Deep Neural Network Informed Markov Chain Monte Carlo Methods

by

Ashlynn Crisp

A STAT 604 Internship Project

Research conducted at Lawrence Livermore National Laboratory
Under the supervision of
Sohail Reddy and Hillary Fairbanks
As part of the Computing Scholars Program

Abstract

In this work, we investigate methods for using deep neural networks for uncertainty quantification with application to groundwater flow models. We use Markov chain Monte Carlo methods to take advantage of observed data and to estimate the uncertainty of the model output. We test gradient-informed methods using gradients obtained through deep neural networks which are computationally cheap to use. Results are shown for the Metropolis-adjusted Langevin Algorithm as well as two algorithms which we introduce based on delayed-acceptance methods.

Fariborz Maseeh Department of Mathematics and Statistics

Portland State University

January 26, 2024

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-TR-857609)

1 Introduction

In subsurface flow modeling, quantifying the uncertainty of model parameters and the corresponding uncertainly on output quantities is a crucial task for groundwater management. Markov chain Monte Carlo (MCMC) methods can take advantage of observed data to estimate parameters in a Bayesian setting. However, MCMC can be slow to converge and produce highly correlated samples when the dimensions of the parameters is high. Using gradients for the posterior distribution can help samplers explore the parameter space more efficiently, but obtaining gradients can be computationally challenging.

The goal of this project is to assess the performance of gradient information provided via backpropagation of deep neural networks (DNNs) in MCMC algorithms. Our application is modeling groundwater flow where the flux q, pressure u, random permeability field k, and source term f are related through

$$\begin{aligned} \nabla \cdot (k(x)\nabla u(x)) &= f(x) && \text{in } \Omega \\ u(x) &= u_D(x) && \text{on } \Gamma_D \\ k(x)\frac{\partial q(x)}{\partial n} &= 0 && \text{on } \Gamma_N \end{aligned} \tag{1}$$

with

$$k^{-1}(x)q(x) = \nabla u(x), \quad \nabla \cdot q(x) = f(x)$$

Here, the uncertain permeability field is $k := \exp(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is modeled as a log-Gaussian random field with covariance function $\text{cov}(x, y) := \sigma^2 e^{(-\kappa||x-y||)}$. Numerical solutions to Equation 1 were performed using NGSolve [8] with quadrilateral 32×32 mesh assuming a unit square domain. We represent realizations of $\boldsymbol{\theta}$ using the Karhunen–Loève (KL)-expansion given by

$$\boldsymbol{\theta} = \sum_{i=1}^{r} \sqrt{\lambda_i}\alpha_i\Phi_i(x) \tag{2}$$

where $\{\Phi_i\}_{i=1}^r$ are the eigenfunctions and $\{\lambda_i\}_{i=1}^r$ are the eigenvalues of the covariance matrix of $\boldsymbol{\theta}$ and each $\alpha_i \sim \mathcal{N}(0, 1)$. We set $\kappa = 0.25$, $\sigma^2 = 1$, and $r = 50$ in this project. Our quantity of interest is the flux Q across the outflow boundary Γ_{out} defined as

$$Q = \frac{1}{|\Gamma_{\text{out}}|}\int_{\Gamma_{\text{out}}} q(\cdot, \omega) \cdot \mathbf{n}\, dS, \tag{3}$$

where $\mathbf{n}$ is the outward unit vector normal to $\Gamma_{\text{out}} \subset \partial D$.

We generated synthetic observed pressure data $\boldsymbol{u}_{\text{obs}}$ which is the m local average pressure evaluations extracted from the field $\boldsymbol{u}$. Let $\mathcal{G} : \boldsymbol{\alpha} \mapsto \boldsymbol{\theta}$, $\mathcal{H} : \boldsymbol{\theta} \mapsto \mathbf{k}$, $\mathcal{J} : \mathbf{k} \mapsto \boldsymbol{u}$, and $\mathcal{K} : \boldsymbol{u} \mapsto \boldsymbol{u}_{\text{obs}}$. The generated data is assumed to follow

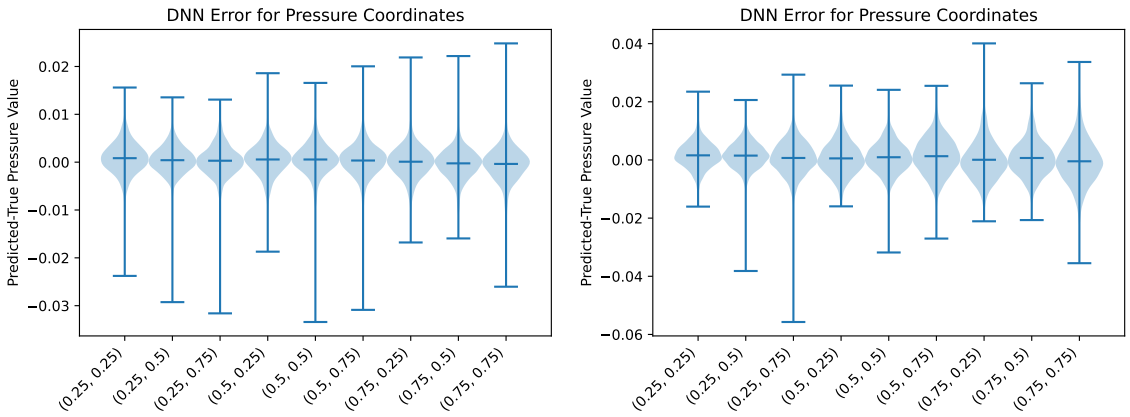
$$\boldsymbol{u}_{\text{obs}} = \boldsymbol{\mathcal{F}}(\boldsymbol{\alpha}_{\text{obs}}) + \epsilon$$

where $\boldsymbol{\mathcal{F}} := \mathcal{K} \circ \mathcal{J} \circ \mathcal{H} \circ \mathcal{G}$ maps $\boldsymbol{\alpha}$ to $\boldsymbol{u}$ and the ϵ term is additive Gaussian noise which was assumed to be uncorrelated. For this work, we set $m = 9$ and the covariance of the error term to be $\sigma_e^2 = 0.005$ so that $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma_e^2\boldsymbol{I})$. Using this observed data, the posterior π for $\boldsymbol{\alpha}$ is given by

$$\pi(\boldsymbol{\alpha}|\boldsymbol{u}_{\text{obs}}) \propto \boldsymbol{\mathcal{L}}(\boldsymbol{u}_{\text{obs}}|\boldsymbol{\alpha})\pi_0(\boldsymbol{\alpha})$$

where $\boldsymbol{\mathcal{L}}$ denotes the likelihood distribution and π_0 denotes the prior distribution on $\boldsymbol{\alpha}$. The likelihood is given by

$$\boldsymbol{\mathcal{L}}(\boldsymbol{u}_{\text{obs}}|\boldsymbol{\alpha}) = \exp\left(-\frac{1}{2\sigma_e^2}||\boldsymbol{\mathcal{F}}(\boldsymbol{\alpha}) - \boldsymbol{u}_{\text{obs}}||^2\right)$$



(a) FCNN

(b) CNN

Figure 1: Neural network accuracies

2 Methods

2.1 Neural Networks

To train the networks, 10,000 samples for $\boldsymbol{\alpha}$ were generated from $N(\mathbf{0}, \mathbf{I})$ and the corresponding values for $\boldsymbol{\theta}, \boldsymbol{u}$, and Q were computed. The generated data is shown in Figures 4 - 7 in the appendix. Of these 10,000 samples, 8,000 were used for training the networks, and the remaining 2,000 were used as the validation set.

We used two neural network architectures: a fully connected neural network (FCNN) and a convolutional neural network (CNN), both developed using TensorFlow [1]. Both networks used min/max scaling for the input data and predicted the min/max-scaled output. After the training was complete, a layer was appended to the networks to unnormalize the output. Both networks were trained with the Adam optimizer [6] and a scheduler to decrease the learning rate as the epochs progressed. All weights were initialized using the Xavier normal method.

The first network was a fully connected neural network (FCNN) mapping $\boldsymbol{\alpha}$ directly to $\boldsymbol{u}$, avoiding the KL-expansion completely. The benefit of this approach is that the eigenfunctions need not be stored (from the original training data) nor recalculated which can be computationally expensive for finely resolved meshes. This network had one hidden layer consisting of 5,000 nodes and a ReLU activation function. The hidden layer was followed by the output layer which predicted the 9 $\boldsymbol{u}$ values and did not have an activation function applied. From the 9 pressure values, a custom layer to compute the log likelihood was appended after training to obtain a map from $\boldsymbol{\alpha}$ to the log likelihood so that the backpropagation could be done in one step. The FCNN used minibatching with a batch size of 32 and was trained for 700 epochs.

The second network was a convolutional neural network (CNN) to map between $\boldsymbol{\theta}$ and $\boldsymbol{u}$. This network used stored KL-expansions eigenmodes to map between $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ using Equation 2. The CNN had two convolutional layers which each used the ReLU activation function. The convolutional layers were followed by 2 dense layers which also had ReLU activation. The output layer was a fully-connected layer without activation. Similarly to the FCNN, the CNN had a custom analytical layer to compute the log likelihood appended after training. The CNN used a batch size of 64 and was trained for 300 epochs.

2.2 Markov Chain Monte Carlo

We used the Metropolis-Hastings (MH) and preconditioned Crank-Nicolson (pCN) algorithms as our baseline MCMC results since neither methods require gradient information. The MH algorithm [3] is a very common and versatile algorithm which can be applied with any prior distribution. The pCN algorithm is popular since it is an efficient algorithm for sampling in high dimensions [4], however, it has constraints on the prior which limits the application of this algorithm to certain problems. The MH and pCN algorithms we used are summarized in Algorithms 1 and 2.

Algorithm 1: Random Walk Metropolis-Hastings

Pick starting value for chain $\boldsymbol{\alpha}_0$. For $i = 1, 2, \ldots, N$:

- Step 1: Given $\boldsymbol{\alpha}_i$, compute proposal as

$$\boldsymbol{\alpha}' = \boldsymbol{\alpha}_i + \boldsymbol{\epsilon_i}$$

  where $\boldsymbol{\epsilon_i} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$ and σ^2 is tuned based on the acceptance rate.

- Step 2: Compute the acceptance ratio

$$\rho = \min\left(1, \frac{\pi(\boldsymbol{\alpha}'|\boldsymbol{u}_{\text{obs}})}{\pi(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}})}\right)$$

- Step 3: Sample $t \sim \text{Unif}(0, 1)$. Set $\boldsymbol{\alpha}_{i+1}$ according to

$$\boldsymbol{\alpha}_{i+1} = \begin{cases} \boldsymbol{\alpha}' \text{ if } t \leq \rho \\ \boldsymbol{\alpha}_i \text{ otherwise} \end{cases}$$

We introduce two MCMC algorithms which were used to test the accuracy of the DNN gradients. The Metropolis-Adjusted Langevin Algorithm (MALA) includes gradients of the target distribution in the proposal and acceptance steps. The MALA algorithm using a DNN is included in Algorithm 3. However, since the DNNs will only provide a prediction of the evaluation of the target density, and therefore gradients corresponding to this prediction, we add an additional stage to MALA which checks the proposals using only the exact posterior distribution. The first method we propose which includes this step is hierarchical MALA (HMALA), defined in Algorithm 4, which is similar to the two-stage MALA algorithm proposed in [5]. This method simply replaces the true evaluation of the target density and the gradients with those provided by the DNN, and if a proposal is accepted by the DNN, it is then passed to the PDE level to be checked with a Metropolis step.

The second method, provided in Algorithm 5, includes a bias correction stage like the one proposed in [7]. The motivation is to correct for bias from the DNNs when predicting the log posterior and in turn also correct the gradients. Since we expect the bias to depend on where the proposal is in the parameter space, we aim to only use local bias information in the correction. To achieve this, only the bias from the previous accepted proposal is used for the correction stage. In bias-corrected HMALA, the DNN stage filters the proposals using the bias-corrected log posterior and corresponding gradient information. If the proposal passes the DNN stage, it is then evaluated on the PDE level. Additionally, MALA with gradients given by forward finite difference was run as a comparison to the MALA chains that use DNN gradients. We also tested the infMALA algorithm provided in [2] which is a mix of the pCN and MALA algorithms.

We tested three priors. The first was placing a $\mathcal{N}(\boldsymbol{\alpha}_{\text{obs}}, 0.01 * \boldsymbol{I})$ prior on $\boldsymbol{\alpha}$ to observe the behavior of the methods when the prior is heavily weighted around the observed data. The second prior was $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$, which was a natural choice since this is the distribution that $\boldsymbol{\alpha}$ is sampled from.

Algorithm 2: Preconditioned Crank-Nicolson

Pick starting value for chain $\boldsymbol{\alpha}_0$. For $i = 1, 2, \ldots, N$:

- Step 1: Given $\boldsymbol{\alpha}_i$, compute proposal as

$$\boldsymbol{\alpha}' = \sqrt{1-\beta^2}\boldsymbol{\alpha}_i + \beta\boldsymbol{\xi}_i$$

  where $\boldsymbol{\xi}_i$ is a sample from the prior distribution $\pi_0 = \mathcal{N}(\mathbf{0}, \Sigma)$ for some covariance matrix Σ. The step size parameter β is tuned based on the acceptance rate.

- Step 2: Compute acceptance ratio ρ where

$$\rho = \min\left(1, \frac{\boldsymbol{\mathcal{L}}(\boldsymbol{u}_{\text{obs}}|\boldsymbol{\alpha}')}{\boldsymbol{\mathcal{L}}(\boldsymbol{u}_{\text{obs}}|\boldsymbol{\alpha}_i)}\right)$$

- Step 3: Sample $t \sim \text{Unif}(0, 1)$. Set $\boldsymbol{\alpha}_{i+1}$ according to

$$\boldsymbol{\alpha}_{i+1} = \begin{cases} \boldsymbol{\alpha}' \text{ if } t \leq \rho \\ \boldsymbol{\alpha}_i \text{ otherwise} \end{cases}$$

Algorithm 3: Metropolis-Adjusted Langevin Algorithm (MALA)

Pick starting value for chain $\boldsymbol{\alpha}_0$. For $i = 1, 2, \ldots, N$:

- Step 1: Given $\boldsymbol{\alpha}_i$, compute proposal as

$$\boldsymbol{\alpha}' = \boldsymbol{\alpha}_i + \frac{\tau}{2}\nabla \log \pi(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}}) + \sqrt{\tau}\boldsymbol{\xi}_i$$

  where $\boldsymbol{\xi}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

- Step 2: Compute the acceptance probability ρ where

$$\rho = \min\left(1, \frac{q(\boldsymbol{\alpha}_i|\boldsymbol{\alpha}')\pi(\boldsymbol{\alpha}'|\boldsymbol{u}_{\text{obs}})}{q(\boldsymbol{\alpha}'|\boldsymbol{\alpha}_i)\pi(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}})}\right)$$

  and

$$q(\boldsymbol{\alpha}', \boldsymbol{\alpha}) \propto \exp\left(-\frac{||\boldsymbol{\alpha}' - \boldsymbol{\alpha}_i - \frac{\tau}{2}\nabla \log \pi(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}})||^2}{2\tau}\right)$$

- Step 3: Sample $t \sim \text{Unif}(0, 1)$. Set $\boldsymbol{\alpha}_{i+1}$ according to

$$\boldsymbol{\alpha}_{i+1} = \begin{cases} \boldsymbol{\alpha}' \text{ if } t \leq \rho \\ \boldsymbol{\alpha}_i \text{ otherwise} \end{cases}$$

Note that replacing π with the DNN posterior π^* in this algorithm produces a DNN MALA chain with gradients $\nabla \log \pi^*(\boldsymbol{\alpha}_i)$ provided through DNN backpropagation.

Algorithm 4: Hierarchical Metropolis-Adjusted Langevin Algorithm (HMALA)

Pick starting value for chain $\boldsymbol{\alpha}_0$. For $i = 1, 2, \ldots, N$:

- Step 1: Given $\boldsymbol{\alpha}_i$, compute proposal as

$$\boldsymbol{\alpha}' = \boldsymbol{\alpha}_i + \frac{\tau}{2}\nabla \log \pi^*(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}}) + \sqrt{\tau}\boldsymbol{\xi}_i$$

  where $\boldsymbol{\xi}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and π^* is the DNN-obtained posterior

- Step 2: Compute acceptance probability ρ^* using π^*

$$\rho^* = \min\left(1, \frac{q^*(\boldsymbol{\alpha}_i|\boldsymbol{\alpha}')\pi^*(\boldsymbol{\alpha}'|\boldsymbol{u}_{\text{obs}})}{q^*(\boldsymbol{\alpha}'|\boldsymbol{\alpha}_i)\pi^*(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}})}\right)$$

  where

$$q^*(\boldsymbol{\alpha}', \boldsymbol{\alpha}) \propto \exp\left(-\frac{||\boldsymbol{\alpha}' - \boldsymbol{\alpha}_i - \frac{\tau}{2}\nabla \log \pi^*(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}})||^2}{2\tau}\right)$$

- Step 3: Sample $t \sim \text{Unif}(0, 1)$. Set $\hat{\boldsymbol{\alpha}}_{i+1}$ according to

$$\hat{\boldsymbol{\alpha}}_{i+1} = \begin{cases} \boldsymbol{\alpha}' \text{ if } t \leq \rho^* \\ \boldsymbol{\alpha}_i \text{ otherwise} \end{cases}$$

- Step 4: If the proposal $\boldsymbol{\alpha}'$ is rejected in Step 3, set $\boldsymbol{\alpha}_{i+1} = \boldsymbol{\alpha}_i$ and return to Step 1.

- Step 5: Compute acceptance ratio ρ using π:

$$\rho = \min\left(1, \frac{\pi(\boldsymbol{\alpha}'|\boldsymbol{u}_{\text{obs}})}{\pi(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}})}\right)$$

- Step 6 Sample $s \sim \text{Unif}(0, 1)$. Set $\boldsymbol{\alpha}_{i+1}$ according to

$$\boldsymbol{\alpha}_{i+1} = \begin{cases} \hat{\boldsymbol{\alpha}}' \text{ if } s \leq \rho \\ \boldsymbol{\alpha}_i \text{ otherwise} \end{cases}$$

Algorithm 5: Bias-corrected Hierarchical Metropolis-Adjusted Langevin Algorithm (bias-MALA)

Pick starting value for chain $\boldsymbol{\alpha}_0$. Initialize $\boldsymbol{\mu}_{\text{bias},0} = \mathbf{0}$ and $\boldsymbol{\Sigma}_{\text{bias},0} = \boldsymbol{O_{9,9}}$. For $i = 1, 2, \ldots, N$:

- Step 1: Given $\boldsymbol{\alpha}_i$, compute proposal as

$$\boldsymbol{\alpha}' = \boldsymbol{\alpha}_i + \frac{\tau}{2}\nabla \log \pi_{\text{bias}}(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}}) + \sqrt{\tau}\boldsymbol{\xi}_i$$

  with $\boldsymbol{\xi}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and

$$\pi_{\text{bias}}(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}}) \propto \hat{\boldsymbol{\mathcal{L}}}_{\text{bias}}(\boldsymbol{u}_{\text{obs}}|\boldsymbol{\alpha}_i)\pi_0(\boldsymbol{\alpha}_i)$$

  where $\hat{\boldsymbol{\mathcal{L}}}_{\text{bias}}(\boldsymbol{u}_{\text{obs}}|\boldsymbol{\alpha}_i)$ is defined as

$$\exp\left(-\frac{1}{2}\left(\hat{\boldsymbol{\mathcal{F}}}(\boldsymbol{\alpha}_i) + \boldsymbol{\mu}_{\text{bias},i-1} - \boldsymbol{u}_{\text{obs}}\right)^T \left(\boldsymbol{\Sigma}_{\text{bias},i-1} + \boldsymbol{\Sigma}_e\right)^{-1} \left(\hat{\boldsymbol{\mathcal{F}}}(\boldsymbol{\alpha}_i) + \boldsymbol{\mu}_{\text{bias},i-1} - \boldsymbol{u}_{\text{obs}}\right)\right)$$

  and $\hat{\boldsymbol{\mathcal{F}}}$ is the DNN.

- Step 2: Compute acceptance probability ρ_{bias}

$$\rho_{\text{bias}} = \min\left(1, \frac{q_{\text{bias}}(\boldsymbol{\alpha}_i|\boldsymbol{\alpha}')\pi_{\text{bias}}(\boldsymbol{\alpha}'|\boldsymbol{u}_{\text{obs}})}{q_{\text{bias}}(\boldsymbol{\alpha}'|\boldsymbol{\alpha}_i)\pi_{\text{bias}}(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}})}\right)$$

  where

$$q_{\text{bias}}(\boldsymbol{\alpha}', \boldsymbol{\alpha}) \propto \exp\left(-\frac{||\boldsymbol{\alpha}' - \boldsymbol{\alpha}_i - \frac{\tau}{2}\nabla \log \pi_{\text{bias}}(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}})||^2}{2\tau}\right)$$

- Step 3: Sample $t \sim \text{Unif}(0, 1)$. Set $\hat{\boldsymbol{\alpha}}_{i+1}$ according to

$$\hat{\boldsymbol{\alpha}}_{i+1} = \begin{cases} \boldsymbol{\alpha}' \text{ if } t \leq \rho_{\text{bias}} \\ \boldsymbol{\alpha}_i \text{ otherwise} \end{cases}$$

- Step 4: If $\boldsymbol{\alpha}'$ is rejected in Step 3, set $\boldsymbol{\alpha}_{i+1} = \boldsymbol{\alpha}_i$.
  Otherwise, compute acceptance ratio ρ

$$\rho = \min\left(1, \frac{\pi(\boldsymbol{\alpha}'|\boldsymbol{u}_{\text{obs}})}{\pi(\boldsymbol{\alpha}_i|\boldsymbol{u}_{\text{obs}})}\right)$$

- Step 5: Sample $s \sim \text{Unif}(0, 1)$. Set $\boldsymbol{\alpha}_{i+1}$ according to

$$\boldsymbol{\alpha}_{i+1} = \begin{cases} \hat{\boldsymbol{\alpha}}' \text{ if } s \leq \rho \\ \boldsymbol{\alpha}_i \text{ otherwise} \end{cases}$$

- Step 6: If $\boldsymbol{\alpha}'$ is accepted, update $\boldsymbol{\mu}_{\text{bias},i+1} = \boldsymbol{\mathcal{F}}(\boldsymbol{\alpha}') - \hat{\boldsymbol{\mathcal{F}}}(\boldsymbol{\alpha}')$ and $\boldsymbol{\Sigma}_{\text{bias},i+1} = \boldsymbol{\mu}_{\text{bias},i+1}\boldsymbol{\mu}_{\text{bias},i+1}^T$

The last prior was a Uniform(-2.5,2.5) to provide a noninformative prior that had wide enough support to include all the $\boldsymbol{\alpha}_{\text{obs}}$ components.

The integrated autocorrelation time (IACT) was used to estimate the average number of iterations needed to obtain an independent sample of Q. The IACT ($\hat{\tau}_Q$) is given by

$$\hat{\tau}_Q = 1 + 2\sum_{\tau=1}^{M} \hat{\rho}_Q(\tau)$$

where

$$\hat{\rho}_Q(\tau) = \frac{1}{N-\tau}\sum_{i=1}^{N-\tau} \frac{(Q^{(i)} - \hat{\mu}_Q)(Q^{(i+\tau)} - \hat{\mu}_Q)}{\hat{\sigma}_Q^2}$$

with $\hat{\mu}_Q$ and $\hat{\sigma}_Q^2$ being the sample mean and variance of $\{Q^{(i)}\}_{i=1}^{N}$. Each MCMC algorithm was started from the same place α_0 which was sampled from $\mathcal{N}(\boldsymbol{\alpha}_{\text{obs}}, 0.01 * \boldsymbol{I})$. The IACT was calculated with the last 5,000 Q samples from each chain with a maximum lag of 500.

3 Results

3.1 Neural Networks

The accuracy of the FCNN and CNN for predicting $\boldsymbol{u}$ and the log likelihood is shown in Figures 1 and 8. Both networks perform well at predicting the pressure values at the nine coordinates of interest, with the FCNN being slightly more accurate. This results in a mean squared error for predicting the log likelihood from the FCNN and CNN of 0.195 and 0.447 respectively.

3.2 Gradients

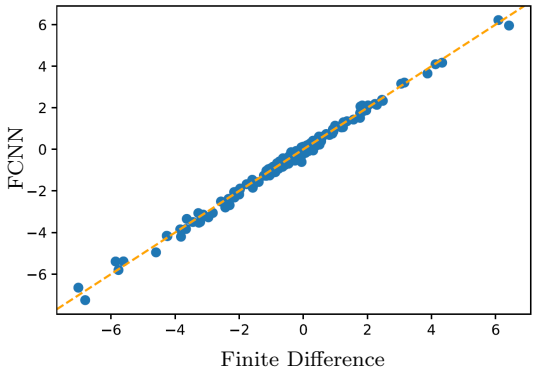
Figure 2 shows the accuracy of the gradients obtained through the FCNN. For each of the three priors, 100 samples of $\boldsymbol{\alpha}$ were selected uniformly from both the HMALA FCNN and MALA FD chains. The gradients of the log posterior using the FCNN are compared to the gradients obtained through central finite difference using a step size of 1e-6. Heatmaps of the error for each $\boldsymbol{\alpha}$ component are shown in Figure 9 and Figure 10 in the appendix. The FCNN does well at estimating the gradients for $\boldsymbol{\alpha}$ samples using the normal prior, with slightly less accuracy for the samples which have the uniform prior.

3.3 Markov chain Monte Carlo

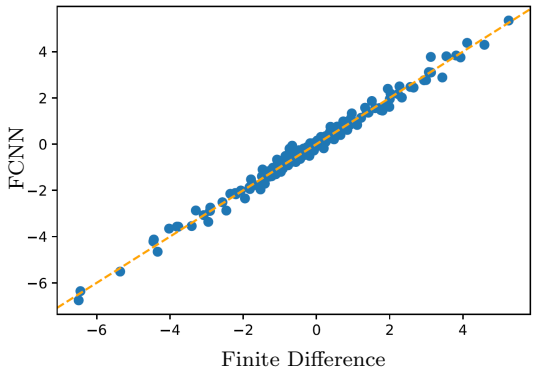
The mean, variance, IACT, and acceptance rate for each of the MCMC methods are included in Table 1. The lowest IACT for all priors was given by pCN with the CNN. Finite difference MALA provided the second lowest IACT for the normal priors but had a high IACT with the uniform prior. The bias-corrected chains required more time to sample due to the likelihood correction. For that step, the gradients of $\boldsymbol{\alpha}$ are needed for each of the 9 points of interest, so the backpropagation step takes longer. Additionally, HMALA and bias-corrected MALA produced estimates for the first $\boldsymbol{\alpha}$ component α_1 and Q which were further from the FD MALA estimates than just using the FCNN for MALA. That is, the FCNN alone without an additional filtering or correction step gave closer estimates to the MALA FD chains. As shown in Figure 11, the posterior density of Q using only the FCNN matches that from FD closely, particularly for the normal prior.

$\mathcal{N}(\boldsymbol{\alpha}_{\text{obs}}, 0.01 * I)$								
	IACT	Accept. rate	$\mathbb{E}[\alpha_1]$	$\mathbb{V}[\alpha_1]$	$\mathbb{E}[Q]$	$\mathbb{V}[Q]$	$\mathbb{E}[\|Q - Q_{\text{obs}}\|]$	Time (min.)
MH PDE	267	0.291	1.627	0.012	4.652	0.248	0.381	7.0
HMALA FCNN	117	0.705	1.618	0.004	4.595	0.073	0.213	7.5
biasMALA CNN	614	0.705	1.623	0.003	4.610	0.046	0.170	90.6
MALA FD	6	0.676	1.623	0.011	4.623	0.195	0.354	NA
$\mathcal{N}(\mathbf{0}, I)$								
	IACT	Accept. rate	$\mathbb{E}[\alpha_1]$	$\mathbb{V}[\alpha_1]$	$\mathbb{E}[Q]$	$\mathbb{V}[Q]$	$\mathbb{E}[\|Q - Q_{\text{obs}}\|]$	Time (min.)
MH PDE	155	0.256	0.269	1.279	2.237	7.357	3.192	6.9
pCN PDE	12	0.266	0.037	1.031	1.641	3.645	3.299	7.1
pcn FCNN	12	0.263	0.038	1.026	1.643	3.619	3.297	7.7
pcn CNN	5	0.269	0.020	0.978	1.571	3.078	3.305	8.4
MALA FCNN	163	0.569	0.038	1.248	1.818	5.497	3.371	5.0
HMALA FCNN	197	0.682	-0.016	0.533	1.243	0.804	3.390	7.2
infMALA FCNN	371	0.653	-0.095	0.325	1.062	0.426	3.553	6.2
biasMALA FCNN	570	0.740	0.042	0.692	1.383	1.101	3.253	37.2
MALA FD	10	0.645	0.065	0.949	1.625	3.436	3.278	282.3
Uniform$(-2.5, 2.5)$								
	IACT	Accept. rate	$\mathbb{E}[\alpha_1]$	$\mathbb{V}[\alpha_1]$	$\mathbb{E}[Q]$	$\mathbb{V}[Q]$	$\mathbb{E}[\|Q - Q_{\text{obs}}\|]$	Time (min.)
MH PDE	573	0.251	0.355	2.226	2.943	8.628	3.081	7.0
MALA FCNN	652	0.580	-0.104	1.208	1.492	2.165	3.277	5.2
HMALA FCNN	519	0.608	0.639	1.832	3.548	10.660	3.053	5.9
biasMALA FCNN	862	0.617	0.853	1.788	2.658	8.631	3.253	44.9
MALA FD	684	0.567	-0.111	1.449	1.822	5.978	3.538	318.9

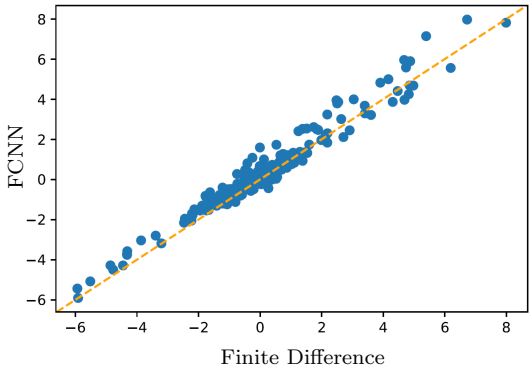
Table 1: Results from MCMC runs using 3 priors. The $\mathbb{E}[\alpha_1]$ and $\mathbb{V}[\alpha_1]$ columns denote the expected value and variance of the first component of the $\boldsymbol{\alpha}$ vector. The time column is only a rough indicator of sampling duration. Due to burnin differences, each chain had 10,000 to 13,000 iterations, and the FD chains were run on a different system and used multiple cores.



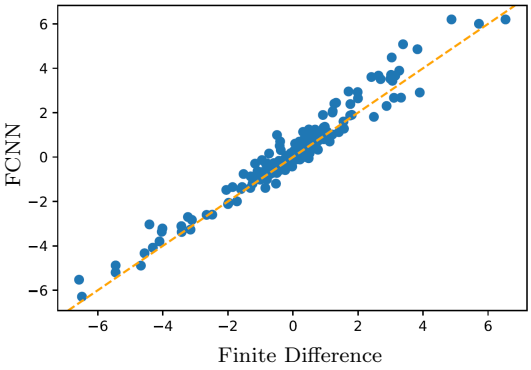
(a) $\boldsymbol{\alpha}$ samples from MALA FCNN with normal prior



(b) $\boldsymbol{\alpha}$ samples from finite difference MALA with normal prior

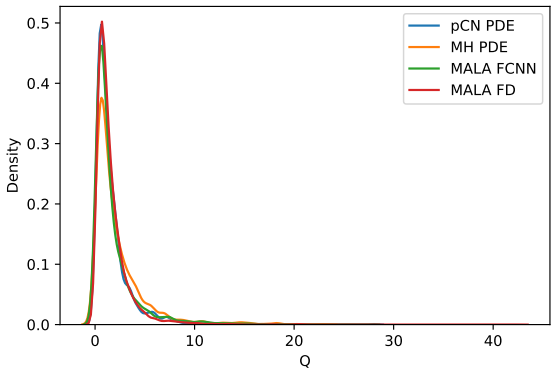


(c) $\boldsymbol{\alpha}$ samples from MALA FCNN with uniform prior

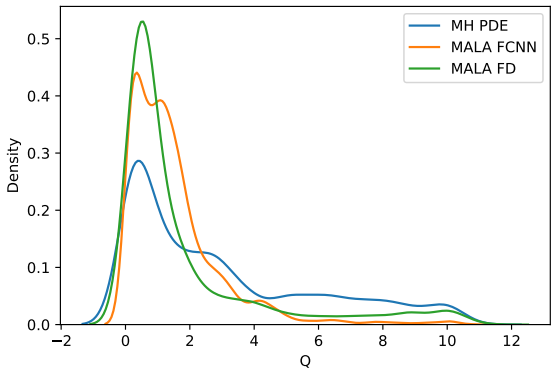


(d) $\boldsymbol{\alpha}$ samples from finite difference MALA with uniform prior

Figure 2: Comparison of gradients obtained through finite difference and DNNs. Each component of the 100 $\boldsymbol{\alpha}$ samples are shown in each plot.



(a) $\mathcal{N}(\mathbf{0}, I)$ prior



(b) Uniform(-2.5,2.5) prior

Figure 3: Posterior density estimates for Q

4 Summary

This project tested the use of a multilayer perceptron and convolutional neural network to obtain the gradients for a few gradient-informed MCMC methods. Both networks did well at predicting the likelihood and providing gradient estimates. It is important to note that the FCNN did not use the KL-expansion and mapped directly from $\boldsymbol{\alpha}$ samples to $\boldsymbol{u}$ estimates. This makes the FCNN useful for problems where the KL-expansion cannot be easily computed.

We used the integrated autocorrelation time to measure the efficiency of the MCMC methods, and MALA chains using finite difference gradients (FD MALA) were obtained to compare with the results from the DNNs. Due to the high cost of calculating finite difference gradients, we found pCN to be the best method since it has a similarly low IACT to FD MALA and is quick to run. In terms of the expected value for Q, MALA FCNN gave similar results to MALA FD at a fraction of the run time. However, MALA FCNN comes with a much higher IACT cost than MALA FD for the normal prior. Additionally, for this problem setup, we saw no benefit to adding a filtering or bias correction step to the MALA algorithms (HMALA and biasMALA), where the DNN makes the proposal which is later checked by the true acceptance ratio calculated from the PDE solver.

It is important to note that this was an easy problem for pCN and MH to handle; there were no issues with convergence for these methods which are not gradient informed. However, given a more difficult target distribution to sample from, there may be more benefit to using MALA. In addition, the pCN algorithm is only applicable to cases where the prior distribution is multivariate normal with mean 0. In the cases were the posterior distribution is more difficult for MCMC to efficiently sample or where the prior distribution does not meet the requirements for pCN, the gradient information provided through the DNNs may be useful.

The success of the DNNs in this project opens up possibilities for other applications of the DNN gradients. They were successful in this project for predicting the posterior distribution and providing sufficiently accurate gradients, and using them in Hamiltonian Monte Carlo might provide efficient parameter space exploration with a low IACT value. Finally, these DNN methods may be useful as a course level approximation as part of a multilevel MCMC framework.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Alexandros Beskos, Mark Girolami, Shiwei Lan, Patrick E Farrell, and Andrew M Stuart. Geometric mcmc for infinite-dimensional inverse problems. *Journal of Computational Physics*, 335:327–351, 2017.

[3] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[4] Simon L Cotter, Gareth O Roberts, Andrew M Stuart, and David White. Mcmc methods for functions: modifying old algorithms to make them faster. 2013.

[5] Paul Dostert, Yalchin Efendiev, Thomas Y Hou, and Wuan Luo. Coarse-gradient langevin algorithms for dynamic data integration and uncertainty quantification. *Journal of computational physics*, 217(1):123–142, 2006.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Mikkel B Lykkegaard, Tim J Dodwell, and David Moxey. Accelerating uncertainty quantification of groundwater flow modelling using a deep neural network proxy. *Computer Methods in Applied Mechanics and Engineering*, 383:113895, 2021.

[8] Joachim Schöberl et al. Netgen/ngsolve. *Software hosted at https://ngsolve.org*, 2017.

Appendix

Data

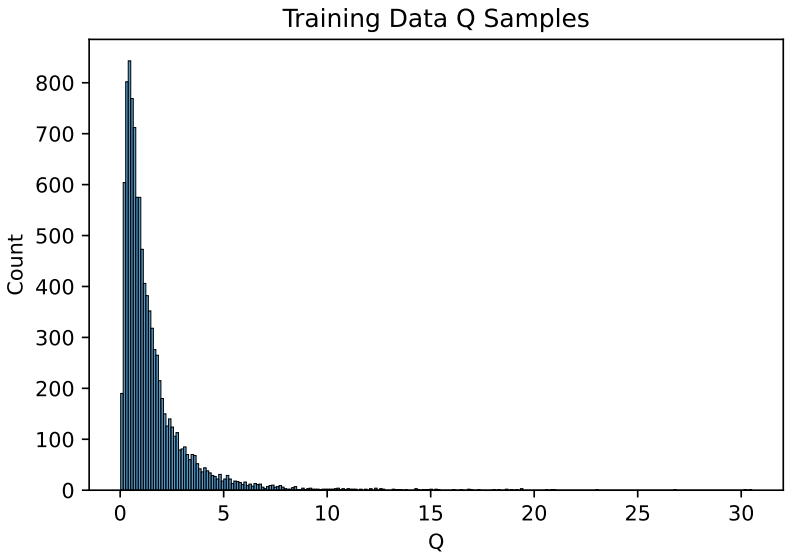


Figure 4: Q samples from generated data

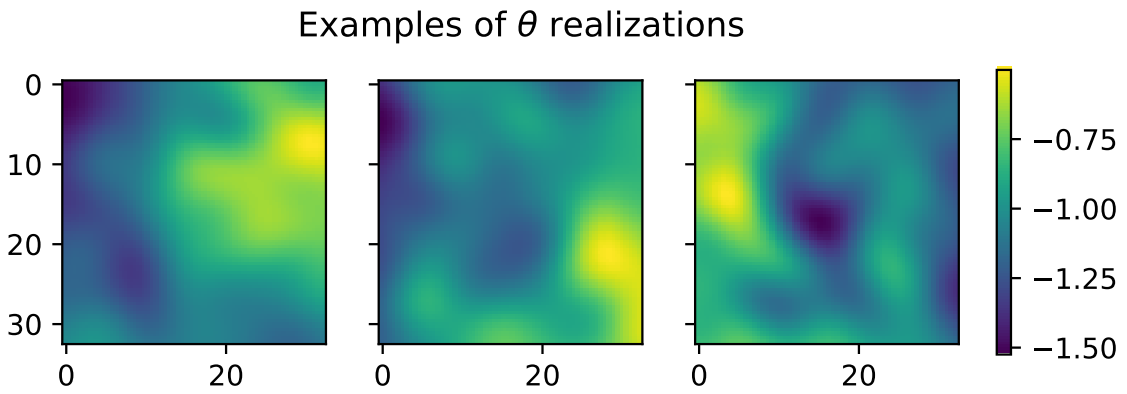


Figure 5: Examples of θ from generated data

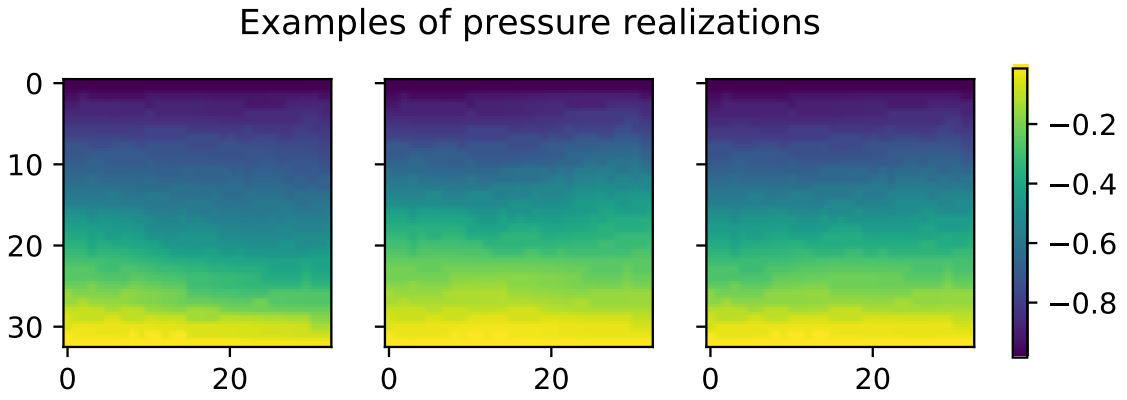


Figure 6: Examples of $\boldsymbol{u}$ fields from generated data

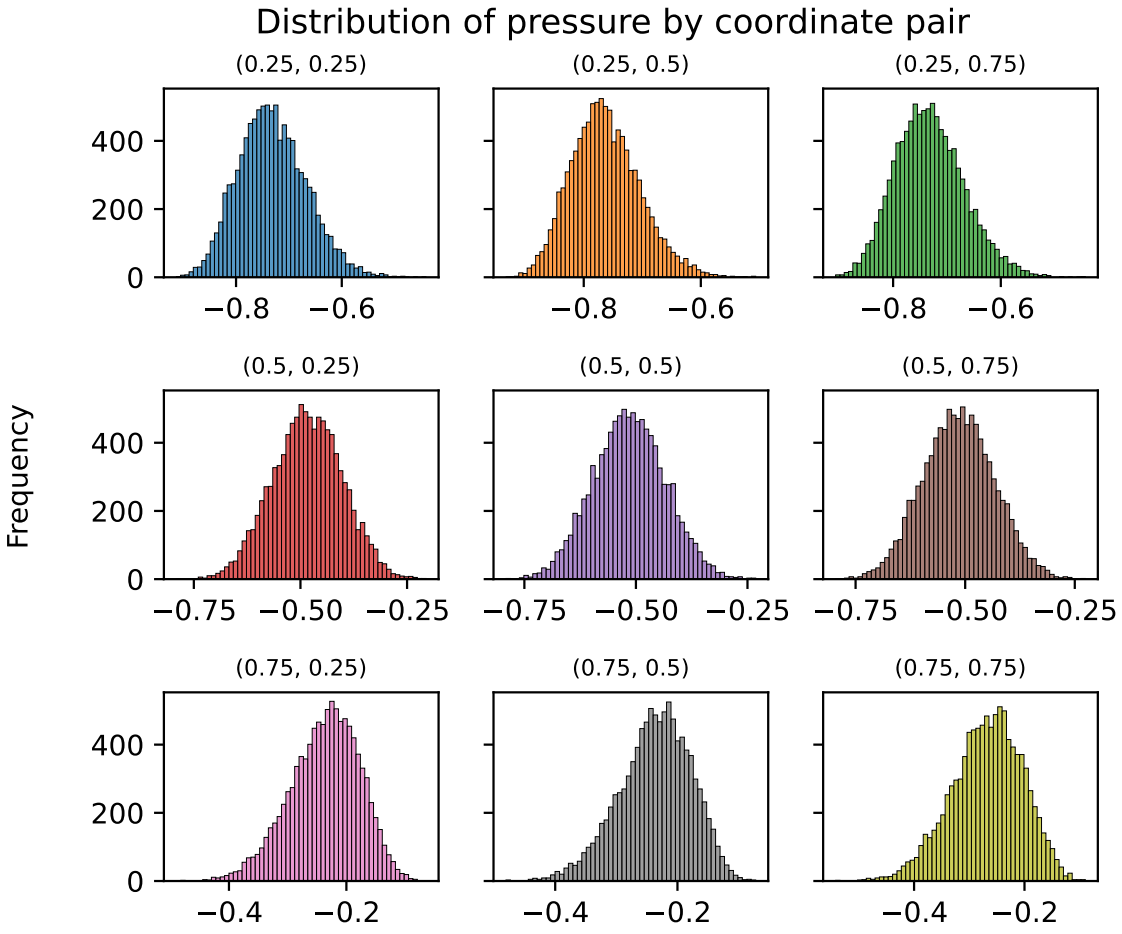
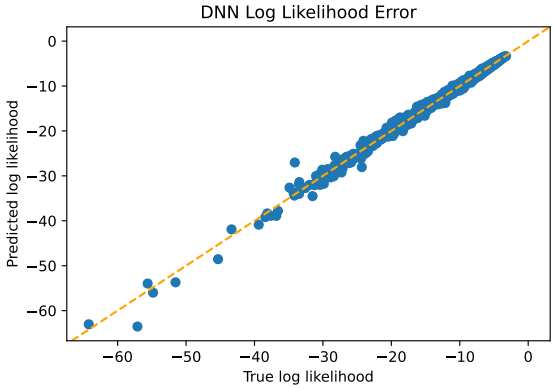
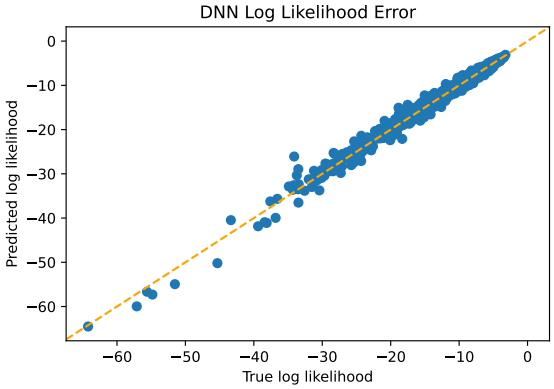


Figure 7: Distribution of pressure values at 9 coordinates of interest from data.

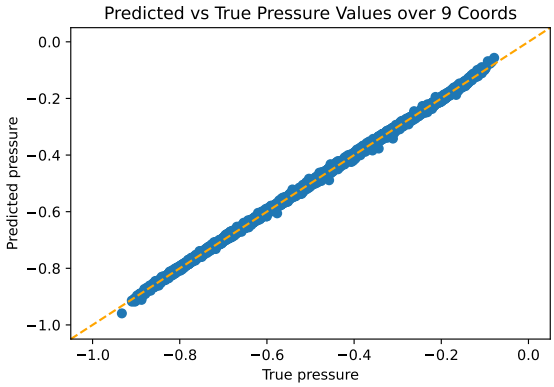
Neural Networks



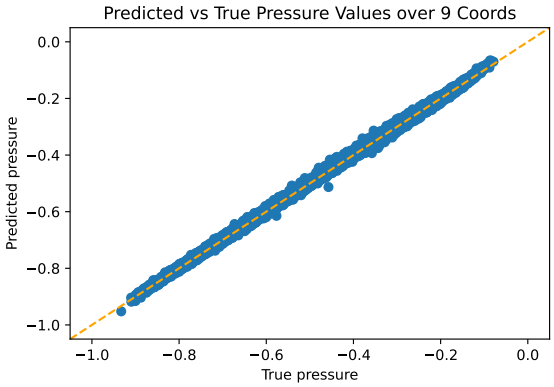
(a) FCNN log likelihood accuracy



(b) CNN log likelihood accuracy



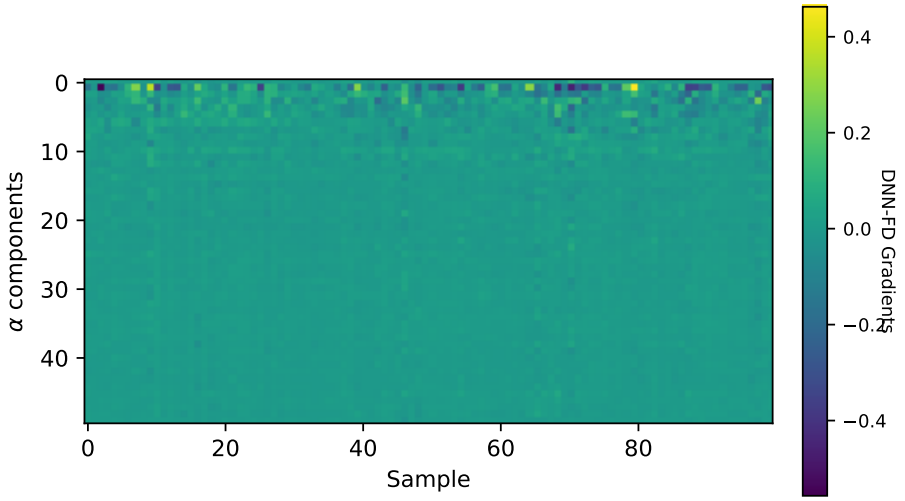
(c) FCNN pressure accuracy



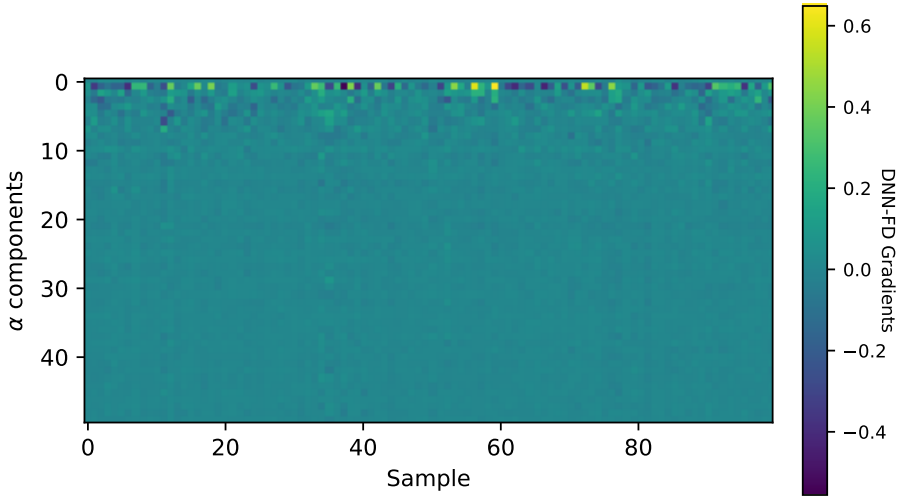
(d) CNN pressure accuracy

Figure 8

Gradients

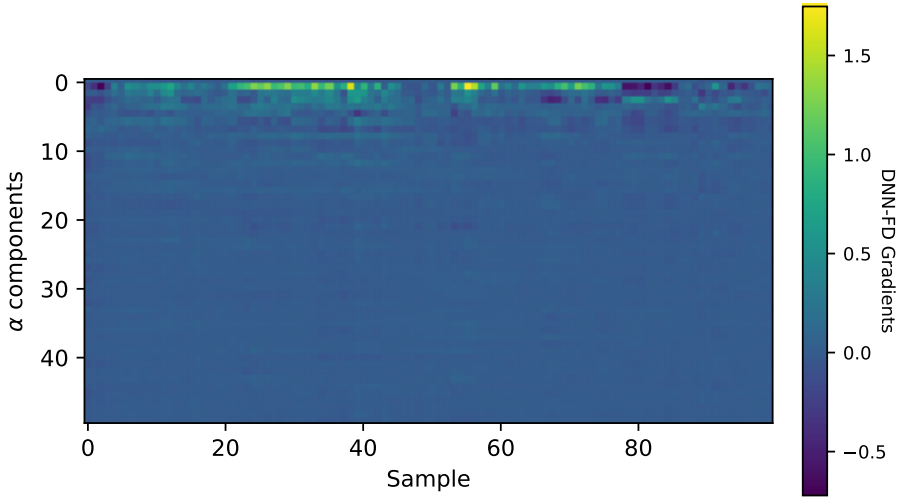


(a) $\boldsymbol{\alpha}$ samples from MALA FCNN with Normal prior

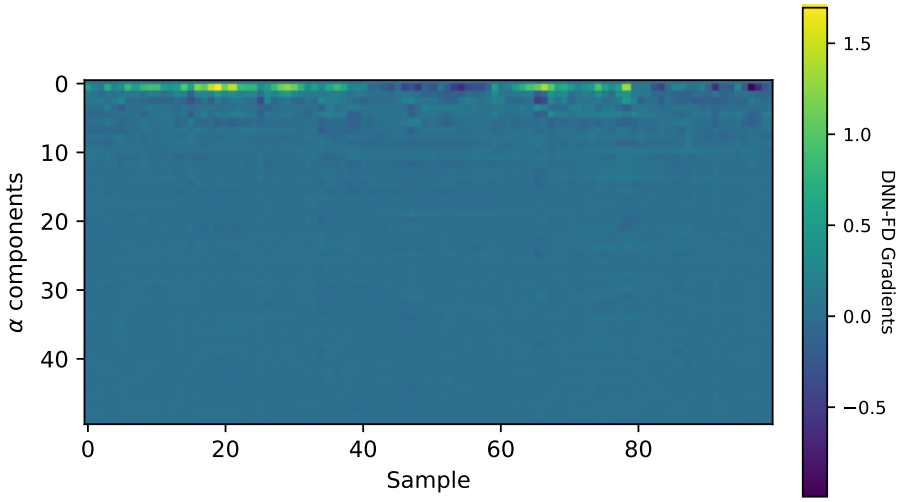


(b) $\boldsymbol{\alpha}$ samples from finite difference MALA with Normal prior

Figure 9: Difference between DNN and finite different gradients for 100 $\boldsymbol{\alpha}$ samples.

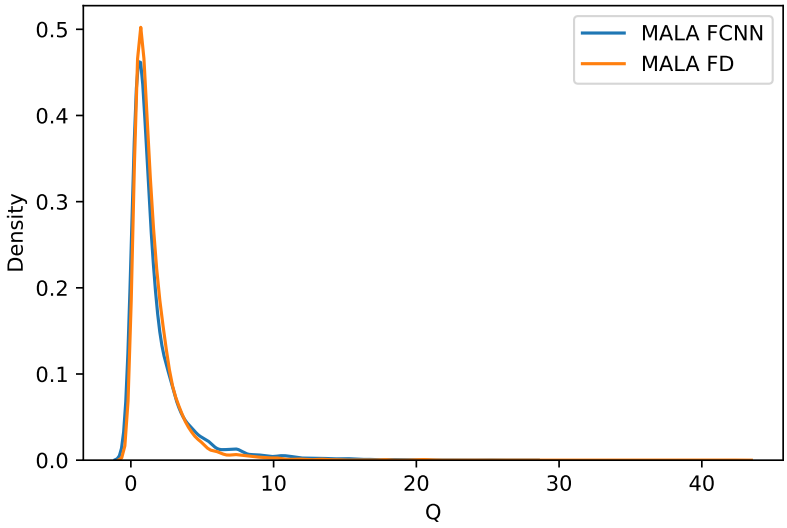


(a) $\boldsymbol{\alpha}$ samples from MALA FCNN with uniform prior

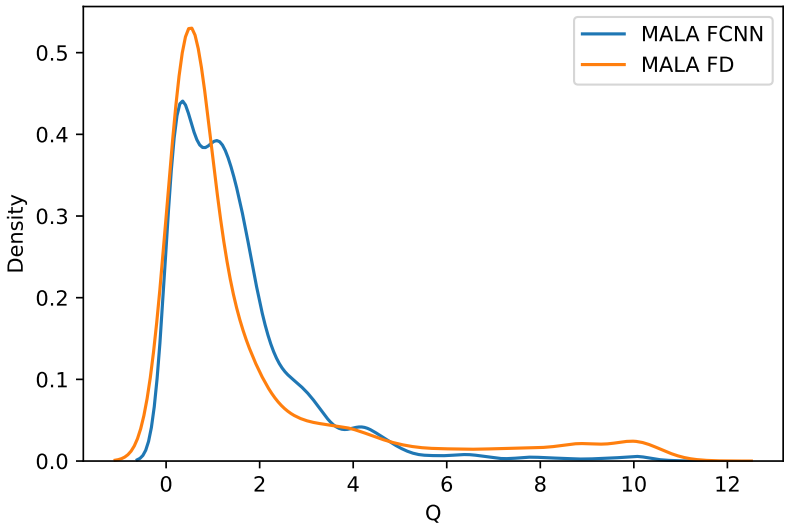


(b) $\boldsymbol{\alpha}$ samples from finite difference MALA with uniform prior

Figure 10: Difference between DNN and finite different gradients for 100 $\boldsymbol{\alpha}$ samples.



(a) $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ prior



(b) Uniform(-2.5,2.5) prior

Figure 11: Posterior density estimates using FD and FCNN MALA