NON PARAMETRIC ALEATORIC UNCERTAINTY QUANTIFICATION WITH NEURAL NETWORKS

Kshitij Kapoor
Department of Computer Science,
Ashoka University (Haryana, India).
kshitij.kapoor1@ashoka.edu.in

Debayan Gupta
Department of Computer Science,
Ashoka University (Haryana, India).
debayan.gupta@ashoka.edu.in

ABSTRACT

Classic methods for aleatoric uncertainty quantification in regression settings make assumptions about the distribution of noise in the dependent variable. Incorrect assumptions can lead to poor model performance and unreliable uncertainty estimates. In this paper, we introduce a simple method for non-parametric aleatoric uncertainty quantification. In particular, we train a neural network model for binary classification. The inputs to our binary classifier are the independent variables and a sample from the marginal distribution of the dependent variable. This binary classifier is trained to predict whether the sample from the marginal distribution of the dependent variable is greater than the dependent variable corresponding to independent variables in the input. Our method can be used for not only quantifying aleatoric uncertainty but also estimating the conditional distribution of the dependent variable.

1 INTRODUCTION AND RELATED WORK

With the increasing penetration of neural network models in critical decision making processes, it is necessary to quantify the uncertainty in model output. While there are components of both aleatoric and epistemic uncertainty in predictions generated by neural networks (Hüllermeier & Waegeman, 2019), we focus our analysis on aleatoric uncertainty. Generally, while modelling aleatoric uncertainty with a neural network in regression settings, we assume that the dependent variable observations have additive noise ϵ. Further, we assume that this additive noise follows a heteroscedastic gaussian distribution that has zero mean i.e. $\epsilon_i \sim N(\mu = 0, \sigma = h(\boldsymbol{x}_i))$ where $\boldsymbol{x}_i \in R^d$ and $h\colon R^d \to R^+$. We can then train a neural network to predict the parameters of the conditional distribution of the dependent variable (Nix & Weigend, 1994). Negative log likelihood is used as the loss function to optimize the weights of this neural network. This trained model can predict the conditional distribution of the dependent variable in one pass. Though easy to train and cheap to infer, models trained with the assumption that the noise is additive and gaussian might lead to poor performance (Cervera et al., 2021). This is because the structure of the aleatoric noise might be misspecified. Further the conditional distribution of the dependent variable might be multimodal. In this scenario, the model would perform very poorly as the predicted mean would be somewhere between these modes and the predicted variance would be very high.

To address issues caused by distribution misspecification and multimodal conditional distribution of the dependent variable, we propose a simple non parametric method for aleatoric uncertainty quantification. Our method revolves around a neural network based binary classification model B with a sigmoid activation applied to the output layer. Let us assume we have a pair comprising an independent variable vector $\boldsymbol{x}_i \in R^d$ and the associated dependent variable $y_i \in R$ and another $y_j \in R$ sampled from the marginal distribution of dependent variable Y. The binary classification model is trained to predict whether $y_i < y_j$. The inputs to this model are the independent variable vector $\boldsymbol{x}_i$ concatenated with the y_j i.e. $B\colon R^{d+1} \to [0, 1]$. We use the binary cross-entropy loss function to train the binary classifier. Hence, once trained binary classifier probability $B(\mathbf{x} = \boldsymbol{x}_i)$ estimates the probability $P(y_i < y_j | \mathbf{x} = \boldsymbol{x}_i)$. By varying the value y_j in a suitable subset of the real number line, we can estimate the cumulative distribution function $F(y|\boldsymbol{x})$. Further, by using automatic differentiation to calculate the gradient $\nabla_{(B(\boldsymbol{x}_i \oplus y))} y$ [1] we can implicitly estimate the conditional probability density function $f(y|\boldsymbol{x})$.

[1] $\oplus$ is for concatenation

Table 1: Mean log likelihood (higher is better) of test data sampled from the synthetic datasets.

Dataset	**Log Likelihood**		
	Ground Truth	Gaussian NLL Model	Nonparametric Model
Linear	-2.176	-3.383	$\mathbf{-2.421}$
Quadratic	-3.646	-4.902	$\mathbf{-4.562}$

2 Methodology

To train this binary classification model we need to generate mini batches that can be used for optimizing the weights of the neural network. To generate a mini batch of size k, we sample k pairs of independent variable vectors and dependent variables $\{(\boldsymbol{x}_1, y_1), \dots (\boldsymbol{x}_k, y_k)\}$. We also draw k samples $\{y_1^*, \dots, y_k^*\}$ from the marginal distribution of the dependent variable Y by simply sampling with replacement from the list of all Y. We then sample $\{l_1, \dots, l_k\}$ from a distribution suitable for multiplicative noise. The input samples are generated by concatenating $\boldsymbol{x}_i$ with $(y_i^* * l_i)$ i.e. $\boldsymbol{x}_i \oplus (y_i^* * l_i)$. The associated binary labels are the result of the comparison $y_i < (y_i^* * l_i)$.

3 Experiments

To evaluate the performance of the proposed model, we experimented with two synthetic datasets. These were constructed such that the conditional distributions of dependent variables could be multimodal. Further, the noise added to the dependent variable was heteroscedastic. Scatterplots of samples from these datasets can be found in Figure 1. We trained the proposed non parametric model using the methodology described in previous sections. For mini batch construction, we chose the log normal distrtibution for multiplicative noise l i.e. $l_i \sim LogNormal(0, 0.1)$. We also trained a neural network to predict parameters of a gaussian distribution that would estimate the conditional distribution of the dependent variable (to compare against our method). This model was trained with the gaussian negative log likelihood loss function. Both models were optimized using the Adam algorithm and had the same numbers of neurons and hidden layers. We then generated another sample from the ground truth distribution and calculated its log likelihood using both methods. Table 1 gives an overview of the negative log likelihood comparisons. For both datasets, the non parametric model outperformed the model trained with gausssian negative log likelihood loss function.

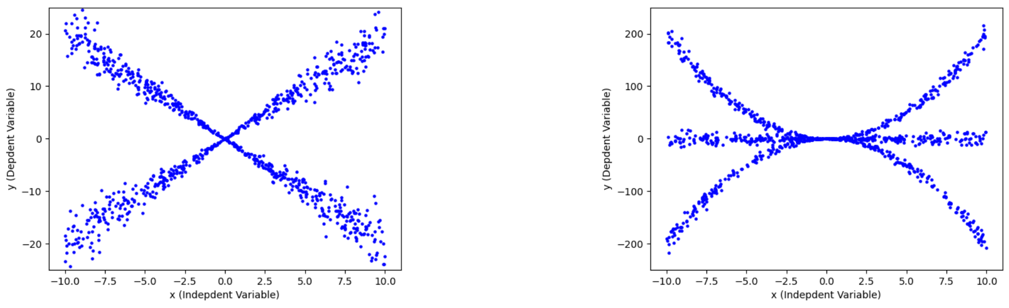

Figure 1: Scatter plot of the synthetic datasets used in experimentation.

4 Conclusion

In this paper we proposed a new non parametric method for aleatoric uncertainty estimation in regression setting. We compared it's performance to that of models trained by minimizing gaussian negative log likelihood loss on synthetic datasets. These results motivate an analysis of the performance of the proposed model on real-world datasets.

ACKNOWLEDGEMENTS

We thank the Mphasis F1 Foundation for supporting this work.

URM STATEMENT

The authors acknowledge that at least one key author of this work meets the URM criteria of ICLR 2024 Tiny Papers Track.

REFERENCES

Maria R. Cervera, Rafael Dätwyler, Francesco D'Angelo, Hamza Keurti, Benjamin F. Grewe, and Christian Henning. Uncertainty estimation under model misspecification in neural network regression. *CoRR*, abs/2111.11763, 2021. URL `https://arxiv.org/abs/2111.11763`.

Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. *CoRR*, abs/1910.09457, 2019. URL `http://arxiv.org/abs/1910.09457`.

D.A. Nix and A.S. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pp. 55–60 vol.1, 1994. doi: 10.1109/ICNN.1994.374138.

A IMPLICIT ESTIMATION OF CONDITIONAL PROBABILITY DENSITY FUNCTION

Once trained, the neural network based binary classification model B can be used for calculating the entire probability density function of the dependent variable y conditioned on any independent variable vector $\boldsymbol{x}$. Because, the binary classification model B estimates $F(y|\boldsymbol{x})$, the gradient $\nabla_{(B(\boldsymbol{x}\oplus y))}y$ estimates the conditional probability density function $f(y|\boldsymbol{x})$. A viable probability density function f should be positive in it's respective domain. Given that there is no guarantee that the output of the binary classification model B is monotonically increasing with respect to y, it is possible that the gradient $\nabla_{(B(\boldsymbol{x}\oplus y))}y$ is negative. Hence, adjustments need to be made to account for this issue. To this end, we define another function $g(y, \boldsymbol{x}) = (\nabla_{(B(\boldsymbol{x}\oplus y))}y)^{+}$. The function $g(y, \boldsymbol{x})$ is the positive part of the the gradient $\nabla_{(B(\boldsymbol{x}\oplus y))}y$, i.e. if the gradient is negative, the function $g(y, \boldsymbol{x}) = 0$. Hence, the minimum value of $g(y, \boldsymbol{x})$ is 0.

We will now walk through an example of how our proposed method works. Assume, that we have trained the neural network based binary classification model B using the methodology described in previous sections. Further assume that the conditional variable vector $\boldsymbol{x}$ is fixed and that the range of the dependent variable y is the subset (l, u) of the real number line. We can then partition this subset (l, u) into n equal size partitions each of width w where $w = (u-l)/n$. These equal sized partitions will be $(\{c_1, c_2\}, \{c_2, c_3\}, \ldots, \{c_n, c_{n+1}\})$ where $c_1 = l$, $c_{n+1} = u$ and $c_2 - c_1 = c_3 - c_2 = \ldots = c_{n+1} - c_n = w$. The midpoint of these partitions will be $\{c_1 + (w/2), c_2 + (w/2), \ldots, c_n + (w/2)\}$. We then calculate the function $g(y, \boldsymbol{x})$ at the midpoints of all the partitions to get the array $a = \{g(c_1 + (w/2), \boldsymbol{x}), g(c_2 + (w/2), \boldsymbol{x}), \ldots, g(c_{n+1} + (w/2), \boldsymbol{x})\}$. We then apply a smoothing filter convolution to the array a to get a^*. The array a^* has estimates of the conditional probability distribution function$f(\mathrm{y}|\boldsymbol{x})$ for midpoints of all the partitions. With partitions of reasonably small width w, the value a_i^* would estimate the conditional probability distribution function $f(y|\boldsymbol{x})$ for any $y \in (c_i, c_{i+1})$. Likelihood can also be calculated similarly. To increase the accuracy of these estimates at the cost of increase in computational complexity, one can simply increase the number of partitions n (which would decrease their width).

B MULTIPLICATIVE NOISE AND LOG-NORMAL DISTRIBUTION

While generating mini-batches for training the binary classifier B, we found that it was necessary to multiply lognormally distributed noise to the samples from the marginal distribution of the dependent variable y^*. This is because when we initially experimented without any noise added to these samples, the conditional CDF and PDF estimates at the extremes of the range of the dependent variable were not well behaved. For example, when experimenting with the linear synthetic dataset, the PDF estimates were not well behaved when the absolute value of the independent variable x was greater than 9 i.e. $|x| > 9$. For example, for $x = 9.5$, the estimated conditional cumulative distribution function were $\hat{F}(y = -30|\mathrm{x} = 9.5) > 0$ and $\hat{F}(y = -30|\mathrm{x} = 9.5) < 1$. Our intuition was that this is because there are insufficient samples at the extremes (of the domain of the dependent variable) from which the neural network model can learn. Hence, to ensure that the estimated conditional cumulative distribution function are well behaved, we had to augment the data by ”stretching” the range of y^* while generating batches for training the network. To make this ”stretching” independent of the scale of dependent variable, we choose multiplicative noise over additive noise.

C CODE FOR REPRODUCING EXPERIMENTS

We have published the code required for generating the synthetic datasets and training the models (`https://github.com/k00lk0der1/NonParametricAleatoricUncertaintyEstimation/`).