# Uncertainty quantification in physics-informed neural networks 16.940 Final Project

Björn Lütjens (lutjens@mit.edu, 912238433)

## I. Introduction

Decision-makers require uncertainty-expressive climate projections to accurately manage climate risks; for example, to take high-stake decisions for building climate-resilient infrastructure. The quantification of uncertainty in local climate projections, however, requires ensemble forecasts of global climate models which are computationally very expensive. To reduce computational cost, global climate models use parameters to capture small-scale dynamics such as cloud formation that happen below the computationally feasible grid resolution [1]. These parameters, called subgrid parametrizations, are often based on physical intuition or low-dimensional function approximators, which can introduce substantial uncertainties in the global climate projections [1].

Recent works, explores high-dimensional function approximators such as random forests [1] or neural networks [2] to approximate the subgrid parametrizations from computationally expensive models that resolve the small-scale dynamics (e.g., cloud-resolving models). When approximating subgrid-parametrizations, however, it is important to accurately model the uncertainty in the parameter to support downstream risk management decisions.

Zhang et al., 2019 [3] estimates parameters, i.e., solves the inverse problem, by estimating the modes/coefficients of an arbitrary polynomial chaos expansion (PCE) as with neural networks. This paper strives towards the reimplementation of [3][1] and consists of four sections:

- (II) Set up a stochastic diffusion equation and an approximation with PCE
- (III) Estimate the PCE coefficients from manually derived PCE coefficients
- (IV) Estimate the PCE coefficients from measurements of the diffusion parameter
- (V) Discussion of physics-informend neural networks, model uncertainty, and climate modeling

## II. Target: 1D elliptic stochastic diffusion equation

### A. Defining the stochastic diffusion equation

Similar to project 1, we are given the 1D elliptic stochastic diffusion equation

$$\frac{\delta}{\delta x}(k(x,\omega)\frac{\delta u(x,\omega)}{\delta x}) = -s(x), \tag{1}$$

16.940 Numerical Methods for Stochastic Modeling, Human Systems Laboratory, Massachusetts Institute of Technology, 77 Mass. Ave., Cambridge, MA, USA {lutjens}@mit.edu

[1]no source code was found online

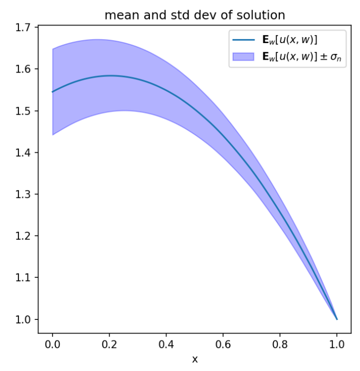


Fig. 1: Solution to the 1D elliptic stochastic diffusion equation

with location, $x \in D = [0,1]$, noise, $\omega \in \Omega$, solution, $u : D \times \Omega \rightarrow \mathbb{R}$, diffusivity, $k : D \times \Omega \rightarrow \mathbb{R}$, and source, $s(x) = 5$. The solution of the stochastic diffusion equation is displayed in Fig. 1. The stochastic diffusion equation has a deterministic Dirichlet boundary condition at $x = 1$

$$u(x = 1, \omega) = u_r = 1, \tag{2}$$

and a random Neumann condition at $x = 0$:

$$k(x,\omega)\frac{\delta u}{\delta x}|_{x=0} = -F(\omega), \tag{3}$$

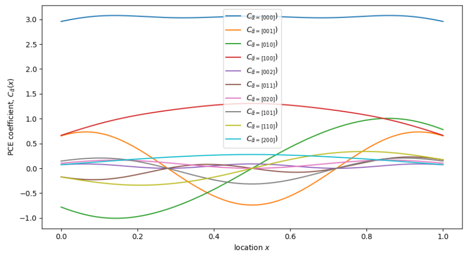
with Flux, $F(\omega) \sim \mathcal{N}(\mu_F = -1.0, \sigma_F^2 = 0.2)$.

### B. Defining the target: polynomial chaos expansion coefficients

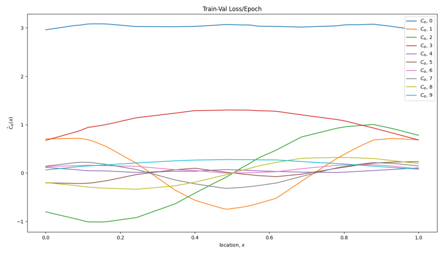
The diffusivity is stochastic assumed to be the unknown subgrid parameterization and will be approximated by an approximate Bayesian neural network. The diffusivity is assumed to follow a Gaussian Process (GP), $Y(x,\omega)$, with $k(x,\omega) = \exp(Y(x,\omega))$. The GP is defined by mean, $\mu_Y = 1.0$, correlation length, $L = 0.3$, variance, $\sigma_Y^2 = 0.3$, exponent, $p_{GP} = 1.0$, and a covariance kernel that is similar to the non-smooth ornstein-uhlenback kernel:

$$C(x_1, x_2) = \sigma_Y^2 \exp(-\frac{1}{p_{GP}}(\frac{|x_1 - x_2|}{L})^p). \tag{4}$$

The Gaussian Process is further disentangled into deterministic spatial domain and stochastic domain via a polynomial chaos expansion (PCE), similar to project 2. The PCE approximates the diffusivity as a linear combination of the PCE coefficients, $C_{\vec{\alpha}}(x)$, and the a set of polynomials,



(a) Target PCE coefficients with $n = 3$ over $x$.



(b) Learned PCE coefficients.

Fig. 2: The learned PCE coefficients (bot) match the target coefficients (top) with MSE$\approx$ 0.0007, but are less smooth.

$\Psi_{\vec\alpha}(\xi_1, ...\xi_n)$. The PCE is then calculated as follows:

$$Y(x) = \sum_{\alpha_1=0}^{\infty}\sum_{\alpha_2=0}^{\infty}...\sum_{\alpha_n=0}^{\infty} C_{(\alpha_1,...,\alpha_n)}(x)\Psi_{(\alpha_1,..\alpha_n)}(\xi_1,...,\xi_n)$$
$$\approx \hat{Y}(x) = \sum_{\vec\alpha\in\{||\vec\alpha||_1\leq n\}} C_{\vec\alpha}(x)\Psi_{\vec\alpha}(\xi_1,...,\xi_n), \tag{5}$$

The set of polynomials is a set of multivariate orthogonal Gaussian-Hermite polynomials,

$$\begin{aligned}\Psi_{\vec\alpha}(\xi_1,...,\xi_n) &= \Pi_{i=1}^n \psi_{\alpha_i}^{(i)}(\xi_i),\\ &= \Pi_{i=1}^n \mathrm{He}_{\alpha_i}^{(i)}(\xi_i), \\ \text{with } \xi_i &\sim \mathcal{N}(0,1).\end{aligned} \tag{6}$$

Each polynomial, $\mathrm{He}_{\alpha_i}(\xi_i)$, has the polynomial degree, $\alpha_i$, given by the total-degree multi-index set, $A = \{\vec\alpha \in \mathbb{N}_0^n : ||\vec\alpha||_1 = \sum_{j=1}^n \alpha_j \leq n\}$. Throughout this write-up the maximum polynomial degree is $n = 3$. Computing the PCE coefficients, $C_{\vec\alpha}(x)$, is typically computationally expensive. The ground-truth coefficients are shown in Fig. 2a and have been computed by approximating the Gaussian Process by a Karhunen-Lóeve-expansion and computing:

$$C_{\vec\alpha}(x_i) = \frac{< \exp(Y(x_i)), \Psi_{\vec\alpha}(\xi_1,...,\xi_n) >}{||\Psi_{\vec\alpha}(\xi)||^2_{L^2_{\mu_\xi}}}, \tag{7}$$

The nominator has been calculated as:

$$< \exp(Y(x_i)), \Psi_{\vec\alpha}(\xi_1,...,\xi_n) >, \tag{8}$$
$$\approx < \exp(\mu_Y + \sum_{j=1}^n \sqrt{\lambda_j}\phi_j(x_i)\xi_j), \Psi_{\vec\alpha}(\xi_1,...,\xi_n) >, \tag{9}$$
$$= \int_{-\infty}^{\infty} \exp(\mu_Y + \sum_{j=1}^n \sqrt{\lambda_j}\phi_j(x_i)\xi_j)$$
$$\Pi_{j=1}^n \mathrm{He}_{\alpha_j}(\xi_j)p(\xi_1,...,\xi_n)d\xi_1...d\xi_n, \tag{10}$$
$$= \exp(\mu_Y)\Pi_{j=1}^n \int_{-\infty}^{\infty} \exp(\sqrt{\lambda_j}\phi_j(x_i)\xi_j)\mathrm{He}_{\alpha_j}(\xi_j)p(\xi_j)d\xi_j, \tag{11}$$
$$\approx \exp(\mu_Y)\Pi_{j=1}^n \sum_{k=0}^{n_{\text{quad}}} \exp(\sqrt{\lambda_j}\phi_j(x_i)\zeta_k)\mathrm{He}_{\alpha_j}(\zeta_k)\omega_k \tag{12}$$

, where (8) uses the KL-expansion, (9) expands the norm, (10) assumes that $\xi_j$ are independent, (11) uses the Gauss-Hermite quadrature rule at $n_{quad} = n^2$ quadrature points, $\zeta_k$, with quadrature weights, $\omega_k$, for integration.

The denominator has been calculated as,

$$||\Psi_{\vec\alpha}(\xi)||^2_{L^2_{\mu_{\vec\xi}}},$$
$$=< \Psi_{\vec\alpha}(\xi_1,...,\xi_n), \Psi_{\vec\alpha}(\xi_1,...,\xi_n) >, \tag{13}$$
$$= \int_{-\infty}^{\infty} \Pi_{i=1}^n\ \psi_{\alpha_i}(\xi_i)\Pi_{j=1}^n\ \ \psi_{\alpha_j}(\xi_j)\ p_{\xi_1,...\xi_n}(\xi_1,...,\xi_n)d\xi_1...d\xi_n, \tag{14}$$
$$= \int_{-\infty}^{\infty} \Pi_{j=1}^n(\psi_{\alpha_j}(\xi_j)\psi_{\alpha_j}(\xi_j)\delta_{jj})\ p_{\xi_1,...\xi_n}(\xi_1,...,\xi_n)d\xi_1...d\xi_n, \tag{15}$$
$$= \Pi_{j=1}^n \int_{-\infty}^{\infty} \psi_{\alpha_j}(\xi_j)\psi_{\alpha_j}(\xi_j)\ p_{\xi_j}(\xi_j)d\xi_j \tag{16}$$
$$= \sqrt{2\pi}\Pi_{j=1}^n \alpha_j!, \tag{17}$$

where (13) expands the norm, (14) assumes that $\psi_j$ are orthonormal, assumes that $\xi_j$ are independent, (16) assumes that $\psi_j$ is Hermite and $\xi_j$ Gaussian.

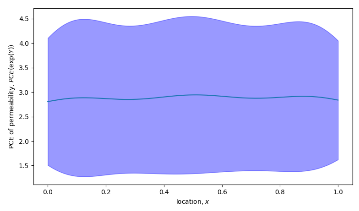
## III. Learning PCE coefficients from analytically derived targets

The computation of PCE coefficients can be computationally expensive, arduous, and error-prone. Approximating the polynomial coefficients from data could alleviate this difficulty. Hence, the following experiments to estimate the PCE coefficients from data.

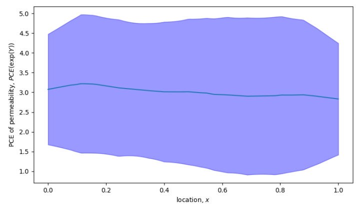
### A. Approach

First, a neural network, $NN_{C_{\vec\alpha}}(x) = \hat{C}_{\vec\alpha}(x) : [x_{\min}, x_{\max}] \to \mathbb{R}^{|A|}$, is trained to approximate PCE coefficients while using the analytically derived PCE coefficients, $C_{\vec\alpha}(x)$, as target. The loss function for parameter, $C$, at episode, $e$, and batch, $b$, is defined as:

$$L_{C,e}(x_b) = \frac{1}{B}\sum_{b=1}^{B}||\sum_{\vec\alpha\in A} C_{\vec\alpha}(x_b) - \hat{C}_{\vec\alpha}(x_b)||_2^2 \tag{18}$$



(a) Estimated diffusion from target PCE coefficients.



(b) Estimated diffusion from learned PCE coefficients.

Fig. 3: Estimated diffusion, $k = \exp(Y)$, from target (top) and learned (bot) PCE coefficients.

### B. Results

Figure 2b shows the learned PCE coefficients and Fig. 3b shows corresponding diffusion, $k = exp(Y)$. The learned PCE coefficients match the target PCE coefficients very well, quantified by a low $\text{MSE}_{C_{\vec{\alpha}}} = \frac{1}{E}\sum_e^E L_{C,e} \approx 0.0007$. The learned coefficients are less smooth, but based on further experiments increasing the network size smoothes the function (not shown). This good performance of the neural network is expected, because the PCE coefficients are a deterministic target and neural networks are a universal function approximator.

The neural network, $NN_{C_{\vec{\alpha}}}(x)$ has a fully-connected, 6-layer, $1\times16\times32\times64\times64\times32\times16\times|A|$-unit network, trained with ADAM with learning rate $0.001$, $\beta = [0.9, 0.999]$, batch size 10, number of epochs 30, with loss function, $\text{MSE}_{C_{\vec{\alpha}}}$, implemented in the automatic differentiation library, pyTorch.

## IV. Learning PCE coefficients from diffusion data

The previous training set-up still requires the estimation of the PCE coefficients via computationally expensive calculation. Thus, we are assuming now that measurements of the diffusion parameter, $k = \exp(Y)$, exist and infer PCE coefficients from this data. The loss function for parameter, $k$, per episode, $e$, then becomes:

$$\begin{aligned} L_{k,e} &= \frac{1}{B}\sum_{b=1}^{B} ||k_b - \hat{k}_{\vec{\alpha}}||_2^2 \\ &= \frac{1}{B}\sum_{b=1}^{B} ||k_b - \exp(\sum_{\alpha\in A} \hat{C}_{\vec{\alpha}}(x_b)\Psi_{\vec{\alpha}}(\xi_1, ..., \xi_n))||_2^2 \end{aligned} \tag{19}$$

The deterministic neural network approximates the correct mean, but not the variance, as shown in Fig. 4 (bot). The poor fit is quantified by the low $\text{MSE}_k = \frac{1}{E}\sum_e^E L_{k,e} \approx 2.4$. The low variance can due to various reasons:

First, the stochasticity in predictions is averaged out during calculation of the loss function. One could overcome this effect by using the same realizations of the random variables $\xi_i$ for generation of the train and test dataset. However, one could not apply this set-up to real-world measurements.

Second, the neural network approximates all PCE coefficients as zero except for the coefficient of the zero-th order polynomials, $C_{[0,0,0]}$, shown in Fig. 4 . Because the zero-th order polynomials are constant, $\text{He}_0(\xi) = 1\forall\xi$, there cannot be variance in the generated diffusion, $k$. However, further experiments have shown that removing the zero-th order from the set of possible polynomials does not enable stochastic predictions.

I would have expected this experiment to estimate the stochasticity, because the experiment in [**?**] is very similar. Greater inspection needs to done to identify how they overcame the issue of a deterministic forecast.

Also, the training set-up removed spatial dependence due to the computational expense. The spatial dependence was removed by assuming that each sample-pair, $[x_b, k_b]$, in the batch of size, $B$, is independent. The spatial dependence could be reintroduced by extending the neural network domain to the full input domain $NN_{C_{\vec{\alpha}}}(x) : \mathbb{R}^{n_{\text{grid}}} \rightarrow \mathbb{R}^{n_{\text{grid}}\times|A|}$. Expanding the domain requires a significantly larger neural network, which was computationally infeasible during this project. However, deep neural networks excel in areas of high-dimensional input spaces, e.g., high-resolution image-to-image-translation [4] and I would expect the neural networks to be computationally feasible and accurate. For the task, so far, simpler regression models such as multivariate linear regression, SVM, or random forests would have probably sufficed, but a subgoal of the write-up was to show feasibility of scalable neural networks.

The hyperparameters in this set-up where the same as in Section III, except for batch size, $B = 1000$, and number of epochs, $E = 100$. The training and test set, $k$, contained each $1000$ samples.

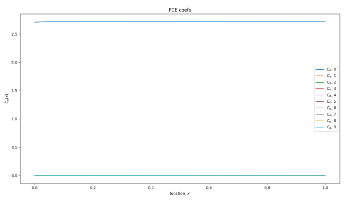
### *Learning PCE coefficients from diffusion measurements*

Training the neural network to approximate the diffusion enables us to incorporate the measurements, $k_{\text{true}}$, as given in project 3. As expected, the resulting curve closely matches the mean, but not the variance, as shown in Fig. 5

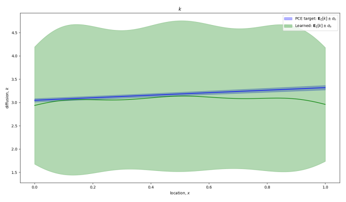
## V. Future work

### A. Physics-Informed Neural Networks

The current work aims to approximate uncertainty by estimating the PCE coefficients from measurements of the diffusion, $k$. In practice, we might not be able to measure, $k$, and might want to infer PCE coefficients from measurements of the solution, $u$. In this case, the loss function per training



(a) Learned PCE coefficients.



(b) Estimated diffusion from learned PCE coefficients.

Fig. 4: Estimated PCE coefficients (top) and diffusion, $k = \exp(Y)$ (bot) when using only diffusion, $k$, as target. The neural network approximates the correct mean $\hat{\mu}_k = \exp(1)$, but not the variance, $\hat{\sigma}_k \approx 0.005$ (note the small y-scale in bottom plot). All coefficients are zero except for the coefficient of zero-th order polynomials, $C_{[0,0,0]}$.

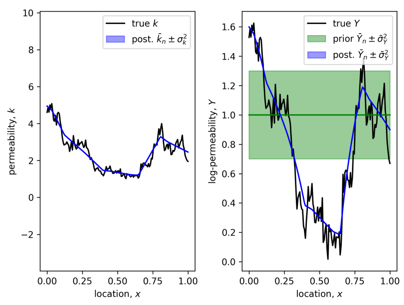


Fig. 5: Estimated diffusion, $k = \exp(Y)$ when using diffusion measurements, $k_{\text{true}}$ (black), as target. The neural network (blue) approximates the correct mean, but not the variance. The prior, $k$, (green) is shown but not used.

sample, $b$, becomes:

$$
\begin{aligned}
L_{u,e,b} &= ||u_{e,b} - \hat{u}||_2^2 \\
&= ||u_{e,b} - \text{ODEsolve}[\frac{\delta}{\delta x}(\hat{k}(x,\vec{\xi})\frac{\delta u(x,\vec{\xi})}{\delta x})]||_2^2 \qquad (20)\\
&\text{with} \\
\hat{k}(x,\vec{\xi}) &= ||(\exp(\sum_{\vec{\alpha}\in A}\hat{C}_{\vec{\alpha}}(x_b)\Psi_{\vec{\alpha}}(\xi_1,...,\xi_n))]||_2^2
\end{aligned}
$$

Including the solver of ordinary differential equations, ODEsolve, in the loss function, however, introduces the problem of high memory complexity and numerical instability:

**Memory complexity.** The loss needs to backpropagated through ODEsolve to compute the gradient wrt. the loss of each network weight, $\theta$. In Section IV this was possible by reverse-mode automatic differentiation [5]. Automatic differentiation is constant in time over the size of input features, which made is successful for training very large neural networks. However, the time and memory scales linearly with the number of operations (as the symbolic derivative of each elementary operation is stored). Numerical differential equation solvers can consist of many steps, which makes the computation of gradients computationally expensive. Recent works achieve a drastic reduction in computational complexity by leveraging the implicit function theorem [6].

**Numerical instability.** There currently is no guarantee that ODEsolve finds a solution, $u$, for every diffusion parameter that the neural network estimates. More work needs to be done in bounding the neural network output to feasible parameters.

## B. Model Uncertainty

Neural networks can fail to extrapolate beyond the training data [7]. It might be possible to identify poor predictive performance via high predictive uncertainty estimates, but deep neural networks do not readily quantify predictive uncertainty [7], [8]. Hence, neural network-based subgrid parametrizations might not be trustworthy due to possible errors in the prediction and the lack of the respective uncertainty.

Luckily, [3] proposes the quantification of model uncertainty in neural networks via an approximation of inference in deep Gaussian Processes, called MC-Dropout [9]. A follow-up work, however, showed that uncertainty estimates from Bayesian Neural Networks (BNNs), trained via HMC, are significantly more accurate than the MC-dropout-based uncertainty estimates [10]. Unfortunately, BNNs are in comparison to MC-dropout uncertainty less scalable to very large neural networks. Future work, will investigate accurate and scalable estimates of model uncertainty, such as deep ensembles which trains an ensemble of deterministic neural networks on overlapping samples of the training dataset [11].

The first step in evaluating model uncertainty would be to measure the PCE coefficients at few locations, $x$, and then show high/low variance in between/on the sampling locations, respectively. In the next experiment one could sample stochastic, $k$, with high/low variance and evaluate if the predicted model uncertainty is higher/lower.

## C. Climate Modeling

The current work dealt with a simple 1D elliptic stochastic diffusion equation. Future work, will investigate more complicated partial differential equations. For example, a stochastic advection-diffusion equation, or the system of Rayleigh-Benard convection equations [12].

## ACKNOWLEDGMENT

The author is very grateful for the preparation and lecture of the material by Prof. Youssef Marzouk and continuous help and feedback in office hours by the teaching assistants, Fengyi Lu, Michael Brennan, and Andrea Scarinci.

## REFERENCES

[1] J. Yuval and P. A. OGorman, "Stable machine-learning parameterization of subgrid processes for climate modeling at a range of resolutions," *Nature Communications*, vol. 11, 2020.

[2] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. Ramadhan, "Universal differential equations for scientific machine learning," *ArXiv*, vol. abs/2001.04385, 2020.

[3] D. Zhang, L. Lu, L. Guo, and G. Karniadakis, "Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems," *J. Comput. Phys.*, vol. 397, 2019.

[4] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CVPR*, 2017.

[5] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 5595–5637, 2017.

[6] X. Li, T.-K. L. Wong, R. T. Q. Chen, and D. Duvenaud, "Scalable gradients for stochastic differential equations," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 108. PMLR, 2020, pp. 3870–3882.

[7] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," p. arXiv:1606.06565, Jun. 2016.

[8] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, University of Cambridge, Cambridge, 9 2016.

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

[10] G. E. K. Liu Yanga, Xuhui Menga, "B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data," *Journal of Computational Physics*, vol. 425, January 2021.

[11] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017, pp. 6402–6413.

[12] K. A. Emanuel, "Rayleigh-benard convection," in *Atmospheric convection*. Oxford University press., 1995, p. pp. 580.