Pattern Recognition Letters 37 (2014) 178–191

Contents lists available at SciVerse ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

# Combination of supervised and unsupervised learning for training the activation functions of neural networks

Ilaria Castelli, Edmondo Trentin *

*Dipartimento di Ingegneria dell'Informazione, Università di Siena, Via Roma 56, 53100 Siena, Italy*

ARTICLE INFO

*Article history:*
Available online 26 June 2013

*Keywords:*
Adaptive activation function
Co-training
Partially supervised learning

ABSTRACT

Standard feedforward neural networks benefit from the nice theoretical properties of mixtures of sigmoid activation functions, but they may fail in several practical learning tasks. These tasks would be better faced by relying on a more appropriate, problem-specific basis of activation functions. The paper presents a connectionist model which exploits adaptive activation functions. Each hidden unit in the network is associated with a specific pair $(f(\cdot), p(\cdot))$, where $f(\cdot)$ is the activation function and $p(\cdot)$ is the likelihood of the unit being relevant to the computation of the network output over the current input. The function $f(\cdot)$ is optimized in a supervised manner, while $p(\cdot)$ is realized via a statistical parametric model learned through unsupervised (or, partially supervised) estimation. Since $f(\cdot)$ and $p(\cdot)$ influence each other's learning process, the overall machine is implicitly a co-trained coupled model and, in turn, a flexible, non-standard neural architecture. Feasibility of the approach is corroborated by empirical evidence yielded by computer simulations involving regression and classification tasks.

## 1. Introduction

Artificial neural networks (ANN) have long been a popular approach to a number of machine learning problems, especially classification and regression tasks. In particular, the popular multilayer perceptron (MLP) with a single hidden layer of sigmoid units and linear output(s) is a "universal" approximator (Cybenko, 1989). Formally, given any continuous and limited function $\varphi(\cdot)$ defined on a compact subset of $\Re^d$, such an MLP exists which realizes a function $\tilde{\varphi}(\cdot)$ whose Chebyshev distance from $\varphi(\cdot)$ is smaller than $\epsilon \in \Re^+$, for any arbitrarily small value of $\epsilon$.

Unfortunately, this theoretical property is of little practical relevance. In fact, it does not specify the right architecture (and, parameters) of the proper MLP (also, it does not tell anything on the actual convergence of the learning process). Moreover, as shown in Bengio (2009), it might as well be that the number of sigmoid units required in order to guarantee the approximation of $\varphi(\cdot)$ could be as huge as to cause severe problems during training (e.g., computational and/or numerical problems, entrapment within local minima), as well as limited generalization capability (due to the complexity of the resulting learning machine), while just a few activation functions of suitable form could fit $\varphi(\cdot)$.

These are some of the reasons why several scientists have began developing the so-called "deep architectures" (Bengio, 2009), i.e. ANNs having a large number of hidden layers which basically realize the progressive composition of nonlinear transformations over the original input. The alternative idea pursued herein is that such difficulties might be tackled by means of smaller architectures, provided that the right, problem-specific activation functions are chosen. Furthermore, these functions could even be neuron-specific, and they are better autonomously learned from examples by the machine. To this end, in this paper we discuss and expand the neural model that we introduced in Castelli and Trentin (2012) and Castelli and Trentin (2012), called trainable-activations multilayer perceptron (TA-MLP). The TA-MLP is a simple ANN whose activation functions are adaptive and realize (quasi) arbitrary functions (as long as these functions are continuous and limited over a closed domain). Eventually, many-layer versions of the TA-MLP turn out to be deep architectures themselves.

Although the literature on ANNs has mostly focused on learning the connection weights, several techniques have long been proposed for learning parameters that characterize the sigmoidal activation functions. Major instances are represented by: the usual, neuron-specific learning of the bias (Hertz et al., 1991); learning the slope (Yamada and Yabuta, 1992) or the smoothness (Hu and Shao, 1992) of sigmoids (shared among all the neurons in the ANN); learning the amplitude, i.e. the range, of (neuron-specific, or shared) sigmoids (Trentin, 2001). In Chen and Chang (1996) a custom sigmoid with neuron-specific adaptive

* Corresponding author. Tel.: +39 0577 234636; fax: +39 0577 233602.
*E-mail address:* trentin@dii.unisi.it (E. Trentin).
*URL:* http://www.dii.unisi.it/~trentin (E. Trentin).

0167-8655/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.patrec.2013.06.013

parameters controlling its "shape" is introduced in the output layer of the ANN. Finally, the idea of learning the activation functions relying on Catmull-Rom splines is exploited in Vecci et al. (1998). That approach leads to a reduction in terms of model complexity, at the expense of a reduced approximation capability (due to the constraints imposed on the number of hidden units).

The alternative solution proposed herein goes as follows. First of all, training a TA-MLP involves applying backpropagation (BP) (Duda et al., 2001) to a standard MLP (the *outer* network), having a (possibly, small) number $m$ of hidden sigmoids. BP training of the outer ANN is iterated until the generalization error (as evaluated on a validation set) does not improve any longer. The sigmoids associated with the hidden units of the outer ANN are then replaced by individual MLP architectures (the *inner* networks). The inner ANNs are then trained to realize the corresponding unit-specific activation functions, aiming at contributing to the solution of the learning problem at hand. The overall model results in a non-standard, not fully connected topology. The architectures of the inner networks within the TA-MLP, determined via any model selection strategy (Claeskens and Hjort, 2008), may as well differ from each other. It is worth noticing that using activation functions realized via connectionist models will not affect the overall capability of the ANN of being a "universal approximator", due to theoretical results drawn from the investigation of non-sigmoid activation functions (Stinchcombe and White, 1989; Chen and Chen, 1995).

Each activation function (i.e., each inner ANN) may be specialized over the input space by means of a well-defined probabilistic criterion. The latter shall be a reasonable, quantitative measure of the impact any specific unit in the model is expected to have on the whole ANN computation when presented with any given input vector. Formally, the generic $h$th hidden unit in the ANN is associated with a pair $(f_h(\cdot), p_h(\cdot))$ where $f_h(\cdot)$ is the unit-specific, adaptive activation function, while $p_h(\cdot)$ is the corresponding probabilistic measure. The latter is the likelihood of the unit being relevant to the TA-MLP output given its current input (or, any meaningful likelihood-related measure of the kind). This probabilistic measure affects the training algorithm, since the weights of the $h$th inner ANN undergo modifications whose magnitude is related to the value of $p_h(\cdot)$ over the current input. In a similar manner, it is involved also in the feed-forward (i.e., test) phase of a trained TA-MLP, since it determines (on a pattern by pattern basis) the contribution each inner net provides to the overall TA-MLP output. It turns out that the estimation of $p_h(\cdot)$ may take a variety of forms, either entirely unsupervised or partially supervised.

The unit-specific likelihood measure $p_h(\cdot)$ associated with $f_h(\cdot)$ affects its optimization and its contribution to the computation of the ANN outputs. A co-training procedure of a supervised model $f_h(\cdot)$ and a (partially) unsupervised model $p_h(\cdot)$ this way emerges. To all practical ends, the underlying idea is that $p_h(\cdot)$ forces $f_h(\cdot)$ to focus on input patterns that are likely to be drawn from a specific probability distribution (while standard, supervised-only 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 (insofar that it presents certain regularities, s.t. it is well described via a probability density function having known form).

Since the technique does not rely on straightforward backpropagation of the partial derivatives of the error function w.r.t. the parameters (as in BP), it does not suffer from the phenomenon of "vanishing gradient" which may be met in standard multilayer networks (Bengio et al., 1994).

The training algorithm is presented in detail in the next section, including a variety of likelihood-related probabilistic measures. At first we assume that the outer ANN has only one hidden layer, while the extension of the algorithm to multi-layer architectures is presented in Section 3. A straightforward, alternative application of the training algorithms in conjunction with the simultaneous learning of the weights of the outer ANN is presented in Section 4. Empirical evidence is presented eventually (in Section 5) that the TA-MLP may improve over the standard MLP (and, over statistical techniques) in classification and regression tasks, possibly requiring a smaller number of free parameters. Section 6 draws the conclusive remarks.

## 2. Training algorithm for 1-hidden layer TA-MLPs

In the following, a supervised training set is assumed, having the usual form $\boldsymbol{D} = \{(\boldsymbol{x}^k, \hat{\boldsymbol{y}}^k), k = 1 \ldots N\}$ where $\boldsymbol{x}^k \in \Re^d$ is a feature vector and $\hat{\boldsymbol{y}}^k \in \Re^n$ is the corresponding target output. A gradient-descent algorithm (such as BP) is used in order to minimize the criterion function

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

where $\boldsymbol{w}$ denotes the (set of all the) connection weights in the TA-MLP, while $\hat{y}_i^k$ and $y_i^k$ are the $i$th component of the target and the TA-MLP output over the $k$-th input pattern, respectively. If the TA-MLP has $\ell$ layers, namely $L_0$ (the input layer, which is not counted), $L_1, \ldots, L_{\ell-1}$ (the hidden layers), and $L_\ell$ (the output layer), we will write $i \in L_l$ to denote the $i$-th unit in layer $L_l$.

The generic $h$th unit in layer $L_l$ receives an input (namely, its activation $a_h$) given by $a_h = \sum_{j \in L_{l-1}} o_j w_{hj}$, where $o_j$ is the output of unit $j \in L_{l-1}$ and $w_{hj}$ is the connection weight from unit $j$ to unit $h$. The output $o_h$ yielded by the $h$th unit is computed applying an activation function $f_h(\cdot)$ to $a_h$, namely $o_h = f_h(a_h)$.

In order to train the (generic) $h$th inner network, $h = 1, \ldots, m$, a training set $\boldsymbol{D}_h = \{(x_h^k, \hat{o}_h^k), k = 1, \ldots, N\}$ is needed (note that $x_h^k$ and $\hat{o}_h^k$ are both scalar quantities). Section 2.1 elaborates on how $\boldsymbol{D}_h$ can be generated. The probabilistic techniques presented in section 2.2 are then applied for weighting individual input patterns for each of the inner networks. This weighting is used in training the inner ANNs, and in testing the resulting TA-MLP (contribution from each hidden neuron to the overall TA-MLP output is weighted accordingly). Maximum-likelihood methods for the estimation of the quantities involved in the probabilistic weighting scheme are outlined in Section 2.3.

### 2.1. Generation of locally-supervised training sets

In order to define $\boldsymbol{D}_h = \{(x_h^k, \hat{o}_h^k), k = 1, \ldots, N\}$, the $k$th input $x_h^k$ for the $h$th inner network can be generated by relying on the activation $a_h$ for unit $h$ over the current pattern $\boldsymbol{x}^k$, namely $x_h^k = a_h$. More effort is required in order to define the corresponding target output $\hat{o}_h^k$. The supervision is available only at the output layer of the outer network, then it is necessary to define a strategy to back-propagate the target. For each output unit $i$ of the outer net, and for each pattern $k$, values of $\hat{o}_h^k$ are sought that satisfy the following equation:

$$\hat{y}_i^k = f_i\left(\sum_{h=1}^{m} \hat{o}_h^k w_{ih}\right). \tag{2}$$

First of all, we compute the target activations $\hat{a}_i$ of the output units of the outer network, by inversion of their activation functions. In both cases of linear or sigmoidal activation function, calculating the corresponding inverse is straightforward. In the former case we aim at target activations such that $\hat{y}_i = f_i(\hat{a}_i)$ which, since $f_i(\hat{a}_i) = \hat{a}_i$, is satisfied by defining $\hat{a}_i$ as $\hat{a}_i = \hat{y}_i$. If the activation is a sigmoid, then $\hat{y}_i = 1/(1 + \exp(-\hat{a}_i))$ is sought, thence we let