Bayesian Neural Network Ensembling for Uncertainty Quantification

Gabriel Arteaga
Department of Information technology
Uppsala University
Uppsala, Sweden
gabrielyanci.arteaga.6822@student.uu.se

Alexander Sabelström
Department of Information technology
Uppsala University
Stockholm, Sweden
alexander.sabelstrom.1040@student.uu.se

I. Introduction

In the rapidly evolving field of machine learning, uncertainty quantification is a vital tool for understanding how certain a model is in its predictions. It explores the likelihood of outcomes in scenarios with unknown properties, arising from the complexity of the real-world or the data-generating process. This uncertainty manifests as two sub-categories: epistemic and aleatoric.

Epistemic uncertainty, sometimes referred to as model uncertainty, arises from the lack of knowledge or understanding about the underlying model structure and parameters. Aleatoric uncertainty, sometimes referred to as data uncertainty, stems from the inherent variability in the data itself. While epistemic uncertainty diminishes as more data is observed, aleatoric uncertainty remains irreducible in its nature [1].

Effectively addressing both epistemic and aleatoric uncertainties is crucial for enhancing the reliability and interpretability of machine learning models, especially in complex and dynamic real-world scenarios. Uncertainty quantification has been applied in several recent real-world applications, particularly in fields where decisions are not allowed to be made from uncertain outcomes, such as in the field of medicine [2].

To comprehend the inherent uncertainty in our model, we propose a novel approach using an ensemble of Neural Networks (NNs). This method incorporates a regularization strategy which facilitates approximate Bayesian inference, with scalability enhancements made from prior work [3]. A notable feature is its ability to differentiate between epistemic and aleatoric uncertainties.

Another challenge in machine learning is the increasing computational complexity associated with larger datasets. In our empirical study, we will explore Gaussian Processes (GPs), a well-known method for uncertainty quantification. A major challenge of applying GPs in real-world applications is their computational complexity, and several methods have been proposed to resolve this issue, such as inducing points [4]. The study investigates the difference in training time and predictive performance with and without inducing points.

The report commences with a section on related work, introducing the current state of the art and highlighting the relevance of our work. This section also delves into uncertainty quantification, introducing key concepts such as Neural Network Ensemble and Gaussian Processes. The Methodology section offers a theoretical overview of the models employed, accompanied by a motivation for our model choices. The report concludes with sections on results and discussion, presenting our findings and exploring their implications.

II. Related Work

A. Neural Network Ensemble

Ensemble methods, a well-established topic in machine learning, involve constructing a model by combining multiple base models. Techniques like boosting, bagging, and random forests represent notable instances of previous ensemble methods. The underlying principle is to train base models with slight variations, contributing to one ensemble model. This ensemble model utilizes the average of predictions from the base models, aiming to enhance overall performance [6].

Lakshminarayanan et al. [7] was the first to introduce Deep NNs as ensemble members into one ensemble model. Beyond enhancing predictive performance, this ensemble approach provided a method to generate reliable uncertainty estimates. This was achieved by interpreting the variance of each ensemble member's prediction as a measure of uncertainty. While the results were promising, certain limitations were identified. One drawback was the deviation from the Bayesian framework in quantifying uncertainty. Additionally, the computational aspect posed challenges, exhibiting limited scalability as the ensemble's cost increased linearly with the number of ensemble members.

Pearce et al. [8] addressed the first limitation by proposing a remedy, a Bayesian approximation. This involved incorporating noise through randomized maximum a posteriori sampling into the NNs' loss function, an approach which they coined as *anchored* regularization. This approach enabled the attainment of an approximation to the true posterior.

A remedy for the scalability limitation was proposed by Wen et al. [3], they proposed a memory efficient way to add ensemble members to an ensemble which addressed the issues associated with scalability. They coined this ensemble method as *BatchEnsemble*.

B. Gaussian Processes

A GP is a collection of random variables, any finite number of which have consistent joint Gaussian distributions [9]. It is a fundamental class of stochastic processes [10] that are widely used in Bayesian optimization [11]. GPs are fully specified by a mean and covariance function [11], where the covariance function is referred to as the kernel. GPs can be used for regression, classification and is a well established tool for uncertainty quantification.

GPs growing computational complexity arises from the need to handle large covariance matrices whose dimensions depend on the number of data points. Using inducing points can effectively reduce the size of the covariance matrix, thereby enabling faster computation [11].

Moreover, it is established that in the limit of infinite network width, a single-layer fully-connected NN with an independent and identically distributed prior over its parameters is equivalent to a GP [12]. Lee et al. [12] have derived an extension of this notion, covering deep NNs. Their work enables an exact Bayesian method for obtaining NN predictions and uncertainty estimates.

III. Methodology

This section presents a theoretical overview of the models used as well as a motivation for our chosen methods.

A. Neural Network Ensemble

We opted to construct a NN ensemble that combines the Bayesian approximation facilitated by anchored regularization [8] with the scalability offered by BatchEnsemble [3]. Additionally, we chose to employ the Gaussian Negative Log-Likelihood Loss (GNLLL), defined as:

$$GNLLL = \frac{1}{2}\left(ln\left[max(\hat{\boldsymbol{\sigma}}^2, \epsilon)\right] + \frac{[\hat{\boldsymbol{y}} - \boldsymbol{y}]^2}{max(\hat{\boldsymbol{\sigma}}^2, \epsilon)}\right) \quad (1)$$

Here, $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{\sigma}}^2$ represent a NN's predicted mean and variance, respectively. The target is denoted as $\boldsymbol{y}$ and the constant ϵ is used to clamp the variance for stability purposes [13], [14]. This approach enables our ensemble members to directly learn the mean and variance of the data.

We then defined the loss function for each ensemble member j as follows:

$$Loss_j = \frac{1}{N} GNLLL_j + \frac{1}{N}||\boldsymbol{\tau}^{1/2} \cdot (\boldsymbol{\theta}_j - \boldsymbol{\theta}_{anc,j})||_2^2 \quad (2)$$

Here, $GNLLL_j$ represents the loss computed using the GNLLL (1) for each ensemble member, $\boldsymbol{\tau}$ denotes the data noise inherent in the data as a hyperparameter, $\boldsymbol{\theta}_j$ corresponds to the weight parameters of each ensemble member. The anchored regularization term $\boldsymbol{\theta}_{anc,j}$ is drawn from a Gaussian distribution which approximates the true posterior distribution:

$$\boldsymbol{\theta}_{anc,j} \sim \mathcal{N}\left(\boldsymbol{\mu}_{prior}, \boldsymbol{\Sigma}_{prior}\right) \quad (3)$$

Finding the optimal prior can be a challenging task. To simplify the search space, we imposed constraints by setting $\mu_{prior} = 0$ and $\sigma_{W_1}^2 = \frac{\sigma_{b_1}^2}{\text{fan\_in}}$, where $\sigma_{W_1}^2$ denotes the weight initialization variance for the first layer. This, in turn, corresponds to the prior variance chosen for parameters in the first weight layer. The term $\sigma_{b_1}^2$ represents the variance for the bias in the first layer, and fan_in corresponds to the number of inputs to a layer.

This approach drew inspiration from Pearce et al.'s [8] prior variance search strategy. While Pearce et al. [8] focused on NNs with a maximum of 2 hidden layers and set $\sigma_{W_2}^2 = \frac{1}{\text{fan\_in}}$, we extended their approach to accommodate deeper networks. Given our use of ReLU activation functions, we adopted the Kaiming normal weight initialization [15]. Consequently, our weight initialization followed the pattern $\sigma_{W_\ell}^2 = \frac{2}{\text{fan\_in}}$ for the deeper layers, where $\ell \in \{2, 3, \ldots, L-1, L\}$. These considerations guided our selection of priors. Specifically, we applied the same prior variance strategy as Pearce et al. [8] for the first layer parameters but made a slight modification to the remaining layers due to our utilization of Kaiming normal weight initialization.

To leverage both the memory efficiency proposed by Wen et al. [3] and the improved training speed associated with a batched approach for training NNs, we adopted the following strategy:

$$\bar{\boldsymbol{W}}_j = \boldsymbol{W} \odot \boldsymbol{F}_j, \text{where } \boldsymbol{F}_j = \boldsymbol{r}_j \boldsymbol{s}_j^T \quad (4)$$

Here, $\boldsymbol{W}$ serves as a shared weight matrix among the ensemble members, and $\boldsymbol{F}_j$ is a rank-one matrix per ensemble member j, resulting from the outer product of trainable vectors $\boldsymbol{r}_j$ and $\boldsymbol{s}_j$. Consequently, $\bar{\boldsymbol{W}}_j$ represents the weight matrix of ensemble member j, resulting from the Hadamard product of $\boldsymbol{W}$ and $\boldsymbol{F}_j$.

To capture both epistemic (model) uncertainty and aleatoric (irreducible) uncertainty, we employed the total variance formulation as generalized by Valdenegro-Toro & Mori [16]:

$$\boldsymbol{\sigma}_{total}^2 = \boldsymbol{\sigma}_{epistemic}^2 + \boldsymbol{\sigma}_{aleatoric}^2 \quad (5)$$

Here, the epistemic uncertainty is defined as:

$$\boldsymbol{\sigma}_{epistemic}^2 = \left(\frac{1}{J}\sum_{j=1}^{J} \hat{\boldsymbol{y}}_j^2\right) - \left(\frac{1}{J}\sum_{j=1}^{J} \hat{\boldsymbol{y}}_j\right)^2 \quad (6)$$

And the aleatoric uncertainty is expressed as:

$$\boldsymbol{\sigma}_{aleatoric}^2 = \frac{1}{J}\sum_{j=1}^{J} \hat{\boldsymbol{\sigma}}_j^2 \quad (7)$$

This formulation allows us to quantify the total uncertainty by combining both sources.

The implementation was conducted using Python as the programming language, and PyTorch [17] served as our Deep Learning framework.

B. Gaussian Processes

In the initial stages of this study, numerous kernels for the GPs were tuned and tested. The potential impact on performance for each kernel was to be explored. Identifying the optimal GP for each dataset was not a primary goal for this study however, but rather to compare them with and without

inducing points. Thereby, we decided to standardize the use of the Radial Basis function (RBF) kernel [18] across all GPs, defined as:

$$k_{RBF}(x_1, x_2) = exp(-\frac{1}{2}(x_1 - x_2)^T \Theta^{-2}(x_1 - x_2)) \quad (8)$$

where Θ is the lengthscale hyperparameter. Constant mean as a prior across all GPs was also used. The RBF kernel is flexible and generalizes well, making it suitable for various datasets of different dimensions and size.

Given a GP $f \sim GP(\mu, K)$, and data X, y, we chose to use the marginal log likelihood as a loss, defined as [19]:

$$Loss = p_f(y|X) = \int p(y|f(X))p(f(X)|X)\ df \quad (9)$$

which will be negated in the optimization step.

Several approaches to find the inducing points have been proposed. We decided to use the K-means [20] method. It works by initially applying K-means on a given dataset, partitioning it into K clusters. The centroids of each cluster is then used as an inducing point. This makes sure that the points are spread across the data space, capturing the general structure of the dataset. We furthermore decided to run the K-means algorithm one time, thereby using only one centroid seed. Increasing the amount of different centroid seeds tested will drastically increase training time for larger datasets, with negligible performance difference.

The GPs were implemented using Python and GPytorch [21], a GP library implemented in PyTorch.

IV. Results

A. Datasets and Metrics

For testing our models and visualizing how different models account for uncertainty, we employed two distinct *toy datasets*. The first dataset, crafted by Izmailov et al. [22], comprises three disjoint clusters of data. The second dataset, named the *"wiggle dataset"* and constructed by Antorán et al. [23], consists of 300 samples generated from $y = \sin(\pi x) + 0.2\cos(4\pi x) - 0.3x + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.025)$ and $x \sim \mathcal{N}(5, 2.5)$. For the sake of reproducibility, we set the seed for drawing the samples to 25.

In addition to the toy datasets, we conducted benchmarking on our models using UCI Regression datasets, specifically the Concrete [24] and Power [25] datasets. To measure predictive performance, we utilized the Root Mean Square Error (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \quad (10)$$

Furthermore, to evaluate the uncertainty in our models we utilized two commonly used metrics in uncertainty quantification, the Mean Prediction Interval Width (MPIW) [5]:

$$MPIW = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n^{high} - \hat{y}_n^{low}) \quad (11)$$

where N is the number of data points, and $\hat{y}_n^{high}$ and $\hat{y}_n^{low}$ represent the upper and lower bounds, respectively. Our second uncertainty metric is the Prediction Interval Coverage Probability (PICP):

$$PICP = \frac{1}{N}\sum_{n=1}^{N}\mathbb{1}_{y_n \leq \hat{y}_n^{high}} \cdot \mathbb{1}_{y_n \geq \hat{y}_n^{low}}. \quad (12)$$

MPIW measures the mean width of all prediction intervals to evaluate the sharpness of the uncertainty estimates, while PICP measures the probability, that the ground truth, y, falls within a specified prediction interval [5]. Following Pearce et al.'s introduction of the High-Quality Prediction Interval (HQ principle) [26], our objective was to create prediction intervals that were as narrow as possible while capturing a specified proportion of data points, set at 95% in our case.

Additionally, we measured training as well as inference time.

B. Setup

The chosen models comprise three distinct types of NNs and two GP models. One GP model utilizes all datapoints, while the other employs inducing points. The first NN model is a *Naive* ensemble, where all ensemble members are trained sequentially. The second model, denoted as the *BatchEnsemble* model, leverages efficiency improvements proposed by Wen et al. [3]. Both the Naive and BatchEnsemble models utilize the GNLLL (1) and the total variance (5) to ensure comparable results.

Additionally, we introduce a novel model, named *"AnchoredBatchEnsemble,"* which combines anchored regularization for approximating Bayesian inference, as proposed by Pearce et al. [8], with the efficiency improvements from BatchEnsemble. This integrated model represents a fusion of Bayesian approximation and efficiency enhancements.

The hyperparameters selected for the datasets are detailed in Table I. Both Naive and BatchEnsemble architectures employed the *Weight Decay* hyperparameter, thus these models were fine-tuned with weight decay regularization. In contrast, AnchoredBatchEnsemble, utilizing Anchored Regularization instead of L2 regularization, incorporated *Data Noise* as its hyperparameter. Additionally, we tuned our models with Dropout regularization [27], but its inclusion consistently resulted in decreased performance across all models and datasets. As the optimal dropout probability was found to be 0, we excluded it from the table.

TABLE I: NN Ensemble Hyperparameters for Different Datasets

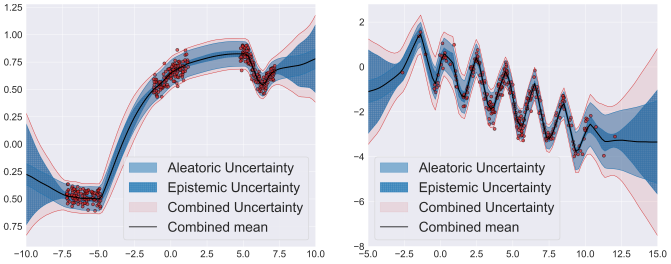
Dataset	Hidden Layers	Hidden Units	Ensemble Size	Data Noise	Weight Decay
Power	3	256	8	1e-4	1e-3
Concrete	6	128	16	1e-4	1e-6

The experiments were performed on an *NVIDIA GeForce RTX 2070 SUPER*. For comparison purposes, all models underwent 1000 training epochs across both datasets.

C. Toy Datasets

The results for the Izmailov dataset [22] and the Wiggle dataset [23] are depicted in Figures 1 and 2. A 2σ-region was utilized, corresponding to a 95% prediction interval. This signifies that our models express 95% confidence that the next observed value will fall within the shaded region.

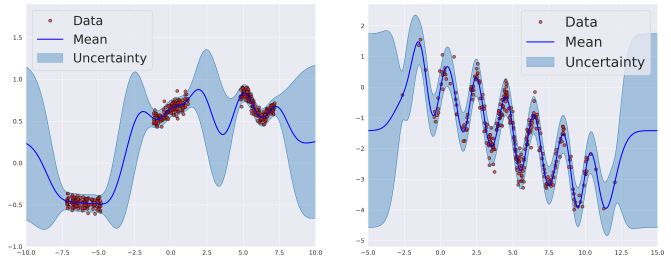
Figure 1 showcases the results obtained from the AnchoredBatchEnsemble, were we utilized an architecture containing four hidden layers. The model manages to disentangle the uncertainty into two categories: epistemic and aleatoric. The dark blue region represents the epistemic uncertainty, which decreases in regions with more data and increases with less data. The aleatoric uncertainty manages to account for the inherent noise in the data.



(a) Izmailov dataset (b) Wiggle dataset

Fig. 1: AnchoredBatchEnsemble

Similarly, Figure 2 illustrates the results generated by the GP. The GP adeptly captures the majority of data points within its uncertainty region.



(a) Izmailov dataset (b) Wiggle Dataset

Fig. 2: Gaussian Processes with RBF Kernel

Both AnchoredBatchEnsemble and GP miss a few points, which aligns with expectations for a 95% prediction interval.

Pearce et al. [8] previously conducted a comprehensive comparison between their anchored regularized method and a GP mimicking a single-layer NN. Their findings suggested that their approximation tends to underestimate variance. To thoroughly evaluate the uncertainty estimates of our proposed method and compare them with those of a GP, it would have been necessary to employ a GP mimicking a deep NN with ReLU non-linearities, as proposed by Lee et al. [12].

D. UCI Datasets

To compare the performance of the different models, we benchmarked their performance. The experiments were conducted using a 90/10 train/test split. Additionally, the NN models utilized a second split to create a validation set, employing an 80/20 split. In contrast, GPs were trained on the entire training dataset.

In Table II you can find the benchmarking results for the Power dataset. The AnchoredBatch approach outperforms the other models both in terms of predictive performance and, according to the HQ principle, in terms of uncertainty quantification. Furthermore, AnchoredBatch outperforms the GP models in terms of inference time.

TABLE II: Results for the Power Dataset

Metric	Anchored Batch	Batch	Naive	GP	GP Induce
RMSE	4.24	4.27	4.86	5.11	5.80
PICP	0.96	0.99	1.00	0.76	0.69
MPIW	17.05	22.09	48.77	12.20	11.91
Train Time (s)	456.06	333.64	1755.97	87.55	52.92
Inference Time (s)	0.02	0.02	0.04	0.31	0.10

In Table III, we present the benchmarking results for the Concrete dataset. Surprisingly, the Naive model outperforms other models in both predictive performance and uncertainty. One plausible explanation for this remarkable performance could be attributed to the set of initial weights, which warrants further investigation. However, the Naive model exhibits a significant drawback compared to other models, with its speed severely diminishing. Looking beyond the Naive model, we observe that the AnchoredBatch model outperforms the remaining models in terms of uncertainty, following the HQ principle. On the other hand, the Batch model performs better in predictive performance and inference time.

TABLE III: Results for the Concrete Dataset

Metric	Anchored Batch	Batch	Naive	GP	GP Induce
RMSE	6.98	6.85	5.59	7.45	7.92
PICP	0.92	0.91	0.98	0.73	0.60
MPIW	24.26	23.39	26.40	15.17	14.90
Train Time (s)	75.59	50.99	496.42	24.49	5.01
Inference Time (s)	0.003	0.002	0.147	0.008	0.004

For both datasets, the GP models perform significantly worse than the other models, as observed through RMSE and PICP. This finding is surprising, considering that GP models are typically expected to excel in accounting for uncertainty. Generally, GP models are favored for their ability to model complex uncertainties in data.

A possible explanation for this unexpected result could be attributed to the choice of kernel. The kernel plays a crucial role in shaping the GP model, and its selection can greatly influence its performance. Further research is needed to investigate how the choice of kernel interacts with the characteristics of the datasets, potentially shedding light on the observed underperformance of GP models. The GPs demonstrates superior efficiency in terms of training time. Moreover, the use of inducing points results in a slight worse predictive performance but with significant reduction in training time.

AnchoredBatch exhibits promising results in both predictive performance and its ability to account for uncertainty. While there is a slight increase in training time due to extra calculations required for anchored regularization, this is negligible when compared with the Naive model. Furthermore, once trained, AnchoredBatch manages to achieve fast predictions.

E. Training time

To assess the efficiency performance gained from utilizing BatchEnsemble, we conducted tests on the training time for different ensemble sizes, specifically powers of 2 up to 2^5, comparing both the Naive model and the proposed AnchoredBatch version. The results, illustrated in Figure 3, reveal insightful patterns.

Initially, the Naive model demonstrates faster training times than the proposed AnchoredBatch version when utilizing one or two ensemble members. This observation aligns with expectations, considering the inherent overhead cost associated with anchored regularization. However, this cost becomes negligible as we increase the ensemble size. Notably, the Naive model exhibits linear growth in training time with an increasing number of ensemble members, while the proposed AnchoredBatch model's training time stagnates in comparison.

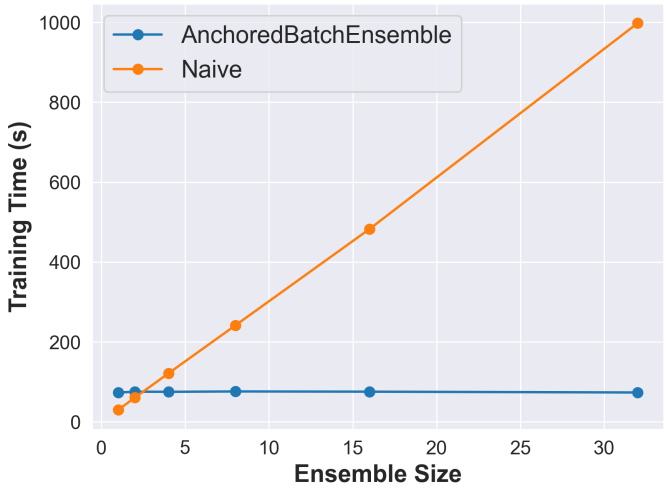


Fig. 3: Training time for different ensemble sizes for the AnchoredBatchEnsemble model and the Naive model

V. Discussion and Conclussion

In this study, we successfully managed to visualize the epistemic and aleatoric uncertainty for our proposed method on the toy datasets. This serves as a critical tool for understanding and interpret the model's confidence in its predictions. Furthermore, it achieved a high prediction coverage given the specified objective.

For the Power dataset, AnchoredBatch demonstrated notable superiority, excelling in predictive performance, uncertainty quantification, and inference time. This highlights its efficiency and effectiveness in handling diverse data characteristics. However, the model's performance on the Concrete dataset showed slight degradation, lacking the consistent superiority observed in the Power dataset.

Regarding training time, our proposed method demonstrated superior performance compared to the Naive model when utilizing an ensemble size larger than two. Furthermore, we anticipate that as the dataset size increases, the overhead from our regularization strategy diminishes, allowing our proposed method to outperform the Naive model even with an ensemble size of two.

Surprisingly, our GP models performed worse in accounting for uncertainty than expected. A thorough investigation into finding an appropriate kernel for the datasets is left for future experiments. Additionally, a qualitative comparative study between our proposed method and a GP would benefit from utilizing a GP mimicking a deep NN with ReLU nonlinearities, a consideration for future work.

Resource limitations prevented extensive testing of models. Given more time or resources, evaluating performance across several benchmarking iterations would provide valuable insights.

References

[1] Hüllermeier, E. and Waegeman, W., 2021. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. Machine Learning, 110, pp.457-506.

[2] Alizadehsani, R., Roshanzamir, M., Hussain, S., Khosravi, A., Koohestani, A., Zangooei, M. H., ... & Acharya, U. R., 2021. Handling of uncertainty in medical data using machine learning and probability theory techniques: A review of 30 years (1991–2020). Annals of Operations Research, pp.1-42.

[3] Wen, Y., Tran, D. and Ba, J., 2020. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. arXiv preprint arXiv:2002.06715.

[4] Garnett, R., 2023. Bayesian optimization. Cambridge University Press.

[5] Yao, J., et al., 2019. Quality of uncertainty quantification for Bayesian neural network inference. arXiv preprint arXiv:1906.09686.

[6] Lindholm, A., Wahlström, N., Lindsten, F. and Schön, T.B., 2022. Machine learning: a first course for engineers and scientists. Cambridge University Press.

[7] Lakshminarayanan, B., Pritzel, A. and Blundell, C., 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. Advances in neural information processing systems, 30.

[8] Pearce, T., Leibfried, F. and Brintrup, A., 2020, June. Uncertainty in neural networks: Approximately bayesian ensembling. In International conference on artificial intelligence and statistics (pp. 234-244). PMLR.

[9] Quinonero-Candela, J. and Rasmussen, C.E., 2005. A unifying view of sparse approximate Gaussian process regression. The Journal of Machine Learning Research, 6, pp.1939-1959.

[10] Lall, S. (2011). *Gaussian stochastic processes*. Stanford: Stanford University.

[11] Garnett, R. (2023). *Bayesian optimization*. Cambridge: Cambridge University Press.

[12] Lee, J., Bahri, Y., Novak, R., Schoenholz, S.S., Pennington, J. and Sohl-Dickstein, J., 2017. Deep neural networks as gaussian processes. arXiv preprint arXiv:1711.00165.

[13] PyTorch, "torch.nn.GaussianNLLLoss," PyTorch Documentation, [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.GaussianNLLLoss.html. [Accessed: 03-Jan-2024].

[14] Nix, D.A. and Weigend, A.S., 1994, June. Estimating the mean and variance of the target probability distribution. In Proceedings of 1994 ieee international conference on neural networks (ICNN'94) (Vol. 1, pp. 55-60). IEEE.

[15] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

[16] Valdenegro-Toro, M. and Mori, D.S., 2022, June. A deeper look into aleatoric and epistemic uncertainty disentanglement. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (pp. 1508-1516). IEEE.

[17] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. and Desmaison, A., 2019. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32.

[18] , title = Kernels — GPyTorch 1.5.1 documentation, howpublished = https://docs.gpytorch.ai/en/stable/kernels.html, note = Accessed: 2024-01-05

[19] GPyTorch (2020) gpytorch.mlls — GPyTorch 0.1.dev97+gf73fa7d documentation. Available at: 1 (Accessed: 15 November 2021).

[20] Scikit-learn developers. K-Means clustering — scikit-learn 1.3.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, 2021. Accessed: 2024-01-05.

[21] Gardner, Jacob, et al. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. In: Advances in Neural Information Processing Systems 31, 2018

[22] Izmailov, P., Maddox, W.J., Kirichenko, P., Garipov, T., Vetrov, D. and Wilson, A.G., 2020, August. Subspace inference for Bayesian deep learning. In Uncertainty in Artificial Intelligence (pp. 1169-1179). PMLR.

[23] Antorán, J., Allingham, J. and Hernández-Lobato, J.M., 2020. Depth uncertainty in neural networks. Advances in neural information processing systems, 33, pp.10620-10634.

[24] Yeh, I.C., 2007. Concrete compressive strength data set. UCI Machine Learning Repository.

[25] Tfekci, P. and Kaya, H., 2014. Combined Cycle Power Plant. UCI Machine Learning Repository.

[26] Pearce, T., Brintrup, A., Zaki, M. and Neely, A., 2018, July. High-quality prediction intervals for deep learning: A distribution-free, ensembled approach. In International conference on machine learning (pp. 4075-4084). PMLR.

[27] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), pp.1929-1958.