New developments in the Feature Space Mapping model [1]

Rafał Adamczak, Włodzisław Duch, and Norbert Jankowski
Department of Computer Methods, Nicholas Copernicus University,
Grudziądzka 5, 87-100 Toruń, Poland.
E-mail: raad,duch,norbert@phys.uni.torun.pl

Abstract

Feature Space Mapping (FSM) model is based on a network explicitly modeling probability distribution of the input/output data vectors. New theoretical developments of this model and results of applications to several classification problems are presented.

1 Introduction

FSM model has been introduced recently [1] as a universal adaptive system based on multidimensional separable functions. Viewed from various perspectives FSM is a neurofuzzy system, a density estimation network, a memory-based system, a data modeling approach, an example of self-organizing system or even an expert system. The main idea is simple: components of the input and output vector define features, and combinations of these features define objects in the feature spaces. These objects are described by the joint density probability of the input/output data vectors using a network of properly parametrized transfer functions. Although this model may be presented in probabilistic framework as a density network it was originally inspired by cognitive model of mind as an approximation to real neurodynamics – high activity of the spiking neurons may be described by the probability density function in the feature space [2]. The importance of density estimation as the basis for neural systems has recently been stressed by many authors (cf. [3]).

Since its introduction the FSM model has been implemented and tested on a number of classification problems and has been used for extraction of logical rules. New theoretical developments, presented in the next section, include parametrization of rotated biradial transfer functions and improvements in the learning algorithm. Initialization of neural networks parameters, including initialization of the FSM model, is described in a separate paper (Duch, Adamczak, Jankowski, this volume). In the third section results of some applications to classification problems are presented.

2 FSM model

Preliminaries: given a set of training examples $\mathcal{D} = \{X^k, Y^k\}$ create a network $M(X, Y; \theta)$, where θ are parameters, which gives outputs approximating the landscape of joint probability density $p(X, Y|\mathcal{D})$. $M(X, Y; \theta)$ should be neither equal to, or proportional to, this density;

[1] Computational Intelligence Laboratory Technical Report 2/97. Shorter version submitted to PNNS'97 conference, Kule, Poland 14-18.10.1997

all that is required is that the maxima of conditional probabilities $Y_p(X) = \max_Y p(Y|X,\mathcal{D})$ and $X_p(Y) = \max_X p(X|Y,\mathcal{D})$ agree with the corresponding maxima of $M(X,Y;\theta)$ obtained by calculation of $Y_M(X) = \max_Y M(X,Y;\theta)$ and $X_M(Y) = \max_X M(X,Y;\theta)$. This task is simpler than the full estimation of joint or conditional probabilities. Bayesian solution is achieved by introducing *a priori* probability distributions of parameters θ (so far it has been possible only for weights in MLP networks) and marginalizing (integrating) over these parameters [9]. In practice committee of networks give results of similar accuracy and are computationally less expensive than mariginalization.

Below FSM model as used for classification problems is described. Since training of the MLP networks or other networks with fixed architecture is NP-hard [3], a robust constructive algorithm is used to build the FSM network.

Architecture: in the simplest case a single layer structure is used. In general a feedforward hierarchical structure is used to speed up searches and define categories (clusters) and supercategories (superclusters). Modular architecture is also useful if several qualitatively different sources of data are given as inputs [4]. Initial architecture is created using clusterization techniques (Duch, Adamczak, Jankowski, this volume) and optimized during learning process.

Transfer functions: there are no restrictions on the type of transfer functions in the FSM model. Good transfer functions $\phi(\mathbf{X},\theta)$ should offer the most flexible densities (flexible shapes of $\phi(\mathbf{X},\theta)$ =const contours) with the smallest number of adaptive parameters θ. Small network with a few complex nodes is equivalent to a large network with simple nodes. Recently we have reviewed various transfer functions suitable for neural networks [5]. The simplest functions with suitable properties for density modeling are of Gaussian or approximated Gaussian type. Our favorite functions are of the biradial type:

$$Bi(\mathbf{X};D,b,s) = \prod_{i=1}^{N} \sigma(e^{s_i}\cdot(X_i - D_i + e^{b_i}))(1 - \sigma(e^{s_i}\cdot(X_i - D_i - e^{b_i}))) \qquad (1)$$

where $\sigma(x) = 1/(1+e^{-x})$. The first sigmoidal factor in the product is growing for increasing input X_i while the second is decreasing, localizing the function around D_i. Shape adaptation of the density $Bi(\mathbf{X};D,b,s)$ is possible by shifting centers $\mathbf{D}$, spreads $\mathbf{b}$ and rescaling slopes $\mathbf{s}$. Radial basis functions are defined relatively to only one center $||\mathbf{X} - D||$. Here two centers are used, $\mathbf{D} + B$ and $\mathbf{D} - B$, therefore we call these functions biradial. Exponential form $B_i = e^{b_i}$ (and e^{s_i}) used instead of b_i (or s_i) parameters stabilizes learning preventing oscillations.

Biradial functions in product form are separable and give localized convex densities. The number of adjustable parameters per processing unit is in this case $3N$. Using two (or just one) new parameters allows for delocalization of biradial functions:

$$SBi(\mathbf{X};D,b,s,\alpha,\beta) = \prod_{i=1}^{N} (\alpha + \sigma(e^{s_i}\cdot(X_i - D_i + e^{b_i})))(1 - \beta\sigma(e^{s_i}\cdot(X_i - D_i - e^{b_i}))) \quad (2)$$

For $\alpha = 0$, $\beta = 1$ this function is identical to the biradial localized functions while for $\alpha = \beta = 0$ each component under the product turns into the usual sigmoidal function. Semi-local transfer functions SBi have $3N + 1$ parameters (if $\beta = 1 - \alpha$), $3N + 2$ parameters (independent α, β) or up to $5N$ parameters (different α_i and β_i used in each dimension). In our simulations we have found that biradial functions lead to faster convergence with smaller number of nodes than Gaussian functions [5].

The function realized by the FSM network represents crisp logical rules if the contours of constant density are defined by cuboids. Fuzzy logical rules require probability densities described by separable functions, with each one-dimensional component equivalent to a membership function (specific for a given data cluster). Increasing the slopes s_i of biradial functions allows for a smooth transition from complex to cuboidal density contours. For crisp logical rule extraction we have also used rectangular functions defined by:

$$G(\mathbf{X}; D, \sigma) = \prod_i^N G(X_i; D_i, \sigma_i) = 1 \text{ if all } |D_i - X_i| < \sigma_i \text{ else } 0 \tag{3}$$

Next step towards even greater flexibility requires individual rotation of densities provided by each unit. Of course one can introduce a rotation matrix operating on the inputs $\mathbf{R} \cdot X$, but in practice it is very hard to parametrize this $N \times N$ matrix with $N - 1$ independent rotation angles and calculate derivatives necessary for error minimization procedures. We have found two ways to obtain rotated densities in all dimensions using transfer functions with just N additional parameters per network node. In the first approach product form of the combination of sigmoids is used

$$\begin{aligned} C_P(\mathbf{X}; D, D', R) &= \prod_i \Big(\sigma(\mathbf{R}_i\mathbf{X} + D_i) - \sigma(\mathbf{R}_i\mathbf{X} + D'_i)\Big) \\ SC_P(\mathbf{X}; D, D', R, \alpha, \beta) &= \prod_i \Big(\alpha \cdot \sigma(\mathbf{R}_i\mathbf{X} + D_i) + \beta \cdot \sigma(\mathbf{R}_i\mathbf{X} + D'_i)\Big) \end{aligned} \tag{4}$$

where $\mathbf{R}_i$ is the i-th row of the rotation matrix $\mathbf{R}$ with the following structure:

$$\mathbf{R} = \begin{bmatrix} s_1 & \alpha_1 & 0 & & \cdots & 0 \\ 0 & s_2 & \alpha_2 & 0 & & \\ \vdots & & & \ddots & & \vdots \\ & & & & s_{N-1} & \alpha_{N-1} \\ 0 & \cdots & & & 0 & s_N \end{bmatrix} \tag{5}$$

If $\alpha = 1$ and $\beta = -1$ the SC_P function is localized and gives similar densities as the biradial functions. Choosing other values non-local rotated transfer functions are created.

In the second approach the density is created by the sum of a "window-type" combinations of sigmoids $L(x; D, D') = \sigma(x + D) - \sigma(x + D')$ in $N - 1$ dimensions and a combination rotated by a vector $\mathbf{K}$:

$$C_K(\mathbf{X}; D, D', W, K) = \sum_{i=1}^{N-1} W_i L(X_i, D_i, D'_i) + W_N L(\mathbf{K} \cdot X, D_N, D'_N) \quad (6)$$

The last density is perpendicular to the $\mathbf{K}$ vector. Treating $C_K(\cdot)$ as the activation function and using sigmoidal output function with a proper threshold $\sigma(C_K(\cdot) - b)$ leaves only the non-zero densities in the direction perpendicular to $\mathbf{K}$. These densities are not hyperelipsoidal, but sufficiently flexible to model complex clusters. An alternative is to use the product form:

$$C_{PK}(\mathbf{X}; D, D', W, K) = L(\mathbf{K} \cdot X, D_N, D'_N) \prod_{i=1}^{N-1} L(X_i, D_i, D'_i) \quad (7)$$

as the transfer function – the output sigmoid is not needed in this case. Rotation adds only $N-1$ parameters for $C_P(\cdot)$ function and N parameters for $C_K(\cdot)$ function.

Learning algorithm: There is a tradeoff between complexity of the network (proportional to the number of network nodes needed to achieve certain accuracy) and complexity of the transfer functions. Overall complexity, counted using the total number of adaptive parameters, should be minimized. In the constructive algorithm this is rather simple: after initialization and expansion of the network in the first training epochs accuracy of classification is checked on the test or validation dataset. At some point this accuracy starts to decrease although the accuracy on the training data still decreases. At this point complexity of the model is optimal and the training is stopped.

Initial structure of the network includes input and output units (one output unit or one output unit per class) and a single layer of hidden units with parameters determined by a clustering algorithm [7]. For on-line learning initialization is performed after a fixed number of incoming training data is collected. Parameter t numbers the training epochs, parameter τ_k is a "local time", growing differently for each network node. One of the problems with RBF networks is their inability to select relevant input features. In FSM feature selection is performed by adding penalty term for small dispersions to the error function:

$$E(V) = E_0(V) + \lambda \sum_{i}^{N} 1/(1+\sigma_i) \quad (8)$$

where $E_0(V)$ is the standard quadratic error function and V represents all adaptive parameters, such as positions and dispersions $\sigma_i = |b_i - b'_i|$ of localized units. The sum runs over all active inputs for the node that is the most active upon presentation of a given training vector. The penalty term encourages dispersions to grow – if σ_i includes the whole range of input data the connection to x_i is deleted. Formally we could assign the weights $W_i = 1/(1+\sigma_i)$ to the X_i input connections and minimize the sum of weights. After the training each node has class label and has localized density in relevant dimensions of the input subspace (and constant density in irrelevant dimensions). An alternative approach to modification of the error function is to expand dispersions after several training epochs as much as possible without increasing the classification error. After initialization of the FSM architecture and parameters learning algorithm proceeds as follows:

1. Increase all dispersions until $M(X;P) > 0.5$ for all training vectors X; in this way approximation to the probability density $p(X|\mathcal{D}$ is more smooth and nodes covering outliers are not created.

2. Estimate dispersions σ_{ini} for each class – used to create new nodes. Estimated dispersions are slowly decreased during training (this is a form of simulated annealing).

3. Start of the learning epoch: read new data vector X; find the node N_m that is maximally active; find the closest node N'_m to the maximally active node, N'_m of the same class as X.

4. If N'_m and N_m belong to the same class C and the activity of $G(N_m)$ is greater than a given threshold (experience shows that 0.6 is a good value) there is no need for further optimization and the program goes back to step 3; otherwise parameters of the N_m node are optimized:

$$
\begin{aligned}
m_n &\leftarrow m_n + 1 && (9)\\
W_n &\leftarrow W_n + \eta \cdot (C - M(\mathbf{X}; D, \sigma)) \cdot \nabla_W M(\mathbf{X}; D, \sigma) && (10)\\
D_{nj} &\leftarrow D_{nj} + \gamma(t) \cdot (X_j - D_{kj})/m_k && (11)\\
\sigma_{nj} &\leftarrow \sigma_{nj} + \alpha(t)(1 - G(\mathbf{X}; D_n, \sigma_n))|X_j - D_{nj}| && (12)\\
\alpha(t) &\leftarrow \Lambda(1 + (t - \tau_k)/\epsilon)^{-2} && (13)\\
\gamma(t) &\leftarrow \Gamma(1 + (t - \tau_k)/\epsilon)^{-1} && (14)
\end{aligned}
$$

   Here ϵ is about 100 (time in epochs) and is constant, Λ and Γ are also constants. The "mass" m_n of nodes is set to zero at the beginning of each epoch. Node that is selected frequently gains a large mass and its parameters are changed more slowly.

5. If N'_m and N_m belong to different classes check two conditions: is $||\mathbf{X} - D(N_m)|| > \sigma(N_m)\sqrt{ln(10)}$, i.e. is the new vector $\mathbf{X}$ sufficiently far from the center of the nearest cluster? If yes, check $G(N_m) <$ Min$_{act}$, i.e. does the activity of the node exceed certain minimum? Min$_{act}$ is set to 0.2 at the beginning of training and after some time, when learning slows down, it is decreased by two. If both conditions are fulfilled create new node;
   otherwise go back to 3.

6. Create new node: initial parameters are: $\mathbf{D}_c = X$; $W_c = C - M(\mathbf{X}; D, \sigma)$. Initial values of dispersion components σ_{ci} are set to σ_{ini} if $|D_{ci} - D_{ni}| > \sigma_{ni} + \sigma_{ini}$, i.e. if the nearest center is far enough to assume standard initial dispersion; otherwise $\sigma_{ci} = |D_{ci} - D_{ni}|$.

7. The second most excited node is also optimized if it belongs to a different class than the most excited node. Three cases are distinguished: if the vector $\mathbf{X}$ is within the range of this node, i.e. $||\mathbf{X} - D||) < \sigma$, dispersion of the node is decreased:

$$\sigma_k = \sigma_k - \vartheta \frac{G(\mathbf{X}; D, \sigma)(|X_k - D_k|) - \sigma_k)}{1 - (t-\tau)/\epsilon}; \quad \tau = \tau + (t-\tau)/2 \tag{15}$$

If the vector X is not within the range of the second most excited node, and the node's activity $G(X; D, \sigma)$ is greater than some threshold (we use 0.6 for this threshold):

$$\sigma_k = \sigma_k - \vartheta \frac{G(\mathbf{X}; D, \sigma)\sigma_k}{1 - (t-\tau)/\epsilon}; \quad \tau = \tau + (t-\tau)/2 \tag{16}$$

Finally if the node is far and is not excited so strongly its dispersion is decreased by:

$$\sigma_k = \sigma_k - G(\mathbf{X}; D, \sigma)\sigma_k; \quad \tau = \tau + (t-\tau)/2 \tag{17}$$

Remarks on the learning procedure:

Increasing the number of nodes leads to 100% classification accuracy on the training set, overfiting the data. The simplest way to avoid it is to assume lower goal for accuracy and check the performance on a test dataset. This requires several stops and checks while the networks adapts itself more and more closely to the data. An alternative approach based on maximum certainty [8] requires only training data but a committee of several networks should be created.

After the training epoch is finished the quality Q_k of each node k is estimated by dividing the number of correctly classified vectors through the number of all vectors handled by this node (i.e. nodes that the winners and their excitation exceeds certain threshold). Classification results may improve if dispersions are reduced after each epoch by $\sigma_k = \sigma_k - (1 - Q)\sigma_k$. Some units may have changed so much that they do not classify any vectors at all, or their quality Q is close to zero. Such nodes are removed from the network, allowing it to grow more "healthy" nodes in other areas of the input space. Sometimes restricting the number of nodes created in each epoch leads to smaller networks, but it may also lead to slower learning without any gains. Distance $||\mathbf{X} - D(N_m)||$ does not have to be Euclidean. If two nodes show almost equal activity and one of them belongs to the wrong class it is selected as the winner to allow further adaptation.

In early implementation [1] at the classification stage we have used gradients to find the node a new vector X should be assigned to, which led to a recursive process. It appears that the activation of nodes is sufficient for correct assignment and thus slow gradient dynamics is not needed. Missing values in input are handled using linear search strategy, as described in [1]. To simplify searching for real time fast classifications, when many network nodes are created during the learning phase, a hierarchical approach may be used. Dendrograms are used to identify superclusters represented by nodes with large dispersion. Activation of a few such supernodes is checked first and only the nodes with densities contained in the winner node are searched further. Search time may always be reduced to the $\log$ of the number of network nodes.

3 Illustrative applications

Improvements in the algorithm presented in the previous section brought improvements in the classification accuracy of the FSM system, comparing to our previous results [6]. Due to the lack of space only a few results for larger datasets are quoted below.

Galaxies data were obtained from ESO-LV catalog (all details of this dataset are described in [10]). The goal is to achieve automatic classification comparable to that of human experts. This is a large data set (more than 5000 galaxies). The network was trained on 1700 ESO-LV galaxies, and tested on the remaining 3517 galaxies. Each vector has 13 features and the training set is divided in two classes: Early-type and Late-type galaxies. Backpropagation algorithm gave for this dataset 89.6% agreement with human experts classifying galaxies. Our previous results gave 98.0% accuracy on the training set and 93.2% on the test set using a network with 384 nodes. With the improvements presented in this paper 96% accuracy on the training set leads to 160 nodes and 93.0% accuracy on the test set, while 98.0% on the training set leads to 172 nodes and 93.6% accuracy on the test set, i.e. with less than half of the number of parameters used previously slightly higher accuracy is achieved.

Letter Image Recognition Data [11] contains 16-dimensional description of 26 English language letters (statistical moments and edge counts). Out of 20.000 vectors the last 5000 were selected as test cases. For 96% of accuracy 590 and for 98% accuracy on the training set 850 nodes were generated. Accuracy of classification on the test set was 90.5% and 91.1%, again a significant improvement over our previous results [6]. Other methods are not doing much better in this case, with backpropagation network achieving only 68% accuracy on the training and 67% on the test set. Interestingly best k-NN is reported with 93.2% accuracy in this case – if this result is reliable this means that our description of the data clusters is still not able to capture all the details of the data distribution.

DNA database [11] contains data on segments (windows) of 60 DNA base-pairs, coded by 3 bits for a total of 180 attributes. In the middle of such window three types of boundaries are possible (intron to exon, exon to intron or neither). There are 2000 training vectors and 1186 test vectors. The network was trained to 96% and 98% accuracy on the training set, leading to 130 and 235 nodes, over five-times less than our previous network with 1250 nodes. Accuracy on the test set was 91.9% and 92.9%, significantly better than the 91.2% accuracy achieved by an MLP network and much better than 85.4% accuracy of k-NN classifier.

The StatLog version of the **satellite image data** generated from Landsat Multi-Spectral Scanner [11] consists of four spectral values of pixels in 3x3 neighborhoods (a total of 36 attributes in 0 to 255 range) in a satellite image, and one of 6 classes assigned to the central pixel in each neighborhood. The training set has 4435 vectors, the test set 2000 vectors. For 96% accuracy on the training set 270 nodes were created and 87.6% accuracy on the test set obtained, while for 98% accuracy on the training set 303 nodes were created and 88.0% accuracy obtained. Accuracy of MLP on the training and test sets were reported as 88.8% and 86.1% respectively, Cascade Corelation gave worse test set accuracy of 83.7%, RBF gave 88.9% and 87.9%, with the best results obtained again by k-NN, 91.1% on the training and 90.6% on the test set.

All four dataset selected here are relatively large and for the two of them k-NN gives the best results, showing that the borders between the classes are quite complex and require large number of reference vectors for accurate description. FSM is an improvement over other neu-

ral classifers and comparing to the k-NN offers great reduction of the reference information as well as much greater speeds of classification.

Acknowledgments

Support by the Polish Committee for Scientific Research, grant 8T11F 00308, is gratefully acknowledged.

References

[1] W. Duch, G.H.F. Diercksen, Feature Space Mapping as a universal adaptive system, *Computer Physics Communications* 87 (1995) 341–371

[2] W. Duch, From cognitive models to neurofuzzy systems – the mind space approach. *Systems Analysis-Modelling-Simulation* 24 (1996) 53–65

[3] C. Bishop, “Neural networks for pattern recognition" (Clarendon Press, Oxford 1995)

[4] W. Duch and N. Jankowski, “Bi-radial transfer functions," in: *Proc. second conference on neural networks and their applications*, Orle Gniazdo, Poland, vol. I, pp. 131–137, 1996.

[5] W. Duch and N. Jankowski, “New Neural Transfer Functions," submitted to *J. of Applied Mathematics and Computer Science*, 1997

[6] W. Duch W, R. Adamczak “Feature Space Mapping network for classification," in: *Proc. second conference on neural networks and their applications*, Orle Gniazdo, Poland, vol. I, pp. 125–130, 1996

[7] R. Adamczak, W. Duch, N. Jankowski, “Initialization of adaptive parameters in neural networks", this volume

[8] S.J. Roberts, W. Penny, “A maximum Certainty Approach to Feedforward Neural Networks", Electronics Letters (submitted, 1997).

[9] D.J. MacKay, “A practical Bayesian framework for backpropagation networks", Neural Computations 4 (1992) 448-472

[10] O. Lahav, A. Naim, L. Sodre Jr. and M. C. Storrie-Lombardi, Neural computation as a tool for galaxy classification: methods and examples, Institute of Astronomy, Cambridge, Technical report CB3 OHA (1995)

[11] C.J. Mertz, P.M. Murphy, UCI repository of machine learning databases, http://www.ics.uci.edu/pub/machine-learning-databases.