Parzen-PNN Gaussian Mixture Estimator: Experiment Report

Fabrizio Benvenuti

January 19, 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 Project objective

The objective of the project is to study *non-parametric* density estimation in $\mathbb{R}^2$. Given an unlabeled sample set $\{x_i\}_{i=1}^n \subset \mathbb{R}^2$ drawn from an unknown target density $p(x)$, we aim to construct an estimate $\hat{p}(x)$ that approximates the ground truth. In this report, the ground-truth densities are two-dimensional Gaussian mixtures with $\{1, 3, 5\}$ components and the estimators compared are the Parzen Window (PW/KDE) and a Parzen Neural Network (PNN).

1.2 Ground truth and sampling

1.2.1 Selected PDFs 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, and 5 components), 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.

1.2.2 Sampling method (ancestral sampling from a mixture)

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.

Equivalently, for a 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:

$$J \sim \text{Categorical}(\pi_1, \dots, \pi_M), \qquad X \mid (J = m) \sim \mathcal{N}(\mu_m, \Sigma_m). \tag{1}$$

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 $\{x_i\}_{i=1}^{n}$.

1.3 Methodologies compared

The PDF will then be estimated non-parametrically using the Parzen Window and the Parzen Neural Network (PNN). These methods are density estimators that learn from an unlabeled sample set (no class labels). The next sections present the theoretical foundations of Parzen Window/KDE, then the PNN construction and its implementation-oriented details.

2 Theoretical Foundations of Density Estimation

2.1 The Parzen Window (KDE) method

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

$$p_n(x_0) = \frac{k_n/n}{V_n}. \tag{2}$$

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

$$\lim_{n\to\infty} V_n = 0, \qquad \lim_{n\to\infty} k_n = \infty, \qquad \lim_{n\to\infty} \frac{k_n}{n} = 0. \tag{3}$$

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 $h_n \to 0$ as $n \to \infty$. Define the window (kernel) function

$$\phi(u) = \begin{cases} 1, & |u_j| \leq \frac{1}{2},\ \forall j = 1, \ldots, d, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

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

$$k_n = \sum_{i=1}^{n} \phi\left(\frac{x_0 - x_i}{h_n}\right). \tag{5}$$

Substituting (5) into (2) gives the Parzen density estimator:

$$p_n(x_0) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{V_n} \phi\left(\frac{x_0 - x_i}{h_n}\right), \qquad V_n = h_n^d. \tag{6}$$

Equation (6) 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.

2.2 Kernel choice

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 good numerical behavior (continuous derivatives, less blockiness, and stable visualization/evaluation on grids). *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.

2.3 Bandwidth parameterization

Notation (bandwidth) Throughout the report we distinguish a user-chosen *base* bandwidth h_1 from the *effective* bandwidth used by the kernels,

$$h_n = \frac{h_1}{\sqrt{n-1}}, \qquad V_n = h_n^d. \tag{7}$$

The subscript n indicates explicit dependence on the sample size (as required by the consistency conditions for Parzen-type estimators). In the experiments, sweeps are reported in terms of h_1, while the kernel computations use h_n.

2.3.1 Bandwidth parameterization (h_1 vs. h_n)

We use the definition in Eq. (7). In particular, the Parzen Window (Gaussian KDE) baseline uses h_n inside the Gaussian normalization and exponent, even though the sweep is reported in terms of h_1.

3 Parzen Neural Networks (PNN)

3.1 Architecture and training algorithm

A Parzen Neural Network is an artificial neural network trained to regress non-parametric Parzen Window density estimates. Given samples $\tau = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$, the network learns an approximation of the Parzen estimator and is then used as a continuous surrogate of the probability density.

```
Algorithm 1: Train Parzen Neural Network
Data: samples $\tau = \{x_1, \dots, x_n\}$, bandwidth $h_1$, kernel $\phi$, ANN
      architecture and optimizer hyperparameters
Result: Trained ANN parameters; unnormalized density surrogate
        $\hat{p}_\theta(\cdot)$
Use effective bandwidth $h_n$ (Eq. (7)) and $V_n = h_n^d$;
for $i = 1$ to $n$ do
    $\tau_i \leftarrow \tau \setminus \{x_i\}$;
    $y_i \leftarrow \frac{1}{n-1} \sum_{x \in \tau_i} \frac{1}{V_n} \phi\left(\frac{x_i - x}{h_n}\right)$;
$S \leftarrow \{(x_i, y_i)\}_{i=1}^n$;
Train ANN by regression on $S$ (e.g. MSE loss);
$\hat{p}_\theta(\cdot) \leftarrow$ function computed by the trained ANN;
```

3.2 Output and non-negativity constraints

3.2.1 Output constraints, log-density parameterization, and targets

A PNN is trained to regress density targets and is used as a non-negative density surrogate $\hat{p} : \mathbb{R}^d \to \mathbb{R}_+$. Non-negativity is enforced by choosing the output as

$$\hat{p}(x) = g(z(x)), \qquad g : \mathbb{R} \to \mathbb{R}_+, \tag{8}$$

so that

$$\hat{p}(x) \geq 0, \qquad \forall x \in \mathbb{R}^d. \tag{9}$$

Typical choices are ReLU or a scaled sigmoid.

Log-density parameterization (what the code defaults to) In the script, the default training in `main()` uses `density_parameterization=log_density`. In this mode the network outputs an unconstrained scalar $s_\theta(x) \in \mathbb{R}$ and we interpret

$$\hat{p}_\theta(x) = \exp(s_\theta(x)). \tag{10}$$

Training becomes a standard regression on log-targets, i.e. we replace y_i with $\log(y_i + \varepsilon)$ and minimize an MSE on log-values. This removes the need to impose non-negativity by an output-layer constraint (ReLU / scaled-sigmoid), because $\exp(\cdot)$ is always non-negative. The code still supports the direct-density parameterization, where the output layer is constrained to be non-negative.

The targets y_i are constructed with a leave-one-out Parzen estimator, which avoids the self-kernel contribution and yields an (asymptotically) unbiased estimate at sample locations:

$$y_i = \frac{1}{n-1} \sum_{x \in \tau \setminus \{x_i\}} \frac{1}{V_n} \phi\left(\frac{x_i - x}{h_n}\right). \tag{11}$$

Since targets are available only at the sample locations, the behavior between and outside samples is determined by the inductive bias of the network architecture.

3.3 Normalization on finite domain

A PNN does not enforce normalization:

$$\int_{\mathbb{R}^d} \hat{p}(x)\, dx \neq 1 \quad \text{in general.} \tag{12}$$

Therefore $\hat{p}$ is interpreted as an unnormalized density estimate. When a proper pdf is required in our implementation, we normalize on the fixed evaluation rectangle $D \subset \mathbb{R}^2$ (the extttPlotter domain):

$$Z = \int_D \hat{p}(x)\, dx, \qquad p_{\text{norm}}(x) = \frac{\hat{p}(x)}{Z}. \tag{13}$$

Numerically, Z is approximated by a Riemann sum on the same uniform grid used for visualization and evaluation.

Limitation of normalization on a finite domain The normalization on D implicitly assumes that most probability mass lies inside the chosen rectangle. If the estimator assigns non-negligible mass outside D, then the normalized density on D will be biased upward and likelihood-style metrics become sensitive to the choice of D. In practice we choose D large enough to cover the mixtures, and we add optional regularization (boundary penalty) to discourage heavy tails and reduce mass near/outside the boundary.

4 Advanced Regularization Techniques

4.1 PDF support and boundary penalty

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. In our code, X is taken to be the same rectangle as the plot/evaluation domain D. Let ξ be the diameter of X and set $\delta = \alpha\xi$ with small $\alpha \in (0, 1)$. Define

$$B_\delta = \{x \in \mathbb{R}^d \mid \mathrm{dist}(x, X) < \delta\}, \qquad \overline{B}_\delta = B_\delta \setminus X, \tag{14}$$

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

$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} (\hat{p}_\theta(x_i) - y_i)^2 + \lambda \frac{1}{k} \sum_{j=1}^{k} \hat{p}_\theta(\tilde{x}_j)^2, \tag{15}$$

which penalizes probability mass near the boundary of X, stabilizes numerical normalization, and mitigates spurious heavy tails.

4.2 Uniform internal supervision

Uniform interior supervision (`num_uniform_points` in the code) The implementation optionally augments the training set with additional points sampled uniformly inside the evaluation domain D. For each uniform point $u_j \sim \mathrm{Unif}(D)$ we compute a Parzen/KDE target using the same kernel (with bandwidth h_n) and we add the pairs $\{(u_j, \hat{p}_{\mathrm{KDE}}(u_j))\}$ to the regression dataset. This stabilizes training by providing supervision away from the sample locations, while still using only sample-derived targets.

5 Experimental Setup and Model Selection

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).

5.1 Dynamic variables

extbfDynamic 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. In the implementation, the Parzen Window sweep varies approximately from 50 to 200 samples per Gaussian component.
- *PW bandwidth.* A set $h_1 \in \mathcal{H}$ spanning small to large smoothing is used.
- *PNN hyperparameters.* We sweep the neural network *architecture* (1 or 2 sigmoid hidden layers and the output nonlinearity: ReLU or scaled sigmoid) *and* the **learning rate** used by Adam. In the code, representative bandwidth values $h_1 \in \{2, 7, 12, 16\}$ are used for PNN training, and learning-rate sweeps are performed over a predefined grid.

5.2 Candidate architectures

Chosen PNN architectures We compare four prompt-compliant MLP configurations (sigmoid hidden layers; ReLU or scaled-sigmoid output):

- **[20] + output scaled-sigmoid (A auto)**
- **[30,20] + output scaled-sigmoid (A auto)**
- **[30,20] + output ReLU**
- **[20] + output ReLU**

This set probes how density-estimation accuracy depends on representation capacity (depth/width) and the non-negativity constraint at the output, while keeping the training objective fixed (regression on Parzen targets).

5.3 Validation criteria

Model selection criteria (how architectures are chosen) The code performs a grid sweep and reports performance curves. From a model-selection standpoint, we adopt the standard non-parametric principle of choosing the *simplest* architecture that achieves near-best validation performance. In a realistic setting where the ground-truth density is unknown, this can be done by splitting the sample set into train/validation subsets and maximizing a validation criterion such as the (average) log-likelihood on held-out points. More principled viewpoints discussed in the literature include cross-validation based likelihood and the Minimum Description Length (MDL) principle.

Concretely, we use the validation negative log-likelihood (NLL) computed on held-out points $\{x_j\}_{j=1}^m$:

$$\mathrm{NLL}(\hat{p}; \{x_j\}_{j=1}^m) = -\frac{1}{m} \sum_{j=1}^{m} \log \big(\hat{p}(x_j) + \varepsilon\big). \tag{16}$$

For PW/KDE, $\hat{p}$ is already normalized. For the PNN, we normalize the non-negative surface on the finite domain D (Section 3.3) before computing validation likelihood.

5.4 Setup steps

5.4.1 Static design choices

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

$$p(x) = \sum_{m=1}^{M} \pi_m \mathcal{N}(x; \mu_m, \Sigma_m), \qquad \pi_m \geq 0,\ \sum_{m=1}^{M} \pi_m = 1, \tag{17}$$

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 extttParzenWindowEstimator). In $d = 2$ dimensions, with effective bandwidth $h_n > 0$,

$$\hat{p}_{\mathrm{PW},h_n}(x) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{N}(x; x_i, h_n^2 I_2) = \frac{1}{n\,(2\pi h_n^2)} \sum_{i=1}^{n} \exp\left(-\frac{\|x - x_i\|^2}{2h_n^2}\right). \tag{18}$$

In the implementation, h_n is computed from a user-chosen h_1 via $h_n = \frac{h_1}{\sqrt{n-1}}$.

Why Gaussian kernels (non-parametric justification) The estimators are *non-parametric*: they do not assume the data comes from a Gaussian mixture. We use Gaussian kernels primarily for their smoothness (continuous, differentiable estimates), stable optimization behavior, and the fact that Gaussian kernels yield universal, flexible kernel expansions. In other words, the kernel is chosen for numerical and approximation properties rather than to "match" the true distribution family.

For PNN estimation, the implementation in `estimator.py` follows Algorithm 1: an MLP is trained by *regression* on leave-one-out Parzen targets computed from the samples. Concretely, for each training sample x_i we compute a target y_i using a Gaussian kernel with effective bandwidth $h_n = \frac{h_1}{\sqrt{n-1}}$ and $V_n = h_n^d$, and we train the network to match these targets (typically with an MSE-type loss).

Output parameterization and normalization used for evaluation The MLP uses sigmoid hidden activations (1 or 2 hidden layers). At the output we enforce non-negativity either with ReLU or with a scaled sigmoid $A\,\sigma(z)$. In addition, the code supports a *log-density* parameterization, where the network outputs an unnormalized log-score and exponentiation is applied when constructing the density. For plotting and evaluation on a fixed grid D, the resulting non-negative surface is normalized by a Riemann-sum approximation of $Z = \int_D \hat{p}_\theta(u)\, du$ so that the plotted density integrates (approximately) to 1 over D.

extbf2) Training/estimating the density.

- *PW:* for each $h \in \mathcal{H}$, compute $\hat{p}_{\mathrm{PW},h}$ from the sample set.
- *PNN (regression on Parzen targets):* compute leave-one-out Parzen targets $S = \{(x_i, y_i)\}_{i=1}^n$ using a Gaussian kernel with effective bandwidth h_n (Eq. (7)), then train the MLP by minimizing an MSE-type loss between the network output and y_i.

5.4.2 Loss function and optimization

Let f_θ be the PNN network and let y_i be the leave-one-out Parzen target associated with x_i. The core training objective is regression:

$$\mathcal{L}_{\mathrm{PNN}}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\big(f_\theta(x_i) - y_i\big)^2, \tag{19}$$

or, in log-density mode, an MSE on log-targets. 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. We sweep learning rates across a predefined grid and record the training loss curve (regression loss on Parzen targets) as well as an evaluation MSE against the ground-truth density on a fixed evaluation grid.

5.4.3 Common evaluation protocol

extbf3) 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:

$$\mathrm{MSE} = \frac{1}{M_D}\sum_{m=1}^{M_D}\big(\hat{p}(u_m) - p(u_m)\big)^2, \qquad \mathrm{RMSE} = \sqrt{\mathrm{MSE}}. \tag{20}$$

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.

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, together with the *learning rate* used by Adam.

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.

Relation to the report objective extbf4) 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.

6 Analysis of Results and Discussion

6.1 Quantitative performance

Quantitative evaluation is performed on a common evaluation grid over a fixed rectangle $D \subset \mathbb{R}^2$. For each mixture and each configuration of hyperparameters, we compute standard pointwise error metrics against the ground-truth density on that grid, including MSE/RMSE (and optionally MAE and maximum absolute error).

In the implementation, the Parzen Window sweep produces an error-surface figure and an overlay figure per mixture (e.g. `figures/Parzen_errors_mixture*.jpeg` and exttfigures/Parzen_overlay_mixture*.jpeg).

6.2 Qualitative analysis (overlays)

In addition to scalar errors, we inspect 3D surface plots of (i) the true mixture density and (ii) the estimated density. Overlays are useful to diagnose qualitative behaviors that a single metric may hide, such as oversmoothing (excessive bias for large bandwidth), spurious bumps (variance for too-small bandwidth), and missing/merged modes.

6.3 Complexity analysis

Let T be the number of query points at which we evaluate a density estimate. *Parzen Window / KDE* requires storing all n samples and evaluating n kernels per query point, so inference is $\mathcal{O}(nT)$. For a *PNN*, generating the leave-one-out targets costs $\mathcal{O}(n^2)$ in the naive implementation (pairwise kernel evaluations), and training adds the usual cost of backpropagation over a network with W parameters. However, once trained, inference does *not* scale with n: evaluating the network on T query points is $\mathcal{O}(WT)$ (up to constant factors from matrix multiplications). This is the main practical advantage of PNNs: they can compress the sample-based estimator into a fixed-size model.

6.4 Effect of the boundary penalty

When boundary regularization is enabled (Section 4.1), the estimator is explicitly discouraged from assigning mass near/outside the domain boundary. This can reduce spurious heavy tails, stabilize finite-domain normalization, and improve validation NLL when the unregularized model places too much probability mass in low-density regions. The effect is best interpreted jointly via (i) the shape changes in overlay plots and (ii) the validation NLL curves.

7 Conclusions

The experiments compare a memory-based non-parametric estimator (PW/KDE) and a learned surrogate (PNN) trained on Parzen-derived targets. PW/KDE provides a direct kernel estimator whose inference cost scales linearly with the sample count. PNNs can approximate the Parzen estimator on the chosen domain D while offering inference whose cost does not grow with n after training.

Overall, the PNN can be interpreted as a "compressed" density model whose accuracy depends on the bandwidth choice, the network capacity, and regularization (boundary penalty and optional uniform supervision), while PW remains the simplest and most direct baseline.