Parzen-PNN Gaussian Mixture Estimator: Experiment Report

Fabrizio Benvenuti

January 11, 2026

Abstract

This report will describe the results and performance differences between Parzen Window and Parzen Neural Network (PNN) estimation methods. They will be benchmarked against two-dimensional Probability Density Functions formed by a Mixture of Gaussians; while varying the cardinality of the extracted point set, the architecture of the neural network, and the hyperparameters of both estimation methods.

1 Introduction

1.1 Selected PDF's overview

The project consists of estimating three previously selected two-dimensional PDFs, each formed by a mixture of an odd number of Gaussians [1,3,5], using a finite number of sample points from them.
The selection process was carried out choosing each Gaussian's weights and statistical parameters (mean and variance) to avoid PDFs with either excessively overlapping peaks or ones that are too distant from each other.
This was done to also check wether high and low variance parts of the PDF were being estimated correctly.

1.2 Sampling Method

The sampling is done by extracting a set of points from the PDF, normalizing the probability of choosing a gaussian based on its weight in the mixture.
The extraction process was implemented like so:

- Use a weighted random choice to select a Gaussian from the mixture.
- Extract a point from the selected Gaussian
- Compute the PDF value at that point by summing the weighted contributions of all Gaussians in the mixture.

2 Estimation Methods

The PDF will then be estimated non-parametrically using the Parzen Window and the Parzen Neural Network, as will be described in the later sections.
This Methods still fall into the supervised learning category, as they both require a training set of samples.
In this section we will briefly describe the theoretical background of both methods.

2.1 Parzen Window Estimation

Given a dataset of independent and identically distributed (i.i.d.) samples $\underline{X_1}, \underline{X_2}, \dots, \underline{X_n}$ drawn from an unknown distribution with density function $p(\overline{x})$, we estimate $p(\overline{x})$ using Parzen Windows as:

$$p_n(\overline{x_0}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{V_n} \phi\left(\frac{\underline{x_0} - \underline{x_i}}{h_n} \right), \tag{1}$$

where:

- n is the cardinality of the training set.
- h_n is the window width (bandwidth).
- V_n is the volume of the window.
- $\phi(\cdot)$ is a kernel function, commonly chosen as the Gaussian:

$$\phi(u) = N(\underline{u}; \underline{0}; I) \tag{2}$$

The choice of h affects the estimator's properties:

- Large h oversmooths the density estimate (high bias, low variance).
- Small h leads to high variance, making the estimate sensitive to noise.

2.2 Parzen Neural Networks (PNNs)

A Parzen Neural Network is any ANN, equipped with a supervised learning algorithm, that is trained as follows:

```
Algorithm 1: Train Parzen Neural Network
  Data: dataset $\tau = \{x_1, \ldots, x_n\}$, initial bandwidth $h_1$, dimension $d$,
        ANN hyperparameters, $\phi(\cdot)$ kernel function
  Result: Trained ANN parameters; probability density function
          estimation $\hat{p}(\cdot)$
  $h_n = \frac{h_1}{\sqrt{n-1}}$;
  $V_n = h_n^d$;
  for $i = 1$ to $n$ do
      $\tau_i \leftarrow \tau \setminus \{x_i\}$;
      $y_i \leftarrow \frac{1}{n-1} \sum_{x \in \tau_i} \frac{1}{V_n} \phi\left(\frac{x_j - x}{h_n}\right)$;
  $S \leftarrow \{(x_i, y_i) \mid i = 1, \ldots, n\}$;
  Train the ANN via supervised learning on the set $S$;
  $\hat{p}(\cdot) \leftarrow$ function computed by the trained ANN;
```

2.2.1 Output Constraints and Theoretical Properties of PNNs

A Parzen Neural Network (PNN) is trained to regress Parzen targets and then used as an (unnormalized) density surrogate $\hat{p} : \mathbb{R}^d \to \mathbb{R}_+$. Since a probability density must satisfy non-negativity, the output layer is chosen as

$$\hat{p}(x) = g(z(x)), \qquad g : \mathbb{R} \to \mathbb{R}_+, \tag{3}$$

so that

$$\hat{p}(x) \geq 0, \quad \forall x \in \mathbb{R}^d. \tag{4}$$

Typical choices are ReLU/softplus or exponential activations.

A PNN does not automatically enforce the normalization condition

$$\int_{\mathbb{R}^d} \hat{p}(x)\, dx = 1. \tag{5}$$

In general,

$$\int_{\mathbb{R}^d} \hat{p}(x)\, dx \neq 1, \tag{6}$$

so $\hat{p}$ should be interpreted as an unnormalized density estimate. When a proper pdf is required, one selects a compact region $X \subset \mathbb{R}^d$ capturing the essential support of the distribution and normalizes on X:

$$Z = \int_X \hat{p}(x)\, dx, \qquad p_{\text{norm}}(x) = \frac{\hat{p}(x)}{Z}. \tag{7}$$

In practice, X is estimated from the data after normalization/standardization (e.g., bounding box, ellipsoid, convex hull, or high-quantile level set). Restricting integration to compact X makes numerical normalization feasible. If X admits uniform sampling and known volume, Monte Carlo integration yields

$$Z \approx \mathrm{vol}(X)\,\frac{1}{M}\sum_{m=1}^{M}\hat{p}(u_m), \qquad u_m \sim \mathrm{Unif}(X). \tag{8}$$

The training targets y_i in Algorithm 1 are constructed with a leave-one-out Parzen Window estimator:

$$y_i = \frac{1}{n-1}\sum_{x\in\tau_i}\frac{1}{V_n}\,\phi\!\left(\frac{x_i - x}{h_n}\right), \quad \tau_i = \tau \setminus \{x_i\}. \tag{9}$$

Removing the self-kernel avoids an artificial peak at x_i and yields an (unbiased) estimate at sample locations. Since targets are available only at $\{x_i\}_{i=1}^{n}$, the behavior between and outside samples is governed by the network inductive bias.

Exploiting the support X (practical and mathematical view). Standard activations are non-local basis functions; therefore fitting $\hat{p}(x_i) \approx y_i$ does not imply $\hat{p}(x) \to 0$ away from the data and may generate heavy tails. To encourage compact support, boundary constraints can be introduced. Let ξ be the diameter of X and set $\delta = \alpha\xi$ with small $\alpha \in (0,1)$. Define

$$B_\delta = \{x \in \mathbb{R}^d \mid \mathrm{dist}(x, X) < \delta\}, \qquad \overline{B}_\delta = B_\delta \setminus X, \tag{10}$$

where $\mathrm{dist}(x, X) = \inf_{y\in X}\|x - y\|$. Sample $k < n$ points $\tilde{x}_j \sim \mathrm{Unif}(\overline{B}_\delta)$ and add zero-density labels $S_\delta = \{(\tilde{x}_j, 0)\}_{j=1}^{k}$. Training on $S \cup S_\delta$ can be interpreted as minimizing

$$\min_{\theta}\ \frac{1}{n}\sum_{i=1}^{n}\big(\hat{p}_\theta(x_i) - y_i\big)^2 + \lambda\,\frac{1}{k}\sum_{j=1}^{k}\hat{p}_\theta(\tilde{x}_j)^2, \tag{11}$$

which penalizes probability mass near the boundary of X. This promotes $\hat{p}(x) \approx 0$ outside the data support, stabilizes numerical normalization, and mitigates spurious heavy tails.

3 Estimation performance Calculation

The performance of each method will be evaluated based on several metrics, including:

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Maximum Absolute Error
- Mean Absolute Error (MAE)

3.0.1 Parzen Window estimation

The Parzen Window method estimates the probability density function (PDF) by placing a kernel at each sampled data point and averaging their contributions over a . The window size (bandwidth) controls the smoothness of the estimate, with smaller windows capturing more detail but increasing variance. In the experiments, the method was evaluated by varying both the number of sampled points and the window size, and the estimation error was measured against the true Gaussian mixture PDF. This approach provides a flexible, non-parametric way to approximate complex distributions from finite samples.

3.0.2 Parzen Neural Network estimation

The latter will consist of 1 or 2 hidden layers with a sigmoid activation function. and the output layer will use either ReLU or sigmoid functions with variable amplitude.

4 Comparison with Other Models

Model	**Training Time**	**Classification Time**	**Memory Usage**	**Robustness**
Parzen Windows	Slow	Slow	High	Good
Parzen Neural Networks	Fast	Slow	Very High	Excellent
k-NN Classifier	Fast	Slow	High	Good
SVMs	Slow	Fast	Low	Excellent
Deep Neural Networks	Very Slow	Fast	Medium-High	Good

Table 1: Comparison of Parzen-based models with other ML techniques.

4.1 Probabilistic Neural Networks (PNN)

A Probabilistic Neural Network is designed for classification tasks by modeling the probability density functions of different classes. Its formulation supports rapid training and high noise tolerance.

4.2 Gaussian Mixture Models (GMM)

Gaussian Mixture Models assume that all data points are generated from a mixture of several Gaussian distributions. This approach is effective for clustering and modeling complex data distributions.

4.3 Integration

In this experiment, combining Parzen window estimation with PNN and GMM leverages the advantages of both non-parametric and probabilistic approaches, providing robust performance for density estimation and classification tasks.

5 Experimental Setup

Different configurations were tested:

- Architectures with varying layers (e.g., 20, 10, 50, etc.).
- Activation functions including Tanh, Sigmoid, and LeakyReLU.
- Variation in parameters such as the number of kernels (nk) and bandwidth (bw).

The performance was monitored by analyzing the loss function over several training epochs.

6 Results

The logs indicate that:

- The configuration with 20 and 10 layers using Tanh activation showed a consistent decrease in loss, achieving values below 0.002 by epoch 400.
- The 50-layer configuration with Sigmoid activation maintained a higher loss value, indicating limited improvement.
- The setup with 30-20-10 layers using LeakyReLU activation displayed rapid loss decrease, suggesting effective learning.

7 Conclusions

The experiment confirms that the network architecture and the activation function significantly affect convergence and performance. The integration of Parzen window estimation with PNN and GMM is promising for density estimation problems. Further research could optimize these configurations for broader applications.