Gaussian-mixture Neural Networks

Gaussian-mixture Neural Networks

Duccio Meconcelli and Edmondo Trentin

DIISM, Università di Siena, Siena, Italy
duccio.meconcelli@student.unisi.it,trentin@dii.unisi.it

Abstract. Density estimation is crucial to statistical pattern recognition, in both the supervised and unsupervised frameworks. It is still an open problem, due to its intrinsic difficulties and to the many shortcomings of statistical parametric and non-parametric techniques. Artificial neural networks (ANN) have long been applied to the estimation of posterior probabilities for pattern classification (a simple supervised learning task), yet only a few attempts have been made to devise ANN-based density estimation algorithms. The paper proposes a novel algorithm for training an ANN from an unlabeled data sample of patterns randomly drawn from an underlying probability density function (PDF), say $p(\cdot)$, such that the ANN learns a robust non-parametric model of $p(\cdot)$. The algorithm leverages both the generalization capabilities of ANNs and the generality of the maximum-likelihood estimates of the parameters of Gaussian mixture models. Therefore, the proposed machine is termed Gaussian-mixture Neural Network (GNN). The best selling points of the GNN lie in its simplicity and effectiveness. Preliminary experimental results are reported and analyzed that involve data samples of variable size randomly drawn from PDFs of known form, either unimodal or multimodal. The GNN favorably compares with established statistical and ANN-based estimators, scoring generally higher than its competitors over a range of evaluation metrics. The code used in the experiments is made publicly available online on GitHub.

Keywords: Density estimation · Gaussian mixture model · artificial neural network · non-parametric estimation · deep learning.

1 Introduction

The estimation of probability density functions (PDF) from unlabeled data is a fundamental problem in machine learning and plays a crucial role in various domains, including pattern recognition, anomaly detection, and clustering [4, 12]. In pattern recognition, estimating a PDF is relevant to both the supervised setup (where the evidence $p(\mathbf{x})$ and the class-conditional PDFs $p(\mathbf{x}|\omega_i)$ for each class ω_i have to be substituted into the tight-hand side of Bayes theorem in order to apply Bayes decision rule [4]) and the unsupervised setup (where the probabilistic law underlying the distribution of an unlabeled collection of observations has to be discovered [4]). Even regression problems, which involve modeling the dependences of a random vector $\mathbf{y}$ on an independent random vector $\mathbf{x}$ can be reformulated in terms of estimating a PDF, that is the conditional PDF $p(\mathbf{y}|\mathbf{x})$.

Traditional statistical approaches [4], either parametric or non-parametric have long been used by practitioners. While parametric techniques rely on the strong assumption of knowing in advance the form of the PDF to be estimated, non-parametric estimators overcome this issue but do present a number of shortcomings, mostly due to their poor generalization capabilities (since they are memory-based), their excessive burden in terms of memory requirements and computational cost over large dataset, and (to the contrary) their lack of robustness over small datasets. See [12] for a detailed analysis of the drawbacks of statistical PDF estimators.

On the other hand, artificial neural networks (ANN) excel at learning intricate nonlinear relationships but they generally lack the statistical robustness of traditional models, and may not be suited for the PDF estimation task, which is intrinsically unsupervised and poses quite a few issues to any perspective ANN-based estimator [9]. Major exceptions are represented by [6] and [5]. The former applies a maximum-likelihood (ML) learning rule for ANNs, but the algorithm relies on a non-specified numerical integration technique for the ANN and does not apply over multidimensional feature spaces. The latter trains the ANN to mimic an empirical estimate of the cumulative distribution function (CDF), retrieving the PDF by taking the derivative of the approximated CDF computed by the ANN. Unfortunately, a good estimate of the CDF does not entail a good estimate of the PDF upon differentiation (e.g., even negative outcomes may emerge). Besides, the algorithm does not apply to multivariate scenarios, as well. More recently, [12, 9, 10] introduced robust ANN-based PDF estimators whose training algorithms exploit non-parametric models suitable to multi-dimensional feature spaces. In [12] the Parzen neural network (PNN) is proposed, along with a study of its convergence and modeling capabilities, and empirical evidence is reported that proves the PNN outperforms the established approaches. In [9] a novel multi-dimensional, efficient technique is devised for computing the numerical integral of the ANN during training, such that the ANN realizes a proper PDF (i.e., having unit integral over its definition domain). The reader is referred to [12, 11] for an exhaustive survey of the literature.

This paper contributes a novel approach to unsupervised PDF estimation via ANNs, termed Gaussian-mixture neural network (GNN), whose training algorithm (presented in Section 2) exploits the ML estimation of the parameters of an underlying Gaussian mixture model (GMM) [4], benefitting from both the GMM and the ANN capabilities and overcoming the respective limitations. The GNN can be seen as a novel instance of the teacher-student learning paradigm. Note that it has nothing to do with the so-called deep neural mixture model [11]. A preliminary experimental investigation is reported on in Section 3, relying on random data samples of various size drawn from PDFs of known forms (either unimodal or multimodal). The GNN is compared with a number of established techniques, confirming the effectiveness of the approach. In fact, being simple and effective are the best selling points of the GNN. The code used in the experiments can be downloaded from GitHub [1]. Conclusions are drawn in Section 4, that pinpoints also the directions of the on-going research on GNNs.

2 The Training Algorithm

In this section we first provide a review of the required fundamentals of GMMs (Section 2.1), then we present the GNN training algorithm (Section 2.2).

2.1 Re-estimation equations for the parameters of a GMM

A GMM is a parametric model that realizes a particular multivariate mixture density function having the following form:

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \Sigma) = \sum_{k=1}^{K} P(w_k)\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k) \tag{1}$$

where $p(\boldsymbol{x}|\boldsymbol{\mu}, \Sigma)$ is the mixture density evaluated over the column vector $\boldsymbol{x}$, K is the number of multivariate Gaussian component densities in the mixture, $P(w_k)$ is the k-th mixing parameter (that is, the prior probability of k-th component PDF) under the assumption that $\sum_{k=1}^{K} P(w_k) = 1$, $\boldsymbol{\mu}_k$ is the (column) mean vector of the k-th Gaussian component and Σ_k is the corresponding covariance matrix. The quantities $\boldsymbol{\mu}$ and Σ represent the set of all the component-specific parameters, namely $\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K)$ and $\Sigma = (\Sigma_1, \ldots \Sigma_K)$, respectively. The generic k-th multivariate Gaussian component $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k)$ is given by the usual equation

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^T \Sigma_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k)\right) \tag{2}$$

where D is the dimensionality of the feature space, $|\Sigma_k|$ is the determinant of the k-th covariance matrix, Σ_k^{-1} is its inverse, and the writing $\boldsymbol{x}^T$ represents the transpose of the generic vector $\boldsymbol{x}$. The parameters of the GMM, namely $P(w_1), \ldots, P(w_K)$, $\boldsymbol{\mu}$, and Σ, are estimated from a sample $\mathcal{T} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ of N independent and identically distributed feature vectors drawn from an underlying, unknown PDF $p(\boldsymbol{x})$ that the GMM is expected to estimate relying on the ML criterion by means of the following iterative steepest ascent algorithm (an instance of the expectation-maximization strategy [3]):

1. At time $t = 0$, initialize the GMM parameters, usually via k-Means clustering [4, 7]
2. For $t = 1, 2, \ldots$ do
   (a) Update the mixing parameters:

$$P^{(t+1)}(w_k) = \frac{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t)}, \Sigma^{(t)})}{N} \tag{3}$$

   (b) Update the mean vectors:

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t)}, \Sigma^{(t)})\mathbf{x}_i}{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t)}, \Sigma^{(t)})} \tag{4}$$

(c) Update the covariance matrices:

$$\Sigma_k^{(t+1)} = \frac{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t+1)}, \Sigma^{(t)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})^T}{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t+1)}, \Sigma^{(t)})} \quad (5)$$

At any given iteration t, the posterior probability $P(w_k|\boldsymbol{x}, \boldsymbol{\mu}^{(t)}, \Sigma^{(t)})$ of the component density w_k given the observation $\boldsymbol{x}$ (and, given the estimate of the GMM parameters at time t) is computed via Bayes' theorem [4].

Algorithm 1 `GNN_Training`.

Input: randomly initialized network *net* to be trained, number K of Gaussian components, number T_{EM} of EM iterations, unlabeled training sample $\mathcal{T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ drawn from an unknown pdf $p(\cdot)$, dimensionality D of the feature space.
Output: the neural estimate $\phi(\cdot)$ of the pdf $p(\cdot)$.

1 Appply k-Means to partition $\mathcal{T}$ into K clusters $C_1, \ldots, C_K$

2 for(k = 1 to K)
2.1 Use the statistics of C_k to initialize $P^{(0)}(w_k), \boldsymbol{\mu}_k^{(0)}, \Sigma_k^{(0)}$

3 $\boldsymbol{\mu}^{(0)} = (\boldsymbol{\mu}_1^{(0)}, \ldots, \boldsymbol{\mu}_K^{(0)})$
4 $\Sigma^{(0)} = (\Sigma_1^{(0)}, \ldots, \Sigma_K^{(0)})$

5 for(t = 1 to T_{EM})
5.1 for(k = 1 to K)

5.1.1 $P^{(t)}(w_k) = \frac{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t-1)}, \Sigma^{(t-1)})}{N}$

5.1.2 $\boldsymbol{\mu}_k^{(t)} = \frac{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t-1)}, \Sigma^{(t-1)})\mathbf{x}_i}{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t-1)}, \Sigma^{(t-1)})}$

5.1.3 $\Sigma_k^{(t)} = \frac{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t)}, \Sigma^{(t-1)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t)})^T}{\sum_{i=1}^{N} P(w_k|\boldsymbol{x}_i, \boldsymbol{\mu}^{(t)}, \Sigma^{(t-1)})}$

5.2 $\boldsymbol{\mu}^{(t)} = (\boldsymbol{\mu}_1^{(t)}, \ldots, \boldsymbol{\mu}_K^{(t)})$
5.3 $\Sigma^{(t)} = (\Sigma_1^{(t)}, \ldots, \Sigma_K^{(t)})$

6 for(i = 1 to N)
6.1 $y_i = \sum_{k=1}^{K} P^{(t)}(w_k)\mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k^{(t)}, \Sigma_k^{(t)})$

7 $\mathcal{S} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}$;

8 BP_Training(*net*, $\mathcal{S}$);
9 $\phi(\cdot)$ = Function_Computed_By(*net*);
10 return $\phi(\cdot)$

2.2 GNN Training Algorithm

Algorithm 1 presents the pseudo-code of `GNN_Training`, the proposed unsupervised training procedure for the GNN. The architecture of the GNN, say *net*, may be of any ANN form suitable to learn a mapping $\phi : \mathbb{R}^D \rightarrow \mathbb{R}$, where D is the dimensionality of the feature space. Any feed-forward ANN (either deep or shallow) may be used to this aim. In this paper we apply ANNs that can be described as instances of the broad family of the multilayer perceptrons, such that the backpropagation (BP) algorithm can be exploited within Algorithm 1. Upon learning from the unlabeled training sample $\mathcal{T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of independent feature vectors drawn from an unknown PDF $p(\cdot)$, the function $\phi(\cdot)$ computed by the GNN is expected to be an estimate of $p(\cdot)$. Formally, according to the statistical meaning of the term, this is a non-parametric estimation technique, insofar that we do not make any prior assumption on the form of $p(\cdot)$ [4]. While any kind of activation functions may be used in the hidden layers of the GNN (at least in principle), e.g. logistic sigmoids or ReLUs, there are some constraints on the output activation function. In fact, the latter shall yield the value $\phi(\mathbf{x})$ of a proper PDF evaluated over current input pattern $\mathbf{x}$, i.e. it must be non-negative and it may possibly range over the $[0, +\infty)$ counterdomain. This is readily achieved by resorting to a ReLU output, to a softmax function, or to a sigmoid with adaptive amplitude [8]. The latter can spontaneously extend over the range of values that naturally span the actual counterdomain of $p(\cdot)$. A linear output unit can be used, as well, as long as possible negative outputs are forced to zero at test time.

Step 1 of the algorithm creates a partitioning of $\mathcal{T}$ into K subsets $C_1, \ldots, C_K$, where K is a hyper-parameter that shall undergo model selection (e.g., via cross-validation). Any clustering algorithm can be used to obtain the partitioning, although k-Means [4] is used hereafter due to its being a simplified version of the ML solution for the parameters of a GMM [4]. Steps 2-2.1 initialize $P^{(0)}(w_k), \boldsymbol{\mu}_k^{(0)}, \Sigma_k^{(0)}$ for $k = 1, \ldots, K$ based on the statistics of C_k as follows: $\boldsymbol{\mu}_k^{(0)}$ is defined as the average of the feature vectors in C_k, $\Sigma_k^{(0)}$ is defined as the empirical covariance matrix of C_k with respect to $\boldsymbol{\mu}_k^{(0)}$, and $P^{(0)}(w_k)$ is defined as the relative frequency of the feature vectors in C_k with respect to the overall cardinality N of $\mathcal{T}$, that is $N_k^{(0)}/N$ where $N_k^{(0)} = |C_k|$. The global parameter vectors $\boldsymbol{\mu}^{(0)}$ and $\Sigma^{(0)}$ are initialized in terms of $\boldsymbol{\mu}_k^{(0)}$ and $\Sigma_k^{(0)}$ (for $k = 1, \ldots, K$) in steps 3 and 4, respectively. Steps 5-5.3 iterate (for $t = 1, \ldots, T_{EM}$) the re-estimation of the overall mixing parameters, of $\boldsymbol{\mu}^{(t)}$, and of $\Sigma^{(t)}$ based on the component-specific re-estimates of $P^{(t)}(w_k)$, $\boldsymbol{\mu}_k^{(t)}$, and $\Sigma_k^{(t)}$ relying on Equations 3-5, thus realizing a hill-climbing procedure over the ML criterion. Step 6 is the core of the algorithm: it generates synthetic target outputs for actually training the GNN. For each training pattern $\mathbf{x}_i$ (for $i - 1, \ldots, N$) a one-dimensional target y_i is created exploiting the right-hand side of Equation 1 applied over the parameters estimates yielded by step 5. It is now straightforward to define a supervised training set $\mathcal{S}$ (step 7 of the algorithm) by labeling each of the original unlabeled training patterns $\mathbf{x}_i$ with the corresponding, synthetically-

generated target output y_i, for $i = 1, \ldots, N$. Afterwards, the algorithm assumes two pseudo-procedures are available, namely `BP_Training`(net, $\mathcal{S}$), called in step 8, which realizes regular BP-based training of the network net over the supervised dataset $\mathcal{S}$, and `Function_Computed_By`(net), called in step 9, which is expected to return some representation of the function $\phi(\cdot)$ computed by net. The function $\phi(\cdot)$ forms the output of Algorithm 1 and it is eventually returned (step 10).

3 Experimental Demonstration

Throughout the experiments, a combination of grid-search and tree-structured Parzen estimator search [2] was used for the selection of the architectures and hyper-parameters of the different models. In particular, for the GNN the following parameters underwent search: number K of components, number of hidden layers, number of neurons per layer, form of the activation functions, number of training epochs, learning rate, mini-batch size, dropout, and optimizer (either Adam or RMSprop). Since the BP-based learning step within Algorithm 1 can be seen as accomplishing a regression task, the search was aimed at maximizing the R^2 score. The latter measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It is computed as:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{6}$$

where N is the number of samples, y_i is the target value, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of the target values.

3.1 Evaluation Metrics

Beside the R^2 score, several popular evaluation metrics were used to assess the quality of the estimated PDF $\hat{p}(\cdot)$ yielded by any estimation algorithm with respect to the actual PDF $p(\cdot)$, namely:

- Mean Squared Error (MSE), that is $MSE = \frac{1}{N}\sum_{i=1}^{N}(p(\mathbf{x}_i) - \hat{p}(\mathbf{x}_i))^2$ where $\mathbf{x}_i$ represents the i-th random vector in a random sample $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ empirically drawn from $p(\cdot)$
- Kullback-Leibler (KL) divergence, i.e. $KL(p(\cdot)||\hat{p}(\cdot)) = \sum_i^N p(\mathbf{x}_i)\log\left(\frac{p(\mathbf{x}_i)}{\hat{p}(\mathbf{x}_i)}\right)$
- Maximum Error (Max Error), that is $\max_{i=1,\ldots,N}|p(\mathbf{x}_i) - \hat{p}(\mathbf{x}_i)|$
- Integrated Square Error (ISE), that is the integral (computed numerically) of the squared difference between $p(\cdot)$ and $\hat{p}(\cdot)$: $ISE = \int(p(\mathbf{x}) - \hat{p}(\mathbf{x}))^2 d\mathbf{x}$
- Explained variance score (EVS), defined as $\text{EVS} = 1 - \frac{\sum_{i=1}^{N}(p(\mathbf{x}_i) - \hat{p}(\mathbf{x}_i))^2}{\sum_{i=1}^{N}(p(\mathbf{x}_i) - \bar{p}(\cdot))^2}$ where $\bar{p}(\cdot)$ is the expectation of $p(\cdot)$ computed by averaging over $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

3.2 Results, Round 1: Estimating an Exponential PDF

In the first experimental round we drew random samples $\mathcal{T}$ (for different values of N) from an exponential PDF, that is $p(x) = \lambda \exp(-\lambda x)$ with inverse scale $\lambda = 1.67$. This estimation task is only apparently innocuous, as pinpointed in [12], due to the peculiar form of $p(x)$ that hardly matches traditional parametric and non-parametric statistical models. The GNN was compared with respect to established density estimation techniques, including both statistical and ANN-based approaches. The statistical techniques used were a parametric model, that is the GMMs itself (allowing for a straight comparison between the performance yielded by the "teacher", i.e. the GMM, and the "student", i.e. the GNN) and two popular non-parametric approaches, namely the Parzen window (PW) [4] and the so-called k_n-nearest neighbor (k_n-NN) density estimator[1] [4]. The PNN was adopted as an ANN-based benchmark, due to its established overpowering other ANNs and machine learning algorithms for PDF estimation [12]. A quantitative comparison is reported in Table 1 in terms of the aforementioned metrics (R^2, MSE, Max error, ISE, KL, and EVS). The results in the Table are ordered for decreasing values of R^2. The best score obtained for each metric (i.e., column) and for each value of N is emphasized using boldface fonts. Some entries of the KL column are undefined due to the fact that the ANN-based estimator may yield a null value of the estimated PDF when fed with certain input patterns $\mathbf{x}_i$, and the KL divergence (Section 3.1) is just not defined when $\hat{p}(\mathbf{x}_i) = 0$.

The results allow us to draw the following conclusions: (1) both the ANN-based approaches outperformed all the established statistical techniques, by all metrics, proving that ANNs establish themselves as excellent candidates for PDF estimation; (2) the GNN yielded the highest scores by all metrics and all values of the training set sizes (for $N = 10, 50, 100, 200$), except for KL with $N = 50, 100, 200$ where the PNN yielded better performance; (3) the GNN (the "student") systematically yielded a dramatic improvement over the GMM (the "teacher"), proving the modeling capabilities of ANNs and, above all, proving that their generalization capabilities can actually ease the model convergence to a general probabilistic law that can reflects the nature of the training data sample and its actual probabilistic properties; (4) even when trained with as few as 10 training patterns, the GNN metrics overpowered the scores yielded by any statistical estimates based on any values of N (up to $N = 200$).

Figure 1 shows a graphical plot of the estimates of the exponential PDF with $N = 100$ yielded by the best GMM ($K = 26$) and the best GNN ($K = 13$), respectively. Both estimates are plotted against the actual PDF. It is seen that the GNN curve resulted in a close estimate to the underlying exponential, while the GMM presented issues when its argument x approached the origin of the axis. The noisy, piecewise-linear blue line shown in the right-hand portion (**b**) of Figure 1 connects the blue dots that represent the individual training pairs (x_i, y_i) for $i = 1, \ldots, 100$. This highlights the fact that the learning and generalization capabilities of the GNN do extrapolate an overall smooth density

[1] Not to be confused with the usual k-nearest neighbor classifier.

Table 1: Estimation of the exponential PDF from samples of variable size N: the results are sorted in descending order of R^2 score.

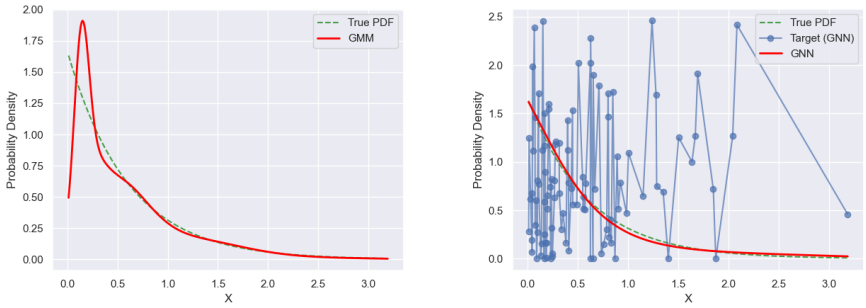
Model	N	R^2	MSE	Max Error	ISE	KL	EVS
GNN	200	**0.9978**	**0.0004**	**0.0766**	**0.0116**	-	**0.997** 8
GNN	100	**0.9953**	**0.0007**	**0.0522**	**0.0233**	0.0095	**0.9954**
PNN	200	0.9949	0.0008	0.2080	0.0261	**0.0014**	0.9952
PNN	100	0.9936	0.0010	0.1900	0.0323	**0.0034**	0.9945
GNN	50	**0.9924**	**0.0012**	**0.0760**	0.0380	0.0212	**0.9926**
PNN	50	0.9911	0.0014	0.2115	0.0444	**0.0076**	0.9911
GNN	10	**0.9722**	**0.0037**	**0.1557**	**0.0668**	**0.0070**	**0.9727**
PW	50	0.9296	0.0113	0.7746	314.40	0.7020	0.9339
PNN	10	0.9291	0.0096	0.2059	0.1705	-	0.9291
PW	100	0.9247	0.0120	0.8547	300.90	0.7020	0.9263
k_n-NN	200	0.9206	0.0129	0.8886	326.50	0.7101	0.9211
PW	200	0.9180	0.0133	0.9090	308.30	0.7101	0.9193
PW	10	0.9142	0.0117	0.4570	74.60	0.3069	0.9146
k_n-NN	100	0.9138	0.0138	0.8010	314.80	0.7020	0.9219
k_n-NN	50	0.9078	0.0147	0.7351	316.26	0.7020	0.9271
GMM	200	0.9070	0.0151	1.1122	344.09	0.7101	0.9070
GMM	100	0.8478	0.0243	1.1380	343.15	0.7020	0.8479
GMM	50	0.7645	0.0376	1.0493	395.08	0.7020	0.7647
k_n-NN	10	0.6933	0.0420	1.2493	115.64	0.3069	0.7145
GMM	10	0.6079	0.0537	0.7735	91.1557	0.3069	0.6116

function from a set of otherwise individually noisy examples. Table 2 reports the optimal number K of components within the GNN training algorithm (as selected via the aforementioned search techniques) as a function of the number N of training patterns, and the corresponding R^2 score.

Table 2: Optimal number K of components in the GNN training algorithm as a function of the training set size N, along with the corresponding R^2 score.

N	K	R^2 Score
10	2	0.9722
50	8	0.9924
100	13	0.9953
200	25	0.9978

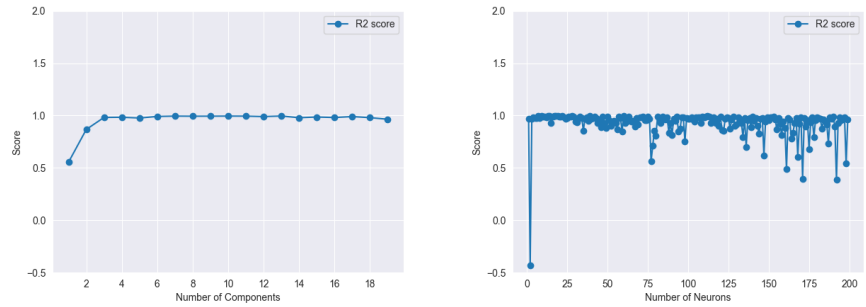
Figure 2 plots the R^2 yielded by the best GNN for the exponential PDF with $N = 100$ as a function of K (left-hand side of the figure) and of the number of hidden neurons (on the right). It is seen that the GNN reached high values of R^2 as soon as $K \geq 4$, and was not affected significantly by larger values for K, although there was a tendency to worsen the performance for $K \geq 15$. As for the impact of the architecture on R^2, it is seen that the number of hidden



(a) Best GMM ($K = 26$) **(b)** Best GNN ($K = 13$)

Fig. 1: Best GMM-based (**a**) and best GNN-based (**b**) estimates of the exponential PDF with $N = 100$.

neurons affects the performance of the GNN to some extent, as expected of any ANN learning process. Trivial architectures (less then 5 hidden neurons) were unfit to the task. Some random fluctuation of the R^2 were observed throughout the overall increase in the number of neurons. More noise in the R^2 curve were observed for the larger architectures, in particular when more then 125 neurons were used. Nonetheless, the vast majority of the non-trivial GNN architectures reached for a high R^2 values regardless of the exact number of neurons.



(a) **(b)**

Fig. 2: Estimates of the exponential PDF (with $N = 100$): R^2 curves as functions of the number K of components (**a**) and of the number of hidden neurons (**b**) in the GNN.

3.3 Results, Round 2: Estimating a multimodal PDF

Random samples $\mathcal{T} = \{x_1, \ldots, x_N\}$ of size N (for $N = 10, 50, 100, 200$) were drawn from a multimodal mixture density with 4 component PDFs (a standard

exponential, i.e. $\exp(-x)$ with $x > 0$, a reflected exponential, i.e. $\exp(x-10)$ with $x < 10$, and two generalized extreme-value distributions (GEV) having different locations μ_1, μ_2, distinct scales β_1, β_2, and the respective *shape* parameters set to 0) and non-identical mixing parameters. This mixture (that can be observed in Figure 3, to be commented later) has the following equation:

$$p(x) = 0.20\exp(-x) + \sum_{i=1}^{2} \frac{P_i}{\beta_i} \exp\left(-\frac{x-\mu_i}{\beta_i}\right) \exp\left\{-\exp\left(-\frac{x-\mu_i}{\beta_i}\right)\right\} + 0.25\exp(x-10)$$

where $P_1 = 0.25$, $P_2 = 0.3$, $\mu_1 = 4.0, \mu_2 = 5.5$, $\beta_1 = 0.8$, and $\beta_2 = 0.7$. The skewness of the GEV makes it all the more difficult to estimate with Gaussian-based statistical approaches. Table 3 reports the results, ordered in decreasing values of R^2. Bold fonts are used to highlight the best results per each metric and per each value of N. Again, the ANN-based estimators outperformed the statistical approaches (that turned out to struggle with the present multimodal setup). The GNN yielded the best scores (including the KL, differently from the previous experimental round) for $N = 100$ and $N = 200$. For smaller datasets, the PNN mostly resulted in slightly better scores (the qualities of the PNN over small samples are established [12]). This is possibly related to the GMM (the "teacher") substantially under-achieving in the present PDF estimation task with $N = 10$ and $N = 50$. Nevertheless, the GNN still resulted in the best scores for some of the metrics when $N = 10$. All in all, based on the present results and those discussed in the previous Section, it is seen that all the non-parametric techniques (either statistical or ANN-based) overpowered the parametric model (the GMM). Figure 3 shows the estimates yielded by the best GMM (left) and by the best GNN (right) for $N = 100$, plotted against the actual multimodal PDF $p(x)$ and against the targets (blue line) generated by steps 6-6.1 of Algorithm 1. Table 4 reports the best GNN architectures (as selected by the search techniques mentioned above) for increasing values of N. Broadly speaking, the optimal depth of the GNN had a tendency to increase with the size of the training set. This was most likely caused by the proneness of ANNs to overfit small datasets whenever their architecture is too complex. Different forms for the activation functions turned up better for different numbers of layers. In general, the logistic sigmoid with adaptive amplitude (Ada-Sigmoid) [8] was selected as the best-performing output activation function.

4 Conclusion and Future Work

The paper contributed a simple and effective algorithm for training an ANN to estimate a PDF from an unlabeled sample of data. In fact, possibly the best selling point of the algorithm lies in its simplicity. The approach exploits both the statistical robustness of GMMs and the nonlinear modeling capabilities of ANNs. Preliminary experimental results showcased the potential of the GNN, the latter outperforming the GMM (i.e., the student has become the master) and the most

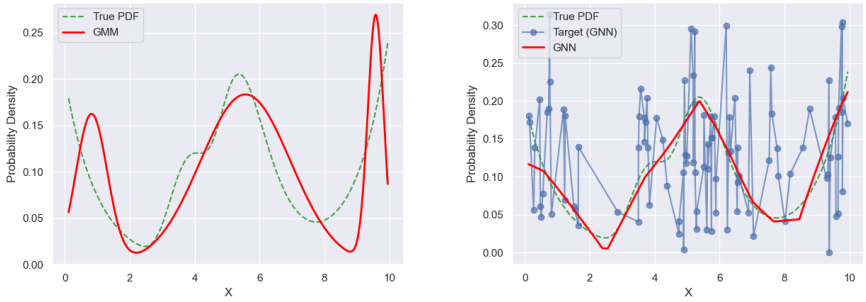
Table 3: Estimation of the multimodal PDF from samples of variable size N: the results are sorted in descending order of R^2 score.

Model	N	R^2	MSE	Max Error	ISE	KL	EVS
GNN	100	**0.9498**	**0.0002**	**0.0624**	**0.0150**	**0.0103**	**0.9500**
GNN	200	**0.9235**	**0.0002**	**0.0704**	**0.0241**	**0.0118**	**0.9240**
PNN	100	0.9047	0.0003	0.1163	0.0286	0.0228	0.9073
PNN	200	0.8899	0.0003	0.0943	0.0347	0.0134	0.8976
PNN	50	**0.8297**	**0.0005**	**0.0607**	**0.0454**	**0.0313**	**0.8341**
GNN	50	0.8035	0.0006	0.0642	0.0524	0.0427	0.8035
PW	100	0.7974	0.0006	0.1288	53.9312	0.1609	0.7998
PW	200	0.7872	0.0007	0.1427	58.5740	0.1627	0.7919
PNN	10	**0.7837**	**0.0006**	0.1010	**0.0492**	0.0555	**0.7838**
GMM	200	0.7536	0.0008	0.1737	68.8703	0.1627	0.7540
k_n-NN	200	0.7404	0.0008	0.1388	55.7257	0.1627	0.7438
k_n-NN	100	0.7298	0.0008	0.1069	59.8825	0.1609	0.7424
GNN	10	0.7250	0.0008	**0.0800**	0.0625	**0.0406**	0.7729
PW	50	0.6615	0.0010	0.1014	52.5236	0.1576	0.6620
k_n-NN	50	0.4752	0.0015	0.1541	64.3591	0.1576	0.5261
GMM	100	0.4636	0.0016	0.1521	69.2541	0.1609	0.4638
PW	10	0.4570	0.0016	0.0938	46.1844	0.1706	0.6277
k_n-NN	10	0.3616	0.0019	0.0941	20.1995	0.1706	0.4160
GMM	10	0.1837	0.0024	0.1113	43.9243	0.1706	0.4851
GMM	50	0.0720	0.0026	0.2248	82.3759	0.1576	0.0913

popular non-parametric estimators. Likewise, the GNN compared quite favorably with the established ANN-based technique. These findings corroborate the idea that combining statistical approaches with ANNs can actually lead to better PDF estimates. On-going research is focusing on (1) extending the experimental investigation to multi-dimensional random vector distributions, and on (2) a study of the modeling and convergence capabilities of the GNN.

References

1. `https://github.com/Duccioo/Gaussian-Mixture-Neural-Network`
2. Akiba, T., *et al.*: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD. pp. 2623–2631. ACM (2019)
3. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum Likelihood from Incomplete Data Via the EM Algorithm. Journal of the Royal Statistical Society: Series B **39**(1), 1–22 (12 1977)
4. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification (2nd Edition). Wiley (2001)
5. Magdon-Ismail, M., Atiya, A.: Density estimation and random variate generation using multilayer networks. IEEE Trans. on Neural Networks **13**(3), 497–520 (2002)
6. Modha, D.S., Fainman, Y.: A learning law for density estimation. IEEE Trans. on Neural Networks **5**(3), 519–23 (1994)



(a) Best GMM **(b)** Best GNN

Fig. 3: Best GMM-based (**a**) and GNN-based (**b**) estimates of the multimodal PDF with $N = 100$.

Table 4: Multimodal PDF: best GNN architectures for increasing values of N. *Legend*: "Layers" is the number of hidden layers, "Neurons" is the number of neurons in the 1st hidden layer, "AF" is short for "activation function", "Ada-Sigmoid" is a sigmoid with adaptive amplitude [8].

N	Layers	Neurons	Hidden AF	Output AF
10	2	54	ReLU	Ada-Sigmoid
50	3	24	Tanh	Ada-Sigmoid
100	4	26	Sigmoid	Linear
200	4	39	Tanh	Ada-Sigmoid

7. Pedregosa, F.*et al.*: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
8. Trentin, E.: Networks with trainable amplitude of activation functions. Neural Networks **14**(4-5), 471–493 (2001)
9. Trentin, E.: Soft-constrained neural networks for nonparametric density estimation. Neural Process. Lett. **48**(2), 915–932 (2018)
10. Trentin, E.: Asymptotic convergence of soft-constrained neural networks for density estimation. Mathematics **8**(4), 572 (2020)
11. Trentin, E.: Multivariate density estimation with deep neural mixture models. Neural Process. Lett. **55**(7), 9139–9154 (2023)
12. Trentin, E., Lusnig, L., Cavalli, F.: Parzen neural networks: Fundamentals, properties, and an application to forensic anthropology. Neural Networks **97**, 137–151 (2018)