Probabilistically Grounded Unsupervised Training of Neural Networks

Edmondo Trentin and Marco Bongini

Abstract The chapter is a survey of probabilistic interpretations of artificial neural networks (ANN) along with the corresponding unsupervised learning algorithms. ANNs for estimating probability density functions (pdf) are reviewed first, including parametric estimation via constrained radial basis functions and nonparametric estimation via multilayer perceptrons. The approaches overcome the limitations of traditional statistical estimation methods, possibly leading to improved pdf models. The focus is then moved from pdf estimation to online neural clustering, relying on maximum-likelihood training. Finally, extension of the techniques to the unsupervised training of generative probabilistic hybrid paradigms for sequences of random observations is discussed.

1 Introduction

Alongside with traditional statistical techniques, learning machines have long been applied to pattern classification, nonlinear regression, and clustering. Despite such a popularity, practitioners are usually not concerned with (nor, even aware of) any probabilistic interpretation of the machines they use. Nonetheless, mathematically grounded interpretation of learnable models turns out to be fruitful in terms of better understanding the nature of the models, as well as of better fitting the (probabilistic) laws characterizing the data samples at hand. In fact, theorems confirm that, under rather loose conditions, artificial neural networks (ANNs) can be trained as optimal estimates of Bayes posterior probabilities. Whilst estimation of probabilistic quantities via ANNs is feasible due to the simplicity of satisfying the probability constraints, probabilistic-grounded unsupervised training of connectionist models (e.g., estimation of probability density functions (pdf), feature extraction/reduction, data clustering) may be much harder. In density estimation, for instance, the ANN must cope with the very nature of the unknown pdf underlying the unlabeled data sample, i.e., a nonparametric function which may possibly take any non-negative, unbounded value, and whose integral over the feature space shall equal 1.

E. Trentin (✉) • M. Bongini
DIISM, University of Siena, Siena, Italy
e-mail: trentin@dii.unisi.it; bongini@dii.unisi.it

M.E. Celebi, K. Aydin (eds.), *Unsupervised Learning Algorithms*,
DOI 10.1007/978-3-319-24211-8_18

Yet, neural models of pdf could fit, and potentially improve, the advantages of parametric and nonparametric statistical estimation techniques. Moreover, these estimates could be valuable for enforcing statistical paradigms for sequential and structured pattern recognition, namely hidden Markov models (HMM) and some probabilistic graphical models, which rely on the so-called emission pdfs (that, in turn, are usually modeled under strong parametric assumptions). Furthermore, some relevant instances of clustering and feature reduction algorithms stem (implicitly or explicitly) from a proper probabilistic description (i.e., the pdf) of the data.

This chapter surveys probabilistic interpretations of ANNs, moving the focus from the traditional supervised learning of posterior probabilities to unsupervised learning of pdfs, clustering, and density estimation for sequence processing. The approaches are expected to overcome the major limitations of traditional statistical methods, possibly leading to improved models. Unsupervised density estimation is reviewed first (Sect. 2). A constrained maximum-likelihood algorithm for unsupervised training of radial basis functions (RBF) networks is handed out in Sect. 2.1, resulting in a connectionist parametric pdf estimation tool. A survey of techniques for nonparametric density estimation via multilayer perceptrons is then reviewed in Sect. 2.2. Starting from pdf estimation and the maximum-likelihood criterion, it is then feasible to move to a popular online neural algorithm for clustering unsupervised samples of data (Sect. 3), and even to probabilistic modeling of sequences of random observation drawn from an unknown pdf, by means of a hybrid ANN/hidden Markov model paradigm (reviewed in Sect. 4). Conclusions are drawn in Sect. 5.

2 Unsupervised Estimation of Probability Density Functions

One major topic in pattern recognition is the problem of estimating pdf [11]. Albeit popular, parametric techniques (e.g., maximum likelihood for Gaussian mixtures) rely on an arbitrary assumption on the form of the underlying, unknown distribution. Nonparametric techniques (e.g., k_n-nearest neighbors [11]) remove this assumption and attempt a direct estimation of the pdf from a data sample. The Parzen Window (PW) is one of the most popular nonparametric approaches to pdf estimation, relying on a combination of local window functions centered on the patterns of the training sample [11]. Although effective, PW suffers from several limitations, including: (1) the estimate is not expressed in a compact functional form (i.e., a probability law), but it is a sum of as many local windows as the size of the sample; (2) the local nature of the window functions tends to yield a fragmented model, which is basically "memory based" and (by definition) is prone to overfitting; (3) the whole training sample has to be kept always in memory in order to compute the estimate of the pdf over any new (test) patterns, resulting in a high complexity of the technique in space and time; (4) the form of the window function chosen has a deep influence on the eventual form of the estimated model, unless an asymptotic case (i.e., infinite sample) is considered; (5) the PW model heavily depends on the choice of an initial width of the local region of the feature space where the windows are centered.

ANNs are, in principle, an alternative family of nonparametric models [18]. Given the "universal approximation" property [4] of certain ANN families (multilayer perceptrons [37] and radial basis function networks [4]), they might be a suitable model for any given (continuous) form of data distributions. While ANNs are intensively used for estimating probabilities (e.g., posterior probabilities in classification tasks [4]), they have not been intensively exploited so far for estimating pdfs. One of the main rationales behind this fact is that connectionist modeling of probabilities is easily (and somewhat heuristically) achieved by standard supervised backpropagation [37], once 0/1 target outputs are defined for the training data [4] (as pointed out in Sect. 1). Moreover, it is also simple to introduce constraints within the model that ensure the ANN may be interpreted in probabilistic terms, e.g., using sigmoid output activations (that range in the $(0, 1)$ interval), along with a softmax-like mechanism [2] which ensures that all the outputs sum to 1. Learning a pdf, on the contrary, is an unsupervised and far less obvious task.

In the following, we discuss the connectionist approaches to the problem of pdf estimation. The former generates within a maximum-likelihood parametric estimation technique under specific assumptions on the form of the underlying pdf, namely a Gaussian-based RBF network. It is presented in Sect. 2.1 and has its point of strength in the overwhelming simplicity. More generic (and recent) nonparametric frameworks are reviewed in Sect. 2.2, relying on multilayer perceptrons.

2.1 Estimating pdfs via Constrained RBFs

Let $\mathscr{T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a random sample of N d-dimensional continuous-valued observations, identically and independently drawn (iid) from the unknown pdf $p(\mathbf{x})$. A radial basis function (RBF)-like network [4] can be used to obtain a model $\hat{p}(\mathbf{x}|\boldsymbol{\theta}_{\hat{p}})$ of the unknown pdf $p(\mathbf{x})$, where $\boldsymbol{\theta}_{\hat{p}}$ is the parameter vector of the RBF. Namely, $\boldsymbol{\theta}_{\hat{p}}$ includes the RBF hidden-to-output weights, and the parameters characterizing the Gaussian kernels. In order to ensure that a pdf is obtained (i.e., non-negative outputs, and a unit integral over the whole definition domain), constraints have to be placed on the hidden-to-output connection weights of the RBF (assuming that normalized Gaussian kernels are used).

Relying on the iid assumption, maximum-likelihood (ML) estimation of the RBF parameters $\boldsymbol{\theta}_{\hat{p}}$ given $\mathscr{T}$ is accomplished maximizing the quantity

$$p(\mathscr{T}|\boldsymbol{\theta}_{\hat{p}}) = \prod_{i=1}^{N} \hat{p}(\mathbf{x}_i|\boldsymbol{\theta}_{\hat{p}}). \tag{1}$$

A hill-climbing algorithm for maximizing $p(\mathscr{T}|\boldsymbol{\theta}_{\hat{p}})$ w.r.t. $\boldsymbol{\theta}_{\hat{p}}$ is obtained in two steps. (1) *Initialization*: start with some initial, e.g., random, assignment of values to the RBF parameters.[1] (2) *Gradient-ascent*: repeatedly apply a learning rule in the form $\Delta\boldsymbol{\theta}_{\hat{p}} = \eta\nabla_{\theta_{\hat{p}}}\{\prod_{i=1}^{N}\hat{p}(\mathbf{x}_i|\boldsymbol{\theta}_{\hat{p}})\}$ with $\eta \in \Re^+$. This is a batch learning rule. In practice, neural network learning may be simplified, yet even improved, with the adoption of an online training scheme that prescribes $\Delta\boldsymbol{\theta}_{\hat{p}} = \eta\nabla_{\theta_{\hat{p}}}\{\hat{p}(\mathbf{x}|\boldsymbol{\theta}_{\hat{p}})\}$ upon presentation of each individual training observation $\mathbf{x}$. The gradient is calculated w.r.t. two distinct families of adaptive parameters.

(1) Mixing parameters $c_1, \ldots, c_n$, i.e., the hidden-to-output weights of the RBF network. A constrained ML estimation process is required to ensure that $c_j \in (0, 1)$ for $j = 1, \ldots, n$, and that $\sum_{j=1}^{n} c_j = 1$. To satisfy the constraints we introduce n latent parameters $\gamma_1, \ldots, \gamma_n$, which are unconstrained, and we let

$$c_i = \frac{\varsigma(\gamma_i)}{\sum_{j=1}^{n}\varsigma(\gamma_j)}, i = 1, \ldots, n \tag{2}$$

where $\varsigma(x) = 1/(1 + e^{-x})$. Each γ_i is then treated as an unknown parameter to be estimated via ML.

(2) d-dimensional mean vector $\boldsymbol{\mu}_i$ and $d \times d$ covariance matrix Σ_i for each of the Gaussian kernels $K_i(\mathbf{x}) = G(\mathbf{x}; \boldsymbol{\mu}_i, \Sigma_i)$, $i = 1, \ldots, n$ of the RBF, where $G(\mathbf{x}; \boldsymbol{\mu}_i, \Sigma_i)$ denotes a multivariate Gaussian pdf having mean vector $\boldsymbol{\mu}_i$, covariance matrix Σ_i, and evaluated over the random vector $\mathbf{x}$. A common (yet effective) simplification is to consider diagonal covariance matrices, i.e., independence[2] among the components of the input vector $\mathbf{x}$. This assumption leads to the following three major consequences: (1) modeling properties are not affected significantly, according to [26]; (2) generalization capabilities of the overall model may turn out to be improved, since the number of free parameters is reduced; (3) i-th multivariate kernel K_i may be expressed in the form of a product of d univariate Gaussian pdfs as:

$$K_i(\mathbf{x}) = \prod_{j=1}^{d}\frac{1}{\sqrt{2\pi}\sigma_{ij}}\exp\left\{-\frac{1}{2}\left(\frac{x_j - \mu_{ij}}{\sigma_{ij}}\right)^2\right\} \tag{3}$$

i.e., the free parameters to be estimated are the means μ_{ij} and the standard deviations σ_{ij}, for each kernel $i = 1, \ldots, n$ and for each component $j = 1, \ldots, d$ of the input space. Recapitulating, the RBF system equation can thus be expressed in the concise form $\hat{p}(\mathbf{x}|\boldsymbol{\theta}_{\hat{p}}) = \sum_{i=1}^{n} c_i G(\mathbf{x}; \boldsymbol{\mu}_i, \Sigma_i)$.

An explicit form for $\Delta\boldsymbol{\theta}_{\hat{p}}$ is now obtained by calculating the partial derivatives of $\hat{p}(\mathbf{x}||\boldsymbol{\theta}_{\hat{p}})$ w.r.t. the two families of free parameters in the model. For a generic mixing parameter $c_i, i = 1, \ldots, n$, from Eq. (2) and since $\hat{p}(\mathbf{x}||\boldsymbol{\theta}_{\hat{p}}) = \sum_{k=1}^{n} c_k K_k(\mathbf{x})$ we have

[1] RBF initialization can be accomplished using more structured approaches. See, for instance, [10].

[2] More precisely, diagonal covariance matrices entail uncorrelated components of the random vector. Uncorrelatedness implies independence only when the components have a joint Gaussian distribution.

$$\begin{aligned}
\frac{\partial \hat{p}(\mathbf{x}|\boldsymbol{\theta}_{\hat{p}})}{\partial \gamma_i} &= \sum_{j=1}^{n} \frac{\partial \hat{p}(\mathbf{x}|\boldsymbol{\theta}_{\hat{p}})}{\partial c_j} \frac{\partial c_j}{\partial \gamma_i} \\
&= \sum_{j=1}^{n} K_j(\mathbf{x}) \frac{\partial}{\partial \gamma_i} \left(\frac{\varsigma(\gamma_j)}{\sum_{k=1}^{n} \varsigma(\gamma_k)} \right) \\
&= K_i(\mathbf{x}) \left\{ \frac{\varsigma'(\gamma_i) \sum_k \varsigma(\gamma_k) - \varsigma(\gamma_i)\varsigma'(\gamma_i)}{[\sum_k \varsigma(\gamma_k)]^2} \right\} + \sum_{j \neq i} K_j(\mathbf{x}) \left\{ \frac{-\varsigma(\gamma_j)\varsigma'(\gamma_i)}{[\sum_k \varsigma(\gamma_k)]^2} \right\} \\
&= K_i(\mathbf{x}) \frac{\varsigma'(\gamma_i)}{\sum_k \varsigma(\gamma_k)} - \sum_j K_j(\mathbf{x}) \frac{\varsigma(\gamma_j)\varsigma'(\gamma_i)}{[\sum_k \varsigma(\gamma_k)]^2} \\
&= K_i(\mathbf{x}) \frac{\varsigma'(\gamma_i)}{\sum_k \varsigma(\gamma_k)} - \left\{ \sum_j c_j K_j(\mathbf{x}) \right\} \frac{\varsigma'(\gamma_i)}{\sum_k \varsigma(\gamma_k)} \\
&= \frac{\varsigma'(\gamma_i)}{\sum_k \varsigma(\gamma_k)} \left\{ K_i(\mathbf{x}) - \hat{p}(\mathbf{x}||\boldsymbol{\theta}_{\hat{p}}) \right\}. \qquad (4)
\end{aligned}$$

As for the means μ_{ij} and the standard deviations σ_{ij} we proceed as follows. Let θ_{ij} denote the free parameter, i.e., μ_{ij} or σ_{ij}, to be estimated. It is seen that $\frac{\partial \hat{p}(\mathbf{x}|\boldsymbol{\theta}_{\hat{p}})}{\partial \theta_{ij}} = c_i \frac{\partial K_i(\mathbf{x})}{\partial \theta_{ij}}$, where the calculation of $\frac{\partial K_i(\mathbf{x})}{\partial \theta_{ij}}$ is accomplished as follows. First, let us observe that for any real-valued, differentiable function $f(.)$ this property holds true: $\frac{\partial f(.)}{\partial x} = f(.) \frac{\partial \log[f(.)]}{\partial x}$. Thence, from Eq. (3) we can write

$$\begin{aligned}
\frac{\partial K_i(\mathbf{x})}{\partial \theta_{ij}} &= K_i(\mathbf{x}) \frac{\partial \log K_i(\mathbf{x})}{\partial \theta_{ij}} \qquad (5) \\
&= K_i(\mathbf{x}) \frac{\partial}{\partial \theta_{ij}} \sum_{k=1}^{d} \left\{ -\frac{1}{2} \left[\log(2\pi\sigma_{ik}^2) + \left(\frac{x_k - \mu_{ik}}{\sigma_{ik}} \right)^2 \right] \right\}.
\end{aligned}$$

For the means, i.e., $\theta_{ij} = \mu_{ij}$, Eq. (5) yields $\frac{\partial K_i(\mathbf{x})}{\partial \mu_{ij}} = K_i(\mathbf{x}) \frac{x_j - \mu_{ij}}{\sigma_{ij}^2}$. For the covariances, i.e., $\theta_{ij} = \sigma_{ij}$, Eq. (5) takes the form $\frac{\partial K_i(\mathbf{x})}{\partial \sigma_{ij}} = K_i(\mathbf{x}) \frac{\partial}{\partial \sigma_{ij}} \left\{ -\frac{1}{2} \log(2\pi\sigma_{ij}^2) - \frac{1}{2} \left(\frac{x_j - \mu_{ij}}{\sigma_{ij}} \right)^2 \right\}$, that is $\frac{\partial K_i(\mathbf{x})}{\partial \sigma_{ij}} = \frac{K_i(\mathbf{x})}{\sigma_{ij}} \left\{ \left(\frac{x_j - \mu_{ij}}{\sigma_{ij}} \right)^2 - 1 \right\}$ which completes the calculation of $\Delta \boldsymbol{\theta}_{\hat{p}}$. The training algorithm is guaranteed to increase the likelihood up to a (possibly local) maximum. Due to the universal approximation properties of RBFs [4] it is seen that, under the same mild conditions assumed therein, the resulting ANN is a universal model of any continuous and bounded pdf of multivariate observations.

2.2 Estimating pdfs via Multilayer Perceptrons

As we say, RBFs realize linear combinations of Gaussian kernels. All in all, the gradient-ascent ML solution handed out in the previous section reduces to a neural implementation of the classic ML parametric estimation for a Gaussian mixture, thence it is seldom palatable.

The search for a suitable, nonparametric ANN-based pdf estimator shall rather rely on a multilayer perceptron (MLP) [4], due to its flexibility, its nice generalization capabilities, and the very nature of its activation functions (yielding a mixture of sigmoids) which pose themselves as a significant alternative to Gaussian-like kernels. Since regular, supervised BP cannot be applied, variations on the theme of MLP training in the unsupervised framework are sought.

The most straightforward idea is to apply ML gradient-ascent training to the MLP, as with RBFs. If $\mathscr{T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is the sample of the i.i.d. data drawn from the unknown pdf $p(.)$, and if we write Ω to denote the ordered set of the MLP parameters (weights and biases), then the likelihood of the model given the sample is $p(\mathscr{T} \mid \Omega) = \prod_{i=1}^{n} p(\mathbf{x}_i \mid \Omega)$, where $p(\mathbf{x}_i \mid \Omega)$ is the MLP output for input $\mathbf{x}_i$. As in the previous section, gradient ascent prescribes the modification Δw of a generic parameter $w \in \Omega$ according to $\Delta w = \eta \partial \prod_{i=1}^{n} p(\mathbf{x}_i \mid \Omega)/\partial w$, where $\eta \in \Re^+$ is the learning rate. Unfortunately, since an unconstrained mixture of sigmoids is not a pdf, training the MLP this way will only lead to an unbounded increase in the value of its weights (hence, of the integral of the function the MLP realizes), yielding a degenerate solution. This phenomenon is known as the "divergence problem" [42]. To the contrary of RBFs, where Kolmogorov's axioms of probability were satisfied by means of simple constraints on the mixing parameters, no simple workarounds are available for constraining the MLP to satisfy the "integral equals 1" requirement that any proper pdf shall meet. Alternative approaches to MLP training for pdf estimation were thus proposed in the literature.

Commonsense suggests that a solution to the divergence problem lies in controlling the growth of the integral of the function $\phi(.)$ realized by the MLP during training. Roughly speaking, the MLP output might be normalized (at the end of each training iteration) by the (approximated) integral of $\phi(.)$, computed via any numerical integration technique. The general mathematical formalization of this idea is proposed in [28]. The latter assumes a maximum log-likelihood criterion where the output activation function of the MLP is the exponential, ensuring the correct range for a pdf, namely $(0, +\infty)$. The bias of the output unit plays basically the role of the normalization factor, that is the integral of the function $\exp(\phi(.))$ realized by the MLP over a given compact set $X \subset \Re^d$. It turns out that the calculations of the gradient of the criterion w.r.t. the MLP parameters Ω involve the computation of two separate integrals (namely, $\int_X \exp(\phi(\mathbf{x}, \Omega))d\mathbf{x}$, and $\int_X \exp(\phi(\mathbf{x}, \Omega))\nabla_\Omega \phi(\mathbf{x}, \Omega)d\mathbf{x}$), on a parameter-by-parameter basis (since the latter integral requires the knowledge of $\nabla_\Omega \phi(\mathbf{x}, \Omega)$ for all—possibly backpropagated—MLP weights and biases), at each step of the BP procedure.

Albeit elegant, this approach presents a few, severe drawbacks. First of all, no algorithmic solution to the effective computation of the integrals is given in [28]. This is pivotal, since numeric integration (to be accomplished at each training iteration and for each training pattern) raises a critical trade-off (in terms of tightness of the integration grid) between numerical stability and computational complexity, especially in multivariate scenarios. Also the output exponential function may be troublesome from the numerical stability standpoint. In fact, the value of its derivatives (involved in the BP procedure) spans over a different numeric range w.r.t. the derivatives of the logistic functions used in the hidden layer. This renders the selection of a common, viable learning rate unlikely at best. Moreover, most real-world, multivariate pdfs have extremely close-to-zero values over all $\Re^d$. That is to say, the expected MLP output occurs at the far left-tail of the exp(.) function, where derivatives have (quasi-)null numerical values. Since (due to the BP mechanism) the derivatives of the output activation are involved in the computation of the gradients throughout the whole MLP, this results in null updates Δw for all the parameters in the MLP (i.e., learning simply gets stuck). These are possibly some of the reasons why the approach was validated empirically only on a very simple, univariate problem (requiring 5000 data points for training) [28]. Further theoretical developments of this technique are discussed in [29], which makes explicit the adoption of a minimum complexity estimation scheme. The analysis proposed in [29] (limited to one-hidden-layer MLPs over the unit interval in $\Re^d$), albeit mathematically valuable, is not instantiated in the form of an algorithm. In particular, the usual problem of the computation of the normalizing integral is not given any practical solution.

As a consequence, alternative approaches were proposed (e.g., [25, 44]), shifting the focus from a direct estimation of the pdf to the estimation of the corresponding cumulative distribution function (cdf). After properly training the MLP model $\phi(.)$ of the cdf, the pdf can be recovered by applying differentiation to $\phi(.)$ (i.e., indirect estimation is accomplished). The idea is sound, since cdfs usually have a (mixture of) sigmoid(s)-like form which fits the nature of the activation functions. Above all, the requirements that $\phi(.)$ has to satisfy to be interpretable as a proper cdf (namely, that it ranges between 0 and 1, and that it is monotonically nondecreasing) appear to be more easily met than the corresponding constraint on pdf models (i.e., the unit integral).

In [44] (where a kernel-based solution is presented) the empirical cdf is obtained from the data sample, and it is basically used for creating target outputs for each training pattern, such that any (differentiable) supervised regression model can then be applied. Of course the empirical cdf results in a rough, discontinuous approximation of the true cdf, but experiments presented in [44] over univariate, simple tasks involving Gaussian pdfs highlight the approach may be viable to some extent. In [25] an MLP is trained to estimate the cdf $F(x)$ of a univariate pdf $p(x)$ using a stochastic approach. Relying on the fact that if x is a random variable distributed according to $p(x)$ then the corresponding random variable $y = F(x)$ has a uniform distribution over $(0, 1)$, target outputs for the BP algorithm are generated as follows. A random sample of "targets" $\{y_1, \dots, y_n\}$ is drawn from this uniform distribution. Since cdfs are intrinsically monotonically increasing, both the original data sample $\{x_1, \ldots, x_n\}$ and the targets $\{y_1, \ldots, y_n\}$ are sorted in a nondecreasing order (s.t. $x_k \leq x_{k+1}$ and $y_k \leq y_{k+1}$ for $k = 1, \ldots, n-1$). A supervised training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ is thus obtained, and BP is applied. A penalty term is added to the squared error criterion function for BP, too, enforcing the monotonicity of the resulting ANN. Resampling of the targets is then accomplished at regular periods of t iterations of BP. The method is theoretically proofed to exhibit a noticeable convergence rate to the true distribution as the number of training patterns increases [25].

Unfortunately, there are drawbacks to the cdf-based approaches. First of all, they require differentiation of the network output $\phi(x)$ w.r.t. its input x in order to recover the estimated value of the pdf sought. This entails two consequences. First, a differentiation method has to be applied at test time, once training has been accomplished (this requires a backpropagation of partial derivatives throughout the multilayer ANN over each new test pattern), increasing complexity at test time and possibly affecting the numerical quality of the results. Second (and much more troubling), a good approximation of the cdf may not necessarily translate into a similarly good estimate of its derivative. In fact, a small squared error between $\phi(.)$ and $F(.)$ does not mean that $\phi(.)$ is free from steep fluctuation that imply huge, rapidly changing values of the derivative. Negative values of $\frac{\partial \phi(x)}{\partial x}$ may occasionally occur, since a linear combination of logistics is not necessarily monotonically increasing, even if the technique proposed in [25] is applied (actually, the penalty term used in [25] involves a pseudo-Lagrange multiplier which enforces monotonicity but does not guarantee it). Another problem of cdf-based algorithms is that they naturally apply to univariate cases, whilst extension to multivariate pdfs is far less realistic. For this reason, [25] proposes also a multivariate deterministic variant of the estimation technique. Targets y for BP are created by means of an empirical estimation of the multivariate cdf over $\mathbf{x} \in \Re^d$ as $y = \frac{1}{n}\sum_{i=1}^{n} g(\mathbf{x} - \mathbf{x}_i)$, where $g(x_1, \ldots, x_d) = 1$ if $x_j \geq 0$ for all $j = 1, \ldots, d$, otherwise $g(x_1, \ldots, x_d) = 0$, and where $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is the training sample. It is clear that this approximation of the cdf requires a huge number of training patterns over high-dimensional feature spaces, in order to ensure a significant coverage of $\Re^d$. This "sampling problem" is somewhat related to the computation of the normalization integral of the pdf involved in the ML approaches reviewed above. The application of differentiation techniques for retrieving the multivariate pdf sought is subject to the same drawbacks we mentioned in the univariate case, but it becomes even more troublesome in severely high-dimensional spaces (requiring approximated, numerical techniques), as pointed out in [25].

Finally, we present a neural-based algorithm for unsupervised, nonparametric density estimation that was recently introduced in [41]. The approach overcomes the limitations of statistical (either parametric or nonparametric) techniques, as well the drawbacks of the aforementioned MLP-based approaches, leading to improved pdf models. The technique is related to cdf-based methods, insofar that a nonparametric statistical model is used to create target output for backpropagation (BP) training [4]. The algorithm is introduced by reviewing the basic concepts of the traditional

Parzen window (PW) estimation [11]. Let us consider a pdf $p(\mathbf{x})$, defined over a real-valued, d-dimensional feature space. The probability that a pattern $\mathbf{x}' \in \mathscr{R}^d$, drawn from $p(\mathbf{x})$, falls in a certain region R of the feature space is $P = \int_R p(\mathbf{x})d\mathbf{x}$. Let then $\mathscr{T} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ be an unsupervised sample of n patterns, identically and independently distributed (i.i.d.) according to $p(\mathbf{x})$. If k_n patterns in $\mathscr{T}$ fall within R, an empirical estimate of P can be obtained as $P \simeq k_n/n$. If $p(\mathbf{x})$ is continuous and R is small enough to prevent $p(\mathbf{x})$ from varying its value over R in a significant manner, we are also allowed to write $\int_R p(\mathbf{x})d\mathbf{x} \simeq p(\mathbf{x}')V$, where $\mathbf{x}' \in R$, and V is the volume of region R. As a consequence of the discussion, we can obtain an estimated value of the pdf $p(\mathbf{x})$ over pattern $\mathbf{x}'$ as:

$$p(\mathbf{x}') \simeq \frac{k_n/n}{V_n} \tag{6}$$

where V_n denotes the volume of region R_n (i.e., the choice of the region width is explicitly written as a function of n), assuming that smaller regions around $\mathbf{x}'$ are considered as the sample size n increases. This is expected to allow Eq. (6) to yield improved estimates of $p(\mathbf{x})$, i.e., to converge to the exact value of $p(\mathbf{x}')$ as n (hence, also k_n) tends to infinity (a discussion of the asymptotic behavior of nonparametric models of this kind can be found in [11]).

The basic instance of the PW technique assumes that R_n is a hypercube having edge h_n, such that $V_n = h_n^d$. The edge h_n is usually defined as a function of n as $h_n = h_1/\sqrt{n}$, in order to ensure a correct asymptotic behavior. The value h_1 has to be chosen empirically, and it heavily affects the resulting model. The formalization of the idea requires to define a unit-hypercube window function in the form

$$\varphi(\mathbf{y}) = \begin{cases} 1 & \text{if } |\, \mathrm{y_j} \,| \leq 1/2, \mathrm{j} = 1, \dots, \mathrm{d} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

such that $\varphi(\frac{\mathbf{x}'-\mathbf{x}}{h_n})$ has value 1 iff $\mathbf{x}'$ falls within the d-dimensional hyper-cubic region R_n centered in $\mathbf{x}$ and having edge h_n. This implies that $k_n = \sum_{i=1}^n \varphi(\frac{\mathbf{x}'-\mathbf{x}_i}{h_n})$. Using this expression, from Eq. (6) we can write

$$p(\mathbf{x}') \simeq \frac{1}{n}\sum_{i=1}^{n} \frac{1}{V_n}\varphi(\frac{\mathbf{x}'-\mathbf{x}_i}{h_n}) \tag{8}$$

which is the PW estimate of $p(\mathbf{x}')$ from the sample $\mathscr{T}$. The model is usually refined by considering smoother window functions $\varphi(.)$, instead of hypercubes, e.g., standard Gaussian kernels with zero mean and unit covariance matrix.

Let us now consider an MLP that we wish to train in order to learn a model of the probability law $p(\mathbf{x})$ from the unsupervised data set $\mathscr{T}$. The idea is to use the PW model as a target output for the ANN, and to apply standard BP to learn the ANN connection weights. An unbiased variant of this idea is proposed, according to the following unsupervised algorithm (expressed in pseudo-code):

```
Input: $\mathscr{T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, $h_1$.
Output: $\tilde{p}(.)$ // the neural estimate of $p(.)$
1. Let $h_n = h_1/\sqrt{n}$
2. Let $V_n = h_n^d$
3. For i=1 to n do // loop over $\mathscr{T}$
3.1         Let $\mathscr{T}_i = \mathscr{T} \setminus \{\mathbf{x}_i\}$
3.2         Let $y_i = \frac{1}{n}\sum_{\mathbf{x}\in\mathscr{T}_i} \frac{1}{V_n}\varphi(\frac{\mathbf{x}_i - \mathbf{x}}{h_n})$ // target output
4. Let $\mathscr{S} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$ // supervised training set
5. Train the ANN via backpropagation over $\mathscr{S}$
6. Let $\tilde{p}(.)$ be equal to the function computed by the
ANN
7. Return $\tilde{p}(.)$
```

Since the MLP output is assumed to be an estimate of a pdf, it must be non-negative. This is granted once standard sigmoids (in the form $y = \frac{1}{1+e^{-x}}$) are used in the output layer. Standard sigmoids range in the $(0, 1)$ interval, while pdfs may take any positive value. For this reason, sigmoids with adaptive amplitude λ (i.e., in the form $y = \frac{\lambda}{1+e^{-x}}$), as described in [40], should be used. A direct alternative is using linear output activation functions, forcing negative outputs to zero once training is completed. Nevertheless, as in the popular k_n-nearest neighbor technique [11], the resulting MLP is not necessarily a pdf (in general, the integral of $\tilde{p}(.)$ over the feature space is not 1).

There are two major aspects of the algorithm that shall be clearly pointed out. First, the PW generation of target outputs (steps 3–3.2) is unbiased. Computation of the target for i-th input pattern $\mathbf{x}_i$ does not involve $\mathbf{x}_i$ in the underlying PW model. This is crucial in smoothing the local nature of PW. In practice, the target (estimated pdf value) over $\mathbf{x}_i$ is determined by the concentration of patterns in the sample (different from $\mathbf{x}_i$) that occurs in the surroundings of $\mathbf{x}_i$. In particular, if an isolated pattern (i.e., an outlier) is considered, its exclusion from the PW model turns out to yield a close-to-zero target value. This phenomenon is evident along the possible tails of certain distributions, and it is observed in the experiments described in [41].

A second relevant aspect of the algorithm is that it trains the MLP only over the locations (in the feature space) of the patterns belonging to the original sample. At first glance, a different approach could look more promising: once the PW model has been estimated from $\mathscr{T}$, generate a huge supervised training set by covering the input interval in a "uniform" manner, and by evaluating target outputs via the PW model. A more homogeneous and exhaustive coverage of the feature space would be expected, as well as a more precise ANN approximation of the PW. As a matter of fact, training the ANN this way reduces its generalization capabilities, resulting in a more "nervous" surface of the estimated pdf, since the PW model has a natural tendency to yield unreliable estimates over regions of the feature space that are not covered by the training sample (again, refer to the experimental demonstration in [41]).

It is immediately seen that, in spite of the simplicity of its training algorithm, eventually the MLP is expected to overcome most of the PW limitations listed above. The experiments reported in [41] highlight that, in addition, the MLP may turn out to be more accurate than the PW estimate.

3 From pdf Estimation to Online Neural Clustering

A connectionist approach to online unsupervised clustering relying on ML parametric techniques is represented by competitive neural networks (CNN) [19]. CNNs are one-layer feed-forward models whose dynamics forms the basis for several other unsupervised connectionist architectures. These models are called *competitive* because each unit competes with all the others for the classification of each input pattern: the latter is indeed assigned to the *winner unit*, which is the closest (according to a given distance measure) to the input itself, i.e., the most representative within a certain set of *prototypes*. This is strictly related to the concept of *clustering* [11], where each unit represents the centroid (or mean, or codeword) of one of the clusters. The propagation of the input vector through the network consists in a projection of the pattern onto the weights of the connections entering each output unit. Connection weights are assumed to represent the components of the corresponding centroid. The aim of the forward propagation is to establish the closest centroid to the input vector, i.e., the winner unit. During training, a simple weight update rule is used to move the winner centroid toward the novel pattern, so that the components of the centroid represent a sort of moving average of the input patterns belonging to the corresponding cluster. This is basically an online version of some partitioning clustering algorithms [11]. In the following, we will derive it as a consequence of maximum-likelihood estimation of the parameters of a mixture of Gaussians under certain assumptions. Of course, the calculations are related to those we carried out in Sect. 2.1 for RBF-based ML estimation of pdfs.

Let us consider the training set $\mathscr{T} = \{\mathbf{x}_k \mid k = 1, \ldots, N\}$, where N input samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ are supposed to be drawn from the *finite mixture* density

$$p(\mathbf{x} \mid \Theta) = \sum_{i=1}^{C} \Pi_i p_i(\mathbf{x} \mid \boldsymbol{\theta}_i) \tag{9}$$

where the parametric form of the component densities $p_i(), i = 1, \ldots, C$ is assumed to be known, as well as the *mixing parameters* $\Pi_1, \ldots, \Pi_C$ (*a priori* probabilities of the components) and $\Theta = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C)$ is the vector of all parameters associated with each component density. In the present setup we want to use the unlabeled training samples to estimate the parameters Θ. In classical pattern recognition this is an unsupervised parametric estimation problem. The following discussion will introduce a connectionist approach to the same problem, leading to the formulation of an unsupervised learning rule for CNN that is consistent (under certain assumptions) with the above-mentioned statistical estimation.

Assuming the samples are independently drawn from $p(\mathbf{x} \mid \Theta)$, the likelihood of the observed data $\mathscr{T}$ given a certain choice of parameters Θ can be written as:

$$p(\mathscr{T} \mid \Theta) = \prod_{j=1}^{N} p(\mathbf{x}_j \mid \Theta). \tag{10}$$

Maximum-likelihood estimation techniques search for the parameters Θ that maximize expression (10), or equivalently the *log-likelihood*, as:

$$l(\Theta) = \log p(\mathscr{T} \mid \Theta) = \sum_{j=1}^{N} \log p(\mathbf{x}_j \mid \Theta). \tag{11}$$

Since the logarithm is a monotonic increasing function, parameters that maximize expressions (10) and (11) are the same. If a certain parameter vector Θ' maximizes $l(\Theta)$ then it has to satisfy the following necessary, but not sufficient, condition:

$$\nabla_{\boldsymbol{\theta}'_i} l(\Theta') = \mathbf{0} \; i = 1, \ldots, C \tag{12}$$

where the operator $\nabla_{\boldsymbol{\theta}'_i}$ denotes the gradient vector computed with respect to the parameters $\boldsymbol{\theta}'_i$, and $\mathbf{0}$ is the vector with all components equal to zero. In other words, we are looking for the zeros of the gradient of the log-likelihood. Substituting Eq. (9) into Eq. (11) and setting the latter equal to zero we can write:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}'_i} l(\Theta') &= \sum_{j=1}^{N} \nabla_{\boldsymbol{\theta}'_i} \log p(\mathbf{x}_j \mid \Theta') \\
&= \sum_{j=1}^{N} \nabla_{\boldsymbol{\theta}'_i} \log \sum_{i=1}^{C} \Pi_i p_i(\mathbf{x}_j \mid \boldsymbol{\theta}'_i) \\
&= \sum_{j=1}^{N} \frac{1}{p(\mathbf{x}_j \mid \Theta')} \nabla_{\boldsymbol{\theta}'_i} \Pi_i p_i(\mathbf{x}_j \mid \boldsymbol{\theta}'_i) \\
&= \mathbf{0}.
\end{aligned} \tag{13}$$

Using Bayes' Theorem we have:

$$P(i \mid \mathbf{x}_j, \boldsymbol{\theta}'_i) = \frac{\Pi_i p_i(\mathbf{x}_j \mid \boldsymbol{\theta}'_i)}{p(\mathbf{x}_j \mid \Theta)} \tag{14}$$

where $P(i \mid \mathbf{x}_j, \boldsymbol{\theta}'_i)$ is the *a posteriori* probability of class i given the observation $\mathbf{x}_j$ and the parameters $\boldsymbol{\theta}'_i$. Equation (13) can thus be rewritten as:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}'_i} l(\Theta') &= \sum_{j=1}^{N} \frac{P(i \mid \mathbf{x}_j, \boldsymbol{\theta}'_i)}{\Pi_i p_i(\mathbf{x}_j \mid \boldsymbol{\theta}'_i)} \nabla_{\boldsymbol{\theta}'_i} \Pi_i p_i(\mathbf{x}_j \mid \boldsymbol{\theta}'_i) \\ &= \sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \boldsymbol{\theta}'_i) \nabla_{\boldsymbol{\theta}'_i} \log \Pi_i p_i(\mathbf{x}_j \mid \boldsymbol{\theta}'_i) \\ &= \mathbf{0}. \end{aligned} \tag{15}$$

In the following, we will concentrate our attention on the case in which the component densities of the mixture are multivariate normal distributions. In this case

$$p_i(\mathbf{x}_j \mid \boldsymbol{\theta}_i) = (2\pi)^{-\frac{d}{2}} \mid \Sigma_i \mid^{-\frac{1}{2}} e^{\{-\frac{1}{2}(\mathbf{x}_j - \boldsymbol{\mu}_i)^t \Sigma_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i)\}}, \tag{16}$$

where d is the dimension of the feature space, t denotes the transposition of a matrix, and the parameters to be estimated for the i-th component density are its mean vector and its covariance matrix, that is to say:

$$\boldsymbol{\theta}_i = (\boldsymbol{\mu}_i, \Sigma_i). \tag{17}$$

Substituting Eq. (16) into Eq. (15) we can write the gradient of the log-likelihood for the case of normal component densities as:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_i} l(\Theta) = \sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \boldsymbol{\theta}_i) \nabla_{\boldsymbol{\theta}_i} \{ &\log \Pi_i (2\pi)^{-\frac{d}{2}} \mid \Sigma_i \mid^{-\frac{1}{2}} \\ &-\frac{1}{2} (\mathbf{x}_j - \boldsymbol{\mu}_i)^t \Sigma_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i) \}. \end{aligned} \tag{18}$$

Suppose now that the only unknown parameters to be estimated are the mean vectors of the Gaussian distributions, e.g., the covariances are supposed to be known in advance. There are practical situations, for example, in data clustering, in which the estimation of the means is sufficient; furthermore, this assumption simplifies the following mathematics, but the extension to the more general case of unknown covariances is rather simple. By setting $\Theta = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_C)$, Eq. (18) reduces to:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_i} l(\Theta) &= \sum_{j=1}^{N} \{ P(i \mid \mathbf{x}_j, \boldsymbol{\mu}_i) \\ &\quad \times \nabla_{\boldsymbol{\mu}_i} [-\frac{1}{2} (\mathbf{x}_j - \boldsymbol{\mu}_i)^t \Sigma_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i)] \} \\ &= \sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \boldsymbol{\mu}_i) \Sigma_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i). \end{aligned} \tag{19}$$

Again, we are looking for the parameters $\Theta' = (\boldsymbol{\mu}'_1, \ldots, \boldsymbol{\mu}'_C)$ that maximize the log-likelihood, i.e., that correspond to a zero of its gradient. From Eq˙ (20), setting $\nabla_{\boldsymbol{\theta}_i} l(\Theta') = \mathbf{0}$ allows us to write:

$$\sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \boldsymbol{\mu}'_i)\mathbf{x}_j = \sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \boldsymbol{\mu}'_i)\boldsymbol{\mu}'_i \tag{20}$$

which finally leads to the following central equation:

$$\boldsymbol{\mu}'_i = \frac{\sum_{j=1}^{N} P(i|\mathbf{x}_j, \boldsymbol{\mu}'_i)\mathbf{x}_j}{\sum_{j=1}^{N} P(i|\mathbf{x}_j, \boldsymbol{\mu}'_i)} \tag{21}$$

which shows that the maximum-likelihood estimate for the i-th mean vector is a weighted average of the samples of the training set, where each sample gives a contribution that is proportional to the *estimated* probability of i-th class given the sample itself. Equation (21) cannot be explicitly solved, but it can be written in quite an interesting and practical iterative form by making explicit the dependence of the current estimate on the number t of iterative steps that have already been accomplished:

$$\boldsymbol{\mu}'_i(t+1) = \frac{\sum_{j=1}^{N} P(i|\mathbf{x}_j, \boldsymbol{\mu}'_i(t))\mathbf{x}_j}{\sum_{j=1}^{N} P(i|\mathbf{x}_j, \boldsymbol{\mu}'_i(t))} \tag{22}$$

where $\boldsymbol{\mu}'_i(t)$ denotes the estimate obtained at t-th iterative step, and the corresponding value is actually used to compute the new estimate at $(t+1)$-th step. Considering the fact that $P(i|\mathbf{x}_j, \boldsymbol{\mu}'_i(t))$ is large when the Mahalanobis distance between $\mathbf{x}_j$ and $\boldsymbol{\mu}'_i(t)$ is small, it is reasonable to estimate an approximation of $P(i|\mathbf{x}_j, \boldsymbol{\mu}'_i(t))$ in the following way:

$$P(i|\mathbf{x}_j, \boldsymbol{\mu}'_i(t)) \approx \begin{cases} 1 \text{ if } \mathrm{dist}(\mathbf{x}_j, \boldsymbol{\mu}'_i(t)) = \min_{l=1,\ldots,C} dist(\mathbf{x}_j, \boldsymbol{\mu}'_l(t)) \\ 0 \text{ otherwise} \end{cases} \tag{23}$$

for a given distance measure dist() (Mahalanobis distance should be used, but Euclidean distance is usually an effective choice), so expression (22) reduces to:

$$\boldsymbol{\mu}'_i(t+1) = \frac{1}{n_i(t)} \sum_{k=1}^{n_i(t)} \mathbf{x}_k^{(i)} \tag{24}$$

where $n_i(t)$ is the number of training patterns estimated to belong to class i at step t, i.e., the number of patterns for which $P(i|\mathbf{x}_j, \boldsymbol{\mu}'_i(t)) = 1$ according to Eq. (23), and $\mathbf{x}_k^{(i)}, k = 1, \ldots, n_i(t)$ is the k-th of these patterns. Note that Eq. (24) is the *k-means* clustering algorithm [11]. To switch to an incremental form for Eq. (24), consider what happens when, after the i-th mean vector has been calculated at step t over $n_i(t)$ patterns, a new pattern $\mathbf{x}^{(i)}_{n_i(t)+1}$ has to be assigned to component i. This occurs when $P(i|\mathbf{x}^{(i)}_{n_i(t)+1}, \boldsymbol{\mu}'_i(t))$ given by relation (23) equals 1, while $P(i|\mathbf{x}^{(i)}_{n_i(t)+1}, \boldsymbol{\mu}'_i(t-1))$ is 0. According to Eq. (24), the new mean vector is:

$$\begin{aligned}\boldsymbol{\mu}'_i(t+1) &= \frac{1}{n_i(t)+1}\left(\sum_{k=1}^{n_i(t)} \mathbf{x}^{(i)}_k + \mathbf{x}^{(i)}_{n_i(t)+1}\right)\\ &= \frac{n_i(t)}{n_i(t)+1}\boldsymbol{\mu}'_i(t) + \frac{1}{n_i(t)+1}\mathbf{x}^{(i)}_{n_i(t)+1}\\ &= \boldsymbol{\mu}'_i(t) + \frac{1}{n_i(t)+1}(\mathbf{x}^{(i)}_{n_i(t)+1} - \boldsymbol{\mu}'_i(t)). \end{aligned} \tag{25}$$

We can thus estimate the new mean vector after presentation of the $n_i + 1$-th pattern in the following way:

$$\boldsymbol{\mu}'_i(t+1) = \boldsymbol{\mu}'_i(t) + \delta_{t+1}(\mathbf{x}^{(i)}_{n_i(t)+1} - \boldsymbol{\mu}'_i(t)) \tag{26}$$

where $\delta_{t+1} = \frac{1}{n_i(t)+1}$ is a vanishing quantity. Using a fixed, small value η instead of δ_{t+1} and representing each component of the i-th mean vector $\boldsymbol{\mu}_i = (\mu_{i1}, \ldots, \mu_{id})$ with the weights $w_{i1}, \ldots, w_{id}$ of the corresponding connections in the competitive neural network model, the following weight update (learning) rule is obtained:

$$\Delta w_{ij} = \eta(x_j - w_{ij}) \tag{27}$$

where w_{ij} is the weight of the connection between input unit j and output unit i (the winner unit), η is the learning rate, and x_j is the j-th component of the pattern, of class i, presented to the network. When a new pattern is fed into the network, its distance is computed with respect to all the output units—using the connection weights as components of the corresponding mean vectors—and, according to relation (23), it is assigned to the nearest unit (mixture component). The latter is referred to as the *winner* unit. Weight update, or *learning*, is then accomplished by modifying the connection weights of the winner unit by a direct application of Eq. (27). The weights of the other units are left unchanged. This is a typical *winner-take-all* approach, where the units are in competition for novel input patterns. This is the rationale for using the name CNN. The way used here to derive the learning rule makes it clear that CNN can be seen as an online version of the popular *k-means* clustering algorithm [11].

The same mathematical formulation can be extended to the case of unknown covariance matrices for the component densities of the mixture, and even for the case of unknown class prior probabilities (the mixing parameters). A very good review is presented in [11]. The extension provides us iterative formulas similar to

Eq. (22), for estimating the means and the other unknown parameters. This means that an online (incremental) connectionist version of these estimates could also be derived. Finally, in addition to their online ever-adapting mechanism, CNNs differ from traditional clustering algorithms insofar that any growing–pruning algorithm for adapting the architecture of the ANN [19] can be used alongside with the aforementioned learning rule, resulting in a clustering scheme capable of discovering spontaneously a suitable number C of clusters for the data at hand (i.e., the user is no longer required to fix C arbitrarily in advance).

4 Maximum-Likelihood Modeling of Sequences

In several relevant applications (e.g., automatic speech recognition, handwritten text recognition, bioinformatics) the patterns to be classified are not static vectors described in a real-valued feature space, but sequences Y of observations, $Y = \mathbf{y}_1, \ldots, \mathbf{y}_T$. These sequences may have a dramatic length (from a few hundreds to millions), which may be different from sequence to sequence. Furthermore, it may happen that an input sequence is made up of subsequences (where boundaries between pairs of adjacent subsequences are not known in advance) which may be drawn individually from different probability distributions. Modeling of these probabilistic distributions is therefore of the utmost relevance to the aforementioned applications, and it is the subject of the following sections.

4.1 Motivation: Beyond Hidden Markov Models and Recurrent ANNs

Let $p(Y)$ denote the overall probability density of any given random observation sequence Y. Roughly speaking, $p(.)$ describes the statistics of sequences of parameterized observations (random "feature vectors") in the feature space. HMM [20, 36] are far the most popular (parametric) model for probability densities defined over sequences (note that a review of HMMs in out of the scope of the present chapter: the interested reader is invited to refer to [36]). Although HMMs are effective approaches, allowing for good modeling under many circumstances, they suffer from some limitations too (see [7]). For instance, the classical HMMs rely on strong assumptions on the statistical properties of the phenomenon at hand: the stochastic processes involved are modeled by first-order Markov chains, and the parametric form of the pdf that represent the emission probabilities associated with all states is heavily constraining. In addition, correlations among the input features are lost in practical implementations, since the covariance matrix of the Gaussian components (used to model the emission probabilities [36]) is usually assumed to be diagonal (in order to reduce the complexity of the machine, and to reduce numerical stability problems). Finally, HMMs are basically memory-based models (a Gaussian component is placed over dense locations of the training data) which is unlikely to generalize—due also to the lack of any regularization techniques. Reduced generalization capabilities are particularly evident when the HMMs are applied to noisy tasks (e.g., speech processing under severe noise conditions).

For these reasons, starting from the late 1980s many researchers began to use ANNs for sequence processing, namely time-delay ANNs [19, 45, 46] and, more recently, recurrent neural networks (RNN). RNNs provide a powerful extension of feed-forward connectionist models by allowing to introduce connections between arbitrary pairs of units, independently from their position within the topology of the network. Self-recurrent loops of a unit onto itself, as well as backward connections to previous layers, or lateral links between units belonging to the same layer are all allowed. RNNs behave like dynamical systems. Once fed with an input, the recurrent connections are responsible for an evolution in time of the internal state of the network. RNNs are particularly suited for sequence processing, due to their ability to keep an internal trace, or memory, of the past. This memory is combined with the current input to provide a context-dependent output. Several RNN architectures were proposed in literature [12, 21, 31], along with a variety of training algorithms, mostly based on gradient-descent techniques. Among the latter ones, particularly remarkable are Recurrent Back Propagation [34], Back-Propagation for Sequences (BPS) [16], Real Time Recurrent Learning [48, 49], Time-dependent Recurrent Back-Propagation [33, 38, 47], and the most popular *Back-Propagation Through Time* [27, 37].

In spite of their ability to classify short-time sequences, ANNs have mostly failed, so far, as a general framework for sequence processing. This is mainly due to the lack of ability to model long-term dependencies in RNNs. The theoretical motivations underlying this problem were well analyzed by Bengio et al. [1]. In the early 1990s this fact led to the idea of combining HMMs and ANNs within a single, novel model, broadly known as *hybrid HMM/ANN* [3, 8, 13, 17, 24, 30, 32, 39]. This direction turned out to be effective, exploiting the long-term modeling capabilities of HMMs and the universal, nonparametric nature of ANNs. The most popular ANN/HMM hybrid system is the classic Bourlard and Morgan's paradigm [6–8, 30], based on a maximum-a-posteriori HMM framework where an MLP is applied in order to estimate the posterior probability of the HMM states. It is a discriminative, supervised learning machine. A fully generative, unsupervised ANN/HMM hybrid approach was later proposed in [42], where the ANN outputs are expected to model the emission probabilities [36] (i.e., pdfs) of the HMM within a maximum-likelihood global optimization scheme. It is reviewed in the next section.

4.2 Unsupervised ML Learning in Generative ANN/HMM Hybrids

The approach revolves around the idea of preserving Bourlard and Morgan's architecture (an MLP having an output unit for each state of an underlying HMM), along with a novel ML training algorithm which may jointly optimize the parameters of the ANN and of the HMM. In so doing, the ANN is expected to model pdfs (the emission probabilities) instead of posteriors. As we observed in Sect. 2, this is a nontrivial task and requires ad-hoc techniques. The model described in this section was introduced in [42] first.

The machine relies on an HMM topology, including standard *initial* probabilities $\boldsymbol{\pi}$ and *transition* probabilities $\mathbf{a} = [a_{ij}]$ estimated by means of the *Baum–Welch* algorithm [36], while the *emission* probabilities $\mathbf{b}(\mathbf{y})$ [36] are estimated by an ANN. An output unit of the ANN holds for each of the states in the HMM, with the understanding that i-th output value $o_i(t)$ represents the emission probability $b_{i,t}$ for the corresponding (i-th) state, evaluated over current observation $\mathbf{y}_t$. In the following, we refer to this ANN with the symbol ψ. Once trained, the machine can be applied to new (e.g., test) sequences by relying on the usual Viterbi algorithm [20, 36]. Learning rules for connection weights and for neuron biases are calculated according to gradient-ascent to maximize a global criterion function, namely the *likelihood* of the model given the observation sequences (ML criterion). The global criterion function to be maximized by the model during training is the *likelihood* L of the observations given the model, that is[3] [2, 35]

$$L = \sum_{i \in \mathscr{F}} \alpha_{i,T}. \tag{28}$$

The sum is extended to the set $\mathscr{F}$ of all possible *final* states [2] within the HMM corresponding to the current training sequence. This HMM is supposed to involve Q states, and T is the length of the current observation sequence $Y = \mathbf{y}_1, \ldots, \mathbf{y}_T$. The *forward* terms $\alpha_{i,t} = P(q_{i,t}, \mathbf{y}_1, \ldots, \mathbf{y}_t)$ and the *backward* terms $\beta_{i,t} = P(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T | q_{i,t})$ for state i at time t can be computed recursively as follows [35]:

$$\alpha_{i,t} = b_{i,t} \sum_j a_{ji} \alpha_{j,t-1} \tag{29}$$

and

$$\beta_{i,t} = \sum_j b_{j,t+1} a_{ij} \beta_{j,t+1} \tag{30}$$

[3]A standard notation is used in the following to refer to quantities involved in HMM training (see [36]).

where a_{ij} denotes the transition probability from i-th state to j-th state, $b_{i,t}$ denotes emission probability associated with i-th state over t-th observation $\mathbf{y}_t$, and the sums are extended to all possible states within the HMM. The initialization of the forward probabilities is accomplished as in standard HMMs [35], whereas the backward terms at time T are initialized in a slightly different manner, namely:

$$\beta_{i,T} = \begin{cases} 1 \text{ if } i \in \mathscr{F} \\ 0 \text{ otherwise.} \end{cases} \tag{31}$$

Assuming that an ANN may represent the emission pdfs in a proper manner, and given a generic weight w of the ANN, hill-climbing gradient-ascent over C prescribes a learning rule of the kind:

$$\Delta w = \eta \frac{\partial L}{\partial w} \tag{32}$$

where η is the *learning rate*. In the following, we derive a learning rule for the ANN aimed at maximizing the likelihood of the observations, which is the usual criterion adopted to train HMMs. Eventually, the learning rule can be applied in parallel with the Baum–Welch algorithm, limiting the latter to the ML estimation of those quantities (namely the initial and transition probabilities in the underlying HMM) that do not explicitly depend on the ANN weights.

Let us observe, after [2], that the following property can be easily shown to hold true by straightforwardly taking the partial derivatives of the left- and right-hand sides of Eq. (29) with respect to $b_{i,t}$:

$$\frac{\partial \alpha_{i,t}}{\partial b_{i,t}} = \frac{\alpha_{i,t}}{b_{i,t}}. \tag{33}$$

In addition, borrowing the scheme proposed by Bridle [9] and Bengio [2], the following theorem can be proved to hold true (see [42]): $\frac{\partial L}{\partial \alpha_{i,t}} = \beta_{i,t}$, for each $i = 1, \dots, Q$ and for each $t = 1, \dots, T$. Given this theorem and Eq. (33), repeatedly applying the chain rule we can expand $\frac{\partial L}{\partial w}$ by writing:

$$\begin{aligned} \frac{\partial L}{\partial w} &= \sum_i \sum_t \frac{\partial L}{\partial b_{i,t}} \frac{\partial b_{i,t}}{\partial w} \\ &= \sum_i \sum_t \frac{\partial L}{\partial \alpha_{i,t}} \frac{\partial \alpha_{i,t}}{\partial b_{i,t}} \frac{\partial b_{i,t}}{\partial w} \\ &= \sum_i \sum_t \beta_{i,t} \frac{\alpha_{i,t}}{b_{i,t}} \frac{\partial b_{i,t}}{\partial w} \end{aligned} \tag{34}$$

where the sums are extended over all states $i = 1, \dots, Q$ of the HMM, and to all $t = 1, \dots, T$, respectively. From now on, attention is focused on the calculation of $\frac{\partial b_{i,t}}{\partial w}$, where $b_{i,t}$ is the output from i-th unit of the ANN at time t. Let us consider a multilayer feed-forward network (e.g., an MLP) the j-th output of which, computed over t-th input observation $\mathbf{y}_t$, is expected to be a nonparametric estimate of the emission probability $b_{j,t}$ associated with j-th state of the HMM at time t. An activation function $f_j(x_j(t))$, either linear or nonlinear, is attached to each unit j of the ANN, where $x_j(t)$ denotes input to the unit itself at time t. The corresponding output $o_j(t)$ is given by $o_j(t) = f_j(x_j(t))$. The net is assumed to have n layers $\mathscr{L}_0, \mathscr{L}_1, \dots, \mathscr{L}_n$, where $\mathscr{L}_0$ is the input layer and is not counted, and $\mathscr{L}_n$ is the output layer. For notational convenience we write $i \in \mathscr{L}_k$ to denote the index of i-th unit in layer $\mathscr{L}_k$.

When considering the case of a generic weight w_{jk} between the k-th unit in layer $\mathscr{L}_{n-1}$ and j-th unit in the output layer, we can write:

$$\begin{aligned}\frac{\partial b_{j,t}}{\partial w_{jk}} &= \frac{\partial f_j(x_j(t))}{\partial w_{jk}} \\ &= f_j'(x_j(t)) \frac{\partial \sum_{i \in \mathscr{L}_{n-1}} w_{ji} o_i(t)}{\partial w_{jk}} \\ &= f_j'(x_j(t)) o_k(t). \end{aligned} \tag{35}$$

By defining the quantity[4]

$$\delta_i(j,t) = \begin{cases} f_j'(x_j(t)) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{36}$$

for each $i \in \mathscr{L}_n$, we can rewrite Eq. (35) in the compact form:

$$\frac{\partial b_{j,t}}{\partial w_{jk}} = \delta_j(j,t) o_k(t). \tag{37}$$

Consider now the case of connection weights in the first hidden layer. Let w_{kl} be a generic weight between l-th unit in layer $\mathscr{L}_{n-2}$ and k-th unit in layer $\mathscr{L}_{n-1}$.

$$\begin{aligned}\frac{\partial b_{j,t}}{\partial w_{kl}} &= \frac{\partial f_j(x_j(t))}{\partial w_{kl}} \\ &= f_j'(x_j(t)) \sum_{i \in \mathscr{L}_{n-1}} \frac{\partial w_{ji} o_i(t)}{\partial w_{kl}} \end{aligned}$$

[4]Dependency on the specific HMM state j under consideration and on current time t is written explicitly, since this will turn out to be useful in the final formulation of the learning rule.

$$
\begin{aligned}
&= f_j'(x_j(t))\frac{\partial w_{jk}o_k(t)}{\partial o_k(t)}\frac{\partial o_k(t)}{\partial w_{kl}}\\
&= f_j'(x_j(t))w_{jk}\frac{\partial f_k(x_k(t))}{\partial w_{kl}}\\
&= f_j'(x_j(t))w_{jk}f_k'(x_k(t))\frac{\partial \sum_{i\in\mathscr{L}_{n-2}} w_{ki}o_i(t)}{\partial w_{kl}}\\
&= f_j'(x_j(t))w_{jk}f_k'(x_k(t))o_l(t).
\end{aligned}
\tag{38}
$$

Similarly to definition (36), we introduce the quantity:

$$\delta_k(j,t) = f_k'(x_k(t))\sum_{i\in\mathscr{L}_n} w_{ik}\delta_i(j,t) \tag{39}$$

for each $k \in \mathscr{L}_{n-1}$, which allows us to rewrite Eq. (38) in the form:

$$\frac{\partial b_{j,t}}{\partial w_{kl}} = \delta_k(j,t)o_l(t) \tag{40}$$

which is formally identical to Eq. (37).

Finally, we carry out the calculations for a generic, hypothetic weight w_{lm} between m-th unit in layer $\mathscr{L}_{n-3}$ and l-th unit in layer $\mathscr{L}_{n-2}$, but the same results will also apply to subsequent layers ($\mathscr{L}_{n-4}, \mathscr{L}_{n-5}, \ldots, \mathscr{L}_0$):

$$
\begin{aligned}
\frac{\partial b_{j,t}}{\partial w_{lm}} &= \frac{\partial f_j(x_j(t))}{\partial w_{lm}}\\
&= f_j'(x_j(t))\sum_{i\in\mathscr{L}_{n-1}}\frac{\partial w_{ji}o_i(t)}{\partial w_{lm}}\\
&= f_j'(x_j(t))\sum_{i\in\mathscr{L}_{n-1}}\frac{\partial w_{ji}o_i(t)}{\partial o_i(t)}\frac{\partial o_i(t)}{\partial w_{lm}}\\
&= f_j'(x_j(t))\sum_{i\in\mathscr{L}_{n-1}} w_{ji}\frac{\partial o_i(t)}{\partial w_{lm}}.
\end{aligned}
\tag{41}
$$

Efforts now concentrate on the development of the term $\frac{\partial o_i(t)}{\partial w_{lm}}$. This is accomplished by writing:

$$
\begin{aligned}
\frac{\partial o_i(t)}{\partial w_{lm}} &= \frac{\partial f_i(x_i(t))}{\partial w_{lm}}\\
&= f_i'(x_i(t))\sum_{k\in\mathscr{L}_{n-2}}\frac{\partial w_{ik}o_k(t)}{\partial w_{lm}}
\end{aligned}
$$

$$
\begin{aligned}
&= f_i'(x_i(t)) \frac{\partial w_{il} o_l(t)}{\partial o_l(t)} \frac{\partial o_l(t)}{\partial w_{lm}} \\
&= f_i'(x_i(t)) w_{il} \frac{\partial f_l(x_l(t))}{\partial w_{lm}} \\
&= f_i'(x_i(t)) w_{il} f_l'(x_l(t)) \frac{\partial x_l(t)}{\partial w_{lm}} \\
&= f_i'(x_i(t)) w_{il} f_l'(x_l(t)) \sum_{k \in \mathscr{L}_{n-3}} \frac{\partial w_{lk} o_k(t)}{\partial w_{lm}} \\
&= f_i'(x_i(t)) w_{il} f_l'(x_l(t)) o_m(t) \qquad (42)
\end{aligned}
$$

which can be substituted into Eq. (41) obtaining:

$$\frac{\partial b_{j,t}}{\partial w_{lm}} = f_j'(x_j(t)) \sum_{i \in \mathscr{L}_{n-1}} w_{ji} f_i'(x_i(t)) w_{il} f_l'(x_l(t)) o_m(t). \qquad (43)$$

Again, as in (36) and (39), for each $l \in \mathscr{L}_{n-2}$ we define the quantity[5]:

$$\delta_l(j,t) = f_l'(x_l(t)) \sum_{i \in \mathscr{L}_{n-1}} w_{il} \delta_i(j,t) \qquad (44)$$

and rewrite Eq. (43) in the familiar, compact form:

$$\frac{\partial b_{j,t}}{\partial w_{lm}} = \delta_l(j,t) o_m(t) \qquad (45)$$

that is formally identical to Eqs. (37) and (40).

Using the above developments to expand Eq. (34) and substituting it into Eq. (32), the latter can now be restated in the form of a general learning rule for a generic weight w_{jk} of the network, by writing:

$$\Delta w_{jk} = \eta \sum_{i=1}^{Q} \sum_{t=1}^{T} \beta_{i,t} \frac{\alpha_{i,t}}{b_{i,t}} \delta_j(i,t) o_k(t) \qquad (46)$$

where the term $\delta_j(i,t)$ is computed via Eq. (36), Eqs. (39) or (44), according to the fact that the layer under consideration is the output layer, the first hidden layer, or one of the other hidden layers, respectively.

Experimental results reported in [42] (where several workaround are presented to prevent occurrences of the divergence problem) show a dramatic improvement in

[5]For lower layers the sum in Eq. (44) must be accomplished over the units belonging to the immediately upper layer.

terms of sequence pdf modeling capability over traditional HMMs. The improvement is particularly significant in sequence processing tasks which involve noisy data [43]. This is likely to be due to: (1) the flexibility of the nonparametric modeling over the parametric assumptions made in standard HMM; and (2) the generalization capabilities of ANNs, in contrast with the generalization problems of traditional HMMs.

5 Conclusions

Traditionally, unsupervised learning in ANNs revolves around methods that do not take the probabilistic nature of phenomena into explicit account. Examples are the auto-associative MLP, Hopfield network, Kohonen's self-organizing maps, etc. [19]. To the contrary, the chapter focuses on learning machines that undergo a proper probabilistic interpretation, and whose training algorithms exploit (and respect) the probability laws that underlie the data samples at hand. In so doing, estimates of probabilistic quantities (that are typically involved in statistical analysis, pattern classification, and clustering) are obtained, providing the practitioner with robust tools, alternative to statistical approaches. The flexibility and the generalization capabilities of ANNs yield less constrained, more general models of pdfs than traditional statistical estimates do. Indeed, empirical evidence reported in the literature shows that ANN-based estimates may improve performance significantly over their statistical counterparts. As well, traditional parametric estimation methods may be used to obtain unsupervised learning rules for ANNs that result in an online, ever-adaptive version of popular partitive clustering algorithms, or that can learn the emission probabilities of an underlying HMM.

In most cases, the algorithms exploit the maximum-likelihood criterion, which replaces the minimum squared error (MSE) criterion typical of standard backpropagation. ML forms the basis for completely unsupervised training algorithms, suitable for RBF-based pdf estimation, CNN-based clustering, and ANN/HMM-based estimation of the probability density of sequences of random vectors. Nonetheless, the community resorted also to regular BP over the MSE loss functional for tackling the divergence problem that may occur in ML training of unconstrained MLPs. To this end, synthetic target outputs for BP are generated automatically from the unsupervised training data by relying on statistical estimates of pdfs or cdfs.

Our expectations for future scenarios embrace the following issues. First and foremost, an exact solution to the divergence problem via an algorithmically constrained MLP that may well undergo ML training. Then, extension of the ANN-based probabilistic models for HMMs to broader families of probabilistic graphical models that operate under a Markovian assumption (e.g., Markov random fields [22], conditional random fields [23], hybrid random fields [14], and dynamic random fields [5]). Finally, just like hybrid statistical models can be learned from samples of sequential data, robust neural estimates of probabilistic laws underlying structured data (like random graphs [15]) are sought.

References

1. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994). Special issue on Recurrent Neural Networks, March 94
2. Bengio, Y.: Neural Networks for Speech and Sequence Recognition. International Thomson Computer, London (1996)
3. Bengio, Y., De Mori, R., Flammia, G., Kompe, R.: Global optimization of a neural network-hidden Markov model hybrid. IEEE Trans. Neural Netw. **3**(2), 252–259 (1992)
4. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1995)
5. Bongini, M., Trentin, E.: Towards a novel probabilistic graphical model of sequential data: a solution to the problem of structure learning and an empirical evaluation. In: Artificial Neural Networks in Pattern Recognition - 5th INNS IAPR TC 3 GIRPR Workshop, ANNPR 2012, pp. 82–92. Trento, Italy, September 17–19, 2012. Proceedings (2012)
6. Bourlard, H., Morgan, N.: Continuous speech recognition by connectionist statistical methods. IEEE Trans. Neural Netw. **4**(6), 893–909 (1993)
7. Bourlard, H., Morgan, N.: Connectionist Speech Recognition. A Hybrid Approach. The Kluwer International Series in Engineering and Computer Science, vol. 247. Kluwer Academic, Boston (1994)
8. Bourlard, H., Wellekens, C.: Links between hidden Markov models and multilayer perceptrons. IEEE Trans. Pattern Anal. Mach. Intell. **12**, 1167–1178 (1990)
9. Bridle, J.: Alphanets: a recurrent 'neural' network architecture with a hidden Markov model interpretation. Speech Comm. **9**(1), 83–92 (1990)
10. Celebi, M.E., Kingravi, H.A., Vela, P.A.: A comparative study of efficient initialization methods for the k-means clustering algorithm. Expert Syst. Appl. **40**(1), 200–210 (2013)
11. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis. Wiley, New York (1973)
12. Elman, J.: Finding structure in time. Cogn. Sci. **14**, 179–211 (1990)
13. Franzini, M., Lee, K., Waibel, A.: Connectionist Viterbi training: a new hybrid method for continuous speech recognition. In: International Conference on Acoustics, Speech and Signal Processing, pp. 425–428. Albuquerque, NM (1990)
14. Freno, A., Trentin, E.: Hybrid Random Fields – A Scalable Approach to Structure and Parameter Learning in Probabilistic Graphical Models. Springer, Berlin (2011)
15. Gilbert, E.N.: Random graphs. Ann. Math. Stat. **30**(4), 1141–1144 (1959)
16. Gori, M., Bengio, Y., De Mori, R.: BPS: a learning algorithm for capturing the dynamical nature of speech. In: Proceedings of the International Joint Conference on Neural Networks, pp. 643–644. IEEE, New York/Washington D.C. (1989)
17. Haffner, P., Franzini, M., Waibel, A.: Integrating time alignment and neural networks for high performance continuous speech recognition. In: International Conference on Acoustics, Speech and Signal Processing, pp. 105–108. Toronto (1991)
18. Haykin, S.: Neural Networks (A Comprehensive Foundation). Macmillan, New York (1994)
19. Hertz, J., Krogh, A., Palmer, R.: Introduction to the Theory of Neural Computation. Addison Wesley, Massachusetts (1991)
20. Huang, X.D., Ariki, Y., Jack, M.: Hidden Markov Models for Speech Recognition. Edinburgh University Press, Edinburgh (1990)

21. Jordan, M.: Serial order: A parallel, distributed processing approach. In: Elman, J., Rumelhart, D. (eds.) Advances in Connectionist Theory: Speech. Lawrence Erlbaum, Hillsdale (1989)
22. Kindermann, R., Snell, J.L.: Markov Random Fields and Their Applications. American Mathematical Society, Providence (1980)
23. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceeding of the 18th International Conference on Machine Learning, pp. 282–289. Morgan Kaufmann, San Francisco (2001)
24. Levin, E.: Word recognition using hidden control neural architecture. In: International Conference on Acoustics, Speech and Signal Processing, pp. 433–436. Albuquerque, NM (1990)
25. Magdon-Ismail, M., Atiya, A.: Density estimation and random variate generation using multilayer networks. IEEE Trans. Neural Netw. **13**(3), 497–520 (2002)
26. McLachlan, G., Basford, K. (eds.): Mixture Models: Inference and Applications to Clustering. Marcel Dekker, New York (1988)
27. Minsky, M., Papert, S.: Perceptrons. MIT, Cambridge (1969)
28. Modha, D.S., Fainman, Y.: A learning law for density estimation. IEEE Trans. Neural Netw. **5**(3), 519–23 (1994)
29. Modha, D.S., Masry, E.: Rate of convergence in density estimation using neural networks. Neural Comput. **8**, 1107–1122 (1996)
30. Morgan, N., Bourlard, H.: Continuous speech recognition using multilayer perceptrons with hidden Markov models. In: International Conference on Acoustics, Speech and Signal Processing, pp. 413–416. Albuquerque, NM (1990)
31. Mozer, M.C.: Neural net architectures for temporal sequence processing. In: Weigend, A., Gershenfeld, N. (eds.) Predicting the Future and Understanding the Past, pp. 243–264. Addison-Wesley, Redwood City (1993)
32. Niles, L., Silverman, H.: Combining hidden Markov models and neural network classifiers. In: International Conference on Acoustics, Speech and Signal Processing, pp. 417–420. Albuquerque, NM (1990)
33. Pearlmutter, B.: Learning state space trajectories in recurrent neural networks. Neural Comput. **1**, 263–269 (1989)
34. Pineda, F.: Recurrent back-propagation and the dynamical approach to adaptive neural computation. Neural Comput. **1**, 161–172 (1989)
35. Rabiner, L., Juang, B.H.: Fundamentals of Speech Recognition. Prentice Hall, Englewood Cliffs (1993)
36. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE **77**(2), 257–286 (1989)
37. Rumelhart, D., Hinton, G., Williams, R.: Learning internal representations by error propagation. In: Rumelhart, D., McClelland, J. (eds.) Parallel Distributed Processing, vol. 1, chap. 8, pp. 318–362. MIT, Cambridge (1986)
38. Sato, M.: A real time learning algorithm for recurrent analog neural networks. Biol. Cybern. **62**, 237–241 (1990)
39. Tebelskis, J., Waibel, A., Petek, B., Schmidbauer, O.: Continuous speech recognition using linked predictive networks. In: Lippman, R.P., Moody, R., Touretzky, D.S. (eds.) Advances in Neural Information Processing Systems, vol. 3, pp. 199–205. Morgan Kaufmann, San Mateo/Denver (1991)
40. Trentin, E.: Networks with trainable amplitude of activation functions. Neural Netw. **14**(4–5), 471–493 (2001)
41. Trentin, E.: Simple and effective connectionist nonparametric estimation of probability density functions. In: Artificial Neural Networks in Pattern Recognition, Second IAPR Workshop, ANNPR 2006, pp. 1–10. Ulm, Germany, August 31–September 2, 2006. Proceedings (2006)
42. Trentin, E., Gori, M.: Robust combination of neural networks and hidden Markov models for speech recognition. IEEE Trans. Neural Netw. **14**(6), 1519–1531 (2003)
43. 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)

44. Vapnik, V.N., Mukherjee, S.: Support vector method for multivariate density estimation. In: Advances in Neural Information Processing Systems, pp. 659–665. MIT, Cambridge (2000)
45. Waibel, A.: Modular construction of time-delay neural networks for speech recognition. Neural Comput. **1**, 39–46 (1989)
46. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K.: Phoneme recognition using time-delay neural networks. IEEE Trans. Acoust. Speech Signal Process. **37**, 328–339 (1989)
47. Werbos, P.: Generalization of backpropagation with application to a recurrent gas market model. Neural Netw. **1**, 339–356 (1988)
48. Williams, R., Zipser, D.: Experimental analysis of the real-time recurrent learning algorithm. Connect. Sci. **1**, 87–111 (1989)
49. Williams, R., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural Comput. **1**, 270–280 (1989)