the Figure 5 an example of inference with two linguistic variables is shown after applying the t-norm to them.

Since the s-norm and the accumulation operations are defined with the same function, an `OR` type rule is equivalent to two or more `AND` type rules, since the accumulation of its results is equivalent. Therefore, by applying the t-norm to these combinations we have all possible combinations of rules. But so far we don't have any adjustment.

If we apply Hadamard's product to a matrix of weights with the same dimension and round its values to $\{0, 1\} \in \mathbb{N}$, we have a way of adjusting which rules are the most relevant and which are not. However, this representation has a problem: these two values define a step function where the error does not propagate because its gradient is 0.

However, if instead of a discrete value, the weight takes a real value and we apply a sigmoid function, the value will be kept between $(0, 1) \subset \mathbb{R}$ with the advantage that the gradient does not freeze, and in a later process the rules whose values exceed certain established ranges can be discarded.

At the end of the inference graph, after applying accumulation, we have as many fuzzy values as there are fuzzy sets at the output for each of the examples in our (m, N_O) output matrix.

C. *Defuzzification component*

This block has the particularity that it does not have any variable to adjust. It is simply an operation that takes a two-dimensional array, i.e. our (m, N_O) matrix with the fuzzy output values, and returns a vector of range (m) with the values in the domain of the output linguistic variable.

III. TESTING AGAINST A KNOWN CONTROLLER

To check the validity of the approach, a known fuzzy controller has been adjusted solely from its inference data, the tipping problem. Its input linguistic variables are *food* and *service*, and its output is texttip. Its fuzzy rules are defined in the Equation 4).

$$
\begin{aligned}
\text{service IS good} &\rightarrow \text{tip IS high}\\
\text{food IS good} &\rightarrow \text{tip IS high}\\
\text{service IS good} \wedge \text{food IS average} &\rightarrow \text{tip IS low}\\
\text{service IS average} \wedge \text{food IS good} &\rightarrow \text{tip IS high}\\
\text{service IS bad} &\rightarrow \text{tip IS low}\\
\text{food IS bad} &\rightarrow \text{tip IS low} \qquad (4)
\end{aligned}
$$

These components, where t-norm equals the minimum function and s-norm equals the maximum function, describe a surface as shown in Figure 6

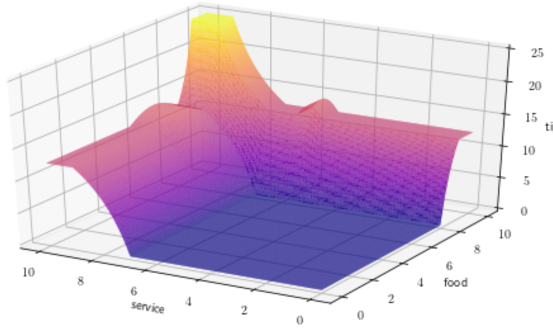


Fig. 6: Fuzzy control surface for the food-service-tip defined.

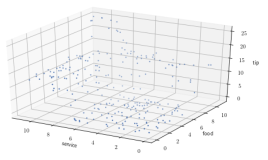
After collecting a uniform 100-point random sample of this surface, a FCS is created following the technique described in this paper, by defining 2 i nputs, e ach composed of three fuzzy sets.

The controller is trained by applying the ADAM [15] gradient descent algorithm with a learning rate of $\alpha = 0.01$ for 500 epochs. In the Figure 7 the correct evolution of the controller through the 500 epochs can be appreciated.

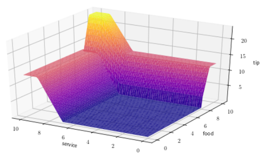
This process shows us that it is possible to obtain FCS adjusted to a pattern of data with a high degree of precision, with the advantage that it is possible therefore to explain the why behind the predictions made by the model with `if-then` rules, unlike other techniques such as ANNs.

IV. APPLICATION TO THE LONGITUDINAL BEHAVIOR

We present in this section the approach to check whether the behaviour of a real driver in an urban environment can

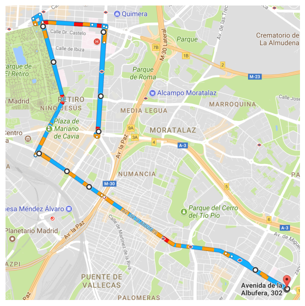


(a) Random sample across the surface (100 points)

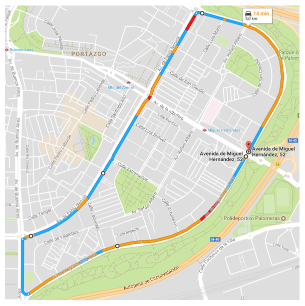


(b) Adjustment after 500 training steps

Fig. 7: Adjusted surface after 500 training steps over a sample of 100 points along the controller surface: (a) The points, and (b) the resulting surface after the training process.



(a) Route R_1



(b) Route R_2

Fig. 8: The two routes, (a) R_1 (training route), and (b) R_2 (test route).

be represented by a FCS. To do this, the technique described in the previous section on a training set has been used, and its performance has been tested on a test set.

Both data sets have been collected from two urban circuits, R_1 and R_2 for three different drivers (Figure 8). These data have been collected at $10Hz$ on an instrumented vehicle with a Camera, a CAN bus reader and a LiDAR attached. Those devices provide enough information to get, after a post-processing task, the following variables: (i) vehicle speed, (ii) distance to leader, (iii) speed of approach to the lead vehicle, (iv) distance to the next Traffic Light Signal (TLS), and (v) status of the next TLS. The training set contains a total of 4469 examples, whereas the test set contains 1771.

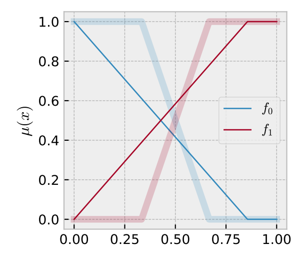
The parameters for the training process are 7 input variables (the next TLS status variable is divided in three variables: status red, status yellow and status green), a learning rate of $\alpha = 0.01$, $250K$ epochs and a training-validation split ratio of 80-20. The results presented in Table I correspond to the Top-4 architectures tested from a wide range of them. The architecture is presented as a sequence of numbers, being each value the number of fuzzy sets the variable is comprised of.

TABLE I: The different trained architectures and their corresponding Root Mean Squared Error (RMSE).

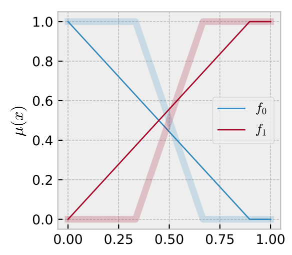
Id	Architecture	RMSE		
		Training	Validation	Test
FCS_1	$2,2,2,2,2,2,2$	0.059	0.064	0.062
FCS_2	$3,3,2,2,2,3,3$	0.073	0.079	0.080
FCS_3	$4,3,2,2,2,3,3$	0.072	0.078	0.088
FCS_4	$5,5,2,2,2,5,5$	0.063	0.068	0.109

In this particular problem it has been observed that training carried out in this fashion causes the RMSE to fall faster in the same number of iterations. It can be observed that the FCS_1 architecture is the one that has obtained the smallest error in training, and it is confirmed that it is the one that has generalized the best. Figure 9 shows the fuzzy partitions for the input variables before and after the training process.

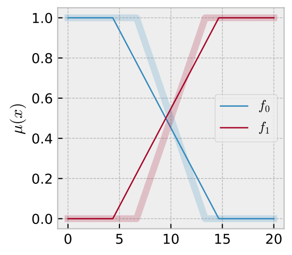
The membership functions for the traffic light state are not included because they are extremely similar to the speed in the Figure 9.



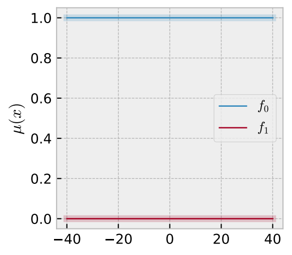
(a) Distance to leader (0 m a 50 m norm.)



(b) Distance to next TLS (0 m a 100 m norm.)



(c) Speed ($0\,\mathrm{m\,s^{-1}}$ a $20\,\mathrm{m\,s^{-1}}$)



(d) Speed to Leader $-40\,\mathrm{m\,s^{-1}}$ a $40\,\mathrm{m\,s^{-1}}$)

Fig. 9: Fuzzy partitions before (less opaque) and after (opaque) the training process for the FCS_1 architecture.

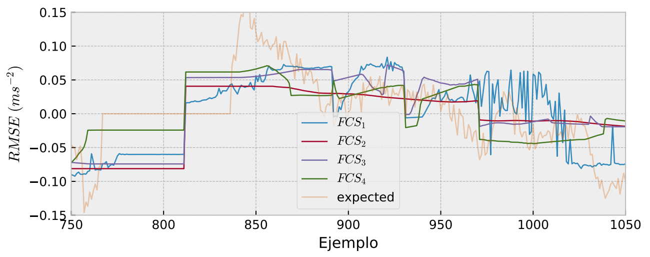


Fig. 10: Comparison between the four architectures over a small excerpt of the full test set.

One detail worth highlighting is the learning process for the linguistic variable *Speed to Leader* (the speed of approach to the lead vehicle. All of the training runs over the architectures described in Table I gave the same result: only one fuzzy set activated all over the domain, and the rest of them deactivated.

The most plausible reason we can find f or t his effect is that, as it is an urban environment, the speeds we treat are very low compared to those working on this sort of models, thus, the difference in speeds are not a determining factor compared to other variables. Figure 10 shows the acceleration profiles estimated by the FCSs against the actual acceleration at each moment in a small section of the whole test dataset.

It can be observed how the controller adjusts with less error in test compared to the rest. FCS_1 presents peaks that approximate the inferred value to the real value, while the rest maintains constant values.

V. CONCLUSIONS

Applying gradient descent to adjust a FCSs quickly enables an optimal controller to be found for a problem. However, the problem of the longitudinal model is not completely treatable with a controller, at least not with the posed variables and data set. However, as a result of the results obtained, we believe that by increasing the variables to be analyzed, with a better precision of the data, and with a greater volume of them, it is possible to increase the precision of these models generated notably.

The fuzzy control systems generated in this fashion are quick enough both training and inferring. They can therefore be used as part of optimization models of their meta-parameters (e.g. number of fuzzy sets per linguistic variable). However, one of their problems is that the rule base generated is extremely large. After all, no method of discarding rules with weights very close to 0 or any method of simplifying equivalent or antagonistic rules has been incorporated. These issues will be addressed in further research.

ACKNOWLEDGMENT

This work has been partially funded by the spanish Ministerio de Economa y Competitividad MINECO (CAV Project TRA2016-78886-C3-3-R) financed this work. We would also like to thank NVIDIA for generously providing us with the Titan X GPU used to perform this study.

REFERENCES

[1] C.-C. Lee, “Fuzzy logic in control systems: fuzzy logic controller. i,” *IEEE Transactions on systems, man, and cybernetics*, vol. 20, no. 2, pp. 404–418, 1990.
[2] G. Feng, “A survey on analysis and design of model-based fuzzy control systems,” *IEEE Transactions on Fuzzy systems*, vol. 14, no. 5, pp. 676–697, 2006.
[3] X. Su, P. Shi, L. Wu, and Y.-D. Song, “A novel control design on discrete-time takagi–sugeno fuzzy systems with time-varying delays,” *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 4, pp. 655–671, 2013.
[4] X.-J. Ma, Z.-Q. Sun, and Y.-Y. He, “Analysis and design of fuzzy controller and fuzzy observer,” *IEEE Transactions on fuzzy systems*, vol. 6, no. 1, pp. 41–51, 1998.
[5] H. Vereecken, A. Schnepf, J. W. Hopmans, M. Javaux, D. Or, T. Roose, J. Vanderborght, M. Young, W. Amelung, M. Aitkenhead *et al.*, “Modeling soil processes: Review, key challenges, and new perspectives,” *Vadose Zone Journal*, vol. 15, no. 5, 2016.
[6] J. L. Castro, “Fuzzy logic controllers are universal approximators,” *IEEE transactions on systems, man, and cybernetics*, vol. 25, no. 4, pp. 629–635, 1995.
[7] F. Herrera, M. Lozano, and J. L. Verdegay, “Tuning fuzzy logic controllers by genetic algorithms,” *International Journal of Approximate Reasoning*, vol. 12, no. 3-4, pp. 299–315, 1995.
[8] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*. John Wiley & Sons, 2000, vol. 7.
[9] F. Guély and P. Siarry, “Gradient descent method for optimizing various fuzzy rule bases,” in *Fuzzy Systems, 1993., Second IEEE International Conference on*. IEEE, 1993, pp. 1241–1246.
[10] Y. Shi, M. Mizumoto, N. Yubazaki, and M. Otani, “A learning algorithm for tuning fuzzy rules based on the gradient descent method,” in *Fuzzy Systems, 1996., Proceedings of the Fifth IEEE International Conference on*, vol. 1. IEEE, 1996, pp. 55–61.
[11] J. Mar and F.-J. Lin, “An anfis controller for the car-following collision prevention system,” *IEEE Transactions on vehicular technology*, vol. 50, no. 4, pp. 1106–1113, 2001.

[12] M. Mannle, “Parameter optimization for takagi-sugeno fuzzy models-lessons learnt,” in *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat. No. 01CH37236)*, vol. 1. IEEE, 2001, pp. 111–116.

[13] H. R. Berenji and P. Khedkar, “Learning and tuning fuzzy logic controllers through reinforcements,” *IEEE Transactions on neural networks*, vol. 3, no. 5, pp. 724–740, 1992.

[14] C.-T. Lin, “A neural fuzzy control system with structure and parameter learning,” *Fuzzy Sets and Systems*, vol. 70, no. 2-3, pp. 183–212, 1995.

[15] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.