ON THE CHOICE OF THE NON-TRAINABLE INTERNAL WEIGHTS IN RANDOM FEATURE MAPS FOR FORECASTING CHAOTIC DYNAMICAL SYSTEMS

Abstract. The computationally cheap machine learning architecture of random feature maps can be viewed as a single-layer feedforward network in which the weights of the hidden layer are random but fixed and only the outer weights are learned via linear regression. The internal weights are typically chosen from a prescribed distribution. The choice of the internal weights significantly impacts the accuracy of random feature maps. We address here the task of how to best select the internal weights. In particular, we consider the forecasting problem where random feature maps are used to learn a one-step propagator map for a dynamical system. We provide a computationally cheap hit-and-run algorithm to select *good* internal weights which lead to good forecasting skill. We show that the number of good features is the main factor controlling the forecasting skill of random feature maps and acts as an effective feature dimension. Lastly, we compare random feature maps with single-layer feedforward neural networks in which the internal weights are now learned using gradient descent. We find that random feature maps have superior forecasting capabilities whilst having several orders of magnitude lower computational cost.

Pinak Mandal,[*] [1] Georg A. Gottwald, [†] [1] Nicholas Cranch, [‡] [1]
[1] The University of Sydney, NSW 2006, Australia

1. Introduction

Estimation and prediction of the state of a dynamical system evolving in time is central to our understanding of the natural world and to controlling the engineered world. Often practitioners are tasked with such problems without the knowledge of the underlying governing dynamical system. In such scenarios a popular approach is to reconstruct the dynamical model from observations of the system [38, 1, 46, 5]. Predicting the future state of the system from these reconstructions is particularly challenging for chaotic dynamical systems. Chaotic dynamical systems cannot be accurately predicted beyond a finite time known as the predictability time due to their sensitive dependence on the initial conditions.

In recent times machine learning has achieved remarkable progress in learning surrogate models for dynamical systems from given data. Recurrent networks such as Long Short-Term Memory networks [45, 48] and gated recurrent units [8] have been successfully applied in a plethora of time series prediction tasks [10, 6, 28]. These methods however often contain learnable parameters of the order of $\mathcal{O}(10^6)$, and require substantial fine tuning of hyperparameters and costly optimization strategies [27]. An attractive alternative is provided by random feature maps [43, 42, 40] and their extensions such as echo state networks and reservoir computers [36, 35, 41, 39]. These architectures can be viewed as a single-layer feedforward network in which the weights and biases of the hidden layer, the so-called internal parameters, are randomly selected before training and then are kept fixed. This renders the costly nonconvex optimization problem of neural networks to a simple linear least-square regression for the outer weights. The output of random feature maps and their extensions is hence a linear combination of a high-dimensional randomized basis. These methods have been shown to enjoy the universal approximation property, which states that in principle they can approximate any continuous function arbitrarily close [44, 3, 17, 13].

We focus here on classical random feature maps [43, 42, 40] which have recently been shown to have excellent forecasting skill for chaotic dynamical systems [16, 15]. The fact that random feature maps enjoy the universal approximation property does not provide practitioners with information on how to choose the internal parameters. The internal parameters are typically drawn from some prescribed distribution such as the uniform distribution on an interval or a Gaussian distribution. The forecasting capability of the learned

[*]pinak.mandal@sydney.edu.au, corresponding author
[†]georg.gottwald@sydney.edu.au
[‡]ncra8185@uni.sydney.edu.au

surrogate map sensitively depends on the choice of the distribution [16]. To generate good internal parameters which lead to improved performance of random feature maps, several data-independent methods such as Monte Carlo and quadrature based algorithms as well as data-dependent methods such as leverage score based sampling and kernel learning have been proposed; for a detailed survey see [29]. Levine and Stuart [26] performed Bayesian optimization to determine the internal weights together with other hyperparamters such as the regularization parameter. Dunbar *et al* [9] choose the distribution of the random weights from a parametric family. Its parameters are chosen to optimize a cost function motivated from Empirical Bayes arguments, with the optimization performed with derivative-free Ensemble Kalman inversion. Here we introduce a computationally cheap, non-parametric, optimization-free and data-driven method to draw internal parameters which lead to improved forecasting skill. We argue that good features, corresponding to good internal parameters, need to explore the expressive range of a given activation function. We consider here as an example the tanh activation function. To allow for good expressivity, good parameters should neither map the training data into the linear range of the activation function nor into the saturated range in which different inputs cannot be discerned. This leads us to a definition of *good features* corresponding to *good internal parameters*. We find that the set of good internal parameters is non-convex but can be expressed as a union of convex sets. To sample from a convex set we employ a hit-and-run algorithm [47, 50]. Hit-and-run algorithms are a class of Markov chain Monte Carlo samplers known for their fast mixing times in convex regions [32, 34, 25]. In recent years, hit-and-run algorithms have also been analyzed for sampling nonconvex regions [7, 22, 2].
The hit-and-run algorithms we develop allow us to generate any desired ratio of good features. We show in numerical experiments that the ratio of good features as defined by our criterion controls the forecasting capabilities of the learned surrogate map. Moreover, we illustrate the mechanism by which the least-square solution enhances good features and suppresses bad ones.
A secondary objective of our work is to demonstrate that a random feature map typically achieves superior forecasting skill when compared to a neural network of the same architecture, trained with gradient descent, while being several orders of magnitude cheaper computationally. We show that the bad performance of the single-layer feedforward network can be attributed to the optimization procedure not being constrained to the set of good internal parameters. This can potentially lead to new design and improved training schemes for more complex network architectures.

The outline of this paper is as follows. In Section 2 we describe the setup of data-driven surrogate maps for dynamical systems and how to assess their forecasting capabilities. Section 3 introduces random feature maps and illustrates how the choice of the internal weights affects the forecasting capabilities of the associated trained surrogate maps. Section 4 defines the set of good internal parameters and introduces hit-and-run algorithms to uniformly sample from this set. Section 5 illustrates the effect of sampling from the good set of internal parameters on the forecasting skill and how the least-square training learns to distinguish good features associated with good parameters from those associated with internal parameters drawn from the complement of the good set. Section 6 compares random feature maps with single-layer feedforward networks in which the internal parameters are learned using backpropagation, and establishes that random feature maps with good parameters far outperform the single-layer feedforward neural network. Section 7 shows that random feature maps with good internal parameters reproduce the long-time statistical behaviour of the underlying dynamical system more accurately than those without. Finally, we conclude in Section 8 with a summary of our results and possible future extensions.

2. Dynamical setup

We consider the forecasting problem for chaotic dynamical systems. Consider the following D-dimensional continuous-time dynamical system,

$$\dot{\mathbf{u}} = \mathcal{F}(\mathbf{u}), \tag{1}$$

with initial data $\mathbf{u}(\mathbf{0}) = \mathbf{u}_0$, which we observe at discrete times $t_n = n\Delta t$ for $n = 0, 1, \ldots, N$. We consider here the case when the full D-dimensional state is observed and observations are noise-free. For the treatment of noisy observations and partial observations see [16, 15]. We view the dynamical system of these observations

in terms of a discrete propagator map,

$$\mathbf{u}_{n+1} = \Psi_{\Delta t}(\mathbf{u}_n). \tag{2}$$

The aim of data-driven modelling is to construct a surrogate map $\hat{\Psi}_{\Delta t}$ from the training data given by the observations that well approximates the true propagator map $\Psi_{\Delta t}$ of (2). In the following we denote variables associated with the surrogate map with a hat, and write the learned surrogate dynamical system as

$$\hat{\mathbf{u}}_{n+1} = \hat{\Psi}_{\Delta t}(\hat{\mathbf{u}}_n), \tag{3}$$

with initial data $\hat{\mathbf{u}}_0 = \mathbf{u}_0$. Throughout this work we use the $D = 3$-dimensional Lorenz-63 system [30, 31] with $\mathbf{u} = (x, y, z)$ and

$$\begin{aligned} \dot{x} &= 10(y - x), \\ \dot{y} &= x(28 - z) - y, \\ \dot{z} &= xy - \frac{8}{3}z, \end{aligned} \tag{4}$$

as the underlying continuous dynamical system (1). The Lorenz-63 system is chaotic with a positive Lyapunov exponent of $\lambda_{\max} \approx 0.91$ [49]. We generate independent training and validation data, $\mathbf{u}_n$ and $\mathbf{u}_n^{\text{validation}}$, sampled at $\Delta t = 0.02$ by randomly selecting independent initial conditions $\mathbf{u}_0$ and $\mathbf{u}_0^{\text{validation}}$, respectively. We discard an initial transient dynamics of 40 time units to ensure that the dynamics has settled on the attractor.

To test the predictive capability of a surrogate model, we apply it to unseen validation data and define the forecast time τ_f associated with the surrogate model,

$$\tau_f = \inf\left\{ t_n \lambda_{\max} : \frac{\|\hat{\mathbf{u}}_n^{\text{validation}} - \mathbf{u}_n^{\text{validation}}\|_2^2}{\|\mathbf{u}_n^{\text{validation}}\|_2^2} > \theta \right\}. \tag{5}$$

The forecast time is measured in Lyapunov time units and measures when the prediction of the learned surrogate map (3), initialized at $\hat{\mathbf{u}}_0^{\text{validation}} = \mathbf{u}_0^{\text{validation}}$, significantly deviates from the true validation trajectory $\mathbf{u}_n^{\text{validation}}$. We employ here an error threshold of $\theta = 0.05$.

3. Random feature maps

We consider random feature maps to learn the surrogate map (3) with

$$\hat{\Psi}_{\Delta t}(\mathbf{u}) = \mathbf{W}\sigma(\mathbf{W}_{\text{in}}\mathbf{u} + \mathbf{b}_{\text{in}}), \tag{6}$$

where $\mathbf{u}$ is the D-dimensional state vector, $\mathbf{W}_{\text{in}} \in \mathbb{R}^{D_r \times D}$ is the internal weight matrix, $\mathbf{b}_{\text{in}} \in \mathbb{R}^{D_r}$ the internal bias and $\mathbf{W} \in \mathbb{R}^{D \times D_r}$ the outer weight matrix. The nonlinear activation function σ is applied component wise and we choose here $\sigma = \tanh$. Random features are characterized by the internal weights $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ being drawn before training from a prescribed distribution $p(w_{\text{in}}, b_{\text{in}})$. The internal weights remain fixed and are not learned as it would be the case for a single-layer feedforward network which has the same architecture as in (6). Random feature maps can hence be seen as a linear combination of features which are the components of the D_r-dimensional random features vectors

$$\boldsymbol{\phi} = \sigma(\mathbf{W}_{\text{in}}\mathbf{u} + \mathbf{b}_{\text{in}}). \tag{7}$$

The matrix $\mathbf{W}$, controlling the linear combinations of the features, is learned from training data $\mathbf{U} \in \mathbb{R}^{D \times N}$, the columns of which are the observations $\mathbf{u}_n$, $n = 1, \ldots, N$ of the system (2). We do so by solving the following regularized optimization problem,

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}; \mathbf{U}, \mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}}), \tag{8}$$

with loss function

$$\mathcal{L}(\mathbf{W}; \mathbf{U}, \mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}}) = \|\mathbf{W}\boldsymbol{\Phi}(\mathbf{U}) - \mathbf{U}\|^2 + \beta\|\mathbf{W}\|^2. \tag{9}$$

Here $\|\cdot\|$ denotes the Frobenius norm, $\beta > 0$ is a regularization hyperparameter, and $\boldsymbol{\Phi}(\mathbf{U})$ is the feature matrix whose n-th column is given by,

$$\boldsymbol{\phi}(\mathbf{u}_{n-1}) = \tanh(\mathbf{W}_{\text{in}}\mathbf{u}_{n-1} + \mathbf{b}_{\text{in}}). \tag{10}$$

The solution of the optimization problem (8) is given explicitly by linear ridge regression as

$$\mathbf{W}^* = \mathbf{U}\mathbf{\Phi}(\mathbf{U})^\top(\mathbf{\Phi}(\mathbf{U})\mathbf{\Phi}(\mathbf{U})^\top + \beta\mathbf{I})^{-1}. \tag{11}$$

The low training cost of random feature maps makes them a very attractive architecture. Training reduces to the determination of the $D_r D$ outer weights $\mathbf{W}^*$ by means of an explicit analytical formula once the inner weights have been set to their fixed values. The total number of parameters for random feature maps is $D_r(2D+1)$ which scales linearly in the feature dimension D_r unlike other architectures such as echo state networks [20, 21] and long short term memory (LSTM) networks [19].

3.1. **The effect of the internal weights on the performance of random feature maps.** Random feature maps enjoy the universal approximation property [44, 3]. This means that for a given feature dimension D_r there exist internal weights such that the random feature map approximates any continuous function. The approximation can get arbitrarily close for increasing feature dimension. The universal approximation property, however, does not guide practitioners on how to find the internal weights $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ which allow for such an approximation. The main objective of this paper is to sample the internal parameters in a way that increases the forecasting skill of the random feature maps when compared to the usually employed data-uninformed random draws from a specified distribution such as a Gaussian or a uniform distribution.

Indeed, the forecasting skill of a learned random feature map (3) sensitively depends on the internal weights. To illustrate the effect of the hyperparameters $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ on the forecast time τ_f we uniformly sample $\mathbf{W}_{\text{in}}$ and $\mathbf{b}_{\text{in}}$ from the intervals $[-w, w]$ and $[-b, b]$, respectively, with $(w, b) \in (0, w_{\max}) \times (0, b_{\max})$. In particular, we use 30×30 regular grid points over $(0, w_{\max}) \times (0, b_{\max})$ with $w_{\max} = 0.4$ and $b_{\max} = 4.0$, and probe the statistics by generating $M = 100$ feature maps for each grid point, while keeping the training data and the validation data fixed for all realizations to focus on the effect of the internal weights. We fix the feature dimension at $D_r = 300$ and the regularization parameter at $\beta = 4 \times 10^{-5}$. Figure 1 shows a contour plot of the mean and the standard deviation of the forecast time τ_f over the domain of the internal weights. We can clearly see that certain regions in the hyperparameter space are associated with good performance with mean forecast times $\tau_f > 4$ while other regions produce poor mean forecast times. Moreover, regions in the hyperparameter space corresponding to high mean forecast times τ_f may have large variance.

Ideally, we would like parameters which have both, high mean forecast time and low variance so that the performance is not dependent on the particular training data used. It is clear that if the internal weights $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ are chosen sufficiently small, the associated features (7) are essentially linear with $\boldsymbol{\phi} \approx \mathbf{W}_{\text{in}}\mathbf{u} + \mathbf{b}_{\text{in}}$ for all input data $\mathbf{u}$. This would reduce the random feature maps to linear models which are known to be incapable of modelling nonlinear chaotic dynamical systems [11, 4]. On the other extreme, for sufficiently large internal weights a tanh-activation function saturates, and one obtains $\boldsymbol{\phi} \approx \pm 1$ independent of the input data $\mathbf{u}$, severely decreasing the expressivity of the random feature map. This suggests that one should choose internal weights which sample the tanh-activation function in its nonlinear non-saturated range. This is illustrated in Figure 2. We shall call features *linear*, if for all data $\mathbf{u}$ the argument of the tanh-activation function lies within the interval centred around the origin in $[-L_0, L_0]$. Those features obtained by the tanh-activation function that are approximately ± 1 for all input data $\mathbf{u}$, i.e. where the arguments of the tanh-activation function lie in either of two unbounded sets $(-\infty, -L_1], [L_1, +\infty)$, we label *saturated* features. Those features which for all input data are neither linear nor saturated, i.e. for which the argument of the tanh-activation function lies in either of the two intervals $(-L_1, -L_0)$ or (L_0, L_1), are labelled *good* features. We use $L_0 = 0.4$ and $L_1 = 3.5$ to define good, linear and saturated features throughout this paper.

We now illustrate the detrimental effect of linear and saturated features on the forecasting skill of random features. From the random feature maps that were used in Figure 1, we select those that lead to particularly large forecast times $\tau_f > 8$ and those that lead to particularly low forecast times $\tau_f < 0.5$. For each of those feature maps we determine the fraction of features that correspond to (a) the linear region $[-L_0, L_0]$, (b) the saturated region $(-\infty, -L_1] \cup [L_1, +\infty)$, and (c) the good region $(-L_1, -L_0) \cup (L_0, L_1)$, averaged over input data $\mathbf{u}_n$. We denote these fractions by p_l, p_s and p_g, respectively. We consider $5,000$ randomly selected data points $\mathbf{u}_n$ on the attractor of the Lorenz-63 system to estimate these fractions. For each group we randomly select 500 samples from the $90,000$ random feature maps used in Figure 1. In Figure 3 we show the histograms of the fractions p_g, p_l and p_s for these two groups. The group with low forecast times $\tau_f < 0.5$ contains a significantly higher fraction of linear or saturated features compared to the group with

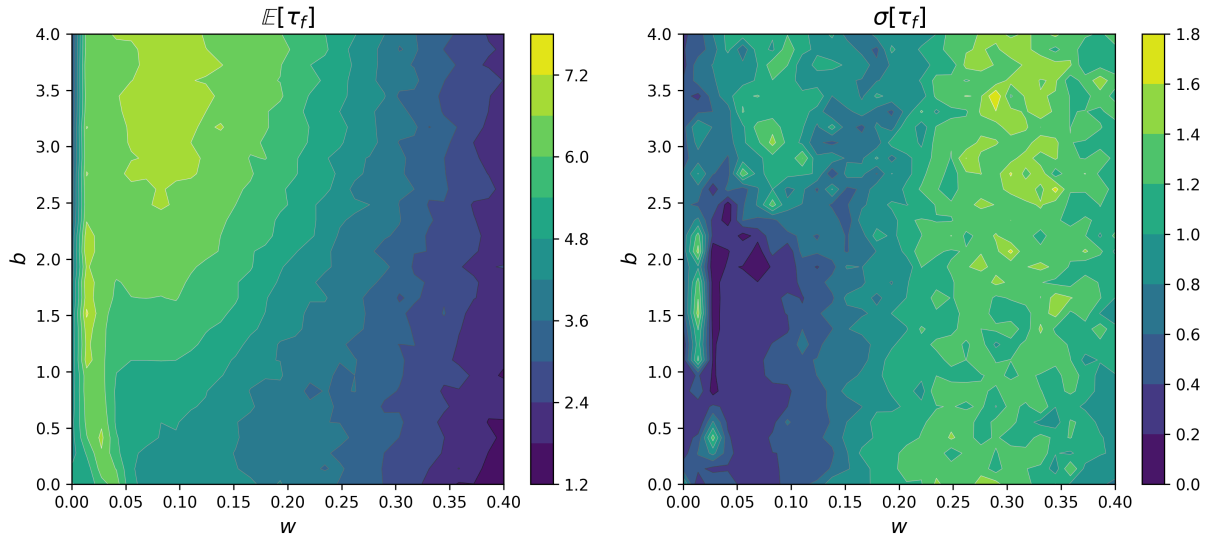

FIGURE 1. Contour plots of the mean and standard deviation of the forecast time τ_f computed using $\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}}$ sampled uniformly from intervals of variable size $[-w, w]$ and $[-b, b]$ respectively. Samples were drawn for grid points (w, b) on a 30×30 regular grid over the domain $(0, 0.4) \times (0, 4.0)$. Averages are taken over $M = 100$ realizations per grid-point (w, b), for a feature dimension $D_r = 300$, training data length $N = 20,000$ and regularization parameter $\beta = 4 \times 10^{-5}$, using the same training and validation data for ecah realization.

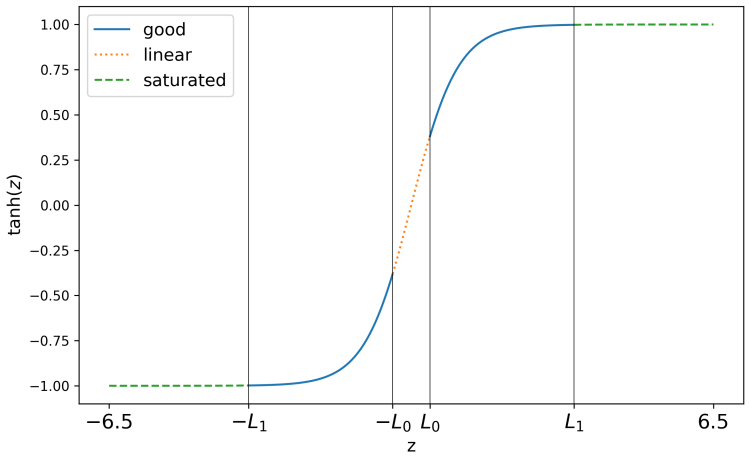

FIGURE 2. Domain and range of features produced by a tanh-activation function with $L_0 = 0.4$ and $L_1 = 3.5$, leading to linear, saturated and good features.

large forecast times $\tau_f > 8$. Conversely, the group with large forecast times $\tau_f > 8$ contains a significantly higher fraction of good features compared to the group with low forecast times $\tau_f < 0.5$. This confirms our hypothesis that good forecast skill is associated with an abundance of good features and a lack of linear and saturated features. We remark that the pronounced peak at $p_s = 0$ is a sampling effect: when sampling uniformly from the grid $(0, w_{\max}) \times (0, b_{\max})$ with $w_{\max} = 0.4$ and $b_{\max} = 4.0$, it is much more likely to draw parameters which correspond to non-saturated features. Such random feature map samples are much more likely to have higher forecast times and hence are concentrated entirely in the $\tau_f > 8$ group.

Figure 4 shows the mean forecast time $\mathbb{E}[\tau_f]$ as a function of the fractions of good, linear and saturated features, for the same random feature maps as shown in Figure 3. It is clearly seen that the forecast skill, as measured by the mean forecast time, monotonically improves for increasing number of good features. Similarly, the forecast skill monotonically degrades with increasing number of saturated features. The effect of the linear features is less obvious for the uniformly initialized random feature maps of Figure 1. However, as we will see later (cf. Figure 12), linear and saturated features have the same negative effect on the mean forecast time.

In the following Section we develop a computationally cheap algorithm to sample from the set of good weights, and show in Section 5 how this increases the forecasting skill of random feature maps.

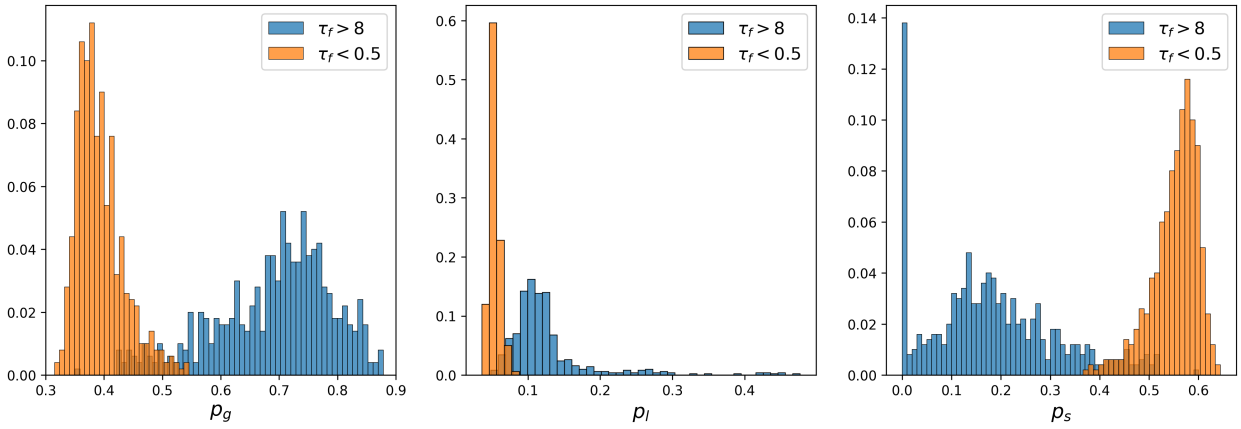

FIGURE 3. Empirical histograms of average fractions of good, linear and saturated features, p_g, p_l and p_s, respectively, for random feature maps corresponding to two groups: large forecast times with $\tau_f > 8$ and low forecast times with $\tau_f < 0.5$. The random feature maps are the same as those used in Figure 1. Each group contains 500 samples and the histograms depict the probability of having a certain value of the respective fractions in each group.

4. How to sample good internal weights

We would like our random feature maps to produce good features $\boldsymbol{\phi}(\mathbf{u}) = \mathbf{W}_{\text{in}}\mathbf{u} + \mathbf{b}_{\text{in}}$ by restricting $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ to be neither linear nor saturated for all training data $\mathbf{u}_n$. To that end, we select $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ such that

$$L_0 < |\mathbf{W}_{\text{in}}\mathbf{u}_n + \mathbf{b}_{\text{in}}| < L_1, \qquad \forall\, n = 0, 1, \dots N. \tag{12}$$

The lower bound L_0 controls the linear features and the upper bound L_1 controls the saturated features (cf. Figure 2). Note that (12) is a vector inequality and is equivalent to D_r scalar inequalities. Denoting the i-th row of $\mathbf{W}_{\text{in}}$ with $\mathbf{w}_i^{\text{in}}$ and the i-th entry of $\mathbf{b}_{\text{in}}$ with b_i^{in}, for each $i \in \{1, 2, \dots D_r\}$ we require

$$L_0 < |\mathbf{w}_i^{\text{in}} \cdot \mathbf{u}_n + b_i^{\text{in}}| < L_1, \qquad \forall\, n = 0, 1, \dots N. \tag{13}$$

Definition 4.1. *We call the i-th row $(\mathbf{w}_i^{\text{in}}, b_i^{\text{in}})$ of the internal parameters $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ good if it satisfies (13). Similarly, we call $(\mathbf{w}_i^{\text{in}}, b_i^{\text{in}})$ linear if*

$$|\mathbf{w}_i^{\text{in}} \cdot \mathbf{u}_n + b_i^{\text{in}}| \leq L_0, \qquad \forall\, n = 0, 1, \dots N, \tag{14}$$

and we call $(\mathbf{w}_i^{\text{in}}, b_i^{\text{in}})$ saturated if

$$|\mathbf{w}_i^{\text{in}} \cdot \mathbf{u}_n + b_i^{\text{in}}| \geq L_1, \qquad \forall\, n = 0, 1, \dots N. \tag{15}$$

For a streamlined discussion we call the i-th column of the outer weight matrix $\mathbf{W}$, good if the associated i-th row of the matrix of internal weights $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ is good and so on.

This categorization of rows of the internal parameters is useful for investigating the effects of different realizations of the random feature map on its forecasting skill. Note that this is not an exhaustive classification

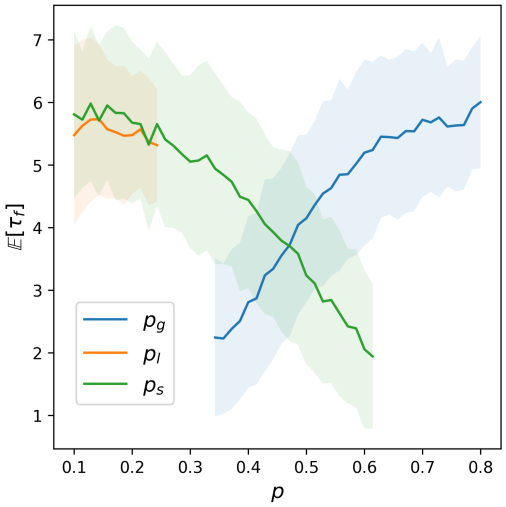

FIGURE 4. Mean forecast time $\mathbb{E}[\tau_f]$ as a function of the fraction p of good, linear and saturated features, respectively. The random feature maps are the same as those used in Figure 1. The mean forecast times are computed over bins $[p-\Delta p, p+\Delta p]$ with $\Delta p = 0.001$. The shaded region delineates one standard deviation from the mean. We only report on bins which contain more than 100 samples.

since there exist rows that satisfy different inequalities for different observations $\mathbf{u}_n$ and do not satisfy (13) for the whole data set $\mathbf{U}$. Although not considered here, such *mixed* rows may be an interesting topic for further exploration. The distinction into "good" (nonlinear, non-saturated) and "bad" (linear or saturated) parameters only pertains to the task of longest possible sequential forecasting. For example, for the task of predicting the state for only a single step, all parameters, good or bad, perform equally well.

We denote the set of good internal weights satisfying (12) by Ω_g. The solution set Ω_g is not convex, but can be written as the disjoint union of two convex sets with

$$\Omega_g = S_- \cup S_+, \tag{16}$$

where

$$S_- = \{(\mathbf{w}, b) \in \mathbb{R}^{D+1} : -L_1 < \mathbf{w} \cdot \mathbf{u}_n + b < -L_0 \ \ \forall\, n = 0, 1, \ldots, N\}, \tag{17}$$

$$S_+ = \{(\mathbf{w}, b) \in \mathbb{R}^{D+1} : +L_0 < \mathbf{w} \cdot \mathbf{u}_n + b < +L_1 \ \ \forall\, n = 0, 1, \ldots, N\}. \tag{18}$$

Since the convex subsets are reflections of each other with

$$S_- = -S_+, \tag{19}$$

it suffices to sample from only one of these convex sets and then uniformly sample the sign of the internal weights to sample from Ω_g. Hence, the sampling problem is effectively a convex problem. Analogously, we define Ω_l and Ω_s to be the solution sets to the problems (14) and (15) respectively, and again sampling these sets are convex problems. We will see that the data-informed constraint for good parameters (12) allows for sufficient variability in the features (cf. Section 4.3).

We present, in the next two subsections, algorithms to effectively sample from the good set Ω_g to enhance the forecasting capabilities of random feature maps, as well as from the sets Ω_l and Ω_s to illustrate their effect on random feature maps in Section 5. A naive choice of sampling algorithm would be to uniformly sample from the D-dimensional hypercube with the 2^D corners defined by the extremal training data points, and checking the inequality (12), if we want to sample from Ω_g, let's say. This, however, is computationally very costly as typically the solution set only occupies a small region within that hypercube.

Our aim is to sample good parameters. There is no *a priori* reason to sample *uniformly* from the convex set of good parameters. However, as we will see, uniform sampling lends itself to an efficient implementation.

Furthermore, we stress that sampling uniformly from the parameter space does not imply uniform sampling in the feature space.

We consider a standard hit-and-run algorithm sampling from Ω_g in Section 4.1 and then present a faster, more efficient hit-and-run algorithm to sample from an equivalent restricted solution set in Section 4.2.

4.1. **Standard hit-and-run sampling of good internal parameters.** We now describe a computationally cheap and easy to implement numerical algorithm to uniformly sample from the solution sets Ω_g, Ω_l and Ω_s. We shall employ hit-and-run algorithms [47, 50]. To uniformly sample a set Ω with hit-and-run, one starts from a feasible point inside the set, considers the line through that point in a randomly chosen direction, and then randomly picks a point on the intersection of that line and the set Ω as a new point. This process is then repeated to generate further samples. For convex sets the hit-and-run samples converge to uniform samples in total variation distance. The convergence depends polynomially on the number of iterations and dimension with the polynomial dependence on dimension being of low order [32, 2, 33]. This and the fast mixing properties make hit-and-run algorithms an attractive method to uniformly sample from Ω_g, Ω_l and Ω_s.

We sample the augmented internal weight matrix $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ row by row. Each sample lies then in a $D+1$-dimensional search space for $(\mathbf{w}_i^{\text{in}}, b_i^{\text{in}})$. Due to (19) it suffices to sample from S_+. In order to perform hit-and-run, given a point, we need to efficiently determine if it lies in S_+. Focusing on a convex conical subset of S_+, it turns out that we can determine if a point belongs to S_+ by checking just two inequalities. Define the convex cone

$$V(\mathbf{s}, b) = \{(\mathbf{w}, b) : \text{sgn}(\mathbf{w}_i) \in \{\mathbf{s}_i, 0\} \ \ \forall\, i = 1, 2, \ldots, D\}, \tag{20}$$

where $\mathbf{s}$ is a D-dimensional sign vector with entries ± 1 labelling the 2^D corners of a D-dimensional hypercube. To control the range of the training data set, we further define the vectors $\mathbf{x}_{\mp}(\mathbf{s}) \in \mathbb{R}^D$ as

$$\begin{aligned} \mathbf{x}_{-,i}(\mathbf{s}) &= \begin{cases} \min\limits_{1\le n\le N} \mathbf{u}_{n,i}, & \text{if } \mathbf{s}_i = 1 \\ \max\limits_{1\le n\le N} \mathbf{u}_{n,i}, & \text{otherwise} \end{cases} \\ \mathbf{x}_{+,i}(\mathbf{s}) &= \begin{cases} \max\limits_{1\le n\le N} \mathbf{u}_{n,i}, & \text{if } \mathbf{s}_i = 1 \\ \min\limits_{1\le n\le N} \mathbf{u}_{n,i}, & \text{otherwise,} \end{cases} \end{aligned} \tag{21}$$

where $\mathbf{u}_{n,i}$ is the i-th entry of the n-th training data point. Now for $(\mathbf{w}, b) \in V(\mathbf{s}, b)$ we have,

$$\begin{aligned} &\max_{1\le n\le N} \ (\mathbf{w}\cdot\mathbf{u}_n + b) \le \mathbf{w}\cdot\mathbf{x}_+(\mathbf{s}) + b, \\ &\text{and} \\ &\min_{1\le n\le N} \ (\mathbf{w}\cdot\mathbf{u}_n + b) \ge \mathbf{w}\cdot\mathbf{x}_-(\mathbf{s}) + b. \end{aligned} \tag{22}$$

Therefore, for $(\mathbf{w}, b) \in V(\text{sgn}(\mathbf{w}), b)$, we have $(\mathbf{w}, b) \in S_+$ if

$$\begin{aligned} &\mathbf{w}\cdot\mathbf{x}_-(\text{sgn}(\mathbf{w})) + b > L_0, \\ &\text{and} \\ &\mathbf{w}\cdot\mathbf{x}_+(\text{sgn}(\mathbf{w})) + b < L_1 \end{aligned} \tag{23}$$

The feasibility inequalities (23) simply check if the internal weights $(\mathbf{w}, b)$ map the training data into the smallest D-dimensional hypercube that contains the training data.

To initialize the hit-and-run algorithm with a feasible point we choose $(\mathbf{w}, b) = (\mathbf{0}, b_0) \in S_+$ for $b_0 \in (L_0, L_1)$. To determine the line segments inside S_+ we use bisection together with the feasibility criterion (23). The hit-and-run algorithm requires a few iterations to ensure that the samples become independent of the initial feasible point $(\mathbf{w}, b) = (\mathbf{0}, b_0)$.

We summarize this hit-and-run algorithm for randomly generating uniform samples from Ω_g in Algorithm 1.

Algorithm 1 Standard hit-and-run sampling for a good row

Input: data $\mathbf{U}$.
Choose number of decorrelation iterations $K \in \mathbb{N}$ and $L_0, L_1 \in \mathbb{R}_{>0}$.
Sample b uniformly from (L_0, L_1).
$\mathbf{w} \leftarrow \mathbf{0}$.
$k \leftarrow 0$.
while $k < K$ **do**
    Randomly select a unit vector $\mathbf{d} \in \mathbb{R}^{D+1}$.
    Determine A = the maximal line segment passing through $(\mathbf{w}, b)$ along direction $\mathbf{d}$ and contained within S_+, using the feasibility criterion (23) and the bisection method.
    Uniformly sample $(\mathbf{w}', b')$ from A.
    $(\mathbf{w}, b) \leftarrow (\mathbf{w}', b')$
    $k \leftarrow k + 1$.
end while
Uniformly sample a scalar z from $\{-1, 1\}$ to determine which set to sample from, S_- or S_+.
if $z = 1$ **then**
    $(\mathbf{w}, b)$ is our final good row sample.
else
    $(-\mathbf{w}, -b)$ is our final good row sample.
end if

4.2. **One-shot hit-and-run sampling.** We now present a reduced hit-and-run algorithm which operates on a smaller D-dimensional search space and does not require computationally costly bisection. This algorithm, which we will coin one-shot hit-and-run algorithm, produces independent samples without the need for sufficiently many iterations to ensure decorrelation from the fixed initial feasible point.

To generate good (or linear or saturated) random feature maps one can restrict the solution set Ω_g (or Ω_l or Ω_s) by first sampling b appropriately and then sampling $\mathbf{w}$ from a D-dimensional search space. For ease of presentation, we describe the algorithm for sampling from the good set Ω_g. We sample b uniformly from the interval (L_0, L_1). The weights $\mathbf{w}$ are then sampled from the restricted solution set $\Omega_g^R = S_+^R \cup S_-^R$ with

$$S_-^R = \{(\mathbf{w}, b) \in S_- : -L_1 < b < -L_0\}, \tag{24}$$

$$S_+^R = \{(\mathbf{w}, b) \in S_+ : +L_0 < b < +L_1\}. \tag{25}$$

Since $S_-^R = -S_+^R$, sampling from the nonconvex set Ω_g^R can again be done by sampling from the convex set S_+^R and then multiplying the sample with 1 or -1 uniformly randomly. This restriction allows us to perform hit-and-run sampling on a D-dimensional random convex set instead of a $(D+1)$-dimensional convex set. Note that fixing b is akin to shrinking the search space from S_+ to πS_+^R where π is the canonical projection: $\pi(\mathbf{w}, b) = \mathbf{w}$. Also note that we can partition πS_+^R according to

$$\pi S_+^R = \bigsqcup_{\mathbf{s} \in \{-1,1\}^D} (\pi S_+^R \cap V(\mathbf{s})), \tag{26}$$

where we use $\bigsqcup$ to denote almost disjoint union, and $V(\mathbf{s}) = \pi V(\mathbf{s}, b)$ are D-dimensional orthants. Let us randomly select a sign vector $\mathbf{s} \in \{-1, 1\}^D$ or equivalently pick the random convex subset $\pi S_+^R \cap V(\mathbf{s})$. Randomly choosing the sign vector or the corresponding convex subset is tantamount to assigning signs randomly to the entries of $\mathbf{w}$. In order to uniformly sample this conical subset we can pick a random direction $\mathbf{d}$ in the cone $V(\mathbf{s})$, determine the maximal line segment starting at the origin parallel to $\mathbf{d}$ that is contained in $\pi S_+^R \cap V(\mathbf{s})$ and uniformly sample a point on this line segment. Figure 5 shows a schematic for this one-shot hit-and-run algorithm. Since $\mathbf{x}_\pm(\mathrm{sgn}(\mathbf{w}))$ is constant for all $\mathbf{w} \in V(\mathbf{s})$, we can analytically determine the maximal line segment without having to resort to bisection. Moreover, the special structure of the cone lets us sample with a single iteration unlike the standard hit-and-run Algorithm 1. Thus the computation of the line segment in the solution set and the final sampling both happen in one shot and therefore the one-shot hit-and-run is much faster than its standard counterpart given by Algorithm 1.

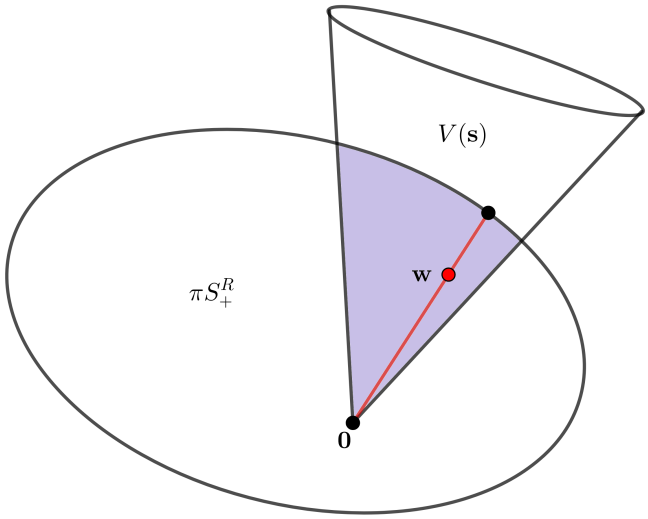

FIGURE 5. Schematic of the one-shot hit-and-run Algorithm 2. The weight point $\mathbf{0}$ is always an interior point of πS_+^R and the cone $V(\mathbf{s})$ is a D-dimensional orthant. The set πS_+^R is drawn as bounded here, but it may be unbounded depending on the training data $\mathbf{u}_n$.

Algorithm 2 summarizes the one-shot hit-and-run sampling of a good row. Note that, depending on the training data $\mathbf{U}$, it is possible for πS_+^R to be unbounded, which is why $+\infty$ appears in the algorithm. We can extend the notion of restriction to the coordinates of $\mathbf{w}$ as well by restricting the intervals where we are allowed to sample them from, which is akin to regularizing parameters in machine learning [24, 23, 14] but we do not consider such algorithms here. Obvious modifications of Algorithm 1 and Algorithm 2 let us sample linear and saturated rows which we refrain from describing here to avoid repetition.

Algorithm 2 One-shot hit-and-run sampling for a good row

Input: data $\mathbf{U}$.
Choose $L_0, L_1 \in \mathbb{R}_{>0}$.
Sample b uniformly from (L_0, L_1).
Select the sign vector $\mathbf{s}$ by uniformly generating D samples from $\{-1, 1\}$.
Randomly select a unit vector $\mathbf{d} \in V(\mathbf{s})$.
$a_0 \leftarrow 0$.
$a_1 \leftarrow \inf\left(\left\{\frac{L_0 - b}{\mathbf{d}\cdot\mathbf{x}_-(\mathbf{s})}, \frac{L_1 - b}{\mathbf{d}\cdot\mathbf{x}_+(\mathbf{s})}\right\} \cap (\mathbb{R}_{>0} \cup \{+\infty\})\right)$ with the convention $\inf \varnothing = +\infty$.
Sample a uniformly from (a_0, a_1).
Uniformly sample a scalar z from $\{-1, 1\}$ to determine which set to sample from, S_- or S_+.
if $z = 1$ **then**
    $(a\mathbf{d}, b)$ is our final good row sample.
else
    $(-a\mathbf{d}, -b)$ is our final good row sample.
end if

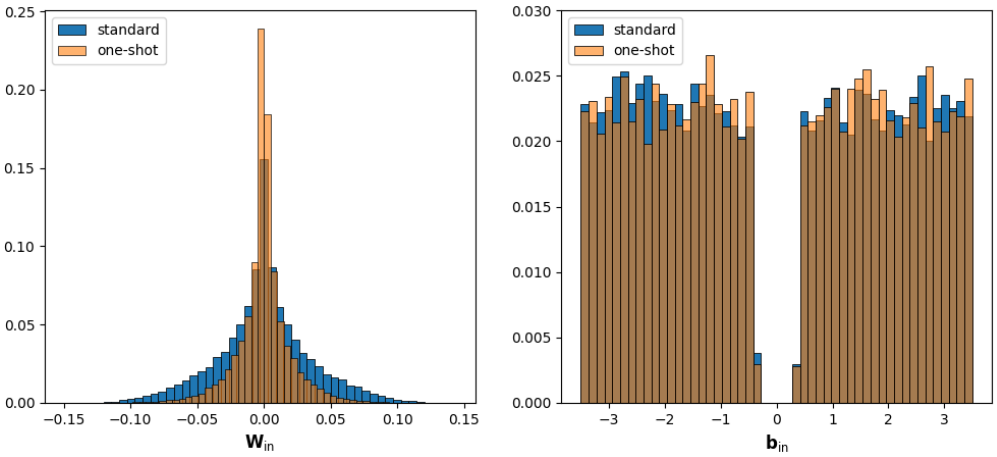

FIGURE 6. Empirical histograms for samples generated using standard and one-shot hit-and-run Algorithms 1 and 2, respectively. The left panel shows the distributions of the entries of $\mathbf{W}_{\text{in}}$ and the right panel shows the distributions of the entries of $\mathbf{b}_{\text{in}}$. For each algorithm $10,000$ rows of internal parameters were generated. The standard hit-and-run Algorithm 1 used $K = 10$ decorrelation iterations.

4.3. **Performance and comparison of the hit-and-run algorithms.** The two hit-and-run Algorithms 1 and 2 are designed to uniformly sample from the $D+1$-dimensional set of good rows Ω_g. This does not imply that the individual entries of $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ are uniformly distributed. The distributions for the entries of weights and biases sampled from the two hit-and-run algorithms are shown in Figure 6. For the standard hit-and-run Algorithm 1 it was found that $K = 10$ decorrelation steps were sufficient and results were very similar for $K = 100$ iterations. It is clearly seen that the distributions are far from being the usually employed uniform or Gaussian distribution. The distributions are very similar for both algorithms. In particular, the standard Algorithm 1 exhibits the same lack of biases b with small absolute value, as the one-shot hit-and-run Algorithm 2.

Whereas the one-shot hit-and-run algorithm excludes biases b with absolute values smaller than L_0 by design, this may seem surprising for the standard hit-and-run algorithm. This can be explained as follows. For $0 < b < L_0$ and $(\mathbf{w}, b) \in S_+$ we require that $\mathbf{w} \cdot \mathbf{u}$ lies in the positive interval $(L_0 - b, L_1 - b)$ for all training data $\mathbf{u}$. Since the directions of the vectors $\mathbf{u}$ in the training data are typically distributed over some range, $\mathbf{w} \cdot \mathbf{u}$ is typically not sign-definite for all data points $\mathbf{u}$. This implies that for all parameters in Ω_g we typically have $|b| > L_0$; a similar argument shows that typically $|b| < L_1$. Hence, for typical data $\mathbf{u}$, we have $\Omega_g^R = \Omega_g$ and the search space of the one-shot hit-and-run Algorithm 2 is the same as that of the standard Algorithm 1.

For the weights and biases which were obtained by sampling uniformly from an interval as in Figure 1, we checked that the weights corresponding to high forecasting skill indeed all satisfy our criterion of being good rows (12). This highlights the advantage of our non-parametric sampling over sampling strategies involving a set of parametrized distributions.

The hit-and-run Algorithms 1 and 2 were designed to uniformly sample from the set Ω_g. This does not imply, however, that $\mathbf{w}_{\text{in}}\mathbf{u} + \mathbf{b}_{\text{in}}$ is uniformly distributed in the interval $(-L_1, -L_0) \cup (L_0, L_1)$. To quantify the occupied range of random features we introduce the following notation. A sample $(\mathbf{w}_i^{\text{in}}, b_i^{\text{in}})$ produces outputs the absolute values of which lie in the interval $[m_i, M_i]$, i.e.

$$\begin{aligned} m_i &= \min_{1 \le n \le N} |\mathbf{w}_i^{\text{in}} \cdot \mathbf{u}_n + b_i^{\text{in}}|, \\ M_i &= \max_{1 \le n \le N} |\mathbf{w}_i^{\text{in}} \cdot \mathbf{u}_n + b_i^{\text{in}}|. \end{aligned} \tag{27}$$

The effective range $\mathcal{R}$ of a random feature map vector can then be defined as

$$\mathcal{R} = \frac{1}{D_r} \sum_{i=1}^{D_r} (M_i - m_i). \tag{28}$$

By the central limit theorem, for $D_r \gg 1$ the effective range (28) approaches a normally distributed random variable. Figure 7 shows the histogram of the range $\mathcal{R}$ for the standard and the one-shot hit-and-run algorithms when only good features are used with $p_g = 1$. As expected both algorithms generate near-Gaussian distributions of $\mathcal{R}$.

The standard hit-and-run algorithm generates a wider range with $\mathbb{E}[\mathcal{R}] = 1.0$ and standard deviation $\sigma[\mathcal{R}] = 0.03$ compared to the one-shot hit-and-run algorithm with $\mathbb{E}[\mathcal{R}] = 0.42$ and standard deviation $\sigma[\mathcal{R}] = 0.02$. One would like the range $\mathcal{R}$ to be as large as possible, assuming that higher variability in the features increases the expressivity of the random feature map, and hence the forecast skill. Figure 8 shows the histogram of the forecast times τ_f for the two algorithms. The standard hit-and-run algorithm exhibits a slightly better forecast skill as measured by an approximately 6% larger mean forecast time of $\mathbb{E}[\tau_f] = 5.4$ compared to that of the one-shot hit-and-run algorithm with $\mathbb{E}[\tau_f] = 5.1$, in accordance with our intuition that larger range is beneficial.

The one-shot hit-and-run Algorithm 2 is ~ 15 times faster than the standard hit-and-run Algorithm 1 (with 10 decorrelation steps) on an M1 CPU. Since the one-shot algorithm does not involve loops for each realization, it allows for efficient parallelization on GPUs. The standard hit-and-run algorithm, on the other hand, involves loops to execute the bisection, which prohibits effective parallelization on GPUs. As a result, the one-shot algorithm is ~ 825 faster than the standard algorithm on an A100 GPU. In applications we hence use the computationally more efficient one-shot hit-and-run Algorithm 2, given the almost equal performance in forecasting (cf.Figure 8).

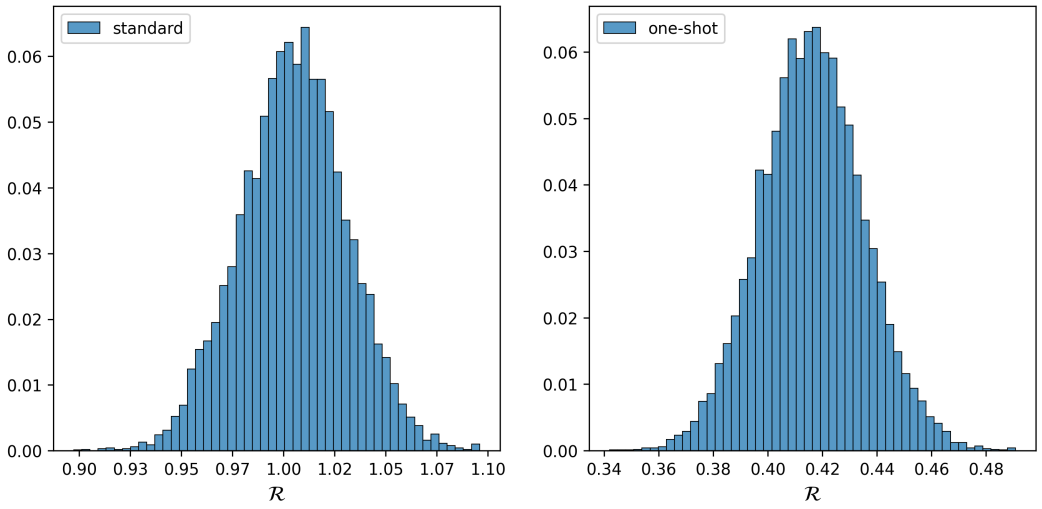

FIGURE 7. Empirical histogram of the effective range $\mathcal{R}$ of random feature maps when only good rows are used with $p_g = 1$ for the standard hit-and-run Algorithm 1 (left) and the one-shot hit-and-run Algorithm 2 (right). Each histogram represents $10,000$ samples differing in the internal weights, the training data and the validation data. The feature dimension is $D_r = 512$ and a regularization parameter of $\beta = 2.79 \times 10^{-5}$ is used with training data length $N = 20,000$. For the standard hit-and-run Algorithm 1, 10 decorrelation steps are used. For the standard hit-and-run algorithm we obtain a mean forecast time of $\mathbb{E}[\tau_f] = 1.0$ with $\sigma[\tau_f] = 0.03$. For the one-shot algorithm we obtain $\mathbb{E}[\tau_f] = 0.42$ with $\sigma[\tau_f] = 0.02$.

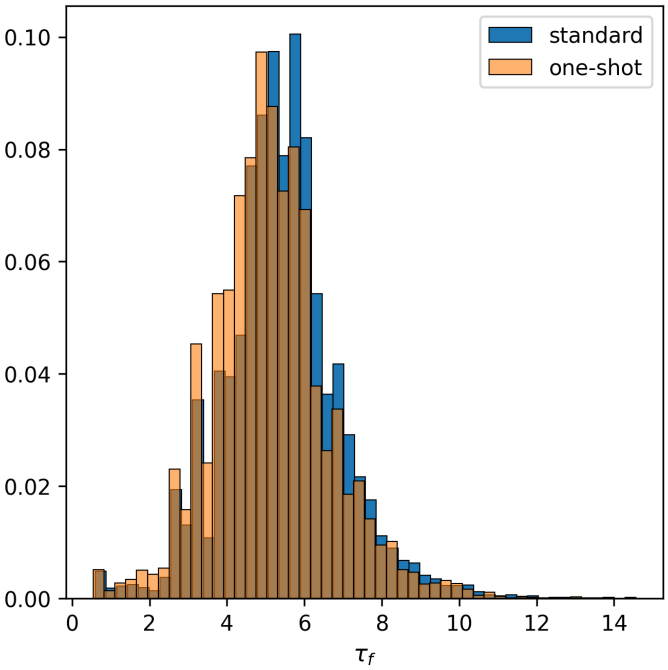

FIGURE 8. Empirical histogram of the forecast time τ_f when only good rows are used with $p_g = 1$ for the standard hit-and-run Algorithm 1 and the one-shot hit-and-run Algorithm 2. The same random feature maps are used as in Figure 7. For the standard hit-and-run algorithm we obtain the mean forecast time $\mathbb{E}[\tau_f] = 5.4$ with $\sigma[\tau_f] = 1.5$. For the one-shot algorithm we obtain $\mathbb{E}[\tau_f] = 5.1$ with $\sigma[\tau_f] = 1.5$. The random feature maps shown here are the same as those in Figure 7.

5. Results for forecasting individual trajectories

In this section we explore how increasing the number of good features improves the forecasting skill of a surrogate map for the Lorenz-63 system (4), and conversely explore the effect of linear and saturated features. To do so we define the number of good, linear and saturated features in a random feature vector of dimension D_r as

$$N_g = p_g D_r, \qquad N_l = p_l D_r, \qquad N_s = p_s D_r, \tag{29}$$

where the respective fractions satisfy $p_g + p_l + p_s = 1$. We construct random feature maps with internal weights $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ with specified fractions of good, linear or saturated rows using the one-shot hit-and-run Algorithm 2.

5.1. Effect of the quality of internal weights on the forecast time τ_f.

In this section we investigate how the forecasting skill of a random feature surrogate model (3) improves with increasing number of good rows N_g. We would like to have internal parameters resulting in large mean forecast times τ_f with relatively small standard deviations. For chaotic dynamical systems we expect a residual variance of the forecast time due to the sensitivity to small changes in the model: small changes in the internal parameters may cause the surrogate models to deviate from each other after some time.

We estimate the mean of the forecast time τ_f and its coefficient of variation as a function of the fraction of good features p_g, varying p_g from $p_g = 0$ with only bad features to $p_g = 1$ with only good features present. For each value of p_g we approximately uniformly distribute the remaining $(1-p_g)D_r$ features over the linear and saturated features with $p_l \approx p_s \approx (1-p_g)/2$. Note that we cannot always impose perfect equality since $N_g = p_g D_r$, $N_l = p_l D_r$ and $N_s = p_s D_r$ are integers. We use 51 equally spaced values of p_g in $[0, 1]$ and

compute averages over 500 realizations for each value of p_g, differing in the draws of the random internal weights, the training data and the validation data.

Figure 9 shows the dependence of the mean forecast time $\mathbb{E}[\tau_f]$ and the associated coefficient of variation $\sigma[\tau_f]/\mathbb{E}[\tau_f]$ on p_g for various values of the feature dimensions D_r and training data lengths N. It is clearly seen that increasing the number of good rows increases the mean forecast time and decreases the coefficient of variation, as desired. As expected, for fixed feature dimension D_r increasing the training data length N is beneficial. A training data length of $N = 78$ is too small to provide any meaningful forecasting skill with mean forecast times below one Lyapunov time for any value of p_g and significantly larger variance. Similarly, for fixed training data length N, increasing the feature dimension D_r is beneficial. The observation that, for fixed data length N, the mean forecast time $\mathbb{E}[\tau_f]$ saturates upon increasing p_g once a sufficiently large number of good features $N_g = p_g D_r$ are present, suggests that the distribution of the forecast time τ_f converges reflecting a residual uncertainty of the chaotic surrogate model. This is confirmed in Figure 10 where we see convergence of the empirical histograms of the forecast time for increasing values of D_r in the case when $p_g = 1$.

Figure 11 shows the dependency of the mean forecast time $\mathbb{E}[\tau_f]$ on the fraction of good rows p_g for different values of D_r. We can clearly see that beyond $N_g = p_g D_r = 256$ (indicated by the vertical line), the mean forecast time $\mathbb{E}[\tau_f]$ depends only on the number of good rows $N_g = p_g D_r$ and not on the overall feature dimension D_r. For smaller number of good rows $N_g < 256$ the mean forecast time depends on the feature dimension D_r with larger feature dimensions implying larger mean forecast times. This suggests that the number of good features N_g constitutes an effective feature dimension D_r^*, which controls the forecast skill of the learned surrogate model. This implies that on average the forecast time τ_f is the same for a random feature surrogate model of dimension D_r with only good features $p_g = 1$ as a surrogate map with a larger feature dimension αD_r with $\alpha > 1$ but only a fraction of $1/\alpha$ good rows. This is confirmed in Figure 12 which shows the empirical histogram of τ_f for fixed number of good features $N_g = p_g D_r = 1,024$. We compare the distribution of the forecast times for random feature maps with $D_r = 1,024$ and $p_g = 1$ to those with $D_r = 2,048$ and $p_g = 0.5$. We show examples when the remaining bad features are either equally distributed between linear and saturated features, or only linear or only saturated. The distributions for all three examples are very similar and match the one with the smaller feature dimension but same number of good features. This leads us to conclude that the number of good rows is the only determining factor for the distribution of τ_f (all other parameters being equal), and that linear and saturated rows are equally ineffective in terms of the forecasting skill.

We briefly discuss the effect of the regularization parameter β on the forecasting skill. We show in Figure 13 the mean forecast time $\mathbb{E}[\tau_f]$ and coefficient of variation $\sigma[\tau_f]/\mathbb{E}[\tau_f]$ as a function of p_g for a range of regularization parameters $\beta \in [2^{-25}, 2^{-13}]$. For fixed feature dimension $D_r = 300$, we see that $\beta = 2^{-19}$ is optimal within this range in terms of the mean forecast time (left panel) and the coefficient of variation (right panel) once sufficiently many good features are present with $p_g > 0.33$. Note that we had previously employed $\beta = 4 \times 10^{-5} \approx 2^{-14.6}$. The large difference in performance for different choices of the regularization parameter β makes clear that, if optimizing for performance, the regularization parameter has to be optimized. This could be achieved by line-search [37] or by Bayesian optimization [26]. Here we have refrained from fine-tuning the regularization parameter as our focus is the sampling algorithm and the effect of different types of features on the forecast skill.

5.2. **Effect of the quality of internal weights on the outer weights W.** In this section we explore how the nature of the internal weights affects the learned solutions of the ridge regression (11) which we denote simply as $\mathbf{W}$, dropping the star.

We begin by recording the Frobenius norm $\|\mathbf{W}\| = \mathrm{Tr}(\mathbf{W}^\top \mathbf{W})$ of the learned outer weights as a function of the fraction of good features p_g (bad features are again roughly equally distributed between linear and saturated features). Figure 14 shows the mean of $\|\mathbf{W}\|$ as a function of p_g on a log-log scale for the simulations used in Figure 9. It is seen that $\|\mathbf{W}\|$ is a decreasing function of the number of good features. The solution of the linear regression problem $\mathbf{W}$ minimizes the loss function (10). Once there are sufficiently many good features, the training data can be sufficiently well fit, decreasing the first term of the loss function. Further increasing the number of good features then allows for a decrease of the regularizing term of the loss function, leading to a decrease of $\|\mathbf{W}\|$. Assuming that the true one-step map $\Psi_{\Delta t}$ in (2)

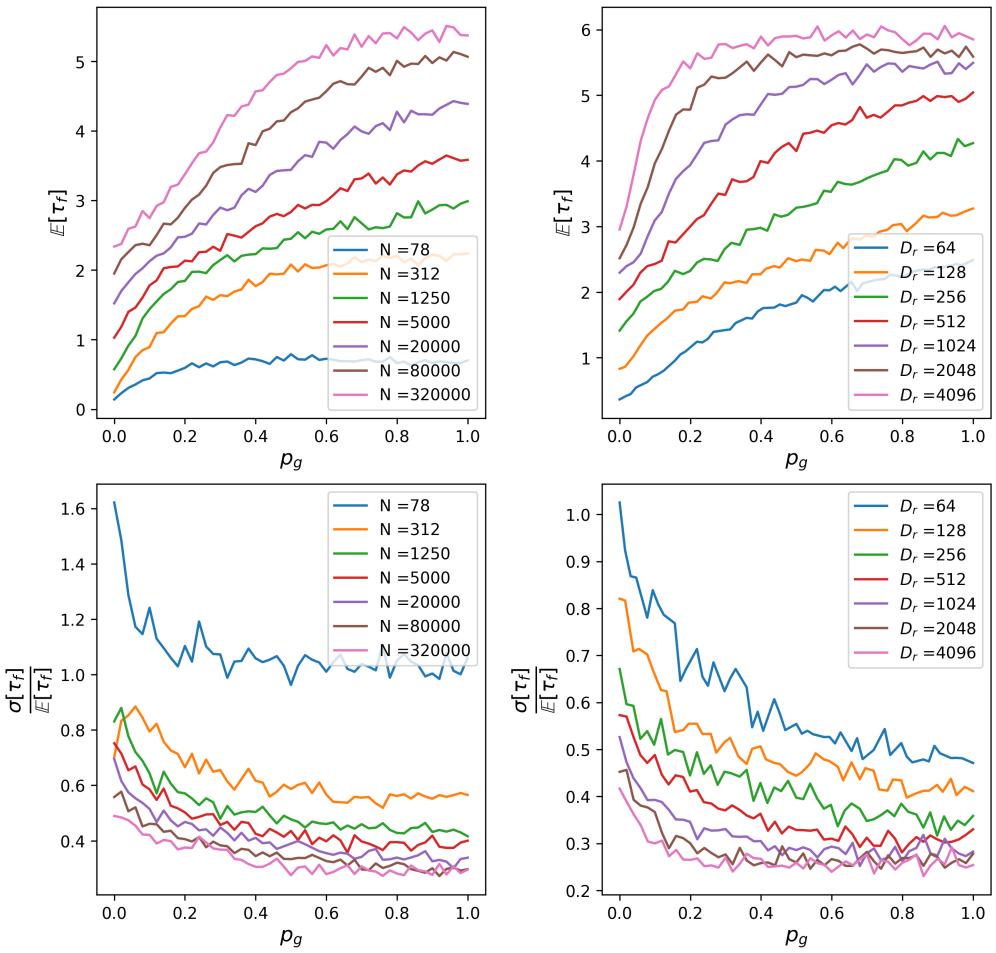

FIGURE 9. The top row depicts the mean of the forecast time $\mathbb{E}[\tau_f]$ as a function of the fraction of good features p_g. The bottom row depicts the coefficient of variation $\sigma[\tau_f]/\mathbb{E}[\tau_f]$ as a function of p_g. Along the first column the feature dimension $D_r = 300$ is kept constant, and along the second column the length of the training data set $N = 20,000$ is kept constant. Expectation are computed over 500 realizations of the internal parameters, the training data and testing data. A regularization parameter of $\beta = 4 \times 10^{-5}$ is employed.

lies in the domain of the random feature map (3) with infinitely many features, the first term of the loss function should scale with the usual Monte-Carlo estimate scaling of $\mathcal{O}(1/D_r)$, suggesting a scaling of the regularization term $\|\mathbf{W}\| \sim 1/\sqrt{D_r}$. In the right panel of Figure 14 we show that the mean of $\|\mathbf{W}\|$ roughly scales as $\|\mathbf{W}\| \sim 1/D_r^{0.54}$ when all the internal weights correspond to good features with $p_g = 1$, suggesting that the true one-step map can be well approximated by random features with a tanh-activation function. We remark that the Monte-Carlo scaling is valid for $D_r > 256$ only, i.e. provided sufficiently many good features are present.

We now investigate how the decrease in the outer weights $\mathbf{W}$ is distributed over the various features. We will see that the outer weights are learned to suppress the bad features provided there are sufficiently many good features allowing for a reduction of the loss function. Let us denote the i-th column of $\mathbf{W}$ by $\mathbf{W}^i$. The columns $\mathbf{W}^i$ are the weights attributed to the features produced by the i-th row of the internal weights $(\mathbf{w}_i^{\text{in}}, b_i^{\text{in}})$. We expect the outer weights corresponding to good rows to be significantly larger than those corresponding to bad rows.

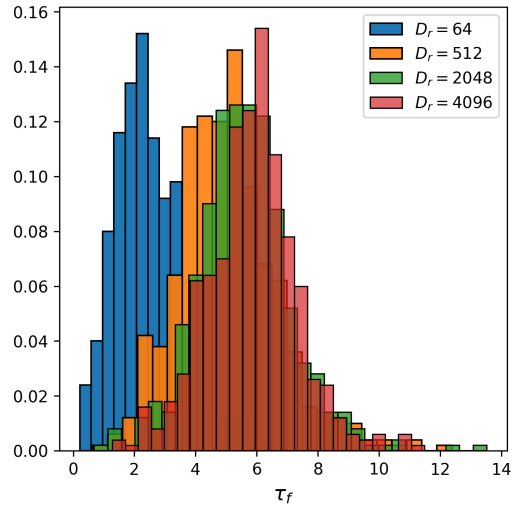

FIGURE 10. Empirical histogram of τ_f for different values of D_r when $p_g = 1$ for increasing feature dimension D_r. The same 500 realizations are used as in Figure 9 with $N = 20,000$.

z

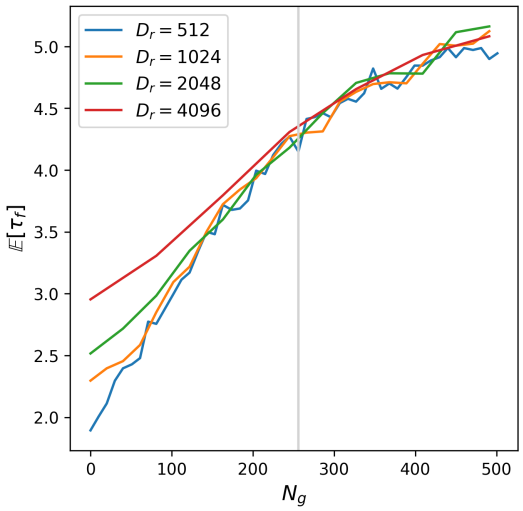

FIGURE 11. Forecast time mean $\mathbb{E}[\tau_f]$ as a function of good features $N_g = p_g D_r$. The vertical line demarcates $N_g = 256$. The range of N_g is restricted to $N_g \leq 512$, corresponding to $p_g = 1$ for the smallest value of the feature dimension $D_r = 512$. The same 500 realizations are used as in Figure 9 with $N = 20,000$.

To study the suppression of bad features by columns of $\mathbf{W}$ which have small norm, we design two sets of numerical experiments: one in which bad features are entirely comprised of linear features and one in which bad features are entirely comprised of saturated features.

In the first set we initialize a random feature map with $D_r = 300$ features consisting of only bad linear features. We then successively replace one linear feature by a good feature, i.e. replacing one inner linear weight row $(\mathbf{w}_i^{\text{in}}, b_i^{\text{in}})$ by a good row. At each step we record the corresponding linear regression solution $\mathbf{W}$. Figure 15 shows the normalized supremum norm of columns of $\mathbf{W}$ after $N_g = 10$, $N_g = 50$ and $N_g = 150$ bad linear features have been replaced by good features. The red dots signify columns which do not contain

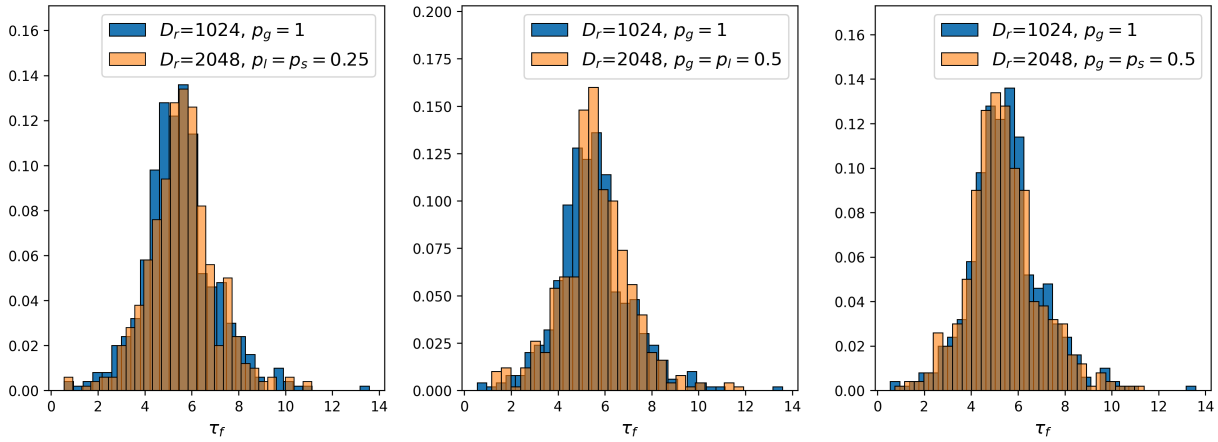

FIGURE 12. Empirical histogram of the forecast time τ_f for $D_r = 1,024$ and $D_r = 2,048$. In each case the number of good rows is $N_g = 1,024$. For $D_r = 2,048$ we show results for an equal number of linear and saturated features with $p_l = p_s = 0.25$ (left), for only linear bad features with $p_l = 0.5, p_s = 0$ (middle) and for only saturated bad features with $p_s = 0.5, p_l = 0$ (right) for $D_r = 2,048$. We used 500 realizations differing in the random draws of the internal parameters, the training data and the validation data. We employed a regularization parameter of $\beta = 4 \times 10^{-5}$ and used training data of length $N = 20,000$.

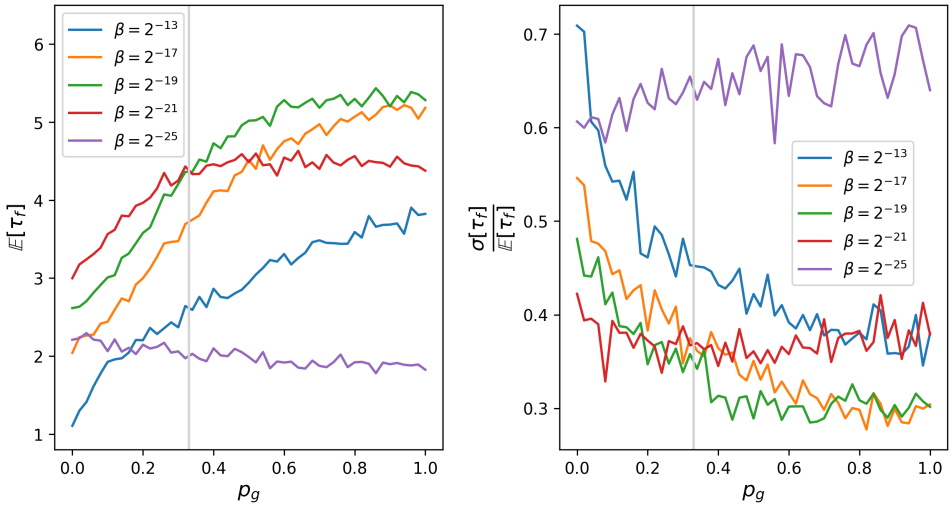

FIGURE 13. Mean forecast time $\mathbb{E}[\tau_f]$ (left) and coefficient of variation $\sigma[\tau_f]/\mathbb{E}[\tau_f]$ (right) as a function of p_g for a range of regularization parameters β. A regularization parameter of $\beta = 2^{-19}$ is optimal among the values presented here for $p_g > 0.33$ (demarcated by a vertical line). Results are shown for fixed $D_r = 300$ and $N = 20,000$.

any entry with absolute value larger than 1. It is clearly seen that linear features are suppressed by the columns of $\mathbf{W}$. Note that not all linear features are entirely suppressed.

In the second experiment, we follow the same procedure as before except we start with only saturated random features. In Figure 16 it is seen that saturated features are suppressed even stronger by the outer weights than linear features. In contrast to linear features, saturated features are effectively fully suppressed once the number of good features exceeds $N_g = 50$.

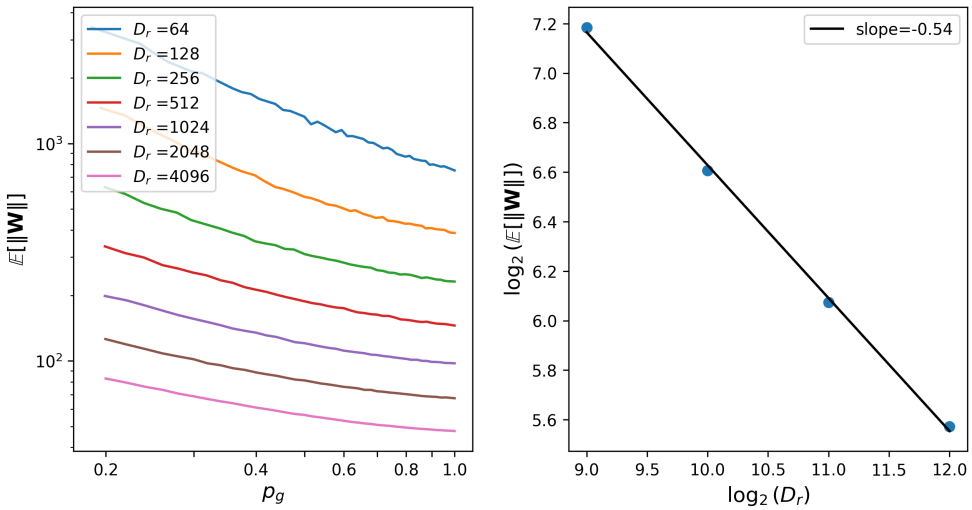

FIGURE 14. Left: The mean of the Frobenius norm of the outer weights, $\|\mathbf{W}\|$, as a function of p_g on a log-log scale. Right: The mean of the Frobenius norm of the outer weights, $\|\mathbf{W}\|$, as a function of the feature dimension D_r for $p_g = 1$. The line of best fit with approximate slope -0.54 is also shown on the right. The data are from the same experiments as shown in Figure 9.

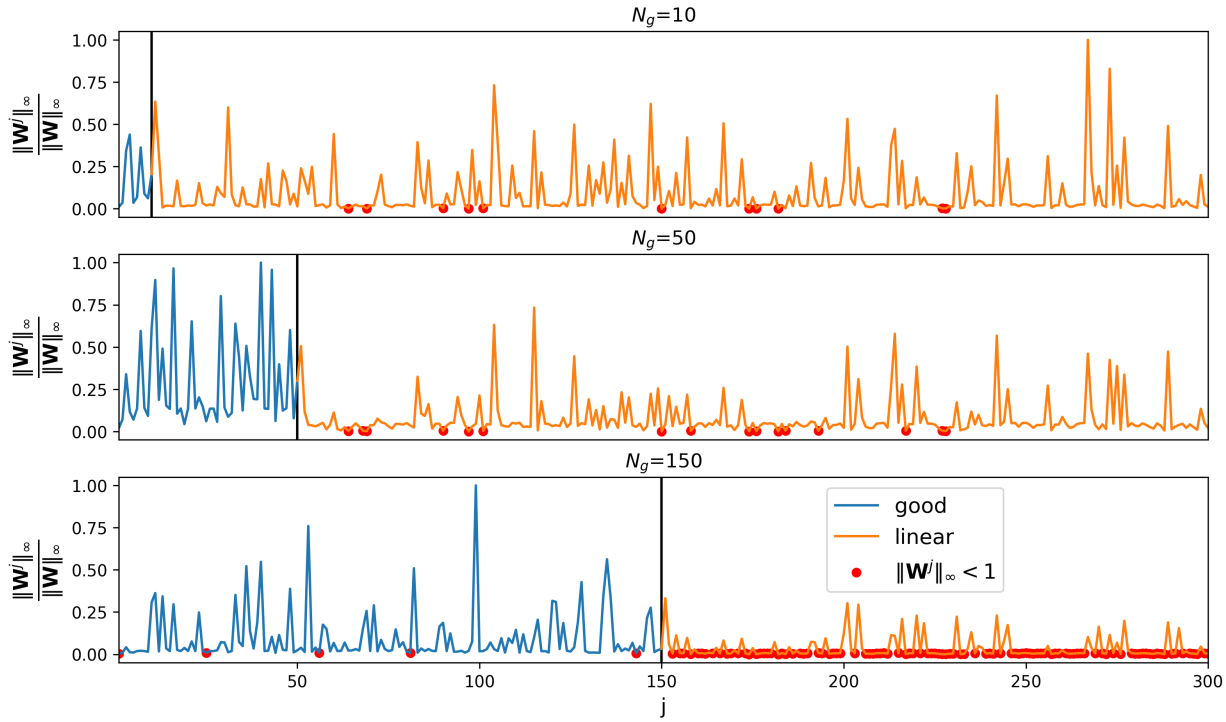

FIGURE 15. Normalized supremum norm of the columns of $\mathbf{W}$ for different numbers of good features with $N_g = 10$, $N_g = 50$ and $N_g = 150$ and otherwise exclusively linear bad features. The x-axis represents column indices. The good and linear columns are indicated in blue and orange, respectively. The red dots signify columns with supremum norm less than 1. The overall feature dimension is $D_r = 300$ and the outer weights were obtained from training data of length $N = 20,000$.

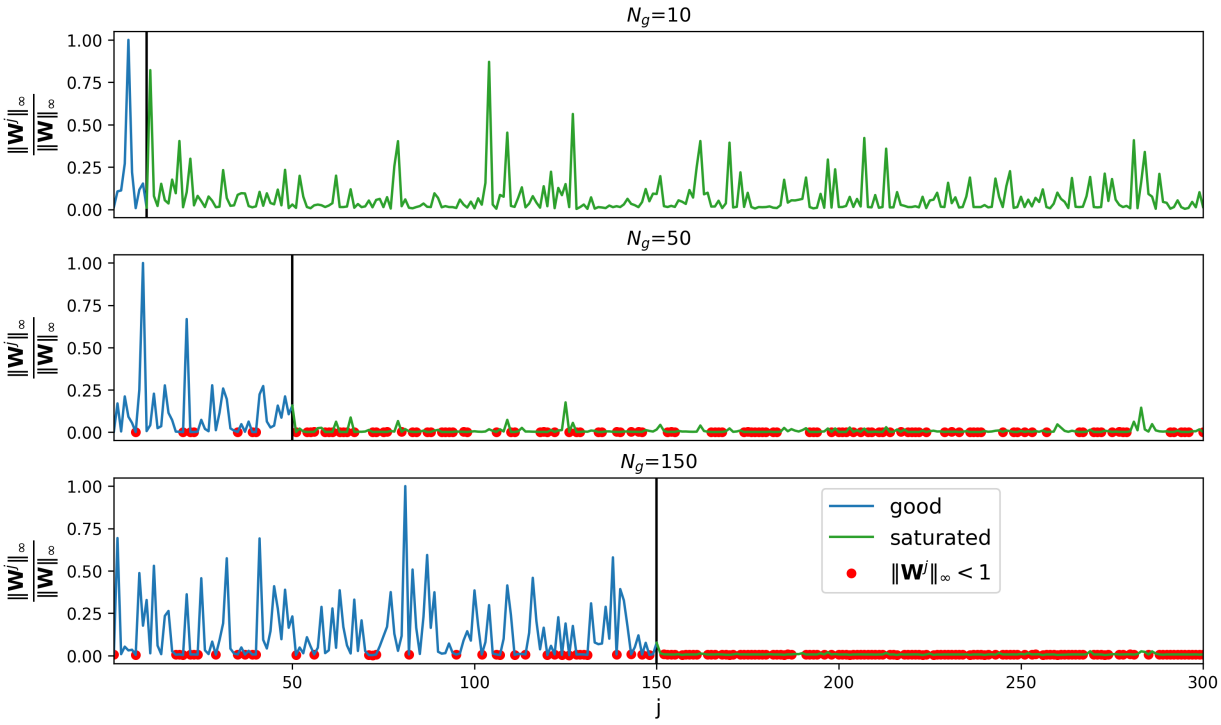

FIGURE 16. Normalized supremum norm of the columns of $\mathbf{W}$ for different numbers of good features with $N_g = 10$, $N_g = 50$ and $N_g = 150$ and otherwise exclusively saturated bad features. The x-axis represents column indices. The good and saturated columns are indicated in blue and green, respectively. The red dots signify columns with supremum norm less than 1. The overall feature dimension is $D_r = 300$ and the outer weights were obtained from training data of length $N = 20,000$.

6. Comparison with a single-layer feedforward network trained with gradient descent

A natural question is if a single-layer feedforward network of the architecture (6) for which the internal weights $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ are learned together with the outer weights $\mathbf{W}$ performs better or worse than random feature maps with fixed good internal weights. In particular, we consider the non-convex optimization problem

$$\mathbf{\Theta}^* = \arg\min_{\Theta} \mathcal{L}(\Theta; \mathbf{U}), \tag{30}$$

with $\mathbf{\Theta} = (\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}}, \mathbf{W})$ and the loss function $\mathcal{L}$ defined in (10). To solve the optimization problem (30) we employ gradient descent. To fairly compare with the results from the random feature model, we set the internal layer width to $D_r = 300$, the regularization parameter to $\beta = 4 \times 10^{-5}$, and the training data length to $N = 20,000$. To initialize the network weights we use the standard Glorot initialization [12]. We use an adaptive learning rate scheduler which is described in Appendix 9.1.

Figure 17 shows the evolution of the mean forecast time $\mathbb{E}[\tau_f]$ and the logarithm of the loss function $\mathcal{L}$ during training. The expectation is computed over 500 different validation data sets. Optimization over all weights clearly allows for a significantly smaller training loss $\mathcal{L}$ compared to random feature maps (cf. Figure 18). The neural network achieves a final value of the loss function of $\mathcal{L} \approx 0.09$ which is a 95% improvement when compared to a random feature map of the same size with only good internal parameters, i.e. $p_g = 1$, which has a loss of $\mathcal{L} \approx 1.73$ on average. However, the situation is very different for the mean forecast time. The mean forecast time $\mathbb{E}[\tau_f]$ is a slowly growing function of the gradient descent steps with the last 10^5 steps resulting in only about 0.32% improvement. The data are plotted every 10^4 steps and therefore the typical fluctuations of gradient descent are not visible. Maybe surprisingly, optimizing

the internal weights via gradient descent does not lead to a better forecasting skill when compared to the random feature map surrogate model. After 1.5×10^6 steps the neural network achieves a mean forecast time of only $\mathbb{E}[\tau_f] \approx 3.75$. Random feature maps of the same size with $p_g = 1$ generate a mean forecast time of $\mathbb{E}[\tau_f] \approx 4.46$ (cf. Figure 9). Furthermore, the training took approximately 8.2×10^4 seconds on the T4 GPU available through Google Colab cloud platform. In contrast, initializing and training a random feature map of the same size took less than 1 second in total, i.e almost $100,000$ times faster.

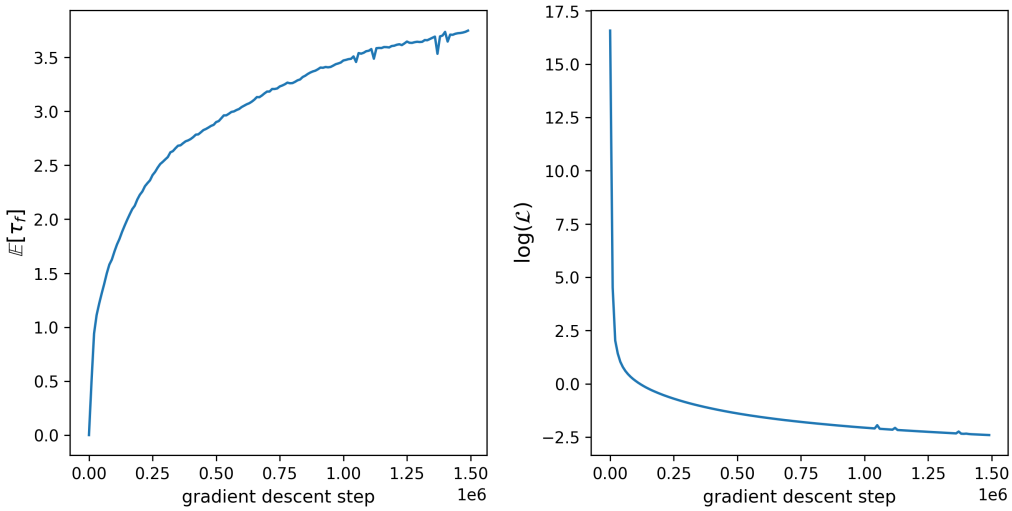

FIGURE 17. Evolution of the mean forecast time $\mathbb{E}[\tau_f]$ and the logarithm of the loss function (10) $\log(\mathcal{L})$ during training of a single-layer feedforward network with gradient descent. For each step $\mathbb{E}[\tau_f]$ is computed using 500 test trajectories. The network with width $D_r = 300$ was trained with training data of length $N = 20,000$ and a regularization parameter $\beta = 4 \times 10^{-5}$. Results are shown every 10^4 gradient descent steps.

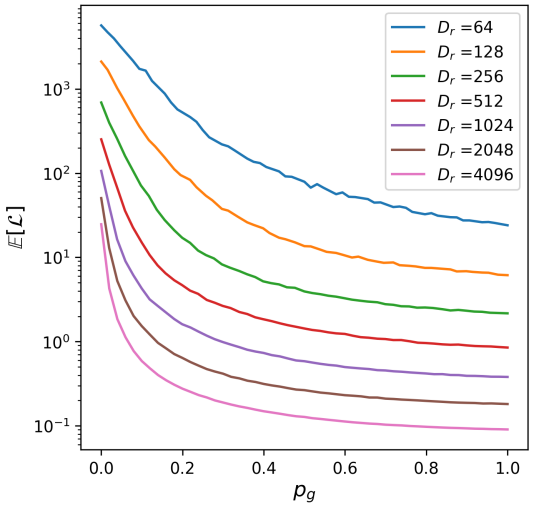

FIGURE 18. Mean loss $\mathcal{L}$ for random feature maps as a function of p_g for different values of the feature dimension D_r. The data shown here correspond to the experiments shown in Figure 9 with $N = 20,000$ and $\beta = 4 \times 10^{-5}$.

The left panel of Figure 19 shows that the mean forecast time $\mathbb{E}[\tau_f]$ and the logarithm of the loss function $\log(\mathcal{L})$ are linearly related for the single-layer neural network. This is a direct manifestation of the exponential sensitivity in chaotic dynamical systems: in each gradient descent step the loss experiences small changes leading to small changes in the learned weights and hence in the resulting surrogate model. These small changes in the chaotic surrogate model lead to an exponential divergence of nearby trajectories. This causes an exponential in time loss of predictability, characterized here by the mean of the forecast time (5). The

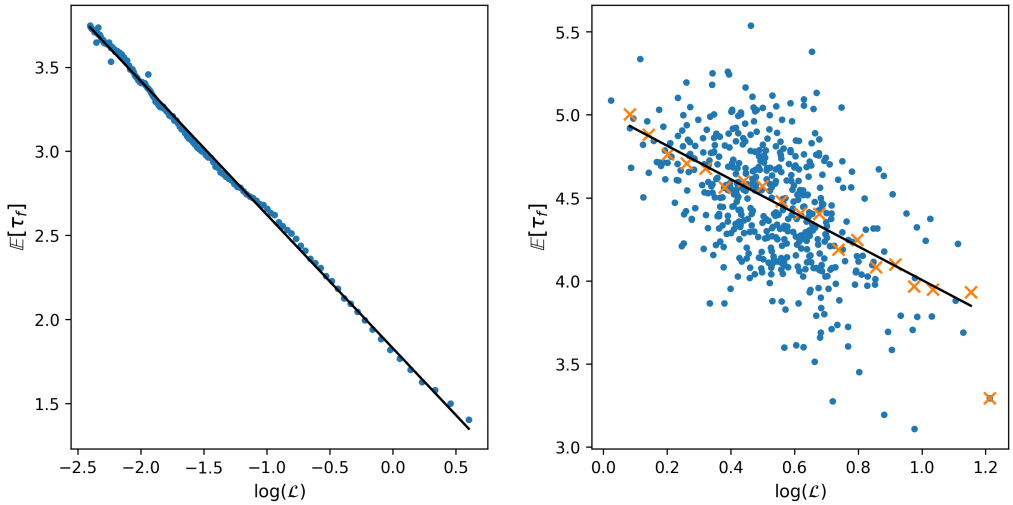

FIGURE 19. Relationship between the mean forecast time $\mathbb{E}[\tau_f]$ and the logarithm of the loss function $\mathcal{L}$ for a single-layer feedforward network (left) and for a random feature map with only good internal parameters, i.e. $p_g = 1$ (right). Each dot in the left panel corresponds to a gradient descent step. Each dot in the right panel corresponds to one realization of a random feature map. The expectation is computed over 500 validation trajectories. Each descent step and each realization use the same training and testing data. The black line in the left panel represents the best-fit line. In the right panel the orange crosses denote the conditional mean $\mathbb{E}[\tau_f | \log(\mathcal{L})]$ and the black line represents the best-fit line. We use a feature dimension of $D_r = 300$, training data length $N = 20,000$ and regularization parameter $\beta = 4 \times 10^{-5}$.

same sensitive dependency on small changes of the surrogate model, quantified by small changes of the loss function, is also present in random feature maps. The right panel of Figure 19 shows the mean forecast time $\mathbb{E}[\tau_f]$ as a function of the logarithm of the loss function for random feature maps. Each dot represents one realization of a random feature map with feature dimension $D_r = 300$, trained on the same data as the single-layer feedforward network. The mean forecast time $\mathbb{E}[\tau_f]$ is computed using the same 500 validation trajectories as the network. Averaged over bins of the logarithm of the loss function, the mean forecast time shows the same linear relationship with the logarithm of the loss function (orange crosses in Figure 19). The slopes of the best-fit lines in Figure 19 show that the forecasting skill of the random feature map improves slightly faster with decreasing loss when compared to the single-layer network with an estimated slope of -1.01 for the random feature map and -0.79 for the neural network.

The discrepancy between the neural network having worse forecasting skill compared to random feature maps despite achieving smaller loss can be explained as follows. Minimizing the loss function $\mathcal{L}$ aims at learning the single-step surrogate map (6). High forecasting skill, however, requires multiple applications of the single-step surrogate model which is not explicitly accounted for in the loss function (10). In Section 3.1 we established that the main controlling factor for achieving high forecasting skill is the number of good features. In random feature maps we can control and maximize this number simply by sampling good parameters according to our hit-and-run Algorithms 1 and 2, respectively. On the other hand, the training of the single-layer feedforward network is only designed to minimize the loss but not to maximize the

number of good features to $N_g = D_r = 300$ in our case. Figure 20 shows the number of different types of rows produced during the training instance of Figure 17. We see that only a single good row was produced in $(\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}})$ during the early steps of the optimization, and this good row was then quickly destroyed during the training process. We checked that even when the network is initialized with only good internal parameters, i.e. $p_g = 1$, training eventually leads to a complete absence of good internal parameters with $p_g = 0$ for reasonable learning rates. To understand the absence of good rows in the trained network, note that for any random feature map $\boldsymbol{\Theta} = (\mathbf{W}_{\text{in}}, \mathbf{b}_{\text{in}}, \mathbf{W})$ essentially lies on the graph of a continuous function due to the intricate relationship between the internal and outer weights dictated by (11). So the set of all possible $\boldsymbol{\Theta}$ for random feature maps has zero Lebesgue measure in $\mathbb{R}^{D_r \times (2D+1)}$. It is therefore highly unlikely that gradient descent finds the lower-dimensional subset of the random feature map weights in its search space which is the full $\mathbb{R}^{D_r \times (2D+1)}$. It would be interesting to see if the network generates good features if the loss function is augmented by a penalty term promoting good features. In any case, random feature maps are significantly cheaper to train.

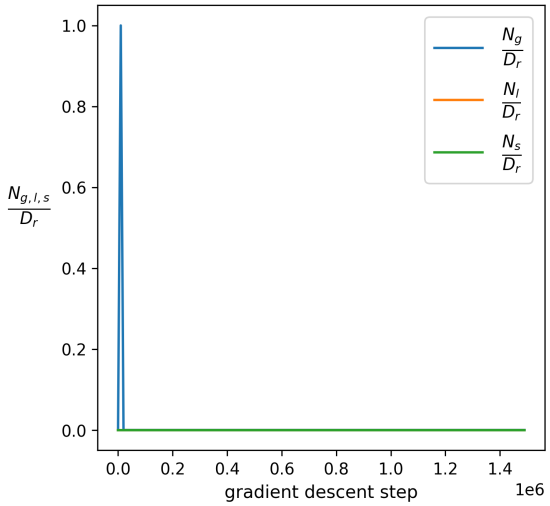

FIGURE 20. Evolution of the number (normalized by D_r) of learned good, linear and saturated rows $(\mathbf{w}_i^{\text{in}}, b_i^{\text{in}})$ in the internal parameters during a single training episode of a single-layer feedforward network. The data shown here correspond to the training instance shown in Figure 17 where we used $D_r = 300$ and $N = 20,000$. Results are shown every 10^4 gradient descent steps.

7. Results for long-time statistical behaviour

We have so far focused on short-term integration and assessed the quality of a surrogate model by its ability to remain close to a reference trajectory. In certain applications such as climate prediction, however, it is more important to reliably recover the statistical properties of the dynamical system. Good short-term forecasting, i.e. high mean forecasting times, does not necessarily imply reliable reproduction of the statistics, and vice versa. In Figure 21 we compare the empirical histograms of the three variables of the Lorenz-63 system (4) with those of the corresponding random feature surrogate maps where we only used good features, i.e. $p_g = 1$, and only used linear and saturated features, i.e. $p_g = 0$ with $p_l = p_s = 0.5$. The histograms are obtained from long simulations over $2,000$ time units, approximating the invariant measure. It is seen that the random feature map with only good features reproduces the invariant measure well. When only linear and saturated features are employed, the histogram of the Lorenz-63 system is less well reproduced. Remarkably, for $p_g = 0$ the histogram is still reasonably well reproduced; in particular, the tail behaviour of the histograms is well reproduced.

We further show in Figure 22 a comparison between the histograms obtained from the random feature map surrogate model that used only good features with $p_g = 1$, and the single-layer feedforward network. It

is clearly seen that the neural network is not able to reproduce the long-time behaviour, with its empirical histogram being wildly different from that of the original Lorenz-63 system. However, the neural network also reproduces the tail behaviour of the histogram very well.

It is remarkable that including global information about the data, in the form of the constraints (13) which take into account information about the attractor, is sufficient to ensure that the trained surrogate model is able to recover the invariant measure. This makes random feature models with only good internal weights a very attractive network architecture for dynamical systems.

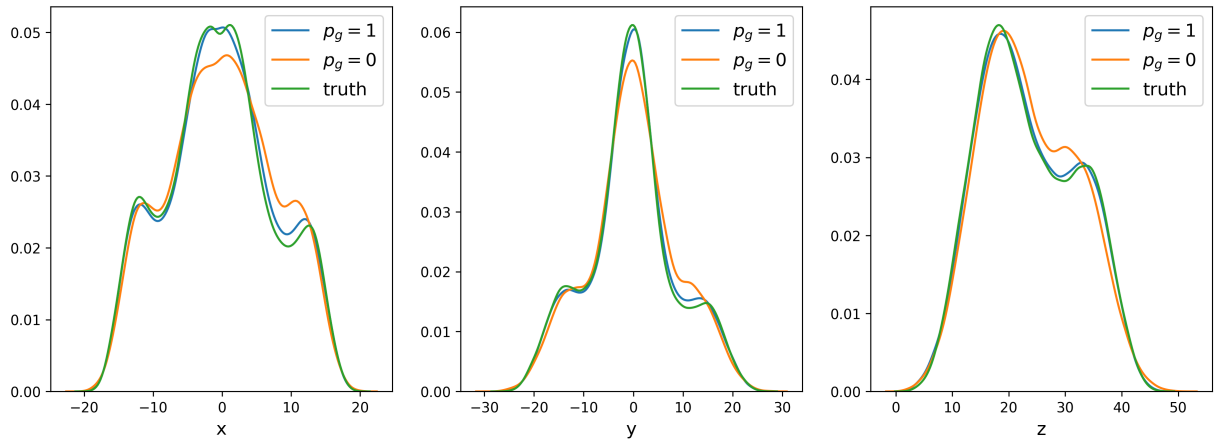

FIGURE 21. Empirical histograms of the marginals of the invariant measure for the Lorenz-63 system obtained from simulating the original dynamical system (4), and for random feature map surrogate model with only good features ($p_g = 1$) and without any good features ($p_g = 0$).

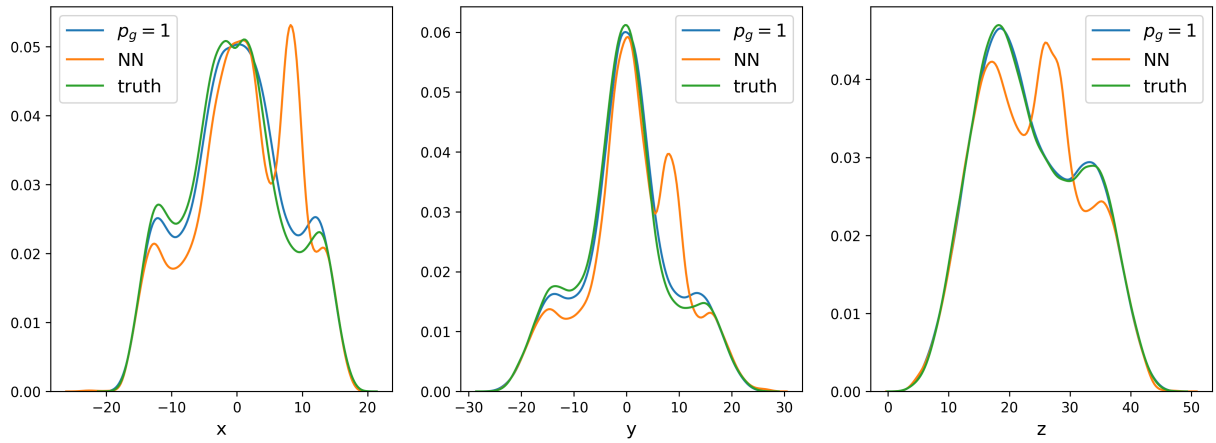

FIGURE 22. Empirical histograms of the marginals of the invariant measure for the Lorenz-63 system obtained from simulating the original dynamical system (4), for a random feature map surrogate model with $p_g = 1$, and for a neural network (NN) surrogate model. The same neural network is used as in Figure 17.

8. Summary and future work

We established the notion of good features and good internal parameters for random feature maps with a tanh-activation function. These good internal weights are characterised by affinely mapping the training

data into the nonlinear, non-saturated domain of the tanh-activation function. We established that the number of good features $N_g \leq D_r$ is the controlling factor in determining the forecasting skill of a learned surrogate map, rather than the feature dimension D_r. Interestingly, the forecasting skill was found to be equally deteriorated by linear features as by saturated features. We developed computationally cheap hit-and-run sampling algorithms to uniformly sample from the set of good internal parameters. The hit-and-run Algorithms 1 and 2 sample the internal weights from a data-informed convex set, rather than obtaining them by minimizing some cost function. We demonstrated how ridge regression engages with a given number of good and bad features. In particular, saturated features are eliminated almost entirely by the outer weights, provided a sufficient number of good features are present. Similarly, linear features are suppressed by the outer weights, albeit to a lesser degree. Once there are sufficiently many good features present to allow for a significant reduction of the data mismatch term of the loss function, regularization kicks in and reduces the norm of the outer weights corresponding to good features.

We further showed that a single-layer feedforward network with the same width D_r trained with gradient descent exhibits inferior forecasting skill compared to a random feature map surrogate map which used only good internal parameters. The neural network achieves a significantly smaller value of the loss function. Good forecasting skill, however, requires multiple applications of the surrogate map, and, as we showed, is controlled by the number of good features. The lower forecasting skill is due to the optimization process not finding solutions on the measure-zero set of good parameters. Even when initialized with good parameters, the gradient descent quickly reduces the number of good internal parameters.

We found numerically that including global information about the attractor, in form of the constraints for the internal weights (13), is sufficient to ensure that the trained random feature map reproduces the statistical properties of the underlying dynamical system and its invariant measure. The rationale for the choice of good internal weights was entirely motivated by the nonlinear non-saturated structure of the tanh-activation function. How far the constraints on the function domain (13) translate into dynamical information ensuring the preservation of the invariant measure remains an open question, planned for further research.

The proposed optimization-free algorithm to choose internal non-trainable parameters can potentially lead to new design and computationally cheap training schemes for more complex network architectures. Our algorithms may be used to further improve the forecasting skill of reservoir computers [41, 11]. In [37] it is shown that the hit-and-run algorithm can be used for initializing modified RFMs, which include skip connections, and deep architectures, to achieve state-of-the-art forecasting skill, outperforming standard reservoir computers with significantly less computational effort and model sizes.

The parameters L_0 and L_1 that define *good* parameters can, in principle, be considered as hyperparameters, and could be tuned, for example, using Bayesian optimization [26]. We have refrained here from doing so as we have not observed significant changes in the forecasting skill for values close to $L_0 = 0.4$ and $L_1 = 3.5$ which we used throughout this work.

We considered here the tanh-activation function as it is widely used in reservoir computing. The separation of the domain into linear, saturated and good regions can readily be extended to other continuous sigmoidal functions. These activation functions are widely used for random feature maps and in reservoir computing architectures. They are less frequently used for deep neural networks where the saturated regions lead to the vanishing gradient problem [14]. However, we note that recently the tanh-function gained again more interest in the machine learning community [51]. It is an interesting question whether our methodology can be extended to other commonly used activation functions such as the ReLU-type functions or trigonometric functions as initially employed for random feature maps [43]. For example, for GELU [18] one may define good features to be those that correspond to an interval around 0 where the function is nonlinear and non-saturated. For classical Fourier random feature maps, one could separate the domain into linear and saturated regions near the roots and maxima/minima of the sin- and cos-functions, and consider the complement as the "good" region; the linear domain of the sin-function needs to be aligned with the saturated domain of the cos-function, and vice versa. Exploring whether the good regions of such functions

allow for sufficient variability of the corresponding features and if this can lead to an improved sampling is planned for future research.

Acknowledgement

The authors would like to acknowledge support from the Australian Research Council under Grant No. DP220100931.

9. Appendix

9.1. **Adaptive learning rate for the single-layer neural network.** We describe in Algorithm 3 the adaptive learning rate algorithm we used when training the single-layer feedforward network in Section 6. Essentially our scheduler computes the decay rate of the loss every I steps and modifies the learning rate by increasing or decreasing it by a constant fraction ξ if necessary. We use an initial rate $\eta_0 = 10^{-3}$, update interval $I = 100$, update fraction $\xi = 0.1$, update threshold $\gamma = -10^{-4}$ and number of gradient descent steps $E = 1.5 \times 10^6$ in our scheduler.

Algorithm 3 Adaptive learning rate scheduler

Input: Choose initial rate η_0, update interval I, update fraction ξ, update threshold γ, number of gradient descent steps E.
$k \leftarrow 1$ (gradient descent step).
$\mathcal{L}_0 \leftarrow$ value of $\mathcal{L}$ at gradient descent step k.
$\eta \leftarrow \eta_0$ (learning rate).
while $k < E$ **do**
  if k is divisible by I **then**
    $\mathcal{L}_1 \leftarrow$ value of $\mathcal{L}$ at gradient descent step k.
    $\Delta \leftarrow \frac{\mathcal{L}_1 - \mathcal{L}_0}{\mathcal{L}_0}$.
    if $\Delta > \gamma$ **then**
      if $\Delta > 0$ **then**
        $\eta \leftarrow \eta(1 - \xi)$
      else
        $\eta \leftarrow \eta(1 + \xi)$
      end if
    end if
    $\mathcal{L}_0 \leftarrow \mathcal{L}_1$
  end if
  $k \leftarrow k + 1$
end while

We tried several other strategies such as finding an optimal learning rate every few steps using bisection, random modifications of the learning rate based on the behavior of the loss function, aggressive constant learning rates, conservative constant learning rates, piecewise linear learning rates etc. We found that the simple strategy presented in Algorithm 3 leads to the lowest final value of the loss function for the same number of gradient descent steps. Figure 23 shows the adaptive learning rate used during the training instance shown in Figure 17.

References

[1] Henry DI Abarbanel, Reggie Brown, John J Sidorowich, and Lev Sh Tsimring. The analysis of observed chaotic data in physical systems. *Reviews of Modern Physics*, 65(4):1331, 1993.

[2] Yasin Abbasi-Yadkori, Peter Bartlett, Victor Gabillon, and Alan Malek. Hit-and-run for sampling and planning in non-convex spaces. In *Artificial Intelligence and Statistics*, pages 888–895. PMLR, 2017.

[3] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

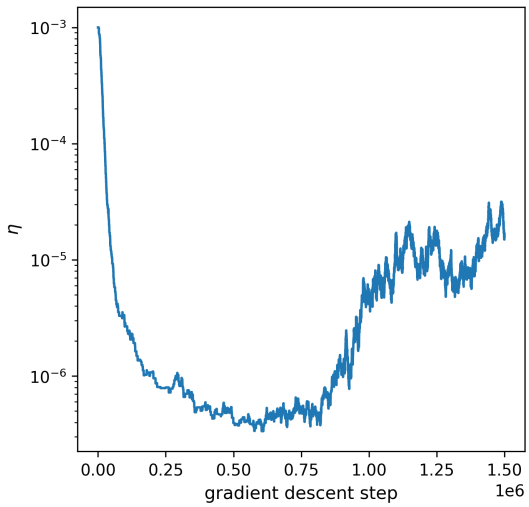

FIGURE 23. Adaptive learning rate η used during the training instance presented in Figure 17, where the loss function is shown.

[4] Erik Bollt. On explaining the surprising success of reservoir computing forecaster of chaos? The universal machine learning dynamical system with contrast to VAR and DMD. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(1):013108, 2021.

[5] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[6] Jian Cao, Zhi Li, and Jian Li. Financial time series forecasting model based on CEEMDAN and LSTM. *Physica A: Statistical mechanics and its applications*, 519:127–139, 2019.

[7] Karthekeyan Chandrasekaran, Daniel Dadush, and Santosh Vempala. Thin partitions: Isoperimetric inequalities and sampling algorithms for some nonconvex families. *arXiv preprint arXiv:0904.0583*, 2009.

[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[9] Oliver R. A. Dunbar, Nicholas H. Nelsen, and Maya Mutic. Hyperparameter optimization for randomized algorithms: a case study on random features. *Statistics and Computing*, 35(3):56, 2025.

[10] Rui Fu, Zuo Zhang, and Li Li. Using LSTM and GRU neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 324–328. IEEE, 2016.

[11] D. Gauthier, E. Bollt, A. Griffith, and W. Barbosa. Next generation reservoir computing, 2021.

[12] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[13] Lukas Gonon and Juan-Pablo Ortega. Fading memory echo state networks are universal. *Neural Networks*, 138:10–13, 2021.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[15] Georg A. Gottwald and Sebastian Reich. Combining machine learning and data assimilation to forecast dynamical systems from noisy partial observations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(10):101103, 10 2021.

[16] Georg A. Gottwald and Sebastian Reich. Supervised learning from noisy observations: Combining machine-learning techniques with data assimilation. *Physica D: Nonlinear Phenomena*, 423:132911, 2021.

[17] Lyudmila Grigoryeva and Juan-Pablo Ortega. Echo state networks are universal. *Neural Networks*, 108:495–508, 2018.

[18] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach*, volume 5. Citeseer, 2002.

[21] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[22] Seksan Kiatsupaibul, Robert L Smith, and Zelda B Zabinsky. An analysis of a variation of hit-and-run for uniform sampling from general regions. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 21(3):1–11, 2011.

[23] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, 4, 1991.

[24] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[25] Aditi Laddha and Santosh S Vempala. Convergence of Gibbs sampling: Coordinate Hit-and-Run mixes fast. *Discrete & Computational Geometry*, 70(2):406–425, 2023.

[26] Matthew E. Levine and Andrew M. Stuart. A framework for machine learning of model error in dynamical systems. *Comm. Amer. Math. Soc.*, 2:283–344, 2022.

[27] Y. Li, J. Lu, and A. Mao. Variational training of neural network approximations of solution maps for physical models. *Journal of Computational Physics*, 409:109338, 2020.

[28] Youru Li, Zhenfeng Zhu, Deqiang Kong, Hua Han, and Yao Zhao. EA-LSTM: Evolutionary attention-based LSTM for time series prediction. *Knowledge-Based Systems*, 181:104785, 2019.

[29] Fanghui Liu, Xiaolin Huang, Yudong Chen, and Johan AK Suykens. Random features for kernel approximation: A survey on algorithms, theory, and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):7128–7148, 2021.

[30] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20(2):130–141, 1963.

[31] Edward N Lorenz. Predictability: A problem partly solved. In *Proc. Seminar on Predictability*, volume 1. Reading, 1996.

[32] László Lovász. Hit-and-run mixes fast. *Mathematical Programming*, 86:443–461, 1999.

[33] László Lovász and Santosh Vempala. Hit-and-run from a corner. In *Proceedings of the thirty-sixth Annual ACM Symposium on Theory of Computing*, pages 310–314, 2004.

[34] László Lovász and Santosh Vempala. The geometry of logconcave functions and sampling algorithms. *Random Structures & Algorithms*, 30(3):307–358, 2007.

[35] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 659–686. Springer, 2012.

[36] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[37] Pinak Mandal and Georg A. Gottwald. Learning dynamical systems with hit-and-run random feature maps, 2025.

[38] Renate Meyer and Nelson Christensen. Bayesian reconstruction of chaotic dynamical systems. *Physical Review E*, 62(3):3535, 2000.

[39] Kohei Nakajima and Ingo Fischer. *Reservoir computing*. Springer, 2021.

[40] Nicholas H. Nelsen and Andrew M. Stuart. The random feature model for input-output maps between Banach spaces. *SIAM Journal on Scientific Computing*, 43(5):A3212–A3243, 2021.

[41] Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical Review Letters*, 120(2):024102, 2018.

[42] A. Rahimi and B. Recht. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561, 2008.

[43] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1177–1184. Curran Associates, Inc., 2008.

[44] Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561. IEEE, 2008.

[45] Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[46] Michael Small, Kevin Judd, and Alistair Mees. Modeling continuous processes from data. *Phys. Rev. E*, 65:046704, 2002.

[47] Robert L Smith. Efficient monte carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32(6):1296–1308, 1984.

[48] Ralf C Staudemeyer and Eric Rothstein Morris. Understanding LSTM–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.

[49] Warwick Tucker. A rigorous ODE solver and Smale's 14th problem. *Foundations of Computational Mathematics*, 2(1):53–117, 2002.

[50] Zelda B Zabinsky, Robert L Smith, S Gass, and M Fu. Hit-and-run methods. *Encyclopedia of Operations Research and Management Science*, pages 721–729, 2013.

[51] Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization, 2025.