Stochastic Adversarial Koopman Model for Dynamical Systems

Kaushik Balakrishnan
Ford Greenfield Labs, Palo Alto, CA
kbalak18@ford.com

Devesh Upadhyay
Ford Research, Dearborn, MI
dupadhya@ford.com

Abstract

We present a general model for non-linear dynamical systems that is based on the Koopman theory and combines an Autoencoder with a Generative Adversarial Network (GAN). We show that coupling a GAN as an additional loss term during training increases the accuracy of the model predictions, and demonstrate it on test problems in fluid dynamics, combustion, and Li-ion battery systems.

1 Introduction

Many engineering and scientific problems are solved using finite difference or finite volume methods, which are time consuming. Thus, recasting the problem in a lower-dimensional state space and advancing the solution in time in this reduced dimensional space is preferred, and one class of methods in this setting are based on the Koopman theory [1]. Several research works have renewed interest in Koopman models in recent years, as evidenced by [2, 3, 4, 5, 6, 7, 8, 9]. We extend the solution procedure using a stochastic latent embedding space, and the Koopman operator applies in this space. Specifically, the Koopman operator applies on the probability distribution of the latent embedding and advances it linearly in time, thereby learning a distribution of possible outcomes. If z is the latent embedding and $p(z)$ is its probability distribution, past studies [5, 6, 8, 9] apply the Koopman operator directly on z; in this study, we apply the Koopman operator on $p(z)$ instead. In addition, a GAN [10] discriminator is also used during training as an additional adversarial loss and this is found to robustify the predictions. A series of test problems are considered for the analysis.

2 The Koopman operator for dynamical systems

We consider dynamical systems of the form $x_{t+1} = F(x_t)$, where F is a non-linear function. In the Koopman model [1], the state vector x_t is mapped on to a Hilbert space $g(x_t)$ and the evolution of the system dynamics in time is linear, where the Koopman operator $\mathcal{K}$ advances the system as: $\mathcal{K}g(x_t) = g(F(x_t)) = g(x_{t+1})$. The system is then projected back to the physical state vector space using an inverse function g^{-1} [11, 12, 13]: $g^{-1}\ (\mathcal{K}g(x_t)) = x_{t+1}$. Recently, autoencoders [14] are used for this task with the encoder learning the mapping g and the decoder g^{-1} directly from data snapshots [5, 6, 8, 9]. During training, randomly sampled sequences of length n_S are considered from the data corpus: $x_t \cdots x_{t+n_S}$, and used to construct the vectors:

$$X = [x_t, x_{t+1}, \cdots, x_{t+n_S-1}]\,;\ X_{+1} = [x_{t+1}, x_{t+2}, \cdots, x_{t+n_S}]\,;\ X_{+1}^{\text{pred}} = \left[x_{t+1}^{\text{pred}}, x_{t+2}^{\text{pred}}, \cdots, x_{t+n_S}^{\text{pred}}\right]. \quad (1)$$

Here, X_{+1}^{pred} is the vector of the model's predictions for the states of the system at subsequent time steps. We take x_t as input and output the sequence X_{+1}^{pred} using the Koopman dynamics recursively n_S times.

When the codomain of the encoding function g is of finite dimension, the Koopman operator $\mathcal{K}$ reduces to a Koopman matrix K [5], and thus the Koopman operation reduces to that of a matrix multiplication. We will use z_t to represent the embedding at time instant t, i.e., $z_t = g(x_t)$ and $x_t = g^{-1}(z_t)$. [9] used a residual Koopman approach where the system dynamics instead of being represented as $z_{t+1} = \mathcal{K} z_t = K z_t$, was represented as $\mathcal{K} z_t = z_{t+1} = z_t + K z_t$ (note that $\mathcal{K}$ is the Koopman operator, whereas K is the corresponding Koopman matrix). Here, the Koopman model learns the residual change required to advance the system in time in the latent embedding space. With this change, the recursive prediction of the future states of the system in the embedding space is:

$$Z_{+1}^{\text{pred}} = \left[\mathcal{K}\, z_t, \mathcal{K}^2\, z_t, \cdots, \mathcal{K}^{n_S}\, z_t\right], \tag{2}$$

where $\mathcal{K}^j$ represents the application of the Koopman opertaor $\mathcal{K}$ j times, and n_S is the desired sequence length.

In [9], a Generative Adversarial Network (GAN) [10] was also coupled to the Deep Koopman model by including an additional loss to train the "generator" networks. It has been demonstrated in computer vision applications that coupling a GAN Discriminator with an autoencoder can improve the quality of samples output from the autoencoder [15]. This is because the feature representations learned by the GAN discriminator also provide additional learning signals for the primary neural networks, and this improves the output quality [15]. In NLP applications too, the use of GAN has provided regularization and improved the quality of model predictions [16]. These observations also extend to dynamical system modeling, as we will demonstrate in this paper.

In this setting, GAN losses are then constructed using X, X_{+1} and X_{+1}^{pred}. Two concatenated pairs are used: the "real" (X, X_{+1}) and "fake" $(X, X_{+1}^{\text{pred}})$. These pairs are fed into the GAN discriminator which outputs a single real value $D(\cdot)$ from which one can construct the GAN objective, following the Wasserstein GAN [17] approach due to its robustness against mode collapse:

$$L^{\text{GAN objective}} = \underset{x\in(X,X_{+1})}{\mathbb{E}}\left[D(x)\right] - \underset{\widetilde{x}\in(X,X_{+1}^{\text{pred}})}{\mathbb{E}}\left[D(\widetilde{x})\right]. \tag{3}$$

To obtain the Koopman matrix K, an auxiliary neural network is used to output K as a function of x_t, similar to [5, 9]. We consider a tridiagonal structure for the Koopman matrix, since the number of non-zero entries is one order of magnitude fewer than a full Koopman matrix. That is, if M is the dimension of the latent embedding, then a full Koopman matrix will have $M \times M$ non-zero entries whereas a tridiagonal Koopman matrix will only have $3M - 2$ non-zero entries. See [18] for more discussions on the choice of using a tridiagonal form for the Koopman matrix, albeit their tridiagonal form had additional constraints not considered here.

The latent embedding z_t is modeled as a Gaussian random variable; thus, z_t involves two components μ_t^z and σ_t^z, each of which $\in \mathcal{R}^M$. The auxiliary network takes μ_t^z and σ_t^z as input and outputs two Koopman matrices K_μ and K_σ which are used to construct the distribution for the next time step:

$$\mu_{t+1}^z = \mu_t^z + K_\mu \mu_t^z; \;\; ln\, \sigma_{t+1}^z = ln\, \sigma_t^z + K_\sigma ln\, \sigma_t^z. \tag{4}$$

Once the distributions for the future states are obtained, we sample $z_{t+1} = \mathcal{N}(\mu_{t+1}^z, \sigma_{t+1}^z)$, which is then passed on to the decoder. We call the model as the Stochastic Adversarial Koopman (SAK) model and a schematic is shown in Fig. 1. A total of four neural networks are used: the encoder, decoder, auxiliary network and the discriminator and their architectures are summarized in the Supplementary Materials. The loss functions used to train the networks are also elaborated in the Supplementary Materials. The main contributions of this paper are summarized as follows:

- Model z_t as a stochastic random variable, i.e., a Gaussian; thus, the embedding is more precisely $p(z_t)$; we have thus defined a Koopman operator in stochastic embedding space
- Use Maximum Mean Discrepancy (MMD) loss for $p(z_t)$, similar to a Wasserstein Autoencoder [19]; this we believe is new in a Koopman setting
- Use a GAN Discriminator as an additional adversarial loss term; this acts as a regularizer and improves the Koopman model predictions; this is novel in the Koopman family of algorithms

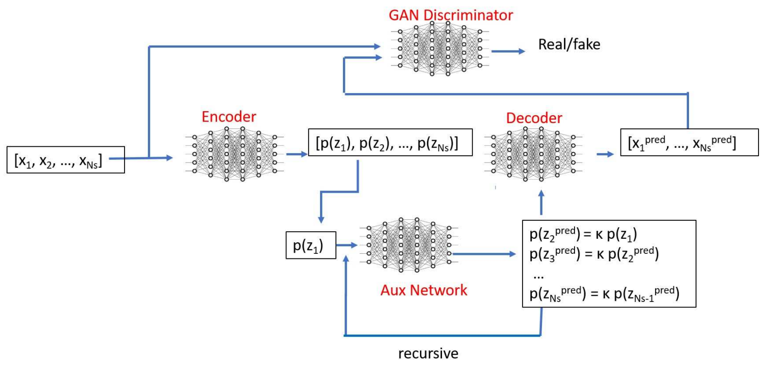
GAN Discriminator
Real/fake
Encoder
Decoder
[x_1, x_2, ..., x_Ns]
[p(z_1), p(z_2), ..., p(z_Ns)]
[x_1^pred, ..., x_Ns^pred]
p(z_1)
Aux Network
p(z_2^pred) = κ p(z_1)
p(z_3^pred) = κ p(z_2^pred)
...
p(z_Ns^pred) = κ p(z_Ns-1^pred)
recursive

Figure 1: Schematic of the Stochastic Adversarial Koopman Model showing the different neural networks and the connections.

3 Experiments

We will now demonstrate the robustness of the Stochastic Adversarial Koopman (SAK) model on a variety of engineering test problems: (1) von Karman vortex shedding behind a cylinder [20]; (2) Flame ball-vortex interaction [21]; and (3) Doyle-Fuller-Newman (DFN) model for Lithium ion batteries [22, 23]. See the Supplementary Materials for more details.

von Karman vortex shedding
The hyperparameter λ_{GAN} (see Supplementary Materials) decides the tradeoff between the GAN loss in comparison with the other supervised loss terms; λ_{GAN} = 0.01 was found to be the optimal value for training the SAK model on the von Karman vortex shedding data. The test time model predictions and ground truth of the velocity components u, v and pressure p at a time instant is shown in Fig. 2 and the model predictions are very similar to the ground truth values, with very low error. To better understand the role of the GAN loss, we repeat the training from scratch with the GAN loss turned off, i.e., λ_{GAN} = 0.0. In Fig. 3 (a), we compare the test time prediction errors with and without the GAN loss term, and λ_{GAN} = 0.01 has lower prediction errors for most time steps. We also investigated the magnitudes of K_μ and K_σ after training (not presented for brevity), and observed the entries along the diagonal to be preponderant for both K_μ and K_σ. Although a few off-diagonal entries also had relatively large magnitude values, no specific structure was evident other than the diagonal preponderance. In [9] a full Koopman matrix ($M \times M$) was output by the auxiliary network, but in the tridiagonal version considered in this study, the Koopman matrices K_μ and K_σ have $3M - 2$ non-zero entries each, thus the number of non-zero entries in the Koopman matrices are now fewer: $O(M^2) \rightarrow O(M)$. We consider another training case with a full Koopman matrix, and compare the mean absolute errors at each time step for the predictions using both formulations in Fig. 2 (b). The errors are nearly similar in magnitude for both formulations despite having one order of magnitude difference in the number of non-zero values in K_μ and K_σ. The tridiagonal Koopman demonstrated here can be seen as an alternative to the Jordan/diagonal block structures for K used in [5, 7].

Flame ball-vortex interaction
The results are presented in Fig. 4 and as evident the SAK model is able to accurately capture the shape of the flame ball as the vortex distorts it counter-clockwise. At late time the surface area of the flame increases, which further consumes more of the pre-mixed fuel-oxidizer mixture. The errors (not shown) are very small compared to the magnitudes of the variables, demonstrating the efficacy of the SAK model.

Conditional SAK model on the DFN data
For this problem, we consider a conditional Koopman model where the Koopman matrices are conditioned on the applied current density I_{app} (in A/m^2), which is fed as input to the auxiliary neural network. The training set consists of the battery discharge states (each state comprises of 6 variables) in time for I_{app} in the range 10-35 A/m^2. n_S = 16 at the beginning of the training and is increased by 2.5% every 20k iteration steps until n_S = 256, after which it is held fixed. We consider a total of 1 million training iterations, as this problem has more variety in the training data, necessitating

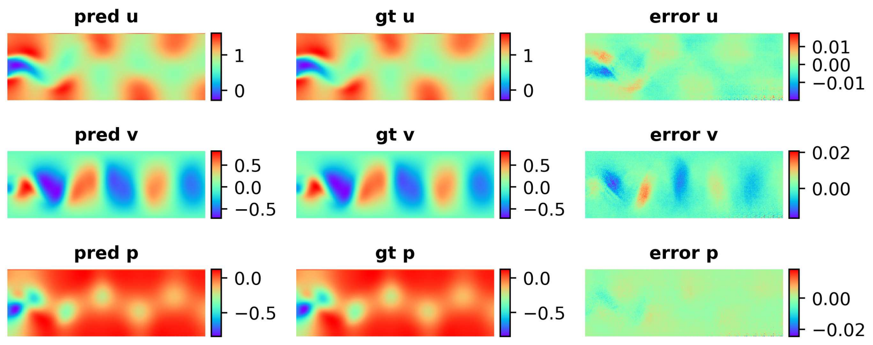

Figure 2: Predictions of u, v and p for the von Karman vortex shedding problem behind a cylinder with the SAK model. SAK model predictions: "pred"; ground truth: "gt".

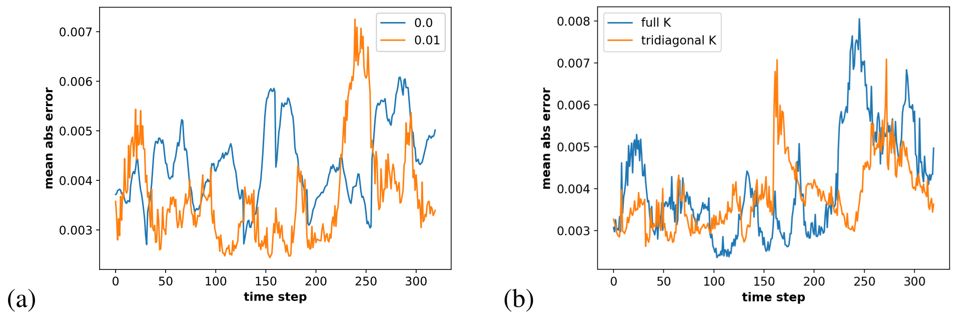

Figure 3: (a) Prediction error for each time snapshot for the von Karman vortex problem with λ_{GAN} = 0.0 and 0.01; (b) mean absolute error for every time snapshot w.r.t. the ground truth for the full and tridiagonal Koopman matrices.

longer training. After the conditional SAK model is trained on the training set, we use it to make predictions for I_{app} values in the test set. The model predictions at time instants: 258, 600, 1100 and 2000 seconds during the battery discharge for I_{app} = 31 A/m^2 are presented in Fig. 5 (in blue), along with the ground truth values (in orange). The model predictions are very accurate, although we do observe some slight differences for the variables "C Sol Surf" and "j main." Thus, the conditional model is able to learn the dynamics of battery discharge.

4 Discussion

A novel model based on the Koopman family of algorithms to train dynamical systems is developed, using a Gaussian stochastic embedding of an autoencoder in the latent space, which is advanced in time. During training, the model also couples a GAN discriminator for an adversarial loss term that is found to improve the accuracy, provided an optimal value for the trade-off parameter λ_{GAN} is used. This optimal value varies for different problems and requires experimentation, which needs to be addressed in future studies. We also extend the model to a conditional Koopman setting where additional input parameters are supplied to the auxiliary network so that the Koopman matrices are conditioned on these inputs. The SAK model predictions are robust and can be used for predicting the dynamics of many real-world systems. While the Koopman family of models take a few days to train, the main advantage of such Reduced Order Models (ROM) is in the design of engineering systems where such models can be trained once from data, and used to make inferences in a matter of $\sim$ 100 milliseconds, which will greatly help engineers in the design iteration process. Our experiments demonstrate inference speed-ups on the order of 10-100X for the problems considered here. The speedup advantages will be more preponderant for 3D problems, which is of interest in a future study.

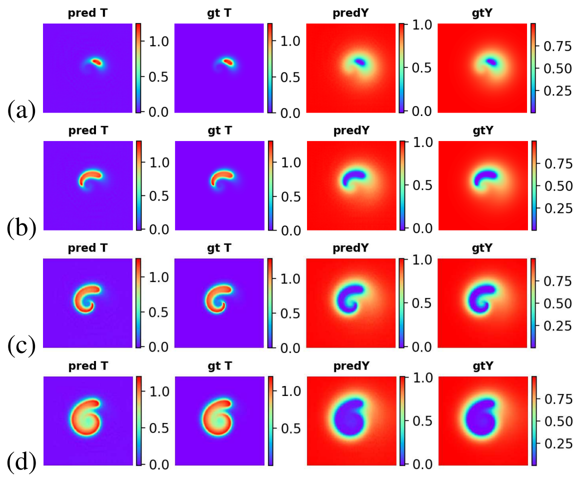

Figure 4: Predictions of T and Y for the flame ball-vortex interaction problem with the SAK model at time step number: (a) 22, (b) 53, (c) 78 and (d) 112. SAK model prediction: “pred"; ground truth: “gt".

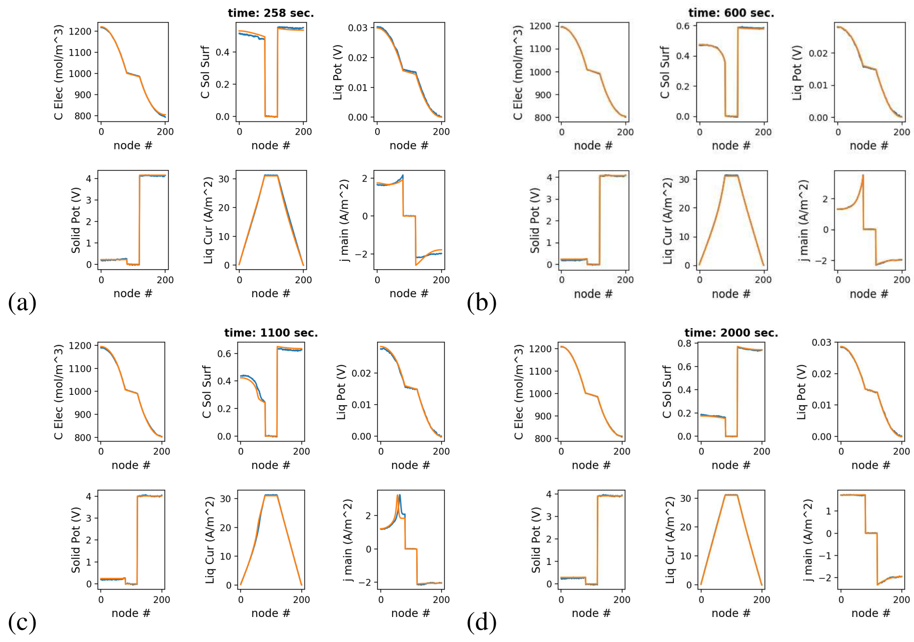

Figure 5: Predictions of the DFN battery model variables for I_{app} = 31 A/m^2 with the conditional stochastic adversarial Koopman model at time instants: (a) 258, (b) 600, (c) 1101 and (d) 2000 seconds during discharge. Ground truth: orange; SAK model predictions: blue.

References

[1] B. O. Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences USA*, 17:315–318, 1931.

[2] I. Mezic and A. Banaszuk. Comparison of systems with complex behavior. *Physica D*, 197:101–133, 2004.

[3] H. Arbabi and I. Mezić. Study of dynamics in post-transient flows using koopman mode decomposition. *Physical Review Fluids*, 2:124402, 2017.

[4] E. Yeung, S. Kundu, and N. Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. *arXiv:1708.06850 [cs.LG]*, 2017.

[5] B. Lusch, J. N. Kutz, and S. L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018.

[6] N. Takeishi, Y. Kawahara, and T. Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[7] A. Salova, J. Emenheiser, A. Rupe, J. P. Crutchfield, and R. M. DSouza. Koopman operator and its approximations for systems with symmetries. *Chaos*, 29:093128, 2019.

[8] J. Morton, F. D. Witherden, A. Jameson, and M. J. Kochenderfer. Deep dynamical modeling and control of unsteady fluid flows. *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[9] K. Balakrishnan and D. Upadhyay. Deep adversarial koopman model for reaction-diffusion systems. *arXiv:2006.05547 [cs.CE]*, 2020.

[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[11] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic mode decomposition with control. *arXiv:1409.6358 [math.OC]*, 2014.

[12] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. Dynamic mode decomposition: Data-driven modeling of complex systems. *Society for Industrial and Applied Mathematics*, 2016.

[13] E. Kaiser, J. N. Kutz, and S. L. Brunton. Data-driven discovery of koopman eigenfunctions for control. *arXiv:1707.01146 [math.OC]*, 2017.

[14] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 7 2006.

[15] A. B. L. Larsen, S. K. Sonderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv:1512.09300 [cs.LG]*, 2016.

[16] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio. Professor forcing: A new algorithm for training recurrent networks. *arXiv:1610.09038 [stat.ML]*, 2016.

[17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[18] S. Pan and K. Duraisamy. Physics-informed probabilistic learning of linear embeddings of nonlinear dynamics with guaranteed stability. *SIAM Journal on Applied Dynamical Systems*, 19(1):480–509, 2020.

[19] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein auto-encoders. *arXiv:1711.01558 [stat.ML]*, 2018.

[20] T. von Karman. Aerodynamics. *McGraw-Hill*, 1963.

[21] O. Roussel and K. Schneider. An adaptive multiresolution method for combustion problems: application to flame ball–vortex interaction. *Computers and Fluids*, 34(7):817–831, 2005.

[22] M. Doyle, T. F. Fuller, and J. Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of The Electrochemical Society*, 140(6):1526, 1993.

[23] T. F. Fuller, M. Doyle, and J. Newman. Simulation and optimization of the dual lithium ion insertion cell. *Journal of The Electrochemical Society*, 141(1):1, 1994.

[24] M. Doyle, T. F. Fuller, and J. Newman. Dualfoil code v5.1 (university of california, berkeley). *http://www.cchem.berkeley.edu/jsngrp/fortran_files/fortran.html*.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[26] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *3rd International Conference for Learning Representations (ICLR)*, 2015.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? blue[Yes]
   (b) Did you describe the limitations of your work? blue[Yes] See the Discussion section
   (c) Did you discuss any potential negative societal impacts of your work? gray[N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? blue[Yes] This work does not involve ethical issues
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? gray[N/A]
   (b) Did you include complete proofs of all theoretical results? gray[N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? blue[Yes] The code will be posted on github before the conference
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? blue[Yes] Supplemental Materials specifies this information
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? orange[No] Only errors w.r.t. the model parameter λ_{GAN} is presented in Fig. 3
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? blue[Yes] This is mentioned in the Supplementary Materials
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? blue[Yes] We cite [8, 5]
   (b) Did you mention the license of the assets? gray[N/A] The model is in public domain
   (c) Did you include any new assets either in the supplemental material or as a URL? blue[Yes] The models are discussed in the supplemental material
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? gray[N/A] All the data is generated by us using open-source codes and we cite them
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? gray[N/A] Data does not contain any offensive content
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? gray[N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? gray[N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? gray[N/A]

Supplementary Materials

1. von Karman vortex shedding behind a cylinder

The classical von Karman vortex shedding behind a cylinder [20] is widely used for validating many Computational Fluid Dynamics (CFD) codes and is of relevance to both the aerospace and automotive industries for aerodynamic drag reduction. The Navier-Stokes equations of fluid dynamics are solved in a 2D domain for flow past a cylinder from left to right at a Reynolds number $Re = 150$. The domain size is 22×4 and the velocity of the flow at the input is 1 (all units are non-dimensionalized). The domain is discretized using a 660×120 grid and the CFD time step $\Delta t = 0.01$. The open source code: https://github.com/dorchard/navier is used for solving the Navier-Stokes equations and generating the data corpus. The flow-field comprises of three variables: u, v, p, where u and v are the

x and y velocity components and p is the pressure. For training the SAK model, only the region immediately behind the cylinder where the vortex shedding is preponderant, is considered; specifically, we consider 360 cells in the horizontal immediately behind the cylinder and the entire 120 cells in the vertical directions. Every 4-th CFD solution snapshot is saved, and thus Δt = 0.04 for the Koopman analysis. We consider 400 time snapshots for training the SAK model, which corresponds to approximately 3.5 cycles of vortex shedding. The SAK model training takes approximately 3 days on a single GPU.

2. Flame ball-vortex interaction
Flame-vortex interaction is a classical engineering problem in combustion engines, where vortices can distort a flame, thereby enhancing the mixing of fuel with the oxidizer. We consider a simple 2D model [21] for dimensionless temperature T and mass fraction/concentration of premixed gas Y and this consists of a system of 2 PDEs. For brevity, these equations are not presented and can be found in [21]. Outflow boundary conditions are used at all four boundaries for the CFD analysis. The initial conditions are are not presented here and can be found in [21]. We use a 2^{nd} order central scheme in space and a 3^{rd} order Runge-Kutta scheme in time to solve the system of equations. For the CFD analysis, the domain size is 20×20 and is discretized using a 512×512 mesh for the finite difference method, with a time step Δt = 10^{-5} for 140k time steps. The solution at every other node is saved once every 1000 time steps, and so the data corpus for training the SAK model is of size 256×256×2 (the 2 is for two variables: T, Y) for 141 snapshots. The SAK model training takes approximately 3 days on a single GPU.

3. Doyle-Fuller-Newman Li-ion battery model
The Doyle-Fuller-Newman (DFN) model [22, 23] consists of a system of PDEs in 1D to solve for the Li-ion concentrations and potential in the electrode particles and the electrolyte of a Li-ion battery. We use the Dualfoil code v5.1 open sourced from the DFN authors [24] for generating the data corpus. Specifically, we consider a fully charged Li-ion battery and vary the applied current density, I_{app}, in the range 10-35 A/m^2, and make predictions of the battery potentials in the electrode and electrolyte, the Li-ion concentrations, and the current as the batetry discharges in time. The cross-section of the battery is discretized using 201 nodes and the Dualfoil code outputs 6 variables: concentration of the Li-ions in the electrolyte (in mol/m^3), concentration of Li-ions at the surface of the solid particles (non-dimensionalized), potentials in the liquid and solid phases (both in Volts), liquid phase current density and current density "j main" (both in A/m^2). These variables are saved every 1 second time interval during the discharge of the battery until the voltage drops to 3 V, and the total number of time snapshots varies depending on I_{app}; for instance, the total number of snapshots is 8454 for I_{app} = 10 A/m^2 and is 2197 for I_{app} = 35 A/m^2, with the other cases in between. More details on the battery modeling system and solution procedures can be found in the Dualfoil manual [24]. We train the SAK model with the Koopman matrix conditioned on I_{app} by supplying this as an input to the auxiliary network. The SAK model is trained on a training set comprising of 18 different I_{app} values in the range $10 \leq I_{app} \leq 35$ A/m^2 and used to make predictions of the state of the battery at different time instants for a value of I_{app} not used in the training. The SAK model training takes approximately 7 days on a single GPU.

Neural Network Architectures

The neural network architecture used in the analysis is summarized here. Note that we have a total of 4 neural networks: *Encoder*, *Decoder*, auxiliary network *AUX* (to obtain K matrices) and the GAN discriminator *DISC*. We will use several different deep learning building blocks: batch normalization (*BN*), Dropout (*Dropout*), convolutional (*conv*) and deconvolutional (*dconv*) operators, and the Relu (*Relu*) activation function. The notation *conv*(k,f,S,s) is used for a convolutional layer with kernel size k, f filters, same padding (identified by S) and a stride of s. The notation *Dense*(n) is used to refer to a fully connected dense layer with n neurons. We first define a bottleneck layer for *Encoder* with N_f filters as input, *BottleNeck*^{e}(N_f), comprising of the following in the same order: *BottleNeck*^{e}(N_f) = *BN* $\rightarrow$ *Relu* $\rightarrow$ *conv*(1,N_f/2,S,1) $\rightarrow$ *BN* $\rightarrow$ *Relu* $\rightarrow$ *conv*(3,N_f/2,S,1) $\rightarrow$ *BN* $\rightarrow$ *Relu* $\rightarrow$ *conv*(1,N_f,S,1). *Encoder* consists of 5 layers of convolutional operations supplemented with bottleneck layers added residually, similar to Resnet [25]. For ease of notation, we will refer to *Encoder*'s residual block as *RES*^{e}(N_f) = *conv*(3,N_f,S,2) + *BottleNeck*^{e}(N_f). Specifically, *Encoder* consists of 5 residual layers in succession: *RES*^{e}(64) $\rightarrow$ *RES*^{e}(128) $\rightarrow$ *RES*^{e}(256) $\rightarrow$ *RES*^{e}(512) $\rightarrow$ *RES*^{e}(512), followed by a *Relu* and a flattening operation. This is then fed into two *Dense*(M) layers to output the μ and σ of the embedding vector, where the dimension M = 64 is used throughout this study.

For the *Decoder* we define a similar bottleneck layer with N_f filters, albeit this time using deconvolutional operations: *BottleNeck*$^d(N_f)$ = *BN* $\rightarrow$ *Relu* $\rightarrow$ *dconv*(1,N_f/2,S,1) $\rightarrow$ *BN* $\rightarrow$ *Relu* $\rightarrow$ *dconv*(3,N_f/2,S,1) $\rightarrow$ *BN* $\rightarrow$ *Relu* $\rightarrow$ *dconv*(1,N_f,S,1). For *Decoder*, we first add the input to the bottleneck layer akin to Resnet [25], which is then passed though a deconvolutional layer, like so: *RES*$^d(N_f)$ = input + *BottleNeck*$^d(N_f)$ $\rightarrow$ *dconv*(3,N_f,S,2). *Decoder* starts with *Dense*$(\cdot)$ with the number of neurons used being the same as the dimension of the *Encoder*'s flattened output. This is reshaped as appropriate and is followed by 5 layers of the *Decoder*'s residual blocks in succession: *RES*^{d}(512) $\rightarrow$ *RES*^{d}(256) $\rightarrow$ *RES*^{d}(128) $\rightarrow$ *RES*^{d}(64) $\rightarrow$ *RES*$^d(n_{out})$. Here, n_{out} is the number of output channels in the data. Note that some problems are 1D whereas others are 2D, and so the appropriate *conv* and *dconv* API calls are used.

For the *AUX* network, we define a fully connected layer with N neurons as *FC*(N) = *Dense*(N) $\rightarrow$ *Relu* $\rightarrow$ *Dropout*. For *Dropout*, we set the probability of keeping the activations to 0.8 at training, and 1.0 at testing. *AUX* network consists of 4 fully connected layers: *FC*(128) $\rightarrow$ *FC*(256) $\rightarrow$ *FC*(512) $\rightarrow$ *Dense*$(2n_{dec})$ (the 2 factor arises since we consider two Koopman matrices K_μ and K_σ). For the full Koopman matrices, n_{dec} = M^2, whereas for the tridiagonal Koopman matrices, n_{dec} = $3M$-2.

For *DISC*, we will use the Leaky Relu activation function, denoted as *LRelu*, with a slope of 0.2 in the negative side. We define a block *B*$^{\text{DISC}}(N_f)$ as *conv*(5,N_f,S,2) $\rightarrow$ *BN* $\rightarrow$ *LRelu*. *DISC* is then constructed as: *conv*(5,64,S,2) $\rightarrow$ *LRelu* $\rightarrow$ *B*$^{\text{DISC}}$(128) $\rightarrow$ *B*$^{\text{DISC}}$(256) $\rightarrow$ *B*$^{\text{DISC}}$(512). The output is then reshaped and passed to a *Dense*(1) without any activation function to represent the Wasserstein distance.

Adam [26] optimizer is used to train the neural networks with a learning rate of 1×10^{-5}. The total number of iterations used for training varies for the different problems, but is usually in the order of 100k-500k, where at each iteration step one sequence of n_S contiguous snapshots are randomly sampled from the data corpus and used to train the networks.

Loss terms

The encoder and decoder are represented as $g(\cdot)$ and $g^{-1}(\cdot)$, respectively. We use the mean squared error (MSE) and the maximum mean discrepancy (MMD) [19] to construct different loss terms: (1) reconstruction loss L^{recon}, (2) prediction loss L^{pred}, (3) code loss L^{code}, (4) gradient loss L^{grad}, (5) L_2 regularization loss L^{reg}, (6) GAN loss L^{GAN}, and (7) discriminator loss L^{disc}. These different losses are summarized below:

$$L^{\text{recon}} = \| x_t - g^{-1}g\left(x_t\right) \|_{\text{MSE}} \tag{5}$$

$$L^{\text{pred}} = \frac{1}{n_S}\sum_{m=1}^{n_S} \| x_{t+m} - g^{-1}\left(\mathcal{K}^m g\left(x_t\right)\right) \|_{\text{MSE}} \tag{6}$$

$$L^{\text{code}} = \text{MMD}[g\left(x_{t+m}\right), \mathcal{K}^m g\left(x_t\right)] \tag{7}$$

$$\begin{aligned} L_j^{\text{grad}} &= \frac{1}{n_S}\sum_{m=1}^{n_S} \| \nabla_j \left[x_{t+m} - g^{-1}\left(\mathcal{K}^m g\left(x_t\right)\right)\right] \|_{\text{MSE}}, j\text{=1, 2, 4} \\ L^{\text{grad}} &= \lambda_1 L_1^{\text{grad}} + \lambda_2 L_2^{\text{grad}} + \lambda_4 L_4^{\text{grad}} \end{aligned} \tag{8}$$

$$L^{\text{reg}} = \lambda_{\text{reg}} \sum w_i^2 \tag{9}$$

$$L^{\text{GAN}} = \underset{\widetilde{x}\in(X, X_{+1}^{\text{pred}})}{\mathbb{E}} [D(\widetilde{x})] \tag{10}$$

$$L^{\text{disc}} = \underset{\widetilde{x} \in (X, X_{+1}^{\text{pred}})}{\mathbb{E}} [D(\widetilde{x})] - \underset{x \in (X, X_{+1})}{\mathbb{E}} [D(x)] \tag{11}$$

Note that the losses L^{recon}, L^{pred}, and L^{code} were also considered in [5, 9], but MSE was considered for all these terms. In this study, since the latent embedding is stochastic, we use the MMD loss [19]:

$$L^{\text{code}} = \frac{1}{n_S(n_S-1)} \sum_{l,j,l \neq j} f\left(z_l, z_j\right) + \frac{1}{n_S(n_S-1)} \sum_{l,j,l \neq j} f\left(z_l^{\text{pred}}, z_j^{\text{pred}}\right) - \frac{2}{n_S^2} \sum_{l,j} f\left(z_l, z_j^{\text{pred}}\right), \tag{12}$$

where z_l is sampled from $\mathcal{N}(g\left(x_{t+l}\right))$ and z_l^{pred} from $\mathcal{N}(\mathcal{K}^l g\left(x_t\right))$. Note that we essentially have samples from two distributions $g\left(x_{t+l}\right)$ and $\mathcal{K}^l g\left(x_t\right)$ that need to be synchronized and so the Maximum Mean Discrepancy (MMD) [19] is one such cost function. Note that the function g takes x_t as input and outputs μ_t^z and σ_t^z. And g^{-1} takes z_t (or z_{t+1}) as input to output x_t (or x_{t+1}). f is the inverse multiquadratics kernel $f(x,y) = C/(C + \parallel x - y \parallel_2^2)$ with C being a constant [19]. In the above equations, $\mathcal{K}^m$ denotes the application of the Koopman operator m times. Gradient losses are also used to improve the overall quality of the output.

The encoder, decoder and the auxiliary network—which we can refer to as the "generator" in GAN parlance—are trained jointly using the loss function:

$$L^{\text{total}} = L^{\text{recon}} + L^{\text{pred}} + \lambda_{\text{code}} L^{\text{code}} + \lambda_{\text{grad}} L^{\text{grad}} + \lambda_{\text{reg}} L^{\text{reg}} + \lambda_{\text{GAN}} L^{\text{GAN}}, \tag{13}$$

and the GAN discriminator is trained using L^{disc}. We use λ_{code} = 100, λ_{grad} = 1 and λ_{reg} = 10^{-3} for all the cases considered in this paper. λ_{GAN} varies for each test case and is obtained from experimentation; it is typically in the range 0–0.1 for best results. The discriminator is trained by minimizing L^{disc}, along with an additional gradient penalty loss term similar to WGAN-GP [17]. We alternate between training the generator one step followed by the discriminator for one step, at each training iteration. At test time, the discriminator is not used.