Bayesian Deep Learning

Modern Techniques for Quantifying Uncertainty

Andrija Petrović

August 7, 2025

Outline

Standard vs. Bayesian Deep Learning

Standard Deep Learning

- We learn a single set of optimal weights $\mathbf{W}^*$ (a **point estimate**).
- Training minimizes a loss function (e.g., MSE, Cross-Entropy).
- Given an input $\mathbf{x}$, the network produces a single, deterministic output $y = f(\mathbf{x}; \mathbf{W}^*)$.
- **It cannot tell us how confident it is in its prediction.**

Bayesian Deep Learning (BDL)

- We learn a **distribution** over weights, the posterior $p(\mathbf{W} \mid \mathcal{D})$.
- Training is a process of inferring this posterior distribution.
- Given an input $\mathbf{x}$, we get a **distribution of outputs** by averaging over all possible networks.
- **It naturally quantifies uncertainty.**

The Two Types of Uncertainty

A key advantage of the Bayesian approach is the ability to decompose uncertainty.

Epistemic Uncertainty

(Uncertainty in the model)

- Uncertainty due to our ignorance about the true model parameters $\mathbf{W}$.
- Also known as "model uncertainty".
- It is **reducible** by collecting more data, especially in regions of high uncertainty.
- Captured by the posterior distribution over weights, $p(\mathbf{W} \mid \mathcal{D})$.

Aleatoric Uncertainty

(Uncertainty in the data)

- Uncertainty from inherent randomness or noise in the data generating process.
- Also known as "data uncertainty".
- It is **irreducible**, even with infinite data (e.g., sensor noise, overlapping classes).
- Captured by the probabilistic likelihood function, e.g., the variance term in $p(y|\mathbf{x}, \mathbf{W}) = \mathcal{N}(y; f(\mathbf{x}; \mathbf{W}), \sigma^2)$.

The Two Types of Uncertainty

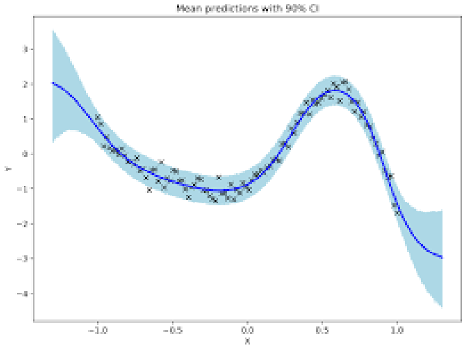

Figure: A BNN's prediction. The wide shaded area far from data is high epistemic uncertainty. The minimum width of the shaded area over data is the aleatoric uncertainty.

Bayesian Linear Regression: A Tractable Case

Let's start with a model where the posterior is tractable to build intuition.

Model Definition

- **Likelihood:** We assume the data is generated by a linear function with Gaussian noise.

$$p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y \mid \mathbf{w}^T \mathbf{x}, \sigma_n^2)$$

Bayesian Linear Regression: A Tractable Case

Let's start with a model where the posterior is tractable to build intuition.

Model Definition

- **Likelihood:** We assume the data is generated by a linear function with Gaussian noise.

$$p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y \mid \mathbf{w}^T \mathbf{x}, \sigma_n^2)$$

- **Prior:** We place a zero-mean Gaussian prior on the weights $\mathbf{w}$. This acts as a regularizer, preferring simpler models (smaller weights).

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \sigma_p^2 \mathbf{I})$$

Bayesian Linear Regression: A Tractable Case

Let's start with a model where the posterior is tractable to build intuition.

Model Definition

- **Likelihood:** We assume the data is generated by a linear function with Gaussian noise.

$$p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y \mid \mathbf{w}^T \mathbf{x}, \sigma_n^2)$$

- **Prior:** We place a zero-mean Gaussian prior on the weights $\mathbf{w}$. This acts as a regularizer, preferring simpler models (smaller weights).

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \sigma_p^2 \mathbf{I})$$

Tractable Posterior

Because the prior and likelihood are conjugate (both are Gaussian), the posterior $p(\mathbf{w} \mid \mathcal{D})$ is also a Gaussian distribution, and we can compute its parameters (mean and covariance) analytically.

Predictions and the Posterior Predictive Distribution

To make a prediction for a new input $\mathbf{x}^*$, we don't use a single weight vector. We average the predictions of all possible weight vectors, weighted by their posterior probability.

Posterior Predictive Distribution

$$p(y^* \mid \mathbf{x}^*, \mathcal{D}) = \int \underbrace{p(y^* \mid \mathbf{x}^*, \mathbf{w})}_{\text{Likelihood}} \underbrace{p(\mathbf{w} \mid \mathcal{D})}_{\text{Posterior}} \, d\mathbf{w}$$

Predictions and the Posterior Predictive Distribution

To make a prediction for a new input $\mathbf{x}^*$, we don't use a single weight vector. We average the predictions of all possible weight vectors, weighted by their posterior probability.

Posterior Predictive Distribution

$$p(y^* \mid \mathbf{x}^*, \mathcal{D}) = \int \underbrace{p(y^* \mid \mathbf{x}^*, \mathbf{w})}_{\text{Likelihood}} \, \underbrace{p(\mathbf{w} \mid \mathcal{D})}_{\text{Posterior}} \, d\mathbf{w}$$

- For linear regression, this integral is tractable. The result is also a Gaussian distribution: $p(y^* \mid \mathbf{x}^*, \mathcal{D}) = \mathcal{N}(y^* \mid \mu_{\text{pred}}, \sigma^2_{\text{pred}})$.
- The mean μ_{pred} is the model's prediction.
- The variance σ^2_{pred} is our total uncertainty, which decomposes into:
  1. Aleatoric uncertainty (from data noise σ_n^2).
  2. Epistemic uncertainty (from our uncertainty in the weights $\mathbf{w}$).

This is precisely the behavior we want to replicate in deep neural networks.

VI for Bayesian Neural Networks (BNNs)

When we move to non-linear functions (neural networks), the posterior $p(\mathbf{W} \mid \mathcal{D})$ becomes intractable.

Solution: Variational Inference

We approximate the true posterior $p(\mathbf{W} \mid \mathcal{D})$ with a simpler, parameterized distribution $q(\mathbf{W}; \boldsymbol{\lambda})$ and optimize its parameters $\boldsymbol{\lambda}$ by maximizing the ELBO.

Recall the ELBO in its "reconstruction + regularization" form:

$$\mathcal{L}(\boldsymbol{\lambda}) = \underbrace{\mathbb{E}_{q(\mathbf{W};\boldsymbol{\lambda})}\left[\log p(\mathcal{D} \mid \mathbf{W})\right]}_{\text{Expected Log Likelihood}} - \underbrace{\mathrm{KL}(q(\mathbf{W}; \boldsymbol{\lambda}) \parallel p(\mathbf{W}))}_{\text{Complexity Cost}}$$

- **Expected Log Likelihood:** Encourages the model to fit the data well.
- **KL Divergence:** Acts as a regularizer, preventing the approximate posterior q from becoming too complex and straying far from our prior beliefs $p(\mathbf{W})$.

Approximation 1: Mean-Field VI

The simplest and most common choice for the variational family q is the **mean-field** (fully-factorized) Gaussian distribution.

The Mean-Field Assumption

We assume all weights w_i are independent in the posterior.

$$q(\mathbf{W};\boldsymbol{\lambda}) = \prod_i q(w_i;\lambda_i) = \prod_i \mathcal{N}(w_i \mid \mu_i, \sigma_i^2)$$

- Instead of one value per weight, we now learn **two**: a mean μ_i and a variance σ_i^2.
- The variational parameters are $\boldsymbol{\lambda} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$.
- **Problem:** This assumption is often too restrictive. In reality, weights in a network are correlated. Mean-field VI cannot model these correlations, often leading to underestimated uncertainty.

The Key: The Reparameterization Trick

To optimize the ELBO using stochastic gradients, we need a low-variance, differentiable estimator. The reparameterization trick is the solution.

The Core Idea

We re-express the sampling process to separate the source of randomness from the parameters we want to optimize.

For a Gaussian weight $w_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$:

1. **Sample** a noise variable ϵ_i from a fixed, parameter-free distribution:

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

2. **Compute** the weight w_i via a deterministic, differentiable transformation:

$$w_i = \mu_i + \sigma_i \cdot \epsilon_i$$

The "Bayes by Backprop" Algorithm

Training Loop (Single Step)

For a minibatch of data $\mathcal{D}_{\text{batch}}$:

1. Sample a noise vector $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. One ϵ_i for each weight.
2. Construct a single instance of the network's weights using the reparameterization trick:

$$\mathbf{W} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$$

   (Note: We often optimize $\log \sigma$ for numerical stability, so $w_i = \mu_i + \exp(\log \sigma_i) \cdot \epsilon_i$).
3. Compute the ELBO estimate using this sampled network $\mathbf{W}$:

$$\widehat{\mathcal{L}} = \log p(\mathcal{D}_{\text{batch}} \,|\, \mathbf{W}) - (\log q(\mathbf{W}; \boldsymbol{\mu}, \boldsymbol{\sigma}) - \log p(\mathbf{W}))$$

   The term in parentheses is the KL divergence, which is often analytic for Gaussians.
4. Compute the gradients of $-\widehat{\mathcal{L}}$ w.r.t. the variational parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ using standard backpropagation.
5. Update $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ using an optimizer like Adam.

Beyond Mean-Field: More Expressive Posteriors

The mean-field assumption is a major limitation. Modern BDL research explores more expressive variational families to better capture posterior correlations.

Matrix-Variate Gaussian

Instead of factorizing over every weight, we can factorize over layers. The weights $\mathbf{W}_l$ of a layer are modeled with a full-covariance Gaussian.

$$q(\mathbf{W}_l) = \mathcal{N}(\mathbf{W}_l; \mathbf{M}_l, \mathbf{U}_l, \mathbf{V}_l)$$

This can capture correlations between weights within the same layer.

Normalizing Flows

We can use a normalizing flow to transform a simple base distribution (like a mean-field Gaussian) into a highly complex, multi-modal posterior approximation.

$$q(\mathbf{W}) = f_K \circ \cdots \circ f_1(q_0(\mathbf{W}_0))$$

This offers maximum flexibility at the cost of higher computational complexity.

These methods provide better uncertainty estimates but come with increased computational cost and implementation complexity.

Alternative 1: MC Dropout

A surprisingly effective and simple technique for getting uncertainty from standard networks.

The Idea (Gal Ghahramani, 2016)

Training a standard neural network with dropout is approximately equivalent to performing variational inference on a specific type of BNN.

The Algorithm:

1. Train a standard neural network with dropout layers as usual.

Alternative 1: MC Dropout

A surprisingly effective and simple technique for getting uncertainty from standard networks.

The Idea (Gal Ghahramani, 2016)

Training a standard neural network with dropout is approximately equivalent to performing variational inference on a specific type of BNN.

The Algorithm:

1. Train a standard neural network with dropout layers as usual.
2. At test time, **do not turn off dropout**. Keep it active.

Alternative 1: MC Dropout

A surprisingly effective and simple technique for getting uncertainty from standard networks.

The Idea (Gal Ghahramani, 2016)

Training a standard neural network with dropout is approximately equivalent to performing variational inference on a specific type of BNN.

The Algorithm:

1. Train a standard neural network with dropout layers as usual.
2. At test time, **do not turn off dropout**. Keep it active.
3. For a given input $\mathbf{x}^*$, perform T stochastic forward passes, each with a different dropout mask.

$$y^{*(t)} = f_{\text{dropout}}(\mathbf{x}^*; \mathbf{W}) \quad \text{for } t = 1, \ldots, T$$

Alternative 1: MC Dropout

A surprisingly effective and simple technique for getting uncertainty from standard networks.

The Idea (Gal Ghahramani, 2016)

Training a standard neural network with dropout is approximately equivalent to performing variational inference on a specific type of BNN.

The Algorithm:

1. Train a standard neural network with dropout layers as usual.
2. At test time, **do not turn off dropout**. Keep it active.
3. For a given input $\mathbf{x}^*$, perform T stochastic forward passes, each with a different dropout mask.
$$y^{*(t)} = f_{\text{dropout}}(\mathbf{x}^*; \mathbf{W}) \quad \text{for } t = 1, \ldots, T$$
4. The set of predictions $\{y^{*(1)}, \ldots, y^{*(T)}\}$ gives an approximation of the predictive distribution.

Alternative 1: MC Dropout

Pros & Cons

Pros: Extremely easy to implement, minimal computational overhead. **Cons:** The quality of uncertainty can be lower than other methods; theoretical justification relies on strong assumptions.

Alternative 2: Deep Ensembles

A simple, robust, and often state-of-the-art baseline for uncertainty estimation.

The Idea (Lakshminarayanan et al., 2017)

Train multiple identical networks and average their predictions. The disagreement between models is a measure of uncertainty.

The Algorithm:

1. Initialize M identical neural networks (e.g., $M = 5$) with different random seeds.

Alternative 2: Deep Ensembles

A simple, robust, and often state-of-the-art baseline for uncertainty estimation.

The Idea (Lakshminarayanan et al., 2017)

Train multiple identical networks and average their predictions. The disagreement between models is a measure of uncertainty.

The Algorithm:

1. Initialize M identical neural networks (e.g., $M = 5$) with different random seeds.
2. Train each of the M networks independently on the full training dataset until convergence.

Alternative 2: Deep Ensembles

A simple, robust, and often state-of-the-art baseline for uncertainty estimation.

The Idea (Lakshminarayanan et al., 2017)

Train multiple identical networks and average their predictions. The disagreement between models is a measure of uncertainty.

The Algorithm:

1. Initialize M identical neural networks (e.g., $M = 5$) with different random seeds.
2. Train each of the M networks independently on the full training dataset until convergence.
3. At test time, get a prediction from each network in the ensemble.

$$y^{*(m)} = f(\mathbf{x}^*; \mathbf{W}_m) \quad \text{for } m = 1, \ldots, M$$

Alternative 2: Deep Ensembles

A simple, robust, and often state-of-the-art baseline for uncertainty estimation.

The Idea (Lakshminarayanan et al., 2017)

Train multiple identical networks and average their predictions. The disagreement between models is a measure of uncertainty.

The Algorithm:

1. Initialize M identical neural networks (e.g., $M = 5$) with different random seeds.
2. Train each of the M networks independently on the full training dataset until convergence.
3. At test time, get a prediction from each network in the ensemble.

$$y^{*(m)} = f(\mathbf{x}^*; \mathbf{W}_m) \quad \text{for } m = 1, \dots, M$$

4. The predictive mean is the average of the individual predictions. The predictive variance is the variance of the predictions.

Alternative 2: Deep Ensembles

Pros & Cons

Pros: Simple, highly parallelizable, often produces the best-quality uncertainty estimates.
Cons: High computational cost (training and storing M models), memory intensive.

Alternative 3: Laplace Approximation

A classic method for approximate inference, recently revitalized for deep learning.

The Idea

Approximate the posterior distribution $p(\mathbf{W} \mid \mathcal{D})$ with a Gaussian centered at the mode of the posterior.

The Algorithm:

1. **Find the MAP estimate:** First, train a standard network with L2 regularization (weight decay). This finds a Maximum A Posteriori (MAP) solution $\mathbf{W}^*$.

Alternative 3: Laplace Approximation

A classic method for approximate inference, recently revitalized for deep learning.

The Idea

Approximate the posterior distribution $p(\mathbf{W} \mid \mathcal{D})$ with a Gaussian centered at the mode of the posterior.

The Algorithm:

1. **Find the MAP estimate:** First, train a standard network with L2 regularization (weight decay). This finds a Maximum A Posteriori (MAP) solution $\mathbf{W}^*$.
2. **Approximate the curvature:** The posterior is approximated as a Gaussian $\mathcal{N}(\mathbf{W} \mid \mathbf{W}^*, \mathbf{H}^{-1})$, where $\mathbf{H}$ is the Hessian (second derivative matrix) of the negative log-posterior evaluated at $\mathbf{W}^*$.

Alternative 3: Laplace Approximation

A classic method for approximate inference, recently revitalized for deep learning.

The Idea

Approximate the posterior distribution $p(\mathbf{W} \mid \mathcal{D})$ with a Gaussian centered at the mode of the posterior.

The Algorithm:

1. **Find the MAP estimate:** First, train a standard network with L2 regularization (weight decay). This finds a Maximum A Posteriori (MAP) solution $\mathbf{W}^*$.
2. **Approximate the curvature:** The posterior is approximated as a Gaussian $\mathcal{N}(\mathbf{W} \mid \mathbf{W}^*, \mathbf{H}^{-1})$, where $\mathbf{H}$ is the Hessian (second derivative matrix) of the negative log-posterior evaluated at $\mathbf{W}^*$.
3. The full Hessian is intractably large ($|\mathbf{W}| \times |\mathbf{W}|$). Practical methods use efficient approximations, like a diagonal or Kronecker-factored (K-FAC) Hessian.

Alternative 3: Laplace Approximation

Pros & Cons

Pros: Post-hoc method (applied after standard training), single forward pass for mean prediction, theoretically grounded. **Cons:** Relies on a Gaussian approximation which may fail for complex posteriors, Hessian approximation can be complex.

How Do We Evaluate Uncertainty?

A good uncertainty estimate should be **calibrated**.

Calibration

A model is perfectly calibrated if its predicted confidence matches its empirical accuracy. For example, for all predictions where the model assigned 80% confidence, it should be correct 80% of the time.

Metrics: Expected Calibration Error (ECE) measures the average gap between confidence and accuracy. Lower is better.

Application 1: Out-of-Distribution (OOD) Detection

BNNs can identify when they are given an input that is fundamentally different from the training data.

- **In-Distribution (ID):** The model has seen similar data. It should make a confident prediction (low epistemic uncertainty).
- **Out-of-Distribution (OOD):** The model has never seen this type of data. It should be highly uncertain (high epistemic uncertainty).

This is critical for safety: a medical AI should say "I don't know" rather than misdiagnose a rare disease it wasn't trained on.

Application 2: Active Learning

Uncertainty can guide data collection to make training more efficient.

The Active Learning Cycle

1. Train a BNN on a small initial labeled dataset.
2. Use the trained BNN to make predictions on a large pool of unlabeled data.
3. Identify the data points where the model is **most uncertain**.
4. Query a human expert (an "oracle") to provide labels for these points.
5. Add the newly labeled data to the training set and repeat the cycle.

This allows the model to achieve high performance with significantly fewer labeled examples compared to random sampling.

Comparison of BDL Methods

Method	**Uncertainty Quality**	**Train Speed**	**Test Speed**	**Implementation**
Mean-Field VI	Medium	Slow	Slow (sampling)	Medium
MC Dropout	Medium-Low	Fast	Slow (sampling)	**Easy**
Deep Ensembles	**High**	**Very Slow** (M models)	Medium (M passes)	Easy
Laplace Approx.	High-Medium	Fast (post-hoc)	**Fast**	Hard (Hessian)

- **No single best method:** The choice depends on the trade-off between performance, computational budget, and implementation effort.
- **Deep Ensembles** are often the strongest (but most expensive) practical baseline.
- **MC Dropout** is a great starting point due to its simplicity.
- **VI and Laplace** offer a more principled Bayesian approach with different computational trade-offs.

Thank You

Questions?
andrijaster@gmail.com