nature computational science

Article

https://doi.org/10.1038/s43588-022-00376-0

Discovering and forecasting extreme events via active learning in neural operators

In the format provided by the authors and unedited

Supplementary Algorithm 1 Sequential search for active learning/training of DNOs and GPs .

Input: Number of iterations: n_{iter}
Initialize: Train GP/DNO on initial dataset of input-output pairs
GP: $\mathcal{D}_0 = \{\mathbf{x}_i, y_i = G(\mathbf{x}_i)\}_{i=1}^{n_{\text{init}}}$ DNO: $\mathcal{D}_0 = \{\mathbf{x}_i, y_i = G(\mathbf{x}_{u,i}\mathbf{\Phi}(x_1, ...x_m), \mathbf{x}_{z,i})\}_{i=1}^{n_{\text{init}}}$
for $n = 1$ **to** n_{iter} **do**
  Select next sample $\mathbf{x}_n$ by maximizing* acquisition function $a(\mathbf{x})$:
    *Maximization using Monte Carlo, See Section **Monte Carlo Optimization of Acquisition Functions** and Supplementary Information Section 7

$$\mathbf{x}_n = \arg\max_{\mathbf{x}\in\mathcal{X}} \ (a(\mathbf{x}; y), \mathcal{D}_{n-1})$$

  Evaluate objective function at $\boldsymbol{x}_n$ and record y_n
  Augment dataset: $\mathcal{D}_n = \mathcal{D}_{n-1} \cup \mathbf{x}_n, y_n$
  Retrain GP/DNO.
end for
return Final GP/DNO Model

Supplementary Algorithm 2 Sequential selection of samples for batch sampling.

Evaluate acquisition function for n_q query points.
for Acquisition samples smaller than batch size $n_a < n_b$
  Choose the maximum score, $\max(a) = a(\mathbf{x}_c)$, from n_q points.
  Augment $\mathbf{x}_a$ with chosen point, $\mathbf{x}_c$.
  Compute the distances, r, between the chosen point and the remaining query points.
  Eliminate all samples from n_q where $r < r_{\min}$.
end for
return samples for the next experiment: $\mathbf{x}_a$

1 SIR Pandemic Model

We implement a simple Susceptible, Infected, Recovered (SIR) model proposed by [1] and reintroduced by [2],

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\beta IS + \delta R \tag{1}$$

$$\frac{\mathrm{d}I}{\mathrm{d}t} = \beta IS - \gamma I \tag{2}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \gamma I - \delta R, \tag{3}$$

where δ is the rate of immunity loss, γ is the recovery rate, and β is the infection rate. Here we take $\delta = 0$ and $\gamma = 0.1$ and adjust β from a scalar to a stochastic infection rate, $\beta(t)$, defined as

$$\beta(t) = \beta_0(\mathbf{x}\mathbf{\Phi}(t) + \phi_0), \tag{4}$$

where $\Phi(t)$ is found via a Karhunen-Loeve expansion of a radial basis kernel with $\sigma_\beta^2 = 0.1$ and length scale $\ell_\beta = 1$, $\beta_0 = 3 \times 10^{-9}$, and $\phi_0 = 2.55$, to ensure all infection rates are non-negative. Initial conditions for the model are $I_0 = 50$ with total population $P = 10^8$, and a step size of 0.1 days is used over 45 days.

2 Dispersive Nonlinear Wave Equation: Majda, McLaughlin, and Tabak Model

The Majda, McLaughlin, and Tabak [3] (MMT) model is a dispersive nonlinear wave equation used for studying 1D wave turbulence. Under appropriate choice of parameters it is associated with the formation of

intermittent events [4]. It is described by

$$iu_t = |\partial_x|^{\alpha} u + \lambda |\partial_x|^{-\beta/4} \left(\left| |\partial_x|^{-\beta/4} u \right|^2 |\partial_x|^{-\beta/4} u \right) + iDu, \tag{5}$$

where u is a complex scalar, exponents α and β are chosen model parameters, and D is a selective Laplacian (described further below). This model gives rise to four-wave resonant interactions that, especially when coupled with large scale forcing and small scale damping, produces a family of spectra revealing both direct and inverse cascades [3, 5]. A realization of the MMT model is shown in Supplementary Figure 1 that demonstrates these complex dynamical properties. Not only does this model provide a rich dynamical response, but also presents a unique utility as a physical model for extreme ocean waves, or rogue waves [6, 7, 8]. Hence, its study is an ideal test bed for both examining the numerical difficulties of predictive models and uncovering insights to physical, real-world applications.

For both ease of computation and for discussion of the terms in the MMT model, we transform the equation into wavenumber space. The pseudodifferential operator $|\partial_x|^{\alpha}$, via the Fourier transform in space becomes: $\widehat{|\partial_x|^{\alpha} u(k)} = |k|^{\alpha} \widehat{u}(k)$ where k is the wavenumber in x. This formulation may be similarly defined on a periodic domain. We choose $\alpha = 1/2$ and $\beta = 0$ as done in [4], reducing equation (5) to

$$\widehat{u}(k)_t = -i|k|^{1/2} \widehat{u}(k) - i\lambda |\widehat{u}(k)|^2 \widehat{u}(k) + \widehat{Du}(k) + f(k), \tag{6}$$

where $f(k)$ is a forcing and $\widehat{Du}(k)$ is a selective Laplacian of the form:

$$\widehat{Du}(k) = \begin{cases} -\left(|k| - k^*\right)^2 \hat{u}(k) & |k| > k^* \\ 0 & |k| \leq k^* \end{cases} \tag{7}$$

where k^* presents the lower bound of wavenumbers subject to dissipation. For the model considered in this study we choose $\lambda = -0.5$, $k^* = 20$, $f(k) = 0$, $dt = 0.001$, and a grid that is periodic between 0-1 with $N_x = 512$ grid points. To propose a stochastic and complex initial condition, $u(x, t = 0)$, we take the complex-valued kernel

$$k(x, x') = \sigma_u^2 e^{i(x - x')} e^{-\frac{(x - x')^2}{\ell_u}}, \tag{8}$$

with $\sigma_u^2 = 1$ and $\ell_u = 0.35$. We then parametrize the stochastic initial conditions by a finite number of random variables, $\mathbf{x}$, using the Karhunen-Loeve expansion of the kernel's correlation matrix,

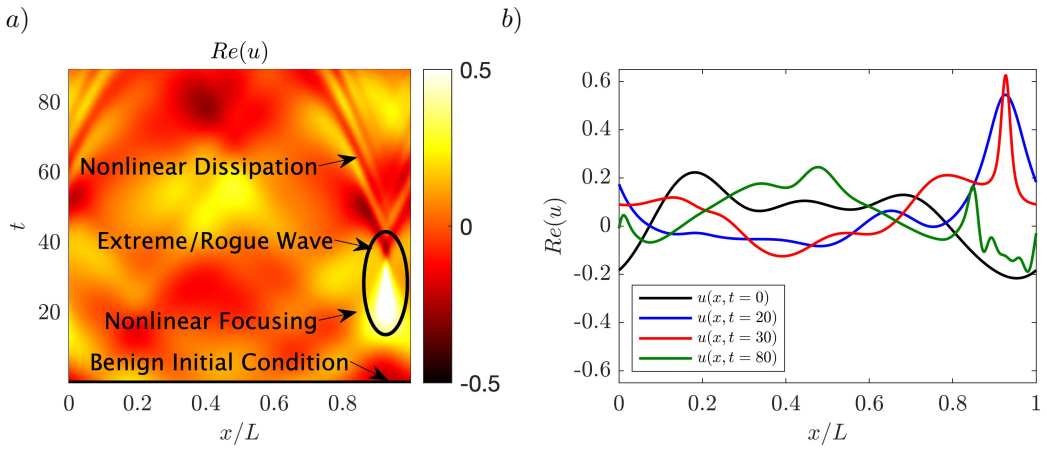
$$u(x, t = 0) \approx \mathbf{x}\mathbf{\Phi}(x), \qquad \forall \qquad x \in [0, 1) \tag{9}$$

where $\mathbf{x} \in \mathbb{C}^m$ is a vector of complex coefficients and both the real and imaginary components of each coefficient are normally distributed with zero mean and diagonal covariance matrix Λ, and $\{\mathbf{\Lambda}, \mathbf{\Phi}(x)\}$ contains the first m eigenpairs of the correlation matrix. This gives the dimension of the parameter space as $2m$ to account for the real and imaginary components of each coefficient. The presented $2D, 4D, 6D, 20D$ results correspond to $m = 1, 2, 3, 10$. For all cases, the random variable x_i is restricted to a domain ranging from -6 to 6, in that 6 standard deviations in each direction from the mean.

3 Efficient Estimation of Structural Fatigue for Ship Design

The LAMP-based problem aims to efficiently estimate the VBM statistics of a unique ship, the ONR topsides flare variant, to inform future design decisions. This real-life industrial application brings significant complexity and numerous challenges. These are documented at length in Guth and Sapsis (2022) [9] and we strongly refer the reader to the above paper as we closely follow their implementation. Briefly, we describe the inputs and outputs below.

Similar to the previous problems, we parametrize a low-dimensional projected subspace of wave episodes using a Karhunen Loève series expansion [10, 11] of the stochastic surface waves, described by a power spectral density such as the JONSWAP spectrum. We next use the proprietary code LAMP (Large Amplitude Motion Program, v4.0.9, May 2019) [12] in order to calculate the forward problem of specific VBM response to a



Supplementary Figure 1: **The MMT emits many nonlinear phenomena in finite time, from focusing to dispersion.** *a*) A realization of the MMT model ($\alpha = 1/2, \beta = 0, \lambda = -0.5$) we discover with DNO-BED, where a benign initial condition leads to an extreme, rogue wave. *b*) Wave height of selected times from *a*).

specific wave episode. A specific wave episode, described by the parametrization $\mathbf{x}$, is mapped onto a VBM time series $y(t)$, whose duration is related to the original interval of the Karhunen Loève expansion.

Finally, as the LAMP code is proprietary and cannot be shared, we perform our active search in a precomputed dataset of 3000 (expensive simulations) LHS samples in $10D$, with each input variable varying from $[-4.5, 4.5]$. This data is provided with the code.

4 Gaussian Process Regression

For low-dimensional problems, Gaussian process (GP) regression [13] is the "gold standard" for Bayesian design. There are several attractive qualities of GPs. They are agnostic to the details of a black-box process (just like neural networks described next) and they clearly quantify both the uncertainty in the model and the uncertainty associated with noise. GPs are also relatively easy to implement and cheap to train.

A Gaussian process $\bar{f}(\mathbf{x})$, where $\mathbf{x}$ is a random variable, is completely specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$. For a dataset $\mathcal{D}$ of input-output pairs $(\{\mathbf{x}, \mathbf{y}\})$ and a Gaussian process with constant mean $m_{\mathbf{0}}$, the random process $\bar{f}(\mathbf{x})$ conditioned on $\mathcal{D}$ follows a normal distribution with posterior mean and variance

$$\mu(\mathbf{x}) = m_0 + k(\mathbf{x}, \mathbf{x})\mathbf{K}^{-1}(\mathbf{y} - m_0) \tag{10}$$

$$\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - k(\mathbf{x}, \mathbf{x})\mathbf{K}^{-1}k(\mathbf{x}, \mathbf{x}) \tag{11}$$

respectively, where $\mathbf{K} = k(\mathbf{x}, \mathbf{x}) + \sigma_\epsilon^2\mathbf{I}$. Equation (10) can predict the value of the surrogate model at any sample $\mathbf{x}$, and (11) to quantify uncertainty in prediction at that sample [13]. Here, we chose the radial-basis-function (RBF) kernel with automatic relevance determination (ARD),

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left[-(\mathbf{x} - \mathbf{x}')^\top \mathbf{L}^{-1}(\mathbf{x} - \mathbf{x}')/2\right], \tag{12}$$

where $\boldsymbol{L}$ is a diagonal matrix containing the lengthscales for each dimension and the GP hyperparameters appearing in the covariance function (σ_f^2 and $\boldsymbol{L}$ in (11) are trained by maximum likelihood estimation).

One setback from the above expression is the inference step in GP regression, where each iteration requires the inversion of the matrix $\mathbf{K}$. Typically performed by Cholesky decomposition, the inversion cost scales as $O(n^3)$, with n being the number of observations [13, 14]. This means that as problems grow to infinite dimensions, inevitably requiring larger datasets, GPs become prohibitively costly. We will show here that as the need for data increases, GPs become much more computationally intensive than our next surrogate model, neural networks.

5 Deep Neural Operators and DeepONet

Unlike GPs, Deep Neural Operators or Deep Neural Networks, do not suffer from data-scaling challenges and are our primary model class for consideration in this manuscript. Deep Neural Networks, when cast as neural operators [15], are specifically well-suited for characterizing infinite dimensional systems, as they may map functional inputs to functional outputs. Although our work is general for any neural network approach, we leverage the architecture proposed by [16] for approximating nonlinear operators: *DeepONet*.

DeepONet seeks approximations of nonlinear operators by constructing two deep neural networks, one representing the input function at a fixed number of sensors and another for encoding the "locations" of evaluation of the output function. The first neural network, termed the "branch", takes input functions, u, observed at discrete sensors, $x_i, i = 1...m$. These input functions can take on several representations, such as initial conditions (i.e. u_0) or forcing functions. The second neural network, termed the "trunk", should be seen as an encoder for inherent qualities of the operator, denoted as z (referred to as y in [16], but changed here for typical active sampling notation). For example, a variable coefficient or exponent in the true nonlinear operator, G, alternatively, and the usual use case for DeepONet, the trunk variable can refer to the evaluation of the operation to an arbitrary point in time and/or space. Together, these networks seek to approximate the nonlinear operation upon u and z as $G(u)(z) = y$, where y will denote the scalar output from the u, z input pair.

Given a set of input-output pairs, $\{[\mathbf{u}, \mathbf{z}], G(\mathbf{u})(\mathbf{z})\}$, DeepONet seeks to minimize the difference between the true operator, $G(u)(z)$, and the dot product between two neural networks $\mathbf{g}(u)$ and $\mathbf{f}(z)$. These network's level of expressivity is governed by the number of neurons (n), the number of layers (l_b, l_t for branch and trunk, respectively), and activation functions. Under sufficient training DeepONet can meet any arbitrary error, ϵ, as

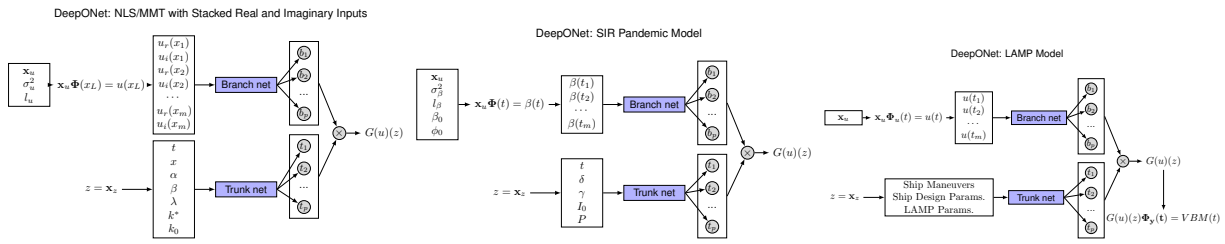
$$| \, G(u)(z) - \langle \underbrace{\mathbf{g}\left(u\left(x_1\right), u\left(x_2\right), \cdots, u\left(x_m\right)\right)}_{\text{branch}}, \underbrace{\mathbf{f}(z)}_{\text{trunk}} \rangle | < \epsilon. \tag{13}$$

Thus, DeepONet can, to an arbitrarily small precision, approximate an infinite dimensional nonlinear operator $G(u)(z)$. However, the functions $\mathbf{g}$ and $\mathbf{f}$ are typically trained under the assumption of plentiful data. We are interested in how DeepONet can be optimally trained with the least amount of data to discover and quantify extreme events.

DeepONet is attractive for these tasks as neural nets generalize well and are incredibly fast to evaluate (compared to their experimental/simulation counterparts) for arbitrarily chosen points. However, there is little work on how one optimally selects the best samples to train a neural net. Previously, the approach taken has included creating an appropriate basis for specific operators that are used to train DeepONet [17]. For the problems we are interested in (i.e. rare and extreme events), this appropriate basis is unknown. Therefore, we envision DeepONet will provide a flexible model to learn seen data, and then provide early predictions of where danger and uncertainty lie in the input/output space through DeepONet's parameterization, that we may then query the underlying system to best inform DeepONet.

5.1 Approaches for quantifying uncertainty in Deep Neural Networks

There are several techniques for quantifying uncertainty in neural networks and three categorizations of these techniques we wish to mention: single deterministic networks, Bayesian neural network inference approaches, and ensemble methods (see [18] and [19] for comprehensive reviews of these methods for uncertainty quantification). Unfortunately, a complex array of advantages and disadvantages of each approach provides no clear favorite. Single model, i.e. deterministic, methods [20, 21] use one forward pass of a deterministic network to learn both the mean and variance of a labeled output. Using only one model leads to cheap training and evaluation. Unfortunately, this single opinion of the underlying system results in substantial sensitivity, an unattractive quality for regression problems quantifying physical instabilities/extreme events. Bayesian neural networks [22, 23, 24, 25] encompass a broad variety of stochastic DNNs that combine Bayesian inference theory with the expressiveness, scalability, and predictive performance enjoyed by deep neural networks. Although the supporting theory behind Bayesian methods, as well as empirical results, imparts faith that such models will lead to the greatest chance of success, they do come with a significant disadvantage, complexity. Bayesian neural networks are significantly more complex than standard neural networks and can be exhausting to train, making them difficult to implement. From an academic viewpoint, this can be overcome.



Supplementary Figure 2: **The application of the MMT, SIR, and LAMP operators to DeepONet.**

However, as a study concerned with translating DNNs and active learning to practical engineering systems that solve real-world problems, we opt for using the straight-forward ensemble approach. An approach that seats itself between the single deterministic model and the infinite model representations of Bayesian neural networks.

6 DeepONet Setup

Application of the MMT, SIR, and LAMP models to the DeepONet architecture requires a well-posed distinction between the functions and parameters that belong in the branch and trunk networks. Supplementary Figure 2 provides a visual of the function/parameter delineations and the architecture for each modeling task.

In this work, we provide all input initial conditions as functions to the branch network. Because of the requirement that all inputs to DeepONet take on real values, the MMT setup requires that we split the input function, u, into its real (u_r) and imaginary (u_i) components. It is critical to keep both components, as each contains unique and coupled information that is propagated in the MMT operation (this is not necessary for the real-valued SIR or LAMP model). We then stack these components as a vector by x position/sensor with the imaginary component directly following the real component at each x position. Technically, the ordering of the inputs does not matter, due to the linear nature of the first layer, but the ordering of the inputs must remain consistent. For increased speed, we also reduce the sensor number of x_m sensors from the 512 used for the direct MMT calculation to 128 points for DeepONet (this corresponds to 256 total inputs values because of the real and imaginary components), 125 points for the SIR infection rate, and 1024 points for the LAMP wave episode. We also give the length scale and variance as input parameters, where applicable, as they define the kernel that emits the Karhunen-Loeve expansion functions. The input functions directly recognize adjustments to these parameters.

The trunk contains parameters that are intrinsic to the MMT, SIR, and LAMP operations. For MMT, these comprise both time and space (t, x) and the chosen constants $\alpha, \beta, \lambda, k^*$. Regarding BED, we may assign each of these parameters to an appropriate prior distribution, such as uniform for space and time. In this study, we choose all values to remain constant and although the trunk is straightforward to implement, it is simply a redundant network here.

We provide the hyperparameters and other quantities used for the various MMT cases performed throughout the manuscript in Supplementary Table 1. In addition to the hyperparameters, we use the ReLu activation function, a learning rate of $l_r = 0.001$, and 1000 epochs for each training procedure. For the SIR search, the same hyperparameters are used with 8 branch layers, 1 redundant trunk layer, and 300 neurons. For the LAMP search, the same hyperparameters are used with 5 branch layers, 1 redundant trunk layer, and 200 neurons.

7 Monte Carlo Optimization at $20D$

Here we show that acquisition samples found through Monte Carlo (MC) searches consistently outperform those selected by optimizers using gradient descent algorithms (when reasonably similar computation times are considered). To demonstrate this behavior, we test four methods for finding 50 batched acquisition samples for the $20D$ MMT case at 500, 2500, and 5000 training samples, $N = 2$ ensemble members, and

Case	Neurons	Branch Layers	Ensemble Size	Experiments
MMT: $2-6D$	200	5	10	10
MMT: $8D$ Batching	200	5	10	10
MMT: $8D$ Ensemble, $n \leq 2500$	200	5	2,4,8,16	10
MMT: $8D$ Ensemble, $n > 2500$	200	6	2,4,8,16	10
MMT: $20D$, $n \leq 2500$	200	5	2	25
MMT: $20D$, $n > 2500$	200	6	2	25
LAMP: $10D$	200	5	2	10
SIR: $2D$	300	8	2,8	1

Supplementary Table 1: **Hyperparameters used for all cases.** All trunk layers = 1 due to their redundancy.

Samples	L-BFGS-B	MC: 10^5	MC+L-BFGS-B	MC: 10^6
500	122 ± 7	3.6 ± 0.1	123 ± 5	$\mathbf{27 \pm 1}$
2500	126 ± 6	4.0 ± 0.2	125 ± 5	$\mathbf{26 \pm 2}$
5000	127 ± 8	3.7 ± 0.1	135 ± 7	$\mathbf{27 \pm 3}$

Supplementary Table 2: **MC is faster than built-in optimizers.** Mean compute times ($k = 25$) for batch minimization ($n_b = 50$) of the $20D$ US-LW acquisition function with $\pm$ one standard deviation. Best values are bold.

over 25 independent experiments. The four approaches are listed below, with each case choosing the 50 best samples with Supplementary Algorithm 2:

1. L-BFGS-B algorithm, implemented within the `scipy Python` package, 10^2 random LHS initial points.
2. MC with 10^5 random uniform initial samples (sampled from a uniform distribution for speed)
3. MC with 10^5, best 100 samples (Supplementary Algorithm 2) passed to L-BFGS-B.
4. MC with 10^6 random uniform initial points.

Supplementary Table 2 and Supplementary Table 3 provide the mean computation times and scores for all cases, respectively. The mean computation times ($\pm$ one standard deviation) are clearly faster for MC methods, by 5-fold, compared to any approach with the L-BFGS-B algorithms. We also note that only the first ensemble member is used for computing and querying p_μ for speed, as well as likely better performance.

Supplementary Table 3 provides the mean scores (over all chosen batch samples and experiments, $n = 1250$), as well as the mean best and worst sample for each experiment ($n = 25$). It is critical to note that at these high dimensions the difference between acquisition scores can be *hundreds* of orders of magnitude different (this is a direct result of weights define by the ratio of two PDFs computed over 20 dimensions). To navigate this computational challenge, we take the $\log_{10}$ of the scores and apply a negative sign such that lower scores are optimal (i.e. $-\log_{10}(a(\mathbf{x}))$). Considering the mean of all chosen points, we see that a naive use of the L-BFGS-B optimizer finds scores that are ≈ 50 orders of magnitude worse than the 10^6 MC approach, regardless of the training complexity of the model. The L-BFGS-B optimizer only provides a marginal advantage for finding the best sample in any random instance, however, the standard deviation is 30 orders of magnitude and suggests the approach is terribly inconsistent. Juxtaposing the min and max results, we see that the MC method only varies by approximately 10 orders of magnitude compared to the 80 orders of the L-BFGS-B.

Overall, the MC method provides greater consistency in finding several attractive acquisition points, while also enjoying ease of implementation and better computational efficiency. Of course, if given enough computational resources and time, the L-BFGS-B method permits pinpointing exceptional acquisition points. Approaches combining both MC and L-BFGS-B, where the MC samples are further optimized (as also implemented here) or performed from more LHS points, can either refine or enrich the set of acquisition points.

Samples	L-BFGS-B	MC: 10^5	MC+L-BFGS-B	MC: 10^6
Mean ($k = 1250$)				
500	110 ± 18	70 ± 5	66 ± 6	$\mathbf{60 \pm 4}$
2500	104 ± 16	67 ± 5	63 ± 5	$\mathbf{56 \pm 4}$
5000	110 ± 17	76 ± 4	67 ± 8	$\mathbf{64 \pm 4}$
Min $k = 25$				
500	$\mathbf{42} \pm 32$	57 ± 6	53 ± 16	$52 \pm \mathbf{5}$
2500	$\mathbf{44} \pm 32$	$53 \pm \mathbf{4}$	48 ± 10	47 ± 6
5000	53 ± 21	60 ± 5	$\mathbf{48} \pm 11$	$54 \pm \mathbf{3}$
Max $k = 25$				
500	126 ± 7	73 ± 3	70 ± 3	$\mathbf{63 \pm 3}$
2500	117 ± 6	70 ± 3	65 ± 3	$\mathbf{60 \pm 3}$
5000	125 ± 5	80 ± 2	72 ± 2	$\mathbf{67 \pm 2}$

Supplementary Table 3: **MC optimization provides consistently improved acquisition scores over built-in optimizers.** Top rows: The mean and standard deviation of all 1250 chosen acquisition samples for the four methods on DNOs trained with 500, 2500, and 5000 samples. Middle rows: The mean and standard deviation of the best acquisition samples for the 25 experiments. Bottom rows: The mean and standard deviation of the worst chosen acquisition samples (i.e. the least optimal sample of the batch). All values are $-\log_{10}(a(\mathbf{x}))$. Best values are bold.

We believe the results found here are chiefly because of the non-convexity of the acquisition function. We may recall the highly non-convex behavior of the 2D acquisition fields in figure 2*b*), even with as little as ≈ 10 samples. This non-convex nature emits many local minima that require many initial samples to provide confidence that the chosen optima are nearly global. This makes the task for built in optimizers extremely difficult, especially for high dimensional problems, and lead to optimizations that become easily fooled or stuck. This difficulty is so challenging that the sparse MC sampling of only 10^6 points, an extremely sparse sampling of a $20D$ space, easily outperforms the optimizers.

8 DNO & Likelihood-Weighted Sampling Implementation Tips

While many of these tips are general to neural network implementation, we reiterate them here to assist others in reproducing these results and applying the method to new stochastic problems.

1. The input function to the DNO should be scaled to vary within values of -1 and 1.
2. The quantity of interest, or output, should also be scaled and normalized to values between -1 and 1.
3. Users should ensure that the DNO is indeed fitting the outputs. This is typically observed with final training errors that are multiple orders of magnitude below the initial training error at epoch 0. To adjust, increase training epochs, layers, and/or width of neurons. Otherwise, the scaling above may have not been appropriately performed.
4. While we did not experience problems, the weights, $w(\mathbf{x})$, could be negatively affected by the output PDF, $p_\mu(\mu)$, in pathological cases. Monitoring the output PDF and its impact on the weights is recommended. We also stress that our approach assumes that extreme events are also rare, as this approach brings attention to the tails. If extremes are not rare, they are not contained within the tails.
5. We found consecutive training and evaluation runs of TENSORFLOW with the same kernel resulted in slowing speeds. To combat this, after a round of training, evaluation, and determination of the next acquisition samples are completed, PYTHON is closed and a new instance is loaded for the next step.

References

[1] Kermack, W. O. & McKendrick, A. G. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character* **115**, 700–721 (1927).

[2] Anderson, R. M. & May, R. M. Population biology of infectious diseases: Part I. *Nature* **280**, 361–367 (1979).

[3] Majda, A. J., McLaughlin, D. W. & Tabak, E. G. A one-dimensional model for dispersive wave turbulence. *Journal of Nonlinear Science* **7**, 9–44 (1997).

[4] Cousins, W. & Sapsis, T. P. Quantification and prediction of extreme events in a one-dimensional nonlinear dispersive wave model. *Physica D: Nonlinear Phenomena* **280**, 48–58 (2014).

[5] Cai, D., Majda, A. J., McLaughlin, D. W. & Tabak, E. G. Spectral bifurcations in dispersive wave turbulence. *Proceedings of the National Academy of Sciences* **96**, 14216–14221 (1999).

[6] Zakharov, V. E., Guyenne, P., Pushkarev, A. N. & Dias, F. Wave turbulence in one-dimensional models. *Physica D: Nonlinear Phenomena* **152**, 573–619 (2001).

[7] Zakharov, V. E., Dias, F. & Pushkarev, A. One-dimensional wave turbulence. *Physics Reports* **398**, 1–65 (2004).

[8] Pushkarev, A. & Zakharov, V. E. Quasibreathers in the MMT model. *Physica D: Nonlinear Phenomena* **248**, 55–61 (2013).

[9] Guth, S. & Sapsis, T. P. Wavegroup-based gaussian process regression for ship dynamics: Between the Scylla of slow Karhunen-Loève convergence and the Charybdis of transient features (Submitted 2022 (https://sandlab.mit.edu/Papers/22_OEJ.pdf)).

[10] Karhunen, K. Über lineare Methoden in der Wahrscheinlichkeitsrechnung. *Ann. Acad. Sci. Fennicae Ser. A. I. Math.-Phys.* **1947**, 79 (1947).

[11] Loève, M. *Processus Stochastiques et Mouvement Brownien*, chap. Fonctions aléatoires du second ordre (Gauthier-Villars, 1948).

[12] Lin, W.-M., Zhang, S. & Weems, K. M. Numerical simulations of surface effect ship in waves. In *Proceedings of the 2010 Conference on Grand Challenges in Modeling & Simulation*, 414–421 (Society for Modeling & Simulation International, Vista, CA, 2010).

[13] Rasmussen, C. E. Gaussian processes in machine learning. In *Summer School on Machine Learning*, 63–71 (Springer, 2003).

[14] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P. & De Freitas, N. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE* **104**, 148–175 (2015).

[15] Kovachki, N. *et al.* Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481* (2021).

[16] Lu, L., Jin, P., Pang, G., Zhang, Z. & Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence* **3**, 218–229 (2021).

[17] Lu, L., Jin, P. & Karniadakis, G. E. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193* (2019).

[18] Gawlikowski, J. *et al.* A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342* (2021).

[19] Psaros, A. F., Meng, X., Zou, Z., Guo, L. & Karniadakis, G. E. Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *arXiv preprint arXiv:2201.07766* (2022).

[20] Sensoy, M., Kaplan, L. & Kandemir, M. Evidential deep learning to quantify classification uncertainty. *arXiv preprint arXiv:1806.01768* (2018).

[21] Malinin, A. & Gales, M. Predictive uncertainty estimation via prior networks. *arXiv preprint arXiv:1802.10501* (2018).

[22] Blundell, C., Cornebise, J., Kavukcuoglu, K. & Wierstra, D. Weight uncertainty in neural network. In *International Conference on Machine Learning*, 1613–1622 (PMLR, 2015).

[23] Gal, Y. & Ghahramani, Z. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, 1050–1059 (PMLR, 2016).

[24] Gal, Y., Islam, R. & Ghahramani, Z. Deep Bayesian active learning with image data. In *International Conference on Machine Learning*, 1183–1192 (PMLR, 2017).

[25] Wilson, A. G. & Izmailov, P. Bayesian deep learning and a probabilistic perspective of generalization. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20 (Curran Associates Inc., Red Hook, NY, USA, 2020).