

Parzen-PNN Gaussian Mixture Estimator: Experiment Report

Fabrizio Benvenuti

January 13, 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 (kernel parameterization), 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 whether 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.

These methods are density estimators that learn from an unlabeled sample set (no class labels).

In this section we will briefly describe the theoretical background of both methods.

2.1 Parzen Window Estimation

Let $Y = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ be i.i.d. samples drawn from an unknown density $p(x)$. For a query point x_0 , consider a region $R_n(x_0) \subset \mathbb{R}^d$ with volume V_n and let k_n be the number of samples falling in R_n . A generic non-parametric density estimator is

$$p_n(x_0) = \frac{k_n/n}{V_n}. \quad (1)$$

Since k_n depends on the random sample, $p_n(x_0)$ is itself a random variable. The estimator is consistent in probability if and only if

$$\lim_{n \rightarrow \infty} V_n = 0, \quad \lim_{n \rightarrow \infty} k_n = \infty, \quad \lim_{n \rightarrow \infty} \frac{k_n}{n} = 0. \quad (2)$$

Two complementary constructions satisfy these conditions: fixing V_n (Parzen Window) or fixing k_n (k -nearest neighbors).

In the Parzen Window method, R_n is chosen as a hypercube centered at x_0 with edge length h_n , so that $V_n = h_n^d$ and $h_n \rightarrow 0$ as $n \rightarrow \infty$. Define the window (kernel) function

$$\phi(u) = \begin{cases} 1, & |u_j| \leq \frac{1}{2}, \forall j = 1, \dots, d, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

The indicator of whether x_i belongs to $R_n(x_0)$ is then $\phi((x_0 - x_i)/h_n)$, yielding

$$k_n = \sum_{i=1}^n \phi\left(\frac{x_0 - x_i}{h_n}\right). \quad (4)$$

Substituting (4) into (1) gives the Parzen density estimator:

$$p_n(x_0) = \frac{1}{n} \sum_{i=1}^n \frac{1}{V_n} \phi\left(\frac{x_0 - x_i}{h_n}\right), \quad V_n = h_n^d. \quad (5)$$

Equation (5) shows that Parzen Window estimation is a kernel machine obtained by averaging localized contributions centered at the samples. Replacing the rectangular kernel ϕ with smooth kernels (e.g. Gaussian) yields classical kernel density estimation.

Literature form vs. what we implement. The classical presentation above uses a *rectangular* window (hard indicator) and estimates $p(x_0)$ by counting samples in a hypercube around x_0 . In our implementation we instead use a *Gaussian* kernel (Gaussian KDE).

Rectangular window advantages: simplest derivation and intuitive counting interpretation. *Rectangular window disadvantages:* discontinuous (blocky) estimates and strong sensitivity to boundary effects.

Gaussian KDE advantages: smooth estimates and better qualitative match to the Gaussian-mixture ground truth used in this report. *Gaussian KDE disadvantages:* global support (non-compact tails) and higher computational cost without acceleration.

For the purposes of comparing smooth density estimators on mixtures of Gaussians, Gaussian KDE is the more appropriate choice.

2.2 Parzen Neural Networks (PNNs)

A Parzen Neural Network is any ANN trained to produce a density estimate. In the classical presentation one may construct Parzen/KDE targets and regress them; in this report we instead train the network by maximum likelihood on samples only, while enforcing that it represents a proper pdf on a fixed compact domain.

Algorithm 1: Train Parzen Neural Network

Data: samples $\tau = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$, compact domain $D \subset \mathbb{R}^d$, ANN architecture and optimizer hyperparameters

Result: Trained ANN parameters; pdf estimate $p_\theta(\cdot)$ on D

Define a positive unnormalized model $\tilde{p}_\theta(x) = g(f_\theta(x))$ with
 $g : \mathbb{R} \rightarrow \mathbb{R}_+$;

Define the normalization constant $Z_\theta = \int_D \tilde{p}_\theta(u) du$;

Define the normalized pdf on D : $p_\theta(x) = \tilde{p}_\theta(x)/Z_\theta$;

Train by maximum likelihood: minimize $\mathcal{L}_{\text{NLL}}(\theta) = -\frac{1}{n} \sum_{i=1}^n \log p_\theta(x_i)$;

2.2.1 Output Constraints and Theoretical Properties of PNNs

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

$$\hat{p}(x) = g(z(x)), \quad g : \mathbb{R} \rightarrow \mathbb{R}_+, \quad (6)$$

so that

$$\hat{p}(x) \geq 0, \quad \forall x \in \mathbb{R}^d. \quad (7)$$

Typical choices are ReLU/softplus or exponential activations.

A PNN does not automatically enforce the normalization condition

$$\int_{\mathbb{R}^d} \hat{p}(x) dx = 1. \quad (8)$$

In general,

$$\int_{\mathbb{R}^d} \hat{p}(x) dx \neq 1, \quad (9)$$

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 :

$$Z = \int_X \hat{p}(x) dx, \quad p_{\text{norm}}(x) = \frac{\hat{p}(x)}{Z}. \quad (10)$$

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

$$Z \approx \text{vol}(X) \frac{1}{M} \sum_{m=1}^M \hat{p}(u_m), \quad u_m \sim \text{Unif}(X). \quad (11)$$

Remark (oracle targets vs. sample-only training). In synthetic experiments one could evaluate the ground-truth density $p(x)$ at each sampled point x_i and train a network with labels $y_i = p(x_i)$. These are called *oracle targets* because they use privileged information (the exact mixture parameters) that would not be available in a real density-estimation setting. They can be useful for debugging, but they do not reflect the actual task of learning from samples only.

In this report we **do not** train the PNN on $y_i = p(x_i)$. We train only from samples by maximizing likelihood under the learned density model (see Experimental Setup).

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 $\hat{p}(x) \rightarrow 0$ 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

$$B_\delta = \{x \in \mathbb{R}^d \mid \text{dist}(x, X) < \delta\}, \quad \bar{B}_\delta = B_\delta \setminus X, \quad (12)$$

where $\text{dist}(x, X) = \inf_{y \in X} \|x - y\|$. Sample $k < n$ points $\tilde{x}_j \sim \text{Unif}(\bar{B}_\delta)$ 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

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n (\hat{p}_\theta(x_i) - y_i)^2 + \lambda \frac{1}{k} \sum_{j=1}^k \hat{p}_\theta(\tilde{x}_j)^2, \quad (13)$$

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 Experimental Setup

Abstract. The goal of the experiment is to compare Parzen Window (PW) estimation and Parzen Neural Network (PNN) estimation on synthetic two-dimensional densities with known ground truth. For each selected Gaussian mixture $p(x)$, datasets of increasing cardinality n are sampled, each estimator is trained/tuned on the sampled data, and its estimate is evaluated against the true density on a common evaluation domain. The comparison is performed while varying (i) the sample size and (ii) the main hyperparameters controlling smoothness and model capacity (PW bandwidth; PNN kernel parameterization and hyperparameters).

3.1 Choices made

Static design choices. The ground-truth densities are fixed mixtures of Gaussians in \mathbb{R}^2 . For a mixture with M components, the density is

$$p(x) = \sum_{m=1}^M \pi_m \mathcal{N}(x; \mu_m, \Sigma_m), \quad \pi_m \geq 0, \quad \sum_{m=1}^M \pi_m = 1, \quad (14)$$

with means $\mu_m \in \mathbb{R}^2$ and positive semidefinite covariances $\Sigma_m \in \mathbb{R}^{2 \times 2}$. Using mixtures provides multimodal targets with controllable overlap and anisotropy.

For PW estimation we use a Gaussian kernel density estimator (as implemented in `ParzenWindowEstimator`). In $d = 2$ dimensions, with bandwidth $h > 0$,

$$\hat{p}_{\text{PW},h}(x) = \frac{1}{n} \sum_{i=1}^n \mathcal{N}(x; x_i, h^2 I_2) = \frac{1}{n(2\pi h^2)} \sum_{i=1}^n \exp\left(-\frac{\|x - x_i\|^2}{2h^2}\right). \quad (15)$$

Why Gaussian kernels. Gaussian kernels are used because the ground truth is itself a mixture of Gaussians: $p(x) = \sum_{m=1}^M \pi_m \mathcal{N}(x; \mu_m, \Sigma_m)$. Using Gaussian kernels makes both PW and the PNN model class naturally matched to the target family: (i) the estimators remain smooth, (ii) the convolution/mixture structure is preserved (Gaussian basis functions yield mixtures of Gaussians), and (iii) with sufficient components/basis functions the estimator can approximate multi-modal Gaussian mixtures with controllable overlap.

For PNN estimation, the implementation in `estimator.py` uses a kernel (RBF) network that parameterizes a density with a standard multilayer perceptron (MLP). Concretely, the network outputs an unnormalized positive score

$$\text{ildep}_\theta(x) = g(f_\theta(x)), \quad g : \mathbb{R} \rightarrow \mathbb{R}_+, \quad (16)$$

and it is turned into a proper pdf on the fixed evaluation rectangle D by explicit normalization:

$$p_\theta(x) = \frac{\tilde{p}_\theta(x)}{Z_\theta}, \quad Z_\theta = \int_D \tilde{p}_\theta(u) du. \quad (17)$$

In the implementation, Z_θ is approximated by a differentiable Riemann sum on the same uniform grid used for evaluation, so that the resulting p_θ integrates (approximately) to 1.

Implementation note. We use a smooth positive output map g (softplus) so that $\tilde{p}_\theta(x) > 0$ and $\log p_\theta(x)$ is well-defined during maximum-likelihood training.

Learnable parameters. All PNN parameters are the neural network weights/biases in f_θ . The shape of f_θ (hidden-layer widths and activation) is the architectural choice that we sweep in the experiments.

How can we match a ground truth with different covariances? Although the ground truth is a Gaussian mixture with possibly anisotropic covariances, an MLP-defined density on D is a flexible function class: with sufficient hidden units it can approximate multi-modal and anisotropic shapes (within D) by learning an appropriate log-density landscape.

Is this still a Parzen Neural Network? Yes in the broad sense: a PNN is a neural model trained to produce a density estimate derived from Parzen/KDE-style targets. When centers are fixed at samples and covariances are fixed, the kernel expansion reduces to a Parzen estimator; learning the kernel parameters distills the Parzen estimate into a compact, trainable representation.

Dynamic variables and grids. The experimental factors are discretized into finite sets to enable controlled sweeps:

- *Sample size.* A set of cardinalities $n \in \mathcal{N}$ is used to probe the bias–variance trade-off and convergence behavior as data increases.
- *PW bandwidth.* A set $h \in \mathcal{H}$ spanning small to large smoothing is used. Small h reduces bias but increases variance, while large h oversmooths.
- *PNN hyperparameters.* We sweep the neural network *architecture*, i.e. the number of hidden layers, the number of units per layer, and the activation function. Each architecture is trained by maximum likelihood with explicit normalization over D .

Chosen PNN architectures (5 variants). We compare five MLP architectures (depth/width and activation diversity):

- [10] + ReLU
- [20,10] + Tanh

- [50] + Sigmoid
- [30,20,10] + LeakyReLU
- [100,50] + ELU

This set is useful because it probes how density-estimation accuracy depends on representation capacity (depth/width) and nonlinearity, while keeping the training objective fixed (MLE).

3.2 Setup steps

3.2.1 Ancestral sampling from a Gaussian mixture (our case)

For each ground-truth mixture $p(x) = \sum_{m=1}^M \pi_m \mathcal{N}(x; \mu_m, \Sigma_m)$, we generate i.i.d. samples using the standard ancestral procedure:

$$J \sim \text{Categorical}(\pi_1, \dots, \pi_M), \quad X | (J = m) \sim \mathcal{N}(\mu_m, \Sigma_m). \quad (18)$$

In practice, this is implemented by sampling an index J with probabilities (π_m) and then sampling X from the selected Gaussian component. For training the estimators we use only the sampled locations $\{x_i\}_{i=1}^n$.

2) Training/estimating the density.

- *PW*: for each $h \in \mathcal{H}$, compute $\hat{p}_{\text{PW},h}$ from the sample set.
- *PNN (maximum likelihood)*: train the kernel-mixture model by maximizing the sample log-likelihood under p_θ . Given samples $\{x_i\}_{i=1}^n$, the objective is

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^n \log p_\theta(x_i) \iff \min_{\theta} \mathcal{L}_{\text{NLL}}(\theta) = -\frac{1}{n} \sum_{i=1}^n \log p_\theta(x_i). \quad (19)$$

3.2.2 Loss function and optimization

Let \hat{p}_θ denote the PNN density model with parameters θ (kernel centers, mixture weights, and covariance parameters). Since p_θ is a proper pdf on D , we can train it by maximum likelihood on samples. The loss minimized in the implementation is the negative log-likelihood (NLL):

$$\mathcal{L}_{\text{NLL}}(\theta) = -\frac{1}{n} \sum_{i=1}^n \log \hat{p}_\theta(x_i). \quad (20)$$

Optimization is performed with Adam, a first-order stochastic gradient method that updates parameters by combining gradients $\nabla_\theta \mathcal{L}$ with exponential moving averages of first and second moments (adaptive step sizes per parameter). In our script the updates are performed for a fixed number of epochs; each epoch corresponds to one full pass over the sampled training points.

3) Common evaluation protocol. Let $D \subset \mathbb{R}^2$ be a fixed evaluation domain (a rectangle covering the mixtures) and let $\{u_m\}_{m=1}^{M_D}$ be a uniform grid on D . The pointwise error is measured against the true density p on the same grid:

$$\text{MSE} = \frac{1}{M_D} \sum_{m=1}^{M_D} (\hat{p}(u_m) - p(u_m))^2, \quad \text{RMSE} = \sqrt{\text{MSE}}. \quad (21)$$

Analogous definitions are used for MAE and maximum absolute error. This yields, for each mixture and each configuration of (n, h) or $(n, \text{architecture})$, a comparable scalar performance measure.

4) Relation to the report objective. Repeating the above steps across mixtures and hyperparameter grids produces the empirical comparison required in the abstract goal: benchmark PW and PNN on known 2D densities while varying the number of extracted samples, the network architecture, and the key hyperparameters controlling smoothness and capacity.

4 Results

The logs indicate that:

- For Parzen Windows, the bandwidth h controls the bias–variance trade-off: small h reduces bias but increases variance, while large h oversmooths the estimate.
- For the neural PNN, performance depends on the MLP architecture (depth/width and activation), since it controls the function class used to represent the log-density on the compact domain D .
- The reported figures summarize the error values over the explored architecture choices for each mixture.

5 Conclusions

The experiment highlights the role of kernel parameterization in PNN performance: more flexible kernel covariances can reduce the number of kernels required to approximate anisotropic mixture components, while simpler parameterizations can still work given sufficient K . The integration of Parzen window estimation with trainable kernel networks is promising for density estimation problems.