Appendix for Paper: Learning to Estimate Point-Prediction Uncertainty and Correct Output in Neural Networks

Xin Qiu
Cognizant
qiuxin.nju@gmail.com

Elliot Meyerson
Cognizant
elliot.meyerson@cognizant.com

Risto Miikkulainen
Cognizant
The University of Texas at Austin
risto@cognizant.com

Abstract

This Appendix supplements the paper: Learning to Estimate Point-Prediction Uncertainty and Correct Output in Neural Networks. Section S1 reviews related work. Section S2 introduces background details for the methods (i.e., neural networks, Gaussian processes, and stochastic variational Gaussian processes) that are used to develop the approach, RIO, in the paper. Section S3 provides theoretical analysis of RIO. Section S4 provides additional details on the experimental setups used in the paper, along with additional details on results.

S1 Related Work

There has been significant interest in combining NNs with probabilistic Bayesian models. An early approach was Bayesian Neural Networks, in which a prior distribution is defined on the weights and biases of a NN, and a posterior distribution is then inferred from the training data [17, 18]. Traditional variational inference techniques have been applied to the learning procedure of Bayesian NN, but with limited success [1, 9, 12]. By using a more advanced variational inference method, new approximations for Bayesian NNs were achieved that provided similar performance as dropout NNs [2]. However, the main drawbacks of Bayesian NNs remain: prohibitive computational cost and difficult implementation procedure compared to standard NNs.

Alternatives to Bayesian NNs have been developed recently. One such approach introduces a training pipeline that incorporates ensembles of NNs and adversarial training [14]. Another approach, NNGP, considers a theoretical connection between NNs and GP to develop a model approximating the Bayesian inference process of wide deep neural networks [16]. Deep kernel learning (DKL) combines NNs with GP by using a deep NN embedding as the input to the GP kernel [26] . Conditional Neural Processes (CNPs) combine the benefits of NNs and GP, by defining conditional distributions over functions given data, and parameterizing this dependence with a NN [7]. Neural Processes (NPs) generalize deterministic CNPs by incorporating a latent variable, strengthening the connection to approximate Bayesian and latent variable approaches [8]. Attentive Neural Processes (ANPs) further extends NPs by incorporating attention to overcome underfitting issues [13]. The above models all require significant modifications to the original NN model and training pipeline. Compared to standard NNs, they are also less computationally efficient and more difficult for practitioners to implement. In the approach that shares the most motivation with RIO, Monte Carlo dropout was

3rd Workshop on Meta-Learning at NeurIPS 2019, Vancouver, Canada

used to estimate the predictive uncertainty of dropout NNs [6]. However, this method is restricted to dropout NNs, and also requires modifications to the NN inference process.

S2 Background

This section reviews notation for Neural Networks, Gaussian Processes, and its more efficient approximation, Stochastic Variational Gaussian Processes. The RIO method, introduced in Section 2 of the main paper, uses Gaussian Processes to estimate the uncertainty in neural network predictions and reduces their point-prediction errors.

S2.1 Neural Networks

Neural Networks (NNs) learn a nonlinear transformation from input to output space based on a number of training examples. Let $\mathcal{D} \subseteq \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}$ denote the training dataset with size n, and $\mathcal{X} = \{\mathbf{x}_i : (\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}, \mathbf{x}_i = [x_i^1, x_i^2, \ldots, x_i^{d_{\text{in}}}] \mid i = 1, 2, \ldots, n\}$ and $\mathcal{Y} = \{\mathbf{y}_i : (\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}, \mathbf{y}_i = [y_i^1, y_i^2, \ldots, y_i^{d_{\text{out}}}] \mid i = 1, 2, \ldots, n\}$ denote the inputs and outputs (i.e., targets). A fully-connected feed-forward neural network with L hidden layers of width N_l (for layer $l = 1, 2, \ldots, L$) performs the following computations: Let z_l^j denote the output value of jth node in lth hidden layer given input $\mathbf{x}_i$, then $z_l^j = \phi(\sum_{k=1}^{d_{\text{in}}} w_l^{j,k} x_i^k + b_l^j)$, for $l = 1$ and $z_l^j = \phi(\sum_{k=1}^{N_{l-1}} w_l^{j,k} z_{l-1}^k + b_l^j)$, for $l = 2, \ldots, L$, where $w_l^{j,k}$ denotes the weight on the connection from kth node in previous layer to jth node in lth hidden layer, b_l^j denotes the bias of jth node in lth hidden layer, and ϕ is a nonlinear activation function. The output value of jth node in output layer is then given by $\hat{y}_i^j = \sum_{k=1}^{N_L} w_{\text{out}}^{j,k} z_L^k + b_{\text{out}}^j$, where $w_{\text{out}}^{j,k}$ denotes the weight on the connection from kth node in last hidden layer to jth node in output layer, and b_{out}^j denotes the bias of jth node in output layer.

A gradient-based optimizer is usually used to learn the weights and bias given a pre-defined loss function, e.g., a squared loss function $\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$. For a standard NN, the learned parameters are fixed, so the NN output $\hat{\mathbf{y}}_i$ is also a fixed point. For a Bayesian NN, a distribution of the parameters is learned, so the NN output is a distribution of $\hat{\mathbf{y}}_i$. However, a pre-trained standard NN needs to be augmented, e.g., with a Gaussian Process, to achieve the same result.

S2.2 Gaussian Processes

A Gaussian Process (GP) is a collection of random variables, such that any finite collection of these variables follows a joint multivariate Gaussian distribution [22]. Given a training dataset $\mathcal{X} = \{\mathbf{x}_i \mid i = 1, 2, \ldots, n\}$ and $\mathcal{Y} = \{y_i = f(\mathbf{x}_i) + \epsilon \mid i = 1, 2, \ldots, n\}$, where ϵ denotes additive independent identically distributed Gaussian noise, the first step for GP is to fit itself to these training data assuming $\mathcal{Y} \sim \mathcal{N}(0, \mathbf{K}(\mathcal{X}, \mathcal{X}) + \sigma_n^2 \mathbf{I})$, where $\mathcal{N}$ denotes a multivariate Gaussian distribution with mean 0 and covariance matrix $\mathbf{K}(\mathcal{X}, \mathcal{X}) + \sigma_n^2 \mathbf{I}$. $\mathbf{K}(\mathcal{X}, \mathcal{X})$ denotes the kernel-based covariance matrix at all pairs of training points with each entry $k_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$, and σ_n^2 denotes the noise variance of observations. One commonly used kernel is the radial basis function (RBF) kernel, which is defined as $k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp(-\frac{1}{2l_f^2} \|\mathbf{x}_i - \mathbf{x}_j\|^2)$. The signal variance σ_f^2, length scale l_f and noise variance σ_n^2 are trainable hyperparameters. The hyperparameters of the covariance function are optimized during the learning process to maximize the log marginal likelihood $\log p(\mathcal{Y}|\mathcal{X})$.

After fitting phase, the GP is utilized to predict the distribution of label y_* given a test point $\mathbf{x}_*$. This prediction is given by $y_*|\mathcal{X}, \mathcal{Y}, \mathbf{x}_* \sim \mathcal{N}(\bar{y}_*, \text{var}(y_*))$ with $\bar{y}_* = \mathbf{k}_*^\top (\mathbf{K}(\mathcal{X}, \mathcal{X}) + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}$ and $\text{var}(y_*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (\mathbf{K}(\mathcal{X}, \mathcal{X}) + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}_*$, where $\mathbf{k}_*$ denotes the vector of kernel-based covariances (i.e., $k(\mathbf{x}_*, \mathbf{x}_i)$) between $\mathbf{x}_*$ and all the training points, and $\mathbf{y}$ denotes the vector of all training labels. Unlike with NN, the uncertainty of the prediction of a GP is therefore explicitly quantified.

S2.3 Stochastic Variational Gaussian Processes

The main limitation of the standard GP, as defined above, is that it is excessively expensive in both computational and storage cost. For a dataset with n data points, the inference of standard GP has time complexity $\mathcal{O}(n^3)$ and space complexity $\mathcal{O}(n^2)$. To circumvent this issue, sparse GP methods were

developed to approximate the original GP by introducing inducing variables [4, 20, 23, 25]. These approximation approaches lead to a computational complexity of $\mathcal{O}(nm^2)$ and space complexity of $\mathcal{O}(nm)$, where m is the number of inducing variables. Following this line of work, the Stochastic Variational Gaussian Process (SVGP) [10, 11] further improves the scalability of the approach by applying Stochastic Variational Inference (SVI) technique, as follows:

Consider the same training dataset and GP as in Section S2.2, and assume a set of inducing variables as $\mathcal{Z} = \{\mathbf{z}_i \mid i = 1, 2, \ldots, m\}$ and $\mathcal{U} = \{u_i = f(\mathbf{z}_i) + \epsilon \mid i = 1, 2, \ldots, m\}$ ($f(\cdot)$ and ϵ are unknown). SVGP learns a variational distribution $q(\mathcal{U})$ by maximizing a lower bound of $\log p(\mathcal{Y}|\mathcal{X})$, where $\log p(\mathcal{Y}|\mathcal{X}) = \log \int p(\mathcal{Y}|\mathcal{U}, \mathcal{X}) p(\mathcal{U}) \mathrm{d}\mathcal{U}$ and $p(\cdot)$ denotes the probability density under original GP. Trainable hyperparameters during the learning process include values of $\mathbf{z}_i$ and hyperparameters of the covariance function of original GP. Given a test point $\mathbf{x}_*$, the predictive distribution is then given by $p(y_*|\mathbf{x}_*) = \int p(y_*|\mathcal{U}, \mathbf{x}_*) q(\mathcal{U}) \mathrm{d}\mathcal{U}$, which still follows a Gaussian distribution. One advantage of SVGP is that minibatch training methods [15] can be applied in case of very large dataset. Suppose the minibatch size is m' and $m \ll m\prime$, then for each training step/iteration, the computational complexity is $\mathcal{O}(m'm^2)$, and the space complexity is $\mathcal{O}(m'm)$. For full details about SVGP, see [10]. Since NNs typically are based on training with relatively large datasets, SVGP makes it practical to implement uncertainty estimates on NNs.

S3 Theoretical Analysis

S3.1 Underlying Rationale of Residual Prediction

This subsection gives a theoretical justification for why residual prediction helps both error correction and uncertainty estimation of an NN. Specifically, such prediction (1) leads to output that is more accurate than that of GP alone, (2) leads to output that is more accurate than that of the NN alone, and (3) leads to uncertainty estimates that are positively correlated with variance of NN residuals.

Consider a dataset $\mathcal{D} = (\mathcal{X}, \mathcal{Y}) = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, with $\mathbf{x}_i$ drawn i.i.d. from a distribution $p(\mathbf{x})$, and

$$y_i = f(\mathbf{x}_i) + g(\mathbf{x}_i) + \epsilon_i,$$

where $f(\cdot) \sim \mathrm{GP}(0, k(\cdot, \cdot))$, $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2 > 0)$, and g is a function with mean zero and variance $\mathbb{E}_{\mathbf{x}}[g^2(\mathbf{x})] = \sigma_g^2 > 0$. Without loss of generality, $f(\cdot)$ represents the component in the target function that can be modeled by a GP, ϵ represents the observation noise, and $g(\cdot)$ represents all the remaining components. With mean squared error as loss, the optimal predictor for y is $h^*(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x})$.

Suppose that, from the perspective of GP given $\mathcal{D}$, g is indistinguishable from noise, but that there is a neural network h_{NN} that has successfully learned some of its structure. Consider the residuals $h^*(\mathbf{x}) - h_{\mathrm{NN}}(\mathbf{x}) = r(\mathbf{x}) = r_f(\mathbf{x}) + r_g(\mathbf{x})$. Here, r_f is the remaining GP component, i.e., $r_f \sim \mathrm{GP}(0, \alpha k(\cdot, \cdot))$, for some non-negative $\alpha \leq 1$. Similarly, r_g is the remainder of g indistinguishable from noise, i.e., $\sigma_g^2 - \mathbb{E}_{\mathbf{x}}[r_g^2(\mathbf{x})] = \delta > 0$. Aside from these indistinguishable functions, assume GP hyperparameters can be estimated optimally.

Let h_{GP} be the posterior mean of the GP trained directly on $\mathcal{D}$, and r_{GP} be that of a GP trained to fit the residuals, which yields the final predictor $h_{\mathrm{GP+NN}} = h_{\mathrm{NN}} + r_{\mathrm{GP}}$. Let $E^g_{\mathrm{GP}}(\mathcal{X})$, $E^g_{\mathrm{NN}}(\mathcal{X})$, and $E^g_{\mathrm{GP+NN}}(\mathcal{X})$ be the expected generalization errors of h_{GP}, h_{NN}, and $h_{\mathrm{GP+NN}}$.

First, following a standard approach [22], consider the eigenfunction expansion $k(\mathbf{x}, \mathbf{x}') = \sum_j \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{x}')$ and $\int k(\mathbf{x}, \mathbf{x}') \phi_i(\mathbf{x}) p(x) d\mathbf{x} = \lambda_i \phi_i(\mathbf{x}')$. Let Λ be the diagonal matrix of the eigenvalues λ_j, and Φ be the design matrix, i.e., $\Phi_{ji} = \phi_j(\mathbf{x}_i)$. The following series of results capture the improvement due to residual estimation (proofs in section S3.3.1).

Lemma S3.1. $E^g_{\mathrm{GP}}(\mathcal{X}) = \mathrm{tr}(\Lambda^{-1} + (\sigma_n^2 + \sigma_g^2)^{-1} \Phi\Phi^\top)^{-1} + \sigma_g^2$.

Lemma S3.2. $E^g_{\mathrm{NN}}(\mathcal{X}) = \alpha \mathbb{E}[f^2(\mathbf{x})] + \sigma_g^2 - \delta$.

Lemma S3.3. $E^g_{\mathrm{GP+NN}}(\mathcal{X}) = \mathrm{tr}[\alpha^{-1}\Lambda^{-1} + (\sigma_n^2 + \sigma_g^2 - \delta)^{-1} \Phi\Phi^\top]^{-1} + \sigma_g^2 - \delta$.

Theorem S3.4. *$E^g_{\mathrm{GP+NN}}(\mathcal{X}) < E^g_{\mathrm{GP}}(\mathcal{X}) - \delta$ and $E^g_{\mathrm{GP+NN}}(\mathcal{X}) < E^g_{\mathrm{NN}}(\mathcal{X})$.*

In particular, the improvement with respect to GP is greater than the reduction in apparent noise. Importantly, this improvement in error corresponds to a predictive variance that is closer to the optimal for this problem, so uncertainty estimates are improved as well. Experiments on real world

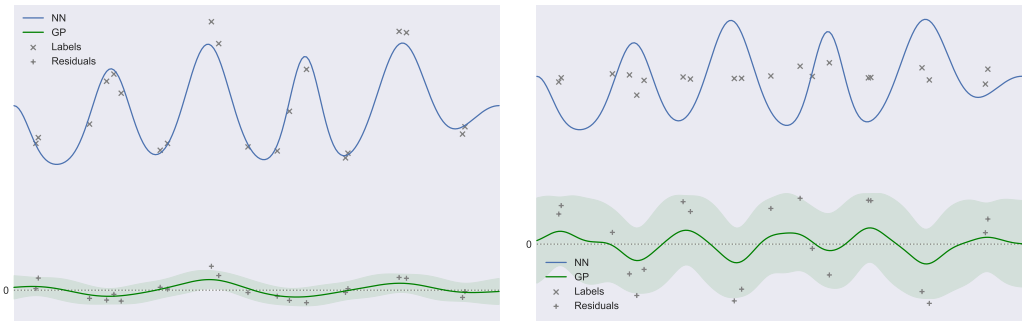

Figure S1: **Capturing uncertainty of more and less accurate NNs.** These figures illustrate the behavior of RIO in two cases: (left) The neural network has discovered true complex structure in the labels, so the residuals have low variance and are easy for the GP to fit with high confidence; (right) The ineffective neural network has introduced unnecessary complexity, so the residuals are modeled with high uncertainty. In both cases, RIO matches the intuition for how uncertain the NN really is.

data confirm that when predicting the residuals of an NN, the estimated noise level of the GP is indeed lower and correlates with the reduction in generalization error. This reduction is possible because the NN is able to extract higher-level features not available to the GP.

These results also lead to a key practical property, which is illustrated in Figure S1.

Proposition 1. *The variance of NN residuals is positively correlated with the uncertainty of* r_{GP}.

Proof. Increases in $\mathbb{E}[r_f^2(x)]$ lead to increases in α, and increases in $\mathbb{E}[r_g^2(x)]$ lead to decreases in δ. So, increases in either $\mathbb{E}[r_f^2(x)]$ or $\mathbb{E}[r_g^2(x)]$ lead to increases of $\mathrm{tr}[\alpha^{-1}\Lambda^{-1}+(\sigma_n^2+\sigma_g^2-\delta)^{-1}\Phi\Phi^\top]^{-1}$, which is the expected predictive variance of r_{GP}. □

This property matches the intuition that the GP's variance should generally be higher for bad NNs than for good NNs. Such a property is crucial to accurately measuring the confidence of NN predictions.

S3.2 Robustness of I/O Kernel

This section provides a justification for why a GP using the proposed I/O kernel is more robust than the standard GP, i.e., using the input kernel alone. The reasoning closely matches that in Section S3.1.

Consider the setup in Section S3.1, but now with $y_i = f_{\text{in}}(\mathbf{x}_i) + f_{\text{out}}(\mathbf{x}_i) + \epsilon_i$, where $f_{\text{in}} \sim GP(0, k_{\text{in}})$ and $f_{\text{out}} \sim GP(0, k_{\text{out}})$, with non-trivial RBF kernels k_{in}, k_{out}. Again, suppose that, due to its high expressivity [3], h_{NN} is indistinguishable from noise from the perspective of GP.

Let $E^g_{\mathrm{I}}(\mathcal{X})$, $E^g_{\mathrm{O}}(\mathcal{X})$, and $E^g_{\mathrm{I/O}}(\mathcal{X})$ be the expected generalization errors of the standard GP, GP with output kernel only, and GP with I/O kernel. Then, the expected result follows (proof in S3.3.2 of appendix).

Theorem S3.5. $E^g_{\mathrm{I/O}}(\mathcal{X}) < E^g_{\mathrm{I}}(\mathcal{X})$ *and* $E^g_{\mathrm{I/O}}(\mathcal{X}) < E^g_{\mathrm{O}}(\mathcal{X})$.

The optimizer associated with GP simultaneously optimizes the hyperparameters of both kernels, so the less useful kernel usually receives a smaller signal variance. As a result, the I/O kernel is resilient to failures of either kernel. In particular, the GP using I/O kernel improves performance even in the case where the problem is so complex that Euclidean distance in the input space provides no useful correlation information or when the input space contains some noisy features. Conversely, when the NN is a bad predictor, and h_{NN} is simply noise, the standard GP with input kernel alone is recovered. In other words, the I/O kernel is never worse than using the input kernel alone, and in practice it is often better.

S3.3 Proofs

S3.3.1 Proofs for Section S3.1

Theorem S3.4 follows from a series of three lemmas. First, following a standard approach [22], consider the eigenfunction expansion $k(\mathbf{x}, \mathbf{x}') = \sum_j \lambda_j \phi_j(\mathbf{x})\phi_j(\mathbf{x}')$ and $\int k(\mathbf{x}, \mathbf{x}')\phi_i(\mathbf{x})p(x)d\mathbf{x} = \lambda_i \phi_i(\mathbf{x}')$. Let Λ be the diagonal matrix of the eigenvalues λ_j, and Φ be the design matrix, i.e., $\Phi_{ji} = \phi_j(\mathbf{x}_i)$.

Lemma S3.1. $E^g_{\mathrm{GP}}(\mathcal{X}) = \mathrm{tr}(\Lambda^{-1} + (\sigma_n^2 + \sigma_g^2)^{-1}\Phi\Phi^\top)^{-1} + \sigma_g^2$.

Proof. Since, from the perspective of GP, g is indistinguishable from noise, the optimal hyperparameter setting for the GP predictor is mean zero, kernel $k(\cdot, \cdot)$ and noise variance $\sigma_n^2 + \sigma_g^2$. The expected generalization error of the GP predictor with posterior mean h_{GP} is

$$\begin{aligned}
E^g_{\mathrm{GP}}(\mathcal{X}) &= \mathbb{E}[(h^*(\mathbf{x}) - h_{\mathrm{GP}}(\mathbf{x}))^2] = \mathbb{E}[(f(\mathbf{x}) + g(\mathbf{x}) - h_{\mathrm{GP}}(\mathbf{x}))^2] \\
&= \mathbb{E}[((f(\mathbf{x}) - h_{\mathrm{GP}}(\mathbf{x})) + g(\mathbf{x}))^2] \\
&= \mathbb{E}[((f(\mathbf{x}) - h_{\mathrm{GP}}(\mathbf{x}))^2 + 2g(\mathbf{x})(f(\mathbf{x}) - h_{\mathrm{GP}}(\mathbf{x})) + g^2(\mathbf{x})] \\
&= \mathbb{E}[((f(\mathbf{x}) - h_{\mathrm{GP}}(\mathbf{x}))^2] + 2\mathbb{E}[g(\mathbf{x})(f(\mathbf{x}) - h_{\mathrm{GP}}(\mathbf{x}))] + \mathbb{E}[g^2(\mathbf{x})] \\
&= \mathbb{E}[((f(\mathbf{x}) - h_{\mathrm{GP}}(\mathbf{x}))^2] + \sigma_g^2,
\end{aligned}$$

where the last step makes use of the fact that the expectation of the product of independent zero-mean random variables is zero. Plugging in a well-known result for the generalization error of Gaussian processes [19, 22, 24] yields the intended result

$$E^g_{\mathrm{GP}}(\mathcal{X}) = \mathrm{tr}(\Lambda^{-1} + (\sigma_n^2 + \sigma_g^2)^{-1}\Phi\Phi^\top)^{-1} + \sigma_g^2.$$

□

Lemma S3.2. $E^g_{\mathrm{NN}}(\mathcal{X}) = \alpha\mathbb{E}[f^2(\mathbf{x})] + \sigma_g^2 - \delta$.

Proof. Making use of the fact that $\mathbb{E}[r_f] = 0$,

$$E^g_{\mathrm{NN}}(\mathcal{X}) = \mathbb{E}[r^2(x)] = \mathbb{E}[(r_f(x) + r_g(x))^2] = \mathbb{E}[r_f^2(x)] + \mathbb{E}[r_g^2(x)].$$

Now, $r_f \sim \mathrm{GP}(0, \alpha k(\cdot, \cdot)) \implies \mathbb{E}[r_f^2(x)] = \alpha\mathbb{E}[f^2(\mathbf{x})]$, and $\sigma_g^2 - \mathbb{E}[r_g^2(\mathbf{x})] = \delta \implies \mathbb{E}[r_g^2(\mathbf{x})] = \sigma_g^2 - \delta$. So, $E^g_{\mathrm{NN}}(\mathcal{X}) = \alpha\mathbb{E}[f^2(\mathbf{x})] + \sigma_g^2 - \delta$. □

Lemma S3.3. $E^g_{\mathrm{GP+NN}}(\mathcal{X}) = \mathrm{tr}[\alpha^{-1}\Lambda^{-1} + (\sigma_n^2 + \sigma_g^2 - \delta)^{-1}\Phi\Phi^\top]^{-1} + \sigma_g^2 - \delta$.

Proof. Here, the goal of the GP is to fit the residuals $h^*(\mathbf{x}) - h_{\mathrm{NN}}(\mathbf{x}) = r(\mathbf{x}) = r_f(\mathbf{x}) + r_g(\mathbf{x})$. Since $r_f \sim \mathrm{GP}(0, \alpha k(\cdot, \cdot))$ and r_g is indistinguishable from noise, the optimal hyperparameter setting for the GP predictor is mean zero, kernel $\alpha k(\cdot, \cdot)$, and noise variance $\sigma_n^2 + \mathbb{E}[r_g^2(\mathbf{x})] = \sigma_n^2 + \sigma_g^2 - \delta$. Denote the posterior mean of the GP residual predictor by r_{GP}. Then, the final predictor for y is $h_{\mathrm{GP+NN}}(\mathbf{x}) = h_{\mathrm{NN}}(\mathbf{x}) + r_{\mathrm{GP}}(\mathbf{x})$. The expected generalization error of $h_{\mathrm{GP+NN}}$ is then

$$\begin{aligned}
E^g_{\mathrm{GP+NN}}(\mathcal{X}) &= \mathbb{E}[(h_*(\mathbf{x}) - h_{\mathrm{GP+NN}}(\mathbf{x}))^2] = \mathbb{E}[(h_*(\mathbf{x}) - h_{\mathrm{NN}}(\mathbf{x}) - r_{\mathrm{GP}}(\mathbf{x}))^2] \\
&= \mathbb{E}[(r_f(\mathbf{x}) + r_g(\mathbf{x}) - r_{\mathrm{GP}}(\mathbf{x}))^2] = \mathbb{E}[((r_f(\mathbf{x}) - r_{\mathrm{GP}}(\mathbf{x})) + r_g(\mathbf{x}))^2] \\
&= \mathbb{E}[(r_f(\mathbf{x}) - r_{\mathrm{GP}}(\mathbf{x}))^2] + 2\mathbb{E}[(r_f(\mathbf{x}) - r_{\mathrm{GP}}(\mathbf{x}))r_g(\mathbf{x})] + \mathbb{E}[r_g^2(\mathbf{x})] \\
&= \mathbb{E}[(r_f(\mathbf{x}) - r_{\mathrm{GP}}(\mathbf{x}))^2] + \mathbb{E}[r_g^2(\mathbf{x})] \\
&= \mathrm{tr}[\alpha^{-1}\Lambda^{-1} + (\sigma_n^2 + \sigma_g^2 - \delta)^{-1}\Phi\Phi^\top]^{-1} + \sigma_g^2 - \delta.
\end{aligned}$$

□

Theorem S3.4. $E^g_{\mathrm{GP+NN}}(\mathcal{X}) < E^g_{\mathrm{GP}}(\mathcal{X}) - \delta$ and $E^g_{\mathrm{GP+NN}}(\mathcal{X}) < E^g_{\mathrm{NN}}(\mathcal{X})$.

Proof. The first condition follows from Lemma S3.1, Lemma S3.3, and the fact that

$$\mathrm{tr}[\alpha^{-1}\Lambda^{-1} + (\sigma_n^2 + \sigma_g^2 - \delta)^{-1}\Phi\Phi^\top]^{-1} < \mathrm{tr}(\Lambda^{-1} + (\sigma_n^2 + \sigma_g^2)^{-1}\Phi\Phi^\top)^{-1}.$$

The second condition follows from Lemma S3.2, Lemma S3.3, and the fact that

$$\mathrm{tr}[\alpha^{-1}\Lambda^{-1} + (\sigma_n^2 + \sigma_g^2 - \delta)^{-1}\Phi\Phi^\top]^{-1} < \alpha\mathbb{E}[f^2(\mathbf{x})].$$

□

S3.3.2 Proof of Theorem S3.5.

Theorem S3.5. $E^g_{\mathrm{I/O}}(\mathcal{X}) < E^g_{\mathrm{I}}(\mathcal{X})$ and $E^g_{\mathrm{I/O}}(\mathcal{X}) < E^g_{\mathrm{O}}(\mathcal{X})$.

Proof. The fact that f_{in} and f_{out} are non-trivial implies $\mathbb{E}[f^2_{\text{in}}(x)] > 0$ and $\mathbb{E}[f^2_{\text{out}}(x)] > 0$. Following the definitions at the beginning of Section S3.3.1, let Λ_{in}, Φ_{in}, and Λ_{out}, Φ_{out} be the diagonal matrices of eigenvalues and the design matrices for the eigenfunction expansions of k_{in} and k_{out}, respectively.

With only the input kernel, that fact that h_{NN} is indistinguishable from noise from the perspective of GP implies f_{out} is also indistinguishable from noise, since it is a function of $\|h_{\mathrm{NN}}(\mathbf{x}') - h_{\mathrm{NN}}(\mathbf{x})\|$. Thus, the GP optimizer estimates the noise variance as $\sigma_n^2 + \mathbb{E}[f^2_{\text{out}}(x)]$, so the generalization error is

$$E^g_{\mathrm{I}}(\mathcal{X}) = \mathrm{tr}(\Lambda_{\text{in}}^{-1} + (\sigma_n^2 + \mathbb{E}[f^2_{\text{out}}(x)])^{-1}\Phi_{\text{in}}\Phi_{\text{in}}^\top)^{-1} + \mathbb{E}[f^2_{\text{out}}(x)].$$

Conversely, the fact that f_{out} is indistinguishable from noise from the perspective of the input kernel implies that f_{in} is indistinguishable from noise from the perspective of the output kernel. So, when only using the output kernel, the GP optimizer estimates the noise variance as $\sigma_n^2 + \mathbb{E}[f^2_{\text{in}}(x)]$. So,

$$E^g_{\mathrm{O}}(\mathcal{X}) = \mathrm{tr}(\Lambda_{\text{out}}^{-1} + (\sigma_n^2 + \mathbb{E}[f^2_{\text{in}}(x)])^{-1}\Phi_{\text{out}}\Phi_{\text{out}}^\top)^{-1} + \mathbb{E}[f^2_{\text{in}}(x)].$$

Now, when considering the I/O kernel, note that $y_i = f_{\text{in}}(\mathbf{x}_i) + f_{\text{out}}(\mathbf{x}_i) + \epsilon_i \implies y_i = f(\mathbf{x}_i) + \epsilon_i$, where $f \sim GP(0, k_{\text{in}} + k_{\text{out}})$. So, with I/O kernel, the GP optimizer correctly estimates all hyperparameters, and the resulting generalization error is

$$\begin{aligned} E^g_{\mathrm{I/O}}(\mathcal{X}) = {} & \mathrm{tr}(\Lambda_{\text{in}}^{-1} + (\sigma_n^2 + \mathbb{E}[f^2_{\text{out}}(x)])^{-1}\Phi_{\text{in}}\Phi_{\text{in}}^\top)^{-1} \\ & + \mathrm{tr}(\Lambda_{\text{out}}^{-1} + (\sigma_n^2 + \mathbb{E}[f^2_{\text{in}}(x)])^{-1}\Phi_{\text{out}}\Phi_{\text{out}}^\top)^{-1}. \end{aligned}$$

Then,

$$\mathrm{tr}(\Lambda_{\text{out}}^{-1} + (\sigma_n^2 + \mathbb{E}[f^2_{\text{in}}(x)])^{-1}\Phi_{\text{out}}\Phi_{\text{out}}^\top)^{-1} < \mathbb{E}[f^2_{\text{out}}(x)] \implies E^g_{\mathrm{I/O}}(\mathcal{X}) < E^g_{\mathrm{I}}(\mathcal{X}),$$

and

$$\mathrm{tr}(\Lambda_{\text{in}}^{-1} + (\sigma_n^2 + \mathbb{E}[f^2_{\text{out}}(x)])^{-1}\Phi_{\text{in}}\Phi_{\text{in}}^\top)^{-1} < \mathbb{E}[f^2_{\text{in}}(x)] \implies E^g_{\mathrm{I/O}}(\mathcal{X}) < E^g_{\mathrm{O}}(\mathcal{X}).$$

□

S4 Empirical Study

S4.1 Experimental Setups

Dataset Description In total, twelve real-world regression datasets from UCI machine learning repository [5] are tested. Table S1 summarizes the basic information of these datasets. For all the datasets, 20% of the whole dataset is used as test dataset and 80% is used as training dataset, and this split is randomly generated in each independent run. During the experiments, all the datasets are tested for 100 independent runs. For each independent run, the same random dataset split are used by all the tested algorithms to ensure fair comparisons. NNGP and ANP are only tested on the four smallest dataset (based on the product of dataset size and feature dimensionality) because they do not scale well to larger datasets. A downloadable link for all source codes is provided at: (https://drive.google.com/open?id=1ZZkoCgOxSG6U1CGgRHkgxJIZhL1B0db9).

Table S1: Summary of testing dataset

abbreviation	full name in UCI ML repository	dataset size	dimension	note
yacht	Yacht Hydrodynamics Data Set	252	6	-
ENB/h	Energy efficiency	768	8	Heating Load as target
ENB/c	Energy efficiency	768	8	Cooling Load as target
airfoil	Airfoil Self-Noise	1505	5	-
CCS	Concrete Compressive Strength	1030	8	-
wine/w	Wine Quality	4898	11	only use winequality-white data
CCPP	Combined Cycle Power Plant	9568	4	-
CASP	Physicochemical Properties of Protein Tertiary Structure	54730	9	-
SC	Superconductivty Data	21263	80	-
CT	Relative location of CT slices on axial axis	53500	384	-

Parametric Setup for Algorithms

- NN: For SC dataset, a fully connected feed-forward NN with 2 hidden layers, each with 128 hidden neurons, is used. For CT dataset, a fully connected feed-forward NN with 2 hidden layers, each with 256 hidden neurons, is used. For all the remaining datasets, a fully connected feed-forward NN with 2 hidden layers, each with 64 hidden neurons, is used. The inputs to the NN are normalized to have mean 0 and standard deviation 1. The activation function is ReLU for all the hidden layers. The maximum number of epochs for training is 1000. 20% of the training data is used as validation data, and the split is random at each independent run. An early stop is triggered if the loss on validation data has not be improved for 10 epochs. The optimizer is RMSprop with learning rate 0.001, and the loss function is mean squared error (MSE).
- RIO, RIO variants and SVGP [10]: SVGP is used as an approximator to original GP in RIO and all the RIO variants. For RIO, RIO variants and SVGP, the number of inducing points are 50 for all the experiments. RBF kernel is used for both input and output kernel. For RIO, RIO variants and SVGP, the signal variances and length scales of all the kernels plus the noise variance are the trainable hyperparameters. The optimizer is L-BFGS-B with default parameters as in Scipy.optimize documentation (https://docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html), and the maximum number of iterations is set as 1000. The training process runs until the L-BFGS-B optimizer decides to stop.
- NNGP [16]: For NNGP kernel, the depth is 2, and the activation function is ReLU. $n_g = 101$, $n_v = 151$, and $n_c = 131$. Following the learning process in original paper, a grid search is performed to search for the best values of σ_w^2 and σ_b^2. Same as in the original paper, a grid of 30 points evenly spaced from 0.1 to 5.0 (for σ_w^2) and 30 points evenly spaced from 0 to 2.0 (for σ_b^2) was evaluated. The noise variance σ_ϵ^2 is fixed as 0.01. The grid search process stops when Cholesky decomposition fails or all the 900 points are evaluated. The best values found during the grid search will be used in the experiments. No pre-computed lookup tables are utilized.
- ANP [13]: The parametric setups of ANP are following the recommendations in the original paper. The attention type is multihead, the hidden size is 64, the max number of context points is 50, the context ratio is 0.8, the random kernel hyperparameters option is on. The size of latent encoder is $64 \times 64 \times 64 \times 64$, the number of latents is 64, the size of deterministic encoder is $64 \times 64 \times 64 \times 64$, the size of decoder is $64 \times 64 \times 64 \times 64 \times 2$, and the deterministic path option is on. Adam optimizer with learning rate 10^{-4} is used, and the maximum number of training iterations is 2000.

Performance Metrics

- To measure the point-prediction error, the Root Mean Square Error (RMSE) between the method predictions and true outcomes on test datasets are calculated for each independent experimental run. After that, the mean and standard deviations of these RMSEs are used to measure the performance of the algorithms.
- To quantitatively measure the quality of uncertainty estimation, average negative log predictive density (NLPD) [21] is used to measure the quality of uncertainty estimation. NLPD is

given by

$$L = -\frac{1}{n}\sum_{i=1}^{n} \log p(\hat{\mathbf{y}}_i = \mathbf{y}_i | \mathbf{x}_i) \tag{1}$$

where $\hat{\mathbf{y}}_i$ indicates the prediction results, $\mathbf{x}_i$ is the input with true associated outcome $\mathbf{y}_i$, $p(\cdot)$ is the probability density function (PDF) of the returned distribution based on input $\mathbf{x}_i$.

- To compare the computation time of the algorithms, the training time (wall clock time) of NN, RIO, all the RIO variants, SVGP and ANP are averaged over all the independent runs as the computation time. For NNGP, the wall clock time for the grid search is used. In case that the grid search stops due to Cholesky decomposition failures, the computation time of NNGP will be estimated as the average running time of all the successful evaluations $\times$ 900, which is the supposed number of evaluations. All the algorithms are implemented using Tensorflow, and tested in the exactly same python environment. All the experiments are running on a machine with 16 Intel(R) Xeon(R) CPU E5-2623 v4@2.60GHz and 128GB memory.

References

[1] D. Barber and C. Bishop. Ensemble learning in bayesian neural networks. In *Generalization in Neural Networks and Machine Learning*, pages 215–237. Springer Verlag, January 1998.

[2] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1613–1622. JMLR.org, 2015.

[3] B. Csaji. Approximation with artificial neural networks. *M.S.'Thesis, Dept. Science, Eotvos Lorand Univ., Budapest, Hungary*, 2001.

[4] L. Csató and M. Opper. Sparse on-line gaussian processes. *Neural computation*, 14:641–68, 04 2002.

[5] D. Dua and C. Graff. UCI machine learning repository, 2017.

[6] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1050–1059. JMLR.org, 2016.

[7] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. M. A. Eslami. Conditional neural processes. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1704–1713. PMLR, 10–15 Jul 2018.

[8] M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. M. A. Eslami, and Y. W. Teh. Neural processes. *CoRR*, abs/1807.01622, 2018.

[9] A. Graves. Practical variational inference for neural networks. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2348–2356, USA, 2011. Curran Associates Inc.

[10] J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, UAI'13, pages 282–290, Arlington, Virginia, United States, 2013. AUAI Press.

[11] J. Hensman, A. Matthews, and Z. Ghahramani. Scalable Variational Gaussian Process Classification. In G. Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 351–360, San Diego, California, USA, 09–12 May 2015. PMLR.

[12] G. E. Hinton and D. van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, pages 5–13, New York, NY, USA, 1993. ACM.

[13] H. Kim, A. Mnih, J. Schwarz, M. Garnelo, S. M. A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh. Attentive neural processes. *CoRR*, abs/1901.05761, 2019.

[14] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6402–6413. Curran Associates, Inc., 2017.

[15] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 265–272, USA, 2011. Omnipress.

[16] J. Lee, Y. Bahri, R. Novak, S. Schoenholz, J. Pennington, and J. Sohl-dickstein. Deep neural networks as gaussian processes. *International Conference on Learning Representations*, 2018.

[17] D. J. C. MacKay. A practical bayesian framework for backpropagation networks. *Neural Comput.*, 4(3):448–472, May 1992.

[18] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996.

[19] M. Opper and F. Vivarelli. General bounds on bayes errors for regression with gaussian processes. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, pages 302–308, Cambridge, MA, USA, 1999. MIT Press.

[20] J. Quiñonero Candela and C. E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *J. Mach. Learn. Res.*, 6:1939–1959, Dec. 2005.

[21] J. Quiñonero-Candela, C. E. Rasmussen, F. Sinz, O. Bousquet, and B. Schölkopf. Evaluating predictive uncertainty challenge. In J. Quiñonero-Candela, I. Dagan, B. Magnini, and F. d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 1–27, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[22] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Jan. 2006.

[23] M. Seeger, C. K. I. Williams, and N. D. Lawrence. Fast forward selection to speed up sparse gaussian process regression. In *IN WORKSHOP ON AI AND STATISTICS 9*, 2003.

[24] P. Sollich. Learning curves for gaussian processes. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, pages 344–350, Cambridge, MA, USA, 1999. MIT Press.

[25] M. K. Titsias. Variational learning of inducing variables in sparse gaussian processes. In *In Artificial Intelligence and Statistics 12*, pages 567–574, 2009.

[26] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep kernel learning. In A. Gretton and C. C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 370–378, Cadiz, Spain, 09–11 May 2016. PMLR.