Neural Networks as Parameterizers of Probability Distributions

ELG 5218 - Uncertainty Evaluation in Engineering Measurements and Machine Learning

Miodrag Bolic

University of Ottawa

February 25, 2026

Roadmap

1. **Why NNs can parameterize distributions at all:** universal approximation $\Rightarrow$ learn maps into parameter spaces.
2. **General distribution parameterizations:**
    - mixtures / MDNs,
    - autoregressive factorizations,
    - latent-variable models (VAEs).
3. **Gaussian likelihood heads:** mean–variance networks (MVE), learning $\sigma(x)$ without labels, failure modes and fixes.
4. **Where it is used:** a summary table.

Step 1: Neural networks as function families

A feedforward neural network defines a parametric family of functions:

$$f_\theta : \mathbb{R}^d \to \mathbb{R}^m$$

with parameters θ (weights/biases).
Universal approximation: with sufficient capacity, networks can approximate broad classes of functions to arbitrary accuracy (under standard conditions). (Cybenko 1989; Hornik 1991)

Key implication

If a distribution can be described using (one or more) functions, then a NN can be used to approximate those functions, hence *parameterize* the distribution.

Step 2: Constrained outputs $\Rightarrow$ valid distributions

To parameterize a distribution, NN outputs must respect constraints:

- probabilities sum to 1: use **softmax** (categorical / mixture weights),
- variances positive: use **exp** or **softplus**,
- covariance Symmetric Positive Definite SPD: e.g., Cholesky factorization.

So we define:

$$\eta(x) = g\big(f_\theta(x)\big),$$

where $g(\cdot)$ enforces constraints and $\eta(x)$ are valid distribution parameters.

Two modeling goals: conditional vs. generative

(A) Conditional density estimation: model $p_\theta(y \mid x)$

$$\eta(x) = f_\theta(x), \quad p_\theta(y \mid x) = p(y; \eta(x)).$$

(B) Generative modeling: model $p_\theta(x)$, often via latent noise z:

$$z \sim p(z), \qquad x = \text{model}_\theta(z), \qquad x \sim p_\theta(x).$$

Different families make $p_\theta(\cdot)$ tractable in different ways.

Generative modeling via latent variables

- Let Z be a *random noise variable*.
$$Z \sim p(z)$$
where $p(z)$ is a simple prior (e.g. standard Gaussian or uniform).
- Let f_θ be a *deterministic neural network*.
$$X_\theta = f_\theta(Z)$$
Since Z is random and f_θ is deterministic, X_θ is also a *random variable*.
- Therefore X_θ has an induced distribution:
$$X_\theta \sim p_\theta(x).$$

Key idea:

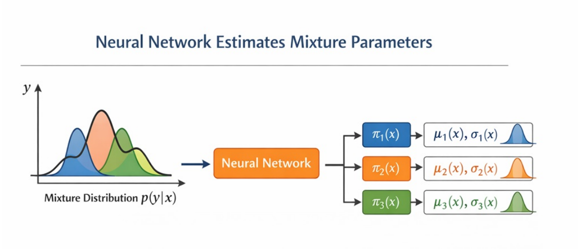
- $p_\theta(x)$ is *not* output by the neural network.
- Instead, $p_\theta(x)$ is the *distribution of samples* produced when noise $Z \sim p(z)$ is transformed through the network f_θ.
- In other words: the NN *defines* the distribution by pushing $p(z)$ through f_θ.

Family 1: Mixtures / Mixture Density Networks (MDNs)

Idea: Approximate complex (multi-modal) densities using mixtures of simple components.
Conditional mixture model:

$$p_\theta(y \mid x) = \sum_{k=1}^{K} \pi_k(x)\, p_k\big(y; \theta_k(x)\big),$$

where a NN outputs $\{\pi_k(x), \theta_k(x)\}_{k=1}^{K}$.
MDNs were introduced as NN-controlled mixture models to represent *arbitrary conditional densities* in principle. (Bishop 1994)



Family 2: Autoregressive factorization (e.g., MADE)

Key theorem (probability chain rule):

$$p(x) = \prod_{d=1}^{D} p(x_d \mid x_{<d}).$$

Neural autoregressive density estimation: a NN outputs parameters of each conditional factor while enforcing the dependency structure.
MADE uses masking to guarantee that output d depends only on $x_{<d}$ and yields tractable likelihood. (Germain et al. 2015)

Autoregressive Model of Order 2 (AR(2))

A classical AR(2) process specifies that the next value depends on the two most recent past values:

$$x_t = a_1 x_{t-1} + a_2 x_{t-2} + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma^2).$$

This implies a conditional density:

$$p(x_t \mid x_{t-1}, x_{t-2}) = \mathcal{N}\big(a_1 x_{t-1} + a_2 x_{t-2},\ \sigma^2\big).$$

Key idea: autoregression factorizes the sequence density using the probability chain rule:

$$p(x_{1:T}) = p(x_1)\, p(x_2 \mid x_1) \prod_{t=3}^{T} p(x_t \mid x_{t-1}, x_{t-2}).$$

Neural Generalization of AR(2)

Replace the linear mapping with a neural network:

$$x_t \mid x_{t-1}, x_{t-2} \sim \mathcal{N}\Big(\mu_\theta(x_{t-1}, x_{t-2}),\ \sigma_\theta(x_{t-1}, x_{t-2})^2\Big).$$

Neural networks:

- Input: (x_{t-2}, x_{t-1})
- Output: mean μ_θ and log-std $\log \sigma_\theta$

Example tiny NNs:

$$\mu_\theta : \mathbb{R}^2 \to \mathbb{R}, \quad 2 \to 8 \to 1,$$

$$\log \sigma_\theta : \mathbb{R}^2 \to \mathbb{R}, \quad 2 \to 4 \to 1.$$

Why useful: NN can model nonlinear dynamics, heteroscedastic noise, and complex temporal patterns that linear AR models cannot capture.

Training Data for AR(2) Neural Networks

Given a time series $x_1, x_2, \ldots, x_T$:

Construct training triples:

$$(x_{t-2}, x_{t-1}) \to x_t, \qquad t = 3, \ldots, T.$$

This yields a dataset:

$$\left\{ \left((x_{t-2}, x_{t-1}),\ x_t \right) \right\}_{t=3}^{T}.$$

Training objective: Gaussian negative log-likelihood

$$\ell_t = \log \sigma_\theta(x_{t-1}, x_{t-2}) + \frac{1}{2} \left(\frac{x_t - \mu_\theta(x_{t-1}, x_{t-2})}{\sigma_\theta(x_{t-1}, x_{t-2})} \right)^2 .$$

Total loss:

$$\mathcal{L}(\theta) = \sum_{t=3}^{T} \ell_t.$$

Optimization: gradient descent on $\mathcal{L}(\theta)$.

Intuition: What the AR(2) Neural Network Learns

Inputs to the networks:

$$x_{t-2},\ x_{t-1}$$

Outputs:

$$\mu_t,\ \sigma_t$$

The NN learns:

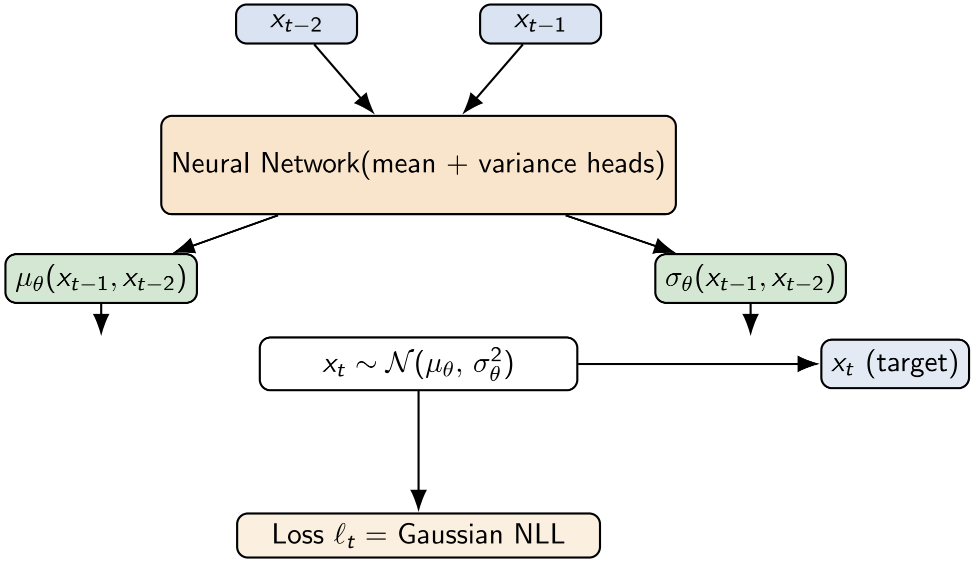
- nonlinear temporal relationships,
- how uncertainty σ_t varies with past states (heteroscedasticity),
- how to represent complex dynamics beyond linear AR(2).

Sampling:

$$x_t \sim \mathcal{N}\big(\mu_\theta(x_{t-1}, x_{t-2}), \sigma_\theta^2(x_{t-1}, x_{t-2})\big).$$

Full sequence generation: draw x_1, x_2 from base distributions, then sequentially generate $x_3, \ldots, x_T$ using NN conditionals.

AR(2) Neural Autoregressive Model


x_{t-2}
x_{t-1}
Neural Network(mean + variance heads)
$\mu_\theta(x_{t-1}, x_{t-2})$
$\sigma_\theta(x_{t-1}, x_{t-2})$
$x_t \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2)$
x_t (target)
Loss ℓ_t = Gaussian NLL

Family 3: Latent-variable models (VAEs)

Generative story:

$$z \sim p(z), \qquad x \mid z \sim p_\theta(x \mid z)$$

where $p_\theta(x \mid z)$ is parameterized by a NN (decoder).
Inference is hard: posterior $p_\theta(z \mid x)$ is usually intractable, so VAEs introduce

$$q_\phi(z \mid x)$$

(encoder NN) and optimize the ELBO. (Kingma and Welling 2013)

Gaussian head = simplest conditional distribution parameterization

Now specialize to a simple conditional family:

$$y \mid x \sim \mathcal{N}\left(\mu_\theta(x),\, \sigma_\theta(x)^2\right).$$

A neural network outputs:

$$(\mu_\theta(x),\ \log \sigma_\theta(x)).$$

This is the classic Mean–Variance Estimation (MVE) network formulation, dating back to early work on learning mean and variance from data via likelihood. (Nix and Weigend 1994)

Gaussian NLL and why $\sigma(x)$ is learnable without labels

Gaussian negative log-likelihood (per datum), up to constants:

$$\mathcal{L}(x, y) = \log \sigma_\theta(x) + \frac{1}{2}\left(\frac{y - \mu_\theta(x)}{\sigma_\theta(x)}\right)^2.$$

Why this provides "supervision" for variance

The loss penalizes:

- large uncertainty ($\log \sigma$ term),
- underestimating uncertainty when residuals are large (scaled residual term).

This likelihood-based learning of variance without variance targets is explicitly derived in classic MVE work. (Nix and Weigend 1994)

Interpretation: optimal σ tracks conditional residual scale

Fix $\mu(x)$ and optimize σ for residual $r = y - \mu(x)$:

$$\ell(\sigma) = \log \sigma + \frac{r^2}{2\sigma^2}.$$

Stationary point satisfies:

$$\frac{d\ell}{d\sigma} = 0 \Rightarrow \sigma^2 = r^2.$$

Over many samples (conceptually at similar x):

$$\sigma(x)^2 \approx \mathbb{E}[(y - \mu(x))^2 \mid x].$$

This explains why variance can be learned from data scatter even without explicit variance labels. (Nix and Weigend 1994)

Pathology: $\sigma(x)$ can "explain away" mean error

Because the NLL trades off mean fit and variance:

$$\log \sigma(x) + \frac{(y-\mu(x))^2}{2\sigma(x)^2},$$

in regions where $\mu_\theta(x)$ is imperfect, increasing $\sigma_\theta(x)$ can reduce loss.
Recent work highlights that likelihood gradients can be *scaled* by predictive variance, potentially compromising mean learning. (Immer et al. 2023)
Predicted $\sigma(x)$ becomes structured (e.g., follows the signal) even if true noise is constant.

Training practice: mean warm-up and regularization

Classic and modern guidance for MVE-style networks includes:

- **Warm-up**: fit mean first with fixed variance; then train mean+variance jointly.
- **Regularize** variance or its smoothness to prevent spurious structure.

These issues and recommendations are discussed explicitly in recent work on optimal training of MVE networks. (Sluijterman et al. 2023)

Training Algorithm for Gaussian-Head (MVE) Networks

Algorithm: Stable Training with Mean Warm-Up

Inputs:

- Dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$
- Network $f_\theta(x)$ producing $(\mu_\theta(x), \log \sigma_\theta(x))$
- Warm-up epochs T_{warm}
- Regularization weight λ

Phase 1: Mean Warm-Up

1. Freeze variance parameters (or set $\sigma_\theta(x) = \text{const}$).
2. For $t = 1, \dots, T_{\text{warm}}$:
   - For each minibatch B:
   - Compute $\mu_\theta(x)$ for all $(x, y) \in B$.
   - Use fixed-variance loss:

$$\ell_i = \frac{(y_i - \mu_\theta(x_i))^2}{2\sigma^2_{\text{fixed}}}.$$

   - Update only θ_μ.

Phase 2: Joint Mean + Variance Training

3. Unfreeze variance parameters.
4. For $t = T_{\text{warm}} + 1, \dots, T_{\text{total}}$:
   - Compute $(\mu_\theta(x), \log \sigma_\theta(x))$ for all $(x, y) \in B$.
   - Compute $\sigma_\theta(x) = \exp(\log \sigma_\theta(x))$.
   - Compute NLL:

$$\ell_i = \log \sigma_\theta(x_i) + \frac{(y_i - \mu_\theta(x_i))^2}{2\sigma_\theta(x_i)^2}.$$

   - Add variance regularization:

$$R = \lambda \| \log \sigma_\theta(x) \|^2 \quad \text{or} \quad \lambda \| \nabla_x \sigma_\theta(x) \|^2.$$

   - Update $(\theta_\mu, \theta_\sigma)$ using loss $L = \frac{1}{|B|} \sum_i \ell_i + R$.

How do we evaluate without “uncertainty labels”?

True uncertainty targets rarely exist. Evaluation uses:

- **Held-out NLL / log predictive density** (proper scoring rule),
- **Calibration** checks (coverage, standardized residuals),
- Robustness under dataset shift (OOD uncertainty).

Deep ensembles are a widely-used baseline emphasizing calibration and log scores for uncertainty evaluation. (Lakshminarayanan et al. 2017)

Calibration test for Gaussian heads

Standardized residuals:

$$z_i = \frac{y_i - \mu_\theta(x_i)}{\sigma_\theta(x_i)}.$$

If the model is well-specified and calibrated:

$$z \sim \mathcal{N}(0, 1).$$

Coverage checks:

- $\Pr(|y - \mu| \leq 1\sigma) \approx 0.68$
- $\Pr(|y - \mu| \leq 2\sigma) \approx 0.95$

These are the practical diagnostics that complement NLL when uncertainty labels are absent. (Lakshminarayanan et al. 2017)

Where NN distribution parameterization is used (summary)

Family	**Distribution form**	**NN outputs**	**Tractability**	**Typical uses**
Gaussian head (MVE) (Nix and Weigend 1994)	$\mathcal{N}(\mu(x), \sigma(x)^2)$	$\mu(x), \log \sigma(x)$	Exact NLL	Probabilistic regression, aleatoric uncertainty
MDN / mixtures (Bishop 1994)	$\sum_k \pi_k(x) p_k(y)$	π_k, θ_k	Exact NLL	Multi-modal $p(y\|x)$, inverse problems
Autoregressive (MADE) (Germain et al. 2015)	$\prod_d p(x_d\|x_{<d})$	params of each conditional	Exact LL	High-dim density estimation, compression
VAE (Kingma and Welling 2013)	$p(x)$	encoder+decoder params	ELBO (approx.)	Representation learning, generation

Take-home messages

- NNs can parameterize distributions because they approximate functions; with constraint mappings, outputs become valid distribution parameters. (Cybenko 1989; Hornik 1991)
- "Universal" density constructions come from mixtures, autoregressive factorizations, flows, latent-variable models, and EBMs. (Bishop 1994; Germain et al. 2015; Papamakarios et al. 2021; Kingma and Welling 2013; LeCun et al. 2006)
- Gaussian mean–variance networks are a simple special case; $\sigma(x)$ is learned indirectly by likelihood (no variance labels needed). (Nix and Weigend 1994)
- Joint mean/variance training can produce pathologies (variance absorbing mean error); warm-up, regularization, and careful calibration matter. (Immer et al. 2023; Stirn et al. 2023; Sluijterman et al. 2023)

References I

- Bishop, Christopher M. (1994). *Mixture Density Networks*. Neural Computing Research Group Technical Report NCRG/94/004. Aston University. URL: `https://eprints.aston.ac.uk/373/`.
- Cybenko, George (1989). "Approximation by Superpositions of a Sigmoidal Function". In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314. URL: `http://aaclab.mit.edu/material/handouts/universalapproximation-1989.pdf`.
- Germain, Mathieu et al. (2015). "MADE: Masked Autoencoder for Distribution Estimation". In: *Proceedings of Machine Learning Research* 37, pp. 881–889. URL: `https://proceedings.mlr.press/v37/germain15.html`.
- Hornik, Kurt (1991). "Approximation Capabilities of Multilayer Feedforward Networks". In: *Neural Networks* 4.2, pp. 251–257. DOI: 10.1016/0893-6080(91)90009-T. URL: `https://web.njit.edu/~usman/courses/cs677_spring21/hornik-nn-1991.pdf`.

References II

- Immer, Alexander et al. (2023). "Effective Bayesian Heteroscedastic Regression with Deep Neural Networks". In: *Advances in Neural Information Processing Systems*. URL: https://papers.nips.cc/paper_files/paper/2023/file/a901d5540789a086ee0881a82211b63d-Paper-Conference.pdf.
- Kingma, Diederik P. and Max Welling (2013). "Auto-Encoding Variational Bayes". In: *arXiv preprint arXiv:1312.6114*. URL: https://arxiv.org/abs/1312.6114.
- Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems*. URL: https://arxiv.org/abs/1612.01474.
- LeCun, Yann et al. (2006). "A Tutorial on Energy-Based Learning". In: Tutorial chapter in Predicting Structured Data (MIT Press). URL: http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf.

References III

- Nix, David A. and Andreas S. Weigend (1994). “Estimating the Mean and Variance of the Target Probability Distribution”. In: *Proceedings of the 1994 IEEE International Conference on Neural Networks*, pp. 55–60. DOI: 10.1109/ICNN.1994.374138. URL: https://ieeexplore.ieee.org/document/374138.
- Papamakarios, George et al. (2021). “Normalizing Flows for Probabilistic Modeling and Inference”. In: *Journal of Machine Learning Research* 22.57, pp. 1–64. URL: https://jmlr.org/papers/volume22/19-1028/19-1028.pdf.
- Sluijterman, Laurens, Eric Cator, and Tom Heskes (2023). “Optimal Training of Mean Variance Estimation Neural Networks”. In: *arXiv preprint arXiv:2302.08875*. URL: https://arxiv.org/pdf/2302.08875v1.
- Stirn, Andrew et al. (2023). “Faithful Heteroscedastic Regression with Neural Networks”. In: *Proceedings of Machine Learning Research* 206. URL: https://proceedings.mlr.press/v206/stirn23a/stirn23a.pdf.