`bensemble`: A Python Library for Unified Bayesian Ensembling

Sobolevsky Fedor, Nabiev Mukhammadsharif, Vasilenko Dmitry, Kasyuk Vadim

Abstract

In this technical report, we introduce `bensemble`, a Python library designed for approximating the posterior distribution of neural network weights to facilitate Bayesian ensembling. Standard deep learning models often yield overconfident predictions. Our library addresses this by providing a unified, PyTorch-compatible interface for four families of Bayesian ensembling methods: Variational Inference (with Local Reparameterization), Variational Rényi, Probabilistic Backpropagation, and Laplace Approximation. `bensemble` operates on the principle that stochastic weight distributions act as generators for diverse model ensembles. It decouples the *inference* phase (learning an approximation $q_\theta(w)$) from the *prediction* phase (sampling deterministic members for Bayesian Model Averaging). We describe the mathematical formulation confirmed by our implementation, detail the API design, and demonstrate the library's capabilities on benchmarks focusing on calibration and adversarial robustness.

1 Introduction

Standard neural networks typically optimize a point estimate of weights, which fails to capture epistemic uncertainty. Bayesian ensembling methods offer a principled alternative by placing a probability distribution over model weights. Formally, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, we seek the posterior distribution $p(w \mid \mathcal{D}) \propto p(\mathcal{D} \mid w)p(w)$.

While exact inference is intractable, approximate methods allow us to learn a surrogate distribution $q_\theta(w)$. The core philosophy of `bensemble` is to treat this approximated posterior as a source from which we sample an ensemble of deterministic models $\{f_{w_k}\}_{k=1}^K$.

1.1 Contributions and Library Goals

We propose `bensemble` to standardize the workflow of probabilistic deep learning.

- **Proposed Solution:** A unified API that abstracts the complexity of variational, ADF-based, and Laplace approximations, allowing users to convert standard PyTorch models into Bayesian ensembles.
- **Differentiation:** `bensemble` implements four distinct approximation families within a shared "fit-sample-predict" interface. We explicitly distinguish between methods using Local Reparameterization (VI) and those using Weight Perturbation (VR), as well as online ADF methods (PBP).
- **Assurance:** The library ensures reproducible uncertainty estimation by decomposing predictive variance into aleatoric and epistemic components and provides sensible defaults for hyperparameters derived from extensive testing.

2 Library Design and Interface

The architecture is built around the `BaseBayesianEnsemble` abstract class.

2.1 Unified Workflow

Inference (Fitting): The `fit` method optimizes the parameters θ of $q_\theta(w)$.

- For **VI/VR**, this uses gradient descent on variational bounds.
- For **PBP**, this uses Assumed Density Filtering (ADF) without standard backpropagation.
- For **Laplace**, this involves MAP training followed by curvature estimation.

Ensemble Generation (Sampling): The `sample_models(n)` method draws K independent weight configurations from $q_\theta(w)$ and instantiates them as standard `nn.Module` objects. This step "freezes" the epistemic uncertainty into a concrete ensemble.

Bayesian Model Averaging (Prediction): The `predict` method estimates the predictive distribution via Monte Carlo integration over the sampled ensemble.

3 Approximation Algorithms

3.1 Variational Inference (VI)

We adopt the practical variational inference framework proposed by Graves [1], approximating the posterior with a diagonal Gaussian $q_\theta(w)$. The objective is to minimize the variational free energy, often interpreted as a Minimum Description Length (MDL) cost:

$$\mathcal{L}(\theta) = \underbrace{\mathbb{E}_{q_\theta(w)}[\mathcal{L}_\mathcal{D}(w)]}_{\text{Error Cost}} + \underbrace{\mathrm{KL}(q_\theta(w)\|p(w))}_{\text{Complexity Cost}}, \tag{1}$$

where $\mathcal{L}_\mathcal{D}$ is the negative log-likelihood.

To optimize the Error Cost efficiently, we employ the Local Reparameterization Trick (LRT) [2]. As noted by Graves, sampling weights directly introduces high variance in gradient estimates. LRT addresses this by sampling pre-activations instead. For a linear layer with inputs X, weight means M, and variances V, the pre-activation Γ is distributed as:

$$\Gamma \sim \mathcal{N}(XM^T, X^2V^T). \tag{2}$$

We sample $\zeta = XM^T + \epsilon \odot \sqrt{X^2V^T}$, where $\epsilon \sim \mathcal{N}(0, I)$, allowing for stable, low-variance backpropagation. The Complexity Cost is computed analytically as the sum of KL divergences for each weight.

3.2 Variational Rényi (VR)

For VR [5], we implement a Weight Perturbation approach. Unlike our VI implementation, here explicit weights $w \sim \mathcal{N}(\mu, \mathrm{softplus}(\rho))$ are sampled during the forward pass. The objective generalizes the ELBO using α-Rényi divergence:

$$\mathcal{L}_{\mathrm{VR}}(\theta, \alpha) = -\frac{1}{1-\alpha} \log \frac{1}{K} \sum_{k=1}^{K} \left(\frac{p(\mathcal{D}, w_k)}{q_\theta(w_k)} \right)^{1-\alpha}. \tag{3}$$

The parameter α (default 1.0) controls the bias-variance trade-off.

3.3 Probabilistic Backpropagation (PBP)

Our implementation of PBP [4] uses Assumed Density Filtering (ADF) in an online fashion. 1. Moment Propagation: Means and variances are propagated analytically. For ReLU activations, we use exact moment matching functions relying on the PDF/CDF of the standard normal. 2. Update Rule: Weights are updated by matching the moments of the tilted distribution $q(w)p(y|x,w)$. We compute gradients of the log-partition function $\log Z$ to update μ and Σ directly, bypassing standard SGD.

3.4 Laplace Approximation (LA)

We implement a scalable Laplace approximation using Kronecker-Factored Approximate Curvature (K-FAC) [3]. 1. MAP Training: A standard network is trained to find the mode w_{MAP}. 2. Curvature Estimation: We capture covariance of activations (A_{l-1}) and pre-activation gradients (G_l). 3. Sampling: The posterior for layer l is approximated as a Matrix Normal distribution $\mathcal{MN}(W_l, \Sigma_l)$. Samples are generated efficiently using eigen-decomposition of Kronecker factors:

$$W_{\text{sample}} = W_{\text{MAP}} + L_V Z L_U^T, \tag{4}$$

where L_V, L_U are Cholesky factors of the inverse regularized covariances.

4 Experiments and Evaluation

4.1 Setup and Hyperparameters

We evaluated the library on the Boston Housing dataset.

- **Architecture:** MLP with sizes 13-50-50-1. ReLU activations.
- **PBP Settings:** Noise shape parameters $\alpha = 6.0, \beta = 6.0$ (Inverse Gamma prior).
- **Laplace Settings:** MAP training with $lr = 10^{-3}$, K-FAC estimated on full training set.
- **VI/VR Settings:** Priors $\sigma = 1.0$, initialized $\rho = -2.0$ (approx $\sigma = 0.12$).

4.2 Results

Table 1 shows that while deterministic models are accurate (low RMSE), they are poorly calibrated (high NLL). PBP and VI achieve the best balance of accuracy and calibration.

Table 1: Performance on Boston Housing (Test Set). Lower is better.

Method	**RMSE**	**NLL**	**ECE**	**Brier**
Deterministic MLP	4.21	47.99	0.033	0.090
MC Dropout	**3.84**	3.68	0.019	**0.055**
Variational Inference (VI)	4.63	**2.88**	0.008	0.099
Probabilistic Backprop (PBP)	5.42	3.17	0.008	0.091
Variational Rényi (VR)	5.36	3.16	**0.006**	0.127
Laplace Approx (LA)	19.38	4.55	0.011	0.241

4.3 Adversarial Robustness

We subjected models to FGSM attacks (Table 2).

Table 2: RMSE under FGSM Adversarial Attacks. **Bold** indicates best performance.

ϵ	Det. MLP	MC Dropout	**VI**	PBP	VR	Laplace
0.00	4.21	**3.80**	4.58	5.42	5.35	19.22
0.10	7.41	**5.16**	**5.16**	5.73	5.76	24.04
0.30	13.13	8.82	**6.40**	6.87	6.64	22.92
0.50	17.70	12.67	**7.71**	8.28	8.74	26.40
1.00	27.74	23.05	**12.22**	12.34	13.50	40.46

Analysis: The deterministic model fails catastrophically at $\epsilon = 0.1$. VI and PBP demonstrate superior robustness. Specifically, PBP's moment matching effectively filters high-frequency adversarial noise, maintaining low error rates even at high perturbation levels.

5 Conclusion

We presented `bensemble`, a library that democratizes access to stochastic weight modeling. By validating implementations of VI (Graves+LRT), VR (Weight Perturbation), PBP (ADF), and LA (K-FAC), we provide a robust toolkit for uncertainty estimation. Our experiments confirm that Bayesian ensembling significantly improves calibration and adversarial robustness compared to standard point estimates.

References

[1] Graves, A. (2011). Practical variational inference for neural networks. *NeurIPS*.

[2] Kingma, D. P., Salimans, T., & Welling, M. (2015). Variational dropout and the local reparameterization trick. *NeurIPS*.

[3] Ritter, H., Botev, A., & Barber, D. (2018). A scalable Laplace approximation for neural networks. *ICLR*.

[4] Hernández-Lobato, J. M., & Adams, R. (2015). Probabilistic backpropagation for scalable learning of Bayesian neural networks. *ICML*.

[5] Li, Y., & Turner, R. E. (2016). Rényi divergence variational inference. *NeurIPS*.