Supplementary Material of "Measurement error models: from nonparametric methods to deep neural networks"

Zhirui Hu, Zheng Tracy Ke and Jun S Liu

This document contains Appendix A to F. In Appendix A, we provide the full algorithm of training NNME. In Appendix B and C, we compare different neural network structures for measurement error models and different prior models for X in NNME respectively. In Appendix D, we show the sensitivity of NNME to the depth of neural networks. In Appendix E, we introduce kriging methods for Gaussian process regression. Finally, in Appendix F, we compare the performances of different methods in an example of two-dimensional functions generated from a neural network.

APPENDIX A: THE ALGORITHM OF TRAINING NNME

We use a gradient-based algorithm, called *Adam* (Kingma and Ba, 2014) to solve (14). Given initial values $(\theta^{(0)}, \gamma^{(0)}, \phi^{(0)})$ and $m^{(0)} = v^{(0)} = 0$, let $g^{(t)} = \nabla Q(\theta^{(t)}, \gamma^{(t)}, \phi^{(t)})$, the update at iteration t is

$$\begin{aligned} m^{(t+1)} &= \alpha_1 m^{(t)} + (1-\alpha_1) g^{(t)} \\ v^{(t+1)} &= \alpha_2 v^{(t)} + (1-\alpha_2)(g^{(t)} \odot g^{(t)}) \\ (\theta^{(t+1)}, \gamma^{(t+1)}, \phi^{(t+1)}) &= (\theta^{(t)}, \gamma^{(t)}, \phi^{(t)}) - \alpha_t \cdot m^{(t+1)} / (\sqrt{v^{(t+1)}} + \epsilon^{(t+1)}) \end{aligned} \tag{A.1}$$

where $\alpha_t > 0$ is the step size at iteration t, often called the learning rate. We use the adaptive learning rate suggested by Kingma and Ba (2014), which is $\alpha_t = \alpha_0 \sqrt{1-\alpha_2^{t+1}}/(1-\alpha_1^{t+1})$, for some default parameters $(\alpha_0, \alpha_1, \alpha_2)$. $\epsilon^{(t+1)} = \epsilon \cdot \sqrt{1-\alpha_2^{t+1}}$, for default parameter ϵ. All the vector operations are element-wise.

We now provide a more detailed explanation of NNME. Note that the calculations in Sections 4.2-4.3 are for $n = 1$. For a general n, write $w^{(n)} = \{w_i\}_{i=1}^n$ and $y^{(n)} = \{y_i\}_{i=1}^n$. At each epoch, we draw samples $\{z_{ik}\}_{1 \le i \le n, 1 \le k \le K}$ IID from $\mathcal{N}(0, I_d)$ and obtain $x_{ik} = x(\phi, z_{ik})$. Similar to (15) and (19), we have

(A.2)

$$\widehat{\nabla_{\theta,\gamma} Q}(\theta, \gamma, \phi | w^{(n)}, y^{(n)}) = \sum_{i=1}^n \sum_{k=1}^K \frac{\beta_{ik}}{\sum_{\ell=1}^K \beta_{i\ell}} \nabla_{\theta,\gamma} \log \left(\frac{p_{\theta,\gamma}(w_i, y_i, x(\phi, z_{ik}))}{q_\phi(x(\phi, z_{ik}) | w_i, y_i)} \right),$$

$$\widehat{\nabla_{\phi} Q}(\theta, \gamma, \phi | w^{(n)}, y^{(n)}) = \sum_{i=1}^n \sum_{k=1}^K \left(\frac{\beta_{ik}}{\sum_{\ell=1}^K \beta_{i\ell}} \right)^2 \left[\frac{\partial}{\partial x_{ik}} \log \left(\frac{p_{\theta,\gamma}(w_i, y_i, x_{ik})}{q_\phi(x_{ik} | w_i, y_i)} \right) \right] \frac{\partial}{\partial \phi} x(\phi, z_{ik}),$$

Algorithm 1 NNME

```
Data: Training data $\{w_i, y_i\}_{i=1}^n$ and pre-specified values $\{x_i\}_{i=1}^m$ for evaluating $f_\theta(x)$.
Input: The density $p_U(\cdot)$ of measurement error, the neural network structure $T$, parameters $(\lambda_0, \lambda_1, \lambda_2)$, number of epochs $\mathrm{Max\_Epoch}$, and number of importance samples $K$.
Pre-processing: Standardize $y$ and each covariate of $w$.
Initialization: Pre-train networks for $f_\theta$ and $g_\gamma$ using $\{w_i, y_i\}_{i=1}^n$ as if there is no measurement error. Initialize $(\theta, \gamma)$ from pre-training, $\phi$ randomly from normal distribution, and $\sigma^2$ by the MSE from pre-training.
  for epoch = 1 to Max_Epoch do
      • Draw Monte Carlo samples $\{z_{ik}\}_{1\le i\le n, 1\le k\le K}$ IID from $\mathcal{N}(0, I_d)$.
      • Compute the gradients $\widehat{\nabla_\theta Q}$, $\widehat{\nabla_\gamma Q}$, and $\widehat{\nabla_\phi Q}$ as in (A.2). Let $\widehat{\nabla_\theta Q}_* = \widehat{\nabla_\theta Q} + 2\lambda_0\theta$,
        $\widehat{\nabla_\gamma Q}_* = \widehat{\nabla_\gamma Q} + 2\lambda_2\gamma$ and $\widehat{\nabla_\phi Q}_* = \widehat{\nabla_\phi Q} + 2\lambda_1\phi$ (to account for the L2 penalty).
      • Update $(\theta, \gamma, \phi)$ via gradient ascent as in (A.1).
      • Update $\sigma^2$ by (20) using the training samples.
  end for
Output: $(\hat\theta, \hat\gamma, \hat\phi)$ and the estimated values $\{f_{\hat\theta}(x_i)\}_{i=1}^m$.
```

where $\beta_{ik} = p_{\theta,\gamma}(w_i, y_i, x_{ik})/q_\phi(x_{ik}|w_i, y_i)$ and the expressions of $p_{\theta,\gamma}(w, y, x)$ and $q_\phi(x|w, y)$ are given by (6) and (13), respectively. The gradient computation involves calculation of $\nabla_\theta f_\theta$, $\nabla_\gamma g_\gamma$, and $(\nabla_\phi \mu_\phi, \nabla_\phi \Sigma_\phi)$, which are computed via the back propagation algorithm (McClelland, Rumelhart and Group, 1986; Hecht-Nielsen, 1992). We also use the L_2-regularization trick to reduce numerical instability. We add a penalty $\lambda_0\|\theta\|^2 + \lambda_1\|\phi\|^2 + \lambda_2\|\gamma\|^2$ to the objective function. It is similar in spirit to the ridge regression and helps stabilize the numerical performance. This L_2 penalty changes nothing of the gradient ascent algorithm except for an additional term to each of the gradients $(\nabla_\theta, \nabla_\phi, \nabla_\gamma)$; see Step 2 in the for loop of Algorithm 1. The noise variance σ^2, which is assumed known in derivations of Sections 4.2-4.3, can be estimated along with training the neural network.

Regarding the input of Algorithm 1, the measurement error density $p_U(\cdot)$ is supplied by the user, as in other measurement error methods. In the literature, it is common to assume that the measurement errors follow $\mathcal{N}(0, \sigma_0^2 I_d)$, where σ_0^2 is estimated from other data source or determined by prior knowledge. By default, we set the number of Monte Carlo samples as $K = 50$ and the maximum number of epochs as Max_Epoch = 500. The neural network structure T including the number of hidden layers of encoder and decoder and the L2-penalty coefficients $(\lambda_0, \lambda_1, \lambda_2)$ are chosen by a 5-fold cross validation. The validation loss is calculated as follows: for each test sample (w_i, y_i), we draw $\{z_{ik}\}_{k=1}^K$ IID from $\mathcal{N}(0, I_d)$ and compute x_{ik} and β_{ik} by plugging in the estimated parameters $(\hat\theta, \hat\gamma, \hat\phi)$ and the testing (w_i, y_i). These numbers are then plugged into (20) to give the validation loss. The implementation of gradient ascent is via *Adam*, with batch size equal to $\min\{512, n\}$. *Adam* requires four parameters $(\alpha_0, \alpha_1, \alpha_2, \epsilon)$ to determine the learning rate (see the paragraph below (A.1)), where we set as the default values as $(0.001, 0.9, 0.999, 10^{-8})$.

To initialize the neural network representing the regression function f_θ, we pre-train it as if there is no measurement error. When the measurement error is small, the fitted function ignoring measurement error would be close to the true function. In pre-training, the loss function is the mean squared error (MSE) plus an L2 penalization term, $\lambda_0\|\theta\|^2$. We also pre-train the neural network g_γ representing the density of X ignoring measurement error. The loss function is the log-likelihood of $\{w_i\}$ with L2 penalization term $\lambda_2\|\gamma\|^2$. As the neural network has many parameters and the objective function is non-convex, we randomly initialize the parameters five times and select the neural network with the smallest training error during pre-training. This provides a good starting point of NNME, and prevents NNME to be trapped into bad local modes.

APPENDIX B: VARIATIONAL INFERENCE VERSUS MAXIMIZING JOINT LIKELIHOOD

The neural network method we propose for measurement error models adopts the variational inference framework, where parameters are estimated by maximizing an evidence lower bound (ELBO) of the marginal log-likelihood. In this appendix we consider a different approach, that is, maximizing the joint likelihood of $\{(x_i, w_i, y_i)\}_{i=1}^n$, with respect to both model parameters and unobserved values of x_i's (MJL, henceforth). See Section 3.2. The neural network structure that implements this estimator is shown in Figure A1.

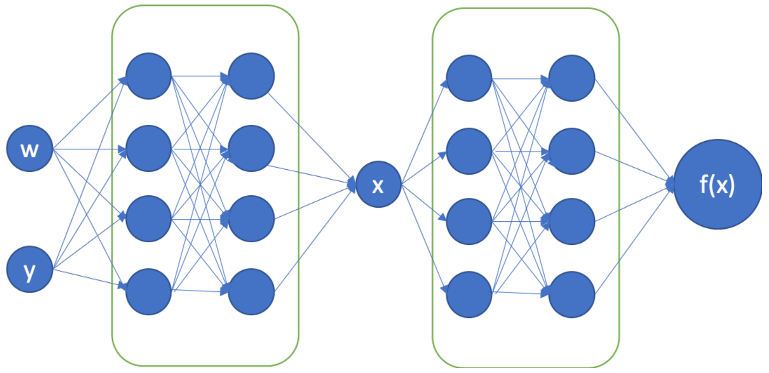
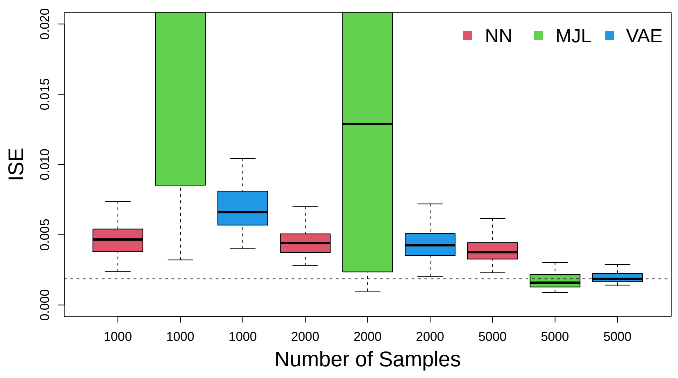


Fig A1: The neural network structure for the maximizing joint likelihood framework.

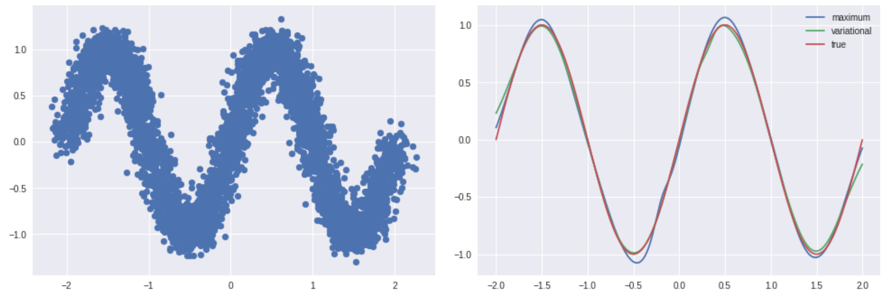
In comparison, the variational inference framework approximates the posterior distribution of x by $\mathcal{N}(\mu_\phi(w,y), \Sigma_\phi(w,y))$ and maximizes an integral with respect to this distribution, which serves as a lower bound of the marginal log-likelihood. See equation (8).

We compare the "maximizing joint likelihood" framework and the variational inference framework. For a fair comparison, we use the basic variational inference approach, where the number of importance samples is $K = 1$. It corresponds to the method VAE in Figure 4. We solve both the MJL and VAE by the gradient ascent algorithm, where the gradient for MJL is computed directly, the gradient for VAE is estimated as in (12) with $K = 1$, and the step size for both is chosen by *Adam* (Kingma and Ba, 2014). The noise variance σ^2 is optimized together with (θ, ϕ), similarly as in Section 4.4.

We consider the simulation setting 1 of Section 4.4, where $f(x) = \sin(\pi x)$ is the true regression function. Figure A2 (a) suggests that the variational inference framework is better than the "maximizing joint likelihood" framework, especially when n is small or moderate. Intuitively the MJL framework "imputes" the "missing data" to maximize the joint likelihood, whereas the VAE approaches marginalizing the "missing data" like in the EM algorithm, but based on a slightly incorrect distribution. In Figure A2 (b) we plot the estimated curves by MJL and VAE, where $n = 5000$ and $\sigma_0 = 0.1$. At each x where $f(x)$ reaches a local maximum



(a) Simulation results of NN, MJL and VAE.



(b) Example of simulated data and fitted results. Left: data. Right: fitted curves.

Fig A2: Comparison of maximizing joint likelihood (MJL) and variational inference (VAE, K=1). Settings are the same as in setting 1 of Section 4.4. In (a), Y-axis: box plots of ISE based on 50 repetitions, X-axis: number of samples, the dash line is the median ISE of VAE when $n = 5000$.

or minimum, the estimated curve by MJL has a bias in the neighborhood of x. This is because that MJL fails to take into account the uncertainty of X given (W, Y). Intuitively, $\hat{f}(x)$ in MJL is determined only by those (w_i, y_i) such that $h_\phi(w_i, y_i) \approx x$, while $\hat{f}(x)$ in VAE is determined by more (w_i, y_i) corresponding to a wider range of $h_\phi(w_i, y_i)$. As a result, when $f(x)$ reaches a local minimum or maximum, it is likely to have extreme values of y_i, and even a few such y_i can drag $\hat{f}(x)$ to be extreme in MJL; this will not happen in VAE because an individual value of y_i is less influential.

APPENDIX C: COMPARISON OF THE PRIOR MODELS FOR X IN NNME

The proposed method, NNME, is flexible to accommodate different kinds of models (prior distributions) for X. In the example of Section 3.2, we tested 4 variants of NNME, where the model for X is either the correct parametric model (2-component Gaussian mixture), or a misspecified parametric model (t-distribution, or 4-component Gaussian mixture), or a neural network model (NICE). For NICE, we use 3 transformations and each transformation g_j is represented by a FNN with 1 hidden layer, 32 nodes per layer and ReLU as the activation function.

The data generation in this experiment is as follows: fixing $\beta = 16$, we generate f from a 2-dimensional Gaussian processes as in (A.3). In detail, we first generate $\{x_i\}_{1\leq i\leq n}$ from the distribution of X, which is a 2-component mixture of multivariate Gaussian distributions as

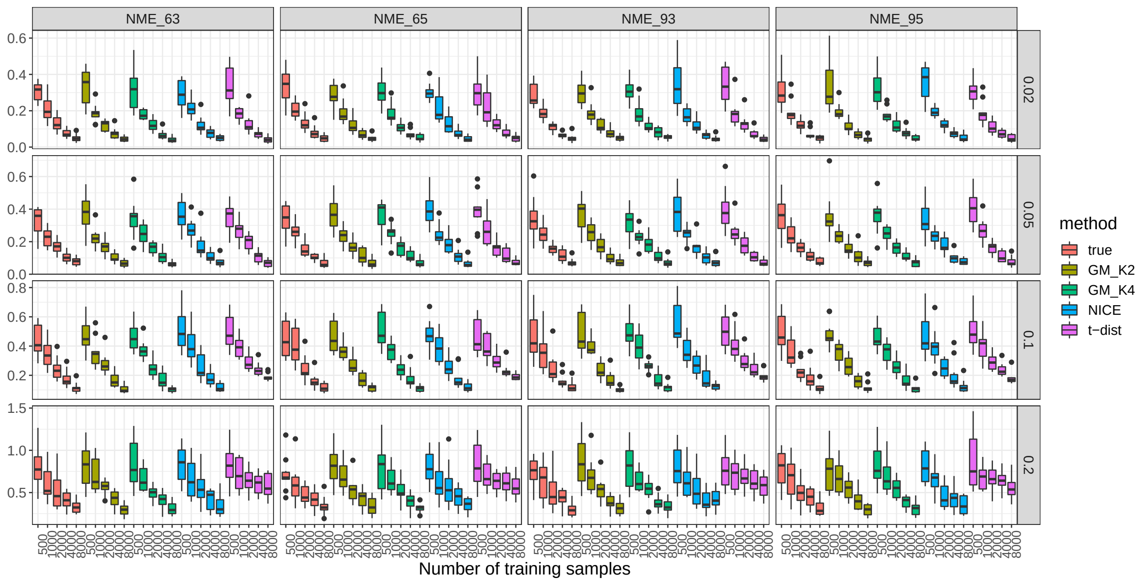


Fig A3: Comparison of different models for X in NNME. f generated from a 2-dimensional Gaussian process (see example in Section 3.2). Y-axis: ISE. X-axis: sample size n.

in (9). Next, we construct an $n \times n$ covariance matrix Σ, with $\Sigma_{ij} = \exp(-\beta \|x_i - x_j\|^2)$, and then generate $(f(x_1), f(x_2), \ldots, f(x_n))$ from $\mathcal{N}(0, \Sigma)$.

In Section 3.2 we have reported the results for a few values of model parameters. Here we investigate more settings. We let the sample size n range in $\{1000, 2000, 4000, 8000\}$ and let the measurement error standard deviation range in $\{0.02, 0.05, 0.1, 0.2\}$. We also vary the neural network structures: Let (ℓ_1, ℓ_2) be the number of hidden layers of the two FNNs for representing f_θ and q_ϕ, respectively. We let (ℓ_1, ℓ_2) range in $\{(6,3), (6,5), (9,3), (9,5)\}$. The performance is measured by the integrated squared error (ISE) in the region $([-1, 0.2] \times [-1, 0.5]) \cup ([-0.5, 1] \times [-0.2, 1])$; according to the distribution of X in (9), the probability of x_i's falling outside this region is negligible, and so we restrict the error evaluation to be in this region. The results are shown in Figure A3, demonstrating that NNME performs robustly with different choices of the X model when n is small, but shows superiority with NICE or 4-component Gaussian mixture when n is large.

APPENDIX D: SENSITIVITY TO THE DEPTH OF NEURAL NETWORKS

As we stated in Section 1, the neural network approach to measurement error models is relatively insensitive to the choice of tuning parameters. Main tuning parameters include the depth of the two FNNs for representing f and the proposal distribution of X. We now investigate the sensitivity of NNME to the depth of neural networks.

We consider a simulation setting where the x_i's are drawn uniformly from $[-1, 1]^2$ and the $f(x_i)$'s are generated from a Gaussian process as in (A.3) with $\beta = 16$ (see Section 5.3 or Appendix C for details about generating data from a Gaussian process). The measurement errors are drawn from $N(0, \sigma_0^2 I_2)$. For a set of values of (n, σ_0), we study the performance of NNME by varying the depth of neural networks. We place ℓ_1 hidden layers in the decoder (for representing f) and ℓ_2 hidden layers in the encoder (for the proposal distribution of X),

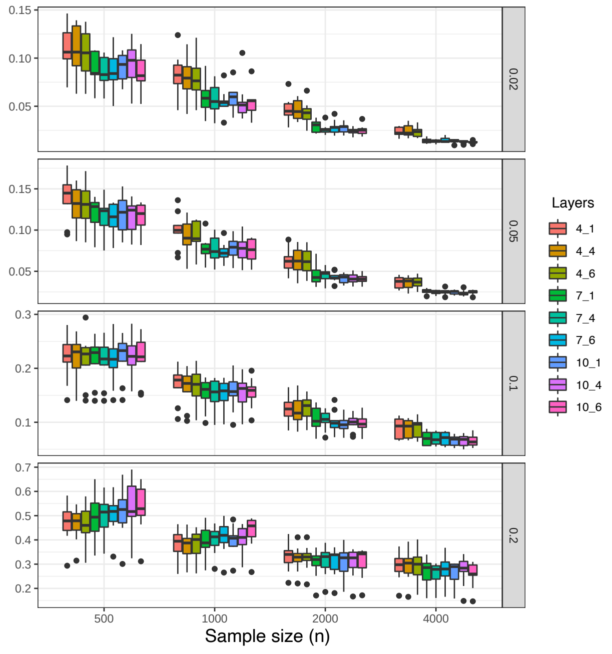


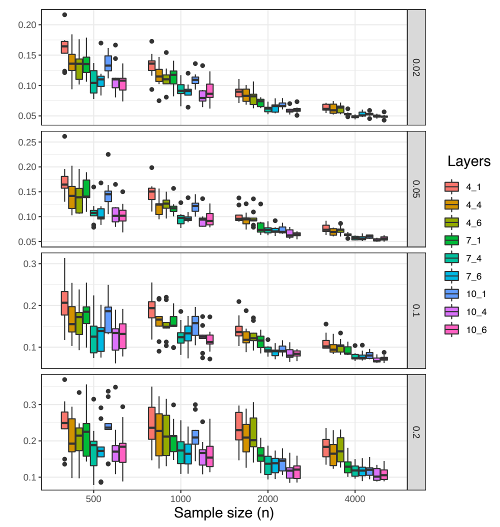
Fig A4: Sensitivity to depth of neural networks. The name '7_4' means that the decoder (for representing f) has 7 layers (6 hidden layers) and the encoder (for the proposal distribution of X) has 4 layers (3 hidden layers); other names are similar. Y-axis: ISE, x-axis: sample size. From top to bottom, σ_0 is 0.02, 0.05, 0.1, and 0.2, respectively.

where each layer has 32 nodes with ReLU activation functions; the encoder has an additional layer with linear activation functions. We let ℓ_1 range in $\{3,6,9\}$ and ℓ_2 range in $\{0,3,5\}$. The integrated squared error (ISE) is evaluate using a 72×72 grid.

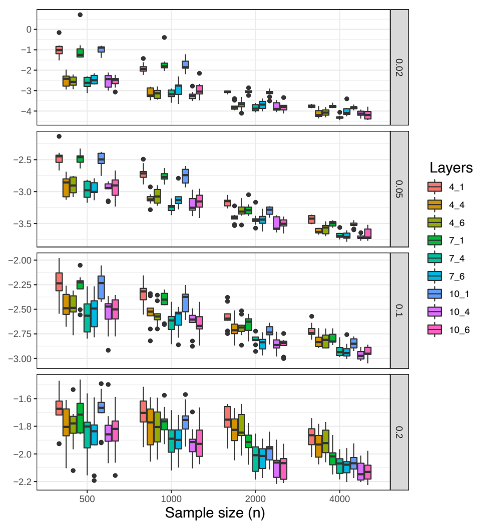
The results are shown in Figure A4. For most values of (n, σ_0), as long as the depth of neural networks satisfies $\ell_1 \geq 6$ and $\ell_2 \geq 3$, the performance is reasonably good. The only exceptions are when the sample size is small (e.g., $n = 500$) or when the measurement error is large (e.g., $\sigma_0 = 0.2$). In such cases, we will see that a procedure like cross validation can select the appropriate number of layers.

We then investigate the selection of depth of neural networks by two criteria. The first is the validation loss we eventually used, which is the weighted residual sum of squares in (20), evaluated on the validation data. To compute this quantity, we need to use estimated parameters from the training data and draw new Monte Carlo samples $z_{1:K}$; see Section 4.4 for details. We call this criterion the "estimated RSS." The second is the loss function (14) in NNME algorithm, evaluated on the validation data. It is an evidence lower bound (ELBO) of the marginal log-likelihood. Again, to obtain an approximation to this quantity, we need to use the fitted parameters from the training data and also draw new Monte Carlo samples $z_{1:K}$. We call it the "estimated ELBO." For both criteria, we calculate the 5-fold cross-validation version. Figure A5 shows the values for different choices of depth of neural networks.

We compare these criteria with the true ISE in Figure A4. As we change the depth of neural networks, the trend in the estimated RSS roughly matches with the trend in the true ISE. Especially, as we mentioned above, if the sample size is small (e.g., $n = 500$) or the measurement error is large (e.g., $\sigma_0 = 0.2$ and $n \leq 1000$), it is not always good to increase the



(a) Estimated RSS on validation data.



(b) Estimated ELBO on validation data.

Fig A5: Selection of depth of neural networks by different criteria.

depth of neural networks. In these cases, the cross-validation procedure, with the estimated RSS as the validation loss, can successfully guide us to the appropriate choice of depth. The other criterion, the estimated ELBO, is a lot more sensitive to the parameters ϕ and tends to select a larger number of layers for the encoder. In comparison, the estimated RSS we used in Section 4.4 is a better option for the validation loss.

APPENDIX E: GAUSSIAN PROCESS REGRESSION AND KRIGING METHODS

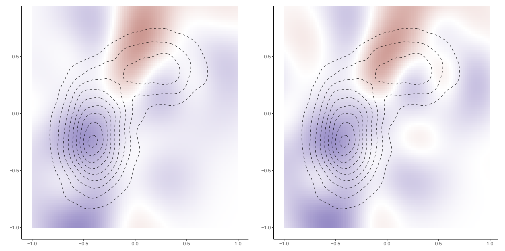
In our simulations, we include the kriging methods for Gaussian process regression, mainly for that they are fast to compute for $d > 1$. Gaussian process regression assumes that $f(x)$ is randomly generated from a stationary Gaussian process (SGP). It is common to use a radial basis exponential kernel in the SGP, i.e.,

$$\text{(A.3)}\quad Y = f(X) + \epsilon,\quad f(x) \sim \mathrm{SGP}(0, K(\cdot,\cdot)),\quad K(x,y) = \tau^2 e^{-\beta\|x-y\|^2},\quad \epsilon \sim \mathcal{N}(0, \sigma^2 I_n).$$

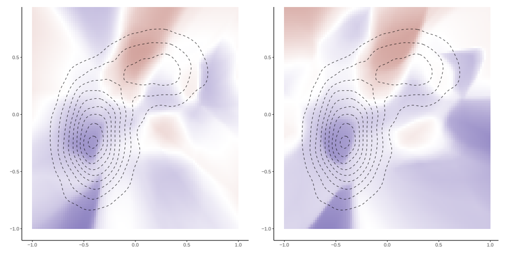
With no measurement error, $\hat{f}(x) = K(x, \mathbf{x}_n)(K(\mathbf{x}_n, \mathbf{x}_n) + \sigma^2 I_n)^{-1}\mathbf{y}_n$ is the the best linear unbiased predictor, where $\mathbf{y}_n = \{y_i\}_{1\le i\le n}$, $K(x, \mathbf{x}_n) = \{K(x, x_i)\}_{1\le i\le n}$, and $K(\mathbf{x}_n, \mathbf{x}_n)$ is a $n \times n$ matrix whose (i,j)th entry is $K(x_i, x_j)$. The kriging method first estimates (τ, β, σ) by maximizing the likelihood and then plugs these parameters into the best linear unbiased predictor. In the presence of measurement errors, Cressie and Kornak (2003) proposed a variant of the best linear unbiased predictor by replacing $K(x, \mathbf{x}_n)$ by $\tilde{K}_*(x, \mathbf{w}_n)$ and $K(\mathbf{x}_n, \mathbf{x}_n)$ by $\tilde{K}(\mathbf{w}_n, \mathbf{w}_n)$, where

$$\tilde{K}(w_i, w_j) = \begin{cases} \tau^2 \frac{\exp\left(-\frac{\beta}{1+4\beta\sigma_0^2}\|w_i - w_j\|^2\right)}{(1+4\beta\sigma_0^2)^{d/2}}, & w_i \ne w_j, \\ \tau^2, & w_i = w_j; \end{cases} \quad \tilde{K}_*(x, w_i) = \tau^2 \frac{\exp(-\frac{\beta}{1+\beta\sigma_0^2}\|x - w_i\|^2)}{(1+\beta\sigma_0^2)^{d/2}}.$$

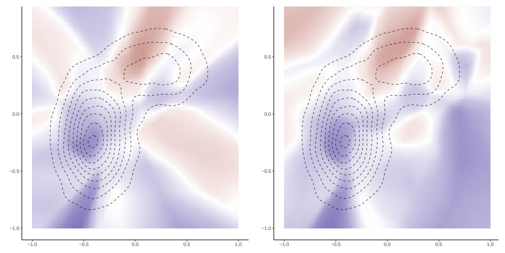
Same as before, σ_0^2 is the variance of measurement errors. The estimator of $f(x)$ is a plug-in version where (τ, β, σ) are estimated by maximizing a pseudo-likelihood. We include both



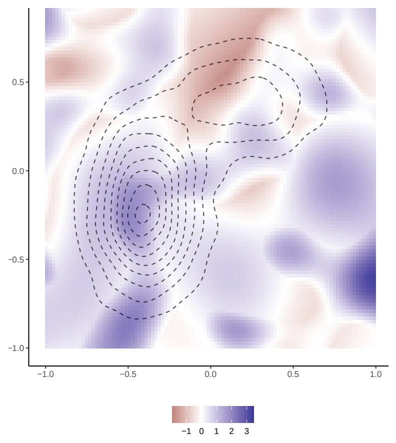
(b) KALE ($n = 500, 2000$).



(c) NNME_GM4 ($n = 2000, 8000$).



(d) NNME_NICE ($n = 2000, 8000$).



(a) True function.

Fig A6: Example 3 in Section 5 (two-dimensional functions generated from Gaussian processes), with measurement error $\sigma_0 = 0.1$, and their estimates. The dashed lines are contours of the density of the training samples. The function value is indicated by the color.

the kriging method that accounts for measurement errors (abbreviated as "KALE") and the standard kriging for error-free cases (abbreviated as "KILE").

In Example 3 of Section 5, we consider a setting where f is the maximum of two Gaussian processes in $\mathbb{R}^2$. An example of the realized $f(x)$ and estimated functions is shown in Figure A6.

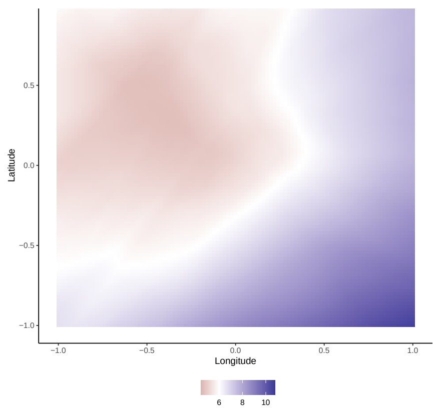
APPENDIX F: AN EXAMPLE OF TWO-DIMENSIONAL FUNCTIONS GENERATED FROM A NEURAL NETWORK

Example 4: Two-dimensional functions generated from a neural network. We let $f : [-1, 1]^2 \to \mathbb{R}$ be defined by a neural network with 5 fully connected layers, 32 nodes per layer, and ReLU as activation function. The weights and thresholds in the neural network are randomly sampled from $\mathcal{N}(0, 1)$ in the first layer and $\mathcal{N}(0, 0.2^2)$ in the rest of layers. One realization of $f(x)$ is shown in Figure A7 (a).

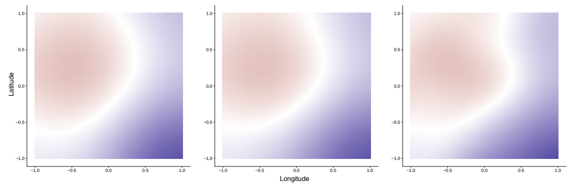
Given $f(x)$, we generate $\{x_i\}_{i=1}^n$ uniformly from $[-1, 1]^2$. Next, we generate w_i's and y_i's according to model (4), where the measurement error has a multivariate normal distribution $\mathcal{N}(0, \sigma_0^2 I_2)$. We fix $\sigma = 0.2$ and let σ_0 takes values in $\{0.1, 0.2\}$. We also vary the sample size by letting n range in $\{500, 1000, 2000\}$. For NNME, we let the decoder network have 5 layers

TABLE A1
Example 4 (two-dimensional function from a neural network). The ISE evaluated at a 72×72 uniform grid, as well as its standard deviation (in brackets), is shown.

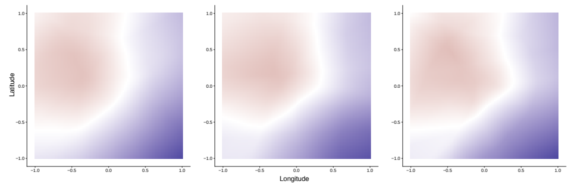
	$\sigma_0 = 0.1,\ \sigma = 0.2$			$\sigma_0 = 0.2,\ \sigma = 0.2$		
	n=500	n=1000	n=2000	n=500	n=1000	n=2000
KILE	.0071 (.0006)	.0052 (.0005)	.0041 (.0004)	.0170 (.0020)	.0148 (.0017)	.0133 (.0015)
KALE	**.0068** (.0005)	.0049 (.0004)	.0037 (.0004)	.0143 (.0017)	.0115 (.0012)	.0096 (.0010)
NN	.0079 (.0007)	.0051 (.0004)	.0042 (.0004)	.0187 (.0024)	.0145 (.0018)	.0136 (.0018)
NNME	.0070 (.0003)	**.0046** (.0003)	**.0034** (.0003)	**.0135** (.0010)	**.0091** (.0008)	**.0074** (.0008)



(a) True function.



(b) KALE estimates with different (n, σ_0).



(c) NNME estimates with different (n, σ_0).

Fig A7: Example 4 in Section 5 (two-dimensional function generated from a neural network). In (b)-(c), the three panels from left to right correspond to $(n, \sigma_0) = (500, 0.2)$, $(1000, 0.2)$, and $(2000, 0.1)$

and the encoder network have 2 layers, where each layer has 32 nodes; we use a parametric model for X, assuming that all the coordinates of X are independently distributed as $2 \cdot t_3$. NN has only a decoder network, which also consists of 5 layers with 32 nodes per layer. We measure the performance of each method by the ISE, evaluated on a 72×72 uniform grid. The ISE averaged over 10 repetitions is reported in Table A1. It suggests that NNME has the best or nearly the best performances in all settings. The two methods ignoring measurement errors, NN and KILE, underperform their respective counterpart. KALE has a similar performance as NNME when $\sigma_0 = 0.1$, but its performance is inferior to NNME's when $\sigma_0 = 0.2$.

REFERENCES

CRESSIE, N. and KORNAK, J. (2003). Spatial statistics in the presence of location error with an application to remote sensing of the environment. *Statistical Science* **18** 436 – 456.

HECHT-NIELSEN, R. (1992). Theory of the backpropagation neural network. In *Neural Networks for Perception* 65–93. Elsevier.

KINGMA, D. P. and BA, J. (2014). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

MCCLELLAND, J. L., RUMELHART, D. E. and GROUP, P. R. (1986). Parallel distributed processing. *Explorations in the Microstructure of Cognition* **2** 216–271.