Adaptive Network-Based Fuzzy Inference System (ANFIS) A Tensorflow 2.0 implementation

Gregor Lenhard[1]

***Abstract*— Developed in the early 1990s by Jyh-Shing Roger Jang, an adaptive network-based fuzzy inference system or adaptive neuro-fuzzy inference system (ANFIS) is a fuzzy inference system based on the framework of adaptive neural networks. Initially introduced by using a hybrid leaning procedure of gradient descent method and the least squares estimate (LSE), we are able to utilize more efficient optimization techniques in the meantime. For this purpose I implemented an ANFIS alogrithm based on Keras on top of Tensorflow 2.0. By implementing ANFIS in Tensorflow allows to specify, train and deploy the model in a deep leaning environment. My source code is publicly available at `https://github.com/gregorLen/MyAnfis`.**

I. ARCHITECTURE OF ANFIS

An adaptive network is a multilayer feed-forward network composed of several layers connected by direct links. A layer consists of a set of nodes, in which each node performs a particular function on its incoming signal to produce an output. In particular, nodes are parameterized functions associated with weights that may or may not be trained to improve the functions behavior towards the desired output. Changing these parameters changes the behavior of a single node as well as the overall behavior of the network. Links specify the direction of signal flow from one node to another and carry no weights. The ANFIS network consists of five layers, namely fuzzy layer (layer 1), rule layer (layer 2), normalization layer (layer 3), defuzzifycation layer (layer 4) and total output layer (layer 5). A full network example is shown in figure 1.

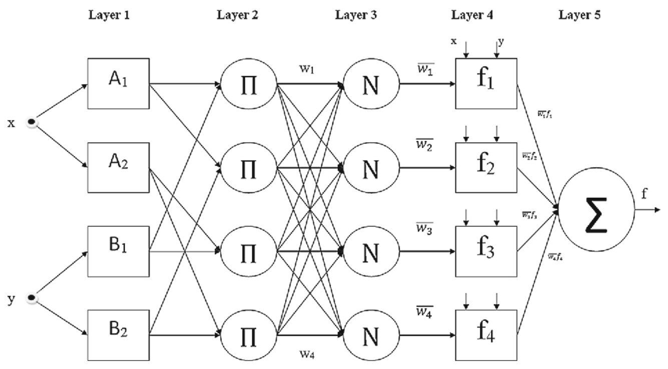

Fig. 1: Basic ANFIS architecture

My python implementation is built along these steps. With given input and output data, the ANFIS method models a fuzzy inference system (FIS) whose membership function parameters are tuned. The main objective of ANFIS is to find the optimal parameter values to fit a given problem as good as possible. This is done by optimization of the error margin between the target and the actual output during the tranining phase. In the original paper of Jang [1] these parameters are tuned by using either a backpropagation algorithm alone, or in permutation with a least squares type method. The implementation within a Tensorflow 2.0 [2] framework gives access to the full repository of Keras optimizers, e.g. *stochastic gradient descent* [3] or *ADAM* [4]. Other features now commin in deep learning, such as experimenting with mini-batching, learning-rate optimization algorithms and different loss functions [5] are also available.

Only parameters in layer 1 (fuzzy layer) and layer 4 (de-fuz layer) are trainable. Parameters of those layers are called *premise parameters* (in layer 1) and *consequence parameters* (in layer 4). Let's assume for simplicity there are only two inputs x and y as shown in figure 1. Premise parameters determine the memberships of a given input combination of x and y which lead to *rules* dependent on the respective memberships. This implementation considers a *Sugeno and Takagi*-type fuzzy inference system, i.e.:

Rule 1: If x is A_1 and y is B_1, then $f_1 = p_1x + q_1r + r_1$.
Rule 2: If x is A_2 and y is B_2, then $f_2 = p_2x + q_2r + r_2$.

Where p_1, p_2, q_1, q_2, r_1 and r_2 are linear *consequence parameters* and A_1, A_2, B_1 and B_2 are the non-linear membership functions which consist of the *premise parameters*. Nodes in layer 1 and layer 4 which are adaptive and therefore trainable are denoted by squared nodes, whereas circled nodes are non-trainable and indicate fixed nodes.

II. LAYERS OF ANFIS

For simplicity, the network under Figure 1 is considered.

Layer 1 (fuzzy layer)

This layer performs a *fuzzyification* of the incoming (crisp) values x and y by applying a *membership function* to obtain fuzzy clusters. Every node in this layer is an adaptive (square) node which produces a membership grade of linguistic label.

$$O_i^1 = \mu_{A_i}(x), \tag{1}$$

where x is the input to node i and A_i is the linguistic label (e.g. *low, medium, high*) associated with this node function. The output O_i^1 specifies the degree to which the given x satisfies the quantifier A_i. Usually, $\mu_{A_i}(x)$ is any kind of

[1]PhD student in Computational Economics and Finance, University of Basel, Switzerland Email: gregor.lenhard@unibas.ch

bell-shaped function with maximum equal to 1 and minimum of 0. In case of a generalized bell-shaped function, this would be:

$$\mu_{A_i}(x) = gbellmf(x; a, b, c) = \frac{1}{1 + [(\frac{(x-c_i)}{a_i})^2]^b}, \tag{2}$$

or the Gaussian functions:

$$\mu_{A_i}(x) = gaussmf(x; a, c) = exp[-(\frac{x - c_i}{a_i})^2], \tag{3}$$

where a_i, b_i, c_i (or a_i, c_i in the Gaussian case) are the *premise parameters* which are tuned during the training epochs.

Layer 2 (rule layer)

Every node in the rule layer is a fixed (circle) node labeled with Π in figure 1. The so-called *firing strength* w_i of each rule is determined here. Each node receives input values μ_{A_i} from the first layer and represents the fuzzy sets (or rules) of the respective input values. Firing strenghts for the rules are generated by multiplying the membership values as the following.

$$O_i^2 = w_i = \mu_{A_i}(x) \times \mu_{B_i}(y), i = 1, 2. \tag{4}$$

Generally, any T-norm operator that performs a fuzzy AND rule can be used as a nod function in this layer. An alternative fuzzy AND would be

$$O_i^2 = w_i = min(\mu_{A_i}(x), \mu_{B_i}(y))i = 1, 2. \tag{5}$$

Layer 3 (normalization layer)

Each node in this layer is a fixed (circle) node labeled with N. The normalization layer calculates normalized firing strengths belonging to each rule which is the ratio of the firing strength of the i-th rule to the total of all firing strengths.

$$O_i^3 = \bar{w_i} = \frac{w_i}{w_1 + w_2 + w_3 + w_4} i \in \{1, 2, 3, 4\} \tag{6}$$

For convinience, outputs of this layer will be called as *normalized firing strengths*.

Layer 4 (defuzzifycation layer)

Every node in this layer is an adaptive (square) node. As inputs, this layer receives normalized firing strengths as well as the original inputs x and y themselves (as shown in figure 1). This layer is called as defuzzification layer and returns weighted values of each rule's node by

$$O_i^4 = \bar{w_i} f_i = \bar{w_i}(p_i x + q_i y + r_i). \tag{7}$$

Notice that $\bar{w_i}$ is the output of the normalized firing strength. The weights in this layer $\{p_i, q_i, r_i\}$ are tuned during the training process. They are called *consequence parameters*. For each rule, there is a weight for each input (p_i and q_i) plus a bias r_i. Consequently the number of consequence parameters for each rule is one more than the number of inputs.

Layer 5 (total output layer)

The output layer is a single fixed (circle) node and produces the final output of ANFIS. It is obtained by summing the outputs received by for each rule in the defuzzifcation layer.

$$O_i^5 = \sum_i \hat{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \tag{8}$$

Knowing the ANFIS architecture as well as all its building blocks, let's review figure 1: it shows shows a 2-input ANFIS with 4 rules. Each input is associated with two membership functions that is A_1 and A_2 for x and B_1 and B_2 for y. So the input space is partitioned into 4 fuzzy subspaces or rules. The premise part of a rules defines a fuzzy subspace, while the consequence part specifies the output within this fuzzy subspace.

III. ANFIS TRAINING APPROACHES

work in progress ...
work in progress ...
work in progress ...
On ANFIS traning, see [6].

IV. APPLICATIONS OF ANFIS

work in progress ...
work in progress ...
work in progress ...
work in progress ...

V. SOURCE CODE

Source code for ANFIS Tensorflow[7] implementation including a sandbox and experiments are available at `https://github.com/gregorLen/MyAnfis`.

REFERENCES

[1] J.-S. Jang, "Anfis: adaptive-network-based fuzzy inference system," *IEEE transactions on systems, man, and cybernetics*, vol. 23, no. 3, pp. 665–685, 1993.

[2] "Effective tensorflow 2 : Tensorflow core."

[3] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

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

[5] "Loss functions."

[6] D. Karaboga and E. Kaya, "Adaptive network based fuzzy inference system (anfis) training approaches: a comprehensive survey," *Artificial Intelligence Review*, vol. 52, no. 4, pp. 2263–2293, 2019.

[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

VI. ACKNOWLEDGEMENTS

[filler for acknowledgements].