Resampling with neural networks for stochastic parameterization in multiscale systems

Daan Crommelin [a,b,*], Wouter Edeling [a]

[a] Centrum Wiskunde & Informatica, Scientific Computing Group, Science Park 123, 1098 XG Amsterdam, The Netherlands
[b] Korteweg–de Vries Institute for Mathematics, University of Amsterdam, Science Park 105-107, 1098 XG Amsterdam, The Netherlands

ARTICLE INFO

ABSTRACT

In simulations of multiscale dynamical systems, not all relevant processes can be resolved explicitly. Taking the effect of the unresolved processes into account is important, which introduces the need for parameterizations. We present a machine-learning method, used for the conditional resampling of observations or reference data from a fully resolved simulation. It is based on the probabilistic classification of subsets of reference data, conditioned on macroscopic variables. This method is used to formulate a parameterization that is stochastic, taking the uncertainty of the unresolved scales into account. We validate our approach on the Lorenz 96 system, using two different parameter settings which are challenging for parameterization methods.

1. Introduction

For modeling and simulation of multiscale systems, a central problem is how to represent processes with small spatial scales and/or fast timescales. Simulations that include all relevant scales in an explicit way are often computationally too expensive, posing major challenges to the use of numerical models in a variety of disciplines (e.g. physics, chemistry, climate science, biology, engineering). However, for many problems, the model output of interest involves only large-scale model variables, therefore a natural approach to make simulations less expensive is to derive reduced models for the large-scale variables only. The effect of unresolved (small-scale) degrees of freedom on the large-scale variables must be parameterized, in order to obtain a reduced model that forms a closed system.

Constructing a parameterization (or model closure) can be approached in different ways, using e.g. analytical or computational methods [1–4]. Here we focus on data-driven approaches, where data of (the effect of) small-scale processes is used to infer a parameterization (e.g. [5–13]). These data can stem from e.g. fully resolved (high resolution) simulations on a limited space/time domain, or from physical measurements. Data-driven methods can be particularly useful when there is no clear separation between small/fast scales and large/slow scales, so that analytical or computational approaches that rely on such a scale gap do not apply.

The last few years have seen a surge of interest in data-driven approaches, due to the rapid developments in machine learning (ML). In e.g. [14–18] (and many more studies), ML techniques are proposed for parameterizing unresolved processes based on data. Although the specific ML techniques vary, in nearly all proposed methods the parameterization is effectively deterministic. However, uncertainties in subgrid-scale responses are important, and these can be accounted for by formulating parameterizations that are stochastic rather than deterministic [7,19–22]. The relevance of stochastic formulations for reduced models can be understood theoretically from the Mori–Zwanzig formalism for model reduction of dynamical systems [13,23,24].

Using ML methods for stochastic parameterization has hardly been explored yet. A recent exception is [25], where an approach using generative adversarial networks (GANs) is proposed. Here we follow another approach to ML-based stochastic parameterization, by combining neural network-based probabilistic classification with resampling/bootstrapping, building on recent work on parameterization with resampling [11,26,27].

2. Preliminaries

In this study we consider multiscale dynamical systems, represented by a set of coupled nonlinear ordinary differential equations (ODEs) for the time-dependent variables $x(t)$ and $y(t)$:

$$\frac{d}{dt}x = f(x, \sigma) \tag{1a}$$

$$\frac{d}{dt}y = g(x, y) \tag{1b}$$

$$\sigma = \sigma(y) \tag{1c}$$

with $t \in \mathbb{R}_0^+$, $x \in \mathbb{R}^N$, $y \in \mathbb{R}^K$ and $\sigma : \mathbb{R}^K \mapsto \mathbb{R}^N$. The initial conditions are denoted $x(0) = x_0$ and $y(0) = y_0$.

This coupled system can be the result of e.g. spatial discretization of a partial differential equation. In (1), x denotes "macroscopic" variables, representing phenomena with large spatial scales and/or long timescales. The y denotes "microscopic" variables, typically with small or fast scales. Despite these shorthand names, we do not assume strong scale separation between x and y, neither in space nor in time. Often, $\dim(x) =: N \ll K := \dim(y)$. The coupling from "micro" to "macro", represented by $\sigma(y)$, can be as simple as $\sigma(y) = y$, however there are many cases where coupling effectively takes place via a quantity (e.g. a discretized spatial field) with the dimension of x rather than of y. We note that σ is implicitly dependent on x, because y is coupled to x via equation (1b).

As already discussed in the introduction, even though the interest is often only in the behavior of x, one needs to simulate both x and y as they are coupled. This can make numerical simulations very expensive. We aim to reduce the computational cost by replacing σ by a data-inferred approximation or surrogate $\tilde{\sigma}^{\text{surr}}(x)$ in (1a) so that a closed, reduced system is obtained involving only x. In this study, we discuss a data-driven approach in which the surrogate is not only x-dependent but it is also stochastic and has memory. The stochasticity and memory-dependence of the surrogates both follow from the Mori–Zwanzig theory, see e.g. [23,24] for more details.

We focus here on the case where σ enters as an additive term for x, i.e. $dx/dt = f(x) + \sigma$ (we note that the methodology presented here can in principle also be used when σ enters as a multiplicative term, however we do not perform tests on such cases here). Furthermore, we consider the discrete-time version of this system, resulting from using an (explicit) numerical time integration scheme for (1) with time step Δt. We define $x_j := x(t_j)$ and $y_j := y(t_j)$. For simplicity we assume a constant time step so that $t_j = j \Delta t$ and thus x_0 and y_0 as defined here coincide with the initial conditions defined earlier. We denote the discrete-time system by

$$x_{j+1} = F(x_j) + r_j \tag{2a}$$
$$y_{j+1} = G(x_j, y_j) \tag{2b}$$
$$r_j := r(y_j) \tag{2c}$$

Clearly, $j \in \mathbb{N}$ is the time index here. The precise form of F and G depends on the time integration scheme used. For instance, a simple forward Euler scheme would result in $F(x_j) = x_j + \Delta t f(x_j)$, $G(x_j, y_j) = y_j + \Delta t g(x_j, y_j)$ and $r(y_j) = \Delta t \sigma(y_j)$.

The model structure in (2) reflects a modular computational set-up that can be encountered in various applications (see e.g. [28,29]), where different submodels (or model components) are used for macroscopic and microscopic processes. For example, a single macromodel can be coupled to multiple micromodels that each cover a different physical process, or a different part of the spatial domain. At every Δt timestep of the macromodel for x, the micromodel(s) is called to propagate y_j to y_{j+1} given x_j (possibly using multiple smaller "micro" time steps in order to cover one "macro" time step Δt) so that r_j can be updated to r_{j+1}.

In terms of the discrete-time system (2), the aim of reduced multiscale modeling with a surrogate is to construct a computationally cheap approximation for updating r_j to r_{j+1}, so that the expensive micromodel is no longer needed to simulate x.

3. A stochastic surrogate model from data

We assume that we have observation data available in the form of time series of (x_j, r_j) generated by the full multiscale model (2). We denote these observations by (x_j^o, r_j^o), $j = 0, 1, \ldots,$

T (note that the data include r but not y). The case we have in mind here is where these data come from numerical simulation of the full multiscale model, e.g. simulation on a limited spatial domain or over a limited time interval. However they can also come from physical experiments or measurements. If only observations of x are available, we can easily construct "observations" of r using $r_j^o = x_{j+1}^o - F(x_j^o)$, from (2a). We assume that there is no significant observational error.

Key to our approach is that we aim to build a surrogate model for the time evolution of r by sampling from the distribution of r_{j+1} conditional on the past states of x and r, i.e. sampling from the conditional distribution

$$r_{j+1} \mid r_j, r_{j-1}, \ldots, x_j, x_{j-1}, \ldots \tag{3}$$

It is usually not known how to obtain this distribution in a systematic way from the model (1) or (2). Therefore we make use of the observations (x_j^o, r_j^o) to build a surrogate.

We note that we do not need to have an explicit expression for the conditional distribution of r_{j+1} (or an approximation of it), we merely need to be able to sample from it. This can be achieved by *resampling* from the observations in an appropriate way. Because we use resampling, we do not have to assume specific structural properties of the conditional distribution of r for constructing the surrogate.

Thus, we construct a "stochastic surrogate" for r by random sampling from $r_{j+1}^o \mid r_j^o, r_{j-1}^o, \ldots, x_j^o, x_{j-1}^o, \ldots$. To make this practical, we assume finite memory (i.e., r_{j+1}^o does not dependent on $r_{j'}^o$ if $j - j'$ is large enough), and we define the feature vectors

$$\tilde{d}_j := (\tilde{r}_j, \tilde{r}_{j-1}, \ldots, \tilde{r}_{j-J}, \tilde{x}_j, \tilde{x}_{j-1}, \ldots, \tilde{x}_{j-J}) \tag{4a}$$
$$d_j^o := (r_j^o, r_{j-1}^o, \ldots, r_{j-J}^o, x_j^o, x_{j-1}^o, \ldots, x_{j-J}^o) \tag{4b}$$

for some finite memory depth J. The feature vectors take values in the feature space, which in this case has dimension $2N(J + 1)$ since $\dim(\tilde{x}) = \dim(\tilde{r}) = N$. It is straightforward to change the definition of the feature vectors, for example using different memory depth (history) for $\tilde{x}$ and $\tilde{r}$ (in (4) it is J for both), or including only every nth time step (subsampling), or using functions of $\tilde{r}$ or $\tilde{x}$, or leaving out either $\tilde{r}$ or $\tilde{x}$ entirely. For ease of exposition, we stick to the definitions in (4).

We sample $\tilde{r}_{j+1}$ randomly from the set S_{j+1} consisting of all observations r_{i+1}^o whose associated feature vector d_i^o is close to $\tilde{d}_j$. Our reduced model then is

$$\tilde{x}_{j+1} = F(\tilde{x}_j) + \tilde{r}_j \tag{5a}$$
$$\tilde{r}_{j+1} : \text{ random sample from } S_{j+1} := \left\{ \forall\, r_{i+1}^o \,\middle|\, \tilde{d}_j \text{ close to } d_i^o \right\} \tag{5b}$$

The resampling step to update $\tilde{r}_j$ is very similar in philosophy to the local bootstrap for Markov processes proposed by [30] and the nearest neighbor (k-NN) resampling scheme by [31]. By the local bootstrap procedure, "pseudo-time series" can be generated that reproduce the temporal dependence properties of observations from a stationary Markov process of order p, see [30].

However, an important difference with the situation considered in [30,31] is that here, we resample a quantity (r) that is (two-way) coupled with another quantity (x) which is *not* resampled. We update $\tilde{r}_j$ by resampling from $r_{i+1}^o \mid r_i^o, x_i^o, r_{i-1}^o, x_{i-1}^o, \ldots$ but we update $\tilde{x}_j$ by using the model (5a). In [30,31] there is no such coupling to another model involved.

Note that we do not use the expectation (sample average) of S_{j+1} to update $\tilde{r}_j$ in (5b). Using the expectation may be well-suited for a one-step ahead prediction of $\tilde{x}_{j+1}$, however it misses the inherent uncertainty of r_{j+1} given its current and past states (as summarized in the feature vector). Another aspect of sampling versus averaging is that the expectation may be "unphysical"
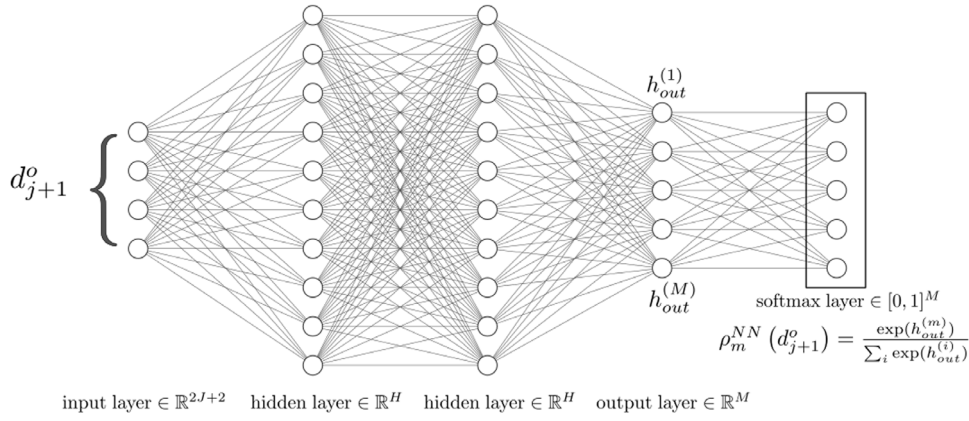
Fig. 1. Diagram of a feed-forward QSN with two hidden layers. Here, $h_{out}^{(i)}$ is the output of the ith neuron in the output layer, to which a softmax layer is attached.

(i.e., yield a value not consistent with the full model (2)) whereas the individual samples were generated by the full model and are thus entirely consistent with it.

In previous studies [11,26] we used resampling (albeit formulated somewhat differently) with an implementation relying on "binning": the feature space was divided into non-overlapping bins (or cells), and $\tilde{d}_j$ was considered close to d_i^o when both were in the same bin. At any time during simulation with the reduced model, the feature vector $\tilde{d}_j$ fell within a single bin. From the set of observation feature vectors d_i^o that fell in the same bin, we randomly selected one and used the associated r_{i+1}^o as $\tilde{r}_{j+1}$.

The results with this implementation were positive, but a drawback is that one quickly runs into curse of dimension problems when binning the feature space. If we use N_{bins} bins in each dimension of the feature space, the total number of bins is $N_{bins}^{2N(J+1)}$. This number grows exponentially with both N and J.

However, instead of binning the *input* features (d_i^o), we can also bin the *output* data (r_{i+1}^o). The advantage is that the curse of dimensionality is strongly reduced in this way, since it involves only one variable and no time-lagged quantities are included. Thus, the space to be binned has dimensions N, rather than $2N(J+1)$. If N is large, the curse of dimensionality will still pose problems. We discuss in Section 4.2 how this can be dealt with. The disadvantage of binning the output data is that, unlike in the previous approach (where the features were binned), no simple map from $\tilde{d}_j$ to the output bin (from which to sample $\tilde{r}_{j+1}$) exists. However, we can learn this mapping from the data using a neural network. This is discussed in the next section.

4. Resampling by neural network classification

We introduce here an approach for resampling by combining binning and probabilistic classification using a neural network. With this approach we can generate $\tilde{r}_{j+1}$ by resampling from the observation data, as in (5), without being hampered by curse of dimension problems that occur if the input feature vector is of high dimension.

The basic idea is the following. As mentioned briefly in the previous section, we discretize the space of the observation $\{r_j^o\}$ by defining a set of M non-overlapping subsets $\{B_m\}_{m=1}^M$ (referred to as "bins" here) so that $r_j^o \in B_1 \cup B_2 \cup \cdots \cup B_M$ for all j. Then we train a neural net to map the feature vector d_j^o to a probability distribution over the bins $\{B_m\}$. This distribution corresponds to the probabilities that r_{j+1}^o sits in the various bins B_m, given the feature vector d_j^o. Denoting the probability distribution as the M-dimensional vector $\rho := (\rho_1, \ldots, \rho_M)^T$, we thus want to train a neural net ρ^{NN} such that

$$\rho_m^{NN}(d_j^o) \approx \mathbb{P}(r_{j+1}^o \in B_m \mid d_j^o). \tag{6}$$

Obviously, $\rho_m^{NN}(d) \in [0, 1]$ for all m, and $\sum_{m=1}^M \rho_m^{NN}(d) = 1$ for any feature vector d.

In the reduced model (5), given $\tilde{d}_j$, we generate $\tilde{r}_{j+1}$ by (i) computing $\rho = \rho^{NN}(\tilde{d}_j)$, (ii) random sampling of a bin index $m \in \{1, 2, \ldots, M\}$ in accordance with ρ, (iii) random sampling of $\tilde{r}_{j+1}$ from all r_i^o in the mth bin, B_m. Steps (ii) and (iii) can be combined: given ρ, we sample $\tilde{r}_{j+1}$ randomly from all r_i^o, with weights $w_i = \sum_m |B_m|^{-1} \rho_m \mathbf{1}(r_i^o \in B_m)$. Here $|B_m|$ denotes the number of training points r_i^o in B_m, and $\mathbf{1}(.)$ is the indicator function.

Training ρ^{NN} can be seen as a task of probabilistic classification, with the bins B_m as classes (M in total). In the next two subsections we discuss classification by a neural network, first for the case with $N = 1$, then for $N > 1$.

4.1. Scalar-valued r and x

In case r and x are scalar ($N = 1$), a neural network for the classification task has $2J + 2$ inputs (the dimension of the feature vector when $N = 1$) and a M-dimensional softmax output layer. This is denoted a quantized softmax network (QSN). Note that this approach has been generalized to predict conditional kernel-density estimates for continuous variables, see [32]. However, here we stick to the discrete version.

We use a feed-forward architecture (see Fig. 1), noting that it can easily be replaced by other architectures in our set-up. The softmax layer computes the output probability mass function (pmf) as

$$\rho_m^{NN}(d_j^o) = \frac{\exp\left(h_{out}^{(m)}\right)}{\sum_{i=1}^M \exp\left(h_{out}^{(i)}\right)}. \tag{7}$$

Here, $h_{out}^{(i)}$ is the output of the ith neuron in the output layer. Finally, a cross-entropy loss function [33] is used for training the network:

$$\mathcal{L} = -\frac{1}{T - J - 1} \sum_{j=J}^{T-1} \sum_{m=1}^M [\log \rho_m^{NN}(d_j^o)] \mathbf{1}(r_{j+1}^o \in B_m). \tag{8}$$

For a given j index, the gradient of (8) with respect the output neurons (required to initialized a back propagation step), is given by $\partial \mathcal{L}/\partial h_{out}^{(m)} = \rho_m^{NN} - \mathbf{1}(r_{j+1}^o \in B_m)$ [33].

We emphasize that the method proposed here is fundamentally different from the approach of using a neural net to predict $\tilde{r}_{j+1}$ directly from $\tilde{d}_j$ as in e.g. [16–18]. In the latter approach, $\tilde{r}_{j+1}$ is modeled as a function of $\tilde{d}_j$, with the neural net embodying the function as a deterministic mapping from the space of $\tilde{d}_j$ to the space of $\tilde{r}_{j+1}$. By contrast, in our approach $\tilde{r}_{j+1}$ is resampled

instead of modeled. Moreover, the mapping from the space of $\tilde{d}_j$ to the space of $\tilde{r}_{j+1}$ in our approach is stochastic, not deterministic. Note that even if the neural net performs poorly, the $\tilde{r}_{j+1}$ that are generated remain consistent with the full model (as they are resampled from the full model), albeit possibly not well-matched with the feature vectors $\tilde{d}_j$. Consistency in the case of vector-valued r (i.e., $N > 1$) is discussed next.

4.2. Vector-valued r and x

For the situation where r and x are vector-valued ($N > 1$), generalization of the approach discussed in the previous section for the scalar case is possible, however it can quickly run into the curse of dimension when done naively. The $N > 1$ case arises for example when the vectors r and x represent spatial fields discretized on a grid, so that the different elements of the vectors are associated with different gridpoints.

There are several possibilities for generalization. On the input side, the most straightforward approach is to enlarge the input layer from $2J + 2$ to $2N(J + 1)$ inputs. This simply reflects the increased dimension of the feature vector. With large N, different network architectures than fully connected feed-forward may well be more effective, in particular convolutional neural networks for when r and x are gridded spatial fields. We leave further exploration of this for future study, here we stick to the feed-forward architecture discussed earlier. Besides different network architecture, a form of spatial localization can be used; this is discussed later in this section.

On the output side, if we bin the space in which the vector r lives, we quickly run into curse of dimension problems again. Instead, we bin all individual elements of the vector r separately. This scales linearly with dimension (N): with N_{bins} bins for each vector element, we have $N \times N_{bins}$ bins in total. As a consequence, the output layer is no longer fully connected to a single softmax layer, see Fig. 2 for a diagram involving $N = 2$ probabilistic outputs. While each element of r has its own independent softmax layer, all elements share the same feature vector and hidden layers. If there are dependencies between different elements of r in the full model (2), they must be learned from the data $\{r_j^o\}$. Such dependencies can be due to e.g. spatial correlations (with different vector elements representing different locations on a spatial grid). The separate elements of the vector $\tilde{r}_j$ generated this way are consistent with the full model, however their combination (i.e., the vector as a whole) may not be consistent if the network is poorly trained (in which case dependencies may be misrepresented).

While the scaling is linear with N, the number of output neurons can still become very large, especially for problems in two or three spatial dimensions (where N can be e.g. $\mathcal{O}(10^6)$). To tackle cases with large N due to gridded spatial fields, one can formulate "local" parameterizations. Let us denote x at time t_j at gridpoint n by $(x_j)_n$, and similarly for $(r_j)_n$. Also, Ω_n denotes a (to be specified) neighborhood of gridpoints centered on (and including) n. A local parameterization to generate $(\tilde{r}_{j+1})_n$ takes only $(\tilde{x}_j)_{n'}$ and $(\tilde{r}_j)_{n'}$ with $n' \in \Omega_n$ (and possibly their histories) as its inputs. Thus, it ignores $(\tilde{x}_j)_{n'}$ and $(\tilde{r}_j)_{n'}$ at other gridpoints $n' \notin \Omega_n$. If we assume conditional spatial homogeneity (given all $(\tilde{x}_j)_{n'}$ and $(\tilde{r}_j)_{n'}$ with $n' \in \Omega_n$, the distribution of $(\tilde{r}_{j+1})_n$ is independent of n), only one neural network for $(\tilde{r}_{j+1})_n$ is needed which can be used for all gridpoints. Moreover, data from different gridpoints can be pooled for training. Note that with such a local parameterization, the neural network is used to produce a scalar quantity, therefore it has only one softmax layer.

Two limiting cases for the neighborhood are $\Omega_n = \{1, \ldots, N\}$ (i.e., Ω_n consists of all gridpoints) and $\Omega_n = \{n\}$ (only gridpoint n itself). The latter, a completely localized parameterization

($\Omega_n = \{n\}$) is standard practice in e.g. atmosphere-ocean science. However, it may fail to capture spatial correlations.

Returning to our approach with N softmax layers, the loss function is also slightly modified, as it now includes summation over the N layers:

$$\mathcal{L} = -\frac{1}{T - J - 1} \sum_{j=J}^{T-1} \sum_{n=1}^{N} \sum_{m=1}^{M} [\log \rho_{m,n}^{NN}(d_j^o)] \mathbf{1}(r_{j+1,n}^o \in B_{m,n}). \quad (9)$$

Here, $\rho_{m,n}^{NN}$ is the QSN predicted probability for the mth bin of the nth softmax layer, and the indicator function $\mathbf{1}(r_{j+1,n}^o \in B_{m,n})$ represents the one-hot encoded data per softmax layer (see Fig. 2). The gradient of (9) with respect to the output neurons retains the same expression as in the case of a single softmax layer, i.e. $\partial \mathcal{L}/\partial h_{out}^{(m,n)} = \rho_{m,n}^{NN} - \mathbf{1}(r_{j+1,n}^o \in B_{m,n})$.

5. Numerical experiments

In this section we present several examples to test the approach proposed in the previous section: model reduction according to (5) with the resampling step in (5b) implemented using QSNs. In these tests, it is not our aim to recreate the exact trajectories of the original coupled model (2) with the reduced model. The loss of information due to model reduction, the stochastic nature of our surrogates, and the nonlinearity of multiscale problems, makes this impossible after a certain time integration length. Instead, our goal for the setup (5) is to reproduce the time-averaged statistical properties of the original macroscopic variables x, e.g.,. the probability density function or the auto-correlation function.

5.1. Model equations Lorenz 96

As a proof of concept, we test our setup on the well-known two-layer Lorenz 96 (L96) system, originally proposed by [34], which is a toy model for the atmosphere. It consists of a set of N ODEs describing the evolution of the macroscopic variables X_n, of which each ODE is coupled to L microscopic variables $Y_{l,n}$ (note that n and l are spatial grid indices here):

$$\frac{dX_n}{dt} = X_{n-1}(X_{n+1} - X_{n-2}) - X_n - F + r_n$$

$$r_n := \frac{h_x}{L} \sum_{l=1}^{L} Y_{l,n}$$

$$\frac{dY_{l,n}}{dt} = \frac{1}{\epsilon} \left[Y_{l+1,n}(Y_{l-1,n} - Y_{l+2,n}) - Y_{l,n} + h_y X_n \right]. \quad (10)$$

The macroscopic and microscopic variables X_n and $Y_{l,n}$ are considered variables on a circle of constant latitude (see Fig. 3), where the indices $n = 1, \ldots, N$ and $l = 1, \ldots, L$ denote the spatial location. Note that there are L microscopic solutions $Y_{l,n}$ for every n, such that $Y = (Y_{1,1}, Y_{1,2}, \ldots, Y_{L,N}) \in \mathbb{R}^K$, where $K = NL$. The circular shape of the domain is imposed through periodic boundary conditions:

$$X_n = X_{n+N}, \quad Y_{l,n} = Y_{l,n+N}, \quad Y_{l+L,n} = Y_{l,n+1}. \quad (11)$$

The two-step Adams–Bashforth method is chosen as the numerical time-discretization procedure, and we set $\Delta t = 0.01$ as the time step. Note that (10) is only used to generate the training data. In line with (5), we only solve the X_n equation with a QSN surrogate for r_n when predicting.

The behavior of the system is governed by the parameter settings of $\{N, L, F, h_x, h_y, \epsilon\}$. Commonly, ϵ is chosen such that $\epsilon \ll 1$, in which case a clear time-scale separation between the macroscopic and microscopic variables is imposed. We will follow [7] and use $\epsilon = 0.5$, such that no clear temporal gap exists,
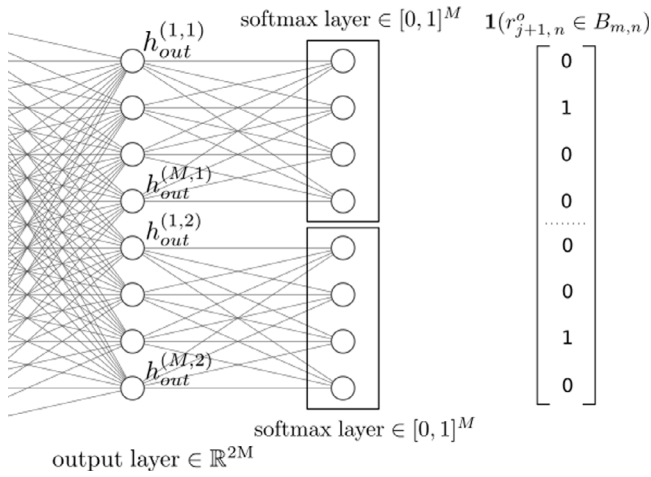
Fig. 2. Diagram of the final layers of a feed-forward QSN with two probabilistic outputs with 4 bins each ($N = 2$, $M = 4$). Here, $h_{out}^{(m,n)}$ is the output of the mth neuron in the output layer attached to the nth softmax layer. Here, $\mathbf{1}(r_{j+1,n}^o \in B_{m,n})$ displays an example one-hot encoded data vector, for both softmax layers.
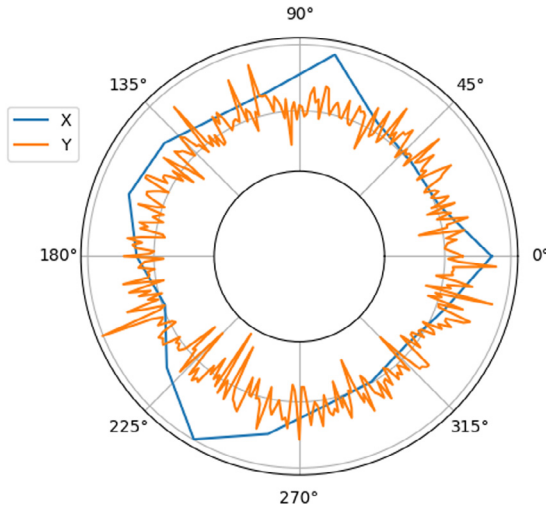

Fig. 3. A snapshot of the solution of the two-layer L96 system (10). Here X denotes the vector of all X_n solutions and Y is the vector containing all $Y_{l,n}$.

which is more realistic for turbulent geophysical flows, and more challenging for parameterizations. Specifically, we will use two different parameter settings:

1. The unimodal setting: $\{N, L, F, h_x, h_y, \epsilon\} = \{18, 20, 10, -1, 1, 0.5\}$,
2. The bimodal setting: $\{N, L, F, h_x, h_y, \epsilon\} = \{18, 20, 10, -2, 1, 0.5\}$.

The naming convention stems from the number of modes in the probability density functions (pdfs) of the X_k variables. The bimodal setting intensifies the feedback from the microscopic ODEs to the macroscopic ODE, by modifying the h_x parameter (see (10)). Specifically, we decrease h_x from -1 to -2. In the unimodal setting, both the X_n and r_n pdfs are unimodal and nearly Gaussian, see Fig. 7. This is no longer the case for $h_x = -2$, when the pdfs become non-symmetric and bimodal. The difference between the two parameter settings can also clearly be seen from a scatter plot of X_n vs r_n, see Fig. 4.

Other authors have used L96 as a benchmark for machine learning methods. For instance in [35], three separate methods

are used to predict the full $X^{(j+1)} = (X_1(t_{j+1}), \ldots, X_N(t_{j+1}))$ vector. Also, the authors of [36] used a neural network to predict the tendency $\Delta X = X^{(j+1)} - X^{(j)}$ one time step ahead. Note that these are deterministic approaches, and are not applied in the context of parameterization (they are used to predict $X^{(j+1)}$ itself rather than $r^{(j+1)}$). Closer to our approach, the interesting recent work of [25] uses conditional Generative Adversarial Networks (GANs) for the purpose of stochastic parameterization of the L96 subgrid-scale term. GANs have a different architecture than our QSNs, and unlike our approach, include a stochastic component in the inputs. The approach from [25] does not involve resampling, and was tested on the L96 model at a (single) different parameter setting.

5.2. Learning procedure

To inform the weights of a QSN, we use back propagation with Stochastic Gradient Descent with the RMSProp optimizer and a learning rate of 0.001 [33]. The number of training iterations was set to 10 000. After experimenting with different networks, we selected 3 hidden layers, each with 256 neurons and leaky Rectified Linear Unit activation functions. The output layer, which feeds into the softmax layers, is linear. Furthermore, the input features d_j^o are standardized to have zero mean and a standard deviation of one, before being fed into the QSN. We create two local surrogates (both trained on a single spatial location, one with neighborhood $\Omega_n = \{n\}$ and one with $\Omega_n = \{1, \ldots, N\}$), as well as a stochastic surrogate for the full vector-valued $r := [r_1, \ldots, r_N]^T$, in which case we have N softmax layers, see Section 4.2.

A common practice in machine learning is to leave out a part of the data set (i.e. to not use it in training), to test the accuracy of the final model. In our case however, the trained neural network itself is not the final model, it is merely a source term in the final model (the macroscopic ODE (5a)). To test the accuracy of the neural network, we therefore have to perform a simulation with two-way coupling between the ODE and the neural net. We leave out the final 50% of the data and test the ability of this coupled system to predict the macroscopic statistics of the test set.

5.3. Results: verification

To visualize the complexity of the bin classification and to verify quality of a trained QSN, consider Fig. 5. Here we show a scatter plot with a lagged X_n feature on both axes. The symbols in the scatter plot are color coded for the corresponding r bin index. We show both the exact results from the training data and the QSN prediction, using the training d_j^o as input features. Fig. 5 verifies that the QSN can learn a good bin index representation in the space of lagged input variables. A typical misclassification error for each softmax layer $n = 1, \ldots, N$ (defined as $\arg_m \max \rho_{m,n}^{NN}(d_j^o) \neq \arg_m \max \mathbf{1}(r_{j+1,n}^o \in B_{m,n})$), is roughly 3%–4%.

To verify the random resampling software we plot the time series of the data r_{j+1}^o and stochastic surrogate $\tilde{r}_{j+1}(d_j^o)$ in Fig. 6. It can be seen that the time series of the surrogate $\tilde{r}_{j+1}(d_j^o)$ is noisier than that of the data r_{j+1}^o. This is due to the random sampling step summarized in (5b): $\tilde{r}_j(d_{j-1}^o)$ and $\tilde{r}_{j+1}(d_j^o)$ are not as strongly correlated as d_{j-1}^o and d_j^o. However, the high-frequency variability in $\tilde{r}_{j+1}$ is much smaller than the low-frequency variability, and gets filtered out by the time integration of $\tilde{x}$ as we will see in the next subsection.

5.4. Results: validation

We simulate the reduced system (5) from $t = 0$ until $t = 1000$, while the training data spanned $t \in [0, 500]$. The sampling interval of the training data is 0.01, same as the time step used
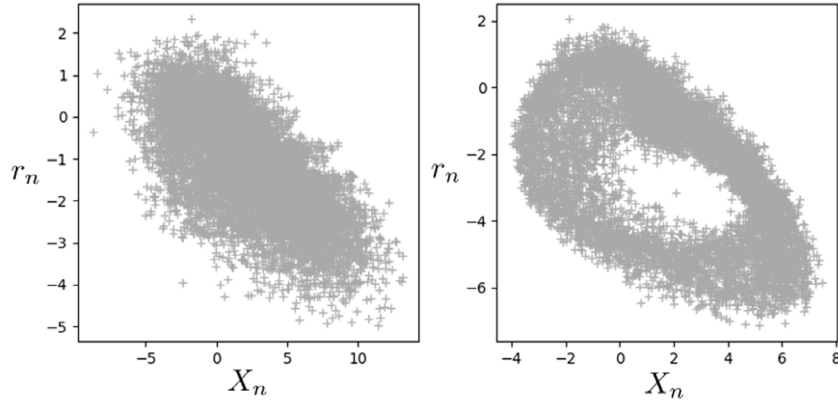
Fig. 4. Scatter plot of X_n vs r_n for $\{N, L, F, h_x, h_y, \epsilon\} = \{18, 20, 10, -1, 1, 0.5\}$ (left), and $\{N, L, F, h_x, h_y, \epsilon\} = \{18, 20, 10, -2, 1, 0.5\}$ (right).
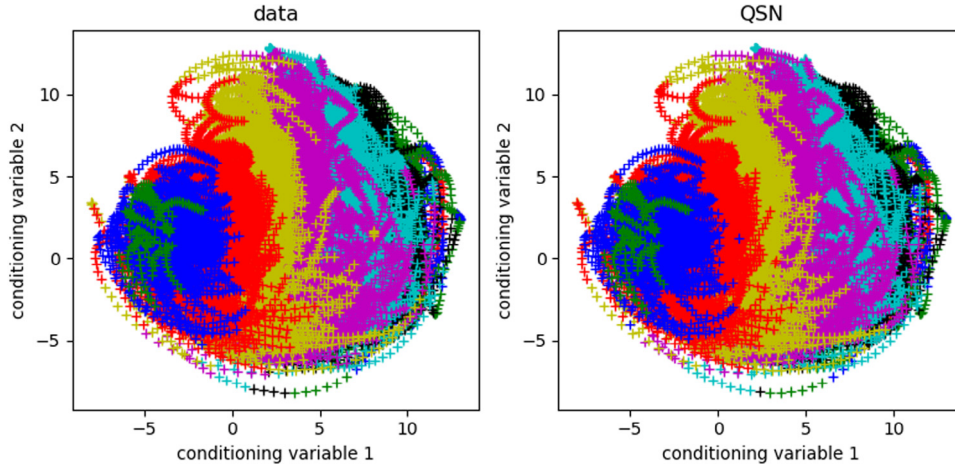

Fig. 5. A twice time-lagged X_n conditioning feature is shown on both axes, and the color of the "+" symbols denotes the index of the bin with highest probability of the corresponding r_{j+1} sample. The left plot shows the output bin indices of the training data, and the right plot shows the predicted indices corresponding to QSN output pmf, when using the training features d_j^o as input. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
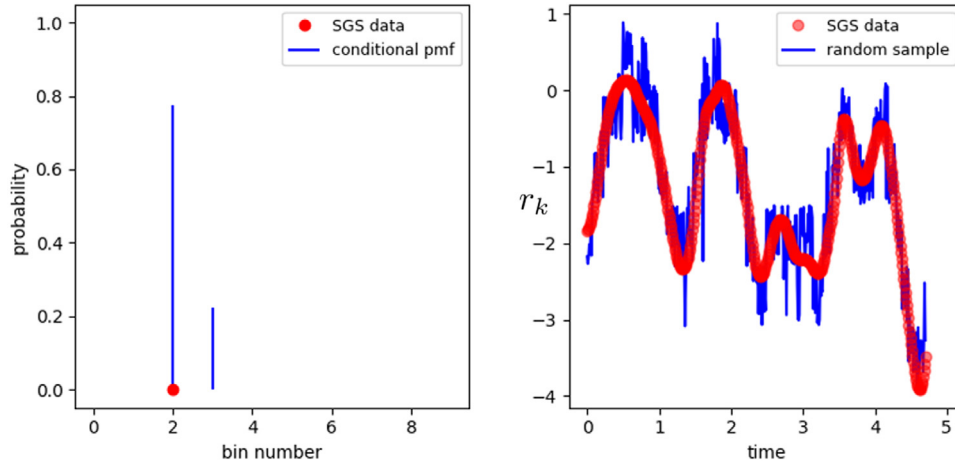
Fig. 6. At any given time t_j, the QSN predicts a conditional pmf (left). The predicted bin index from this pmf feeds into our data resampling software, which outputs a random $\tilde{r}_{j+1}$. Its time series (for a single spatial point n), along with the subgrid-scale (SGS) data r are shown on the right.

for numerical integration, see Section 5.1. Subsampling the data is possible (see also [7]), e.g. by adapting the definition of the feature vector as mentioned in Section 3, however we do not use it here. Our macroscopic statistics of interests are the probability density function and auto-correlation function of X_n, as well as the cross-correlation function of X_n and X_{n+1}. In addition, we

compute the same statistics for r_n. All statistics are averaged over all n, and computed on the test set only, i.e. using data from $t \in [500, 1000]$.

Our main goal here is twofold: We investigate the importance of the length of the history (memory depth) in the feature vector (J in definition ((4))). Furthermore, we demonstrate the relevance
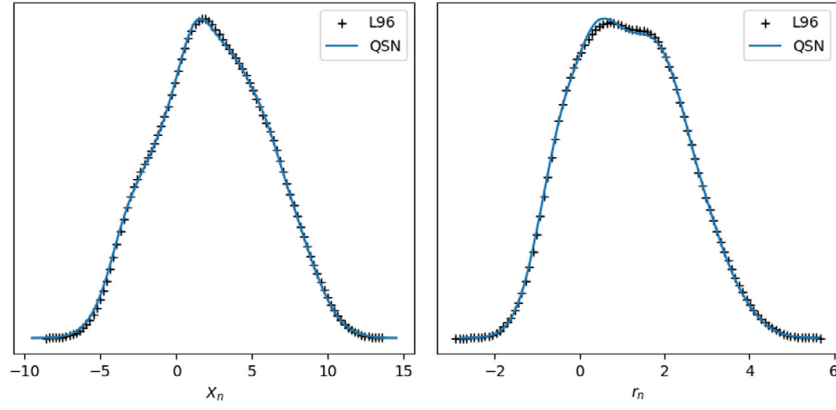
Fig. 7. Probability density functions of X_n (left) and r_n (right). Dots denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.
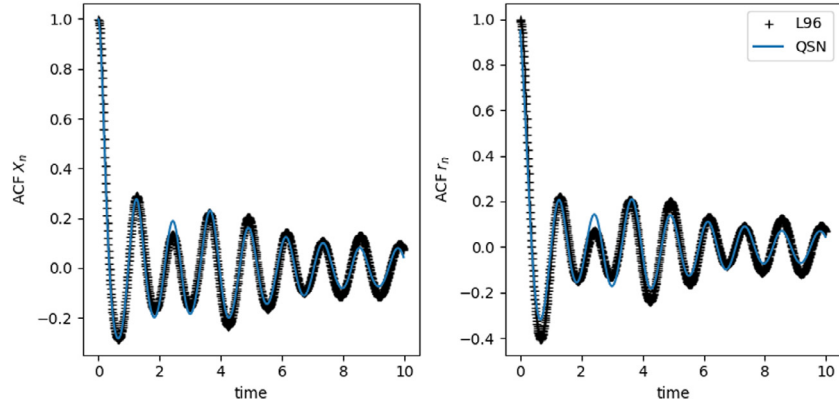

Fig. 8. Auto-correlation function of X_n (left) and r_n (right). Dots denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.
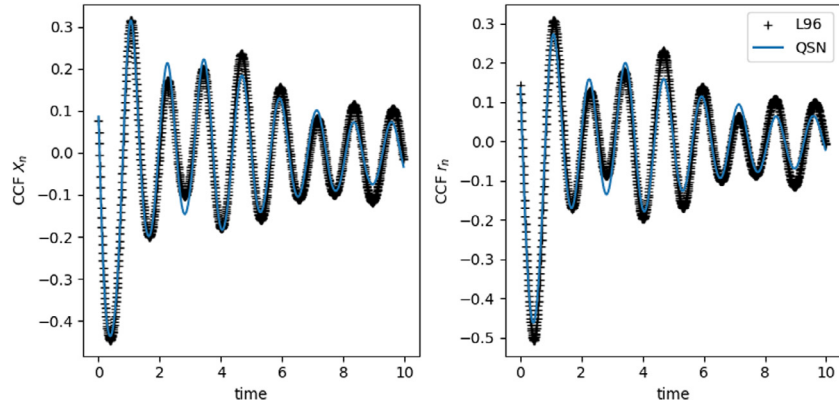

Fig. 9. Cross-correlation function of X_n and X_{n+1} (left) and r_n and r_{n+1} (right). Dots denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.

of a stochastic approach by comparing resampling with using averages (as discussed in Section 3). In addition, we compare the performance of a surrogate that is trained and applied locally, and a surrogate which predicts the entire $\tilde{r}_{j+1} \in \mathbb{R}^N$ vector at once.

5.4.1. Short vs long memory

For our first test case we will the unimodal parameter values. We first create a full-vector QSN with a short memory, e.g. with two time-lagged $X := (X_1, \ldots, X_N)$ vectors in d_j^o. Let $X^{(j)} := X(t_j)$. The statistical results, when using $d_j^o = (X^{(j)}, X^{(j-9)})$, are shown in Figs. 7–9. Despite the short memory in d_j^o, these display a good match between the statistics of the reduced and the full L96 system. Note that in this case $d_j^o \in \mathbb{R}^{2N}$, which would be

impossible in the case of a surrogate that bins the input space, as in [11].

Above, we used a parameter setting for the L96 model that makes parameterization challenging because of the lack of scale separation ($\epsilon = 0.5$ as discussed). Here, we put more strain on the QSN by employing the bimodal parameter setting. This is a harder test case, and more care needs to be taken with the specification of the feature vector d_j^o. In fact, the amount of memory in d_j^o becomes very important. If we use 10 X vectors, i.e. $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-9)})$, we obtain the results of Figs. 10–12, which display a clear failure of capturing the reference statistics. Also note the bimodal nature of the reference pdfs. We performed further tests with 25, 50 and 75 lagged X vectors,
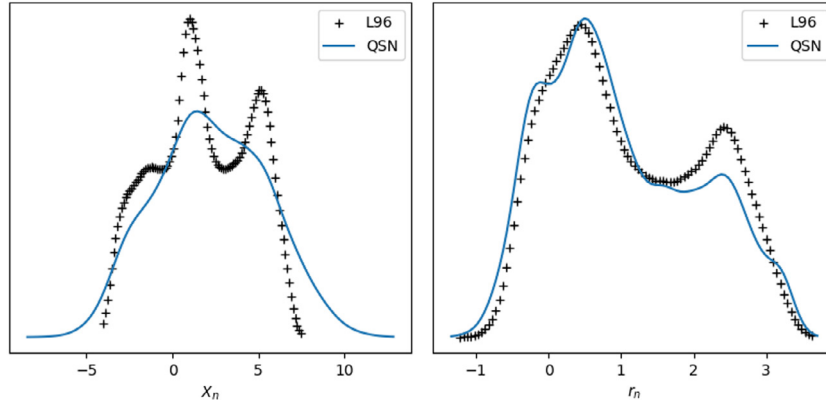
Fig. 10. Probability density functions of X_n (left) and r_n (right), using $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-9)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.
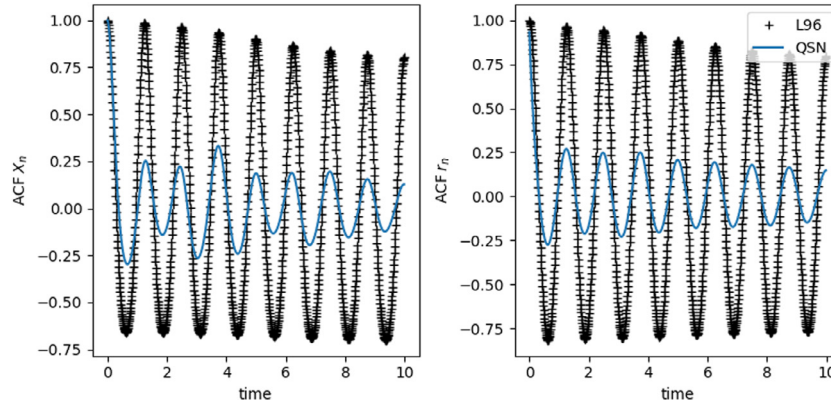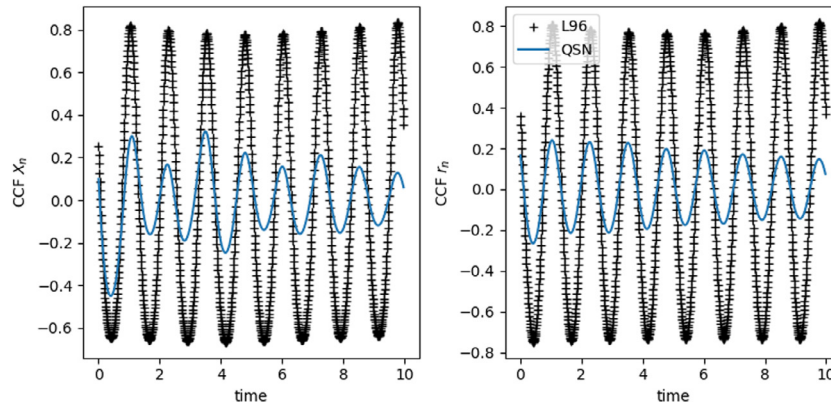

Fig. 11. Auto-correlation function of X_n (left) and r_n (right), using $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-9)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.

Fig. 12. Cross-correlation function of X_n and X_{n+1} (left) and r_n and r_{n+1} (right), using $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-9)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.

and only obtained good statistics with 75 vectors. These results are shown in Figs. 13–15.

5.4.2. Local parameterization

In Section 4.2 we discussed the set-up of a local parameterization, in which the QSN has only one softmax layer and the prediction for gridpoint n depends on quantities at gridpoints in the neighborhood Ω_n centered on n. Here we show results for the

bimodal case using $\Omega_n = \{1, \ldots, N\}$ (results with $\Omega_n = \{n\}$ are in the next subsection). For memory depth we take $J = 74$, a value that gave good results in the previous section.

The probability density functions are in Fig. 16, autocorrelation functions in Fig. 17. As can be seen, the reduced system with this surrogate succeeds in capturing the reference statistics, similar to the results from the previous section where a full-vector QSN was used with $J = 74$ (Figs. 13–15).
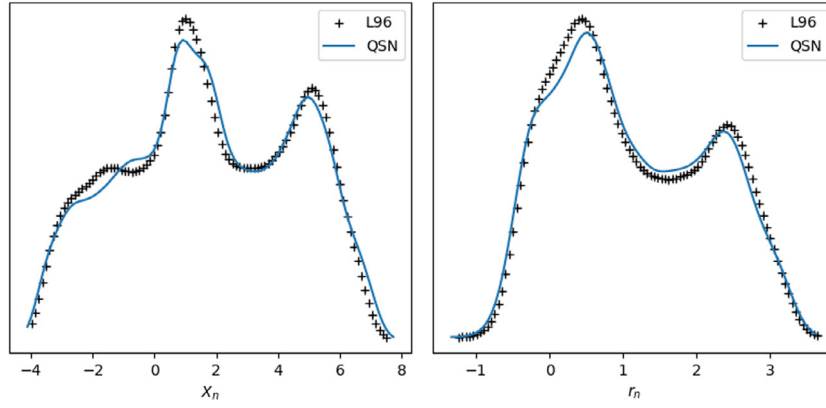
Fig. 13. Probability density functions of X_n (left) and r_n (right), using $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-74)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.
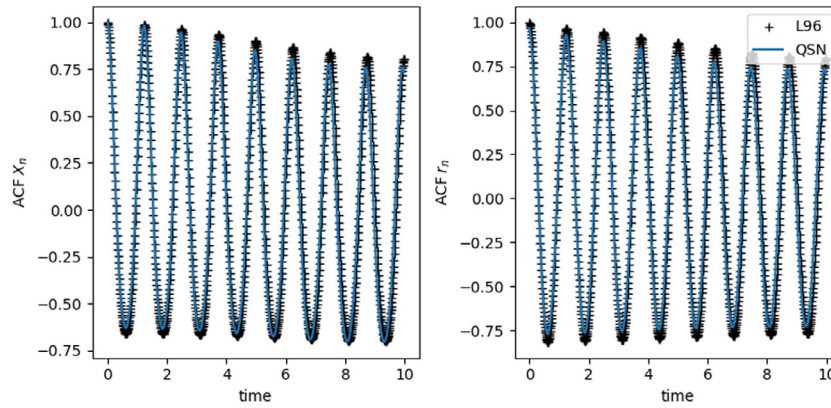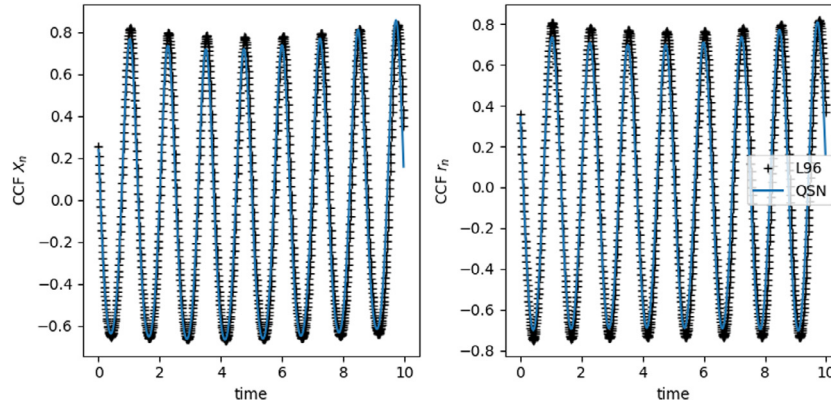

Fig. 14. Auto-correlation function of X_n (left) and r_n (right), using $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-74)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.

Fig. 15. Cross-correlation function of X_n and X_{n+1} (left) and r_k and r_{k+1} (right), using $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-74)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.

5.4.3. Stochastic vs deterministic

We recall (5b) and the discussion in Section 3 about resampling versus using bin averages. Here we compare sampling from the set S_{j+1} (i.e. the output bin in the case of a QSN), with using the sample mean of S_{j+1}. If we also use $\arg_m \max \rho^{NN}(\tilde{d}_j)$ as the predicted output bin index, we obtain a completely deterministic surrogate $\tilde{r}_{j+1}$. We tested this approach on the full-vector surrogates of the preceding section, and obtained similar results as with resampling. However, when applying the surrogate locally, we find a significant impact due to the stochastic nature of $\tilde{r}_{j+1}$. As mentioned, the local surrogate is trained on a single spatial

location (here with neighborhood $\Omega_n = \{n\}$), and during prediction it is applied independently for each X_n equation. Such an approach is not uncommon, see e.g. [7,25,37], and it matches the local nature of traditional parameterization schemes.

As an example, consider the auto-correlation results of Figs. 18–19, which show the results of the stochastic and deterministic surrogate respectively. Note that here, $h_x = -1$ was used, i.e. the easier of the two parameter setting we consider. The stochastic surrogate clearly outperforms its deterministic counterpart in this case. Other statistics (cross-correlation, pdf) showed similar results.
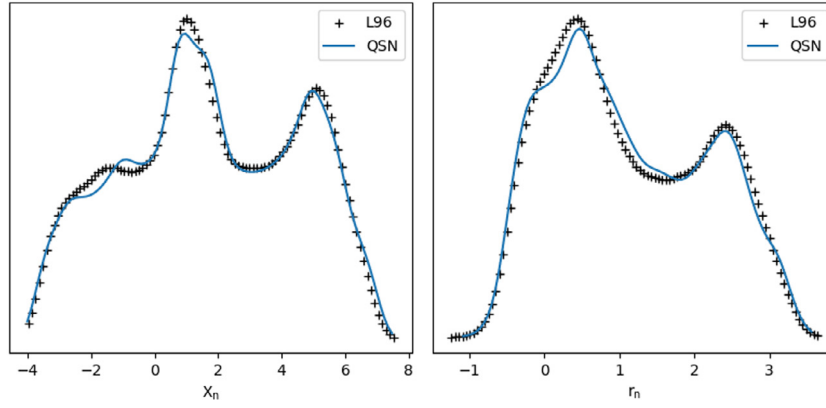
Fig. 16. Probability density functions of X_n (left) and r_n (right), using the local parameterization with neighborhood $\Omega_n = \{1, \ldots, N\}$ and $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-74)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.
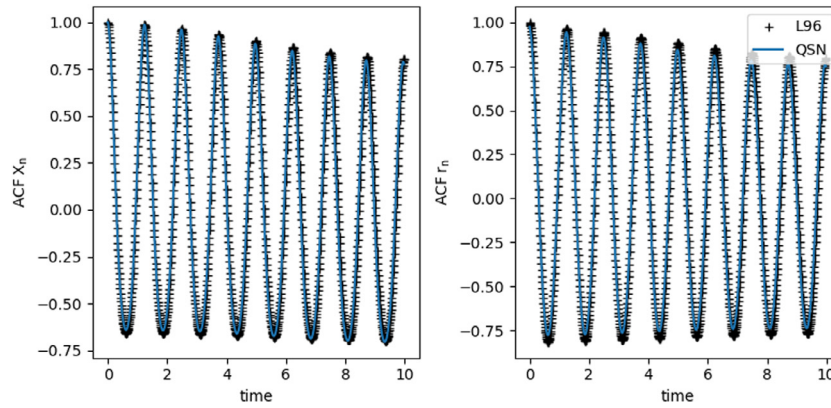

Fig. 17. Auto-correlation function of X_n (left) and r_n (right), using the local parameterization with neighborhood $\Omega_n = \{1, \ldots, N\}$ and $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-74)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with QSN surrogate.
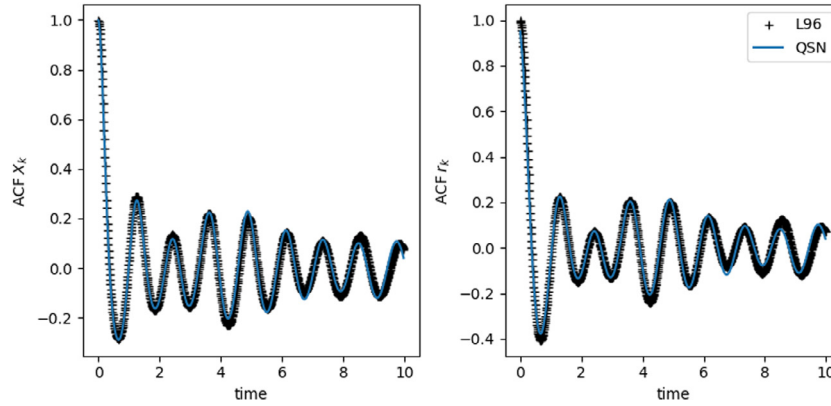

Fig. 18. Auto-correlation function of X_n (left) and r_n (right), using $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-74)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with the *local and stochastic* QSN surrogate.

5.4.4. X-only vs X − r conditioning

In Section 3 we outlined the general case in which the feature vector consists of time-lagged x and r variables. Yet, thus far we have only shown X-only surrogates for the L96 system. We found that including r_j in d_j^o can contribute to obtain a small training error, especially in the case of local surrogates. However, we did not obtain robust statistical predictions when doing so. These results are in line with those from [25]. As mentioned in Section 5.1, these authors considered stochastic parameterizations for L96 by means of conditional GANs. For a variety of tests, their X-only GANs clearly outperformed the GANs conditioned on both X and

r. One possible cause may be overfitting, due to the very strong correlation between r_j and r_{j+1} [25].

5.5. Software

The source code used to generate the results can be downloaded from [38].

6. Discussion and future challenges

The preceding results show that, for the L96 system, the amount of memory (J) in d_j^o was not of major importance in the
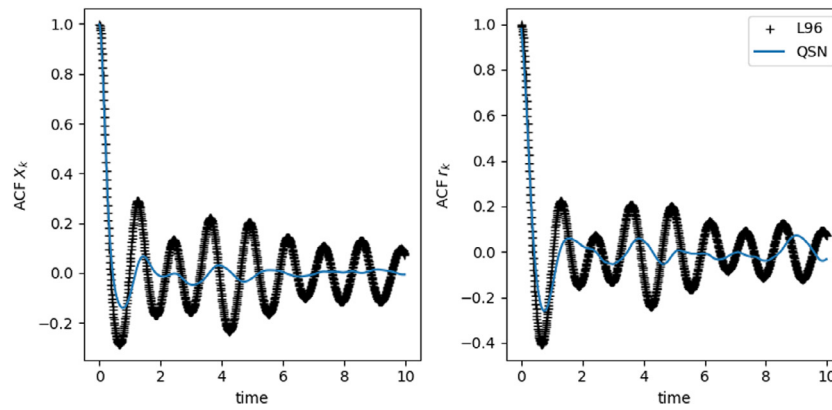
Fig. 19. Auto-correlation function of X_n (left) and r_n (right), using $d_j^o = (X^{(j)}, X^{(j-1)}, \ldots, X^{(j-74)})$. Plus symbols denote the full two-layer solution and solid lines denote the reduced model with the *local and deterministic* QSN surrogate.

tests with the unimodal parameter settings ($h_x = -1$). However, the more challenging bimodal parameter settings ($h_x = -2$, right subplot of Fig. 4) resulted in a problem for which memory became crucial. In general, we expect that for more complicated (geophysical) flow problems, memory will play an important part. It is clear however, that the "optimal" d_j^o is problem dependent, and a systematic procedure for designing the best feature vector is an interesting avenue for future research. This could involve changing the network architecture (e.g. combining resampling with Long Short-Term Memory networks [39]), or finding optimal time lags using approaches as described in [40].

Another clear avenue for future research is to apply machine learning with resampling as proposed here to more complex flows. In Section 4.2, we have discussed ways to deal with the large output dimension, including spatially localized parameterization. An interesting test problem is a two-dimensional ocean model, as in e.g. [20,26,27].

Finally, as mentioned in Section 5.2, we train the QSN separately on the data, and afterwards a validation procedure involves a two-way coupling between the ODEs and the QSN. This gave satisfactory results for the L96 model, but for more complicated problems, such an "offline" training strategy could lead to instabilities in the "online", two-way coupled simulation as discussed in [37]. Developing new learning procedures, in which the neural network is trained while it is part of the larger dynamical system is of interest.

7. Conclusion

We presented a machine-learning method for the conditional resampling of subgrid-scale data of multiscale dynamical systems, resulting in a stochastic parameterization for the unresolved scales. The current model is comprised of a feed-forward architecture with (multiple) softmax layers attached to the output. The output data is divided into a finite number of non-overlapping intervals (denoted as 'bins'), and the softmax layers predict a discrete probability density function over these bins, conditioned on time-lagged macroscopic input features. First a bin is sampled from this distribution, which is followed by randomly selecting a reference subgrid-scale data point from the identified bin. This stochastic surrogate model then replaces the original subgrid-scale term in the dynamical system, and we validate the method by examining the ability of this system to capture the long-term statistics of the resolved, macroscopic variables.

In this initial study we considered the Lorenz 96 system at two different parameter settings. We were able to validate our

method on this problem, provided that a sufficient number of time-lagged variables were included in the feature vector. Finally, we also found that overall, the stochastic nature of the surrogate led to more robust performance.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Weinan E, Bjorn Engquist, Xiantao Li, Weiqing Ren, Eric Vanden-Eijnden, Heterogeneous multiscale methods: A review, Commun. Comput. Phys. 2 (3) (2007) 367–450.

[2] Grigoris Pavliotis, Andrew Stuart, Multiscale Methods: Averaging and Homogenization, Springer, 2008.

[3] Ioannis G. Kevrekidis, Giovanni Samaey, Equation-free multiscale computation: Algorithms and applications, Annu. Rev. Phys. Chem. 60 (2009) 321–344.

[4] Weinan E, Principles of Multiscale Modeling, Cambridge University Press, 2011.

[5] F. Sarghini, G. De Felice, S. Santini, Neural networks based subgrid scale modeling in large eddy simulations, Comput. & Fluids 32 (1) (2003) 97–108.

[6] Vladimir M. Krasnopolsky, Michael S. Fox-Rabinovitz, Complex hybrid models combining deterministic and machine learning components for numerical climate modeling and weather prediction, Neural Netw. 19 (2) (2006) 122–134.

[7] Daan Crommelin, Eric Vanden-Eijnden, Subgrid-scale parameterization with conditional Markov chains, J. Atmos. Sci. 65 (8) (2008) 2661–2675.

[8] K. Nimsaila, I. Timofeyev, Markov chain stochastic parametrizations of essential variables, Multiscale Model. Simul. 8 (5) (2010) 2079–2096.

[9] Jesse Dorrestijn, Daan T. Crommelin, A. Pier Siebesma, Harm J.J. Jonker, Stochastic parameterization of shallow cumulus convection estimated from high-resolution model data, Theor. Comput. Fluid Dyn. 27 (1–2) (2013) 133–148.

[10] Ming Ma, Jiacai Lu, Gretar Tryggvason, Using statistical learning to close two-fluid multiphase flow equations for a simple bubbly system, Phys. Fluids 27 (9) (2015) 092101.

[11] Nick Verheul, Daan Crommelin, Data-driven stochastic representations of unresolved features in multiscale models, Commun. Math. Sci. 14 (5) (2016) 1213–1236.

[12] Huan Lei, Nathan A. Baker, Xiantao Li, Data-driven parameterization of the generalized langevin equation, Proc. Natl. Acad. Sci. 113 (50) (2016) 14183–14188.

[13] Fei Lu, Kevin K. Lin, Alexandre J. Chorin, Data-based stochastic model reduction for the Kuramoto–Sivashinsky equation, Physica D 340 (2017) 46–57.

[14] Jaideep Pathak, Alexander Wikner, Rebeckah Fussell, Sarthak Chandra, Brian R. Hunt, Michelle Girvan, Edward Ott, Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model, Chaos 28 (4) (2018) 041101.

[15] Noah D. Brenowitz, Christopher S. Bretherton, Prognostic validation of a neural network unified physics parameterization, Geophys. Res. Lett. 45 (12) (2018) 6289–6298.

[16] Stephan Rasp, Michael S. Pritchard, Pierre Gentine, Deep learning to represent subgrid processes in climate models, Proc. Natl. Acad. Sci. 115 (39) (2018) 9684–9689.

[17] R. Maulik, O. San, A. Rasheed, P. Vedula, Subgrid modelling for two-dimensional turbulence using neural networks, J. Fluid Mech. 858 (2019) 122–144.

[18] T. Bolton, L. Zanna, Applications of deep learning to ocean data inference and subgrid parameterization, J. Adv. Modelling Earth Syst. 11 (1) (2019) 376–399.

[19] Tim N. Palmer, A nonlinear dynamical perspective on model error: A proposal for non-local stochastic-dynamic parametrization in weather and climate prediction models, Q. J. R. Meteorol. Soc. 127 (572) (2001) 279–304.

[20] P.S. Berloff, Random-forcing model of the mesoscale oceanic eddies, J. Fluid Mech. 529 (2005) 71–95.

[21] Judith Berner, Ulrich Achatz, Lauriane Batte, Lisa Bengtsson, Alvaro de la Cámara, Hannah M. Christensen, Matteo Colangeli, Danielle R.B. Coleman, Daan Crommelin, Stamen I. Dolaptchiev, et al., Stochastic parameterization: Toward a new view of weather and climate models, Bull. Am. Meteorol. Soc. 98 (3) (2017) 565–588.

[22] T.N. Palmer, Stochastic weather and climate models, Nature Rev. Phys. 1 (7) (2019) 463–471.

[23] Alexandre Joel Chorin, Ole H. Hald, Stochastic Tools in Mathematics and Science, Springer, 2009.

[24] Kevin K. Lin, Fei Lu, Data-driven model reduction, Wiener projections, and the Mori–Zwanzig formalism, 2019, arXiv preprint arXiv:1908.07725.

[25] D.J. Gagne, H.M. Christensen, A.C. Subramanian, A.H. Monahan, Machine learning for stochastic parameterization: Generative adversarial networks in the Lorenz'96 model, J. Adv. Modelling Earth Syst. 12 (3) (2020).

[26] Nick Verheul, Jan Viebahn, Daan Crommelin, Covariate-based stochastic parameterization of baroclinic ocean eddies, Math. Clim. Weather Forecast. 3 (1) (2017) 90–117.

[27] Wouter Edeling, Daan Crommelin, Towards data-driven dynamic surrogate models for ocean flow, in: Proceedings of the Platform for Advanced Scientific Computing Conference, 2019, pp. 1–10.

[28] Alfons Hoekstra, Bastien Chopard, Peter Coveney, Multiscale modelling and simulation: A position paper, Phil. Trans. R. Soc. A 372 (2021) (2014) 20130377.

[29] Fredrik Jansson, Gijs van den Oord, Inti Pelupessy, Johanna H. Grönqvist, A. Pier Siebesma, Daan Crommelin, Regional superparameterization in a global circulation model using large eddy simulations, J. Adv. Modelling Earth Syst. 11 (9) (2019) 2958–2979.

[30] Efstathios Paparoditis, Dimitris N. Politis, The local bootstrap for Markov processes, J. Statist. Plann. Inference 108 (1–2) (2002) 301–328.

[31] Upmanu Lall, Ashish Sharma, A nearest neighbor bootstrap for resampling hydrologic time series, Water Resour. Res. 32 (3) (1996) 679–693.

[32] Luca Ambrogioni, Umut Güçlü, Marcel A.J. van Gerven, Eric Maris, The kernel mixture network: A nonparametric method for conditional density estimation of continuous random variables, 2017, arXiv preprint arXiv:1705.07111.

[33] Charu C. Aggarwal, Neural networks and deep learning, Springer, 2018.

[34] Edward N. Lorenz, Predictability: A problem partly solved, in: Proc. Seminar on Predictability, Vol. 1, 1996.

[35] Ashesh Chattopadhyay, Pedram Hassanzadeh, Krishna Palem, Devika Subramanian, Data-driven prediction of a multi-scale Lorenz 96 chaotic system using a hierarchy of deep learning methods: Reservoir computing, ANN, and RNN-LSTM, 2019, arXiv preprint arXiv:1906.08829.

[36] Peter D. Dueben, Peter Bauer, Challenges and design choices for global weather and climate models based on machine learning, Geosci. Model Dev. 11 (10) (2018) 3999–4009.

[37] Stephan Rasp, Online learning as a way to tackle instabilities and biases in neural network parameterizations, 2019, arXiv preprint arXiv:1907.01351.

[38] W.N. Edeling, Easysurrogate - phys_D branch (github repository), 2020, https://github.com/wedeling/EasySurrogate/tree/phys_D.

[39] Sepp Hochreiter, Jürgen Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[40] Ray J. Frank, Neil Davey, Stephen P. Hunt, Time series prediction and neural networks, J. Intell. Robot. Syst. 31 (1–3) (2001) 91–103.