Semi-unsupervised Weighted Maximum-Likelihood Estimation of Joint Densities for the Co-training of Adaptive Activation Functions

Ilaria Castelli and Edmondo Trentin

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

Abstract. The paper presents an explicit maximum-likelihood algorithm for the estimation of the probabilistic-weighting density functions that are associated with individual adaptive activation functions in neural networks. A partially unsupervised technique is devised which takes into account the joint distribution of input features and target outputs. Combined with the training algorithm introduced in the companion paper [2], the solution proposed herein realizes a well-defined, specific instance of the novel learning machine. The extension of the overall training method to more-than-one hidden layer architectures is pointed out, as well. A preliminary experimental demonstration is given, outlining how the algorithm works.

Keywords: Expectation maximization, partially unsupervised learning, co-training, adaptive activation function.

1 Introduction

In the companion paper an extension of the multilayer perceptron (MLP), named trainable-activations multilayer perceptron (TA-MLP), is introduced [2]. A TA-MLP is a flexible neural model having adaptive activation functions learned during the training procedure. The hidden units can compute task-specific arbitrary functions, learned according to the nature of the data. Each of them specializes over the input space according to a probabilistic criterion. The latter can be formalized by associating a pair $(f(\cdot), p(\cdot))$ with each hidden unit in the model, where $f(\cdot)$ is the adaptive activation function and $p(\cdot)$ is the corresponding likelihood measure. The quantity $f(\cdot)$ is realized by means of a MLP.

A partially-unsupervised probabilistic framework is used in order to let each hidden unit specialize on a part of the original problem. Each hidden unit h contributes to the output according to the probability $P(h \mid \boldsymbol{x})$ of that unit being "competent" on pattern $\boldsymbol{x}$. As explained in [2], a maximum-likelihood estimation of the parameters of a Gaussian mixture model (GMM) is required in order to compute $P(h \mid \boldsymbol{x})$. The GMM is expected to have as many component

F. Schwenker and E. Trentin (Eds.): PSL 2011, LNAI 7081, pp. 62–71, 2012.

densities as the number of hidden units in the TA-MLP (basically, each neuron specializes over a Gaussian distribution). The estimate of the GMM is then used within Bayes theorem in order to determine $P(h \mid \boldsymbol{x})$ [2].

As we say, each adaptive activation function relies on a standard MLP, called inner network. Flexibility of the overall learning machine can be increased further (e.g., when facing severe learning tasks) by replacing the inner net, in turn, with a TA-MLP. This may be applied recursively, as many times as necessary. In so doing, multiple levels of model expansion and adaptation are obtained. At each level, estimation of a new GMM is needed. Given an inner network h, let us call g_h its g-th hidden unit. When estimating $P(g_h \mid \boldsymbol{x})$, i.e. the posterior probability of the g_h-th inner network (within the h-th hidden unit of the outer MLP) given its input $\boldsymbol{x}$, we need to take into account the probabilistic weight introduced at the previous level(s), i.e. $P(h \mid \boldsymbol{x})$. Then, maximum likelihood estimation (after the very first level) involves a weighting factor inherited from the previous levels. In Section 2 we introduce a simple refinement of the usual expectation-maximization (EM) algorithm for the estimation of GMM parameters [1] that accounts for this peculiar "pattern weighting" mechanism. Furthermore, as observed in [2], calculations occur in the joint input-output space at training time (taking benefit from knowledge of the target outputs). On the other way around, at test time the optimal parameters are projected back onto the bare input subspace. The overall training algorithm emerging from the combination of the general scheme proposed in [2] and the weighted estimation technique introduced below can be further extended to more-than-one hidden later (outer) MLPs, as well. The complete algorithm is handed out in Section 3. A preliminary experimental demonstration of how the TA-MLP works is given in Section 4, while Section 5 draws some conclusive remarks.

2 Maximum Likelihood Estimation with Weighted Patterns

Let us define a dataset $\boldsymbol{D} = \left\{\left(\boldsymbol{x^k}, \boldsymbol{y^k}\right), k = 1, \ldots, N\right\}$, 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. In our partially-supervised framework we take benefit from the knowledge of the target outputs during training [2], and we define $\boldsymbol{z^k} = \left(\boldsymbol{w'_h x^k}, \boldsymbol{y^k}\right)$, where $\boldsymbol{w'_h} = (w_{h1}, w_{h2}, \ldots, w_{hd})$ is the vector of weights that connect the input layer to the h-th adaptive hidden unit. In this notation $\boldsymbol{w'_h}$ is meant to be a row vector and $\boldsymbol{x^k}$ is a column vector, that is, $\boldsymbol{w'_h x^k}$ is a scalar quantity. Then, let us define the dataset $\boldsymbol{D'} = \left\{\boldsymbol{z^k}, k = 1, \ldots, N\right\}$. In what follows we will work with the generic h-th inner net, i.e. all the input patterns $\boldsymbol{x^k}$, $k = 1, \ldots, N$, are projected onto the subspace defined by the weights $\boldsymbol{w'_h}$. Assuming that $\boldsymbol{z^1}, \ldots, \boldsymbol{z^N}$ are i.i.d., the likelihood of the parameters given $\boldsymbol{D'}$ is

$$p(\boldsymbol{D'} \mid \boldsymbol{\theta}) = \prod_{k=1}^{N} p(\boldsymbol{z^k} \mid \boldsymbol{\theta}) \tag{1}$$

and $p(\boldsymbol{z^k} \mid \boldsymbol{\theta})$ is expressed as a Gaussian mixture model (GMM):

$$p(\boldsymbol{z^k} \mid \boldsymbol{\theta}) = \sum_{j=1}^{c} P(\omega_j) p(\boldsymbol{z^k} \mid \omega_j, \boldsymbol{\theta_j}). \tag{2}$$

where $\boldsymbol{\theta_j} = (\boldsymbol{\mu_j}, \Sigma_j)$ are the parameters of the j-th Gaussian component (i.e. the mean and the covariance matrix) and $P(\omega_j)$ is its mixing parameter. If each pattern $\boldsymbol{z^k}$ has a weight v^k associated to it (in our case it is the probabilistic weight), equation (2) becomes

$$p(\boldsymbol{z^k} \mid \boldsymbol{\theta}) = v^k \sum_{j=1}^{c} P(\omega_j) p(\boldsymbol{z^k} \mid \omega_j, \boldsymbol{\theta_j}). \tag{3}$$

We can write the log-likelihood as

$$\log p(\boldsymbol{D'} \mid \boldsymbol{\theta}) = \sum_{k=1}^{N} \log \left\{ p(\boldsymbol{z^k} \mid \boldsymbol{\theta}) \right\}. \tag{4}$$

In order to optimize the log-likelihood w.r.t. its parameters $\boldsymbol{\theta}$ the estimation of the optimal parameters for each component of the mixture is needed. We assume that $\boldsymbol{\theta_i}$ is functionally independent from $\boldsymbol{\theta_j}$ when $i \neq j$. We assume also identifiability of the components of the mixture, i.e. $\boldsymbol{\theta} \neq \tilde{\boldsymbol{\theta}} \Rightarrow \exists \boldsymbol{z} \in \boldsymbol{D'} : p(\boldsymbol{z} \mid \boldsymbol{\theta}) \neq p(\boldsymbol{z} \mid \tilde{\boldsymbol{\theta}})$. Then, we compute the gradient of equation (4) w.r.t. the parameters of the generic i-th component, $\boldsymbol{\theta_i}$, and set it equal to zero:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta_i}} \log p(\boldsymbol{D'} \mid \boldsymbol{\theta}) &= \sum_{k=1}^{N} \frac{1}{p(\boldsymbol{z^k} \mid \boldsymbol{\theta})} \nabla_{\boldsymbol{\theta_i}} \left\{ v^k \sum_{j=1}^{c} P(\omega_j) p(\boldsymbol{z^k} \mid \omega_j, \boldsymbol{\theta_j}) \right\} \\
&= \sum_{k=1}^{N} \frac{1}{p(\boldsymbol{z^k} \mid \boldsymbol{\theta})} \nabla_{\theta_i} \left\{ v^k P(\omega_i) p(\boldsymbol{z^k} \mid \omega_i, \boldsymbol{\theta_i}) \right\} \\
&= \sum_{k=1}^{N} \frac{v^k P(\omega_i)}{p(\boldsymbol{z^k} \mid \boldsymbol{\theta})} \nabla_{\boldsymbol{\theta_i}} \left\{ p(\boldsymbol{z^k} \mid \omega_i, \boldsymbol{\theta_i}) \right\} \\
&= \sum_{k=1}^{N} \frac{v^k P(\omega_i \mid \boldsymbol{z^k}, \boldsymbol{\theta})}{p(\boldsymbol{z^k} \mid \omega_i, \boldsymbol{\theta_i})} \nabla_{\boldsymbol{\theta_i}} \left\{ p(\boldsymbol{z^k} \mid \omega_i, \boldsymbol{\theta_i}) \right\} \\
&= \sum_{k=1}^{N} v^k P(\omega_i \mid \boldsymbol{z^k}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta_i}} \log \left\{ p(\boldsymbol{z^k} \mid \omega_i, \boldsymbol{\theta_i}) \right\} = \mathbf{0}
\end{aligned} \tag{5}$$

where $\mathbf{0}$ is a vector whose entries are all equal to zero. Compared to the classical unweighted estimation, we have the additional weighting factors v^k. Since each component of the mixture is Gaussian, $\boldsymbol{\theta_j} = (\boldsymbol{\mu_j}, \Sigma_j)$, we have:

$$\log p(\boldsymbol{z^k} \mid \omega_j, \boldsymbol{\theta_j}) = -\log \left\{ (2\pi)^{d/2} \mid \Sigma_j \mid^{1/2} \right\} - \frac{1}{2} (\boldsymbol{z^k} - \boldsymbol{\mu_j})^{\intercal} \Sigma_j^{-1} (\boldsymbol{z^k} - \boldsymbol{\mu_j}). \tag{6}$$

Taking the gradient of (6) w.r.t. $\boldsymbol{\mu_j}$ we obtain

$$\nabla_{\boldsymbol{\mu_j}} \log p(\boldsymbol{z^k} \mid \omega_j, \boldsymbol{\theta_j}) = \Sigma_j^{-1}(\boldsymbol{z^k} - \boldsymbol{\mu_j}) \tag{7}$$

and equation (5) can be rewritten as

$$\sum_{k=1}^{N} P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta}) v^k \Sigma_j^{-1}(\boldsymbol{z^k} - \boldsymbol{\mu_j}) = \mathbf{0} \tag{8}$$

that is a set of $d+d^2$ equations that represent necessary conditions to be satisfied by the maximum-likelihood estimator. It follows that

$$\sum_{k=1}^{N} P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta}) v^k \Sigma_j^{-1} \boldsymbol{z^k} = \sum_{k=1}^{N} P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta}) v^k \Sigma_j^{-1} \boldsymbol{\mu_j} \tag{9}$$

and then

$$\boldsymbol{\mu_j} = \frac{\sum_{k=1}^{N} P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta}) v^k \boldsymbol{z^k}}{\sum_{k=1}^{N} P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta}) v^k} \tag{10}$$

In a similar manner, the gradient of (6) w.r.t. Σ_j can be calculated, yielding:

$$\Sigma_j = \frac{\sum_{k=1}^{N} P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta}) v^k (\boldsymbol{z^k} - \boldsymbol{\mu_j})(\boldsymbol{z^k} - \boldsymbol{\mu_j})^\intercal}{\sum_{k=1}^{N} P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta}) v^k}. \tag{11}$$

Finally, the mixing coefficients can be calculated taking the gradient of the log-likelihood w.r.t. $P(w_j)$ while imposing the constraint $\sum_{i=1}^{c} P(w_i) = 1$. This can be done using a Lagrange multiplier and maximizing the quantity

$$\begin{aligned} L &= \sum_{k=1}^{N} \log \left\{ p(\boldsymbol{z^k} \mid \boldsymbol{\theta}) \right\} + \lambda \left(\sum_{i=1}^{c} P(\omega_i) - 1 \right) \\ &= \sum_{k=1}^{N} \log \left\{ v^k \sum_{i=1}^{c} P(\omega_i) p(\boldsymbol{z^k} \mid \omega_i, \boldsymbol{\theta_i}) \right\} + \lambda \left(\sum_{i=1}^{c} P(\omega_i) - 1 \right) \end{aligned} \tag{12}$$

We then calculate the partial derivative of equation (12) w.r.t. the generic mixing parameter $P(w_j)$ and set it equal to zero:

$$\begin{aligned} \frac{\partial L}{\partial P(\omega_j)} &= \frac{\partial}{\partial P(\omega_j)} \sum_{k=1}^{N} \log \left\{ p(\boldsymbol{z^k} \mid \boldsymbol{\theta}) \right\} + \lambda \left(\sum_{i=1}^{c} P(\omega_i) - 1 \right) \\ &= \sum_{k=1}^{N} \frac{1}{p(\boldsymbol{z^k} \mid \boldsymbol{\theta})} \frac{\partial}{\partial P(\omega_j)} \left\{ v^k \sum_{i=1}^{c} P(\omega_i) p(\boldsymbol{z^k} \mid \omega_i, \boldsymbol{\theta_i}) \right\} + \lambda \\ &= \sum_{k=1}^{N} v^k \frac{p(\boldsymbol{z^k} \mid \omega_j, \boldsymbol{\theta_j})}{p(\boldsymbol{z^k} \mid \boldsymbol{\theta})} + \lambda \\ &= \sum_{k=1}^{N} v^k \frac{P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta})}{P(\omega_j)} + \lambda = 0 \end{aligned} \tag{13}$$

where we used the equality

$$\frac{p(\boldsymbol{z^k} \mid \omega_j, \boldsymbol{\theta_j})}{p(\boldsymbol{z^k} \mid \boldsymbol{\theta})} = \frac{P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta})}{P(\omega_j)} \tag{14}$$

given by Bayes theorem. Multiplying both sides of equation (13) by $P(w_j)$ and summing over j making use of the constraint $\sum_{i=1}^{c} P(w_i) = 1$, we obtain

$$\begin{aligned}
\lambda &= -\sum_{j=1}^{c} P(\omega_j) \sum_{k=1}^{N} v^k \frac{P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta})}{P(\omega_j)} \\
&= -\sum_{k=1}^{N} v^k \sum_{j=1}^{c} \frac{P(\omega_j) P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta})}{P(\omega_j)} \\
&= -\sum_{k=1}^{N} v^k \sum_{j=1}^{c} P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta}) \\
&= -\sum_{k=1}^{N} v^k
\end{aligned} \tag{15}$$

Substitution of (15) into (13) gives

$$\frac{\sum_{k=1}^{N} v^k P(\omega_j \mid \boldsymbol{z^k}, \boldsymbol{\theta})}{P(\omega_j)} = \sum_{k=1}^{N} v^k. \tag{16}$$

Finally, solving for $P(\omega_j)$:

$$P(\omega_j) = \frac{\sum_{k=1}^{N} v^k P(\omega_j \mid \boldsymbol{x^k}, \boldsymbol{\theta})}{\sum_{k=1}^{N} v^k}. \tag{17}$$

Note that the derived maximum-likelihood estimation does not have a closed-form analytical solution. Then, following the classical EM approach [1] an iterative algorithm based on a gradient ascent procedure is exploited. Parameters $\boldsymbol{\theta}$ (i.e. $\boldsymbol{\mu_j}$, Σ_j and $P(\omega_j)$, for each Gaussian component j) are initialized arbitrarily (to this end, the *k-means* algorithm [1] is applied in this paper). Then, at each iteration, the E-step consists in computing $P(\omega_j|\boldsymbol{z^k}, \boldsymbol{\theta})$ according to the current value of $\boldsymbol{\theta}$ and for each component j. In the M-step parameters are re-estimated using such values of $P(\omega_j|\boldsymbol{z^k}, \boldsymbol{\theta})$ according to equations (10), (11) and (17) [1,3].

3 Extension to Multiple Hidden Layers

In this section we extend the training algorithm presented in the companion paper [2] to more-than-one hidden layer architectures. Algorithm 1 hands out the pseudo-code. The (outer) MLP is assumed to have L layers ($L-1$ hidden layers

with a layer-specific number of hidden units, and an output layer). The extension is rather straightforward. The basic idea requires an initialization via standard backpropagation (BP) as in [2]. The activation functions of the topmost hidden layer are basically trained as in the single-hidden-layer setup (called routine `Train` in the pseudo-code), via computation of the inverse of the output activation functions and backpropagation of the target outputs (referred to as routine `BackpropagateTargets` in Algorithm 1), as explained in [2]. Estimation of the corresponding GMM takes place according to the calculations given in Section 2. The weighted, joint pdf estimation of the GMM parameters is referred to as routine `EstimateGMM` in the pseudo-code. So far, the only novelty is that the input dataset (for estimation of the GMM and the training of the inner networks) is no longer obtained from the original input patters, but from the outputs yielded by the previous hidden layer (computed via routine `FeedForward`). Estimation of GMMs and training of inner MLPs within the lower hidden layers (down to the bottom-most) occur in an iterative fashion, following (i) a forward propagation of the original inputs up to the required layer, and (ii) a progressive backward propagation step of target outputs.

In Algorithm 1 actual inputs to l-th layer, for the k-th pattern are referred to as $\boldsymbol{x}^{\boldsymbol{k}}(l)$, while $\widehat{\boldsymbol{x}}^{\boldsymbol{k}}(l)$ indicate the desired inputs (i.e. obtained through inversion of the activation functions for the L-th layer, and through MLP-inversion for the hidden layers, see below). The target outputs backpropagated at l-th layer is referred to as $\boldsymbol{o}^{\boldsymbol{k}}(l)$ [2]. $\boldsymbol{D_h}(l) = \left\{\left(x_h^k(l), o_h^k(l)\right), k = 1, \ldots, N\right\}$ denotes the training set for the h-th inner net in l-th layer, where $x_h^k(l)$ and $o_h^k(l)$ are the h-th entry of vector $\boldsymbol{x}^{\boldsymbol{k}}(l)$ and $\boldsymbol{o}^{\boldsymbol{k}}(l)$, respectively. Finally, $\boldsymbol{D_{GMM}}(l) = \left\{\left(\boldsymbol{x}^{\boldsymbol{k}}(l), \boldsymbol{o}^{\boldsymbol{k}}(l+1)\right), k = 1, \ldots, N\right\}$ is the dataset used for GMM estimation at l-th layer.

The only catch is the definition of suitable target outputs at a generic layer, starting from the outputs of the upper layer. This is accomplished by means of the so-called MLP *inversion* method [5], according to the calculations outlined in [4]. The method is conceptually simple: starting from a neural network which realizes a transformation $\mathbf{y} = \phi(\mathbf{x}, \mathbf{w})$ for a given (pre-trained) set of weights $\mathbf{w}$, the MLP inversion principle prescribes the transformation of the input patterns $\mathbf{x}$ into new patters $\mathbf{x}'$ which better fit the target criterion function $C(\cdot)$. The update rule for creating $\mathbf{x}'$ follows the usual gradient descent approach, aimed at minimizing $C(\cdot)$ w.r.t. $\mathbf{x}$ (while the weights $\mathbf{w}$ are clamped to their original values). In summary, we let

$$\mathbf{x}' = \mathbf{x} - \eta \nabla_{\mathbf{x}} C(\cdot) \tag{18}$$

whose explicit calculation is accomplished in a way similar to standard BP, $\eta \in \mathbb{R}^+$ being a learning rate. The routine realizing such an inversion scheme over a generic MLP is referred to as `Invert`() in the pseudo-code, and it has to be applied to all inner networks in the model.

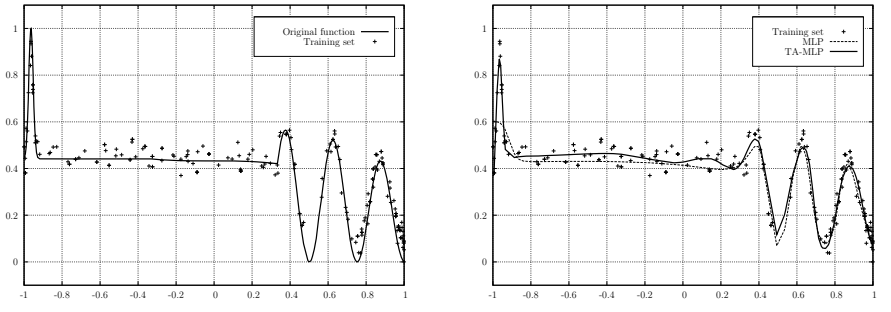
Algorithm 1. Training multilayer networks with adaptive activation functions

Input: $\boldsymbol{D} = \left\{\left(\boldsymbol{x}^{\boldsymbol{k}}, \boldsymbol{y}^{\boldsymbol{k}}\right), k = 1 \dots N\right\}$
$L \leftarrow$ number of layers (except input layer)
for $l = 1$ **to** $L - 1$ **do**
  $m(l) \leftarrow$ number of units in l-th layer
end for
for $k = 1$ **to** N **do**
  $\widehat{\boldsymbol{x}}^{\boldsymbol{k}}(L) \leftarrow$ inverse of activation functions of L-th layer over $\boldsymbol{y}^{\boldsymbol{k}}$
  $\boldsymbol{o}^{\boldsymbol{k}}(L) = \boldsymbol{y}^{\boldsymbol{k}}$
end for
for $l = L - 1$ **down to** 1 **do**
  $\mathbf{D_{GMM}(l)} \leftarrow \emptyset$
  for $h = 1$ **to** $m(l)$ **do**
    $\boldsymbol{D_h}(l) \leftarrow \emptyset$
  end for
  for $k = 1$ **to** N **do**
    $\boldsymbol{x}^{\boldsymbol{k}}(l) \leftarrow$ `FeedForward`$(\boldsymbol{x}^{\boldsymbol{k}})$ up to l-th layer
    $\boldsymbol{D_{GMM}}(l) \leftarrow \boldsymbol{D_{GMM}}(l) \bigcup \{(\boldsymbol{x}^{\boldsymbol{k}}(l), \boldsymbol{o}^{\boldsymbol{k}}(l+1))\}$
  end for
  `EstimateGMM` over $\boldsymbol{D_{GMM}}(l)$
  for $k = 1$ **to** N **do**
    $\boldsymbol{o}^{\boldsymbol{k}}(l) \leftarrow$ `BackpropagateTargets`$(\widehat{\boldsymbol{x}}^{\boldsymbol{k}}(l+1))$
    for $h = 1$ **to** $m(l)$ **do**
      $\boldsymbol{D_h}(l) \leftarrow \boldsymbol{D_h}(l) \bigcup \{(x_h^k(l), o_h^k(l))\}$
    end for
  end for
  for $h = 1$ **to** $m(l)$ **do**
    `Train` h-th inner net on $\boldsymbol{D_h}(l)$
  end for
  for $k = 1$ **to** N **do**
    $\widehat{\boldsymbol{x}}^{\boldsymbol{k}}(l) \leftarrow$ `Invert`(inner networks)
  end for
end for

4 Demonstration

In this section we present a preliminary evaluation of the proposed model on a synthetic regression task. We generated piecewise functions defined over three intervals. In each interval the function is a mixture of basic functions, namely: a sinusoid multiplied by a quadratic function, a Gaussian mixture multiplied by a linear function and a cubic function. The order of the intervals is randomly generated for each piecewise function. Finally, random Gaussian noise was added to the function. The standard deviation of the noise is a random value varying between 0.01 and 0.05. The input and output range were normalized in $[-1, 1]$ and $[0, 1]$, respectively. The cardinality of the training, validation and test sets was 200, 100 and 200 patterns, respectively.

The model was evaluated making use of two common criteria, namely the mean squared error (MSE), that is $MSE = \frac{1}{N}\sum_{k=1}^{N}\left(y^k - \widetilde{y}^k\right)^2$, and the integrated squared error (ISE), defined as $ISE = \int_{\mathcal{I}}\left(f(x) - \widetilde{f}(x)\right)^2 dx$, where $\mathcal{I}$ is the interval where the x variable is defined. The integral was evaluated using Simpson method. To this end, the range of the input variable was divided into 1000 intervals. Figure 1a shows the original function (solid line) and the training set obtained by adding Gaussian noise (later used to train the outer network). We set $m = 2$ and then replaced each of the hidden units with the corresponding inner MLPs. The architecture of the latter ones was determined through a cross-validation procedure.



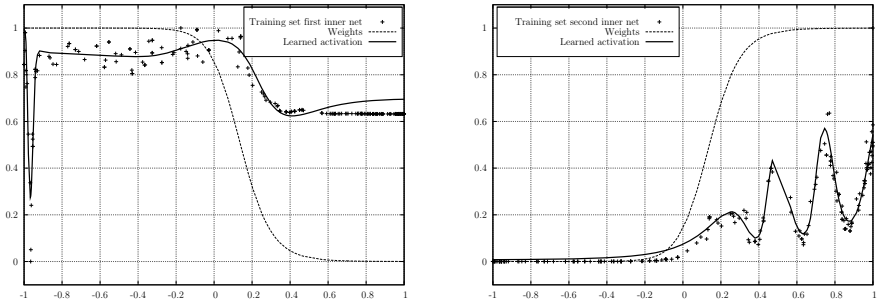
(a) (b)

Fig. 1. (a) Synthetic function and training set - (b) Comparison between MLP and TA-MLP

Figures 2a and 2b show the training set, the probabilistic weights and the activation function learned by the two inner networks, respectively.

Table 1 shows the comparison between standard MLPs and TA-MLPs. For each network we indicate the total number of free parameters of the model, the number of hidden units (in case of TA-MLPs, the number of units of inner networks is also indicated in brackets), the MSE on both training and test sets, and the ISE. The best five results for both models are reported (in the order) in the table. The first row of the table shows that for a fixed number of free parameters (for both models) the TA-MLP achieves slightly better result than the MLP in terms of ISE and MSE, on both training and test sets. This confirms the algorithm is effective. Moreover, increased flexibility of the model over the training sample does not affect its generalization capabilities (i.e. proper, non-overfitting activation functions are actually learned). The subsequent rows show how the performance of TA-MLPs remains stable when the number of free parameters decreases. In traditional MLPs, on the other end, an increased number of free parameters does not entail a comparable improvement in terms of performance.

Figure 1b shows the approximations obtained with the best TA-MLP (solid line) and with the best standard MLP (dashed line), respectively, along with



(a) (b)

Fig. 2. Activation functions learned by (a) first inner network and (b) second inner network (solid lines), together with their training set (points) and their probabilistic weights (dashed lines)

Table 1. Comparison between MLPs and TA-MLPs

MLP					**TA-MLP**				
#Par	#Hid	MSEtrain	MSEtest	ISE	#Par	#Hid	MSEtrain	MSEtest	ISE
40	13	0.033	0.037	0.00137	40	2(5-6)	0.032	0.035	9.25e-4
49	16	0.046	0.052	0.00390	37	2(4-6)	0.033	0.035	9.42e-4
46	15	0.047	0.052	0.00393	34	2(4-5)	0.034	0.038	9.55e-4
34	11	0.047	0.052	0.00392	43	2(6-6)	0.033	0.035	9.67e-4
37	12	0.047	0.052	0.00399	37	2(5-5)	0.035	0.038	9.97e-4

the corresponding training sets. It is seen that modeling the first peak exhibited by the training data turned up infeasible via standard MLP, while using the TA-MLP the very peak turns out to be modeled suitably (via the inner network which focused on the corresponding, specific region). The activation functions learned by the two inner networks (problem-specific, and quite different from regular sigmoids) are shown in solid lines in figures 2a and 2b.

5 Conclusion

The paper developed an explicit, weighted maximum-likelihood solution to the problem of estimating the density functions (defined over the joint input/output space) associated wit the neurons of neural nets having adaptive activation functions. Combining the result with the generic training scheme introduced in the companion paper [2], a complete algorithm for this family of connectionist models emerges. The algorithm was extended to multi-layer architectures in a natural way. A preliminary experimental demonstration (over a synthetic regression task) proved the resulting approach being effective. It is seen that in the 1-hidden-layer scenario the overall machine can be described as a particular case of the traditional mixture of neural experts [6], having a novel training/gating

policy. In the multi-layer setup this dual interpretation does not hold any longer, and we are faced with a novel, non-standard neural network (non fully-connected, and possibly having different depths along separate branches of its graphical structure), where the probabilistic measures associated with each adaptive neuron are defined over non-linearly transformed images of the original data. Efforts are currently focused toward (i) the definition of a robust, automatic technique for the selection of suitable, neuron-specific topologies for the inner MLPs (relying on the evaluation of a cross-validated log-likelihood criterion), as well as on (ii) a thorough experimental comparative analysis of the behavior of the proposed machine.

References

1. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley (2001)
2. Castelli, I., Trentin, E.: Supervised and Unsupervised Co-Training of Adaptive Activation Functions in Neural Nets. In: Schwenker, F., Trentin, E. (eds.) PSL 2011. LNCS (LNAI), vol. 7081, pp. 52–61. Springer, Heidelberg (2012)
3. Bishop, C.M.: Pattern Recognition and Machine Learning. Information Science and Statistics. Springer, Heidelberg (2007)
4. Trentin, E., Gori, M.: Inversion-based nonlinear adaptation of noisy acoustic parameters for a neural/HMM speech recognizer. Neurocomputing 70(1-3), 398–408 (2006)
5. Linden, A., Kindermann, J.: Inversion of multilayer nets. In: Proc. of IJCNN 1989, Washington DC, pp. 425–430 (1989)
6. Hertz, J.A., Palmer, R.G., Krogh, A.: Introduction to the Theory of Neural Computation. Santa Fe Institute Studies in the Sciences of Complexity. Westview Press (1991)