PI3NN: Out-of-distribution-aware prediction intervals from three neural networks

Anonymous authors
Paper under double-blind review

Abstract

We propose a novel prediction interval (PI) method for uncertainty quantification, which addresses three major issues with the state-of-the-art PI methods. First, existing PI methods require retraining of neural networks (NNs) for every given confidence level and suffer from the crossing issue in calculating multiple PIs. Second, they usually rely on customized loss functions with extra sensitive hyperparameters for which fine tuning is required to achieve a well-calibrated PI. Third, they usually underestimate uncertainties of out-of-distribution (OOD) samples leading to over-confident PIs. Our PI3NN method calculates PIs from linear combinations of three NNs, each of which is independently trained using the standard mean squared error loss. The coefficients of the linear combinations are computed using root-finding algorithms to ensure tight PIs for a given confidence level. We theoretically prove that PI3NN can calculate PIs for a series of confidence levels without retraining NNs and it completely avoids the crossing issue. Additionally, PI3NN does not introduce any unusual hyperparameters resulting in a stable performance. Furthermore, we address OOD identification challenge by introducing an initialization scheme which provides reasonably larger PIs of the OOD samples than those of the in-distribution samples. Benchmark and real-world experiments show that our method outperforms several state-of-the-art approaches with respect to predictive uncertainty quality, robustness, and OOD samples identification.

1 Introduction

Neural networks (NNs) are widely used in prediction tasks due to their unrivaled performance in modeling complex functions. Although NNs provide accurate predictions, quantifying the uncertainty of their predictions is a challenge. Uncertainty quantification (UQ) is crucial for many real-world applications such as self-driving cars and autonomous experimental and operational controls. Moreover, effectively quantified uncertainties are useful for interpreting confidence, capturing out-of-distribution (OOD) data, and realizing when the model is likely to fail.

A diverse set of UQ approaches have been developed for NNs, ranging from fully Bayesian NNs [1], to assumption-based variational inference [2; 3], and to empirical ensemble approaches [4; 5; 6]. These methods require either high computational demands or strong assumptions or large memory costs. Another set of UQ methods is to calculate prediction intervals (PIs), which provides a lower and upper bound for an NN's output such that the value of the prediction falls between the bounds for some target confidence level γ (e.g., 95%) of the unseen data. The most common techniques to construct the PI are the delta method (also known as analytical method) [7; 8], methods that directly predict the variance (e.g., maximum likelihood methods and ensemble methods) [9; 10] and quantile regression methods [11; 12]. Most recent PI methods are developed on the high-quality principle—a PI should be as narrow as possible, whilst capturing a specified portion of data. Khosravi et al. [13] developed the Lower Upper Bound Estimation method by incorporating the high-quality principle directly into the NN loss function for the first time. Inspired by [13], the QD approach in [14] defined a loss function that can generate a high-quality PI and is able to optimize the loss using stochastic gradient descent as well. Built on QD, the PIVEN method in [15] adds an extra term in the loss to enable the calculation of point estimates and the PI method in [16] further integrates a penalty function to the loss to improve the training stability of QD. The SQR [17] method proposed a loss function to learn all quantiles of a target variable with one NN. Existing PI methods suffer from one or more of the following limitations: (1) Requirement of NNs retraining for every given confidence level γ and suffering from the crossing issue [18] when calculating PIs for differnt γ; (2) Requirement of hyper-parameter fine tuning to get a well-calibrate PI; (3) Lack of OOD identification capability resulting in unreasonably narrow PIs for OOD samples (See Section 2.1 for details).

To address these limitations, we develop PI3NN (prediction interval based on three neural networks)—a novel method for calculating PIs. We first lay out the theoretical foundation of the PI3NN in Section 3.1 by proving Lemma 1 that connects the ground-truth upper and lower bounds of a PI to a family of models that are easy to approximate. Another advantage of the model family introduced in Lemma 1 is that it makes the NN training independent of the target confidence level, which makes it possible to calculate multiple PIs for a series of confidence levels without retraining NNs. On the basis of the theoretical foundation, we describe the main PI3NN algorithm in Section 3.2. Different from existing PI methods [14; 15; 16; 17] that design complicated loss functions to obtain a well-calibrated PI by fine-tuning their loss hyper-parameters, our method simply uses the standard mean squared error (MSE) loss for training. Additionally, we theoretically prove that PI3NN has a non-crossing property in Section 3.2.1. Moreover, we address the OOD identification challenge by proposing a simple yet effective initialization scheme in Section 3.3, which provides larger PIs of the OOD samples than those of the in-distribution (InD) samples.

The main contributions of this work are summarized as follows:

1. Our PI3NN method can calculate PIs for a series of confidence levels without retraining NNs; and the calculated PIs completely avoid the crossing issue as proved in Section 3.2.1.
2. The theoretical foundation in Section 3.1 enables PI3NN to use the standard MSE loss to train three NNs without introducing extra hyper-parameters that need to be fine-tuned for a good PI.
3. We develop a simple yet effective initialization scheme and a confidence score in Section 3.3 to identify OOD samples and reasonably quantify their uncertainty.

1.1 Related work

Non-PI approaches for UQ. Early and recent work was nicely summarized and reviewed in these three survey papers [19; 20; 21]. The non-PI approaches use a distribution to quantify uncertainty, which can be further divided into Bayesian [1] and non-Bayesian methods. Bayesian methods—including Markov chain Monte Carlo [22] and Hamiltonian Monte Carlo [23]—place priors on NN weights and then infer a posterior distribution from the training data. Non-Bayesian methods includes evidential regression [6] that places priors directly over the likelihood function and some ensemble learning methods that do not use priors. For example, the DER method proposed in [6] placed evidential priors over the Gaussian likelihood function and training the NN to infer the hyperparameters of the evidential distribution. Gal and Ghahramani [3] proposed using Monte Carlo dropout to estimate predictive uncertainty by using Dropout (which can be interpreted as ensemble model combination) at test time. Deep ensembles [4] employed a combination of ensembles of NNs learning and adversarial training to quantify uncertainty with a Gaussian distributional assumption on the data. Pearce et al. [5] proposed an anchored ensembling by using the randomized MAP sampling to increase the diversity of NN training in the ensemble.

2 Background

We are interested in building PIs for the output of the regression problem $y = f(\boldsymbol{x}) + \varepsilon$ from a training set $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, where $\boldsymbol{x} \in \mathbb{R}^d$, $y \in \mathbb{R}$, and ε is the random noise. We do not impose distributional assumption on the noise ε. Since the output of $f(\boldsymbol{x})$ is polluted by the noise, the output y of the regression model $y = f(\boldsymbol{x}) + \varepsilon$ is also a random variable. For a given confidence level $\gamma \in (0, 1)$, the ground-truth $100\gamma\%$ PI, denoted by $[L_\gamma^{\text{true}}(\boldsymbol{x}), U_\gamma^{\text{true}}(\boldsymbol{x})]$, is defined by

$$\mathbb{P}[L_\gamma^{\text{true}}(\boldsymbol{x}) \leq y \leq U_\gamma^{\text{true}}(\boldsymbol{x})] = \gamma. \tag{1}$$

Note that $L_\gamma^{\text{true}}(\boldsymbol{x})$ and $U_\gamma^{\text{true}}(\boldsymbol{x})$ are not unique for a fixed γ in the definition of Eq. (1), because the probability of y outside the PI, i.e., $\mathbb{P}[y > U_\gamma^{\text{true}}(\boldsymbol{x})$ or $y < L_\gamma^{\text{true}}(\boldsymbol{x})]$, may be split in any way between the two tails. In this work, we aim to approximate the PI that satisfies

$$\mathbb{P}[y > U_\gamma^{\text{true}}(\boldsymbol{x})] = (1-\gamma)/2 \quad \text{and} \quad \mathbb{P}[y < L_\gamma^{\text{true}}(\boldsymbol{x})] = (1-\gamma)/2, \tag{2}$$

which is *unique* because the probability outside the PI is equally split between the two tails.

2.1 Motivation

Recent effort on PI methods, e.g., QD, SQR and PIVEN, tend to exploit the NNs to learn the upper and lower bounds in Eq. (2). Despite their promising performance, these methods suffer from some unsatisfactory drawbacks. This effort is motivated by their following limitations:

- *Requirement of retraining NNs for every given γ and the crossing issue when calculating multiple PIs.* Existing PI methods usually incorporate γ into their loss functions for training NNs, so that each NN can only predict PI for a specific γ, which is not convenient for users. On the other hand, even multiple NNs can be trained for PIs with multiple γ values, the approximate PIs often encounter the crossing issue, e.g., the upper bounds for different γ values may cross each other, which is not reasonable. To alleviate this issue, a non-crossing constraint is usually added the loss as a regularization to encourage non-crossing PIs. However, the non-crossing constraint may deteriorate the quality of the approximate PI, because due to the trade-off between the original loss and the non-crossing constraint.
- *Requiring hyper-parameter fine tuning.* Recently developed PI methods [14; 15; 16] tend to design complicated loss functions to obtain a well-calibrated PI. Although these work has achieved promising results, their performance is sensitive to the unusual hyperparameters introduced into their customized loss functions. Thus, hyperparameter fine tuning is usually required for each specific problem to achieve satisfactory upper and lower bounds.
- *Lack of OOD identification capability*. OOD identification is a critical metric to evaluate the performance of an UQ method. It has been received significant attention in recent UQ method development in the machine learning community. However, the OOD identification has not been deeply studied for PI methods in solving the regression problem. Even though there are some promising empirical results on OOD-aware PIs [15], the underlying mechanism is still not clear, making it difficult to extend the occasional success to a general setting.

3 The PI3NN method

The main contribution is presented in this section. Section 3.1 shows a theoretical justification of our method, where Lemma 1 plays a critical role to connect the ground-truth upper and lower bounds to Eq. (5) and Eq. (6) that are easier to approximate. Section 3.2 introduces the main PI3NN algorithm inspired by Lemma 1. Section 3.3 describes how to turn on the OOD identification capability in the PI3NN method. The features of our methods are illustrated using a simple example in Section 3.4.

3.1 Theoretical justification

To proceed, we first rewrite Eq. (2) to an equivalent form

$$\mathbb{E}\big[\mathbf{1}_{y>U_\gamma^{\rm true}(\boldsymbol{x})}\big] - (1-\gamma)/2 = 0 \quad \text{and} \quad \mathbb{E}\big[\mathbf{1}_{y<L_\gamma^{\rm true}(\boldsymbol{x})}\big] - (1-\gamma)/2 = 0, \tag{3}$$

where $\mathbf{1}_{(\cdot)}$ is the indicator function, defined by

$$\mathbf{1}_{y>U_\gamma^{\rm true}(\boldsymbol{x})} = \begin{cases} 1, & \text{if } y > U_\gamma^{\rm true}(\boldsymbol{x}), \\ 0, & \text{otherwise,} \end{cases} \quad \text{and} \quad \mathbf{1}_{y<L_\gamma^{\rm true}(\boldsymbol{x})} = \begin{cases} 1, & \text{if } y < L_\gamma^{\rm true}(\boldsymbol{x}), \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

For simplicity, we take $U_\gamma^{\rm true}(\boldsymbol{x})$ as an example in the rest of Section 3.1, and the same derivation can be applied to $L_\gamma^{\rm true}(\boldsymbol{x})$. In definition, $U_\gamma^{\rm true}(\boldsymbol{x})$ has the following three properties:

(P1) $U_\gamma^{\rm true}(\boldsymbol{x}) \geq \mathbb{M}[y] = f(\boldsymbol{x}) + \mathbb{M}[\varepsilon]$, where $\mathbb{M}[\cdot]$ denotes the median of a random variable,

(P2) $U_\gamma^{\rm true}(\boldsymbol{x}) - f(\boldsymbol{x})$ is independent of $\boldsymbol{x}$, because ε is independent of $\boldsymbol{x}$,

(P3) $U_\gamma^{\rm true}(\boldsymbol{x})$ is unique for a given confidence level $\gamma \in (0,1)$.

Next, we show that the ground-truth $U_\gamma^{\rm true}(\boldsymbol{x})$ is a member of the following model family:

$$\hat{U}(\boldsymbol{x}|\alpha) = \mathbb{M}[y] + \alpha\, \mathbb{E}\left[(y - \mathbb{M}[y])\mathbf{1}_{y-\mathbb{M}[y]>0}\right], \tag{5}$$

where $\alpha \geq 0$ is a scalar model parameter and $\mathbb{M}[\cdot]$ denotes the median of y. To this end, we prove the following lemma:

Lemma 1. *For a given $\gamma \in (0,1)$, there exists a unique $\alpha(\gamma)$, such that $\hat{U}(\boldsymbol{x}|\alpha(\gamma)) = U_\gamma^{\rm true}(\boldsymbol{x})$.*

The proof of this lemma is given in Appendix A. The same result can be derived for the lower bound, i.e., for a given $\gamma \in (0,1)$, there exists a unique $\beta(\gamma)$, such that $\hat{L}(\boldsymbol{x}|\beta(\gamma)) = L_\gamma^{\rm true}(\boldsymbol{x})$, where $\hat{L}(\boldsymbol{x}|\beta)$ is defined by

$$\hat{L}(\boldsymbol{x}|\beta) = \mathbb{M}[y] - \beta\,\mathbb{E}\left[(\mathbb{M}[y]-y)\mathbf{1}_{\mathbb{M}[y]-y>0}\right]. \tag{6}$$

Lemma 1 connects the ground-truth upper and lower bounds to the model families $\hat{U}(\boldsymbol{x}|\alpha)$ and $\hat{L}(\boldsymbol{x}|\beta)$, such that the task of approximating $U_\gamma^{\rm true}(\boldsymbol{x})$ and $L_\gamma^{\rm true}(\boldsymbol{x})$ can be decomposed into two sub-tasks:

- Approximate the models $\hat{U}(\boldsymbol{x}|\alpha)$ and $\hat{L}(\boldsymbol{x}|\beta)$ by training NNs;
- Calculate the optimal values of $\alpha(\gamma)$ and $\beta(\gamma)$ for any $\gamma \in (0,1)$.

3.2 The main algorithm

PI3NN accomplishes the above two sub-tasks in four steps. *Step 1-3* build approximations of $\hat{U}(\boldsymbol{x}|\alpha)$ and $\hat{L}(\boldsymbol{x}|\beta)$ by training three independent NNs, denoted by $f_{\boldsymbol{\omega}}(\boldsymbol{x})$, $u_{\boldsymbol{\theta}}(\boldsymbol{x})$, $l_{\boldsymbol{\xi}}(\boldsymbol{x})$, which approximate $\mathbb{M}[y]$, $\mathbb{E}[(y-\mathbb{M}[y])\mathbf{1}_{y-\mathbb{M}[y]>0}]$, and $\mathbb{E}[(\mathbb{M}[y]-y)\mathbf{1}_{\mathbb{M}[y]-y>0}]$, respectively. After the three NNs are trained, we calculate the optimal values of $\alpha(\gamma)$ and $\beta(\gamma)$ in *Step 4* using root-finding techniques.

Step 1: Train $f_{\boldsymbol{\omega}}(\boldsymbol{x})$ to approximate the mean $\mathbb{E}[y]$. This step follows the standard NN-based regression process using the standard MSE loss. The trained $f_{\boldsymbol{\omega}}(\boldsymbol{x})$ will serve two purposes. The first is to provide a baseline to approximate $\mathbb{M}[y]$ in *Step 2*; the second is to provide a point estimate of $\mathbb{E}[f]$. In this step, we use the standard L_1 and L_2 regularization to avoid over-fitting.

Step 2: Add a shift ν to $f_{\boldsymbol{\omega}}(\boldsymbol{x})$ to approximate the median $\mathbb{M}[y]$. A scalar ν is added to $f_{\boldsymbol{\omega}}(\boldsymbol{x})$, such that each of the two data sets $\mathcal{D}_{\rm upper}$ and $\mathcal{D}_{\rm lower}$, defined by

$$\begin{aligned}\mathcal{D}_{\rm upper} &= \big\{(\boldsymbol{x}_i, y_i - f_{\boldsymbol{\omega}}(\boldsymbol{x}_i) - \nu)\;\big|\; y_i \ge f_{\boldsymbol{\omega}}(\boldsymbol{x}_i) + \nu, i = 1,\dots,N\big\},\\ \mathcal{D}_{\rm lower} &= \big\{(\boldsymbol{x}_i, f_{\boldsymbol{\omega}}(\boldsymbol{x}_i) + \nu - y_i)\;\big|\; y_i < f_{\boldsymbol{\omega}}(\boldsymbol{x}_i) + \nu, i = 1,\dots,N\big\},\end{aligned} \tag{7}$$

contains 50% of the total training samples in $\mathcal{D}_{\rm train}$. Note that $\mathcal{D}_{\rm upper}$ and $\mathcal{D}_{\rm lower}$ include data points above and below $f_{\boldsymbol{\omega}}(\boldsymbol{x}) + \nu$, respectively. The value of the shift ν is calculated using a root-finding method [24] (e.g., the bisection method) to find the root (i.e., the zero) of the function $Q(\nu) = \sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_{\rm train}} \mathbf{1}_{y_i > f_\omega(\boldsymbol{x}_i)+\nu} - 0.5N$. This is similar to finding the roots of Eq. (10), so we refer to Lemma 2 for the existence of the root.

Step 3: Train $u_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $l_{\boldsymbol{\xi}}(\boldsymbol{x})$ to learn $\mathbb{E}[(y-\mathbb{M}[y])\mathbf{1}_{y-\mathbb{M}[y]>0}]$ and $\mathbb{E}[(\mathbb{M}[y]-y)\mathbf{1}_{\mathbb{M}[y]-y>0}]$. We use $\mathcal{D}_{\rm upper}$ to train $u_{\boldsymbol{\theta}}(\boldsymbol{x})$, and use $\mathcal{D}_{\rm lower}$ to train $l_{\boldsymbol{\xi}}(\boldsymbol{x})$. To ensure the outputs of $u_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $l_{\boldsymbol{\xi}}(\boldsymbol{x})$ are positive, we add the operation $\sqrt{(\cdot)^2}$ to the output layer of both NNs. The two NNs are trained separately using the standard MSE loss, i.e.,

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \sum_{\mathcal{D}_{\rm upper}} (y_i - f_{\boldsymbol{\omega}}(\boldsymbol{x}_i) - \nu - u_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2, \quad \boldsymbol{\xi} = \arg\min_{\boldsymbol{\xi}} \sum_{\mathcal{D}_{\rm lower}} (f_{\boldsymbol{\omega}}(\boldsymbol{x}_i) + \nu - y_i - l_{\boldsymbol{\xi}}(\boldsymbol{x}_i))^2. \tag{8}$$

Step 4: Calculate the PI via root-finding methods. Using the three NNs trained in *Step 1-3*, we build approximations of $\hat{U}(\boldsymbol{x}|\alpha)$ and $\hat{L}(\boldsymbol{x}|\alpha)$, denoted by

$$U(\boldsymbol{x}|\alpha) = f_{\boldsymbol{\omega}}(\boldsymbol{x}) + \nu + \alpha u_{\boldsymbol{\theta}}(\boldsymbol{x}), \qquad L(\boldsymbol{x}|\beta) = f_{\boldsymbol{\omega}}(\boldsymbol{x}) + \nu - \beta l_{\boldsymbol{\xi}}(\boldsymbol{x}). \tag{9}$$

For a sequence of confidence levels $\Gamma = \{\gamma_k\}_{k=1}^K$, we use the bisection method [24] to calculate the optimal value of $\alpha(\gamma_k)$ and $\beta(\gamma_k)$ by finding the roots (i.e., the zeros) of the following functions:

$$\begin{aligned} Q_{\rm upper}(\alpha) &= \sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_{\rm upper}} \mathbf{1}_{y_i > U(\boldsymbol{x}_i|\alpha)} - \lceil N(1-\gamma_k)/2\rceil,\\ Q_{\rm lower}(\beta) &= \sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_{\rm lower}} \mathbf{1}_{y_i < L(\boldsymbol{x}_i|\beta)} - \lceil N(1-\gamma_k)/2\rceil,\end{aligned} \tag{10}$$

for $k = 1,\dots,K$. Note that Eq. (10) can be reviewed as the discrete version of Eq. (3) and/or Eq. (13). When $\mathcal{D}_{\rm upper}$ and $\mathcal{D}_{\rm lower}$ have finite number of data points, it is easy to find $\alpha(\gamma)$ and $\beta(\gamma)$

such that $Q_{\text{upper}}(\alpha(\gamma)) = Q_{\text{lower}}(\beta(\gamma)) = 0$ up to the machine precision (See Section 3.2.1 for details). Then, the final K PIs for the confidence levels $\Gamma = \{\gamma_k\}_{k=1}^K$ is given by

$$\big[L(\boldsymbol{x}|\beta(\gamma_k)), U(\boldsymbol{x}|\alpha(\gamma_k))\big] = \big[f_{\boldsymbol{\omega}}(\boldsymbol{x}) + \nu - \beta(\gamma_k) l_{\boldsymbol{\xi}}(\boldsymbol{x}), f_{\boldsymbol{\omega}_k}(\boldsymbol{x}) + \nu + \alpha(\gamma_k) u_{\boldsymbol{\theta}}(\boldsymbol{x})\big]. \tag{11}$$

Remark 1 (Generate PIs for multiple γ without re-training the NNs)**.** *Note that the NN training in* Step 1-3 *is independent of the confidence level γ, so the PIs for multiple γ can be obtained by only performing the root finding algorithm in* Step 4 *without re-training the three NNs. In contrast, most existing PI methods (e.g., QD and PIVEN) require re-training NNs when γ is changed. This feature, ultimately enabled by Lemma 1, makes our method more efficient and convenient to use in practice.*

3.2.1 Theoretical analysis on the non-crossing property

The goal of this subsection is to theoretically prove that our algorithm can completely avoid the "crossing issue" [18] suffered by most existing PI methods without adding any non-crossing constraints to the loss function. To proceed, we first prove the existence of the roots of the functions in Eq. (10).

Lemma 2 (Existence of the roots)**.** *Given the trained models $f_{\boldsymbol{\omega}}(\boldsymbol{x})$, $u_{\boldsymbol{\theta}}(\boldsymbol{x})$, $l_{\boldsymbol{\xi}}(\boldsymbol{x})$ and ν, if the training set $\mathcal{D}_{\text{train}}$ does not have repeated samples (i.e., $(\boldsymbol{x}_i, y_i) \neq (\boldsymbol{x}_j, y_j)$ if $i \neq j$), then, for any fixed $\gamma > 0$, there exist $\alpha(\gamma)$ and $\beta(\gamma)$ such that $Q_{\text{upper}}(\alpha(\gamma)) = Q_{\text{lower}}(\beta(\gamma)) = 0$.*

The proof of this lemma is given in Appendix B. In practice, the reason why it is easy to find the roots of Q_{upper} and Q_{lower} is that the number of training samples are finite, so that the bisection method can vary α and β *continuously* in $[0, \infty)$ to search for "gaps" among the data points to find the zeros of $Q_{\text{upper}}(\alpha)$ and $Q_{\text{lower}}(\beta)$. Below is the theorem about the non-crossing property of the PIs obtained by our method. Lemma 2 will be used to in the proof of the theorem.

Theorem 1 (Non-crossing property)**.** *Given the trained models $f_{\boldsymbol{\omega}}$, $u_{\boldsymbol{\theta}}$, $l_{\boldsymbol{\xi}}$ and ν, if $\gamma > \gamma'$ satisfying $\lceil N(1-\gamma)/2 \rceil < \lceil N(1-\gamma')/2 \rceil$, then $U(\boldsymbol{x}|\alpha(\gamma)) > U(\boldsymbol{x}|\alpha(\gamma'))$ and $L(\boldsymbol{x}|\beta(\gamma)) > L(\boldsymbol{x}|\beta(\gamma'))$.*

The proof of this theorem is given in Appendix C. Theorem 1 tells us that the width of the PI obtained by our method will monotonically increasing with γ, which completely avoids the crossing issue. Again, this property is ultimately enabled by the equivalent representation of the PI in Lemma 1.

3.3 Identifying out-of-distribution (OOD) samples

When using the trained model $f_{\boldsymbol{\omega}}(\boldsymbol{x})$ to make predictions for $\boldsymbol{x} \notin \mathcal{D}_{\text{train}}$, it is required that a UQ method can identify OOD samples and reasonably quantify their uncertainties, i.e., for $\boldsymbol{x} \notin \mathcal{D}_{\text{train}}$, the PI's width increases with the distance between $\boldsymbol{x}$ and $\mathcal{D}_{\text{train}}$. Inspired by the expressivity of ReLU networks, we add the OOD identification feature to the PI3NN method by initializing the bias of the output layer of $u_{\boldsymbol{\theta}}$ and $l_{\boldsymbol{\xi}}$ to a relatively large value. Specifically, when the OOD identification feature is turned on, we perform the following initialization before training $u_{\boldsymbol{\theta}}$ and $l_{\boldsymbol{\xi}}$ in *Step 3*.

- Pretrain $u_{\boldsymbol{\theta}}$ and $l_{\boldsymbol{\xi}}$ using the training set $\mathcal{D}_{\text{train}}$ with default initialization and compute the mean outputs $\mu_{\text{upper}} = \sum_{i=1}^N u_{\boldsymbol{\theta}}(\boldsymbol{x}_i)/N$ and $\mu_{\text{lower}} = \sum_{i=1}^N l_{\boldsymbol{\xi}}(\boldsymbol{x}_i)/N$ based on the training set.
- Initialize the biases of the output layers of $u_{\boldsymbol{\theta}}$ and $l_{\boldsymbol{\xi}}$ to $c\,\mu_{\text{upper}}$ and $c\,\mu_{\text{lower}}$, where c is a large number (e.g., $c = 10$ in this work), such that the initial outputs of $u_{\boldsymbol{\theta}}$ and $l_{\boldsymbol{\xi}}$ are significantly larger than μ_{upper} and μ.
- Re-train $u_{\boldsymbol{\theta}}$ and $l_{\boldsymbol{\xi}}$ following *Step 3* using the MSE loss.

The key in above initialization scheme is the increase of the bias of the output layer of $u_{\boldsymbol{\theta}}$ and $l_{\boldsymbol{\xi}}$. It is known that a ReLU network provides a piecewise linear function. The weights and biases of hidden layers defines how the input space is partitioned into a set of linear regions [25]; the weights of the output layer determines how those linear regions are combined; and the bias of the output layer acts as a shifting parameter. The weights and biases are usually initialized with some standard distribution, e.g., uniform or Gaussian. Setting the biases to $c\mu_{\text{upper}}$ and $c\mu_{\text{lower}}$ with a large value of c will significantly increase the output of the initial $u_{\boldsymbol{\theta}}$ and $l_{\boldsymbol{\xi}}$ (See Figure 1 for demonstration). During the training, the loss in Eq. (8) will encourage the decrease of $u_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $l_{\boldsymbol{\xi}}(\boldsymbol{x})$ only for the in-distribution (InD) samples (i.e., $\boldsymbol{x} \in \mathcal{D}_{\text{train}}$), not for the OOD samples. Therefore, after training, the PI will be wider in the OOD region than in the InD region. Additionally, due to continuity of the ReLU network, the PI width (PIW) will increase with the distance between $\boldsymbol{x}$ and $\mathcal{D}_{\text{train}}$ showing a decreasing confidence. Moreover, we can define the following *confidence score* by exploiting the width of the PI, i.e.,

$$\Lambda(\boldsymbol{x}) = \min\left\{ \frac{\sum_{i=1}^{N}(U(\boldsymbol{x}_i|\alpha(\gamma)) - L(\boldsymbol{x}_i|\beta(\gamma))/N}{U(\boldsymbol{x}|\alpha(\gamma)) - L(\boldsymbol{x}|\beta(\gamma))}, 1.0 \right\}, \tag{12}$$

where the numerator is the mean PI width (MPIW) of the training set $\mathcal{D}_{\text{train}}$ and the denominator is the PIW of a testing sample $\boldsymbol{x}$. If $\boldsymbol{x}$ is an InD sample, its confidence score should be close to one. As $\boldsymbol{x}$ moves away from $\mathcal{D}_{\text{train}}$, the PIW gets larger and thus the confidence score becomes smaller.

3.4 An illustrative example

We use a one-dimensional non-Gaussian cubic regression dataset to illustrate PI3NN. We train models on $y = x^3 + \varepsilon$ within $[-4, 4]$ and test them on $[-7, 7]$. The noise ε is defined by $\varepsilon = s(\zeta)\zeta$, where $\zeta \sim \mathcal{N}(0, 1)$, $s(\zeta) = 30$ for $\zeta \geq 0$ and $s(\zeta) = 10$ for $\zeta < 0$. Top panels of Figure 1 illustrate the four steps of the PI3NN algorithm. After *Step 1-3* we finish the NNs training and obtain $f_{\boldsymbol{\omega}}(\boldsymbol{x}) + \nu - l_{\boldsymbol{\xi}}(\boldsymbol{x})$ and $f_{\boldsymbol{\omega}}(\boldsymbol{x}) + \nu + u_{\boldsymbol{\theta}}(\boldsymbol{x})$. Then for a given series of confidence levels γ, we use root-finding technique to calculate the corresponding α and β in *Step 4* and obtain the associated $100\gamma\%$ PIs defined in Eq. (11). Figure 1 shows the 90%, 95% and 99% PIs. In calculation of the series of PIs for the multiple confidence levels, PI3NN only trains the three NNs once and the resulted PIs have no "crossing issue". Bottom panels of Figure 1 demonstrate the effectiveness of PI3NN's OOD identification capability and illustrate that it is our bias initialization scheme (Section 3.3) enables PI3NN to identify the OOD regions and reasonably quantify their uncertainty.

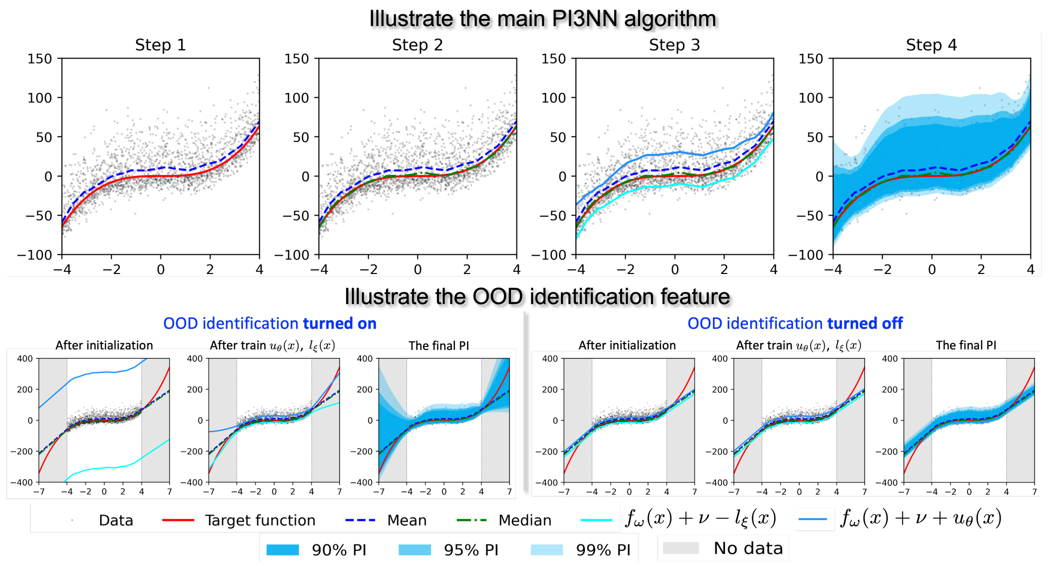

Figure 1: Top panels illustrate the four steps of our PI3NN algorithm. Bottom panels illustrate the effectiveness of the OOD identification feature. As shown in bottom left, when turning on the OOD identification feature by initializing the bias of the output layer of $u_{\boldsymbol{\theta}}$ and $l_{\boldsymbol{\xi}}$ to a large value, PI3NN can accurately identify the OOD regions $[-7, -4] \cup [4, 7]$ by giving them increasingly large PIs as their distance from the training data gets large. In bottom right, if we turn off the OOD identification by using the default initialization, PI3NN will not identify the OOD regions by giving them a narrow uncertainty bound.

4 Experimental evaluation

We evaluate our PI3NN method by comparing its performance to four top-performing baselines—QD [14], PIVEN [15], SQR [17], and DER [6] on three experiments. The first experiment focuses on evaluation of the accuracy and robustness of the methods in calculating PIs. The last two experiments focus on the assessment of the OOD identification capability. The codes for reproducing our experimental results of all the methods are provided in Appendix.

4.1 UCI REGRESSION BENCHMARKS

We first evaluate PI3NN to calculate 95% PIs on nine UCI datasets [26]. The quality of PI is measured by the Prediction Interval Coverage Probability (PICP) which represents the ratio of data samples that fall within their respective PIs. We are interested in calculating a well-calibrated PIs, i.e., the test PICP is close to the desired value of 0.95 and meanwhile has a narrow width. In this experiment, we focus on the metric of PICP, not the PI width (PIW) because it is meaningful to compare PIWs only when the methods have the same PICPs. A small PICP certainly produces small PIWs, but the small PICP may be much lower than the desired confidence level causing unreasonable PIs. In this experiment, for different model settings, some PI methods produce a wide range of PICPs while others result in a relatively narrow range; it is unfair to compare the PIWs between methods that produce different ranges of PICPs.

We use a single hidden layer ReLU NN for all the methods. For each method, we test multiple scenarios with different hyper-parameter configurations, different training-testing data splits. For each scenario, we use the average of 5 runs (with 5 random seeds) to generate the result. All the methods consider the variation of the hidden layer size. QD and PIVEN additionally consider the variation of the extra hyper-parameters introduced in their loss functions. See Appendix for details about the experimental setup.

		Boston	Concrete	Energy	Kin8nm	Naval	Power	Protein	Wine	Yacht
PI3NN	Mean	0.95	0.94	0.95	0.94	0.95	0.95	0.95	0.95	0.95
	Std	0.03	0.02	0.03	0.006	0.006	0.004	0.003	0.02	0.02
	Best	0.94	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.94
	Worst	0.88	0.89	0.87	0.93	0.94	0.96	0.94	0.91	0.90
QD	Mean	0.85	0.84	0.87	0.91	0.94	0.94	0.93	0.90	0.93
	Std	0.06	0.05	0.04	0.02	0.03	0.01	0.02	0.03	0.06
	Best	0.94	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.93
	Worst	0.69	0.66	0.78	0.86	0.59	0.92	0.90	0.83	0.71
PIVEN	Mean	0.83	0.83	0.82	0.87	0.94	0.94	0.91	0.82	0.88
	Std	0.07	0.05	0.05	0.02	0.01	0.01	0.02	0.04	0.08
	Best	0.94	0.94	0.93	0.91	0.95	0.95	0.94	0.91	0.93
	Worst	0.51	0.50	0.67	0.82	0.91	0.90	0.88	0.68	0.52
SQR	Mean	0.76	0.83	0.83	0.86	0.87	0.88	0.87	0.85	0.82
	Std	0.17	0.12	0.11	0.13	0.14	0.13	0.13	0.12	0.13
	Best	0.94	0.95	0.96	0.95	0.95	0.95	0.95	0.95	0.94
	Worst	0.39	0.52	0.58	0.56	0.54	0.56	0.51	0.42	0.48
DER	Mean	0.87	1.0	0.98	1.0	1.0	1.0	1.0	0.98	0.83
	Std	0.03	0.0	0.04	0.0	0.0	0.0	0.004	0.008	0.1
	Best	0.94	1.0	0.94	1.0	1.0	1.0	0.98	0.97	0.93
	Worst	0.80	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.61

Table 1: Evaluation of 95% PI on testing data. We show the mean, standard deviation (std), best, and worst PICP (across hyper-parameter configurations) for all methods. The best performer should produce PICP values closest to the desired 0.95. DER tends to overestimate the PI resulting in PICPs close to 1.0 for most datasets and produces the worst performance. For the four PI methods, our PI3NN shows the top performance by giving the best mean PICP (closest to 0.95) with the smallest std across hyper-parameter configurations. QD, PIVEN, and SQR require a careful hyper-parameter tuning to obtain the comparable best PICP as the PI3NN, but for each hyper-parameter set, their PICP std is large and the worst PICP can be much lower than the desired 0.95.

Table 1 summarizes the results. It shows the mean, standard deviation (std), best, and worst PICP (across hyper-parameter configurations) for all the methods on the testing datasets. DER, because of the Gaussian assumption, tends to overestimate the PI resulting in PICPs close to 1.0 for most datasets and produces the worst performance in terms of PICP. For the four PI methods, our PI3NN achieves the best mean PICP (closest to 0.95) with the smallest std. PI3NN is also the top performer in terms of the best and the worst PICP. This suggests that PI3NN can produce a well-calibrated PI and its performance is less affected by the hyper-parameter configuration. In contrast, QD, PIVEN, and SQR require a careful hyper-parameter tuning to obtain the comparable best PICP as the PI3NN, but for each hyper-parameter set, their PICPs vary a lot with a large std and the worst PICP can be much lower than the desired 0.95 resulting in an unreasonable PI. Thus, this experiment demonstrates the advantage of our method in producing a well-calibrated PI without time-consuming hyperparameter fine tuning, which makes it convenient for practical use.

4.2 OOD IDENTIFICATION IN A 10-DIMENSIONAL FUNCTION EXAMPLE

We use a 10-dimensional cubic function $f(\boldsymbol{x}) = \frac{1}{10}(x_1^3 + \cdots + x_{10}^3)$ to demonstrate the effectiveness of the proposed bias initialization strategy in Section 3.3 for OOD identification. The training data of $\boldsymbol{x}$ is generated by drawing 5,000 samples from the normal distribution $\mathcal{N}(0,1)$; the corresponding training data of outputs y is obtained from $y = f(\boldsymbol{x}) + \varepsilon$ with ε following a Gaussian distribution $\mathcal{N}(0,1)$. We define a test set with 1,000 OOD samples, where $\boldsymbol{x}$ are drawn from a shifted normal distribution $\mathcal{N}(2,1)$. We calculate the 90% PI for both the training (InD) and the testing (OOD) data using all the five methods. For PI3NN, we consider two situations with the OOD identification turned on and turned off. We set the constant c in Section 3.3 to 10 to turn on PI3NN's OOD identification feature, and use the standard initialization to turn the feature off. We use PI width (PIW) to evaluate the method's capability in OOD identification. A method having an OOD identification capability should produce larger PIWs of the OOD samples than those of the InD samples. Additionally, we use the confidence score defined in Eq. (12) to quantitatively evaluate the method's capability in OOD identification. InD samples should have a close-to-one confidence score and the OOD samples should have a remarkably smaller confidence score than that of the InD datasets.

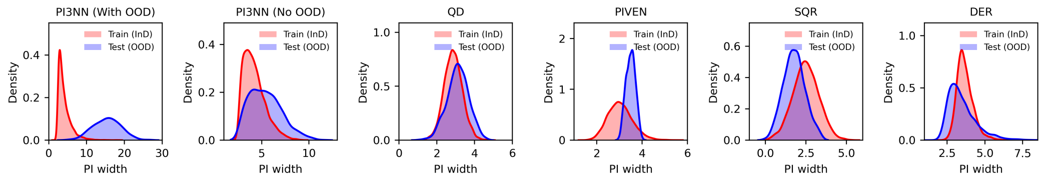

Figure 2: Probability density functions of the PI width for the training (InD) and testing (OOD) data for all the five methods. When we use a probability density function (PDF) to fit PIWs of the InD and OOD samples, respectively, we should be able to see two separated PDFs with the PDF of OOD samples shifting to the right having larger PIWs. If the two PDFs are overlapped to each other, then it indicates the method can not identify the OOD samples by reasonably quantifying their uncertainty.

	PI3NN (With OOD)	PI3NN (No OOD)	QD	PIVEN	SQR	DER
Train (InD)	0.91 ± 0.15	0.92 ± 0.12	0.94 ± 0.08	0.94 ± 0.08	0.91 ± 0.12	0.95 ± 0.08
Test (OOD)	0.28 ± 0.08	0.80 ± 0.18	0.90 ± 0.10	0.87 ± 0.05	0.98 ± 0.07	0.94 ± 0.11

Table 2: Mean $\pm$ Standard Deviation of the confidence score (Eq. (12) with $\gamma = 0.9$) for the 10D cubic example

Figure 2 shows the PDFs of the PIWs of the InD and OOD samples. The figures indicate that our PI3NN method, with OOD identification feature turned on, can effectively identify the OOD samples by assigning them larger and well-separated PIWs than those of the InD dataset. The other methods are not able to identify the OOD samples by mixing their PIWs into the InD data's PIWs, showing overlapped PDFs. These methods' performance is similar to the case when the PI3NN turns off its OOD identification capability. **Table 2** lists the mean and std of the confidence scores for the InD and OOD datasets. This table further shows that PI3NN with OOD identification feature turned on can effectively identify the OOD samples by giving them smaller confidence scores than those of the InD sampels, while other methods produce over-confident (wider) PIs for the OOD samples and cannot separate OOD samples from InD dataset.

4.3 OOD IDENTIFICATION IN A FLIGHT DELAY DATASET

We test the performance of our method in OOD detection using the Flight Delay dataset [27], which contains about 2 million US domestic flights in 2008 and their cases for delay. We use the PI3NN method to predict the regression map from 6 input parameters, i.e., Day of Month, Day of Week, AirTime, Flight Distance, TaxiIn (taxi in time), and TaxiOut (taxi out time), to the Arrival Delay. The US federal aviation administration (FAA) categories general commercial airports into several ranks, i.e., *Rank 1 (Large Hub)*: 1% or more of the total departures per airport, *Rank 2 (Medium Hub)*: 0.25% - 1.0% of the total departures per airport, *Rank 3 (Small Hub)*: 0.05% - 0.25% of the total departures per airport, and *Rank 4 (NonHub)*: <0.05% but more than 100 flights per year.

The training set consists of randomly select 18K data from Rank 4 (Nonhub) airports' data. We also randomly select another 20K data from each rank to form 4 testing sets. It is known that there is a data shift from Rank 1 airports to Rank 4 airports, because the bigger an airport, the longer the Taxi time, the averaged AirTime and the Flight Distance. Thus, an OOD-aware PI method should be able to identify those data shift. The setup for PI3NN method and the baselines are given in Appendix.

Table 3 shows the confidence scores (i.e., Eq. (12) with $\gamma = 0.9$) computed by our method and the baselines. Our method with the OOD identification feature turned on successfully detected the data shift by giving the Rank 1,2,3 testing data lower confidence scores. In contrast, The other methods as well as our method (with OOD identification turned off) fail to identify the data shift from the training set to the Rank 1,2,3 testing sets.

Data	Rank 1	Rank 2	Rank 3	Rank 4
PI3NN (With OOD)	0.24 ± 0.05	0.32 ± 0.09	0.72 ± 0.22	0.92 ± 0.24
PI3NN (No OOD)	0.72 ± 0.32	0.79 ± 0.32	0.88 ± 0.27	0.90 ± 0.26
QD	0.89 ± 0.24	0.90 ± 0.24	0.92 ± 0.16	0.83 ± 0.14
PIVEN	0.83 ± 0.23	0.87 ± 0.21	0.99 ± 0.04	0.99 ± 0.03
SQR	0.98 ± 0.06	0.96 ± 1.41	0.97 ± 0.09	0.94 ± 0.10
DER	0.98 ± 0.13	1.00 ± 0.01	1.00 ± 0.00	1.00 ± 0.01

Table 3: Mean $\pm$ Standard Deviation of the confidence score (Eq. (12) with $\gamma = 0.9$) for the flight delay data.

5 Conclusion and Discussion

This work proposes a different route from the state-of-the-art PI methods [4; 15; 16; 17] on how to generate a series of PIs with the non-crossing property and with the OOD identification feature. Instead of designing sophisticated loss functions, we only use the standard MSE loss in the PI3NN method. Both the architecture and the training process for the three neural networks used in PI3NN are standard, such that choosing the commonly seen hyperparameters, e.g., the learning rate, the number of hidden neurons, the number of layers, is almost a textbook problem that can be easily conducted without fine tuning by non-expert users. Additionally, the OOD identification capability of PI3NN is also simply designed by a small modification of the bias of the output layer, which is demonstrated to be very effective in the experiments.

The limitations of the PI3NN method include: (1) For a target function with multiple outputs, each output needs to have its own PI and OOD confidence score. The PI and the confidence score cannot oversee all the outputs. For example, this could make it challenging to apply PI3NN to NN models having image as outputs, (e.g., autoencoders). (2) The effectiveness of the OOD detection approach depends on that there are sufficiently many piecewise linear regions (of ReLU networks) in the OOD area. So far, this is achieved by the standard random initialization (ensure uniform distributed piecewise linear regions at the beginning of training) and L_1/L_2 regularization (ensure the linear regions not collapse together around the training set). However, there is no guarantee of uniformly distributed piecewise linear regions after training. Improvement of this requires significant theoretical work on how to manipulate the piecewise linear function defined by the ReLU network. This work is for purely research purpose and will have no negative social impact. (3) When the noise ε has heavy tails, our method will struggle when the distance between tail quantiles becomes very large. In this case, the number of training data needed to accurately capture those tail quntiles may be too large to afford. It is a common issue for all distribution-free PI methods, and reasonable distributional assumption may be needed to alleviate this issue.

REFERENCES

[1] D. J. C. MacKay, "A practical bayesian framework for backpropagation networks," *Neural Comput.*, vol. 4, p. 448–472, May 1992.

[2] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *Journal of Machine Learning Research*, vol. 14, no. 4, pp. 1303–1347, 2013.

[3] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1050–1059, PMLR, 20–22 Jun 2016.

[4] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 6405–6416, Curran Associates Inc., 2017.

[5] T. Pearce, F. Leibfried, and A. Brintrup, "Uncertainty in neural networks: Approximately bayesian ensembling," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics* (S. Chiappa and R. Calandra, eds.), vol. 108 of *Proceedings of Machine Learning Research*, pp. 234–244, PMLR, 26–28 Aug 2020.

[6] A. Amini, W. Schwarting, A. Soleimany, and D. Rus, "Deep evidential regression," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 14927–14937, Curran Associates, Inc., 2020.

[7] R. D. D. VlEAUX, J. Schumi, J. Schweinsberg, and L. H. Ungar, "Prediction intervals for neural networks via nonlinear regression," *Technometrics*, vol. 40, no. 4, pp. 273–282, 1998.

[8] J. T. G. Hwang and A. A. Ding, "Prediction intervals for artificial neural networks," *Journal of the American Statistical Association*, vol. 92, no. 438, pp. 748–757, 1997.

[9] J. Carney, P. Cunningham, and U. Bhagwan, "Confidence and prediction intervals for neural network ensembles," in *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, vol. 2, pp. 1215–1218 vol.2, 1999.

[10] T. Heskes, "Practical confidence and prediction intervals," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, (Cambridge, MA, USA), p. 176–182, MIT Press, 1996.

[11] R. Koenker and G. Bassett, "Regression quantiles," *Econometrica*, vol. 46, no. 1, pp. 33–50, 1978.

[12] R. Koenker and K. F. Hallock, "Quantile regression," *Journal of Economic Perspectives*, vol. 15, pp. 143–156, December 2001.

[13] A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya, "Lower upper bound estimation method for construction of neural network-based prediction intervals," *IEEE Transactions on Neural Networks*, vol. 22, no. 3, pp. 337–346, 2011.

[14] T. Pearce, A. Brintrup, M. Zaki, and A. Neely, "High-quality prediction intervals for deep learning: A distribution-free, ensembled approach," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4075–4084, PMLR, 10–15 Jul 2018.

[15] E. Simhayev, G. Katz, and L. Rokach, "Piven: A deep neural network for prediction intervals with specific value prediction," 2021.

[16] T. S. Salem, H. Langseth, and H. Ramampiaro, "Prediction intervals: Split normal mixture from quality-driven deep ensembles," in *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)* (J. Peters and D. Sontag, eds.), vol. 124 of *Proceedings of Machine Learning Research*, pp. 1179–1187, PMLR, 03–06 Aug 2020.

[17] N. Tagasovska and D. Lopez-Paz, "Single-model uncertainties for deep learning," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[18] F. Zhou, J. Wang, and X. Feng, "Non-crossing quantile regression for distributional reinforcement learning," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 15909–15919, Curran Associates, Inc., 2020.

[19] L. V. Jospin, W. L. Buntine, F. Boussaïd, H. Laga, and M. Bennamoun, "Hands-on bayesian neural networks - a tutorial for deep learning users," *CoRR*, vol. abs/2007.06823, 2020.

[20] H. Wang and D.-Y. Yeung, "A survey on bayesian deep learning," 2021.

[21] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, V. Makarenkov, and S. Nahavandi, "A review of uncertainty quantification in deep learning: Techniques, applications and challenges," 2021.

[22] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[23] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter, "Bayesian optimization with robust bayesian neural networks," in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.

[24] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical Mathematics (Texts in Applied Mathematics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[25] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding Deep Neural Networks with Rectified Linear Units," *arXiv e-prints*, p. arXiv:1611.01491, Nov. 2016.

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

[27] 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, (Arlington, Virginia, USA), p. 282–290, AUAI Press, 2013.

APPENDIX

A THE PROOF OF LEMMA 1

Proof. Substitute $y = f(\boldsymbol{x}) + \varepsilon$ into Eq. (5), we can rewrite Eq. (5) as $\hat{U}(\boldsymbol{x}|\alpha) = f(\boldsymbol{x}) + \mathbb{M}[\varepsilon] + \alpha\mathbb{E}[(\varepsilon - \mathbb{M}[\varepsilon])\mathbf{1}_{\varepsilon-\mathbb{M}[\varepsilon]>0}]$, where $\mathbb{E}[(\varepsilon - \mathbb{M}[\varepsilon])\mathbf{1}_{\varepsilon-\mathbb{M}[\varepsilon]>0}] = \mathbb{E}[(y - \mathbb{M}[y])\mathbf{1}_{y-\mathbb{M}[y]>0}]$ is strictly positive and independent of $\boldsymbol{x}$. Then, we have that $\hat{U}(\boldsymbol{x}|\alpha)$ satisfies the properties (P1) and (P2).

It is easy to see that, for a fixed $\boldsymbol{x}$, there exists a unique value $\tilde{\alpha}(\gamma, \boldsymbol{x})$, which depends on both γ and $\boldsymbol{x}$, such that $\mathbb{E}[\mathbf{1}_{y>\hat{U}(\boldsymbol{x}|\tilde{\alpha}(\gamma,\boldsymbol{x}))}] - (1-\gamma)/2 = 0$. On the other hand, since $\hat{U}(\boldsymbol{x}|\alpha)$ in Eq. (5) satisfies the property (P2), we have that $\hat{U}(\boldsymbol{x}|\tilde{\alpha}(\gamma, \boldsymbol{x})) - f(\boldsymbol{x}) = \mathbb{M}[\varepsilon] + \tilde{\alpha}(\gamma, \boldsymbol{x})\mathbb{E}[(\varepsilon - \mathbb{M}[\varepsilon])\mathbf{1}_{\varepsilon-\mathbb{M}[\varepsilon]>0}]$ is independent of $\boldsymbol{x}$. Hence, $\tilde{\alpha}(\gamma, \boldsymbol{x})$ is actually independent of $\boldsymbol{x}$.

Combining the above results, we have that there exists a unique $\alpha(\gamma)$ independent of $\boldsymbol{x}$, such that

$$\mathbb{E}\big[\mathbf{1}_{y>\hat{U}(\boldsymbol{x}|\alpha(\gamma))}\big] - (1-\gamma)/2 = 0. \tag{13}$$

Comparing Eq. (13) and the definition of the upper bound in Eq. (3), we can see that $\hat{U}(\boldsymbol{x}|\alpha(\gamma))$ is also a upper bound of the $100\gamma\%$ PI. Due to the uniqueness of the upper bound, i.e., the property (P3), we have $\hat{U}(\boldsymbol{x}|\alpha(\gamma)) = U_{\gamma}^{\text{true}}(\boldsymbol{x})$, which concludes the proof. □

B The proof of Lemma 2

Proof. We only need to prove $Q_{\text{upper}}(\alpha(\gamma)) = 0$, and the same derivation can be applied to $Q_{\text{lower}}(\beta(\gamma))$. Also, for notational simplicity, we assume N is an even number, so that $N/2$ is an integer. We first define

$$g(\alpha) = \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_{\text{upper}}} \mathbf{1}_{y_i > U(\boldsymbol{x}_i|\alpha)}, \quad \text{for } \alpha \in [0, \infty), \tag{14}$$

based on the definition of Q_{upper} in Eq. (10). Since N is a finite integer, $g(\alpha)$ can be written as a step function, i.e.,

$$g(\alpha) = \sum_{k=0}^{N/2} k\, \mathbf{1}_{A_k}(\alpha), \tag{15}$$

where $\mathbf{1}_{A_k}(\alpha)$ is defined by

$$\mathbf{1}_{A_k}(\alpha) = \begin{cases} 1, & \text{if } \alpha \in A_k, \\ 0, & \text{if } \alpha \notin A_k, \end{cases} \tag{16}$$

with A_k defined by

$$A_k = \left\{ \alpha \in [0, \infty) \,\middle|\, \#\{y_i > U(\boldsymbol{x}_i|\alpha) \text{ for } (\boldsymbol{x}_i, y_i) \in \mathcal{D}_{\text{upper}}\} = k \right\}.$$

In other words, A_k is a sub-interval of $[0, \infty)$, such that the number of samples in $\mathcal{D}_{\text{upper}}$ satisfying $y_i > U(\boldsymbol{x}_i|\alpha)$ is equal to k for $\alpha \in A_k$.

Since $(\boldsymbol{x}_i, y_i) \neq (\boldsymbol{x}_j, y_j)$ for $i \neq j$, the Lebesgue measure of A_k, denoted by $\lambda(A_k)$ is strictly positive, i.e., $\lambda(A_k) > 0$. From classic measure theory, we know that for any set in $\mathbb{R}$ with strictly positive Lebesgue measure, there are *infinite* real numbers in the set. As such, there are infinite real numbers in each A_k for $k = 0, \ldots, N/2$.

Hence, for a given γ, any real number $\alpha(\gamma) \in A_{\lceil N(1-\gamma)/2 \rceil}$ will satisfy that $g(\alpha(\gamma)) = \lceil N(1-\gamma)/2 \rceil$, i.e., $Q_{\text{upper}}(\alpha(\gamma)) = 0$, which concludes the proof. □

C The proof of Theorem 1

Proof. We only need to prove $U(\boldsymbol{x}|\alpha(\gamma)) > U(\boldsymbol{x}|\alpha(\gamma'))$, and the same derivation can be applied to $L(\boldsymbol{x}|\beta)$. Since $u_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0$, we have that $U(\boldsymbol{x}|\alpha)$ is monotonically increasing with α for any $\boldsymbol{x}$. Thus, if $\alpha(\gamma) > \alpha(\gamma')$ then $U(\boldsymbol{x}|\alpha(\gamma)) > U(\boldsymbol{x}|\alpha(\gamma'))$. So we only need to prove $\alpha(\gamma) > \alpha(\gamma')$. We use the proof by contradiction approach to prove $\alpha(\gamma) > \alpha(\gamma')$, i.e., we will derive a contradiction from the assumption that $\alpha(\gamma) \leq \alpha(\gamma')$.

It is easy to see that the function

$$g(\alpha) = \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_{\text{upper}}} \mathbf{1}_{y_i > U(\boldsymbol{x}_i|\alpha)}, \quad \text{for } \alpha \in [0, \infty), \tag{17}$$

is a monotonically decreasing function of α. Thus, if $\alpha(\gamma) \leq \alpha(\gamma')$, we have $g(\alpha(\gamma)) \geq g(\alpha(\gamma'))$. On the other hand, we know from Lemma 2 that $\alpha(\gamma)$ and $\alpha(\gamma')$ satisfy

$$g(\alpha(\gamma)) = \lceil N(1-\gamma)/2 \rceil \;\text{ and }\; g(\alpha(\gamma')) = \lceil N(1-\gamma')/2 \rceil, \tag{18}$$

which leads to $\lceil N(1-\gamma)/2 \rceil \geq \lceil N(1-\gamma')/2 \rceil$. This is a contradiction with the condition of the theorem that $\lceil N(1-\gamma)/2 \rceil < \lceil N(1-\gamma')/2 \rceil$. Therefore, the assumption $\alpha(\gamma) \leq \alpha(\gamma')$ is incorrect, such that we have $\alpha(\gamma) > \alpha(\gamma') \Rightarrow U(\boldsymbol{x}|\alpha(\gamma)) > U(\boldsymbol{x}|\alpha(\gamma'))$, which concludes the proof. □

D Experiment setups and parameters

D.1 Setup for the UCI examples

We evaluated the performance of the five method on 9 widely used UCI data sets, including Boston housing (boston), Concrete compressive strength (concrete), Energy efficiency (energy), KINematics

8 inputs non-linear medium unpredictability/noise (kin8nm), Combined Cycle Power Plant (power), Physicochemical Properties of Protein Tertiary Structure (Protein), Wine quality (wine), and yacht Hydrodynamics (yacht). Since data splitting will affect the results for those problems, we pre-split the each data set with fixed splitting random seed, and generated 3 train/test (90%/10%) data pairs used for all methods, so as to ensure a fair comparison. Even different methods are built on top of different ML/DL platform, software versions, we intend to standardize the data pre-processing steps (e.g. data normalization) to ensure same data is imported to all models.

For each method, we test multiple scenarios with different hyper-parameter configurations, different training-testing data splits and random seeds. All the methods consider the variation of the hidden layer size. QD and PIVEN additionally consider the variation of the extra hyper-parameters introduced in their loss functions.

We use a single hidden layer ReLU NN for all the methods. For each method, we test multiple scenarios with different hyper-parameter configurations, different training-testing data splits. Specifically, we generate 3 pre-split data pairs for all experiments. The other universally applied hyperparameter for all methods is the number of hidden neurons: [50, 100, 200]. QD and PIVEN additionally consider the variation of the extra hyper-parameters introduced in their loss functions. For each scenario, we use the average of 5 runs (with 5 random seeds) to generate the result.

For PI3NN method, we use the hyper-parameters for all experiments: learning rate (0.01), Adam optimizer with MSE loss for all three networks.

For QD method, we include additional two hyper-parameter combinations: soften parameter: [100., 160., 200.] and lambda_in parameter: [5., 15., 25.] which are essential parameters for QD method. In addition, we keep the default parameters in the original QD code for boston and concrete data sets, and our tuned parameters for rest of 7 data. Specifically, the maximum epochs for all data is 300 except for concrete (800); similarly, the learning rate for concrete is 0.03 and 0.02 for other data sets; the decay rate and sigma_in are set to 0.9 and 0.1 respectively for 7 data sets except for concrete (0.98 and 0.2). The outer layer bias are set to [3., -3]. Adam optimizer is used in the experiments, and we use 100 batch size from the original code.

For DER method, the maximum epochs is set to 40. We follow the author tuned learning rate and batch size provided in the original code. The learning rate and batch size are (1e-3, 8) for boston, (5e-3, 1) for concrete, (2e-3, 1) for energy, (1e-3, 1) for kin8nm, (1e-3, 2) for power, (5e-4, 1) for naval, (1e-4, 32) for wine, (1e-3, 64) for protein, and (5e-4, 1) for yacht.

For PIVEN method, similar to QD, we include the lambda_in ([5.0, 15.0, 20.0]) and sigma_in ([0.05, 0.2, 0.4]) for the hyper-parameter combinations. The rest of the parameters are provided by the original authors in the code, including: batch size (100), outer layer biases [3., -3.], soften parameter (160.0). Meanwhile, we use the author tuned maximum epochs, learning rate, decay rate for boston (300, 0.02, 0.9), concrete (800, 0.03, 0.98), energy (1200, 0.016, 0.97), kin8nm (800, 0.0134725, 0.993922), naval (1000, 0.006, 0.998), power (500, 0.014075, 0.987099), protein (600, 0.002, 0.999), wine (1000, 0.01, 0.98) and yacht (2000, 0.005, 0.98).

For SQR method, we added three hyper-parameter combinations based on the original paper and code, which are the learning rate [1e-2, 1e-3, 1e-4], dropout rate [0.1, 0.25, 0.5, 0.75] and weight decay [0, 1e-3, 1e-2, 1e-1, 1]. maximum epochs is 5,000 for all data sets by default. Additional 20% validation data is split from the training data. These parameter combinations with the split seed, random seed, number of neurons yield much more resutls than other methods, thus, we reduced the results data size by fixing the learning rate (1e-2), dropout rate (0.1) and weight decay (0.1) for the final evaluations.

D.2 Setup for the 10D example

The 10-dimensional cubic function is used for generating the synthetic data for demonstrating OOD identification capability. As mentioned in Section 4.2, we generate a training data with 10 input features and size of 5,000, and 1,000 OOD testing data.

For PI3NN method, we use three networks with single layer (100 hidden nodes) with ReLU activation, the MSE loss is used with SGD optimizer (learning rate = 0.01). Maximum epochs is 3,000. We use comparable setups for the baseline methods. Specifically, we take the hyperparameter values (suggested by the authors of those methods) and perform hyperparameter tuning around those suggested values, and choose the best performance for each baseline method. The hyperparameters we tune include the hidden layer width, the learning rate and the extra hyperparameters introduced into the baseline method. For hidden layers, we use 100,200,300 hidden neurons as candidates. For learning rates, we use 0.0001, 0.001, 0.01 as candidates. For the exclusive hyperparameters, we choose five candidate values (including the suggested ones). For each hyperparameter configuration, we run an ensemble of 5 runs to get the results. After tuning, we have the following setup for the baseline methods. For QD method, we use single hidden layer network (200 hidden nodes) with Adam optimizer (lr=0.02). Activation function is ReLU, and batch size is 100. The soften parameter, labmda_in, and sigma_in are set to 160.0, 15.0 and 0.4, respectively. For PIVEN method, sigma_in (0.2), learning rate (0.01). It shares the same network structure, softer parameter and lambda_in with QD method. For DER method, we use single hidden layer with 200 nodes network, 1e-4 learning rate, 256 batch size. For SQR method, we use single layer network with 100 neurons network. We fix the learning rate, dropout rate, weight decay rate to 1e-3, 0.1 and 0, respectively. We also use the default setting by taking 20% of the training data as validation set in this method.

D.3 Setup for the flight delay example

The flight delay data (`https://www.kaggle.com/vikalpdongre/us-flights-data-2008`) contains the flight information in the U.S. in the year 2008. We separate the data by airports into 4 ranks based on the percentage of departure flights, which including: Large Hub (Rank 1), receives 1% or more of the annual departures; Medium Hub (Rank 2), which receives 0.25%-1% of the annual departures; Small Hub (Rank 3) and Nonhub (rank4) receives 0.05%-0.25% and <0.05% of the departures, respectively. We select 5% of the Rank 4 data as the training data, another 5% of the Rank 4 is picked as 'test 4' data, and we consider this as in-distribution data. Rest of three testing data ('test 3', 'test 2', and 'test 1') are selected from the Rank 3, 2 and 1, respectively with the 5% total amount of data. We pick the 6 input features ('DayofMonth', 'DayOfWeek', 'AirTime', 'Distance', 'DepDelay', 'TaxiOut'), and select 'ArrDelay' as the output feature.

For PI3NN method, in the experiment, we use the same network architecture (i.e., one hidden layer ReLU network containing 100 hidden neurons) and as the one used for the UCI datasets. The Adam optimizer is used with learning rate being 0.01 and the maximum epochs being 50,000. Conventional L1 and L2 regularization are implemented with both penalties set to 0.02. The scalar parameter (i.e. the one controlling the initial bias of the output layer) is set to 10 (the same as the test on the 10D function).

We also perform hyperparameter tuning for each of the baseline methods. The universal hyperparameter, i.e., the number of hidden neurons come from the pool [50,100,200]. For the exclusive hyperparameters, we choose five candidate values (including the suggested ones). For each hyperparameter configuration, we run an ensemble of 5 runs to get the results. After tuning, we have the following setup for the baseline methods.

For QD method, we use single layer network with 100 neurons, Adam optimizer with 0.02 learning rate, 50 epochs, 160.0 soften parameter, 0.1 sigma_in, 15.0 lambda_in, 100 batch size, 0.9 decay rate.

For DER method, we use single hidden layer with 100 neurons network structure, 512 batch size, 1e-4 learning rate and 100 maximum epochs for the experiments.

For PIVEN method, lambda_in and sigma_in are set to 15.0 and 0.2. Same single layer 100 nodes network structure is applied. Soften parameter is 160.0. Batch size and maximum epochs are 100 and 500, respectively. We use 0.01 learning rate and 0.99 decay rate

For SQR method, we use same network structure [100 neurons] and maximum epochs (500) with PIVEN method. We fixed the learning rate (1e-3), dropout rate (0.1), and weight decay rate (0) in the flight delay experiments.

E Code and data availability

We include the code and data for reproducing the results generated in this study. The data used for generating our table and plots are pre-generated for fast evaluations. All the code with running

instructions are made available for download through the following anonymous URL:
https://figshare.com/s/b59c11bfa35b89cd061f

We include detailed instructions on running the code in the uploaded supplemental materials.

The original code for the methods except PI3NN can be found through the following URL:
https://github.com/TeaPearce/Deep_Learning_Prediction_Intervals
https://github.com/aamini/evidential-deep-learning
https://github.com/elisim/piven
https://github.com/facebookresearch/SingleModelUncertainty

Note that different methods may require different environment to run the code: PI3NN is based on TensorFlow 2.0 (for UCI and Flight delay cases), and PyTorch (Cubic1D, Cubic10D), QD is based on TensorFlow 1.0 and can runs under compatibility mode with TF2.0, DER is based on TF2.0, PIVEN is based on TF1.0, and SQR method is based on PyTorch. More details are included in the supplemental materials.