ANFIS vignette

Cristobal Fresno[1,2] and Elmer A. Fernández[1,2]

[1]Bioscience Data Mining Group, Catholic University of Cordoba
[2]CONICET, Argentina

April 23, 2012

Abstract

Adaptative-Network-Based Fuzzy Inference System (ANFIS) were first introduced by *Jang* 1993 [2] and recently used in different applications as regressors and/or classification systems [1]. Here, an implementation of ANFIS is presented as a R library within this vignette by one regressor and one classification example.

1 Introduction

Multi-Layer Neural Networks have neurons which model different kind of activation functions $f(.)$ in response to synapses strength, i. e, $f(\sum_{i=0}^{inputs} x_i w_i)$ where the i-th input x_i is weighted by w_i. However, ANFIS uses different kind of neurons to model fuzzy precedent-consequent rules usually of the type if-then. The result is usually a weighted linear combination of the inputs by the fuzzy membership $\mu * (\sum_{i=0}^{inputs} x_i a_i)$, where a_i is the i-th coefficient for the i-th input x_i. This type of rules are known in the literature as Takagi and Sugeno 1983 [4] type-3 fuzzy rules. In order to work, ANFIS uses two type of nodes: fixed (circles) and adaptive (squares) as seen in figure 1.

The first layer uses fuzzy neurons (with its associated parameters), defining a partition in the input space. Whenever a pattern $\mathbf{x} = (x, y)$ is presented, the respective membership functions $\mu_{A_1}(x), ..., \mu_{B_2}(y)$ are obtained. Then the second layer, obtains the rule associated weight w_i as the product of membership members members $\prod_{j \in R_i} \mu_j$ for each rule R_i. The third layer obtains the rule power $\bar{w}_i$ by means of normalization N of the rules. This power is used to weight the next layer linear combination output $f_i = \bar{w}_i(p_i x + q_i y + r_i)$, in order to obtain the final output as the sum of them $\sum$.
The parameters of the fuzzy nodes are known as *premises*, while the associated to the linear combination as *consequents*. It worth notices that premises are usually set by an expert, because the represent transferable domain knowledge. In addition membership functions must be "*well-formed*" like a Gaussian shape function and satisfy $\varepsilon - completeness$, in order to approximate any non-linear function (Stone-Weierstrass theorem [2]). This condition is satisfied when at least one $\mu_{A_i}(x) \geq \varepsilon = 0.5$, which implies an overlap between the input membership functions. On the other hand, depending on the learning rule

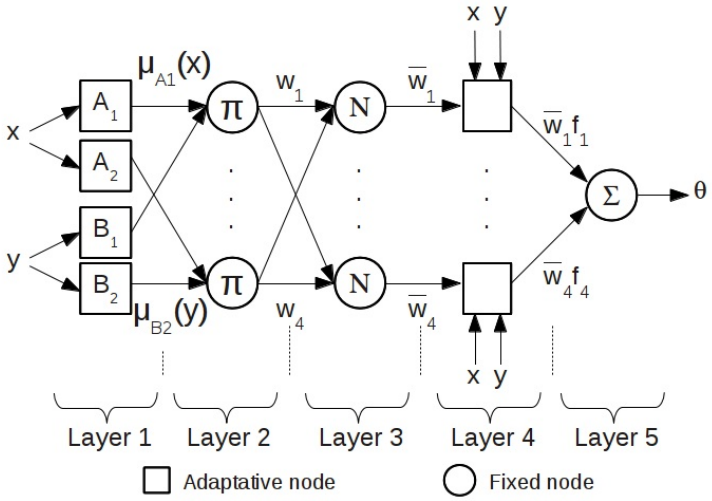


Figure 1: ANFIS example with two inputs, two membership functions in each input and four rules.

the consequents can be set and updated in different ways: gradient descendant (randomly) or hybrid learning (calculated). In this implementation we choose the hybrid learning.

1.1 Hybrid learning

Provided that ANFIS' coefficients can be written as a partition $S = S_{premices} \oplus S_{consequents}$, Jang [2] propose to use a *hybrid supervised learning* scheme. In this context, the premises are updated by a gradient descendant fashion, while the consequents are approximated by the solution of the linear system $AX = B$ where X is the consequents vector.
The learning consists in pushing forward the patterns until the fourth layer with *fixed* $S_{premices}$ and obtain the $S_{consecuentes}$ with least square approximation (LSE)[3] using (1)

$$\begin{array}{rcll} S_{i+1} & = & \frac{1}{\lambda}\left(S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}}\right), & X_0 = 0_{1xP}, S_0 = \gamma I_{MxM} \\ X_{i+1} & = & X_i + S_{i+1} a_{i+1}(b_{i+1}^T - a_{i+1}^T X_i), & \gamma \gg 0, i = 0, 1, ..., P-1 \end{array} \quad (1)$$

where S_i is the covariance matrix, P is the total pattern count, M the number of consequents, a_i^T and b_i^T the i-th row of A and B respectively and $0 < \lambda \leq 1$ el memory factor. Once they are determined the network output is obtained θ^5, in order to back propagate the cost E to update the premises coefficients α with the gradient descendant method (2) and learning coefficient η.

$$\begin{array}{l} E = \sum_p \sum_i (\xi_{ip} - \theta_{ip}^5)^2 \\ \frac{\partial E}{\partial \alpha} = \sum_p \sum_i (\xi_{ip} - \theta_{ip}^5)(-2) \sum_j^R (\sum_n x_n a_{jn}) \left\{ \frac{\frac{\partial w_j}{\partial \alpha} \sum_{k \in R} w_k - w_j \sum \frac{\partial w_j}{\partial \alpha}}{(\sum w_k)^2} \right\} \\ \frac{\partial w_j}{\partial \alpha} = \frac{\partial MF}{\partial \alpha}|_x \prod_{m \neq f(\alpha)} MF_m(x) \\ \Delta_\alpha = -\eta \frac{\partial E}{\partial \alpha} \end{array} \quad (2)$$

2 Case study: bidimentional $Sinc(x, y)$

The goal is to train an ANFIS in order to approximate a bidimentional real function (3). A regular grid of $p = 121$ points in a rectangular region $R_{(x,y)}$ (figure 2) is defined as the supervised training set X, ξ of (3). The cost function $E = \sum_{\mu=1}^{P} (\xi_\mu - \theta_\mu)^2$ is to be minimize with the hybrid learning algorithm, i. e., LSE for the consequents and descendant gradient for the premises. The premises update is carried out using an adaptative learning coefficient η according to (4), where the intial parameter $k = 0.1$ is set and α represent a given coefficient to be updated.

$$\zeta(x, y) = sinc(x)sinc(y), R_{(x,y)} = \{(x, y)/(x, y) \in [-10, 10] \times [-10, 10]\} \quad (3)$$

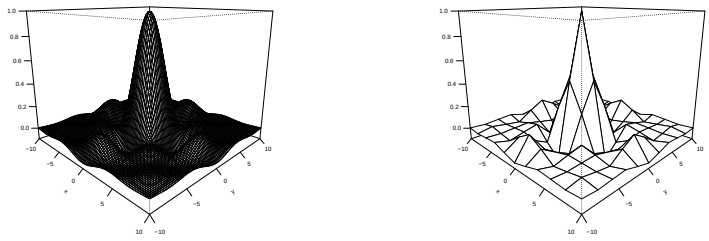


Figure 2: Function domain (3). Left) discretización with $p = 1000$. Right) discretización with $p = 121$.

$$\Delta_\alpha = -\eta \frac{\partial E}{\partial \alpha}; \eta = \frac{k}{\sqrt{\sum_\alpha \left(\frac{\partial E}{\partial \alpha}\right)^2}}; k = \begin{cases} 1.1k & \Delta_\alpha^{t-3} > \Delta_\alpha^{t-2} > \ldots > \Delta_\alpha^{t} \\ 0.9k & if \begin{pmatrix} \Delta_\alpha^{t-4} < \Delta_\alpha^{t-3} \wedge \\ \Delta_\alpha^{t-3} > \Delta_\alpha^{t-2} \wedge \\ \Delta_\alpha^{t-2} < \Delta_\alpha^{t-1} \wedge \\ \Delta_\alpha^{t-1} > \Delta_\alpha^{t} \end{pmatrix} \\ k & remaining_cases \end{cases} \quad (4)$$

2.1 Example 1: using 4 bell functions in each input

Four bell fuzzy functions (5) with parameter a, b and c are to be used in each of the two inputs.

$$\mu_{A|B_i}(x) = \frac{1}{1 + \left[\left(\frac{x - c_i}{a_i}\right)^2\right]^{b_i^2}} \quad (5)$$

The training set can be obtained with the following R commands:

```
> library(anfis)
> options(cores=4) #Setting 4 cores as default value for multicore
```

```
> trainingSet <- trainSet(seq(-10, 10, length= 11),seq(-10, 10, length= 11))
> X <- trainingSet[,1:2]
> Y <- trainingSet[,3,drop=FALSE]
```

Then, in order to create an ANFIS object we first need to define de structure of each input in terms of membership function. Only then we can call the initialize S4 class method by the command new.

```
> membershipFunction <- list(
+   x=c(new(Class="BellMF",parameters=c(a=4,b=1,c=-10)),
+       new(Class="BellMF",parameters=c(a=4,b=1,c=-3.5)),
+       new(Class="BellMF",parameters=c(a=4,b=1,c=3.5)),
+       new(Class="BellMF",parameters=c(a=4,b=1,c=10))),
+   y=c(new(Class="BellMF",parameters=c(a=4,b=1,c=-10)),
+       new(Class="BellMF",parameters=c(a=4,b=1,c=-3.5)),
+       new(Class="BellMF",parameters=c(a=4,b=1,c=3.5)),
+       new(Class="BellMF",parameters=c(a=4,b=1,c=10))))
> anfis <- new(Class="ANFIS",X,Y,membershipFunction)
> anfis

ANFIS network
Trainning Set:
         dim(x)= 121x2
         dim(y)= 121x1
Arquitecture:  2 ( 4x4 ) - 16 - 48 ( 48x1 ) - 1
Network not trained yet
```

The previous output shows an ANFIS with 2 inputs with $(4x4)$ four membership functions in each input, giving 16 rules, 48 total consequents in a $(48x1)$ matrix and 1 ouput node. Also notice that the model is **not trainned yet**. In adittion, the $\varepsilon - completeness$ can be graphically checked for the inputs (figure 3).

```
> plotMFs(anfis)
```

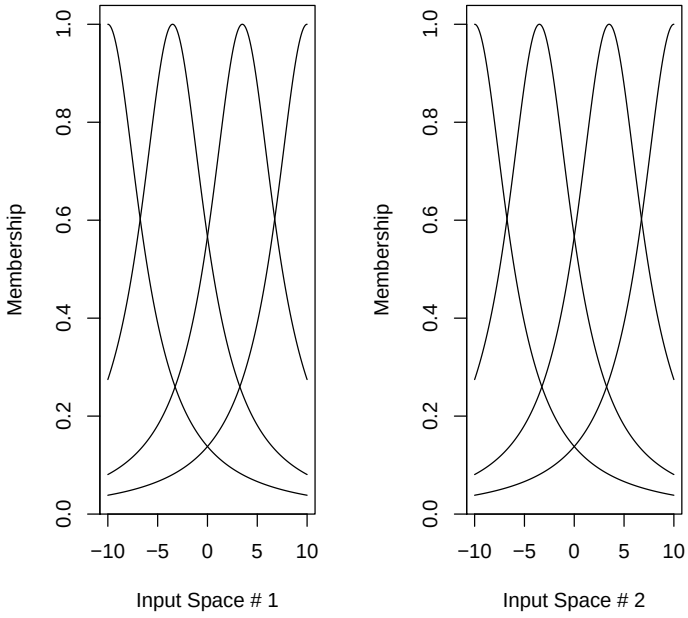


Figure 3: membership input domain fuzzy partition

Now we are ready to train the network using the hybrid off-line supervised algorithm with Jang adaptative learning rate, and inspect some of the slots of the class to see if premises and consequents where updated.

```
> trainOutput <- trainHybridJangOffLine(anfis, epochs=27,
+             tolerance=1e-5, initialGamma=1000, k=0.01)

> getPremises(anfis)[[input=1]][[premise=1]]

MembershipFunction:  Bell Membership Function
Number of parameters: 3
        a         b         c
 4.096336  1.424759 -9.859497
Expression: expression(1/(1 + (((x - c)/a)^2)^(b^2)))

> getConsequents(anfis)[1:2,] #First 2 consequents

[1] -0.05231742 -0.05231742

> getErrors(anfis) #Training errors

 [1] 1.198835 1.196361 1.193836 1.191252 1.188335 1.185016 1.181204 1.176757
 [9] 1.171449 1.164842 1.155866 1.139594 1.120668 1.117384 1.114357 1.111489
[17] 1.108699 1.105904 1.103004 1.099889 1.096505 1.093029 1.090474 1.088865
[25] 1.088039 1.087993

> getTrainingType(anfis)

[1] "trainHybridJangOffLine"
```

or the classical model values for R models such as seen in *nnet* or *lm*, which also works for ANFIS class objects.

```
> names(coef(anfis))

[1] "premises"    "consequents"

> coefficients(anfis)$premises[[input=1]][[mf=1]]

MembershipFunction:  Bell Membership Function
Number of parameters: 3
        a         b         c
 4.096336  1.424759 -9.859497
Expression: expression(1/(1 + (((x - c)/a)^2)^(b^2)))

> coefficients(anfis)$consequents[1:2,]

[1] -0.05231742 -0.05231742

> fitted(anfis)[1:5,]

[1] -0.010704785 -0.008268002  0.017956414  0.016553445 -0.033262425

> residuals(anfis)[1:5,]
```

```
[1]  0.013664374  0.001540105 -0.015422948 -0.006260532  0.008528575

> summary(anfis)

ANFIS network
Trainning Set:
         dim(x)= 121x2
         dim(y)= 121x1
Arquitecture:  2 ( 4x4 ) - 16 - 48 ( 48x1 ) - 1
Last trainnig error:  1.087993

Call: trainHybridJangOffLine(object = anfis, epochs = 27, tolerance = 1e-05,
    initialGamma = 1000, k = 0.01)

Statistics for Off-line training
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.088   1.101   1.119   1.137   1.180   1.199
```

Now we can inspect how the training process went by ploting the training error of the $anfis$ object and if we still have $\varepsilon - completeness$ (figure 4). In conclusion the network is too small, because it cannot gather the domain rules leaving the central region un mapped.

```
> par(mfrow=c(1,2))
> plot(anfis)
> plotMF(anfis,seq(from=-10,to=10,by=0.01),input=1,
+        main="Membership Function")
```

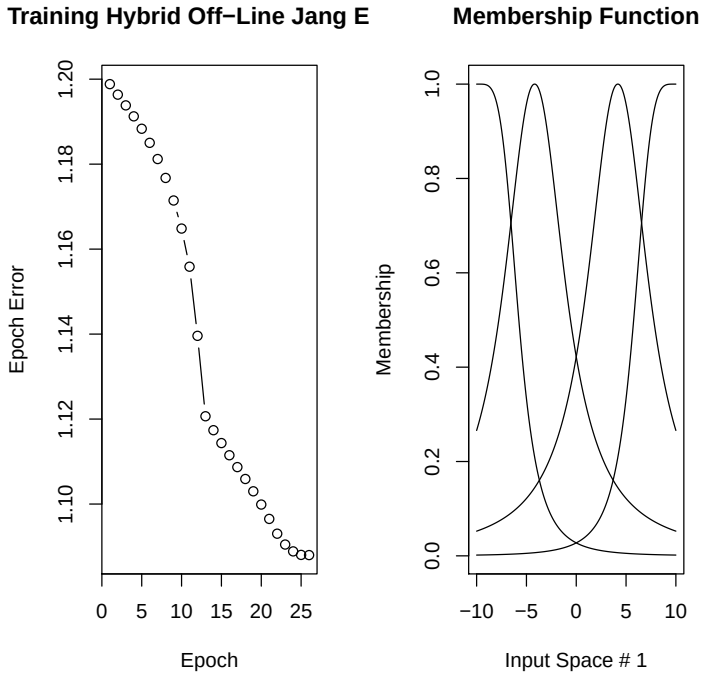


Figure 4: off-line training epoch errors and membership functions for input 1

2.2 Example 2: using 5 bell functions for each input

Let's include 5 bell functions in each input space also satisfing $\varepsilon - completeness$. In this example we also incorporate the stepwise learning feature of the implementation, i. e., the learning process can keep updating the network parameters. In this oportunity we also used the default values for *initialGamma* and *tolerance*.

```
> membershipFunction <- list(
+     x=c(new(Class="BellMF",parameters=c(a=4,b=1,c=-10)),
+         new(Class="BellMF",parameters=c(a=4,b=1,c=-5)),
+         new(Class="BellMF",parameters=c(a=4,b=1,c=0)),
+         new(Class="BellMF",parameters=c(a=4,b=1,c=5)),
+         new(Class="BellMF",parameters=c(a=4,b=1,c=10))),
+     y=c(new(Class="BellMF",parameters=c(a=4,b=1,c=-10)),
+         new(Class="BellMF",parameters=c(a=4,b=1,c=-5)),
+         new(Class="BellMF",parameters=c(a=4,b=1,c=0)),
+         new(Class="BellMF",parameters=c(a=4,b=1,c=5)),
+         new(Class="BellMF",parameters=c(a=4,b=1,c=10))))
> anfis2 <- new(Class="ANFIS",X,Y,membershipFunction)
> trainOutput <- trainHybridJangOffLine(anfis2, epochs=20, k=0.01)
> trainOutput <- trainHybridJangOffLine(anfis2, epochs=60,
+                   k=trainOutput$k)
```

Notice that if the MembershipFunctions properly partitionate the input space, it will keep on descending and will be a stable learning (figure 5)!!!!

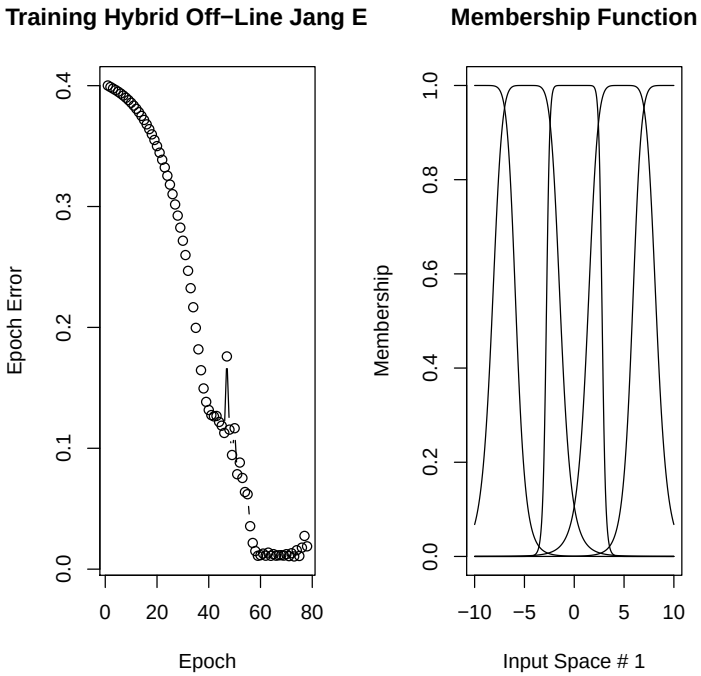


Figure 5: off-line training epoch errors and membership functions for input 1

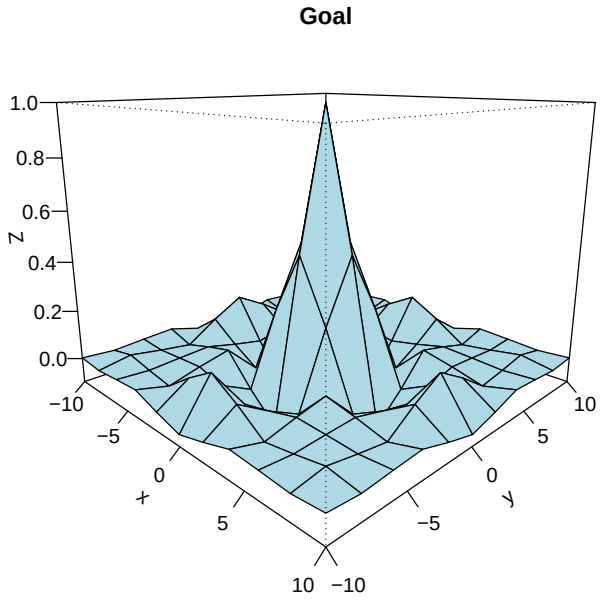
2.3 Example 3: using 5 normalized Gaussian functions for each input instead of bell

ANFIS implementation uses *membershipfunction* library which has a hierarchical S4 class structure. In this context, MembershipFunction is a virtual ancestral with heirs that extend the *evaluateMF* and *derivativeMF* functions. Thus, no additional modification of ANFIS class is required if we prefere to use normalized Gaussian membership functions instead. In this example we use 5 of them for each input and introduce *predict* network output function.

```
> membershipFunction <- list(
+     x=c(new(Class="NormalizedGaussianMF",parameters=c(mu=-10,sigma=2)),
+         new(Class="NormalizedGaussianMF",parameters=c(mu=-5,sigma=2)),
+         new(Class="NormalizedGaussianMF",parameters=c(mu=0,sigma=2)),
+         new(Class="NormalizedGaussianMF",parameters=c(mu=5,sigma=2)),
+         new(Class="NormalizedGaussianMF",parameters=c(mu=10,sigma=2))),
+     y=c(new(Class="NormalizedGaussianMF",parameters=c(mu=-10,sigma=2)),
+         new(Class="NormalizedGaussianMF",parameters=c(mu=-5,sigma=2)),
+         new(Class="NormalizedGaussianMF",parameters=c(mu=0,sigma=2)),
+         new(Class="NormalizedGaussianMF",parameters=c(mu=5,sigma=2)),
+         new(Class="NormalizedGaussianMF",parameters=c(mu=10,sigma=2))))
> anfis3 <- new(Class="ANFIS",X,Y,membershipFunction)
> trainOutput <- trainHybridJangOffLine(anfis3, epochs=10)
> y <- predict(anfis3,X)
> partition <- seq(-10, 10, length= 11)
> Z <- matrix(Y,ncol=length(partition),nrow=length(partition))
> z <- matrix(y[,1],ncol=length(partition),nrow=length(partition))

> par(mfcol=c(2,1))
> persp(x=partition,y=partition,Z,theta = 45, phi = 15, expand = 0.8,
+             col = "lightblue",ticktype="detailed",main="Goal",
+             xlim=c(-10,10),ylim=c(-10,10),zlim=c(-0.1,1),xlab="x",
+             ylab="y")
> persp(x=partition,y=partition,z,theta = 45, phi = 15, expand = 0.8,
+             col = "lightblue",ticktype="detailed",
+             main="Fitted training Patterns",xlim=c(-10,10),
+             ylim=c(-10,10),zlim=c(-0.1,1),xlab="x",ylab="y")
> graphics.off()
```

In conclusion we need to make a better chooice of the $MembershipFunction$ in order to accurate model the problem domain (figure 6)!!! In this context, the modification of membership evaluation not only reduced the amount of epochs required to obtain an acceptable cost E but, also reduced the amount of premises parameters by 10 (5 mf x 2 inputs x (3-2) parameter difference). This does not mean that bell functions are not usefull at all. They have other properties that may be needed in other problems.



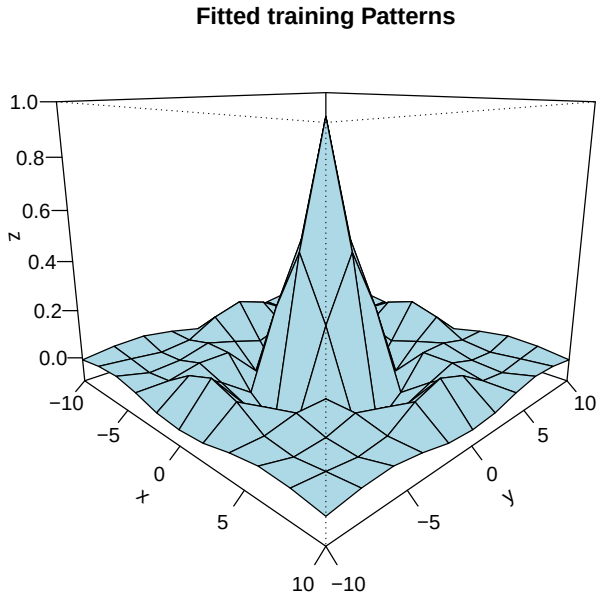


Figure 6: surface comparison for trainingSet of $Sinc(x, y)$ and network trained output

Now, if keep on training more some more epochs, we will keep on descending on the training error (figure 7). But, although $\varepsilon - completeness$ have been lost, we can use the MembershipFunctions in order to get a clue of domain !!!!

```
> trainOutput <- trainHybridJangOffLine(anfis3,epochs=43,k=trainOutput$k)
> y <- predict(anfis3,X)
> z <- matrix(y[,1],ncol=length(partition),nrow=length(partition))
```

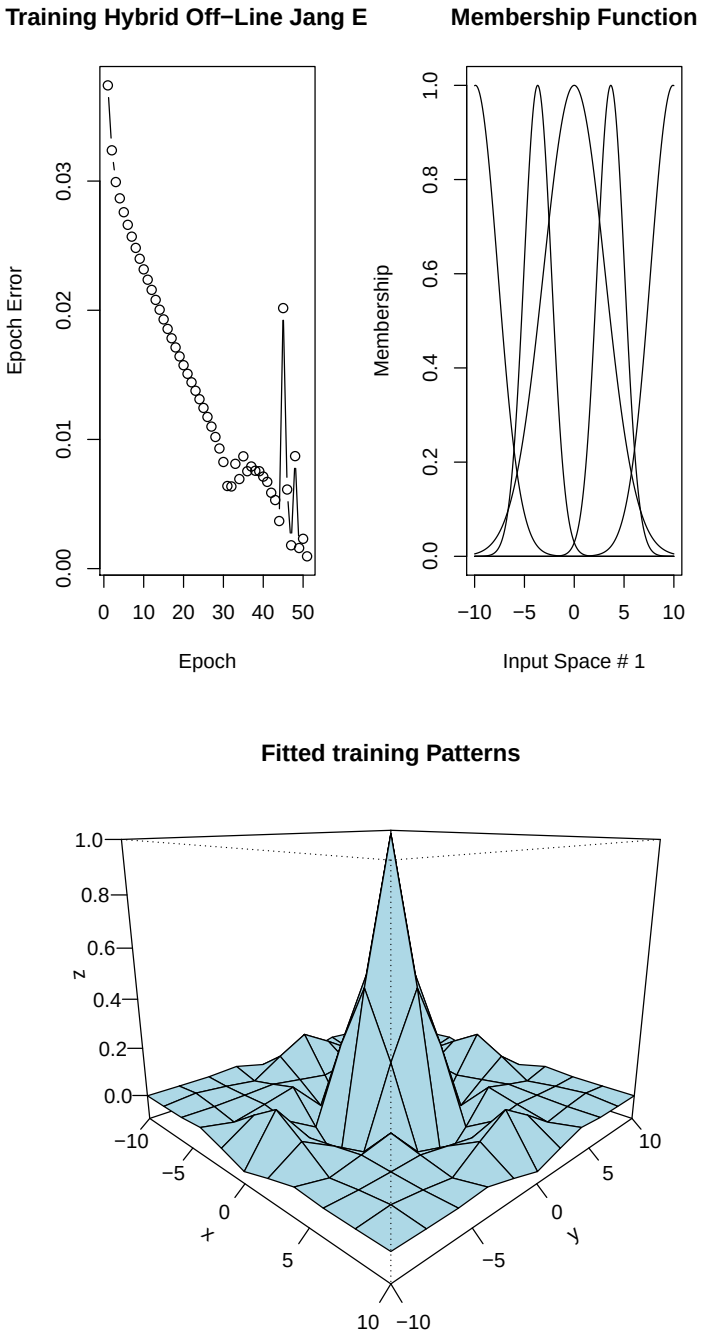


Figure 7: updated training error, membership and surface

3 Example 4: using 5 normalized Gaussian functions for each input with hybrid momentum learning and adaptative η

An other posibility is to use momentum term φ and adaptative η (6)

$$\Delta_\alpha^t = -\eta \frac{\partial E}{\partial \alpha} + \varphi \Delta_\alpha^{t-1}$$

$$\eta = \begin{cases} \eta + a & if \Delta_\alpha^t - \Delta_\alpha^{t-1} < 0 \\ a = \eta b; \eta = \eta * (1-b) & if \Delta_\alpha^t - \Delta_\alpha^{t-1} > 0 \end{cases} \tag{6}$$

where t stands for the actual iteration and $t-1$ the previous, a and b are increment step and precentage reduction for adaptative eta respectively. Notice that k is no longer needed and additional parameters are included as required by function call.

```
> anfis4 <- new(Class="ANFIS",X,Y,membershipFunction)
> trainOutput <- trainHybridOffLine(anfis4, epochs=5, tolerance=1e-5,
+               initialGamma=1000, eta=0.1, phi=0.2, a=0.1, b=0.1,
+               delta_alpha_t_1=list())
```

The results are equivalents to Jang's proposal, it's just another posibility. Indeed, Jang's adaptative η works faster than this alternative. However, the hybrid learning is not a fully gradient descendant method. This means that the network parameters do not follow a Liapunov bounded cost E, hence the training error path can increase making this alternative a suboptimal chioce to Jang's.

4 Example 5: example 3 with 2 outputs

The ANFIS implementation also handles multiple outputs. In this example the network is trainned with two output one with $Sinc(x, y)$ and the other with $1 - Sinc(x, y)$.

```
> anfis5 <- new(Class="ANFIS",X,cbind(Y[,1],1-Y[,1]),membershipFunction)
> anfis5

ANFIS network
Trainning Set:
         dim(x)= 121x2
         dim(y)= 121x2
Arquitecture:  2 ( 5x5 ) - 25 - 150 ( 75x2 ) - 2
Network not trained yet
```

Notice that the ANFIS structure has duplicate its consequents parameters due to the second output (75x**2**). This is posible because it has mantainned the same set of rules but duplicated the linear combination parameter for each rule power. Moreover, the same domain partition is required for this toy example (figure 8).

```
> trainOutput <- trainHybridJangOffLine(anfis5, epochs=10)
> y <- predict(anfis5,X)
> z1 <- matrix(y[,1],ncol=length(partition),nrow=length(partition))
> z2 <- matrix(y[,2],ncol=length(partition),nrow=length(partition))
```

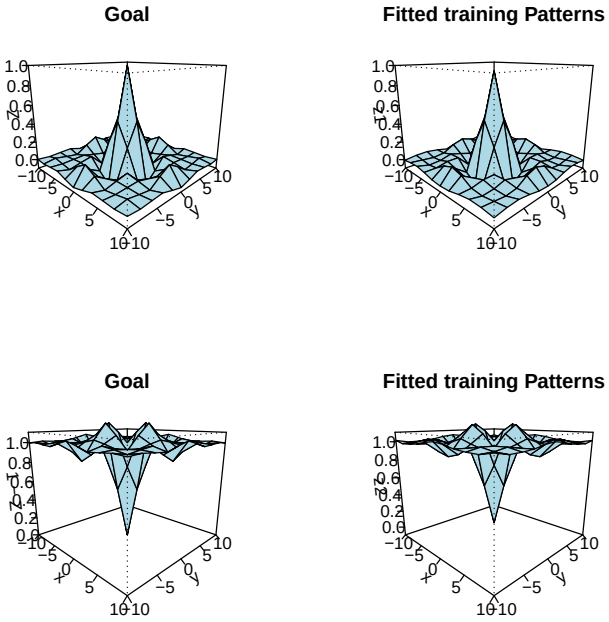


Figure 8: two ANFIS output predicted surfaces

5 Case study: iris classification

The well-known iris database is ussually used as a classification benchmarck. The dataset consists of 150 individuals and 4 atributes (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width) of tree species (setosa, versicolor, virginica) with 50 observations of each one. In this oportunity, we will compare neural network (*nnet*) library against ANFIS.

5.1 Example 6: nnet vs anfis

In order to give a fair comparison we will preform little modifications over the nnet example section, spliting the database in two parts: one for training and the other for testing. The same sets are used for both configurations.

5.1.1 NNET

A neural network with 4 inputs, 2 hiden nodes and 3 output nodes (4-2-3) with 19 weights is trained as follows. Notice that the ranom seed is set to 1 and a shuffing of the training set is also preformed.

```
> library(nnet)
> library(xtable)
> set.seed(1)
> #Database separation in halfs
> ir <- rbind(iris3[,,1],iris3[,,2],iris3[,,3])
> ir <- as.matrix(scale(ir))
> targets <- class.ind( c(rep("s", 50), rep("c", 50), rep("v", 50)) )
> samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
> samp <- sample(samp,length(samp))
```

Now we train the model using the default parameters and also define a scoring class functions for predicted classes, to make a confusion matrix.

```
> ir1 <- nnet(ir[samp,], targets[samp,], size = 2, rang = 0.1,
+         decay = 5e-4, maxit = 200)
> test.cl <- function(true, pred) {
+     true <- max.col(true)
+     cres <- max.col(pred)
+     table(true, cres)
+ }
> confusion <- test.cl(targets[-samp,], predict(ir1, ir[-samp,]))
> confusion
```

In table 1 the confusion matrix of the test set is presented. A test accuracy of 92% was achieved for this model.

5.1.2 ANFIS

Unlike neural networks, ANFIS grows in the parameter space much faster than the firsts (7)

	1	2	3
1	22	0	3
2	0	25	0
3	3	0	22

Table 1: confusion nnet matrix for test dataset where column (rows) stands for predicted (true) classes

$$Parameters = in * MF(in) * coef(MFs) + (MFs^{in}) * (in + 1) * out \quad (7)$$

where in stands for the number of inputs, $MF(.)$ is the number of membership functions in each input, $coef(.)$ is the number of coefficients for each membership function and out is the number of nodes in the output layer. Clearlly, it is not posible to make a straight comparison as seen in table 2.

	inputs	MFs	parameters
1	1	3	24
2	1	4	32
3	2	3	93
4	3	3	342
5	4	1	23
6	4	3	1239

Table 2: anfis parameters for different configurations with normalized Gaussian membership function, MFs per input and 3 outputs.

It worths to notice the following properties of ANFIS configurations:

1. If the number of membership functions in each input is one, i. e., $MF(in) = 1$ only one rule R exists. Hence, the rule power is always equal to one and the chain rule has no influence over the premises update (they remain constant). Consequently, if off-line training is applied more than one epoch will not make any improvement over network preformance.

2. In addition to 1, ANFIS tries to make domain input partitions. Thus, no sense make to use fewer membership functions.

3. The number of parameters grow linearly in the output space, whereas the true growth is in rule/input combination.

Now let's define an ANFIS with only input 3 and 3 Normalized Gaussian MFs, in order to be close to the 19 weights of the nnet model (23 in this case). In addition, an on-line implementation is introduce to make also a fair comparison. Notice the forgetting factor $\lambda = 0.99$ close to off-line (1) value and the need of an empty covariance matrix S.

```
> input <- 3
> X <- ir[samp,input,drop=FALSE]
```

```
> Y <- targets[samp,]
> membershipFunction <- list(
+   x=c(new(Class="NormalizedGaussianMF",parameters=c(mu=-1.5,sigma=1)),
+       new(Class="NormalizedGaussianMF",parameters=c(mu=0.08,sigma=1)),
+       new(Class="NormalizedGaussianMF",parameters=c(mu=1.67,sigma=1))))
> anfis6 <- new(Class="ANFIS",X,Y,membershipFunction)
> trainOutput <- trainHybridJangOnLine(anfis6, epochs=30,
+     tolerance=1e-15, initialGamma=1000, k=0.01, lamda=0.99,
+     S=matrix(nrow=0,ncol=0))
> confusion <- test.cl(targets[-samp,],
+                  predict(anfis6, ir[-samp,input,drop=FALSE]))
> confusion
```

	1	2	3
1	7	8	10
2	8	8	9
3	8	9	8

Table 3: confusion ANFIS matrix for test dataset where column (rows) stands for predicted (true) classes

In table 3 the confusion matrix of the test set is presented. A test accuracy of only 30.67% was achieved for this model, with a stable training epoch error as seen in figure 5.1.2.

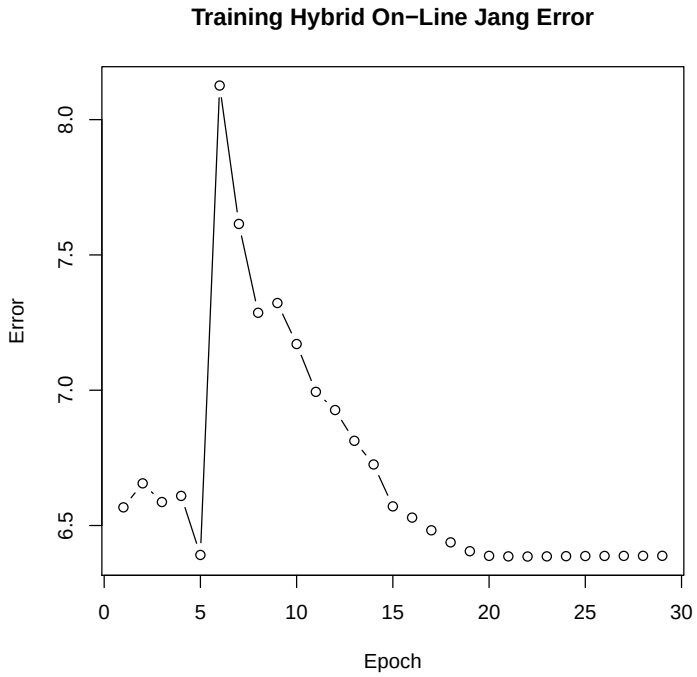


Figure 9: anfis online training epoch error for the third attribute input, 3 MFs and 3 outputs

References

[1] Hertz,J. Krogh,A. and Palmer,R.G ˜ (1990) *Introduction to the theory of neural computation*, Westview Press, Oxford, USA.

[2] Jang,J.S.R.˜(1993) *ANFIS: Adaptive-network-based fuzzy inference system*, IEEE Transactions on systems, man and cybernetics, **23**:3, 665-685.

[3] Jang,J.S.R. Sun,C-T, and Mizutani,E ˜ (1997)*Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence*, Prentice Hall, USA.

[4] Takagi,T. and Sugeno,M.˜(1984) *Derivation of fuzzy control rules from human operator's control actions*, Fuzzy information, knowledge representation, and decision analysis: proceedings of the IFAC Symposium, Marseille, France, 19-21 July 1983, pp 55-60.