Bayesian Neural Networks and Monte Carlo Dropout

ELG 5218 - Uncertainty Evaluation in Engineering Measurements and Machine Learning

Miodrag Bolic

University of Ottawa

February 25, 2026

Roadmap

What This Lecture Covers (Implementation Focus)

Assumed known: MCMC fundamentals, KL divergence, ELBO derivation, Bayesian basics.

How methods are implemented:

1. BNN training loop and heteroscedastic loss
2. MCMC for BNNs: SGLD, HMC — practical algorithms
3. Variational inference: Bayes-by-Backprop in detail
4. MC Dropout: exact training and inference procedure
5. **Why VI when MC Dropout is so simple?** — decision guide

Key references

Gal & Ghahramani (2016)
Blundell et al. (2015)
Kendall & Gal (2017)
Welling & Teh (2011) — SGLD

From Point Estimates to Weight Distributions

Standard (deterministic) NN

1. Choose loss $\mathcal{L}(\boldsymbol{w})$ (NLL, MSE, ...)
2. Minimise: $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
3. Predict: $\hat{y} = f_{\hat{\boldsymbol{w}}}(x^*)$

$\hat{\boldsymbol{w}}$ is a **single point** in parameter space.

Bayesian neural network (BNN)

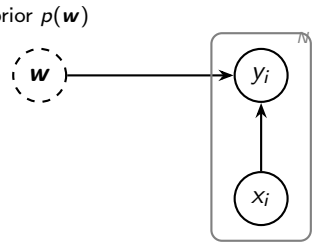
1. Specify prior: $p(\boldsymbol{w})$
2. Observe data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$
3. Infer posterior: $p(\boldsymbol{w} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \boldsymbol{w})\, p(\boldsymbol{w})$
4. Predict: $p(y^* \mid x^*, \mathcal{D}) = \int p(y^* \mid x^*, \boldsymbol{w})\, p(\boldsymbol{w} \mid \mathcal{D})\, \mathrm{d}\boldsymbol{w}$

$p(\boldsymbol{w} \mid \mathcal{D})$ is a **distribution** over parameters.

Physical intuition for an engineer

$p(\boldsymbol{w} \mid \mathcal{D})$ encodes **which model configurations are consistent with the data**. Regions of input space with few training samples $\Rightarrow$ many plausible $\boldsymbol{w}$ $\Rightarrow$ wide predictive distribution.

Graphical Model View of a BNN



Latent (dashed): $\boldsymbol{w}$ — unknown weights
Observed (solid): (x_i, y_i) — data
Plate: replicated over N examples

Bayes' theorem for BNNs

$$\underbrace{p(\boldsymbol{w} \mid \mathcal{D})}_{\text{posterior}} \propto \underbrace{p(\mathcal{D} \mid \boldsymbol{w})}_{\text{likelihood}} \cdot \underbrace{p(\boldsymbol{w})}_{\text{prior}}$$

Likelihood models

Regression: $y_i \mid x_i, \boldsymbol{w} \sim \mathcal{N}\left(f_{\boldsymbol{w}(x_i),\, \sigma^2}\right)$
Classification: $y_i \mid x_i, \boldsymbol{w} \sim \mathrm{Cat}(\mathrm{softmax}(f_{\boldsymbol{w}(x_i)}))$

Common priors

$p(\boldsymbol{w}) = \prod_j \mathcal{N}(w_j,\, 0)\, \tau^2$
(i.i.d. zero-mean Gaussian, equivalent to L2 regularization at MAP)

Posterior Predictive = Bayesian Model Averaging

Posterior predictive distribution

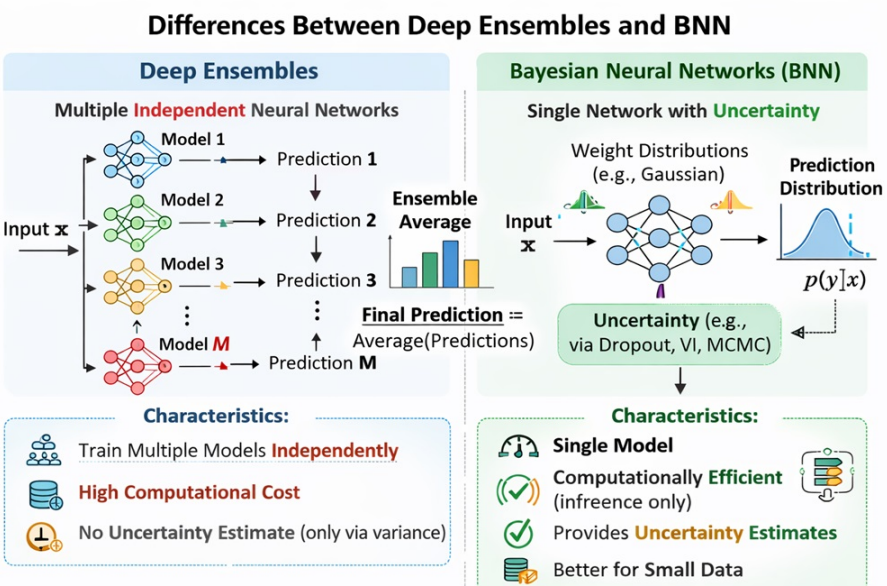
$$p(y^* \mid x^*, \mathcal{D}) = \int p(y^* \mid x^*, \boldsymbol{w})\, p(\boldsymbol{w} \mid \mathcal{D})\, \mathrm{d}\boldsymbol{w} \approx \frac{1}{S} \sum_{s=1}^{S} p(y^* \mid x^*, \boldsymbol{w}^{(s)}), \quad \boldsymbol{w}^{(s)} \sim p(\boldsymbol{w} \mid \mathcal{D})$$

Intuition: each $\boldsymbol{w}^{(s)}$ defines a different network function $f_{\boldsymbol{w}^{(s)}}$. The BNN *averages over many plausible networks*.

Why this differs from an ensemble

In a BNN the samples $\boldsymbol{w}^{(s)}$ come from the *posterior* (data-informed), not from re-training with different random seeds.

Deep Ensembles vs. BNNs



Why the Posterior is Intractable

The posterior requires normalising the likelihood:

$$p(\boldsymbol{w} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{w})\, p(\boldsymbol{w})}{p(\mathcal{D})}, \qquad p(\mathcal{D}) = \int p(\mathcal{D} \mid \boldsymbol{w})\, p(\boldsymbol{w})\, \mathrm{d}\boldsymbol{w}.$$

Practical difficulties

- $\boldsymbol{w} \in \mathbb{R}^d$ for $d =$ thousands–billions in modern nets
- The integral $p(\mathcal{D})$ is computationally intractable (high-dimensional, non-Gaussian)
- The posterior is **multi-modal**: many local minima, complex geometry

Three families of approximations

1. **MCMC** (HMC/NUTS, SGLD): stochastically sample $\boldsymbol{w}^{(s)} \sim p(\boldsymbol{w} \mid \mathcal{D})$ — asymptotically exact, expensive
2. **Variational inference**: optimise a tractable $q_\phi(\boldsymbol{w}) \approx p(\boldsymbol{w} \mid \mathcal{D})$ — fast, biased
3. **MC Dropout**: use dropout masks as an implicit variational distribution — cheapest, good baseline

BNN Training: The Heteroscedastic Loss

Standard regression outputs a single value. A BNN regression head outputs *both* mean and log-variance:

$$\text{output: } (\hat{\mu}_{\boldsymbol{w}}(x),\ \log \hat{\sigma}^2_{\boldsymbol{w}}(x))$$

Heteroscedastic NLL loss

$$\mathcal{L}_{\text{het}} = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{(y_i - \hat{\mu}_{\boldsymbol{w}}(x_i))^2}{2\,e^{s_i}} + \frac{s_i}{2}\right]$$

where $s_i = \log \hat{\sigma}^2_{\boldsymbol{w}}(x_i)$.

Why not MSE? MSE ignores aleatoric noise. Large-noise samples should contribute less — the heteroscedastic loss provides this automatic down-weighting.

PyTorch implementation

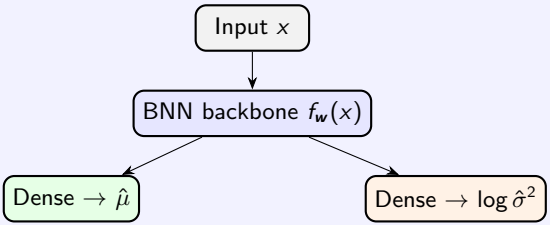
```
def hetero_nll(y, mu, log_var):
    # y, mu, log_var: (B,)
    prec = torch.exp(-log_var)
    return 0.5 * (
        prec * (y - mu)**2
        + log_var
    ).mean()
```

Numerical tip

Predict $\log \hat{\sigma}^2$, not $\hat{\sigma}^2$, to guarantee positivity and avoid gradient explosion.

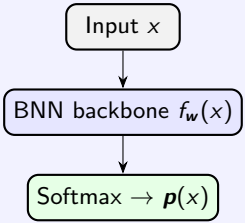
BNN Architecture: Head Design for Uncertainty

Regression: dual-head design



Loss = heteroscedastic NLL.

Classification: softmax head



Loss = cross-entropy.
Uncertainty from MC-averaged softmax, not from the architecture.

Where dropout goes

Dropout is placed *inside* the network (after hidden layers). The heads themselves typically do *not* have dropout so that mean and variance outputs are not further masked.

Stochastic Gradient Langevin Dynamics (SGLD)

HMC requires full-batch gradients and is too slow for large nets. **SGLD** injects Gaussian noise into SGD updates to obtain asymptotically correct posterior samples.

SGLD update rule

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\epsilon_t}{2}\nabla_{\boldsymbol{w}}\tilde{U}(\boldsymbol{w}_t) + \boldsymbol{\eta}_t, \quad \boldsymbol{\eta}_t \sim \mathcal{N}(0,\, \epsilon_t I)$$

where $\tilde{U}(\boldsymbol{w}) = -\frac{N}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \log p(y_i|x_i, \boldsymbol{w}) - \log p(\boldsymbol{w})$ is the stochastic potential (negative log-posterior, mini-batch scaled).

Full SGLD algorithm

Initialise $\boldsymbol{w}_0$; set step schedule $\{\epsilon_t\}$
for $t = 1, 2, \ldots$ **do**
  Sample mini-batch $\mathcal{B} \subset \mathcal{D}$
  Compute stochastic gradient $g_t = \nabla_{\boldsymbol{w}}\tilde{U}(\boldsymbol{w}_t)$
  Sample $\boldsymbol{\eta}_t \sim \mathcal{N}(0,\, \epsilon_t I)$
  $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \frac{\epsilon_t}{2} g_t + \boldsymbol{\eta}_t$
  if $t > t_{\text{burn-in}}$ and $t \bmod K = 0$ **then**
    Save $\boldsymbol{w}_t$ as a posterior sample
  end if
end for

Intuition

SGD = gradient descent.
SGLD = SGD + noise injection.

The noise prevents the chain from collapsing to a mode.
The step size schedule $\epsilon_t \to 0$ ensures the chain converges to $p(\boldsymbol{w}|\mathcal{D})$ rather than oscillating.

Practical notes

- Burn-in: discard first $t_{\text{burn-in}}$ samples (chain not yet mixed)
- Thinning: save every K-th sample to reduce correlation
- Step size matters: too large $\Rightarrow$ inaccurate; too small $\Rightarrow$ slow mixing
- Cyclical SGLD (Zhang et al. 2020) improves sample diversity

SGLD: Posterior Predictive and Inference

After collecting S posterior samples $\{\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(S)}\}$:

Monte Carlo predictive

$$p(y^*|x^*, \mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} p(y^*|x^*, \boldsymbol{w}^{(s)})$$

$$\hat{\mu} = \frac{1}{S}\sum_{s} f_{\boldsymbol{w}^{(s)}}(x^*), \quad \hat{\sigma}^2_{\text{ep}} = \frac{1}{S}\sum_{s}\left(f_{\boldsymbol{w}^{(s)}}(x^*) - \hat{\mu}\right)^2$$

$$\hat{\sigma}^2_{\text{al}}(x^*) = \frac{1}{S}\sum_{s=1}^{S} \sigma^2_{\boldsymbol{w}^{(s)}}(x^*)$$

- **Epistemic** = variance of the means across weight samples: "If I draw different weights from the posterior, how much do the predicted means disagree?"
- **Aleatoric** = mean of the per-sample noise variances: "On average across weight samples, how much irreducible observation noise does the model predict at this point?"

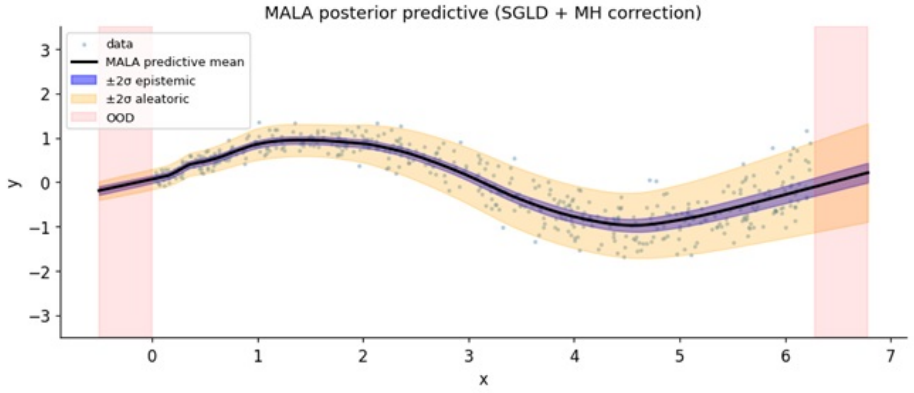
When to use SGLD over MC Dropout

Use SGLD when:

- The model has *no* dropout layers
- You need a benchmark posterior (e.g. for validating VI)
- Dataset is small-to-medium (fit in GPU memory or fast iteration)
- You need asymptotically correct samples

Drawback:
Storage of S full model checkpoints ($\sim S\times$ model size).

SGLD Porterior Predictive Example



Variational Inference: The Optimisation View

Core idea: replace sampling with optimisation. Find $q_\phi(\boldsymbol{w})$ that minimises:

$$D_{\mathrm{KL}}(q_\phi(\boldsymbol{w}) \,\|\, p(\boldsymbol{w}|\mathcal{D})) \;\equiv\; \max_{\phi} \; \mathcal{L}(\phi)$$

where

$$\mathcal{L}(\phi) = \underbrace{\mathbb{E}_{\boldsymbol{w}\sim q_\phi}[\log p(\mathcal{D}|\boldsymbol{w})]}_{\text{expected log-likelihood} \uparrow} - \underbrace{D_{\mathrm{KL}}(q_\phi(\boldsymbol{w}) \,\|\, p(\boldsymbol{w}))}_{\text{KL regulariser} \downarrow}$$

Mean-field Gaussian variational family

$$q_\phi(\boldsymbol{w}) = \prod_j \mathcal{N}(w_j; \mu_j, \sigma_j^2)$$

Each weight w_j has its own learnable (μ_j, σ_j).

Number of parameters: $2d$ instead of d.

Reparameterisation: $w_j = \mu_j + \sigma_j \varepsilon_j, \quad \varepsilon_j \sim \mathcal{N}(0, 1)$

Analytic KL for Gaussian families

$$D_{\mathrm{KL}}(q_\phi \,\|\, p) = \frac{1}{2} \sum_j \left[\frac{\mu_j^2 + \sigma_j^2}{\tau^2} - 1 - \log \frac{\sigma_j^2}{\tau^2} \right]$$

This term is differentiable: gradients flow directly from the KL to (μ_j, σ_j).

Bayes by Backprop (Blundell et al., 2015)

Algorithm

Store mean $\boldsymbol{\mu}$ and (softplus-parameterised) std $\boldsymbol{\sigma} = \mathrm{softplus}(\boldsymbol{\rho})$ for every weight.
Per mini-batch $\mathcal{B}$:

1. Sample $\varepsilon \sim \mathcal{N}(0, I)$; compute $\boldsymbol{w} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \varepsilon$
2. Estimate the stochastic ELBO:

$$\widehat{\mathcal{L}} = \frac{N}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \log p(y_i \mid x_i, \boldsymbol{w}) - D_{\mathrm{KL}}(q_\phi(\boldsymbol{w}) \,\|\, p(\boldsymbol{w}))$$

3. Gradient step on $(\boldsymbol{\mu}, \boldsymbol{\rho})$ via backprop

Bayes by Backprop: Full Training Algorithm

BbB training algorithm (Blundell 2015)

1: **Initialise:** $\boldsymbol{\mu} \leftarrow$ Xavier; $\boldsymbol{\rho} \leftarrow -3$ ($\sigma = \text{softplus}(\rho) \approx 0.05$)
2: **for** each mini-batch $\mathcal{B}$ of size B **do**
3: **Sample** noise: $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, I)$
4: **Sample** weights: $\boldsymbol{w} = \boldsymbol{\mu} + \text{softplus}(\boldsymbol{\rho}) \odot \boldsymbol{\varepsilon}$
5: **Forward pass** with $\boldsymbol{w}$, compute predictions
6: **Compute stochastic ELBO:**

$$\widehat{\mathcal{L}} = \frac{N}{B} \sum_{i \in \mathcal{B}} \log p(y_i | x_i, \boldsymbol{w}) - D_{\mathrm{KL}}(q_\phi \,\|\, p)$$

7: **Backprop** through $\boldsymbol{\mu}$ and $\boldsymbol{\rho}$ (reparameterisation)
8: **Adam/SGD** update on $(\boldsymbol{\mu}, \boldsymbol{\rho})$
9: **end for**
10: **Return:** $q_\phi(\boldsymbol{w}) = \prod_j \mathcal{N}(w_j; \mu_j, \text{softplus}(\rho_j)^2)$

PyTorch BbB linear layer

```
class BayesLinear(nn.Module):
  def __init__(self, in_f, out_f,
               prior_std=1.0):
    super().__init__()
    self.mu  = nn.Parameter(
        torch.zeros(out_f, in_f))
    self.rho = nn.Parameter(
        torch.full((out_f,in_f),-3.))
    self.prior_std = prior_std

  def forward(self, x):
    sigma = F.softplus(self.rho)
    eps   = torch.randn_like(sigma)
    w     = self.mu + sigma * eps
    # KL for this layer
    kl = 0.5*(
      (self.mu**2 + sigma**2)
      / self.prior_std**2
      - 1 - 2*torch.log(sigma)
      + 2*math.log(self.prior_std)
    ).sum()
    return F.linear(x, w), kl
```

BbB: Inference and Practical Pitfalls

Test-time inference

1: **for** $s = 1, \ldots, S$ **do**
2: Sample $\varepsilon \sim \mathcal{N}(0, I)$
3: $\boldsymbol{w}^{(s)} = \boldsymbol{\mu} + \text{softplus}(\boldsymbol{\rho}) \odot \varepsilon$
4: $\hat{y}^{(s)} = f_{\boldsymbol{w}^{(s)}}(x^*)$
5: **end for**
6: $\hat{\mu} = \frac{1}{S} \sum_s \hat{y}^{(s)}$
7: $\hat{\sigma}^2_{\text{ep}} = \frac{1}{S} \sum_s (\hat{y}^{(s)} - \hat{\mu})^2$

KL cost annealing trick

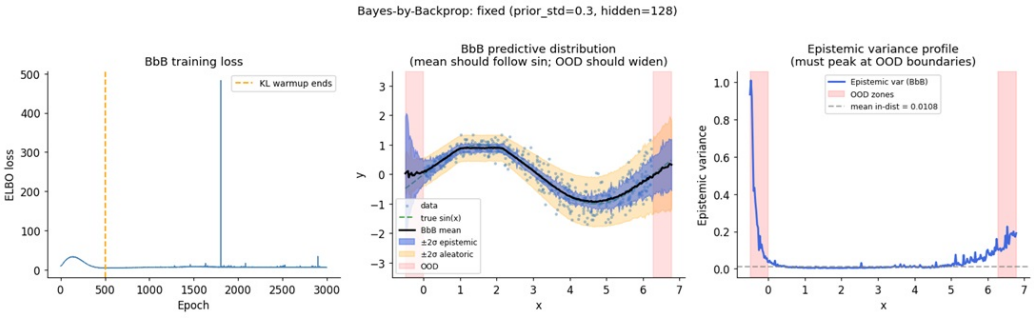
Early training: the KL term can dominate and prevent the model from fitting data (posterior collapse).

Solution: multiply KL by a coefficient β that increases from 0 to 1 over the first T_{warm} epochs.

Common pitfalls

1. **Double parameters**: BbB doubles trainable parameters vs. standard NN — memory cost.
2. **High gradient variance**: single-sample ELBO is noisy. Use local reparameterisation: sample pre-activations $a = x^T \mu + x^T(\sigma \odot \varepsilon)$ instead of $\boldsymbol{w}$.
3. **Prior choice**: τ controls posterior regularisation. Too small $\Rightarrow$ weights shrink; too large $\Rightarrow$ prior is uninformative and KL vanishes.
4. **Initialisation**: start ρ at -3 so $\sigma \approx 0.05$ (small initial uncertainty).

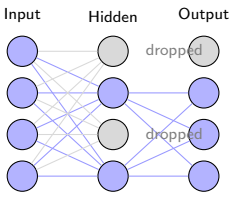
Variational Inference Example



Standard Dropout (Srivastava et al., 2014) — Recap

During training: randomly zero each neuron with probability p:

$$\tilde{h}_j = h_j \cdot m_j, \quad m_j \sim \text{Bernoulli}(1-p)$$



Training:

- Each forward pass uses a different mask $\boldsymbol{m}^{(s)}$
- Acts as a regulariser (implicit ensemble)

Standard test time:

- Dropout *disabled*
- Weights scaled by $(1-p)$
- Returns a **single** prediction

MC Dropout: Keep Dropout On at Test Time

Key idea (Gal & Ghahramani, 2016): leave dropout *enabled* during inference and run T stochastic forward passes.

MC Dropout inference

For test input x^*, sample T masks $\boldsymbol{m}^{(1)}, \ldots, \boldsymbol{m}^{(T)}$:

$$\hat{y}^{(t)} = f_{\hat{\boldsymbol{w}}, \boldsymbol{m}^{(t)}}(x^*), \quad t = 1, \ldots, T$$

- **Predictive mean:** $\hat{\mu}(x^*) = \frac{1}{T} \sum_{t=1}^{T} \hat{y}^{(t)}$
- **Epistemic variance:** $\hat{\sigma}^2_{\text{ep}}(x^*) = \frac{1}{T} \sum_{t=1}^{T} (\hat{y}^{(t)} - \hat{\mu})^2$
- **Total variance** (if model also outputs $\hat{\sigma}^2_{al}$):

$$\hat{\sigma}^2_{\text{total}} = \hat{\sigma}^2_{\text{ep}} + \frac{1}{T} \sum_{t=1}^{T} \hat{\sigma}^{2,(t)}_{\text{al}}$$

Variational Interpretation of MC Dropout

Gal & Ghahramani (2016): dropout training approximately minimises a **variational objective**.

Dropout as VI (key result)

Let $\boldsymbol{m} \sim \text{Bernoulli}(1-p)^d$ be the dropout mask. Define the variational distribution:

$$q(\boldsymbol{w}) = q(\boldsymbol{M}, \boldsymbol{\mu}), \qquad W_l = \text{diag}(\boldsymbol{m}_l) \cdot M_l$$

Then training with dropout $\approx$ minimising:

$$\min_{q(\boldsymbol{w})} \mathbb{E}_{\boldsymbol{w}\sim q}[-\log p(\mathcal{D} \mid \boldsymbol{w})] + D_{\text{KL}}(q(\boldsymbol{w}) \,\|\, p(\boldsymbol{w}))$$

Practical upshot

Each dropout mask $\boldsymbol{m}^{(t)}$ is a **sample** from an implicit approximate posterior over weights.

MC averaging over T masks $\approx$ Bayesian model averaging.

Important caveats

- The variational family is **restrictive**
- Uncertainty quality depends on dropout rate p, placement, and T
- May underestimate or misrepresent uncertainty for some problems

MC Dropout: Exact Training Procedure

Training (regression with aleatoric noise)

1: Build network with **Dropout(p)** after each hidden layer
2: Output head: two scalars $(\hat{\mu}, \log \hat{\sigma}^2)$ per sample
3: **for** each mini-batch $\mathcal{B}$ **do**
4: Forward pass **(dropout is active by default)**
5: Compute heteroscedastic NLL:

$$\mathcal{L} = \frac{1}{B} \sum_{i \in \mathcal{B}} \left[\frac{(y_i - \hat{\mu}_i)^2}{2e^{s_i}} + \frac{s_i}{2} \right]$$

where $s_i = \log \hat{\sigma}_i^2$
6: Backprop + Adam/SGD on $\hat{\boldsymbol{w}}$ (point estimate)
7: **end for**
8: **No changes to training vs. standard NN**

MC Dropout: Full Inference Procedure

Algorithm (regression)

Training: standard dropout + NLL loss (no changes needed)

Test-time inference:

for $t = 1, \ldots, T$ **do**
    sample mask $\boldsymbol{m}^{(t)} \sim \mathrm{Ber}(1-p)^d$
    $\hat{y}^{(t)}, \hat{\sigma}^{2,(t)} \leftarrow f_{\hat{\boldsymbol{w}}, \boldsymbol{m}^{(t)}}(x^*)$
end for
$\hat{\mu} \leftarrow \frac{1}{T} \sum_t \hat{y}^{(t)}$
$\hat{\sigma}^2_{\text{ep}} \leftarrow \frac{1}{T} \sum_t (\hat{y}^{(t)} - \hat{\mu})^2$
$\hat{\sigma}^2_{\text{al}} \leftarrow \frac{1}{T} \sum_t \hat{\sigma}^{2,(t)}$
$\hat{\sigma}^2_{\text{total}} \leftarrow \hat{\sigma}^2_{\text{ep}} + \hat{\sigma}^2_{\text{al}}$

PyTorch one-liner

Enable dropout at test:

```
model.eval()
for m in model.modules():
if isinstance(m, nn.Dropout):
m.train()
```

Then loop T forward passes.

Choosing T

$T \in [20, 100]$ is typically sufficient.
Plot variance-of-variance vs T to diagnose convergence.
T=50 is a sensible default.

Why VI (Bayes-by-Backprop) When MC Dropout Exists?

MC Dropout requires no architecture changes — so why would anyone do VI?

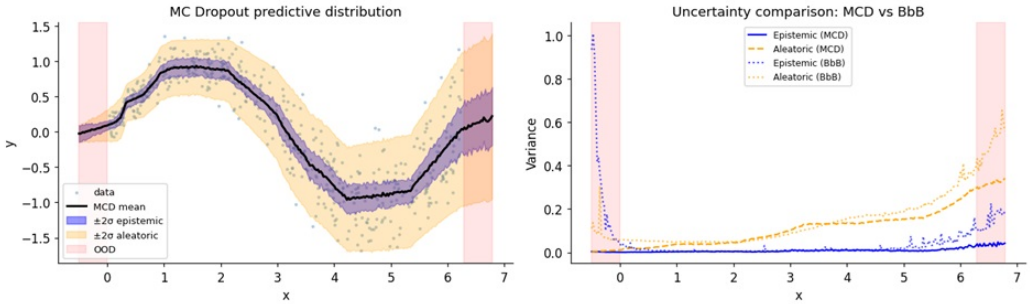
MC Dropout strengths

- **Zero training overhead**: retrofit onto any existing model
- **Same parameter count**: no memory increase
- **No special layers needed**
- **Works for any architecture**: CNNs, RNNs, Transformers
- Inference cost: $T\times$ one forward pass

MC Dropout limitations

- Restricted variational family: Bernoulli masks only
- Dropout rate p is a fixed hyperparameter — may be wrong
- Uncertainty quality depends sensitively on p and placement
- Cannot capture *weight correlations* across layers
- Often **underestimates epistemic uncertainty** on OOD data

MC Dropout Predictive distribution

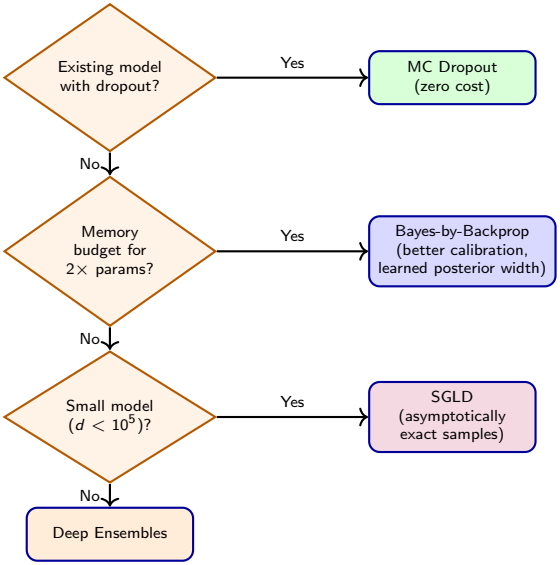


- **BbB epistemic** (blue dotted): large, peaks sharply at both OOD boundaries — *rich posterior*.
- **MCD epistemic** (blue solid): small, barely rises at OOD — *coarser posterior approximation*.
- **Both aleatoric curves** (orange): similar shape, growing rightward — consistent learned noise, as expected since both methods use the same heteroscedastic NLL loss.

Takeaway

MC Dropout trades *posterior richness* for *simplicity*: no extra parameters, no KL term, easy to retrofit to any existing network — but at the cost of less expressive uncertainty estimates, particularly for OOD detection.

MC Dropout vs. BbB vs. SGLD: Decision Guide



Uncertainty Quality: An Honest Comparison

Property	MC Dropout	BbB	SGLD	Ensembles
Training cost	$1\times$	$1.5\times$	$1\times$ (long)	$M\times$
Parameter overhead	None	$2\times$	None (checkpoints)	$M\times$
Inference cost	$T\times$ fwd	$S\times$ fwd	$S\times$ fwd	$M\times$ fwd
Posterior quality	Approximate	Approximate	Asymptotic	Empirical
Calibration	Moderate	Good	Best	Good
OOD detection	Good	Good	Good	Better
Architecture changes	None	Yes (layers)	None	None
Easy to tune	Moderate	Hard	Hard	Easy

When MC Dropout fails

Long-range correlations in weight posterior (deep, narrow networks); very small dropout rates effectively give nearly-deterministic models.

Methods of Approximate Inference for BNNs

MCMC

HMC / NUTS / SG-MCMC

Define prior $p(\mathbf{w})$ and likelihood $p(\mathcal{D} \mid \mathbf{w})$

Run Markov chain targeting posterior $p(\mathbf{w} \mid \mathcal{D})$

Keep posterior samples $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(S)}$

Predict at x^* using posterior samples

Average predictions + quantify spread (credible intervals)

Sampling-based (high fidelity, expensive)

Variational Inference

Bayes by Backprop

Choose family $q_\phi(\mathbf{w})$ (e.g., factorized Gaussian)

Optimise ELBO w.r.t. ϕ via gradients + reparam.

Obtain approximate posterior $q_{\phi^*}(\mathbf{w})$

Sample $\mathbf{w} \sim q_{\phi^*}(\mathbf{w})$ and predict on x^*

MC average + uncertainty from $q_{\phi^*}(\mathbf{w})$ samples

Optimisation-based (scalable, can be biased)

MC Dropout

Dropout as approx. Bayesian inference

Train network with dropout (+ weight decay)

At test time keep dropout ON

Run T stochastic forward passes with different dropout masks

Collect $\hat{y}_1, \ldots, \hat{y}_T$ for x^*

Estimate predictive mean and variance from samples

Practical stochastic approx. (easy to retrofit to NNs)

Target quantity: predictive distribution $p(y^* \mid x^*, \mathcal{D})$

Practical Guidelines: SGLD, BbB, and MC Dropout

SGLD

Architecture

- ReLU activations (not Tanh)
- Separate mean and noise heads

Training

- Phase 1: Adam warm-up (∼600 epochs) to MAP
- Phase 2: SGLD sampling step size $\eta \approx 5 \times 10^{-6}$
- Prior std $= 1.0$
- Gradient: scale by N, add prior term $\mathbf{w}/\sigma_p^2$

Sampling

- Burn-in $\geq$ 500 epochs
- Thin every 5–10 epochs
- Collect $S \geq 50$ samples

Key risk

Too large $\eta \Rightarrow$ random walk, too small $\eta \Rightarrow$ no mixing

Bayes by Backprop

Architecture

- ReLU activations (not Tanh)
- hidden $= 128$

Initialisation

- $\boldsymbol{\mu}$: Xavier uniform
- $\boldsymbol{\rho} = -3.0$ $\Rightarrow \sigma = \mathrm{softplus}(-3) \approx 0.05$
- Prior std $= 1.0$

Training

- KL warmup: linear ramp with $\beta_{\min} = 0.01$ floor over 1500 epochs
- $\eta_{\min} = 10^{-4}$ (cosine LR)
- Gradient clip $= 5.0$

Key risk

Posterior collapse if KL engages before NLL converges

MC Dropout

Architecture

- ReLU activations (not Tanh)
- hidden $= 128$, $p = 0.3$
- Dropout after each hidden layer

Training

- Standard NLL (no KL term)
- Weight decay $= 10^{-4}$ (acts as Gaussian prior)
- Gradient clip $= 5.0$
- Cosine LR, $\eta_{\min} = 10^{-4}$

Inference

- $T \geq 200$ stochastic passes

Key risk

Lower epistemic uncertainty than BbB; weaker OOD detection by design

Diagnosing Approximate Inference: Did It Work?

Approximate inference can *look* plausible while being systematically wrong.

Five mandatory sanity checks

1. **Epistemic uncertainty shrinks with data:** Hold out x^*; retrain with more data nearby; verify $\hat{\sigma}^2_{\text{ep}}(x^*)$ decreases.
2. **Uncertainty grows away from data:** Evaluate on OOD inputs (e.g. unseen arrhythmia class); epistemic uncertainty should be higher than in-distribution.
3. **Posterior samples are diverse:** Plot T stochastic forward passes; they should differ visibly (if they collapse to one line, the posterior has collapsed).
4. **Prior sensitivity:** Try $\tau \in \{0.1, 1.0, 10.0\}$; results should be qualitatively robust.
5. **NLL on held-out set:** Better UQ method $\equiv$ lower predictive NLL (a proper scoring rule; accuracy alone is not sufficient).

Summary

Implementation take-aways

1. **Training**: use heteroscedastic NLL loss with two output heads $(\hat{\mu}, \log \hat{\sigma}^2)$ to learn both epistemic (via MC dropout/BbB) and aleatoric (via learned variance) uncertainty.
2. **SGLD**: inject Gaussian noise proportional to $\sqrt{\epsilon_t}$ into SGD updates; discard burn-in; asymptotically exact but expensive.
3. **Bayes-by-Backprop**: store $(\boldsymbol{\mu}, \boldsymbol{\rho})$ per weight; reparameterise; maximise stochastic ELBO with analytic KL; doubles parameter count.
4. **MC Dropout**: cheapest retrofit; keep dropout on at test time with `model.eval()+enable_dropout()`; $T = 50$ passes; freeze BatchNorm.
5. **Choice rule**: MCD first; BbB if you can redesign; SGLD for benchmarks; Ensembles for safety-critical.
6. **Biomedical applications**: ECG OOD arrhythmia detection, PPG SpO_2 noise decomposition, EEG active annotation, EMG sensor drift alerting.

Key References

- **Gal & Ghahramani (2016):** "Dropout as a Bayesian Approximation" — MC Dropout theory.
- **Blundell et al. (2015):** "Weight Uncertainty in Neural Networks" — Bayes-by-Backprop.
- **Kendall & Gal (2017):** "What Uncertainties Do We Need?" — epistemic/aleatoric decomposition.
- **Welling & Teh (2011):** "Bayesian Learning via Stochastic Gradient Langevin Dynamics" — SGLD.
- **Lakshminarayanan et al. (2017):** "Simple and Scalable Predictive Uncertainty using Deep Ensembles."
- **Zhang et al. (2020):** "Cyclical Stochastic Gradient MCMC for Bayesian Deep Learning."
- **Ovadia et al. (2019):** "Can You Trust Your Model's Uncertainty?" — benchmarks under dataset shift.
- **Gawlikowski et al. (2023):** "A Survey of Uncertainty in Deep Neural Networks."

Questions?