ProbNN: Probabilistic Neural Network Regression

Kevin Mota da Costa

January 24, 2026

Abstract

This document presents the mathematical formulation and training methodology of *ProbNN*, a probabilistic neural network framework for regression with heteroscedastic uncertainty. The model jointly learns a predictive mean and input-dependent uncertainty by minimizing a Gaussian negative log-likelihood. We describe the neural network architecture, forward and backward propagation, optimization via gradient descent, and numerical stabilization strategies. Benchmark examples illustrate the model's ability to capture local and global structures, perform extrapolation, and remain stable in the presence of discontinuities.

1 Neural Networks for Regression

A feedforward neural network defines a parametric mapping

$$f_\theta : \mathbb{R} \to \mathbb{R},$$

where θ denotes all weights and biases.

For a network with L layers, the forward propagation is given by

$$z^{(1)} = W^{(1)}x + b^{(1)}, \tag{1}$$
$$a^{(1)} = \phi(z^{(1)}), \tag{2}$$
$$\vdots \tag{3}$$
$$z^{(L)} = W^{(L)}a^{(L-1)} + b^{(L)}, \tag{4}$$
$$h = z^{(L)}, \tag{5}$$

where $\phi(\cdot)$ is a nonlinear activation function.

The vector h represents a learned feature embedding of the input.

2 Probabilistic Output Heads

Instead of predicting a single deterministic value, ProbNN models the conditional distribution

$$p(y \mid x)$$

as a Gaussian:

$$y \mid x \sim \mathcal{N}\big(\mu(x), \sigma^2(x)\big).$$

The network therefore branches into two output heads:

$$\mu(x) = h^\top W_\mu + b_\mu, \tag{6}$$

$$s(x) = h^\top W_s + b_s, \tag{7}$$

where $s(x)$ is an unconstrained latent variable.

To ensure positivity of the predictive standard deviation, we define

$$\sigma(x) = \text{softplus}(s(x)) + \epsilon, \tag{8}$$

with $\epsilon > 0$ for numerical stability.

3 Likelihood and Loss Function

Each observation is given as $(x_i, y_i, \delta y_i)$, where δy_i represents known measurement uncertainty.

The total predictive variance is

$$v_i = \delta y_i^2 + \sigma(x_i)^2. \tag{9}$$

The Gaussian negative log-likelihood (NLL) is

$$\mathcal{L} = \frac{1}{2N} \sum_{i=1}^{N} \left[\frac{(y_i - \mu_i)^2}{v_i} + \log v_i \right]. \tag{10}$$

An optional regularization on uncertainty is added:

$$\mathcal{L}_{\text{total}} = \mathcal{L} + \lambda_\sigma \frac{1}{N} \sum_i s_i^2. \tag{11}$$

This formulation balances data fit and calibrated uncertainty.

4 Backpropagation and Gradients

Training consists of minimizing $\mathcal{L}_{\text{total}}$ with respect to all network parameters.

4.1 Gradients at the Output

The gradient with respect to the predictive mean is

$$\frac{\partial \mathcal{L}}{\partial \mu_i} = -\frac{1}{N} \frac{(y_i - \mu_i)}{v_i}. \tag{12}$$

The gradient with respect to σ is

$$\frac{\partial \mathcal{L}}{\partial \sigma_i} = \left(\frac{1}{v_i} - \frac{(y_i - \mu_i)^2}{v_i^2} \right) \sigma_i. \tag{13}$$

Using $\sigma = \text{softplus}(s)$, the chain rule gives

$$\frac{\partial \mathcal{L}}{\partial s_i} = \frac{\partial \mathcal{L}}{\partial \sigma_i} \cdot \text{sigmoid}(s_i), \tag{14}$$

which explains the appearance of the sigmoid function in training.

4.2 Backpropagation Through the Trunk

Gradients from both heads are combined:

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial h_\mu} + \frac{\partial \mathcal{L}}{\partial h_s}.$$

For each hidden layer,

$$\delta^{(l)} = \left(\delta^{(l+1)} W^{(l+1)\top}\right) \odot \phi'(z^{(l)}), \tag{15}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = a^{(l-1)\top} \delta^{(l)}, \tag{16}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \sum_i \delta_i^{(l)}. \tag{17}$$

The derivatives $\phi'(z)$ correspond exactly to the activation backward passes (e.g. $\tanh'(z) = 1 - \tanh^2(z)$).

5 Optimization

Parameters are updated using stochastic gradient descent:

$$\theta \leftarrow \theta - \eta \, \nabla_\theta \mathcal{L}_{\text{total}}, \tag{18}$$

where η is the learning rate.

L2 regularization adds $2\lambda_{L2} W$ to the corresponding gradients, improving numerical stability.

6 Normalized Residuals

Model calibration is assessed using normalized residuals:

$$r_i = \frac{y_i - \mu(x_i)}{\sqrt{\delta y_i^2 + \sigma(x_i)^2}}. \tag{19}$$

A well-calibrated probabilistic model produces residuals approximately distributed as $\mathcal{N}(0, 1)$.

7 Benchmark Examples and Interpretation

7.1 Smooth Baseline: $\sin(x) + 0.5x$

Figure 1 shows a smooth regression problem with limited training data. The model captures the local trend while increasing uncertainty outside the data support, demonstrating controlled extrapolation.

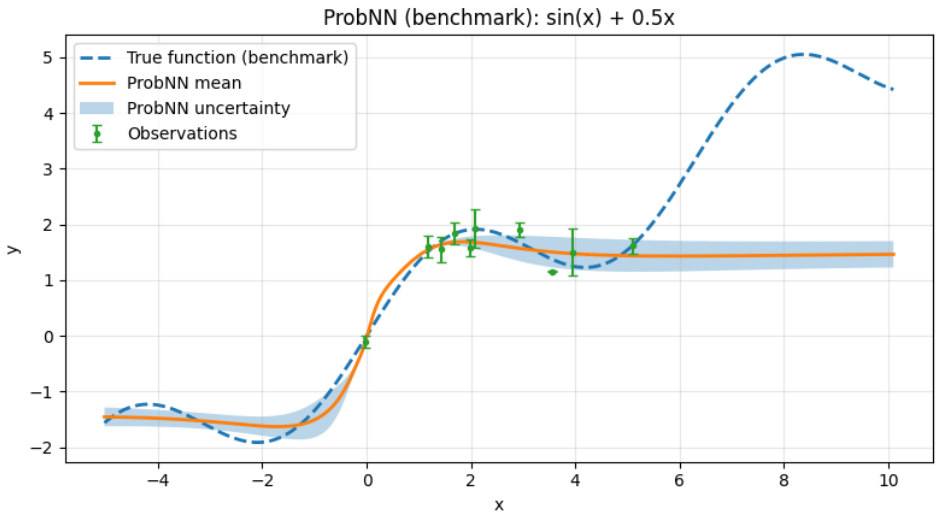

Figure 1: Baseline regression with extrapolation-aware uncertainty.

7.2 Generic Multi-Scale Function

This example combines multiple frequencies and trends. The network learns both local oscillations and global structure, while uncertainty grows in regions with no observations.

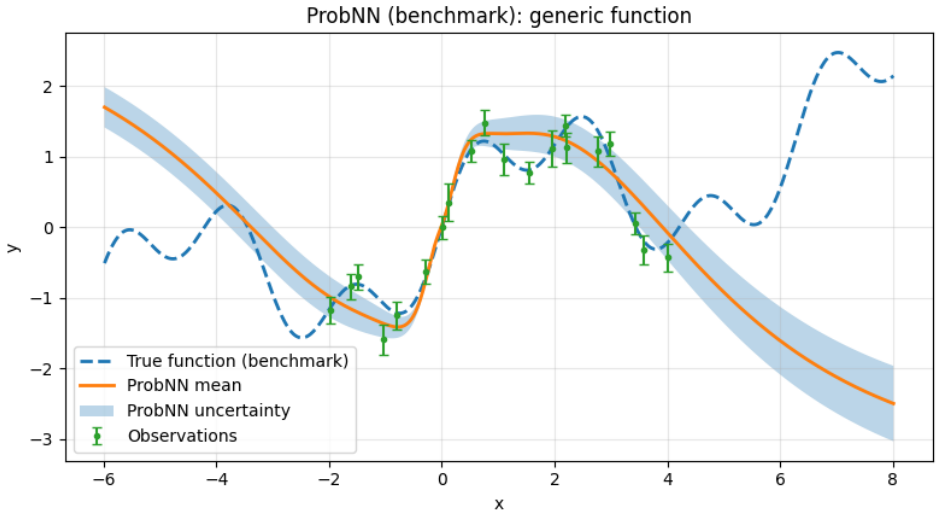

Figure 2: Generic multi-scale function regression.

7.3 Discontinuous Target

Discontinuities pose a challenge for gradient-based models. ProbNN does not force a sharp jump; instead, it expresses epistemic uncertainty near the discontinuity, remaining numerically stable.

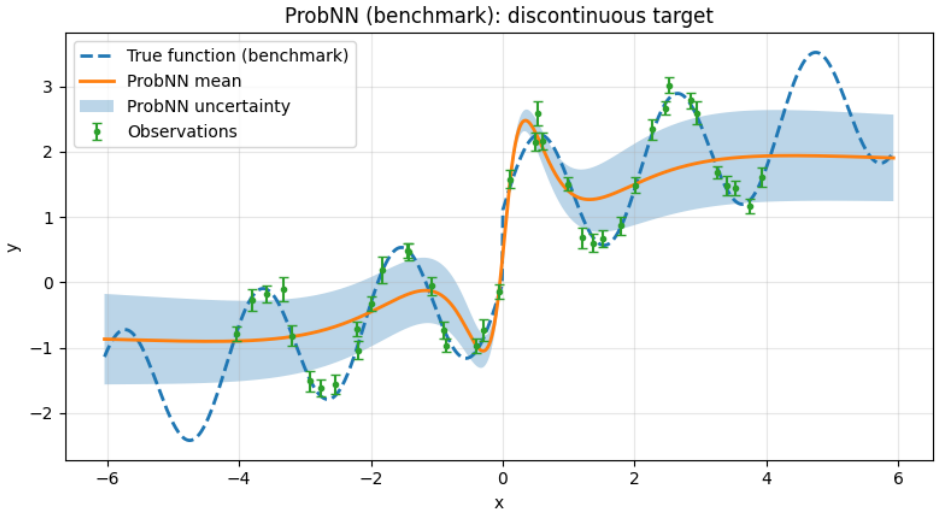

Figure 3: Discontinuous target with uncertainty inflation near the jump.

8 Conclusion

ProbNN combines neural networks and probabilistic modeling to perform regression with calibrated uncertainty. By explicitly optimizing a Gaussian likelihood, the framework naturally handles heteroscedastic noise, extrapolation, and local instabilities, while remaining fully differentiable and trainable via backpropagation.