Supervised and Unsupervised Co-training of Adaptive Activation Functions in Neural Nets

Ilaria Castelli and Edmondo Trentin

Dipartimento di Ingegneria dell'Informazione
Università di Siena, via Roma 56, Siena, Italy
{castelli,trentin}@dii.unisi.it

Abstract. In spite of the nice theoretical properties of mixtures of logistic activation functions, standard feedforward neural network with limited resources and gradient-descent optimization of the connection weights may practically fail in several, difficult learning tasks. Such tasks would be better faced by relying on a more appropriate, problem-specific basis of activation functions. The paper introduces a connectionist model which features adaptive activation functions. Each hidden unit in the network is associated with a specific pair $(f(\cdot), p(\cdot))$, where $f(\cdot)$ (the very activation) is modeled via a specialized neural network, and $p(\cdot)$ is a probabilistic measure of the likelihood of the unit itself being relevant to the computation of the output over the current input. While $f(\cdot)$ is optimized in a supervised manner (through a novel backpropagation scheme of the target outputs which do not suffer from the traditional phenomenon of "vanishing gradient" that occurs in standard backpropagation), $p(\cdot)$ is realized via a statistical parametric model learned through unsupervised estimation. The overall machine is implicitly a co-trained coupled model, where the topology chosen for learning each $f(\cdot)$ may vary on a unit-by-unit basis, resulting in a highly non-standard neural architecture.

Keywords: Co-training, partially unsupervised learning, adaptive activation function.

1 Introduction

Neural networks are one of the most common models used in the machine learning community: they have been successfully used for regression, classification and function approximation tasks. In spite of their popularity and their nice theoretical properties, practical training difficulties are often met in severe learning tasks. Indeed, the model could require a high number of hidden units in order to perform well. This would lead to an architecture with a high number of free parameters, more difficult to train, prone to overfit the training data and to get stuck into poor local minima of the criterion function. In this paper we introduce a neural model having adaptive activation functions, learned during the training procedure itself. The aim is to define a more flexible model (yet possibly simpler overall) in which the hidden units can compute arbitrary functions. Learning problems that would require a huge number of logistic activation functions are expected to turn up way simpler once their solution relies on a basis of “right”, problem-specific activation functions. Since no such basis is known in advance, the approach we propose suggests learning the functions from scratch, according to the very nature of the data at hand. Each function is specialized over the input space by means of a well-defined likelihood criterion. This can be formalized by saying that each hidden unit in the model is associated with a pair $(f(\cdot), p(\cdot))$ where $f(\cdot)$ is the unit-specific, adaptive activation function, while $p(\cdot)$ is the corresponding likelihood measure.

Neural networks are usually trained over a supervised dataset defined as $\boldsymbol{D} = \left\{\left(\boldsymbol{x^k}, \boldsymbol{y^k}\right), k = 1 \ldots N\right\}$, where $\boldsymbol{x^k} \in \mathbb{R}^d$ is a vector of observed features and $\boldsymbol{y^k} \in \mathbb{R}^n$ is a target vector. The net is taught to reproduce the target output $\boldsymbol{y^k}$ when the feature vector $\boldsymbol{x^k}$ is presented in input. Usually, a gradient descent algorithm, like backpropagation [1], is used in order to minimize the criterion function

$$C(\boldsymbol{w}) = \frac{1}{2} \sum_{k=1}^{N} \sum_{i=1}^{n} \left(y_i^k - \widetilde{y}_i^k(\boldsymbol{w})\right)^2 \tag{1}$$

where $\boldsymbol{w}$ are the connection weights, y_i^k and $\widetilde{y}_i^k(\boldsymbol{w})$ are the target and the output of the i-th output unit of the network over k-th input pattern, respectively. Each unit j in layer L_l, receives an activation given by $a_j = \sum_{u \in L_{l-1}} q_u w_{ju}$, where q_u is the output of u-th unit in the previous layer and w_{ju} is the connection weight from unit $u \in L_{l-1}$ to unit $j \in L_l$. The output of the unit is computed applying an activation function $f_j(\cdot)$ to a_j, namely $o_j = f_j(a_j)$. According to the universal approximation theorem of neural networks [2], multilayer perceptrons (MLPs) having one hidden layer made of sigmoidal units (see figure 1a) are universal approximators on a compact subset of $\mathbb{R}^d$. Although this theorem guarantees the existence of a network able to approximate any function given at least one hidden layer and sigmoid transfer functions, the network may need an arbitrary amount of weights. From a practical point of view, the number of hidden units required could be arbitrarily high, leading to difficulties during the training phase and to limited generalization capability. In the following, we outline a viable way out relying on adaptive activation functions. It is worth noting that using such functions $f(\cdot)$ realized via connectionist models will not affect the overall network's capability of being a “universal approximator”, due to theoretical results drawn from the investigation of non-sigmoid activation functions [3,4]. As we say, a probabilistic weighting strategy is used in order to train and apply $f(\cdot)$ within the overall learning machine. A unit-specific likelihood measure $p(\cdot)$ is associated with $f(\cdot)$, affecting its optimization and its contribution to the computation of the network outputs. A co-training procedure of a supervised model $f(\cdot)$ and a partially unsupervised model $p(\cdot)$ will emerge. To all practical ends, the underlying idea is that $p(\cdot)$ forces $f(\cdot)$ to focus on input patters that are likely to be drawn from a specific probability distribution (whilst standard backpropagation implicitly assumes a uniform distribution over all input patterns), simplifying the learning task by reducing it to easier sub-tasks whose support is homogeneous (meaning that it presents certain regularities).

The idea of learning activation functions while training the network has been investigated in [5] where Catmull-Rom splines are proposed. In so doing, a reduction in terms of model complexity is achieved, but the constraints imposed on the number of hidden units do not guarantee universal approximation. Conversely, if we do not impose any constraint on the number of hidden units and we use a MLP to model the activation functions, the whole model is still a universal approximator.

In order to realize a network of adaptive activation functions, a simple MLP architecture (the *outer* network) with a limited number m of hidden units is initialized and trained with backpropagation (BP) first. If the learning task is not trivial, the connectionist solution obtained this way is expected to be far sub-optimal. Once BP training has been completed (e.g., when the generalization error evaluated over a validation set does not improve any longer) the hidden units are replaced with simple MLP architectures (the *inner* networks). The architecture of the inner networks may differ on a unit-by-unit basis. This results in a non-standard, not-fully-connected topology (figure 1b). Inner networks are then trained in order to contribute solving the overall learning problem. The algorithm for our trainable-adaptive multilayer perceptron (TA-MLP) is presented in detail in the next section. In the following we assume that the outer network has only one hidden layer, while the extension of the algorithm to deeper architectures is presented in the companion paper [6]. Furthermore, since the technique does not rely on straightforward BP of the partial derivatives of the error function w.r.t. the parameters, it does not suffer from the phenomenon of "vanishing gradient" which may be met in multilayer standard networks. It turns out that estimation of $p(\cdot)$ may take a variety of forms, either entirely unsupervised or partially-unsupervised. Explicit solution of the latter estimation problem is presented in the companion paper, where an experimental demonstration of the overall algorithm is given. Preliminary conclusions are drawn in Section 3.

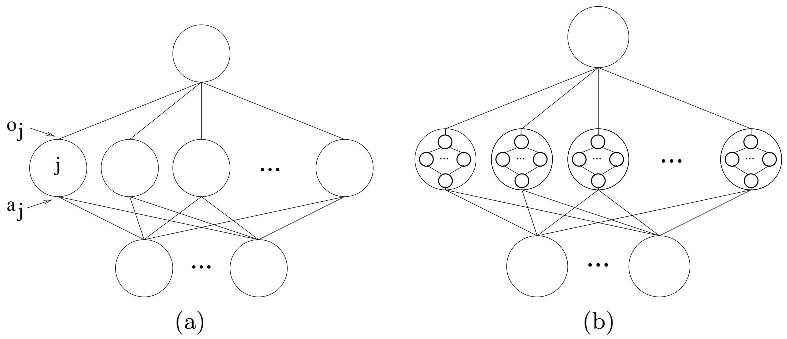

Fig. 1. (a) Classical MLP - (b) MLP with adaptive activation functions

2 The Training Algorithm

In order to train the inner networks, a training set $\boldsymbol{D_h} = \left\{ \left(x_h^k, o_h^k \right), k = 1, \ldots, N \right\}$ must be specified for each net h, with $h = 1, \ldots, m$. Note that both x_h^k and o_h^k are scalar quantities. Regular backpropagation can then be applied. Section 2.1 elaborates on how $\boldsymbol{D_h}$ is generated. Once the creation of the network-specific training sets is accomplished, the probabilistic technique presented in Section 2.2 is applied for weighting individual input patterns on a network-by-network basis. Partially-supervised maximum-likelihood estimation of the quantities involved in the probabilistic weighting scheme is outlined in Section 2.3.

2.1 Generation of Locally-Supervised Training Sets

The k-th input pattern x_h^k for the h-th inner network can be easily obtained from its activation a_h:

$$x_h^k = \sum_{u \in L_0} x_u^k w_{hu} \tag{2}$$

where x_u^k is the u-th entry of the original input vector x^k and L_0 is the input layer. More effort is required in order to define the target outputs. The supervision is available only at the output layer of the outer network, then it is necessary to define a strategy to back-propagate it. For each output unit i of the outer net, and for each pattern k, values of o_h^k are sought that satisfy the following equation:

$$\tilde{y}_i^k = f_i \left(\sum_{h=1}^{m} o_h^k w_{ih} \right). \tag{3}$$

First of all, we compute the target activations a_i of the output units of the outer network, by inversion of their activation function. In both cases of linear or sigmoidal activation function, computing the inverse is trivial. In the former case we have $y_i = f_i(a_i) = a_i$, that is $a_i = y_i$. If the activation is a sigmoid, i.e. $y_i = 1/(1 + \exp(-a_i))$, then $a_i = -\log(1 - y_i) + \log(y_i)$. In the latter case it is assumed that $y_i \in (0, 1)$. Then, the target activations a_i should be further backpropagated in order to compute the desired outputs o_h^k for each inner net. For clarity, the overall procedure is outlined in figure 2a and 2b. The former shows the trivial case where the outer network only has one output unit, while in the latter the straightforward extension to several output units is represented. Two different methods may be exploited: gradient descent and inversion of the weight matrix. Both are effective, and they generally lead to closely similar solutions. Of course, the target outputs o_h^k must be determined for each pattern in the training set, but for notational convenience we concentrate on a generic target o_h (i.e., from now on we drop the index k).

A gradient descent procedure can be exploited in almost exactly the same way as in the backpropagation algorithm. We are interested in minimizing the criterion function $C(\cdot)$ (see equation (1)) w.r.t. the output of the inner networks, o_h, with $h = 1, \ldots, m$. At this stage the weights between hidden and output

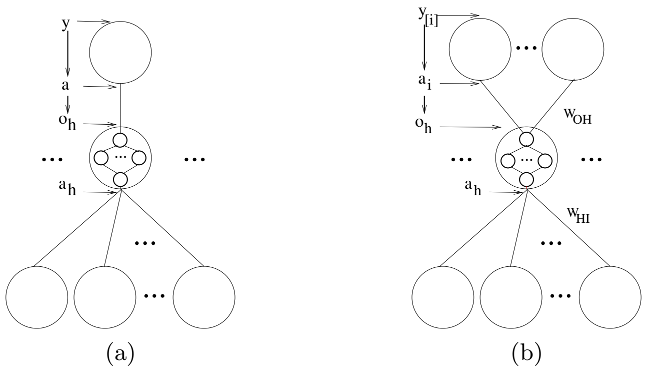

Fig. 2. Back-propagation of the target with (a) single a (b) multiple output units

layer of the outer MLP are considered to be constants (i.e., they are kept fixed at the values reached after the BP initialization of the outer network). At every iteration of BP the target output o_h of the generic h-th inner network is updated to a new value o'_h, according to the following rule:

$$o'_h = o_h + \Delta o_h = o_h - \eta \frac{\partial C}{\partial o_h} \tag{4}$$

where η is the learning rate and, for instance, if the output activation is linear

$$\begin{aligned} \frac{\partial C}{\partial o_h} &= \frac{\partial}{\partial o_h} \left\{ \frac{1}{2} \sum_{i=1}^{n} (y_i - \widetilde{y}_i)^2 \right\} \\ &= \frac{1}{2} \sum_{i=1}^{n} \frac{\partial}{\partial o_h} (y_i - \widetilde{y}_i)^2 \\ &= \sum_{i=1}^{n} (y_i - \widetilde{y}_i) \frac{\partial}{\partial o_h} \left(y_i - \sum_{j=1}^{m} w_{ij} o_j \right) \\ &= - \sum_{i=1}^{n} (y_i - \widetilde{y}_i) w_{ih} \end{aligned} \tag{5}$$

where n is the number of output units of the outer network, y_i and $\widetilde{y}_i$ are respectively the target and current output of the i-th output unit of the outer net.

A faster approach (albeit possibly less stable from a numeric standpoint) is provided by the inversion of the weight matrix of the outer network $W_{OH} \in \mathbb{R}^{n\text{X}m}$ that connects the hidden to the output layer. Upon inversion of the output activation functions, we are provided with the array of desired activation at the n output units of the outer MLP, $A_O \in \mathbb{R}^n$. If $O_H \in \mathbb{R}^m$ is the desired array of outputs of the inner networks, then $A_O = W_{OH} O_H$ and

$$O_H = W_{OH}^{-1} A_O. \tag{6}$$

Since the matrix W_{OH} usually does not have full rank, its pseudo-inverse is exploited. At this point, we have a generic, basic technique for creating the training sets for the inner networks. The next section investigates how a probabilistic weight $p(\cdot)$ is associated with each inner MLP. It will turn out that such probabilistic weights affect the very generation of target data, namely equation (5) and (6).

2.2 Probabilistic Weighting of Patterns

Since m hidden units are available, the original learning problem can be split into m smaller and easier tasks, and every inner net is specialized on one of such problems. This would be easily done having a method to evaluate the "competence" of each inner net on a given pattern. For this purpose, the posterior probability $P(h \mid \boldsymbol{x^k})$ of the h-th inner net given pattern $\boldsymbol{x^k}$, can be exploited. Explicit calculation of $P(h \mid \boldsymbol{x^k})$ relies on a neuron-specific probability density function (pdf), namely $p_h(\cdot)$, that is the probabilistic quantity $p(\cdot)$ which we associate with each of the adaptive activation functions $f(\cdot)$ as anticipated in Section 1. In order to train inner networks we define a modified, neuron-specific criterion function in which every pattern $\boldsymbol{x^k}$ is weighted by $P(h \mid \boldsymbol{x^k})$, i.e. its probability of being in the region of competence of h-th inner net:

$$C_h\left(\boldsymbol{w_h}\right) = \frac{1}{2}\sum_{k=1}^{N} P(h \mid \boldsymbol{x^k})\left(o_h^k - \widetilde{o}_h^k\right)^2 \tag{7}$$

where $\boldsymbol{w_h}$ are the connection weights of the inner net itself and $\widetilde{o}_h^k$ is its output. In so doing, the individual contribution each pattern $\boldsymbol{x^k}$ gives to the training of h-th inner MLP is proportional to the probability of the very MLP being competent over $\boldsymbol{x^k}$. Probabilistic weighting are also exploited while computing target data for inner networks. Indeed, each inner net is expected to contribute to the activation of the output units of the outer net proportionally to $P(h \mid \boldsymbol{x^k})$. The weights from the hidden to the output units of the outer MLP can then incorporate the probabilistic weight. This means that, when the outer net is fed with pattern $\boldsymbol{x^k}$, the activation of its i-th output unit is

$$a_i = \sum_{h=1}^{m} o_h^k w_{ih} P(h \mid \boldsymbol{x^k}) = \sum_{h=1}^{m} o_h^k \widetilde{w}_{ih} \tag{8}$$

where we defined the variable weight $\widetilde{w}_{ih}$ (as a function of $\boldsymbol{x^k}$) as $\widetilde{w}_{ih} = w_{ih} P(h \mid \boldsymbol{x^k})$. Computation of the target dataset for inner networks (see equation (6)) is then redefined in terms of these modified weights:

$$A_O = \widetilde{W}_{OH} O_H \quad \text{and} \quad O_H = \widetilde{W}_{OH}^{-1} A_O. \tag{9}$$

Two examples drawn from a synthetic, one-dimensional regression task (generated as discussed in the companion paper [6]) are presented in figures 3a and 3b. The dots indicate the datasets obtained applying equation (9), while dashed lines represent the probabilistic weights themselves. Finally, note that the constraint imposed through equation (8) makes it possible to recover the target output during the feedforward phase. The next section outlines the steps for the actual calculation of $P(h \mid \boldsymbol{x^k})$.

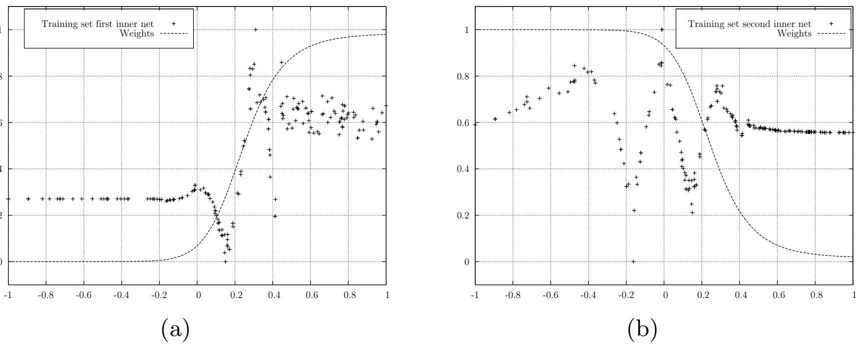

Fig. 3. Synthetic training set (dots) and the corresponding probabilistic weights (dashed line) generated for (a) first inner network and (b) second inner network

2.3 Partially Supervised Maximum-Likelihood Estimation of the Probabilistic Weights

In this section we point out how the probabilistic weights $P(h \mid \boldsymbol{x^k})$ may be computed. Let us introduce a general, fully unsupervised framework first. Later on, we will extend the approach in a semi-supervised fashion, such that the probabilistic weights can be estimated by taking benefit from the knowledge of the neuron-specific target outputs during training. According to Bayes theorem the posterior probability $P(h \mid \boldsymbol{x})$ of the h-th inner net given the pattern $\boldsymbol{x}$ is

$$P(h \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid h)P(h)}{p(\boldsymbol{x})}.$$

In practice, we associate a pdf $p_h(\boldsymbol{x}) = p(\boldsymbol{x} \mid h)$ with each adaptive neuron $h = 1, \ldots, m$. A classical Gaussian mixture model (GMM) can be used to estimate the likelihood term $p(\boldsymbol{x} \mid h)$ [1]. If we denote with $\boldsymbol{\theta}$ the parameters of the GMM, then

$$P(h \mid \boldsymbol{x}, \boldsymbol{\theta}) = \frac{p(\boldsymbol{x} \mid h, \boldsymbol{\theta})P(h)}{p(\boldsymbol{x} \mid \boldsymbol{\theta})}. \tag{10}$$

Assuming that the feature vectors $\boldsymbol{x^1}, \ldots, \boldsymbol{x^N}$ in the training sample are i.i.d. according to $p(\boldsymbol{x} \mid \boldsymbol{\theta})$, the likelihood of the parameters given the data is

$$p(\boldsymbol{x^1}, \ldots, \boldsymbol{x^N} \mid \boldsymbol{\theta}) = \prod_{k=1}^{N} p(\boldsymbol{x^k} \mid \boldsymbol{\theta})$$

where (following the usual GMM approach for a generic pattern $\boldsymbol{x}$)

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \sum_{j=1}^{c} P(\omega_j) p(\boldsymbol{x} \mid \omega_j, \boldsymbol{\theta_j}) = \sum_{j=1}^{c} P(\omega_j) \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_j}, \Sigma_j) \tag{11}$$

where c is the number of Gaussian components, $\boldsymbol{\mu_j}$, Σ_j and $P(\omega_j)$ are respectively the mean, the covariance matrix and the probability of the j-th component $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_j}, \Sigma_j)$, and $\boldsymbol{\theta_j} = (\boldsymbol{\mu_j}, \Sigma_j)$. Since we are interested in computing a probabilistic measure for each inner network h, we associate each activation function with a specific component density of the Gaussian mixture, i.e. $c = m$. In so doing, we are implicitly giving a rough probabilistic interpretation of the sigmoids realized by standard activation functions. In fact, the sigmoid (with a specific bias b and smoothness σ) is the cumulative distribution function of a corresponding logistic density function, that is close to a Gaussian distribution having mean b and variance $(\pi^2/3)\sigma^2$ (technically, the gap between multivariate Gaussian components and univariate distributions is going to be closed shortly). Standard maximum-likelihood estimation techniques can now be applied [6] in order to find $\boldsymbol{\theta_j}$, $j = 1, \ldots, m$, providing us with a complete algorithm.

So far, a viable and fully unsupervised approach has been outlined. A partially-supervised extension of the framework may benefit from the knowledge of the target outputs for the adaptive neurons during training. We perform an estimation of the joint probability of input and output data, i.e. instead of applying equation (11) we are interested in computing

$$p\left(\boldsymbol{x^k}, \boldsymbol{y^k} \mid \boldsymbol{\theta}\right) = \sum_{j=1}^{c} P(\omega_j) p(\boldsymbol{x^k}, \boldsymbol{y^k} \mid \omega_j, \boldsymbol{\theta_j}). \tag{12}$$

A GMM can still be used. When the outer network is fed with pattern $\boldsymbol{x^k}$, the latter is projected first onto a subspace defined by the weight matrix W_{HI} (i.e., the connections between the input and the hidden layer). This defines the activations of the hidden units, that forms the input of the inner nets. Each Gaussian component is then defined on a different, univariate subspace, depending on the weights $\boldsymbol{w'_h} = (w_{h1}, w_{h2}, \ldots, w_{hd})$ connecting the input layer to the h-th adaptive hidden unit (see figure 4).

This translates in defining m univariate Gaussian probability density functions, each one defined on the subspace obtained applying to the input patterns the linear transformation given by the weights W_{HI}. Equation (12) is rewritten as

$$p\left(\boldsymbol{x^k}, \boldsymbol{y^k} \mid \boldsymbol{\theta}\right) = \sum_{j=1}^{c} P(\omega_j) p'(\boldsymbol{x^k}, \boldsymbol{y^k} \mid \omega_j, \boldsymbol{\theta_j}) \tag{13}$$

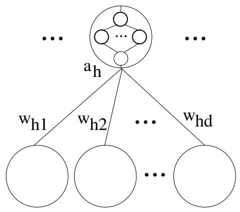

Fig. 4. Projection in hidden subspaces

where, referring to the h-th hidden unit,

$$p'(\boldsymbol{x^k}, \boldsymbol{y^k} \mid \omega_j, \boldsymbol{\theta_j}) = p(\boldsymbol{w'_h x^k}, \boldsymbol{y^k} \mid \omega_j, \boldsymbol{\theta_j}). \tag{14}$$

In this notation $\boldsymbol{w'_h}$ is meant to be a row vector and $\boldsymbol{x^k}$ is a column vector, then $\boldsymbol{w'_h x^k}$ is a scalar quantity. If we let $\boldsymbol{z^k} = \left(\boldsymbol{w'_h x^k}, \boldsymbol{y^k}\right)$ then we can rewrite equation (13) in the form

$$p\left(\boldsymbol{x^k}, \boldsymbol{y^k} \mid \boldsymbol{\theta}\right) = \sum_{j=1}^{c} P(\omega_j) p(\boldsymbol{z^k} \mid \omega_j, \boldsymbol{\theta_j}) \tag{15}$$

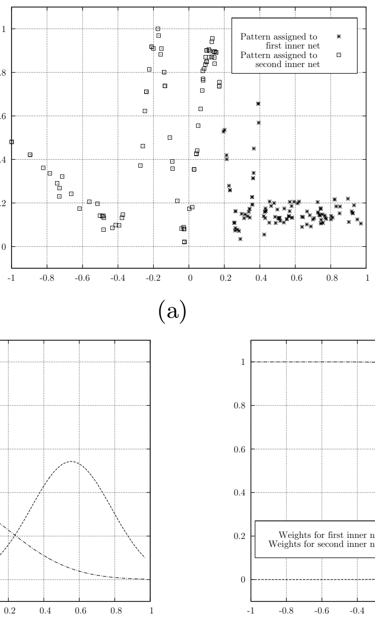

Fig. 5. (a) Pattern assigned to each Gaussian component after EM - (b) Gaussian components projected in input space - (c) probabilistic weights

An explicit maximum-likelihood solution of this estimation problem (including the probabilistic weighting of pattern we outlined in Section 2.2) based on the expectation-maximization (EM) algorithm, is developed in the companion paper [6]. During the test phase, the target $\boldsymbol{y^k}$ is not available, and then it is not possible to compute the exact value of $P(h \mid \boldsymbol{x^k}, \boldsymbol{y^k})$. In practice, we project the Gaussian components in the original input space.

A graphic, illustrative example (taken from the same regression task plotted in figure 3a and 3b) is given in figure 5. Figure 5a shows the partition of the input patterns after running the EM algorithm. Each pattern $\boldsymbol{x^k}$ is assigned to the Gaussian component h for which $P(h \mid \boldsymbol{x^k})$ is higher. Figures 5b and 5c show respectively the two Gaussian components projected in input space and the probabilistic weights (the posterior probabilities $P(h \mid \boldsymbol{x})$ for $h = 1, 2$).

3 Preliminary Conclusions

The paper introduced the idea of adaptive activation functions in order to improve the learning capability of ANNs. A general form for the gradient-based training algorithm was outlined. Each adaptive activation is associated with a probabilistic measure $p(\cdot)$. Estimation of the latter may take place according to a standard, unsupervised maximum-likelihood, or in a partially unsupervised framework which exploits the joint pdf of feature vectors and target outputs. Explicit solution of the ML estimation in the latter scenario are developed in the companion paper [6], where the extension of the algorithm to multi-layer architectures is pointed out, too, and an experimental demonstration of the proposed model is given.

References

1. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley (2001)
2. Cybenko, G.: Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals, and Systems 4, 303–314 (1989)
3. Stinchcombe, M., White, H.: Universal Approximation using Feedforward Networks with Non-Sigmoid Hidden Layer Activation Functions. In: International Joint Conference on Neural Networks, IJCNN 1989, vol. 1, pp. 613–617 (1989)
4. Chen, T., Chen, H.: Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and its Application to Dynamical Systems. IEEE Transaction on Neural Networks 4, 911–917 (1995)
5. Vecci, L., Piazza, F., Uncini, A.: Learning and Approximation Capabilities of Adaptive Spline Activation Function Neural Networks (1998)
6. Castelli, I., Trentin, E.: Semi-unsupervised Weighted Maximum-Likelihood Estimation of Joint Densities for the Co-Training of Adaptive Activation Functions. In: Schwenker, F., Trentin, E. (eds.) PSL 2011. LNCS (LNAI), vol. 7081, pp. 62–71. Springer, Heidelberg (2012)