Analyzing Koopman approaches to physics-informed machine learning for long-term sea-surface temperature forecasting

Julian Rice[1,2], Wenwei Xu[1], and Andrew August[3]

[1]Marine and Coastal Research, Pacific Northwest National Laboratory
[2]California Polytechnic State University, San Luis Obispo
[3]Computing and Analytics, Pacific Northwest National Laboratory

July 2020

Abstract

Accurately predicting sea-surface temperature weeks to months into the future is an important step toward long term weather forecasting. Standard atmosphere-ocean coupled numerical models provide accurate sea-surface forecasts on the scale of a few days to a few weeks, but many important weather systems require greater foresight. In this paper we propose machine-learning approaches sea-surface temperature forecasting that are accurate on the scale of dozens of weeks. Our approach is based in Koopman operator theory, a useful tool for dynamical systems modelling. With this approach, we predict sea surface temperature in the Gulf of Mexico up to 180 days into the future based on a present image of thermal conditions and three years of historical training data. We evaluate the combination of a basic Koopman method with a convolutional autoencoder, and a newly proposed "consistent Koopman" method, in various permutations. We show that the Koopman approach consistently outperforms baselines, and we discuss the utility of our additional assumptions and methods in this sea-surface temperature domain.

1 Introduction

1.1 Background

1.1.1 Long-term sea-surface temperature forecasting

The sea surface temperature in the Gulf of Mexico is of great environmental, economic, and social significance. It regulates the low oxygen "dead zone"[16], fuels intense tropical cyclones [21], such as Katrina and Harvey, and modulates thunderstorms in the southern US [7]. The Gulf of Mexico also has unique flow features such as the Loop Current, which forms when warm subtropical waters intrude into the Gulf through the Yucatan Channel and eventually push out through the Florida Straits. The Loop Current periodically sheds warm eddies, which propagate westward in the Gulf for 9-12 months [21], causing distinct temperature dynamics. East of Florida, much of the tail end of the Loop Current forms the Gulf Stream, another unique and important flow feature that pushes warm water into the North Atlantic.

Current state-of-the-art SST predictions are from atmosphere-ocean coupled numerical models, which do not make predictions beyond a month [5] [14]: for example NOAA's Real Time Ocean Forecast System (RTOFS) only forecasts up to five days [22]. Additionally, numerical models are computationally expensive and rely on initial boundary conditions which are generally hard to obtain. It is also notable that the eddy shedding events which involve the rapid growth of nonlinear instability is particularly difficult to forecast in numerical models [14].

In this study, we evaluate machine-learning (ML) techniques for long-range SST forecasting. More specifically, we well predict SST in the Gulf of Mexico up to 180 days in the future from a single thermal image of SST, leveraging techniques based in Koopman operator theory.

1.1.2 Nonlinear dynamical systems

For discrete-time dynamics problems, we seek a transformation φ that maps a system's state z at time t its

state at the next time step:

$$z_{t+1} = \varphi(z_t) \qquad (1)$$

where $S \subset \mathbb{R}^n$ is the space of all possible states, $z \in S$, and $\varphi : S \to S$. We focus specifically on systems where φ is time-invariant; that is, $\forall t \in \mathbb{Z}$ equation (1) holds.

In dynamical systems problems where φ is nonlinear, a machine-learning approach is often appealing, since the system can be too complex to model analytically or numerically. However, a purely data-driven approach eschews physics knowledge that can be extremely helpful in developing meaningful and generalizable approximations of φ. Thus, we choose to utilize physics-informed machine-learning, whereby our understanding of physical systems is incorporated into machine-learning models.

Additionally, we often do not have access to the state itself, only "observables" of it, which are simply functions of the state that are much more easily measured:

$$y_t = f(z_t) \qquad (2)$$

where $f : S \to \mathbb{R}$, and $y_t \in \mathbb{R}$ is the value of an observable f of the system's state at time t. Thus, we need a more complex formulation than Eqn. (1).

1.1.3 A Hitchhiker's Guide to Koopman Operator Theory

Koopman operator theory [12] has recently been (re-)discovered [4] to be a valuable tool in modelling dynamical systems: it suggests that such a nonlinear dynamical system can undergo a transformation into an infinite dimensional space in which it evolves linearly in time.

To begin, consider an observable function f, per Eqn. (2). The Koopman operator, utilizing some transformation $T : S \to S$, is as follows:

$$\mathcal{K}_T f(x) = f \circ T(x) \qquad (3)$$

Since the Koopman operator is, mathematically, just a specific reformulation of the composition operator, its linearity follows simply [1]:

$$\begin{aligned}
\mathcal{K}_T[g_1 + g_2](x) &= [g_1 + g_2] \circ T(x) \\
&= g_1 \circ T(x) + g_2 \circ T(x) \qquad (4) \\
&= \mathcal{K}_T g_1(x) + \mathcal{K}_T g_2(x)
\end{aligned}$$

To apply the operator to our dynamical system, we use the system dynamics φ as the transformation T. Now, instead of the system evolving nonlinearly in the state space from z_t to z_{t+1} (as in Eqn. (1)), it evolves linearly in the space of the observables (with a

slight abuse of notation in the last term, to make the evolution of the observable clear):

$$y_{t+1} = f(z_{t+1}) = \mathcal{K}_\varphi f(z_t) = \mathcal{K}_\varphi y_t \qquad (5)$$

Complicating matters is the fact that since the Koopman operator evolves the system an infinite dimensional space, but this can be overcome by the "somewhat naive but useful" [1] ideation of the operator as a left-multiplication by infinite dimensional matrix. To make our problem computable, we can then make the assumption (common in data-driven approaches to the Koopman operator [2]) that there exists a mapping χ that encodes the majority of the operator's action into a finite dimensional matrix C:

$$C = \chi \circ \mathcal{K}_\varphi \circ \chi^{-1} \qquad (6)$$

and conversely,

$$\chi^{-1} \circ C \circ \chi = \mathcal{K}_\varphi \qquad (7)$$

Finally, we can notice that applying the Koopman operator repeatedly simplifies to an expression for predicting any $k \in \mathbb{N}$ number of steps into the future:

$$\begin{aligned}
\mathcal{K}_\varphi \circ \mathcal{K}_\varphi \circ \ldots &= \chi^{-1} \circ C \circ \chi \circ \chi^{-1} \circ C \circ \ldots \\
&= \chi^{-1} \circ C^k \circ \chi
\end{aligned} \qquad (8)$$

Therefore we arrive at our formulation of the problem that we will utilize in building our network architectures:

$$y_{t+k} = \chi^{-1} \circ C^k \circ \chi(y_t) \qquad (9)$$

If we properly construct networks in accordance with Eqn. (9), we know that those networks will function as good approximations of the Koopman operator. Thus we have applied physics-informed constraints to the network, though these constraints are informed by the physics of dynamical systems in general, not the specific physical laws governing a particular system[1]. The network is in turn more interpretable; we could, for example, analyze the encoder/decoder structure to better understand how the latent space relates to the space of the observables; or, we could calculate the eigenvalues of C to gain insight into the system's stability.

1.1.4 Convolutions

Convolutions are a common operation in machine learning on spatial data. To provide a basic intuitive description, convolution works by learning a kernel that condenses multiple datapoints into one; in Fig. 1, the shaded 3x3 in the blue input layer is
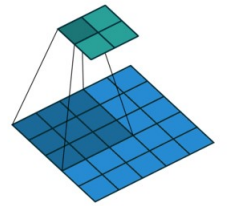

Figure 1: Example Convolution [11]

---

[1]This is not a bad thing: This generality is precisely what makes Koopman networks so useful, because they can be applied to a multitude of different dynamical systems problems.

operated on by the kernel, producing the 1x1 shaded value in the green output layer. The rest of the output is generated in a similar way, by shifting the kernel along the dimensions of the input.

This operation can be viewed as a simple but effective incorporation of prior knowledge into machine learning models [25]. Unlike fully-connected layers—where each node takes input from every node in the previous layer—convolutions enforce *locality*, by taking input from spatially close nodes in the previous layer, and *translation invariance*, by applying the same operation at all spatial locations in the data.

1.2   Related Work

There have been various approaches to SST forecasting with physics-agnostic machine learning models. Generic neural network (NN) approaches to SST forecasting have been developed for at least two decades [24]. Recently, Long-Short Term Memory [28][29] [30] and Autoencoder [8] architectures have been popular. Another interesting approach has been to learn to model the error of numerical models with a neural network [15].

Application of NNs to physics problems generally has a long history [13] [3], as an abundance of data and difficulty in accurately modeling certain systems has made the approach appealing. An expressly physics-informed approach to machine learning, however, was pioneered by Raissi et al. [18][17], who utilized NNs to model partial differential equations describing physical systems. See Willard et al. [27] for an thorough survey of the physics-informed ML approach.

Recently, the application of physics-informed ML to SST forecasting has received some attention. de Bezenec et al. [6] utilize physical knowledge of advection-diffusion to predict sea-surface temperature. Erichson et al. [9] incorporate constraints inspired by Lyapunov stability in their forecasting of various fluid flows, including sea-surface temperature.

Although the Koopman operator was developed in 1931 [12], it has recently begun to receive attention for its use in analyzing dynamical systems [4]. This has spawned many machine-learning approaches to Koopman operator theory [23][20]. Our research here is inspired most immediately by Azencot et al.'s [2] recent work on "consistent Koopman" networks, in which they use SST prediction as one of many tests for their novel method.

2   Data and Method

All code is implemented in Keras (v2.4) with a Tensorflow (v2.2) backend. Code is available at

https://github.com/JRice15/physics-informed-autoencoders.

2.1   Dataset

We use the NOAA Optimal-Interpolation (OI) SST High-Resolution dataset [19], which measures daily snapshots of sea-surface temperature at 0.25° spatial resolution globally, by interpolating satellite, buoy, and ship measurements to minimize bias.

For our experiments, we focus on a 70x150 region of this dataset consisting of the Gulf of Mexico and a portion of the Caribbean (see Fig. 2 for an example snapshot of this region). Though the thermal measurements are not actually photographic images, we may refer to them as "images" and individual locations as "pixels," as a useful shorthand.

We use three consecutive years (1095 days) as a training set, and the next year as a validation set during training. At test time, for longer term predictions, we use the three years following the training set as the test set.
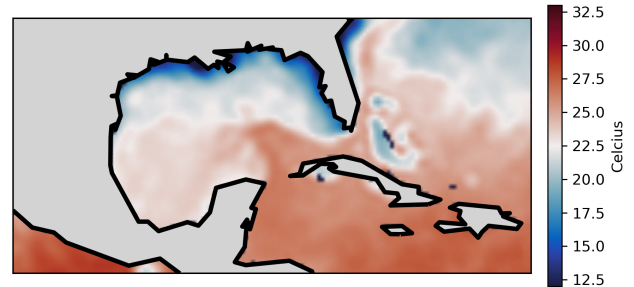

Figure 2: Example SST snapshot, day 140 in the training dataset

2.1.1   Pre- and Post- Processing

The measurements on land are set as zero, which cannot be confused as a temperature measurement as that sea surface temperature never occurs in the region. The data is then scaled to be approximately between -1 and 1. At test time, all locations where a zero (land) measurement appears are masked away, and do not contribute to error measurements. This is also done during training for convolutional networks, so that it does not contribute to the loss. We don't bother masking during training for fully-connected networks, as we find that it is trivial for them to learn the mask in the output layer (for example, by learning the weight to be zero and bias to the target mask value), however we still mask during testing so that averages are all computed identically.

2.2 Physics Informed Autoencoders

For more detailed descriptions of the following network architectures, see Appendix A. For training details, such as hardware used and learning rate schedules, see Appendix B.

2.2.1 Autoencoders: Introduction and Motivations

To create a neural networks for SST prediction, we utilize an autoencoder (AE) framework. Autoencoders are a class of neural network that compresses data to a much smaller form without losing much information, by learning an *encoder* (and its inverse, a *decoder*). Looking at our formulation of the problem in Eqn. (9), one will be positively astounded to see that it naturally suggests such an autoencoder architecture, with χ and its inverse acting as an encoder and decoder respectively, for each of which we will substitute a clearer notation: χ_e and χ_d. Furthermore, recent experiments (see Related Work, 1.2) suggest that sea-surface temperature evolution can be explained largely by low-dimensional dynamics, further pointing to an autoencoder approach.

Each network will thus consist of an encoder, χ_e, a dynamics matrix than can be applied multiple times, C, and a decoder that maps back to the space of the observables, χ_d:

$$\hat{y}_{t+k} = \chi_d \circ C^k \circ \chi_e(y_t) \qquad (10)$$

We will use $k = 14$ (days) for all of our experiments.

We develop and compare four implementations of Koopman autoencoders (KAEs) in sections 2.2.3–2.2.6, the relationships between which may be helpful to visualize:

	No Consistency	Consistency
Fully-Connected	Simple KAE	Cons KAE
Convolutional	Conv KAE	ConvCons KAE

Table 1: Koopman Network Relationships

2.2.2 Learning the Koopman Operator

In order for a network to learn a proper approximation of our Koopman-theoretic formulation in Eqn. (9), we first must instruct the network to generate accurate predictions given input y_t. It can do this by learning to minimize the difference between the ground truths y_{t+1} and its predictions $\hat{y}_{t+1}$:

$$\|\hat{y}_{t+1} - y_{t+1}\|_{\mathrm{MSE}} \qquad (11)$$

where $\|\cdot\|_{\mathrm{MSE}}$ denotes the mean-squared error (MSE), which squares its input element-wise and then averages over all dimensions and examples. However, since our goal is long-term prediction, we minimize the error over k steps into the future:

$$\mathcal{L}_{pred} = \frac{1}{k} \sum_{n=1}^{k} \|\hat{y}_{t+n} - y_{t+n}\|_{\mathrm{MSE}} \qquad (12)$$

In addition to producing accurate predictions, we must also ensure that the network's elements are properly corresponding to their role in our Koopman-theoretic framework. For C, this is achieved by making it any linear operation, in our case matrix multiplication. To ensure that χ_e and χ_d serve only to encode and decode, and not learn any dynamics – and conversely, that C does not contribute to the encoding or decoding – we implement a second loss term:

$$\mathcal{L}_{id} = \|\chi_d \circ \chi_e(y_t) - y_t\|_{\mathrm{MSE}} \qquad (13)$$

This enforces the relation that $\chi_d \circ \chi_e \approx I$, the identity.

The loss of the network is then the sum of the terms derived in Equations (12) and (13), weighted by respective hyperparameters λ:

$$\mathcal{L} = \lambda_{id}\mathcal{L}_{id} + \lambda_{pred}\mathcal{L}_{pred} \qquad (14)$$

This loss (sometimes with a few additional terms) will be used by all four networks below.

2.2.3 Simple KAE

The Simple Koopman Autoencoder (SimpleKAE) is an autoencoder with three encoder layers, a dynamics operation, and three decoder layers. Each layer of the encoder or decoder is a standard fully-connected layer, consisting of a matrix multiplication of the weight and the input and addition of a bias term, followed by a hyperbolic tangent activation:

$$x_{out} = \tanh(W_i x + b_i)$$

2.2.4 Consistent KAE

The Consistent Koopman Autoencoder (ConsKAE), recently introduced by Azencot et al. [2], extends the SimpleKAE's structure based on the assumption that in the latent space, dynamics evolve linearly both forward and backward in time. This assumption is not necessarily true in all dynamical systems; entropic systems may evolve deterministically forward in time, yet it may be impossible to reconstruct past states from present ones. Testing consistent Koopman networks will help reveal to what extent this assumption applies to this SST domain.

For this network, we assume there exists a backward dynamics Koopman operator $\mathcal{K}_\psi$ in addition to the forward dynamics of $\mathcal{K}_\varphi$, such that

$$f(z_{t-1}) = \mathcal{K}_\psi f(z_t) \qquad (15)$$

$\forall t \in \mathbb{Z}$, where ψ (the inverse of φ) maps z_t to z_{t-1}. While we could technically use C^{-1} to predict backward in our machine-learning approach, Azencot et al.
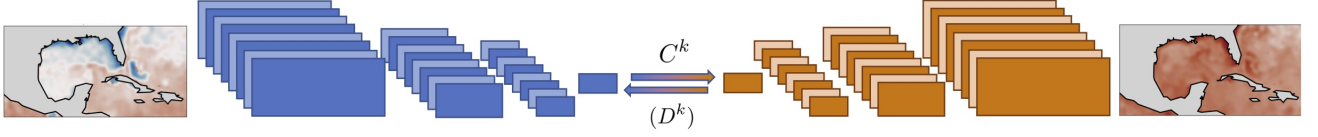
Figure 3: Convolutional Network Architecture. The encoder translates a 70x150 snapshot into an encoded 1x24 representation in the latent space, which is transformed forward in time by the dynamics C (and optionally backward by D, if it is a consistent Koopman network), and then decoded back to a 70x150 prediction.

find that the network is better able to learn backward dynamics via a seperate operation, D:

$$\hat{y}_{t-k} = \chi_d \circ D^k \circ \chi_e(y_t) \tag{16}$$

The error of backward predictions is computed analogously to Eqn. (12) for another loss term, $\mathcal{L}_{bwd}$. Additionally, in order to enforce that the backward and forward dynamics are *consistent*—meaning that they are approximate inverses—we use the following loss term:

$$\mathcal{L}_{cons} = \sum_{n=1}^{\kappa} \left[\frac{1}{2n} \|D_{n*}C_{*n} - I_n\|_F^2 \right. \\ \left. + \frac{1}{2n} \|C_{n*}D_{*n} - I_n\|_F^2 \right] \tag{17}$$

where D_{n*} denotes the upper n rows, and D_{*n} the leftmost n columns, of D. The $n \times n$ identity matrix is represented by I_n, and $\|\cdot\|_F$ is the Frobenius norm. For the derivation of this term, see section 4 of Azencot et al. [2].

The ConsKAE thus uses the following loss, with the two new loss terms weighted by respective λ hyperparameters:

$$\mathcal{L} = \lambda_{id}\mathcal{L}_{id} + \lambda_{pred}\mathcal{L}_{pred} + \lambda_{bwd}\mathcal{L}_{bwd} + \lambda_{cons}\mathcal{L}_{cons} \tag{18}$$

Azencot et al. [2] tested their model on this same SST domain, and was our inspiration for doing the same. We will compare their results with ours, though we utilized what we believe to be a more robust and informative testing scheme.

2.2.5 Convolutional KAE

The Convolutional Koopman Autoencoder (ConvKAE) uses the same dynamics as the SimpleKAE, but uses a convolutional encoder/decoder instead of fully-connected layers. The encoder consists of four layers, the first three of which consist of 16-filter convolutions and hyperbolic tangent activations followed by max-pooling, while the last is a 1-filter convolution alone. The decoder is the reverse, with transposed convolutions and upsampling replacing convolutions and max-pooling, respectively. See Fig. (3) for a graphic representation.

2.2.6 Convolutional-Consistent KAE

A combination of the previous two implementations, the Convolutional Consistent KAE (ConvConsKAE) uses both the convolutional encoding/decoding scheme of the Convolutional KAE, and paired forward and backward dynamics of the Consistent KAE.

2.3 Baselines

2.3.1 Persistence

Persistence is what it sounds like: a model that passes its input to its output, unchanged. Comparison to this baseline reveals the extent to which a network learns the actual dynamics of the system, and not simply a good encoding-decoding scheme.

2.3.2 Learned Average

The Learned Average model (LrndAvg) learns and outputs a temporally constant spatial distribution that is the same shape as its input. This distribution is learned by minimizing the mean-squared error between its inputs and its output, so that over the course of training a value akin to the weighted average at each spatial location is learned. Comparison to LrndAvg reveals whether has devolved to guessing the average and is not taking its input into account—or, perhaps even worse, whether it is making a legitimate effort that is still no better than guessing the average.

2.4 Testing Methods and Metrics

We trained each model on 12 different seed values, meaning 12 trials with unique network initializations. To test the models for long-term prediction, we record each seed's average performance on 180 day prediction over three years of test data (see section 2.1 for description of this dataset).

In testing, we evaluate the following metrics, in terms of a prediction $\hat{y}_t$ and target y_t.

- Relative prediction error (RelPred) measures the error scaled by the intensity of the target:

$$E_{relpred} = \frac{\|\hat{y}_t - y_t\|_F}{\|y_t\|_F} \tag{19}$$

This is used so that the error of hot and cold months can be compared on more equal footing.

- Celcius Mean-Absolute Error (MAE) is simply the average absolute difference between $\hat{y}_t$ and y_t, in units that are universally interpretable (unlike relative prediction error). We will prefer these units because of their universal meaning.

We also measured mean-squared error and mean-absolute error, though they will not appear here as they find they do not add any information not already contained in the two metrics above.

We calculate not only multiple metrics but multiple aggregations of those metrics among the 12 trials of each model: average, median, min, max, and standard deviation.

3 Experimental Results

3.1 Baselines

To begin, we evaluate the baselines' performances in Fig. 4. LrndAvg stays relatively flat (as expected), with the error hovering around 1.8 degrees. The Persistence model starts off well, but rapidly diverges to almost twice the error of the LrndAvg. While the LrndAvg model represents a good target to beat, Persistence is more of an upper-bound on how reasonably *poor* a model can perform.
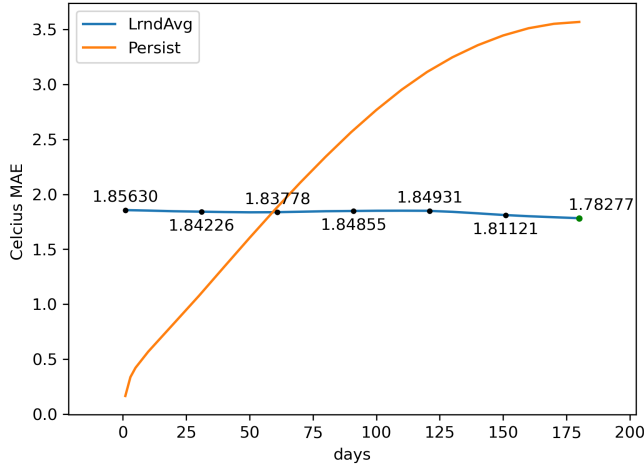

Figure 4: Celcius MAE of Baselines

3.2 Koopman Autoencoders

3.2.1 Initial Observations

In our experiments, the SimpleKAE appears to have performed the best, followed by the ConvKAE, ConsKAE, and lastly ConvConsKAE. This order is present in the average (Fig. 6) and median aggregations of model seeds. It is the same according to the minimum

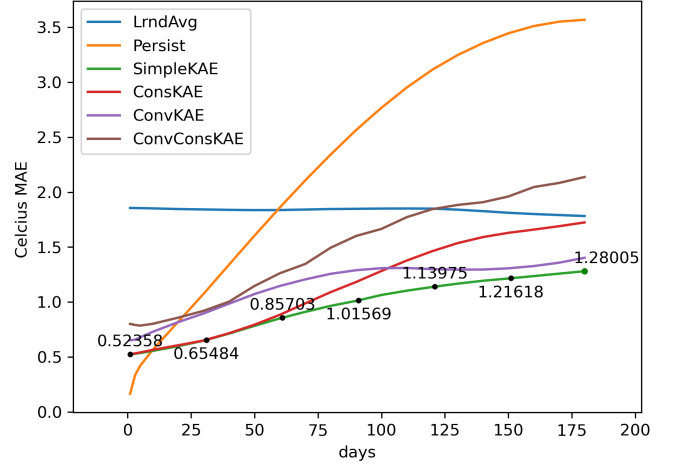aggregation (Fig. 7), except that ConvConsKAE performs better ConsKAE:

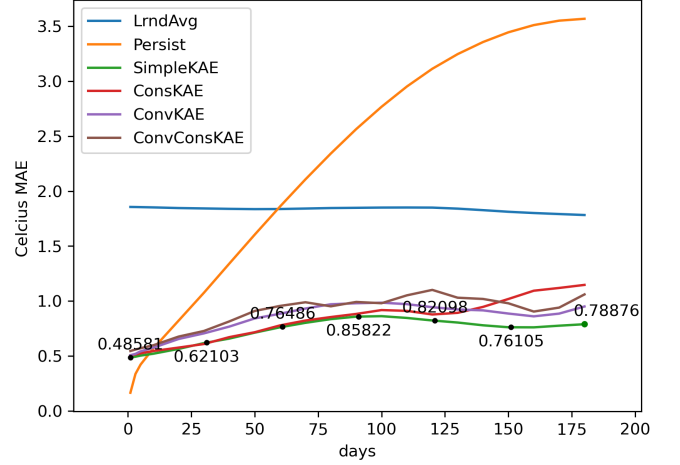Figure 6: Average Celcius MAE of Koopman Models

Figure 7: Minimum Celcius MAE of Koopman Models

The median and maximum errors of each model finish in identical order to the average. It should be noted that the median is noticeably lower than the average, suggesting high outliers. Using the relative prediction error metric yields a very similar picture, enough so that it need not be included.

3.2.2 Analysis of Variance

Complicating the above results, we must note that there is a relatively large variance within each model's 12 seeds. Take for example the spread of the best performing model's seeds, in Fig. 8, which confirms the existence of high outliers:
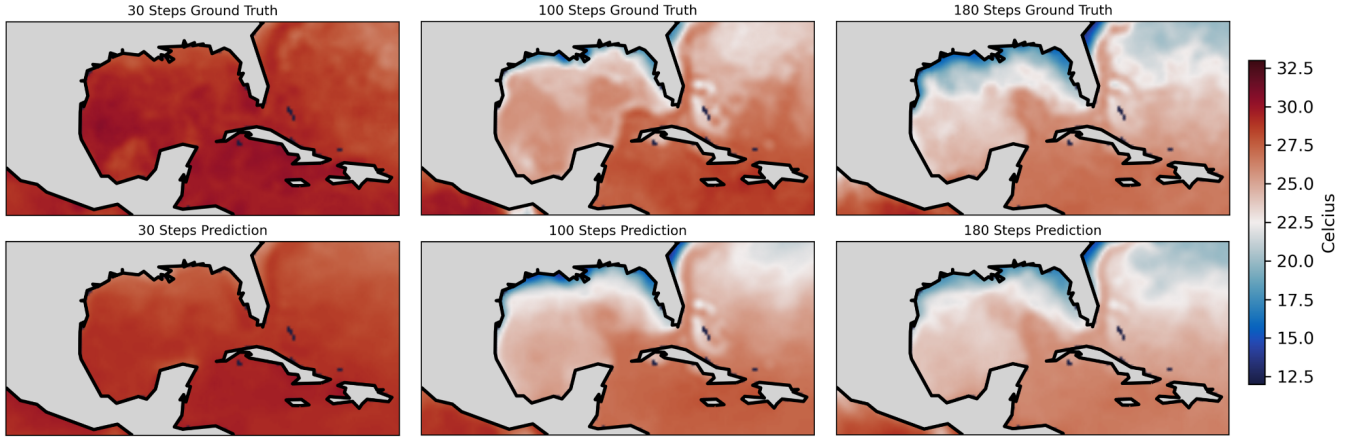
Figure 5: Selected day 30, 100, and 180 ground truths (top) and best SimpleKAE seed predictions (bottom). Though the predictions are lacking the fine-grain detail of the ground truth—which is understandable given that the predictions are decoded from a vector of length 12—the predictions follow the ground truth remarkably closely on the large scale. Note the similarity to ground truth at day 180, including a fairly accurate prediction of both the Loop Current and the Gulf Stream.
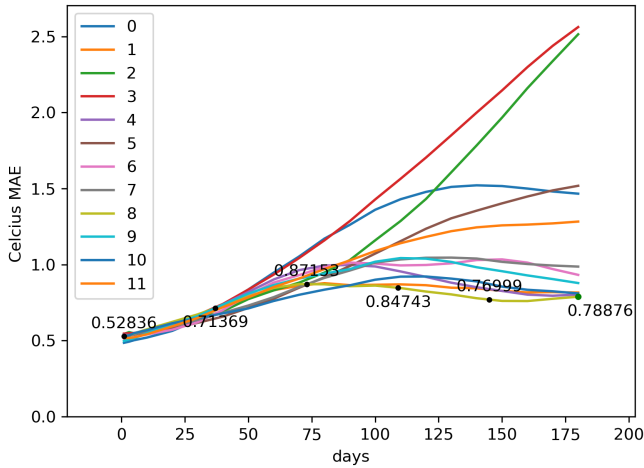

Figure 8: SimpleKAE Seeds

Figure 9: Average Celcius MAE with 90% Confidence

There is great variation around their mean of 1.28° Celcius, most notably on the high end with some seeds finishing at double the mean. Interestingly, it is nearly impossible to tell from their prediction performance during the first 25 or so days whether a given seed will perform well or poorly at 180 days. Similarly, training performance is not very indicative of how well a seed will perform at long-term prediction; all models achieved very similar training losses (which takes into account only 14 prediction days). This suggests that during training, the model is able to find minima in the optimization space for each seed that perform similarly in the short term, but that some of those minima are much more or less stable than others for long-term forecasting. Fig. 9 shows all models' variances, with 90% confidence bounds calculated using the standard deviation of the seeds from their mean.
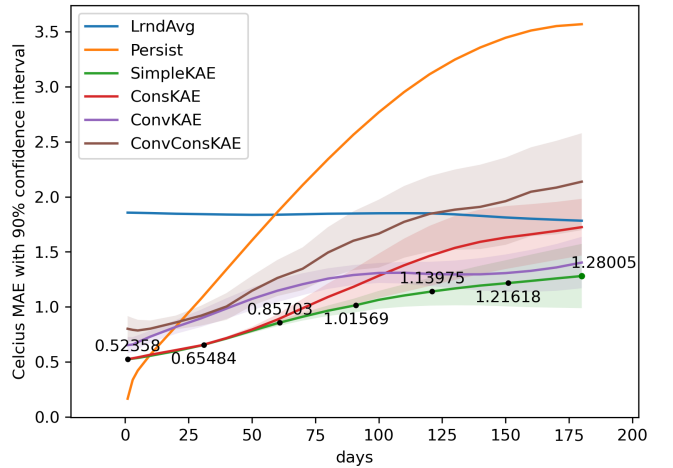
The key conclusion from from Fig. 9 is that many confidence intervals overlap with other means, suggesting that we may not be able to say we are certain they are significantly different. To this end, we apply Welch's t-test [26] to determine how confident we are that the day 180 means are in fact different:

	LrndAvg	Simple	Cons	Conv
LrndAvg	-	98.375	**28.378**	97.853
Simple	98.375	-	92.610	**40.467**
Cons	**28.378**	92.610	-	**85.780**
Conv	97.853	**40.467**	**85.780**	-
ConvCons	**78.852**	98.497	**80.076**	97.320

Table 2: Welch t-Test Confidences (two-tailed; values in percent) for the difference between means. We consider $\geq 90\%$ to be significant. Methods with non-significant differences are bolded.

Table 2 shows the confidences that the means of any two methods are in fact significantly different. We chose a confidence of 90% because of our small sample sizes, though a 95% confidence would not have made drastic changes (only the SimpleKAE v. ConsKAE significance).

From this data, we cannot conclude that the SimpleKAE performs any better than the ConvKAE. In fact, for all Koopman methods, we can see that the differences between one method and the next best method (according to their day 180 average MAE; see Fig. 6) are not significant. However, analysis in terms of the models' relationships per Table 1 reveals that there are statistically significant differences between consistent Koopman methods and their non-consistent counterparts (though there are not significant differences between non-convolutional and convolutional methods), at least at 90% confidence. Given that using 95% confidence bound would change this relation, we cannot be very confident that our results do simply reflect a bad batch of seeds for the consistent methods.

4 Discussion

Our results consistently validate a Koopman-theoretic approach to long-term SST forecasting, a challenging dynamical systems problem. The best initializations of all Koopman methods significantly outperformed both the Persistence and LrndAvg baselines.

Our results suggest that the consistent Koopman methods do not tend to perform as well as their non-consistent baselines in this SST domain. This is contradictory to the results of Azencot et al. [2], who reported the following data:

Model	RelPred Error		
	Min	Max	Avg.
SimpleKAE	0.315	1.062	0.701
ConsKAE	0.269	1.028	0.632

Table 3: Day 180 Relative Prediction Error on SST Dataset, with 18 seeds [2]

They report an improvement in relative prediction error of 0.069, but with a variation between min and max of 0.759. Without further information, it is difficult to know what significance to assign these results. We find the following, using the same relative prediction error metric:

Model	RelPred Error		
	Min	Max	Avg.
SimpleKAE	0.292	0.863	0.451
ConsKAE	0.412	1.151	0.583

Table 4: Our Day 180 Relative Prediction Error, with 12 seeds

We find approximately double the difference between means but in the opposite direction, though we do find a similar min-max difference. We attribute the differences to both our more robust testing scheme; while Azencot et al. test on the 30 days following the training set, we test on the three years following. However, they tested 18 seeds to our 12, suggesting our results may be more susceptible to a bad batch of seeds. There are also slight differences in the number of epochs trained and learning rate schedules of the models, though we do not expect these differences to have a great effect.

On other datasets where it is known that consistency applies—such as computer-generated data of ideal fluid flow past a cylinder, where we know that past states can be predicted from the present just as easily as future states—Azencot et al. demonstrated significant improvements over the SimpleKAE. Their results are much less conclusive in the more challenging real-world domain of SST forecasting, however. Given the unexpectedly large variation between seeds and our differing results, it is hard to draw conclusions as to whether the consistent method is better than the its simple counterpart. At the very least, it seems reasonable to say that Koopman consistency is not especially beneficial for forecasting in this domain—though possibly not especially harmful either.

Our experiments also suggest that the assumptions inherent in a convolutional encoder/decoder architecture are not especially beneficial. Although the *locality* of the data seems intuitively to be correct, it is possible that assuming *translation invariance* in the encoding scheme can make it more difficult to create encodings that extract the important information from all regions—e.g., the same filter may not be able to extract the most useful encoding from both the coast of Louisiana and the surroundings of Cuba.

Thus our primary conclusion is that informed machine-learning is only beneficial when the *right* prior information and assumptions are used. Though our experiments have validated the effectiveness of a Koopman-theoretic method, the additional assumptions inherent in convolutional and consistent Koopman methods do not seem to be especially beneficial in the complex and challenging domain of long-term sea-surface temperature forecasting.

5 Acknowledgements

References

[1] Hassan Arbabi. Introduction to koopman operator theory of dynamical systems, 06 2018.

[2] Omri Azencot, N Benjamin Erichson, Vanessa Lin, and Michael W Mahoney. Forecasting sequential data using consistent koopman autoencoders. *arXiv preprint arXiv:2003.02236*, 2020.

[3] M. Baymani, Sohrab Effati, Hamid Niazmand, and A. Kerayechian. Artificial neural network method for solving the navier–stokes equations. *Neural Computing and Applications*, 26:765–773, 05 2014.

[4] Marko Budišić, Ryan Mohr, and Igor Mezić. Applied koopmanism. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(4):047510, 2012.

[5] François Counillon and Laurent Bertino. High-resolution ensemble forecasting for the gulf of mexico eddies and fronts. *Ocean Dynamics*, 59(1):83–95, 2009.

[6] Emmanuel de Bezenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124009, 2019.

[7] Roger Edwards and Steven J Weiss. Comparisons between gulf of mexico sea surface temperature anomalies and southern us severe thunderstorm frequency in the cool season. In *Preprints, 18th Conf. on Severe Local Storms, San Francisco, CA, Amer. Meteor. Soc*, pages 317–320, 1996.

[8] N Benjamin Erichson, Lionel Mathelin, Zhewei Yao, Steven L Brunton, Michael W Mahoney, and J Nathan Kutz. Shallow learning for fluid flow reconstruction with limited sensors and limited data. *arXiv preprint arXiv:1902.07358*, 2019.

[9] N Benjamin Erichson, Michael Muehlebach, and Michael W Mahoney. Physics-informed autoencoders for lyapunov-stable fluid flow prediction. *arXiv preprint arXiv:1905.10866*, 2019.

[10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, May 2010. PMLR.

[11] Irhum Shafkat. Intuitively understanding convolutions for deep learning. `https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1`, 6 2018. Accessed 8/13/2020.

[12] Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the national academy of sciences of the united states of america*, 17(5):315, 1931.

[13] Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[14] L-Y Oey, Tal Ezer, George Forristall, C Cooper, Steven DiMarco, and S Fan. An exercise in forecasting loop current and eddy frontal positions in the gulf of mexico. *Geophysical Research Letters*, 32(12), 2005.

[15] Kalpesh Patil, M. C. Deo, and M. Ravichandran. Prediction of sea surface temperature by combining numerical and neural techniques. *Journal of Atmospheric and Oceanic Technology*, 33(8):1715–1726, 08 2016.

[16] Nancy N Rabalais, R Eugene Turner, and William J Wiseman Jr. Gulf of mexico hypoxia, aka "the dead zone". *Annual Review of ecology and Systematics*, 33(1):235–263, 2002.

[17] M Raissi, P Perdikaris, and GE Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, arxiv preprint (2017). *arXiv preprint arXiv:1711.10566*, 2017.

[18] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[19] Richard W. Reynolds, Thomas M. Smith, Chunying Liu, Dudley B. Chelton, Kenneth S. Casey, and Michael G. Schlax. Daily High-Resolution-Blended Analyses for Sea Surface Temperature. *Journal of Climate*, 20(22):5473–5496, 11 2007.

[20] Samuel E. Otto and Clarence W. Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

[21] Lynn K Shay, Gustavo J Goni, and Peter G Black. Effects of a warm oceanic feature on hurricane opal. *Monthly Weather Review*, 128(5):1366–1383, 2000.

[22] T Spindler, I Rivin, and L Burroughs. Noaa real-time ocean forecasting system rtofs-(atlantic): National weather ser-vice. *EMC/MMAB, Technical Procedure Bulletin*, 2006.

[23] Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems*, pages 1130–1140, 2017.

[24] FT Tangang, WW Hsieh, and B Tang. Forecasting the equatorial pacific sea surface temperatures by neural network models. *Climate Dynamics*, 13(2):135–147, 1997.

[25] Laura von Rueden, Sebastian Mayer, Jochen Garcke, Christian Bauckhage, and Jannis Schücker. Informed machine learning - towards a taxonomy of explicit integration of knowledge into machine learning, 03 2019.

[26] B. L. Welch. The generalization of 'student's' problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 01 1947.

[27] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating physics-based modeling with machine learning: A survey. *arXiv preprint arXiv:2003.04919*, 2020.

[28] Changjiang Xiao, Nengcheng Chen, Chuli Hu, Ke Wang, Zewei Xu, Yaping Cai, Lei Xu, Zeqiang Chen, and Jianya Gong. A spatiotemporal deep learning model for sea surface temperature field prediction using time-series satellite data. *Environmental Modelling & Software*, 120, 2019.

[29] Y. Yang, J. Dong, X. Sun, E. Lima, Q. Mu, and X. Wang. A cfcc-lstm model for sea surface temperature prediction. *IEEE Geoscience and Remote Sensing Letters*, 15(2):207–211, 2018.

[30] Q. Zhang, H. Wang, J. Dong, G. Zhong, and X. Sun. Prediction of sea surface temperature using long short-term memory. *IEEE Geoscience and Remote Sensing Letters*, 14(10):1745–1749, 2017.

A Network Architectures

Other than dynamics layers, all weights are initialized with the Glorot Normal (aka Xavier Normal) initializer [10], while all biases are initialized to zero. Forward dynamics weights are initialized by taking the singular value decomposition of a matrix generated via a random normal distribution, and then multiplying U and V^T to create the weight. Backward dynamics weights, when applicable, are initialized with the pseudo-inverse of the forward weights' transpose. Neither dynamics use a bias term.

A.1 Fully-Connected Networks

Component	Layer	Activation	(Output) Shape
Input	-	-	10500
Encoder	Fully-Connected 1	tanh	96
	Fully-Connected 2	tanh	96
	Fully-Connected 3	-	12
Dynamics	Fully-Connected	-	12
Decoder	Fully-Connected 1	tanh	96
	Fully-Connected 2	tanh	96
	Fully-Connected 3	-	10500

Table 5: Fully-Connected Architecture. This architecture was developed by Azencot et al [2]. They performed a parameter search when developing this model.

A.2 Convolutional Networks

Component	Layer	Kernel Size	Size Modification	Activation	(Output) Shape
Input	-	-	-	-	70x150
Encoder	Convolution 1	3x3	2x2 max-pooling	tanh	34x74x16
	Convolution 2	3x3	2x2 max-pooling	tanh	16x36x16
	Convolution 3	5x5	2x2 max-pooling	tanh	6x16x16
	Convolution 4	5x5	flattening	-	24
Dynamics	Fully-Connected	-	-	-	24
Decoder	Zero Padding	-	unflatten, 0x0x1x1 padding	-	7x17
	Convolution 1	5x5	-	tanh	7x17x16
	Convolution 2	5x5	2x2 upsampling	tanh	22x42x16
	Convolution 3	3x3	2x2 upsampling	tanh	48x88x16
	Convolution 4	3x3	2x2 upsampling	-	100x180
	Cropping & Masking	-	crop to 70x150	-	70x150

Table 6: Our Convolutional Architecture

At the beginning of the decoder, one row and column of zero padding is applied to the bottom and right of the 6x16 snapshot, and then at the end of the decoder the output is cropped to the target shape, to make the input and output shapes of the AE the same. All locations where land is present is masked away in both the network's output and in the target images, so that it does not affect the loss (see section 2.1.1). All upsampling uses the bilinear interpolation technique. Output shape is given in the form (height x width x channels).

Though we did not perform an exhaustive parameter search, we did explore many different avenues when developing this architecture. Here are some of the options we explored, and the reason why we did not use them when it was more complex than simply worse performance, as guidance for those seeking to improve upon our results:

- Batch normalization: we suspect this failed because of the seasonality of SST. When the data is normalized, it can be more difficult to tell the difference between warmer and colder states that are equally distributed.

- ReLU activation: We suspect that ReLU performed poorly because it loses information by design, which we want to minimize in the AE framework. Additionally, because of its simplicity relative to the hyperbolic tangent, we would likely need a deeper network to get similar results. We found the ReLU often devolved toward the LrndAvg's output.

- Various numbers of convolution output channels. We use 16/16/16/1; with less than that, such as 6/6/6/1, the AEs output was not as smooth, and did not perform as well. With larger than that, such as 16/32/64/1, the model became unstable, and would often diverge suddenly during long-term prediction at test time.

- Convolutional dynamics: we explored this briefly, since convolution is also a linear operator, but it performed very poorly.

- Average pooling and nearest-neighbor upsampling

- Kernel size ordering: we use the larger kernels for the more compressed sizes, to reduce the number of operations and thus increase the speed of the network.

B Training

Nvidia Tesla K80 and P100-PCIE GPUs were used for training. 14 forward prediction steps (and 14 backward, for consistent KAEs) were used for training losses. All models were trained for 2000 epochs (with early stopping) with the Adam optimizer, 0.5 gradient norm clipping, and a batch size of 64. Learning rate was on a schedule, starting at 0.01 and reducing by a factor of 0.6 ($lr_{new} = 0.4 \times lr_{old}$) at epochs 50, 400, 800, 1200, and 1600. Early stopping came into effect if 420 epochs passed with no improvement in validation loss.