18

Partially Connected Locally Recurrent Probabilistic Neural Networks

Todor D. Ganchev, Konstantinos E. Parsopoulos, Michael N. Vrahatis, and Nikos D. Fakotakis
University of Patras
Greece

1. Introduction

In this chapter, we review existing locally recurrent neural networks and introduce a novel artificial neural network architecture that merges the locally recurrent probabilistic neural networks (LRPNN) with swarm intelligence algorithms and concepts.

In particular, we develop an enhanced LRPNN model, referred to as *Partially Connected LRPNN* (PC-LRPNN). In contrast to LRPNN, where the recurrent layer consists of a set of fully connected neurons, the proposed new architecture assumes a *swarm* of neurons in the recurrent layer. Each neuron of the swarm presumes a neighbourhood of neurons with which it communicates through interconnections. The locality that determines the neighbourhoods is defined based on existing neighbourhood and communication schemes proposed in the swarm intelligence literature. Obviously, the PC-LRPNN offers a more general scheme, in which the fully connected LRPNN can be considered as a particular case, where all links in the recurrent layer are implemented.

The neighbourhood topology of the new, swarm-based recurrent layer can be either static or dynamic. Dynamic neighbourhoods have been studied extensively in the field of swarm intelligence, since swarms with dynamic communication schemes among individuals have been shown to achieve remarkably better results than swarms with static communication schemes in the field of optimization. Also, the plasticity of the neighbourhoods can be useful in cases where better fit to unknown data is required. In the present chapter we will limit our exposition to the static neighbourhoods, which are defined once during training, and remain unchanged during the operation of the PC-LRPNN. However, the concepts that we introduce here can be extended further to the dynamic counterparts.

The aforementioned local neighbourhoods and communications schemes facilitate the optimization of the recurrent layer linkage, which leads to much faster operation of the neural network, when compared to the fully linked structure. Furthermore, it significantly reduces the computational load for the overall training of the recurrent layer, which is performed at each case using the Particle Swarm Optimization (PSO) algorithm. Equipping the PC-LRPNN with PSO, results in an efficient hybrid scheme that takes advantage of the virtues of the probabilistic neural networks (PNN), recurrent neural networks (RNN), swarm intelligence concept, and that can tackle successfully real-life classification problems that assume temporal or spatial correlations among subsequent events.

Source: Recurrent Neural Networks, Book edited by: Xiaolin Hu and P. Balasubramaniam, ISBN 978-953-7619-08-4, pp. 400, September 2008, I-Tech, Vienna, Austria

2. Locally recurrent neural networks

A large number of recurrent and locally recurrent neural networks (LRNNs) have been studied in the literature. All they posses the valuable virtue to learn temporal dependences among the training data, which allows for context awareness, and thus, for improved recognition capabilities when compared to their non-recurrent counterparts. This advantage has proved useful in numerous applications of the LRNNs on real-life problems, which among others include: nonlinear system identification (Back & Tsoi, 1992; Lin et al., 1998); grammatical inference (Lin et al., 1998); weather prediction (Aussem et al., 1995); speech recognition (Kasper et al., (1995, 1996)); protection of power systems (Cannas et al., 1998); speaker verification (Ganchev et al., (2003, 2004, 2007)); wind speed prediction (Barbounis & Theocharis, (2007a, 2007b)), etc.

The locally recurrent global feedforward architecture was originally proposed by Back and Tsoi (Back & Tsoi, 1991), who considered an extension of the Multilayer Perceptron (MLP) neural network to exploit contextual information. In their work, they introduced the Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) synapses, able to utilize temporal dependencies in the input data. The FIR synapse has connections to its own, current and delayed, inputs, while the IIR synapse has also connections to its past outputs.

Ku and Lee (Ku & Lee, 1995) proposed Diagonal Recurrent Neural Networks (DRNN) for the task of system identification in real-time control applications. Their approach is based on the assumption that a single feedback from the neuron's own output is sufficient. Thus, they simplify the fully connected neural network to render training easier.

A comprehensive study of several MLP-based Locally Recurrent Neural Networks is available in (Campolucci et al., 1999). They introduced a unifying framework for the gradient calculation techniques, called Causal Recursive Back-Propagation. All aforementioned approaches consider gradient-based training techniques for neural networks, which, as it is well known, require differentiable transfer functions.

From the abundance of LRNN, in the present work, we will consider primary architectures originating from the family of the Probabilistic Neural Network (PNN). Specifically in the present section we will briefly outline the Locally Recurrent Probabilistic Neural Network (LRPNN), which was introduced (Ganchev et al., 2003) as an extension of the feed-forward Probabilistic Neural Network (PNN) architecture (Specht, (1988, 1990)). This structure is used as basis for the novel partially connected LRPNN (PC-LRPNN), which we will discuss in the next sections.

In brief, the LRPNN was derived from the original PNN by incorporating an additional hidden layer, referred to as recurrent layer, between the summation layer and the output competitive layer of the PNN structure. The recurrent layer consists of neurons possessing feedbacks from all other neurons in that layer. Due to this recurrent layer, the LRPNN, in contrast to the original PNN, is sensitive to the context in which the individual input data appear, and thus, it is capable to learn temporal regularities and the sequence of occurrence of events. Specifically, in the frame of speech processing this new capability of the LRPNN enables detecting and exploiting the abundance of correlations among speech features vectors estimated for successive speech frames. Exploiting these correlations was found important for improving the classification accuracy in the speaker verification task (Ganchev et al., (2003, 2004, 2007)).

As presented in earlier studies (Ganchev et al., (2003, 2004)) in the LRPNN architecture each neuron in the recurrent layer receives as input not only current values of its inputs, but also

the N previous outputs of all neurons in that layer. Broadly speaking, the input, acting on a recurrent neuron located in the recurrent hidden layer of an LRPNN, is a sum of two differences: The first difference is between the weighted probability of the given class and the sum of weighted probabilities computed for all other classes. These probabilities are computed at the output of the summation layer of the LRPNN. The second difference is between the weighted past output values of the given unit and the sum of the weighted past output values of all other neurons in this layer. Thus, in the proposed architecture, the probability of belonging to a specific class is combined with the probabilities computed for the other classes, and more importantly with the past values of the outputs of the recurrent units for all classes. This incorporation of previous information enables the LRPNN network to take advantage of the temporal context, which results in producing smoother in the time output scores, improved confidence levels, and consequently more accurate final decisions.
In the present chapter, we elaborate further on the LRPNN architecture by studying ways to optimize the recurrent layer linkage. In contrast to LRPNN, where the recurrent layer consists of a set of fully connected neurons, the introduced here new PC-LRPNN architecture assumes a *swarm* of neurons in the recurrent layer. Each neuron of the swarm presumes a neighbourhood of neurons with which it communicates through interconnections. The locality that determines the neighbourhoods is defined based on existing neighbourhood and communication schemes proposed in the swarm intelligence literature. When compared to the original LRPNN architecture, the PC-LRPNN has a greater capacity to adapt (its recurrent layer linkage) to the training dataset. This is due to the additional degree of freedom provided by the recurrent layer linkage selection that can be controlled for a fine-tuning of the neural network to the problem at hand. Obviously, the fully connected LRPNN architecture can be regarded as a particular case of the PC-LRPNN, which implements the full linkage in the recurrent layer.

3. Particle swarms and particle swarm optimization

The *particle swarm* is a community of individual performers, known as *particles*, which communicate/share information and collaborate on finding optimal regions in the search space. In the literature, the particle swarm is synonym to Particle Swarm Optimization (PSO) algorithm, which has become an attractive alternative to other optimization techniques (Clerc and Kennedy, 2002).
In brief, PSO is a stochastic optimization, population-based algorithm. It was introduced in 1995 by Kennedy and Eberhart (Kennedy & Eberhart, 1995), inspired by social behaviour simulation models. Features such as information exchange and neighbour alignment are inherent in such models, allowing the emergence of intelligent behaviour in swarms of simple agents with limited field of action. Similarly to evolutionary algorithms, PSO exploits a population, called a *swarm*, of potential solutions, called *particles*, which adapt their position stochastically at each iteration of the algorithm.
In contrast to standard evolutionary approaches, PSO promotes cooperativeness rather than competition among the solutions. More specifically, instead of using explicit mutation and selection operators in order to modify the population and favour the best performing individuals, PSO uses an adaptable position shift, called *velocity*, to move each particle to a new position at each iteration of the algorithm. The particles are moving towards promising regions of the search space by exploiting information springing from their own experience during the search as well as from the experience of other particles. For this purpose, a memory of the best position ever visited by each particle in the search space is retained.

In the context of single-objective optimization, the PSO can be outlined formally as follows: Let S be an n-dimensional search space, $f : S \to \mathbb{R}$ be the objective function, and N be the number of particles that comprise the swarm,

$$\mathbb{S} = \{x_1, x_2, \ldots, x_N\}. \tag{1}$$

Then, the ith particle is a point in the search space,

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \in S, \tag{2}$$

as well as its best position,

$$p_i = (p_{i1}, p_{i2}, \ldots, p_{in}) \in S, \tag{3}$$

which is the best position ever visited by x_i during the search. The velocity of x_i is also an n-dimensional vector,

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{in}). \tag{4}$$

In order to avoid biasing the swarm in specific parts of the search space, the particles as well as their velocities are randomly initialized in the search space.

Let $NG_i \subseteq \mathbb{S}$ be a set of particles that exchange information with x_i. This set is called the *neighbourhood* of x_i and it will be discussed later. Let also, g, be the index of the best particle in NG_i, i.e.,

$$f(p_g) \le f(p_l), \qquad \text{for all } l \text{ with } x_l \in NG_i, \tag{5}$$

and t denote the iteration counter. Then, the swarm is manipulated according to the equations (Eberhart & Shi, 2000),

$$v_{ij}(t+1) = w\, v_{ij}(t) + c_1\, r_1\, (p_{ij}(t) - x_{ij}(t)) + c_2\, r_2\, (p_{gj}(t) - x_{ij}(t)), \tag{6}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \tag{7}$$

where $i = 1, 2, \ldots, N$; $j = 1, 2, \ldots, n$; w is a positive parameter called *inertia weight*; c_1 and c_2 are two positive constants called *cognitive* and *social* parameter, respectively; and r_1, r_2, are realizations of two independent random variables that assume the uniform distribution in the range [0, 1]. The best position of each particle is updated at each iteration by setting

$$p_i(t+1) = x_i(t+1), \qquad \text{if } f(x_i) < f(p_i), \tag{8}$$

otherwise it remains unchanged. Obviously, an update of the index g is also required at each iteration.

The inertia weight was not used in early PSO versions. However, experiments showed that the lack of mechanism for controlling the velocities could result in *swarm explosion*, i.e., an unbounded increase in the magnitude of the velocities, which resulted in swarm divergence. For this purpose, a boundary, v_{max}, was imposed on the absolute value of the velocities, such that, if $v_{ij} > v_{max}$ then $v_{ij} = v_{max}$, and if $v_{ij} < -v_{max}$ then $v_{ij} = -v_{max}$. In later, more sophisticated versions, the new parameter was incorporated in the velocity update equation, in order to control the impact of the previous velocity on the current one, although the use of v_{max} was not abandoned.

Intelligent search algorithms, such as PSO, must demonstrate an ability to combine *exploration*, i.e., visiting new regions of the search space, and *exploitation*, i.e., performing more refined local search, in a balanced way in order to solve problems effectively (Parsopoulos & Vrahatis, (2002, 2004, 2007)). Since larger values of w promote exploration, while smaller values promote exploitation, it was proposed and experimentally verified that declining values of the inertia weight can provide better results than fixed values. Thus, an initial value of w around 1.0 and a gradually decline towards 0.0 are considered a good choice. On the other hand, the parameters c_1 and c_2 are usually set to fixed and equal values such that the particle is equally influenced by its own best position, p_i, as well as the best position of its neighbourhood, p_g, unless the problem at hand implies a different setting.
An alternative velocity update equation was proposed by Clerc & Kennedy, (2002),

$$v_{ij}(t+1) = \chi\,[v_{ij}(t) + c_1\, r_1\, (p_{ij}(t) - x_{ij}(t)) + c_2\, r_2\, (p_{gj}(t) - x_{ij}(t))], \tag{9}$$

where χ is a parameter called *constriction factor*. This version is algebraically equivalent with the inertia weight version of (6). However, the parameter selection in this case is based on the stability analysis due to Clerc and Kennedy (2002), which expresses χ as a function of c_1 and c_2. Different promising models were derived through the analysis of the algorithm, with the setting $\chi = 0.729$, $c_1 = c_2 = 2.05$, providing the most promising results and robust behaviour, rendering it the default PSO parameter setting.
Regardless of the PSO version used, it is clear that its performance is heavily dependent on the information provided by the best positions, p_i and p_g, since they determine the region of the search space that will be visited by the particle. Therefore, their selection, especially for p_g, which is related to information exchange, plays a central role in the development of effective and efficient PSO variants. Moreover, the concept of neighbourhood mentioned earlier in this section, raises efficiency issues. A neighbourhood has been already defined as a subset of the swarm. The most straightforward choice would be to consider as neighbours of the particle x_i, all particles enclosed in a sphere with centre x_i and a user-defined radius in the search space. Despite its simplicity, this approach increases significantly the computational burden of the algorithm, since it requires the computation of all distances among particles at each iteration. This deficiency has been addressed by defining neighbourhoods in the space of particles' indices instead of the actual search space.
Thus, the neighbours of x_i are determined based solely on the indices of the particles, assuming different *neighbourhood topologies*, i.e., orderings of the particles' indices. The most common neighbourhood is the *ring topology*, depicted in Fig. 1 (left), where the particles are arranged on a ring, with x_{i-1} and x_{i+1} being the immediate neighbours of x_i, and x_1 following immediately after x_N. Based on this topology, a neighbourhood of radius r of x_i is defined as

$$NG_i(r) = \{x_{i-r}, x_{i-r+1}, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{i+r-1}, x_{i+r}\}, \tag{10}$$

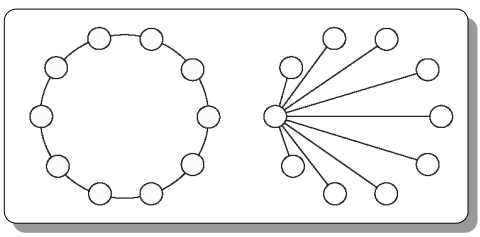


Fig. 1. The ring (left) and star (right) neighbourhood topologies of PSO

and the search is influenced by the particle's own best position, p_i, as well as the best position of its neighbourhood. This topology promotes exploration, since the information carried by the best positions is communicated slowly through the neighbours of each particle. A different topology is the *star topology*, depicted in Fig. 1 (right) where all particles communicate only with a single particle, which is the overall best position, p_g, of the swarm, i.e., $NG_i \equiv \mathbb{S}$. This topology promotes exploitation, since all particles share the same information. This is also called the *global variant* of PSO, denoted as *gbest* in the relative literature, while all other topologies with $NG_i \subset \mathbb{S}$, define *local variants*, usually denoted as *lbest*. Different topologies have also been investigated with promising results (Kennedy, 1999; Janson & Middendorf, 2005).

4. The partially connected locally recurrent probabilistic neural network

The LRPNN was derived (Ganchev et al., 2003) from the original PNN (Specht, 1988) by incorporating an additional hidden layer, referred to as *recurrent layer*, between the summation layer and the output competitive layer of the PNN structure. This recurrent layer consists of neurons possessing feedbacks with all other neurons in that layer. Elaborating on the LRPNN, here, we introduce the *Partially Connected LRPNN* (PC-LRPNN) architecture. Fig. 2 presents the simplified structure of a PC-LRPNN for classification in K classes. In contrast to the fully connected LRPNN, where each neuron in the recurrent layer communicates with all other neurons in that layer (i.e. *global communication* is enabled), in the PC-LRPNN the recurrent layer linkage is implemented only partially, depending on the problem at hand and the actual training data. This is illustrated in Fig. 2, where the

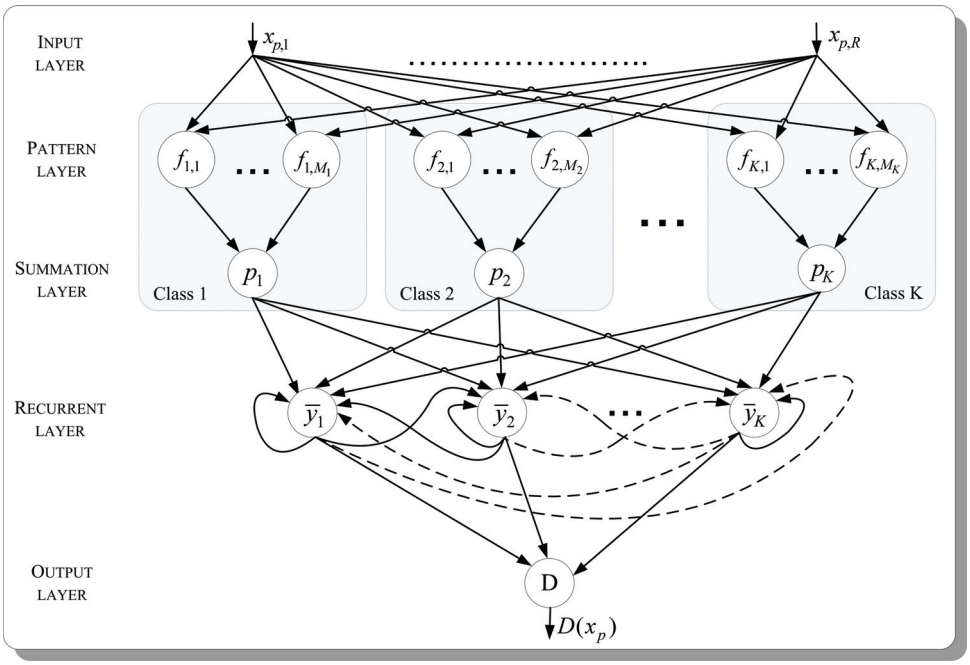


Fig. 2. Structure of the Partially Connected Locally Recurrent Probabilistic Neural Network

dashed line indicates that the linkage between neurons $\overline{y}_1$, $\overline{y}_2$ and $\overline{y}_K$ might not be implemented. In general, the concept of partially connected recurrent layer can be regarded as defining local neighbourhoods for each of the recurrent layer neurons. This can be viewed as establishing a *swarm of neurons* which cooperate (i.e. exchange information) in order to categorize more precisely a given unknown input. However, in contrast to the classic particle swarms that are utilized in the PSO schemes, here the local neural neighbours are not defined by the specific values of the neurons' indexes but the swarm members are selected during training, on a competitive basis, and in data-dependent manner, with respect to certain predefined criterion. In practice, the size of neighbourhood and the recurrence depth (i.e. the depth of memory) in the recurrent layer are specified depending on *a priori* knowledge about the specific problem at hand, or are identified heuristically after some experimentation with a representative dataset.

However, before describing any specific strategy for implementing the (partial) linkage of the recurrent layer, for comprehensiveness of exposition we briefly outline the PC-LRPNN architecture. In brief, the first two hidden layers the PC-LRPNNs, as their predecessor — the PNNs, implement the Parzen window estimator (Parzen, 1962) by using a mixture of Gaussian basis functions. If a PC-LRPNN for classification in K classes is considered, the class conditional probability density function $p_i(\mathbf{x}_p \mid k_i)$ is defined as:

$$p_i(\mathbf{x}_p \mid k_i) = \mathrm{f}_i(\mathbf{x}_p) = \frac{1}{(2\pi)^{d/2}\sigma_i^d} \cdot \frac{1}{M_i} \sum_{j=1}^{M_i} \exp\left(-\frac{1}{2\sigma_i^2} (\mathbf{x}_p - \mathbf{x}_{ij})^T (\mathbf{x}_p - \mathbf{x}_{ij}) \right), \; i = 1,2,...,K, \quad (11)$$

where for simplicity of further notations $p_i(\mathbf{x}_p \mid k_i)$ is replaced by $\mathrm{f}_i(\mathbf{x}_p)$. Here $\mathbf{x}_{ij}$ is the j th training vector from class κ_i, $\mathbf{x}_p$ belonging to the set $\mathbf{X} = \{\mathbf{x}_p\}$, with $p = 1,2,...,P$, is the p th input vector, d is the dimension of the input vectors, and M_i is the number of training patterns in class κ_i. Each training vector $\mathbf{x}_{ij}$ is assumed a centre of a kernel function, and consequently the number of pattern units in the first hidden layer of the neural network is given by the sum of the pattern units for all the classes. The standard deviation σ_i acts as a smoothing factor, which softens the surface defined by the multiple Gaussian functions. Instead of the simple covariance matrix, $\{\sigma_i^2 I\}$, where I represents the identity matrix, the full covariance matrix can be computed using the Expectation Maximization algorithm, as proposed in (Yang & Chen, 1998; Mak & Kung, 2000) and elsewhere. Since the computation of the covariance matrix, or the optimization of the smoothing factor σ_i, does not interfere with the development of the PNN we discuss, for simplicity of exposition, we consider here the simple case, where the value of the standard deviation is identical for all pattern units belonging to a specific class. Moreover, σ_i can be the same for all pattern units, irrespective of their class belonging, as it was originally proposed (Specht, 1990).

Next, the class conditional probability density functions $\mathrm{f}_i(\mathbf{x}_p)$ for each class κ_i, estimated through (11), act as inputs for the recurrent layer. In general, the recurrent layer can be considered as a form of Infinite Impulse Response (IIR) filter that smoothes the probabilities generated for each class, by incorporating information about the probabilities computed for all other classes, and more importantly, by exploiting one or more past values of the outputs for all classes.

The recurrent layer is composed of recurrent neurons, which in addition to the inputs coming from the summation layer also possess feedbacks from their own past outputs and from current and past outputs of the neurons of the other classes. Fig. 3 illustrates the linkage of a single neuron belonging to the hidden recurrent layer. As shown in the figure, beside the PDFs from all classes, $\mathrm{f}_i(\mathbf{x}_p)$, $i=1,2,...,K$, this neuron also receives feedbacks from its past outputs, $\overline{\mathrm{y}}_i(\mathbf{x}_{p-t})$, $t=1,2,...,N$, with i denoting the current neuron number, as well as from current $\mathrm{y}'_{j\neq i}(\mathbf{x}_p)$, $j=1,2,...,K$ and past $\overline{\mathrm{y}}_{j\neq i}(\mathbf{x}_{p-t})$, $j=1,2,...,K$, $t=1,2,...,N$, outputs from all other neurons belonging to that layer. Here, the subscript p stands for the serial number of the input vector $\mathbf{x}_p$. On its own side, the current neuron provides to the other neurons of the recurrent layer its current $\mathrm{y}_i(\mathbf{x}_p)$ and past $\overline{\mathrm{y}}_i(\mathbf{x}_{p-t})$, $t=1,2,...,N$ outputs, again with p standing for the specific input vector.

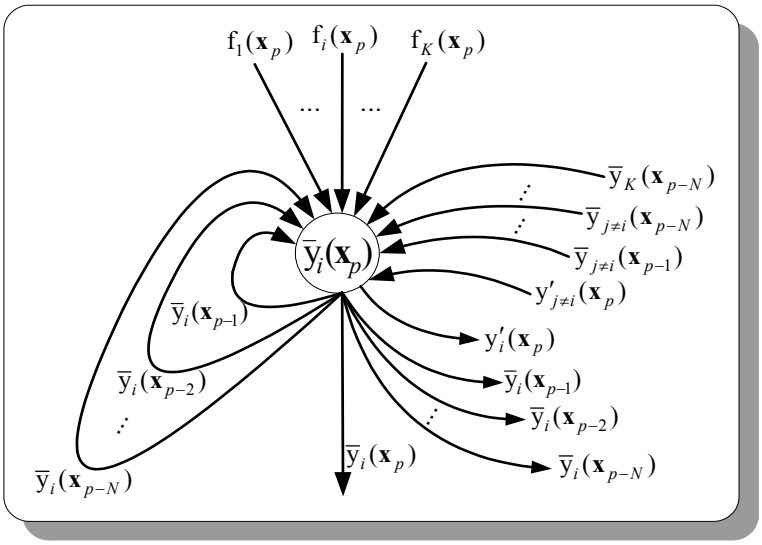


Fig. 3. Linkage of a neuron that belongs to the recurrent layer

A detailed structure of the recurrent neurons is provided in Fig. 4. As the figure presents, the inputs $\mathrm{f}_i(\mathbf{x}_p)$, $i=1,2,...,K$, denoting the class conditional PDFs, are weighted by the coefficients $b_{i,j}$. The two indexes of the weights of $b_{i,j}$ with $i=1,2,...,K$ and $j=1,2,...,K$ stand for the current recurrent neuron and for the class to which the corresponding input belongs. The first two indexes of the weights $a_{i,j,t}$ have the same meaning as for the weights $b_{i,j}$, and the third index $t=1,2,...,N$ shows the time delay of the specific output before it appear as an input.

All feedbacks $\overline{\mathrm{y}}_i(\mathbf{x}_{p-t})$, $t=1,2,...,N$ that originate from the present neuron i, and the links $\overline{\mathrm{y}}_{j\neq i}(\mathbf{x}_{p-t})$, $j=1,2,...,K$, $t=1,2,...,N$ coming from the other neurons $j\neq i$ of the recurrent layer are weighted by the coefficients $a_{i,i,t}$, $t=1,2,...,N$ and $a_{i,j\neq i,t}$, $j=1,2,...,K$, $t=1,2,...,N$, respectively.

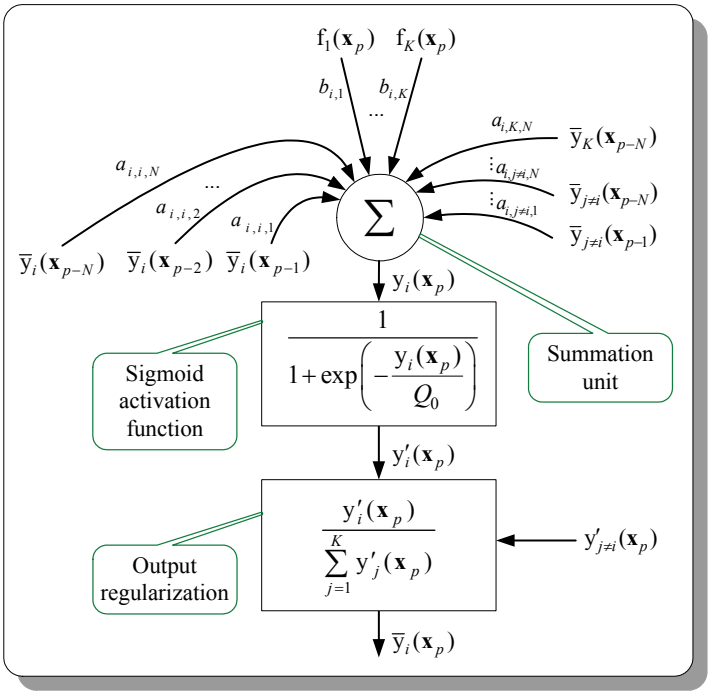


Fig. 4. Internal structure of the *i*th neuron from the recurrent layer of the PC-LRPNN

The summation units' output $y_i(\mathbf{x}_p)$ of the locally recurrent layer is computed by:

$$y_i(\mathbf{x}_p) = \left[b_{i,i} f_i(\mathbf{x}_p) - \sum_{\substack{k=1 \\ i\neq k}}^{K} b_{i,k} f_k(\mathbf{x}_p) \right] + \sum_{t=1}^{N} \left[a_{i,i,t} \bar{y}_i(\mathbf{x}_{p-t}) - \sum_{\substack{k=1 \\ i\neq k}}^{K} a_{i,k,t} \bar{y}_k(\mathbf{x}_{p-t}) \right], \; i = 1,2,...,K, \quad (12)$$

where $f_i(\mathbf{x}_p)$ is the probability density function of each class κ_i, $\mathbf{x}_p$ is the p th input vector, K is the number of classes, N is the recurrence depth, $\bar{y}_i(\mathbf{x}_{p-t})$ is the normalized past output for class κ_i that has been delayed on t time steps, and $a_{i,j,t}$ and $b_{i,j}$ are weight coefficients. The output $y_i(\mathbf{x}_p)$ of each summation unit from the recurrent layer is subject to the regularization transformation:

$$\bar{y}_i(\mathbf{x}_p) = \frac{\mathrm{sgm}\left(y_i(\mathbf{x}_p)\right)}{\sum_{j=1}^{K} \mathrm{sgm}\left(y_j(\mathbf{x}_p)\right)}, \; i = 1,2,...,K, \quad (13)$$

which retains the probabilistic interpretation of the output of the recurrent layer. Here, the designation sgm refers to the sigmoid activation function.

Subsequently, in the output layer, often referred as *competitive layer*, the Bayesian decision rule (14) is applied to distinguish class κ_i, to which the input vector $\mathbf{x}_p$ is categorized:

$$\mathrm{D}(\mathbf{x}_p) = \underset{i}{\operatorname{argmax}} \left\{ h_i c_i \, \overline{\mathrm{y}}_i(\mathbf{x}_p) \right\}, \; i = 1,2,...,K, \tag{14}$$

where h_i is *a priori* probability of occurrence of a pattern from class κ_i, and c_i is the cost function associated with the misclassification of a vector belonging to class κ_i.

Finally, provided that all classes are mutually exclusive and exhaustive, we can compute the Bayesian confidence for every decision $\mathrm{D}(\mathbf{x}_p)$ by applying the Bayes' theorem:

$$P(k_i \mid \mathbf{x}_p) = \frac{h_i \, \overline{\mathrm{y}}_i(\mathbf{x}_p)}{\sum_{j=1}^{K} h_j \, \overline{\mathrm{y}}_j(\mathbf{x}_p)}, \; i = 1,2,...,K. \tag{15}$$

The posterior probability $P(k_i \mid \mathbf{x}_p)$ for the p th input vector belonging to class κ_i is computed by relying on the *a priori* probabilities h_i and the temporally smoothed PDFs $\overline{\mathrm{y}}_i(\mathbf{x}_p)$.

The decision $\mathrm{D}(\mathbf{x}_p)$, and the confidence for every decision $P(k_i \mid \mathbf{x}_p)$, are computed for every input vector. However, in many practical applications (such as speaker verification, speaker identification, emotion detection, etc) every test trial (usually a speech utterance) consists of multiple feature vectors. Therefore, the probability $P(k_i \mid \mathbf{X})$ all test vectors originating from a given test trial $\mathbf{X} = \{\mathbf{x}_p\}, \;\; p = 1,2,...,P$ to belong to class κ_i, can be computed by:

$$P(k_i \mid \mathbf{X}) = \frac{\sum_{p=1}^{P} \left[\mathrm{D}(\mathbf{x}_p) = k_i \right]}{\sum_{j=1}^{K} \sum_{p=1}^{P} \left[\mathrm{D}(\mathbf{x}_p) = k_j \right]}, \;\; i = 1,2,...,K, \tag{16}$$

where $\sum_{p=1}^{P} \left[\mathrm{D}(\mathbf{x}_p) = k_i \right]$ is the number of vectors $\mathbf{x}_p$ classified by the Bayesian decision rule (14) as belonging to class κ_i. In applications that assume an exhaustive taxonomy any of the inputs $\mathbf{x}_p$ falls in one of the classes κ_i, and therefore the equality:

$$P = \sum_{j=1}^{K} \sum_{p=1}^{P} \left[\mathrm{D}(\mathbf{x}_p) = k_j \right], \tag{17}$$

where P is the number of test vectors in the given trial $\mathbf{X}$, is always preserved.

However, in many real-world applications computing the probability $P(k_i \mid \mathbf{X})$ is not sufficient as a final outcome from the PC-LRPNN. In such cases, a final decision is made by applying the Bayesian decision rule:

$$D(\mathbf{X}) = \underset{i}{\operatorname{argmax}} \left\{ P(k_i \mid \mathbf{X}) \right\}, \;\; i = 1,2,...,K, \tag{18}$$

or alternatively, the outcome of (16) is assessed with respect to a predefined threshold θ:

$$P(k_i \mid \mathbf{X}) \begin{cases} > \theta & \text{decision \#1} \\ \leq \theta & \text{decision \#2} \end{cases} . \tag{19}$$

Most often, the threshold θ is computed on a data set, referred to as development or validation data, which is independent from the training and testing data. A necessary requirement for obtaining a reasonable estimate of θ is the development data to be representative, i.e., they have to bear a resemblance to the real-world data on which the PC-LRPNN will operate within the corresponding application.

5. Training the PC-LRPNN

In general, the training of the PC-LRPNNs is similar to the three-step training procedure of the original fully connected LRPNNs (Ganchev et al, 2004) except for one extra step that is PC-LRPNN specific. Specifically, in the LRPNN, the first two steps implement the usual strategy for training PNNs, while the third step adjusts the weights in the recurrent layer. In the PC-LRPNN the third training step is preceded by procedure which selects the actual linkage that will be implemented in the recurrent layer, i.e. the PC-LRPNN are trained in four steps. In the following we provide a concise description of the entire training process of the PC-LRPNN.

STEP 1: In brief, by analogy to the original PNN, the first training step creates the actual topology of the network. In the first hidden layer, a pattern unit for each training vector is created by setting its weight vector equal to the corresponding training vector. In order to reduce the amount of neurons, i.e. the computational load during operation, the training data can be compressed by performing some sort of clustering (for instance, k-means) as pre-processing of the training dataset. An alternative approach could be to employ pruning and discard redundant neurons, or to build the first layer in multistep manner by adding a new neuron only when there is compelling need this to be done.
The outputs of the pattern units associated with the class κ_i are then connected to one of the second hidden layer summation units. The number of summation units is equal to the number of target classes K. The outputs of the summation units can be fed to some or all neurons of the recurrent layer, depending on the implemented linkage.

STEP 2: The second training step is the computation of the smoothing parameter σ_i for each class. To this end, various approaches (Meisel, 1972; Cain, 1990; Specht, 1992; Musavi et al., 1992; Specht & Romsdahl, 1994; Masters, 1993; Georgiou et al., (2006, 2008), etc) have been proposed. Although other methods can be employed, here we will mention only the one (Cain, 1990) due to its simplicity. According to that approach, any σ_i is proportional to the mean value of the minimum distances among the training vectors in class κ_i:

$$\sigma_i = \lambda \frac{1}{M_i} \sum_{j=1}^{M_i} \min \left\{ \left\| \mathbf{x}_{i,j} - \mathbf{x}_{i,j \neq i} \right\|_2^2 \right\} \tag{20}$$

where $\mathbf{x}_{i,j}$ is the j th pattern unit (located in the pattern layer) for class κ_i; $\| . \|_2$ corresponds to the 2-norm on $\mathbb{R}^d$ (reminding that $\mathbf{x}_{i,j}$ are the stored training data, and therefore, $\mathbf{x}_{i,j} \in \mathbb{R}^d$); d is the dimensionality of the input data; the expression

$$\min\left\{\left\|\mathbf{x}_{i,j}-\mathbf{x}_{i,j\neq i}\right\|_2^2\right\} \tag{21}$$

represents the smallest Euclidean distance computed between j th pattern unit of class κ_i and all other pattern units from the same class; and M_i is the number of training patterns in class κ_i. The constant λ, which controls the degree of overlapping among the individual Gaussian functions, is usually selected in the range $\lambda \in [1.1, 1.4]$. If the smoothing parameter is common for all classes, either it is chosen empirically, or it is computed by applying (20) on the entire training data set.

Step 3: For the PC-LRPNNs, the third training step selects the recurrent layer linkage to be implemented. This linkage could be static, i.e. defined once during training, or dynamic, i.e. changing during operation of the PC-LRPNN, depending on the input sequences. Furthermore, it could be expected that many of the recurrent layer neurons will participate in multiple class-specific neighbourhoods, which are then combined to assemble the recurrent layer linkage, but there could be neurons that do not participate in any swarms and are left detached from their neighbours. Usually, the linkage selection is performed in a data-dependent manner but it could be also based on the indexes of the individual neurons, if there is such necessity they to be pre-specified or bounded.

In fact, the linkage selection consists in identifying a sufficient subset of connections which typically is much smaller than the size of the full linkage. An assortment of strategies can be applied for identifying the optimal subsets of interacting neurons, i.e. the scope of swarm, and the neighbourhood for each target class. For instance, examples could be strategies based on identifying the Top-C competitor classes for a given input sequence, and implementing the linkage only for the recurrent neurons corresponding to these classes. The linkage to the less-promising competitors, which are not members of the Top-C club, is not implemented. An alternative strategy could be to perform pruning of the connections, starting from the fully connected LRPNN and iteratively identifying and discarding links which are not contributing for maximizing the overall performance. Yet, another strategy could be to start from the simplest reasonable topology and continue adding connections until the performance of the PC-LRPNN increases, or predefined limits are reached. Other strategies might involve optimization of the linkage of each particular recurrent neuron or the amount of memory it possesses, and then organize teams of super-neurons, etc. Obviously, the most successful strategies should exploit any *a priori* knowledge about the problem at hand and be able to interpret properly the information available in the training dataset.

At this point, we need to remember that in the PC-LRPNNs we deal with classification scheme of the type winner-takes-all, and that the scores acting on the input of the recurrent layer are in fact the probabilities computed by the summation units in the previous layer. These probabilities compete for distinguishing the winning class, and in non-trivial multi-class problems there exist more than one probability bigger than zero. For this type of classification scheme, we can consider a straightforward but efficient and effective strategy that builds the recurrent layer linkage by identifying a neighbourhood for a given recurrent neuron in terms of its closest competitors for the prise. In such a strategy, we follow a two stage procedure:

1. Firstly, we identify the Top-C competitors for each target class, by feeding the original non-compressed training data for that class at the input of the already trained pattern layer. At the output of the class-specific summation units (residing in the summation layer), the outcome will be a set of $M_i \times K$ probabilities, with M_i indicating the number of feature vectors in the training dataset for class κ_i and K the total number of target classes. Having computed the matrix $M_i \times K$ for a specific class κ_i, we can identify the Top-C competitors by computing the average score per class, and sorting these values.
2. Subsequently, we implement symmetric connections only among these Top-C recurrent neurons. Here, symmetric stands for the case where each neuron that receives information form another neuron also supplies back to this neuron the equivalent information about its own class. Thus, the relationship between the two neurons is symmetric in terms of linkage. However, in the general case symmetry might not be reasonable or desirable and should not be imposed unless the properties of the underlying training data indicate such necessity, or there exists some a priori knowledge about the problem at hand.

Eventually, the recurrent layer linkage is formed as union of all class-specific neighbourhoods. This can be expressed as follows: Let L_0 be the $K \times K$ matrix which represents the connections originating form the output of the summation layer to the inputs of the recurrent layer neurons, and $l_0(i,j)=1$ indicates that the specific connection from the summation unit corresponding to class κ_i is connected to the recurrent neuron for class κ_j. Alternatively, the value $l_0(i,j)=0$ would indicate that the specific connection was not implemented. The individual elements of the L_0 matrix, i.e. $l_0(i,j)$, can be referred to as the mask which determines if the specific coefficients $b_{i,j}$ (refer to (12)) will be present or not. Obviously, it is mandatory for the diagonal elements of L_0 to have non-zero values, i.e. $l_0(i,i)=1$, for any $i=1,2,...,K,$ so a connection between the summation and recurrent layer in class κ_j is always guaranteed. As explained earlier, the rest of the linkage $l_0(i,j)\big|_{i\neq j}$ can be identified in a data-dependent manner (for instance, by following the Top-C strategy), or by utilizing *a priori* knowledge.

By analogy, let the $K \times K$ matrixes L_n, with $n=1,2,...,N,$ stand for the links that originate from the past outputs of the recurrent layer neurons to the inputs of neurons in the same layer. Here n is the index of delay, and the elements of L_n serve as a mask, which determines if the coefficients $a_{i,j,n}$ (refer to (12)) will exist, or not. Again, let the elements of L_1, $l_1(i,j)=1$, indicate that there exists a connection between the past output at time $t-1$ of the recurrent neuron for class κ_i and the input of the recurrent neuron for class κ_j, and $l_1(i,j)=0$ indicate for lack of connection. The same logic applies for the other matrixes L_n, but in contrast with L_0 there are no restrictions about the values of their elements, $l_n(i,j)$, i.e. there could be a case where all $l_n(i,j)=0$. In such a case, all recurrent feedbacks from past states as well as the connections between the recurrent neurons are dismissed, which is equivalent to recurrence depth $N=0$. When this is combined with strategy Top-1 (and all coefficients $b_{i,i}=1$), the PC-LRPNN becomes functionally identical to the original PNN. On

the other hand, when all $l_n(i,j)=1$ and $l_0(i,j)=1$ the structure of the PC-LRPNN coincides with the one of the fully connected LRPNN.
Eventually, the overall linkage of the recurrent layer is the composite matrix

$$L=\left[L_0 \vdots L_1 \vdots \ldots L_n \ \ldots \vdots L_N\right],\ n=1,2,\ldots,N, \tag{22}$$

with dimensionality $(N+1)xKxK$, where K is the total number of target classes, and N is the recurrence depth.
In general, the linkage defined by the matrixes L_n and L_0 can be identified using different strategies, or yet the same Top-C strategy. Furthermore, in the simplest scenario, the matrixes L_n could be duplicates of L_0, so L to have a repeating structure, however, this is not a requisite by any means. Once the proper linkage L is identified the weight of each connection needs to be estimated.

STEP 4: Finally, the forth training step consists in computation of the recurrent layer weights, using the uncompressed training data exploited at step three. In previous work (Ganchev et al, (2003, 2004)), we studied training strategies that aim at adjusting the weights in the recurrent layer in a manner that maximizes the classification accuracy on the training data set. Here we rely on another more successful strategy that was developed recently (Ganchev, in-press-2008). In brief, this new training strategy does not rely on a quantitative measure accounting for the classification performance on the training dataset, but merely aims at maximizing the probability for the target class and simultaneously minimizing the probabilities computed for the non-target classes over the training dataset. This leads to a simplification of the error function and reduction in the number of steps necessary for evaluating the goodness of the recurrent layer weights at each iteration.

Specifically, the new error function that is subject to minimization here involves the complementary to one value of the probability $P(\mathbf{X}_{k_i}\,|\,k_i)$, and the compound probability for $\mathbf{X}_{k_i}$ belonging to any other class:

$$\mathrm{E}(\mathbf{w})=\sum_{i=1}^{K} m_i\left(1-P(\mathbf{X}_{k_i}\,|\,k_i)\right)P(k_i)+\frac{1}{K-1}\sum_{i=1}^{K}\sum_{\substack{j=1\\ j\neq i}}^{K} m_j P(\mathbf{X}_{k_i}\,|\,k_j)P(k_j)\,. \tag{23}$$

Here $\mathbf{X}_{k_i}$ are the training data for class κ_i, and $P(k_i)$ are the *a priori* probability of class κ_i. Finally, the constants m_i and m_j determine the relative importance of (or alternatively the significance of misclassification of an input belonging to) the corresponding class κ_i or $\kappa_{j\neq i}$, respectively.
The first term in equation (23) estimates the distance between the probability $P(\mathbf{X}_{k_i}\,|\,k_i)$ and one, i.e. the error with respect to the probability computed for a perfect match to the model. This term causes the output for class κ_i of the trained recurrent layer to strive towards value one for input vectors that resemble the training dataset for that class. The second term in equation (23) is the cumulative error of $\mathbf{X}_{k_i}$ being acknowledged as belonging to any of the competitive classes $\kappa_{j\neq i}$. This second term contributes towards restraining the output values produced by the competitive classes for input data that belong to class κ_i.

The minimization of total error $E(\mathbf{w})$ is performed by employing a PSO algorithm Type 1 (Clerc & Kennedy, 2002), which was found more successful and/or much faster than other PSO implementations, such as the basic PSO (Eberhart & Shi, 2000), the local PSO as in (Liang et al., 2006), and the UPSO (Parsopoulos & Vrahatis, 2005).

6. Numerical evaluation

The experimentations reported in the present section aim at illustrating the operation of the PC-LRPNNs, but also serve as a scene for discussing the advantages and disadvantages of the PC-LRPNN, when compared to the original PNN and the fully connected LRPNN. Specifically, for the purpose of experimentations, we selected two interesting problems of different difficulty. Both of these problems are important for the development of human-friendly spoken dialogue applications, and by that reason they currently enjoy significant attention by the speech processing community. The first one is the text-independent speaker identification task, which is of moderate difficulty, and the second one is the speaker-independent emotion recognition task, which is well-known as an extremely challenging problem. In the following paragraphs we offer a brief outline of these tasks:

Task 1: Text-independent speaker identification

Speaker identification is multiple-class decision problem where the identity of a given speaker is judged based on a comparison of a sample of her\his voice against multiple pre-defined models. The outcome of this process is either a decision about the identity of the speaker or a notice that the present input cannot be categorized as any of the known speakers. In the closed set speaker identification that we consider here, the input speech utterances always belong to someone of the known speakers. Here, text-independence referrers to the specific aspect that no explicit modelling of the linguistic contents of the input utterance is performed. Thus, the outcome of the identification process is not dependent on the exact linguistic contents of the phrase, but only on the degree of proximity between the input speech signal and the predefined speaker models.

Task 2: Speaker-independent emotion recognition

The emotion classification task is a multiple-class decision problem, where the emotional state of a given speaker is judged based on comparison of an input (typically a speech utterance) against multiple pre-defined models for the emotional states of interest. Here, the notion for speaker-independency refers to the fact that the models for the emotional states of interest are general for a large population of people, and were built utilizing the speech of people who do not present in the test datasets. Emotion recognition from speech is a very challenging task mainly due to the inherent speaker-dependency of emotion expression but also due to the well-known multi-functionality of speech (Batliner & Huber, 2007).

6.1 Experimental protocol

Common training and testing protocols were followed in all experiments. All classifiers considered in the present evaluation (GMM, PNN, LRPNN, PC-LRPNN) were trained with common task-specific train datasets, and trials were performed with common task-specific test datasets.

For the purpose of the speaker identification task, ten female speakers, extracted from the PolyCost v1.0 telephone-speech speaker recognition corpus (Hennebert et al., 2000) were modelled as authorized users. The training data, comprised of ten utterances, containing

both numbers and sentences, obtained from the first session of each speaker. In average, about 17 seconds of voiced speech per speaker were available for training each user model. The test dataset consisted of 450 target trials, including 45 utterances per speaker. Each test trial involved approximately 3 seconds of speech. All speech recordings are of telephone bandwidth, sampled at 8 kHz, and A-law compressed.

For the purpose of the experimentations with the emotion classification task, we utilized the recordings of all eight speakers available in the Emotional Prosody Speech and Transcripts database (LDC, 2002). All recordings were split in utterances, with respect to the provided annotations, and then were down-sampled to 8 kHz and band-limited to telephone quality bandwidth.

In the specific experimental setup considered here, we carry out recognition of three emotional states: neutral, anger, and panic. This combination is of particular interests for practical applications, but also has proved as a very challenging set, since the members of the pairs: hot anger – panic, and cold anger – neutral, share a number of common prosodic characteristics.

The training dataset consist of the available recordings for the seven speakers and the recordings of the remaining speaker were used as the test dataset. Since one of the speakers had only neutral recordings, the training data for the anger and panic models were built from the recordings of six speakers. The amount of available data for training the speaker-independent models for the three emotional categories of interest was much different: approximately 1650, 380 and 180 seconds of speech for neutral, anger and panic, respectively. For the purpose of fair training of the emotion models, we performed k-means clustering as a pre-processing of the training dataset. The resultant codebooks, one per speaker, one per emotion category, were of size 256 feature vectors. Subsequently, these codebooks were used to train the neural network-based classifiers.

On the other hand, the GMM-based classifier was trained directly from the uncompressed dataset, for achieving a higher precision of the emotion models. The diagonal covariance GMM emotion models were trained via a standard version of the Expectation Maximization algorithm (McLachlan & Krishnan, 1997) with a maximum of 200 iterations. Training termination criterion was applied, and training process was interrupted if there was no error reduction among subsequent iterations.

The amount of target trials per category was 115, 29 and 18 utterances for the neutral, anger and panic, respectively. Each test trial consisted of approximately 3 seconds of speech.

In both tasks, only the voiced parts of the speech signal was parameterized to Mel-frequency cepstral coefficients (MFCC) with a rate of 100 feature vectors per second. We utilized the MFCC implementation of Slaney (Slaney, 1998), but adapted for sampling frequency of 8 kHz. This resulted in a filter-bank of thirty-two filters, which cover the frequency range [133, 3954] Hz, from which we computed 29 cepstral coefficients. In all experiments, we excluded the first cepstral coefficient (i.e. the one with index zero) from the feature vector, to avoid dependence on the recording setup (distance to the microphone, communication channel and handset mismatch, etc). Finally, in all experiments we considered a common feature vector consisting of the MFCC parameters {*MFCC(1)* ,…,*MFCC(28)*}. All parameters of the feature vector were normalized to fit in a common dynamic range.

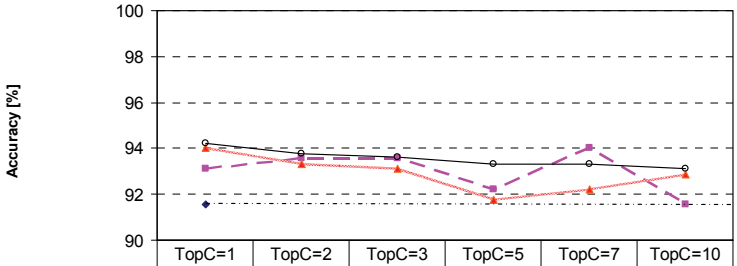
6.2 Experimental results

In this section, we study the performance of the PC-LRPNN for different connectivity range of the neighbourhood, Top-C, and different recurrence depth, N, of the recurrent layer. Comparisons with the PNN, GMM and the fully connected LRPNN are provided as follows:

The PC-LRPNN vs. the PNN and LRPNN, in the speaker identification task

Since in this task we consider identification of 10 different voices, i.e. we have 10 classes, we can note that in the case Top-C=10, the PC-LRPNN is equivalent to the fully connected LRPNN. On the other hand, in the case of $N=0$, Top-C=1, the PC-LRPNN has the same number of weights in the recurrent layer as the number of connections between the summation and competitive layers in the PNN. However, there is no equivalence between these two structures, mainly because the weights of the connections between the summation and recurrent layers in the PC-LRPNN are adjusted during training, while in the PNN they are all equal. This gives to the PC-LRPNN the capability to model better the training data.

In Fig. 5, we present the performance of the PC-LRPNN in the speaker identification task, and in Table 1, we show the number of recurrent layer weights for different values of the recurrence depth, N, and different values of the neighbourhood, Top-C. As the figure presents, for the PNN we obtained recognition accuracy of 91.6%, which we consider as the



	TopC=1	TopC=2	TopC=3	TopC=5	TopC=7	TopC=10
pcLRPNN(N=0)	93.1	93.6	93.6	92.2	94.0	91.6
pcLRPNN(N=1)	94.0	93.3	93.1	91.8	92.2	92.9
pcLRPNN(N=2)	94.2	93.8	93.6	93.3	93.3	93.1
PNN	91.6					

Fig. 5. Performance of the PC-LRPNN classifier on the speaker identification task, for different values of the recurrence depth, N, and different size of the recurrent layer neighbourhoods, Top-C.

Number of recurrent layer weights	*N=0*	*N=1*	*N=2*
Top-C=1	10	20	30
Top-C=2	30	60	90
Top-C=3	46	92	138
Top-C=5	68	136	204
Top-C=7	90	180	270
Top-C=10	100	200	300

Table 1. The number of recurrent layer weights to be trained, for different recurrence depth, N, and different size of the recurrent layer neighbourhood, Top-C

baseline. It is interesting to note that the performance of the PC-LRPNN for $N=0$, Top-C=1, i.e. when there are no recurrent feedbacks in the recurrent layer and the connection between the summation layer and recurrent layer neurons is implemented only for the top scoring candidates, is higher than the baseline, PNN, although the number of weights is equal. As explained above, this advantage of the PC-LRPNN comes from the fact that the values of these weights are trained in a data-dependent manner, while in the original PNN, these weight are equal. Furthermore, for recurrence depth $N=0$, the PC-LRPNN with neighbourhood Top-C=7 demonstrated the highest recognition accuracy (94.0%), which is higher than the recognition accuracy for the case of Top-C=10, i.e. the equivalent to the fully connected LRPNN. The last can be explained with the smaller number of weights to be adjusted for the case of Top-C=7, and the limited amount of training data.

Next, as Fig. 5 presents, the highest recognition accuracy among all (94.2%) was achieved for the PC-LRPNN with $N=2$, and Top-C=1. The top performance here illustrates both the importance of the recurrence depth (i.e. the memory about past states) and the capability of the PC-LRPNN to implement partial linkage in the recurrent layer. As the figure presents, the second best performance is shared between the PC-LRPNN with $N=1$, and Top-C=1, and the already discussed $N=0$, and Top-C=7. It is interesting to note that the recurrent PC-LRPNN ($N=1$) achieves this performance with only 20 weights, while the non-recurrent PC-LRPNN ($N=0$) needs 90 (please refer to Table 1.)

Speaking generally, we can conclude that the presented example on the speaker identification task illustrates undoubtedly that the PC-LRPNNs provide higher recognition accuracy than the baseline PNN. This advantage is mainly due to the exploitation of information from the competitive classes, which the recurrent layer neurons utilize for proper selection of the class belonging of a given input sequence. Furthermore, we observed that the PC-LRPNNs show performance even better than the one of the fully connected LRPNN. This superiority is due mainly to the additional degree of freedom that the PC-LRPNNs possess, i.e. the better flexibility to adjust the implementation of the recurrent layer linkage to the available training data. Finally, it is worth mentioning that due to the reduced number of weights, the PC-LRPNN are trained and operate much faster than the fully connected LRPNN.

The PC-LRPNN vs. the PNN, LRPNN and GMM, in the emotion recognition task

In the emotion recognition task, we firstly experimented with a state-of-the-art GMM-based classifier to identify the maximum performance that can be obtained (for a context-blind classifier) in our experimental setup. In Fig. 6, we present the identification accuracy obtained for different number of components in a Gaussian mixture. As the figure presents,

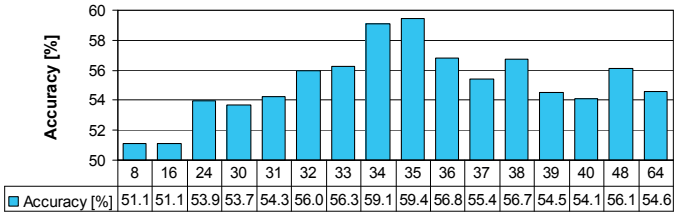
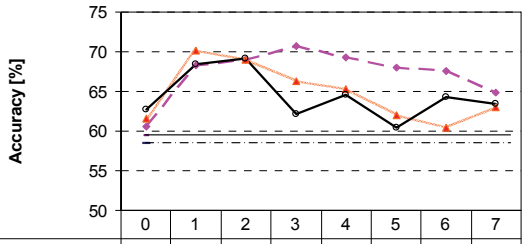


Fig. 6. Performance of the GMM classifier for different number of components on the speaker-independent emotion recognition task

the highest recognition accuracy (59.4%) was observed for the case of Gaussian mixture with 35 components. This performance will be considered as the baseline.
In Fig. 7, we present the recognition accuracy obtained for the PNN, PC-LRPNN and the LRPNN for different values of the recurrence depth, N, and different size of the recurrent layer neighbourhoods, Top-C. Table 2 presents the number of weights in the recurrent layer that need to be trained for the PC-LRPNN and LRPNN. Since in the present experimental setup we have three emotional categories, the PC-LRPNN with Top-C=3 is equivalent to the fully connected LRPNN.



	0	1	2	3	4	5	6	7
GMM	59.4							
PNN	58.5							
pcLRPNN(Top-1)	60.5	68.2	69.0	70.7	69.3	68.0	67.6	64.9
pcLRPNN(Top-2)	61.6	70.1	69.0	66.3	65.3	62.0	60.5	63.0
pcLRPNN(Top-3)	62.7	68.4	69.2	62.1	64.6	60.4	64.2	63.5

Fig. 7. Performance of the PC-LRPNN classifier on the emotion recognition task, for different values of the recurrence depth, N, and different size of the recurrent layer neighbourhoods, Top-C.

As the figure presents, the recognition accuracy obtained for the PNN is inferior to the one for the GMM classifier. The difference in performance of approximately 1% can be explained by the fact that here we employ the original homoscedastic PNN, which utilizes uniform smoothing factor σ_i for all classes, while the diagonal GMM employed here adjusts the variance for each class and thus is able to adapt better to the underlying distribution of the training data.

Number of recurrent layer weights	*N=0*	*N=1*	*N=2*	*N=3*	*N=4*	*N=5*	*N=6*	*N=7*
Top-C=1	3	6	9	12	15	18	21	24
Top-C=2	7	14	21	28	35	42	49	56
Top-C=3	9	18	27	36	45	54	63	72

Table 2. The number of recurrent layer weights to be trained, for different recurrence depth, N, and different size of the recurrent layer neighbourhood, Top-C

As Fig. 7 presents, the recognition accuracy observed for the PC-LRPNN and the fully connected LRPNN is superior to the one observed for the GMM and the PNN. Inspecting the recognition accuracy presented on the figure and the number of weights in the recurrent layer, presented in Table 2, we can notice that there are some relations among the performance results for similar number of coefficients. Furthermore, the general trend of the plots for different neighbourhood size, Top-C, seems to agree with respect to the

increase of the recurrence depth, N. On the present experimental setup, the PC-LRPNN outperforms entirely the fully connected LRPNN, due to its better capacity to adapt to the training data. Exception here is the case for recurrence depth $N = 6$, where the LRPNN outperforms significantly the partially connected counterpart.
Finally, the significant advantage of the PC-LRPNN and LRPNN over the PNN and GMM can be summarized as follows:

1. LRPNNs and PC-LRPNNs process the information coming from the competitive classes (for Top-C > 1) and the target class;
2. the recurrent structures are capable to capture temporal dependences among subsequent feature vectors, and thus, are capable to exploit the context in which a given input appears;
3. the recurrent layer is trained in a constructive manner to maximize the probability generated for the target class and to minimize to probabilities generated by the competitive classes, which favours resolving ambiguous situations.

The smoothing factor σ_i, the PC-LRPNN vs. the PNN

Utilizing the experimental setup of the emotion recognition task, and the best performing locally recurrent neural network, i.e. PC-LRPNN (Top-C=1, N =3), we would like to discuss an interesting phenomenon concerning the optimal value of the smoothing factor, σ_i.
Extensive experimentations with the PNN, PC-LRPNNs and fully connected LRPNNs, demonstrated that the estimation of the smoothing factor σ_i on the training dataset does not lead to optimal performance on the test dataset. This was especially topical in the emotion recognition task. To illustrate this phenomenon, in Fig. 8, we plot the performance of the PNN and the best performing PC-LRPNN for different values of the smoothing factor, σ_i.
For comprehensiveness of exposition we computed the recognition accuracy obtained on the training dataset, presented in the figure with dashed line, and on the test datasets, presented with solid line.

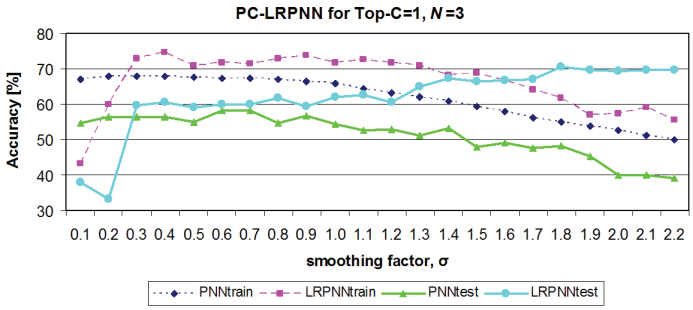


Fig. 8. Performance of the PNN and PC-LRPNN (Top-C=1, N =3), for different values of the smoothing factor σ_i

As the figure presents, for both the PNN and PC-LRPNN there was a significant gap between the recognition accuracy obtained on the training dataset and on the test data. Looking at the plots for the PNN, we can see that the trend of this difference in performance is a relatively smooth monotonically decreasing function. However, for the PC-LRPNN, the initial difference, for small values of σ_i, tends to decrease when σ_i increases. Furthermore,

the best performance for the PNN was obtained for σ_i =0.6, and the best performance for the various PC-LRPNN and LRPNN in the different experiment was for σ_i in the range [1.2, 2.0], even though the recognition performance for the training dataset was significantly lower, when compared to the optimal σ_i computed through (Cain, 1990), or any other method. Although the degree of learning for the training dataset varied greatly in the experiments with different PC-LRPNNs, and mostly ranged between 75% and 100%, the best performance on the test dataset was observed always for significantly higher values of σ_i, when compared to the best one for the training dataset. The last indicates that in challenging problems, for which it is known that there is significant mismatch between the training and operational conditions, the computation of the value of σ_i should be performed on another independent dataset, referred to as development or validation data. The development data are independent from the training and test datasets and serve for fine-tuning of the overall performance.

7. Conclusion and future research directions

Although the research on locally recurrent neural networks has a long record of history, the potential of development has not been exhausted. Moreover, in the last few years, there is a resumption of interest to the field, and recently some new paradigms appeared. These new architectures are in anticipation of further in depth studies, and further improvements and elaboration.

Speaking specifically for the family of LRPNNs, there is a compelling need for further studies that will investigate comprehensively how the recurrent layer linkage can be optimized for specific problem on specific dataset. Perhaps, new strategies for automatic selection of neighbourhood size and the specific neighbours of each recurrent neuron that arise directly from the training data will appear. It will be particularly interesting to study new algorithms for developing dynamically varying neighbourhoods, which depend on the input during operation of the neural network, and which go beyond the predefined during training look-up tables.

Finally, despite the progress made during the past decades, we deem that the locally recurrent neural networks still await for their golden time, when they will have significantly better biological plausibility. The human brain is still a source of inspiration, and we are looking forward to see how the development of the neuroscience will contribute further for the progress in the field of recurrent neural networks.

8. References

Aussem, A.; Murtagh, F. & Sarazin, M. (1995). Dynamical recurrent neural networks – towards environmental time series prediction, *International Journal of Neural Systems*, No.6, June 1995, pp.145–170.

Back, A.D. & Tsoi, A.C. (1991). FIR and IIR Synapses, a new neural network architecture for time series modelling, *Neural Computation*, Vol.3, 1991, pp.375–385.

Back, A.D. & Tsoi, A.C. (1992). Nonlinear system identification using multilayer perceptrons with locally recurrent synaptic structure, *Proceedings of 1992 IEEE-SP Workshop on Neural Networks for Signal Processing II*, 1992, pp.444–453.

Barbounis, T.G. & Theocharis, J.B. (2007a). A locally recurrent fuzzy neural network with application to the wind speed prediction using spatial correlation, *Neurocomputing*, Vol.70, No. 7–9, 2007, pp.1525–1542.

Barbounis, T.G. & Theocharis, J.B. (2007b). Locally recurrent neural networks for wind speed prediction using spatial correlation, *Information Sciences*, Vol. 177, No. 24, December 2007, pp.5775–5797.

Batliner, A. & Huber, R. (2007). Speaker characteristics and emotion classification, In: *Speaker Classification I, LNAI 4343*, C. Műller (Ed.), Springer, 2007, pp.138–151.

Cain, B.J. (1990). Improved probabilistic neural network and its performance relative to the other models, *Proceedings of the SPIE, Applications of Artificial Neural Networks*, Vol.1294, 1990, pp.354–365.

Cannas, B.; Celli, G.; Marchesi, M. & Pilo F. (1998). Neural networks for power system condition monitoring and protection, *Neurocomputing*, Vol.23, 1998, pp.111–123.

Clerc, M. & Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space, *IEEE Transactions on Evolutionary Computation*, 2002, Vol.6, pp.58–73.

Eberhart, R.C. & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization, *Proceedings of the Congress on Evolutionary Computing*, 2000, pp.84–88.

Ganchev, T.; Tasoulis, D.K.; Vrahatis, M.N. & Fakotakis, N. (2003). Locally recurrent probabilistic neural network for text-independent speaker verification, *Proceedings of 8th European Conference on Speech Communication and Technology, EUROSPEECH 2003*, September 1-4, 2003, Vol. 3, pp.1673–1676.

Ganchev, T.; Tasoulis, D.K.; Vrahatis, M.N. & Fakotakis, N. (2004). Locally recurrent probabilistic neural networks with application to speaker verification, *GESTS International Transaction on Speech Science and Engineering*, December 2004, Vol.1, No.2, pp.1–13.

Ganchev, T.; Tasoulis, D.K.; Vrahatis, M.N. & Fakotakis, N. (2007). Generalized locally recurrent probabilistic neural networks with application to text-independent speaker verification, *Neurocomputing*, Vol.70, No.7–9, 2007, pp. 1424–1438.

Ganchev, T. (in-press-2008). Enhanced training for the locally recurrent probabilistic neural networks, to apear in *International Journal on Artificial Intelligence Tools*, 2008.

Georgiou, V.L.; Pavlidis, N.G.; Parsopoulos, K.E.; Alevizos, Ph.D. & Vrahatis, M.N. (2006). New self-adaptive probabilistic neural networks in bioinformatic and medical tasks, *International Journal of Artificial Intelligence Tools*, 2006, Vol. 15, No. 3, pp. 371–396.

Georgiou, V.L.; Alevizos, Ph.D. & Vrahatis, M.N. (2008). Novel approaches to probabilistic neural networks through bagging and evolutionary estimating of prior probabilities, *Neural Processing Letters*, Vol. 27, No. 2, pp. 153–162.

Janson, S. & Middendorf, M. (2005). A hierarchical particle swarm optimizer and its adaptive variant, *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, Vol.35, No.6, December 2005, pp.1272–1282.

Hennebert, J.; Melin, H.; Petrovska, D. & Genoud, D. (2000). POLYCOST: A telephone-speech database for speaker recognition, *Speech Communication*, Vol.31, No.2-3, pp. 265–270.

Kasper, K.; Reininger, H.; Wolf, D. & Wust, H. (1995). A speech recognizer based on locally recurrent neural networks, *Proceedings of the International Conference on Artificial Neural Networks*, 1995, Vol. 2, pp.15–20.

Kasper, K.; Reininger, H. & Wust, H. (1996). Strategies for reducing the complexity of a RNN-based speech recognizer, *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing, ICASSP 1996*, Vol.6, pp.3354–3357.

Kennedy, J. (1999). Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance, *Proceedings of the 1999 Congress on Evolutionary Computation, CEC'99*, Vol.3, pp.1931–1938.

Kennedy, J. & Eberhart, R.C. (1995). Particle swarm optimization, *Proceedings of the IEEE International Conference on Neural Networks*, *ICNN 1995*, Vol.4, pp.1942–1948.

LDC (2002). University of Pennsylvania, Linguistic Data Consortium, Emotional prosody speech and transcripts (LDC2002S28), Available at: www.ldc.uppen.edu/Catalog/CatalogEntry.jsp?cataloId=LDC2002S28

Liang, J.J.; Qin, A.K.; Suganthan, P.N. & Baskir, S. (2006). Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, *IEEE Transactions on Evolutionary Computation*, 2006, Vol.10, No.3, pp.281–295.

Lin, T.; Horne, B.G. & Giles, C.L. (1998). How embedded memory in recurrent neural network architectures helps learning long-term temporal dependencies, *Neural Networks*, 1998, Vol.11, pp.861–868.

Mak, M.W. & Kung, S.Y. (2000). Estimation of elliptical basis function parameters by the EM algorithm with application to speaker verification, *IEEE Transactions on Neural Networks*, Vol.11, No.4, 2000, pp. 961–969.

McLachlan, G.J. & Krishnan, T. (1997). *The EM algorithm and extensions*. Wiley Series in Probability and Statistics. New York: Wiley, 1997.

Meisel, W. (1972). *Computer-oriented approaches to pattern recognition*. Academic Press, New York, 1972.

Parsopoulos, K.E. & Vrahatis, M.N. (2002). Recent approaches to global optimization problems through particle swarm optimization, *Natural Computing*, Vol.1, No.2-3, pp.235-306.

Parsopoulos, K.E. & Vrahatis, M.N. (2004). On the computation of all global minimizers through particle swarm optimization, *IEEE Transactions on Evolutionary Computation*, Vol.8, No. 3, pp.211–224.

Parsopoulos, K.E. & Vrahatis, M.N. (2005). Unified particle swarm optimization for tackling operations research problems, *Proceedings of the IEEE Swarm Intelligence Symposium, SIS 2005*, June 2005, pp.53–59.

Parsopoulos, K.E. & Vrahatis, M.N. (2007). Parameter selection and adaptation in unified particle swarm optimization, *Mathematical and Computer Modelling*, Vol.46, No.1-2, pp.198–213.

Slaney, M. (1998). Auditory toolbox. Version 2. Technical Report #1998-010, Interval Research Corporation.

Specht, D.F. (1988). Probabilistic neural networks for classification, mapping, or associative memory, *Proceedings of the IEEE Conference on Neural Networks*, 1988, Vol.1, pp.525–532.

Specht, D.F. (1990). Probabilistic neural networks, *Neural Networks*, 1990, Vol.3, No.1, pp.109–118.

Specht, D.F. (1992). Enhancements to probabilistic neural networks, *Proceedings of the IEEE International Joint Conference on Neural Networks*, *IJCNN 1992*, Vol.1, pp. 761–768.

Specht, D.F. & Romsdahl, H. (1994). Experience with adaptive PNN and adaptive GRNN, *Proceedings of the IEEE International Conference on Neural Networks*, *ICNN 1994*, Vol.2, pp.1203–1208.

Yang, Z.R. & Chen, S. (1998). Robust maximum likelihood training of heteroscedastic probabilistic neural networks, *Neural Networks*, 1998, Vol.11, No.4, pp.739–748.