

Generating Linguistic Rules from Data Using Neuro-fuzzy Framework

Jacek M. Zurada and Andrzej Lozowski
Department of Electrical Engineering, University of Louisville
Louisville, Kentucky 40292, USA
e-mail: jmzura02@@starbase.spd.louisville.edu

Keywords: linguistic rules generation, rule extraction, fuzzy inference

Abstract

Soft computing techniques offer unique advantages for extracting linguistic rules from data. Noisy numerical data characterizing input/output mappings are first used to develop a neural network model. Fuzzy techniques are then employed for extracting linguistic IF—THEN rules from binary-valued model responses. The algorithm presented is able to tune both the resolution and the number of rules. Also, the generated rules are reducible in a Boolean sense.

I. INTRODUCTION

High performance computing makes extensive use of numerical data which characterize input/output relationships of different systems. Both conventional and learning-based algorithms (including machine learning and neurocomputing) are commonly employed to compute systems' responses and to provide output categorization for specific input conditions. Such mapping of real-valued inputs into discrete outputs plays an important role in any classification type effort such as diagnostics, advising, and decision-making. Areas of application include engineering, medicine, chemical analysis, control, agriculture, financial and insurance management and others.

With models of such systems of interest developed through learning from abundant data, the user can input facts and compute the system's output. Numerical inputs produce output categories, but most if not all neural network-based models provide no interpretation, justification or explanation of responses. Intertwined mapping of linear combinations, nonlinearly squashed and later combined themselves again, offer little guidance as to what the underlying decision rules are. This concerns either an explanation for a specific input instance, or a set of rules underlying overall properties across the full range of inputs rele-

vant to the model.

We further assume that we are interested in formulating common sense rules in either ordinary linguistic format or in Boolean-type logic sentences understandable for humans. Furthermore, we assume that noisy data which describe a time-invariant or slowly varying nonlinear system are available to build a model of acceptable accuracy. Again, no expert knowledge of the system is required, but learning from data can embed the knowledge in it in a distributed form. Moreover, no analytical formulas are known for the system or its model and therefore, only numerical relationships present in data are contributive to the rule-producing effort.

II. OVERVIEW OF EXISTING APPROACHES

The issue of rule extraction from systems' models has been approached from several different perspectives. In some cases, neural networks can provide clues as to what the most dominant present or absent inputs contributing to the output are [1, 2]. This requires decoded binary outputs architecture associated with analysis of weight distribution fanning in into a specific output node. Output layer weights of large negative or positive values are interpreted as combining hidden neurons' responses regarded as factors present or absent, respectively. Tracing the hidden layer weights back to the inputs from 'factor' neurons can often lead to identification of contributing inputs in terms of their presence or absence. The disadvantage of this rather heuristic approach is that it lacks general proof and may not always be effective.

In another approach towards rule extraction, inputs are considered on a one by one basis. To find the impact of a single input, it is nullified and network output is evaluated for change of classification [3]. This procedure is repeated for each input and the total number of misclassifications is counted as a so-called charac-

teristic number, N_i . The lower is N_i , the larger is the contribution of the i -th input. This approach, however, resolves the credit assignment problem only for inputs being delayed outputs. It is therefore only efficient in cases when we deal with system state being output at the same time, and hence is of limited validity.

A large base of literature on rule extraction is devoted to the concept of Knowledge Base Neural Networks [4, 5, 6, 7]. Such networks are created from initial implicit rules embedded into neural networks which are then refined via supervised training. Through subsequent weight clustering, averaging, and eliminating of irrelevant weights, but preferably of single neurons, networks are becoming amenable for interpretation.

A number of heuristic algorithms has been devised to combinatorially search for logic or linguistic rules. Although the produced rules, if they exist, can be pervasively simple, the KBNN method faces such practical difficulties as necessity of a priori implicit rules which must initialize the network. Other limitations are related to the decision regions eventually formed by a refined network. The rules are more involved than a rather rigid rule format often allows for them to be.

One of the difficulties encountered in all the above approaches is that networks for rule extraction are not minimal. As a result, resulting rules will replicate the redundancy already present in oversized neurons' layers and weights. To circumvent this limitation, learning with weight/neuron suppression has been developed. Most successful approaches involve structural learning [8] and convergence suppression and divergence facilitation [9] first proposed by Yasui. The procedures of weights and neuron removal after learning involving complexity penalty terms have proven to be effective in extracting rules in a number of cases. Examples include logic function learning, edible mushroom rules, and extraction of binary pattern input features.

The main difficulty in generating logic rules understandable to humans for all of the method reviewed above is that decision regions often form irregular and disjoint regions in high-dimensional input space. The arithmetic of input space partitioning by hyperplanes does not, in general, correspond to logic operators such as AND, OR, M of N true, etc. In fact, for these operators to be applicable, the original input coordinates would have to be changed to translate the typical complexity of decision regions into the simplicity of logic rules. As a result of our study we therefore have reasons to believe that fuzzy rule extraction methodology

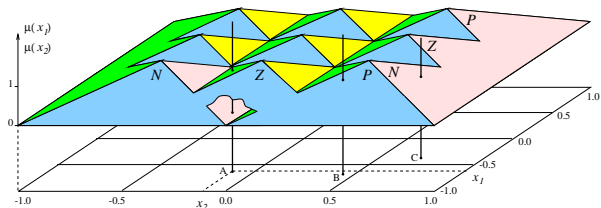


Figure 1: Graphic interpretation of membership functions, minimum operation and crisp input instances.

bypasses many of the rule extraction difficulties discussed above [10].

III. FUZZY RULE EXTRACTION ALGORITHM

The goal of rule extraction is to create a set of rules $\{r_k\}$ based on some existing decision system. Each rule is an implication $r_k = \pi_k \Rightarrow \varrho_k$ where π_k and ϱ_k are the input and output instances, respectively. Assume that the system is a neural network classifier with two inputs and one output. An input instance $\mathbf{x} = \langle x_1, x_2 \rangle$ is a vector $\mathbf{x} \in R^2$ whose entries represent parameters specified in a given problem. Each input instance \mathbf{x} should be classified to one of m classes. Without loss of generality assume $m = 2$, where the output instance is a class $\varrho_1 \in \{no, yes\}$.

For the sake of rule production we attempt to replace the decision system with a fuzzy classifier whose reasoning rules will linguistically describe the system function. Thus the input \mathbf{x} needs to be fuzzified. Each entry x_i of the input can be represented by a linguistic variable \tilde{x}_i which is a vector of memberships $\mu_{\pi_i}(x_i)$ as in [11]. We arbitrarily choose the set of fuzzy classes $\pi_i \in \{N, Z, P\}$ which refer to *negative*, *zero* and *positive*, as shown in Fig. 1. This choice can be modified as needed. Membership functions for each entry are also arbitrarily chosen to be triangular. Note that with this fuzzification scheme up to 9 DNF rules can be created.

To create rules the whole set of data points and their classifications are investigated. Some of the points are classified as belonging to class *yes*, whereas the others belong to class *no*. We deal with both classes separately. Let D_{yes} be a set of points classified as *yes*. For a given point \mathbf{x} that belongs to D_{yes} , memberships of its entries are first found. Consider point A in Fig. 1. The coordinates of this point are $(-0.6, -0.2)$. Due to the fuzzification scheme memberships of the point coordinates are $\mu_N(x_1) = 0.8$, $\mu_Z(x_1) = 0$, $\mu_P(x_1) = 0$, and $\mu_N(x_2) = 0.4$, $\mu_Z(x_2) = 0.6$, $\mu_P(x_2) = 0$.

Now we proceed to evaluate the membership function $\mu_{yes}(y)$ based on the fuzzy reasoning technique.

Define a t-norm $T_{yes}^{\pi_k}(x)$ as

$$T_{yes}^{\pi_k}(x) = \min \{ \mu_{\pi_1}(x_1), \mu_{\pi_2}(x_2) \}; \quad \pi_k = \langle \pi_1, \pi_2 \rangle$$

separately for each instance π_k . In Fig. 1 the t-norms are represented by polyhedrons due to the triangular shapes of the entry membership functions. For point A the t-norm $T_{yes}^{\langle N, N \rangle} = 0.4$ and $T_{yes}^{\langle N, Z \rangle} = 0.6$. All other t-norms for this point are equal 0. In the same manner t-norms for all other points in D_{yes} need to be evaluated. Eventually each instance will result with a list of t-norms graphically represented by points located at the corresponding polyhedron walls. Note that one point can belong to several polyhedrons. Thus hypothetically more than one rule can be created by a single data point in this approach as opposed to machine learning or a method introduced in [12].

Successively, s-norms for each instance are calculated. S-norm $S_{yes}^{\pi_k}$ is a maximum t-norm associated with instance π_k :

$$S_{yes}^{\pi_k} = \max T_{yes}^{\pi_k}.$$

Graphically the s-norm $S_{yes}^{\pi_k}$ is a vertical coordinate (membership value) of the point closest to the vertex of the polyhedron corresponding to instance π_k . For example, assuming that points B and C belong to class *yes*, $S_{yes}^{\langle N, P \rangle}$ equals the t-norm of point B, as this point is located higher than point C at the polyhedron $\langle N, P \rangle$.

The procedure described above is then repeated for all data points classified as belonging to class *no*. In this manner instances π_k are validated by s-norms $S_{yes}^{\pi_k}$ and $S_{no}^{\pi_k}$. At this stage instance π_k can be used for creating the rule $\varrho_i \in P_i$ based on these s-norms. We introduce the uncertainty margin ε which is a number from the range $[0, 1]$. The considered system has only one output; therefore, $\varrho_k = \langle \varrho_{k1} \rangle$. As long as $S_{yes}^{\pi_k}$ and $S_{no}^{\pi_k}$ differ from each other by more than ε , instance π_k creates a rule whose THEN-part is found from:

$$\varrho_{k1} = \begin{cases} yes & \text{if } S_{yes}^{\pi_k} - S_{no}^{\pi_k} > \varepsilon, \\ no & \text{if } S_{no}^{\pi_k} - S_{yes}^{\pi_k} > \varepsilon, \\ - & \text{otherwise.} \end{cases}$$

Here “-” means that the rule invoked by instance π_k is undecidable. The larger the ε , the more undecidable entries are created. Rule r_k becomes totally undecidable as ε approaches 1 since the membership function value belongs to the range $[0, 1]$. Following the introduced algorithm, a rule for each input instance π_k can be created and thus the whole rule set $\{r_k\}$ is generated.

In this approach the number of input fuzzy classes as well as the fuzzification scheme are chosen arbitrarily. Certainly, the fuzzy classifier under construction would improve its accuracy if the gravity centra of the fuzzy classes and the membership function slopes were optimized for a given task. This issue is of a great importance for every rule extraction algorithm [13].

IV. EXAMPLE

The algorithm of fuzzy rule extraction has been tested on the IRIS problem. A data set Q of 150 examples, each consisting of four leaf characteristics and an expert classification, were used as a training set for a neural network classifier with four inputs and three outputs. For fuzzy rule extraction the input fuzzifiers have been formed using the standard triangular membership function shapes. Each input has been quantized into three classes with a center of gravity located in the middle and both ends of the range of changes of the input. Various rule sets composed of 81 rules were created using the introduced algorithm with different values of the parameter ε . Totally undecidable rules were subsequently pruned from the sets. Performance of the resulting fuzzy rule sets was dependent on ε . Obviously, large ε leads to a very compact rule set, however the system makes more misclassification errors as compared to the one created with small ε . An example of a rule set minimized to a disjunctive normal form [4] is shown in Fig. 2. The rules can be represented graphically on a four-dimensional hypercube whose corners and sides correspond to input instances while the classification is indicated in three grey-levels (see Fig. 3). On the figure variables x_1 , x_2 and x_3 are the vertical, horizontal and axial dimensions, respectively, and variable x_4 is represented as three different cubes for three different input classes π_4 .

V. CONCLUSIONS

This paper presents a method of extracting crisp rules from a trained neural network classifier. The rules have the form of those used in fuzzy reasoning. Furthermore, the neural network is a necessary element if the training data is noisy, since the neural network provides filtration of the training data and thus the number of conclusive rules becomes reasonable. Even one noisy data point would cause an enormous increase in the number of created rules if the rules were obtained based on the training data instead of the network outputs. Here we assume that the network is trained with a sufficiently large final MSE δ to allow smooth filtration of data.

Moreover, by choosing the parameter ε , the number

π_1	π_2	π_3	π_4	q_1	q_2	q_3
long	wide	medium	medium	no	—	—
long	thin	short	medium	no	yes	—
\neg long	thin	medium	thin	no	yes	—
\neg short	wide	long	medium	no	—	—
\neg long	\neg wide	short	wide	no	yes	no
\neg long	\neg wide	long	thin	no	yes	no
\neg short	wide	\neg short	wide	no	no	yes
—	\neg wide	\neg short	\neg thin	no	—	—
\neg long	wide	medium	medium	yes	—	—
\neg long	—	short	thin	yes	no	—
\neg long	\neg thin	short	medium	yes	no	—
\neg long	\neg thin	medium	thin	yes	no	—
long	thin	medium	medium	—	no	—
long	wide	long	medium	—	no	yes
—	wide	medium	medium	—	no	—
—	\neg wide	long	medium	—	no	yes
—	\neg wide	\neg short	wide	—	no	yes
medium	thin	medium	medium	—	yes	—
\neg short	medium	medium	medium	—	yes	—
short	wide	medium	medium	—	—	no
long	medium	medium	medium	—	—	no
medium	—	medium	medium	—	—	no
\neg long	—	short	medium	—	—	no
\neg long	—	\neg long	thin	—	—	no
long	\neg medium	medium	medium	—	—	yes

Figure 2: Rules created for the IRIS problem with $\varepsilon = 0.01$.

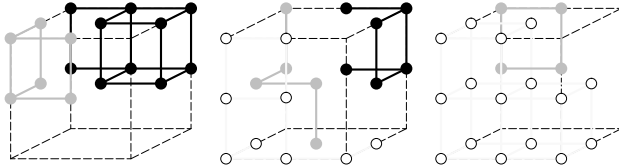


Figure 3: Graphic representation of the rules providing classification.

of rules (after pruning the totally undecidable rules) can be adjusted; the number of rules is related to the accuracy of a fuzzy classifier using the created rules. The aim is to extract knowledge from the NN, not to replicate the classifier, since it is more important for the rules to be as compact as possible even if classification with such rules is less than optimum. In addition, the numeric complexity of the rule extraction algorithm increases with the number of fuzzy classes for each input and with the number of inputs with a factor larger than 1. However, the algorithm presented is relatively insensitive to the size of the training data, and even large data sets can be handled efficiently.

REFERENCES

[1] Y. O. Yoon, R. W. Brobst, P. R. Bergstresser, and L. L. Peterson, "A desktop neural network for dermatology diagnosis," *Journal of Neural Network Computing*, pp. 43–52, (Summer) 1989.

[2] J. M. Zurada, *Introduction to Artificial Neural Systems*. Boston: PWS, 1992.

[3] N. B. Karayiannis and A. N. Venetsanopoulos, "Decision making using neural networks," *Neurocomputing*, vol. 6, pp. 363–374, 1994.

[4] G. G. Towell, J. W. Shavlik, and M. O. Noordewier, "Refinement of approximately correct domain theories by knowledge-based neural networks," in *Proc. of the 8th Nat. Conf. on Artificial Intelligence*, (Boston, MA), pp. 861–866, MIT Press, 1990.

[5] G. G. Towell and J. W. Shavlik, "Extracting refined rules from knowledge-based neural networks," in *Machine Learning* (D. Kibler, ed.), pp. 71–101, 1993.

[6] M. W. Craven and J. W. Shavlik, "Learning symbolic rules using artificial neural networks," in *Proc. of the 10-th Int. Conf. on Machine Learning*, (Amherst, Mass.), pp. 73–80, June 27–29, 1994.

[7] J. W. Shavlik, "Combining symbolic and neural learning," in *Machine Learning* (D. Sleeman, ed.), pp. 321–331, 1994.

[8] M. Ishikawa, "Structural learning with forgetting," *Neural Networks*, vol. 9, no. 3, pp. 509–521, 1996.

[9] S. Yasui, A. Malinowski, and J. M. Zurada, "Convergence suppression and divergence facilitation: New approach to prune hidden layer and weights in feedforward neural networks," in *Proc. of IEEE International Symposium on Circuits and Systems*, vol. 1, (Seattle, WA), pp. 121–124, Apr. 29–May 3, 1995.

[10] A. Lozowski, T. J. Cholewo, and J. M. Zurada, "Crisp rule extraction from perceptron network classifiers," in *Proc. of the IEEE International Conference on Neural Networks*, (Washington DC), June 3–6, 1996.

[11] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[12] L. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22.

[13] R. Setiono and H. Liu, "Symbolic representation of neural networks," *IEEE Computer*, vol. 29, pp. 71–77, Mar. 1996.